Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY

18CAU301 DATA STRUCTURES 4H – 4C

Instruction Hours / week: L: 4 T: 0 P: 0 Marks: Int : 40 Ext : 60 Total: 100

Scope

Data structures and algorithms are the building blocks in computer programming.

This course will give students a
comprehensive introduction of common data structures, and algorithm design and analysis. This course also intends
to teach data structures and algorithms for solving real problems that arise frequently in computer applications, and
to teach principles and techniques of computational complexity.

Objectives

- understand common data structures and algorithms, and be able to implement them
- Judge efficiency trade-offs among alternative data structure implementations or combinations.
- analyze the complexities of data structures and algorithms;
- choose appropriate data structures and algorithms for problem solving.

Unit-I

Arrays-Single and Multi-dimensional Arrays, Sparse Matrices (Array and Linked Representation).Stacks
Implementing single / multiple stack/s in an Array; Prefix, Infix and Postfix expressions, Utility and conversion of
these expressions from one to another; Applications of stack; Limitations of Array representation of stack

Unit-II

Linked Lists Singly, Doubly and Circular Lists (Array and Linked representation); Normal and Circular,
representation of Stack in Lists; Self Organizing Lists; Skip Lists Queues, Array and Linked representation of
Queue, De-queue, Priority Queues

Unit-III

Trees - Introduction to Tree as a data structure; Binary Trees (Insertion, Deletion , Recursive and Iterative

Traversals on Binary Search Trees); Threaded Binary Trees (Insertion, Deletion, Traversals); Height-Balanced
Trees (Various operations on AVL Trees).

Unit-IV

Searching and Sorting, Linear Search, Binary Search, Comparison of Linear and Binary Search, Selection Sort,
Insertion Sort, Insertion Sort, Shell Sort, Comparison of Sorting Techniques

Unit-V

Hashing - Introduction to Hashing, Deleting from Hash Table, Efficiency of Rehash Methods, Hash Table
Reordering, Resolving collusion by Open Addressing, Coalesced Hashing, Separate Chaining, Dynamic and Extendible Hashing, Choosing a Hash Function, Perfect Hashing, Function

Text Book:

1. Aaron M. Tenenbaum, Moshe J. Augenstein, YedidyahLangsam, (2009). Data Structures Using C and C++, (2nd ed.), PHI.

Suggested readings

1. Aaron M. Tenenbaum, Moshe J. Augenstein, Yedidyah Langsam,(2003). Data Structures Using Java.
2. Aaron M. Tenenbaum, Moshe J. Augenstein, YedidyahLangsam, (2009). Data Structures Using C and
C++, (2nd ed.), PHI.
3. Adam Drozdek, (2012). Data Structures and algorithm in C++, (3rd ed.), Cengage Learning.
4. Goodrich, M. and Tamassia, R.,(2013). Data Structures and Algorithms Analysis in Java, (4th ed.), Wiley.
Herbert Schildt, (2014). " Java The Complete Reference (English) 9th Edition Paperback", Tata McGraw
Hill.
5. John Hubbard,(2009). Data Structures with JAVA, (2nd ed.), India, McGraw Hill Education.
6. Malik, D. S., Nair, P.S., (2003). Data Structures Using Java, Course Technology.
7. Malik, D.S., (2010). Data Structure using C++,(2nd ed.), Cengage Learning.
8. Mark Allen Weiss, (2011). Data Structures and Algorithms Analysis in Java, Pearson Education, (3rd ed.).
9. Robert L. Kruse, (1999). Data Structures and Program Design in C++, Pearson.
10. Robert Lafore, (2003). Data Structures and Algorithms in Java, (2nd ed.), Pearson Macmillan Computer
Publications.
11. Sartaj Sahni, (2011). Data Structures, Algorithms and applications in C+, (2 nd ed.) Universities Press.

Websites

1. http://en.wikipedia.org/wiki/Data_structure
2. http://www.cs.sunysb.edu/~skiena/214/lectures/
3. www.amazon.com/Teach-Yourself-Structures-Algorithms

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY

SUBJECT NAME: DATA STRUCTURES SUBJECT CODE: 18CAU301

LECTURE PLAN

UNIT 1

S. No	Lecture Duration (Hr)	Topics to be Covered	Support Materials
1	**1**	Introduction to Arrays	W1, T1:24-26
2	**1**	Single and Multi-dimensional Arrays	W1, T1:33-37
3	**1**	Sparse Matrices	W1
4	**1**	Stacks Implementing single / multiple stack/s in an Array	T1:85-94
5	**1**	prefix, infix and postfix Expressions	W1, T1; 95,96
6	**1**	Utility and conversion of these expressions from one to another	T1:98-100
7	**1**	Applications of stack	T1:778,79
8	**1**	Limitations of Array representation of stack	W1, T1:86,87
9	**1**	**Recapitulation of Important Questions**	
Total no. of Hours planned for Unit – 1: 9 Hours			

Text Book:

T1 **:** Aaron M. Tenenbaum, Moshe J. Augenstein, YedidyahLangsam, (2009). Data Structures Using C and C++, (2nd ed.), PHI

Reference Book:

R1: Adam Drozdek, (2012). Data Structures and algorithm in C++, (3rd ed.), Cengage Learning.

Website :

W1 : http://en.wikipedia.org/wiki/Data_structure

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY

SUBJECT NAME: DATA STRUCTURES SUBJECT CODE: 18CAU301

LECTURE PLAN

UNIT 2

S. No	Lecture Duration (Hr)	Topics to be Covered	Support Materials
1	**1**	**Introduction to** Linked Lists	W1, T1:186
2	**1**	Singly, Doubly and Circular Lists	T1:188-190
3	**1**	Normal and Circular Representation	T1:190
4	**1**	Representation of Stack in Lists	W1
5	**1**	Self Organizing Lists	T1:203-206
6	**1**	Skip Lists Queues	W1
7	**1**	Array and Linked representation of Queue	T1:174-176
8	**1**	De-queue, Priority Queues	T1:180-183
9	**1**	**Recapitulation of Important Questions**	
Total no. of Hours planned for Unit – 2: 9 Hours			

Text Book:

T1 **:** Aaron M. Tenenbaum, Moshe J. Augenstein, YedidyahLangsam, (2009). Data Structures Using C and C++, (2nd ed.), PHI

Reference Book:

R1: Adam Drozdek, (2012). Data Structures and algorithm in C++, (3rd ed.), Cengage Learning.

Website :

W1 : http://en.wikipedia.org/wiki/Data_structure

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY

SUBJECT NAME: DATA STRUCTURES SUBJECT CODE: 18CAU301

LECTURE PLAN

UNIT 3

S. No	Lecture Duration (Hr)	Topics to be Covered	Support Materials
1	**1**	Introduction to Trees	W1, T1:249
2	**1**	Introduction to Tree as a data structure	T1:249
3	**1**	Binary Trees :Insertion, Deletion	T1:249,250
4	**1**	Recursive on Binary Search Trees	T1:250
5	**1**	Iterative Traversals on Binary Search Trees	T1:251
6	**1**	Threaded Binary Trees	W1
7	**1**	Insertion, Deletion, Traversals on Binary Tree	T1:251-255
8	**1**	Height-Balanced Trees (Various operations on AVL Trees).	T1:275
9	**1**	**Recapitulation of Important Questions**	
Total no. of Hours planned for Unit – 3: 9 Hours			

Text Book:

T1 **:** Aaron M. Tenenbaum, Moshe J. Augenstein, YedidyahLangsam, (2009). Data Structures Using C and C++, (2nd ed.), PHI

Reference Book:

R1: Adam Drozdek, (2012). Data Structures and algorithm in C++, (3rd ed.), Cengage Learning.

Website :

W1 : http://en.wikipedia.org/wiki/Data_structure

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY

SUBJECT NAME: DATA STRUCTURES SUBJECT CODE: 18CAU301

LECTURE PLAN

UNIT 4

S. No	Lecture Duration (Hr)	Topics to be Covered	Support Materials
1	**1**	Introduction to Searching	W1, R1:301-305
2	**1**	Introduction to Sorting	W1, R1:305-307
3	**1**	Linear Search Algorithm	W1, R1:308-312
4	**1**	Binary Search Algorithm	W1, R1:313-315
5	**1**	Comparison of Linear and Binary Search	W1
6	**1**	Selection Sort, Insertion Sort	W1, R1:316-325
7	**1**	Shell Sort	W1
8	**1**	Comparison of Sorting Techniques	W1
9	**1**	**Recapitulation of Important Questions**	
Total no. of Hours planned for Unit – 4: 9 Hours			

Text Book:

T1 **:** Aaron M. Tenenbaum, Moshe J. Augenstein, YedidyahLangsam, (2009). Data Structures Using C and C++, (2nd ed.), PHI

Reference Book:

R1: Adam Drozdek, (2012). Data Structures and algorithm in C++, (3rd ed.), Cengage Learning.

Website :

W1 : http://en.wikipedia.org/wiki/Data_structure

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY

SUBJECT NAME: DATA STRUCTURES SUBJECT CODE: 18CAU301

LECTURE PLAN

UNIT 5

S. No	Lecture Duration (Hr)	Topics to be Covered	Support Materials
1	**1**	Introduction to Hashing	W1, T1:468
2	**1**	Deleting from Hash Table	T1:473
3	**1**	Efficiency of Rehash Methods	T1:474
4	**1**	Hash Table Reordering	T1:476
5	**1**	Resolving collusion by Open Addressing	W1, T1:470-472
6	**1**	Coalesced Hashing, Separate Chaining, Dynamic and Extendible Hashing	W1, T1:485-487
7	**1**	Choosing a Hash Function	W1, T1:505-507
8	**1**	Perfect Hashing, Function	W1, T1:508-512
9	**1**	**Recapitulation of Important Questions**	
10	**1**	**Discussion on Previous End Semester Question Paper**	
11	**1**	**Discussion on Previous End Semester Question Paper**	
12	**1**	**Discussion on Previous End Semester Question Paper**	
Total no. of Hours planned for Unit – 5: 12 Hours			

Text Book:

T1 **:** Aaron M. Tenenbaum, Moshe J. Augenstein, YedidyahLangsam, (2009). Data Structures Using C and C++, (2nd ed.), PHI

Reference Book:

R1: Adam Drozdek, (2012). Data Structures and algorithm in C++, (3rd ed.), Cengage Learning.

Website :

W1 : http://en.wikipedia.org/wiki/Data_structure

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY
SUBJECT NAME: DATA STRUCTURES
SUBJECT CODE: 18CAU301
UNIT – I

Introduction

Algorithm: An algorithm is a finite set of instructions which if followed accomplish a particular task.
Criteria to be satisfied by an algorithm

- Input: zero or more
- Output: one or more result
- Definiteness: every instruction must be clear and unambiguous
- Effectiveness: every instruction must be sufficiently basic

Data type: Refers to the kinds of data that variables may "hold" in the programming language.
Data objects: Refers to a set of elements. For example data object integers, D, refers to set { 0, ±1, ±2, …..}
Data structure: A data structure is a set of domains , a designated domain d , a set of functions and a set of axioms
Developing programs

- The process of developing program is broken into five phases:
- Requirement: Identify the input(s) and output(s)
- Design: Assume that the basic operations to be performed on data objects already exist in form of procedures.
- Top-down approach: Partitions the solutions into subtasks. Each subtask is similarly decomposed until the tasks are expressed within a programming language
- Bottom-up approach: Different parts of the problem are directly solved in a programming language and then these pieces are combined into a complete program.
- Analysis: If more than one algorithm exists for a single problem then compare them and choose the more desirable one. If not able to distinguish between the two then choose one arbitrarily to work.
- Refinement and coding: Modern pedagogy suggests:
- All processing which is independent of the data representation be written out first
- Try to isolate operations that depend upon the choice of data representation
- Verification: Consists of three different aspects:
- Program proving: Before executing a program it is necessary to prove that it is correct. If a correct proof can be obtained then one is assured that for all possible combination of
- inputs, the program and its specification agree.
- Testing is the art of creating sample data upon which to run the program. Tools are available to aid the testing process. Such tools instruments the given source code and tells:
- The number of times a statement was executed
- The number of times a branch was taken
- The smallest and largest values of all variables
- If the program fails to respond correctly then debugging is needed to determine what went wrong and how

to correct it.

Analyzing program

- Efficiency of an algorithm can be measured in terms of:
- Execution time (time complexity)
- The amount of memory required (space complexity)
- The limitations of the technology available at time of analysis determine which one of the above two is more important. Time complexity comparisons are more interesting than space complexity comparisons among the algorithms.
- Two approaches for analyzing the efficiency of an algorithm are: a priori estimates and posteriori estimates

Priori estimation

- Determines the time complexity of:
- The amount of time a single execution of a statement will take Number

of times a statement is executed

- The product of the above two give the amount of time taken by the statement

Time complexity:

- A measure of the amount of time required to execute an algorithm.
- If the execution time of an algorithm is determined by choosing a real time machine and an existing compiler then this exact time would not apply to many machines or to any machine because instruction set, processing speed and compiler may differ from machine to machine.
- Therefore the factors that should not affect time complexity analysis are:

✓ The machine language instruction set chosen to implement the algorithm

✓ The quality of the compiler

✓ The speed of the computer on which the algorithm is to be executed

✓ Moreover it is difficult to get reliable timing figure because of Multiprogramming or time sharing environment

Space complexity

- The better the time complexity of an algorithm is, the faster the algorithm will carry out his work in practice. Apart from time complexity, its space complexity is also important: This is essentially the number of memory cells which an algorithm needs. A good algorithm keeps this number as small as possible, too.
- There is often a time-space-tradeoff involved in a problem, that is, it cannot be solved with few computing time and low memory consumption. One then has to make a compromise and to exchange computing time for memory consumption or vice versa, depending on which algorithm one chooses and how one parameterizes it.
- Time complexity analysis can be based on:
  - Number of arithmetic operations performed
  - Number of comparisons made
  - Number of times through a critical loop
  - Indirect addressing of memory.
  - Assignment of numbers to variables

Sparse Matrix

- A useful application of linear list is the representation of matrices that contain a preponderance of zero elements. These matrices are called sparse matrices. Consider the matrix
- The elements A(0,1), A(0,2) and A(0,3) contain the number of rows, number of columns and ber of nonzero entries.

- Transpose of sparse matrix:

✓ Here the elements in i.j position gets into j,i position. ie. the rows and columns are interchanged.

✓ The elements on the diagonal remain unchanged.

✓ Find all the elements in column 1 and store them into row 1, and all elements in column in 2 and store them in row 2 etc.

Stack and its operations

- A stack is an ordered list in which all insertions and deletions are made at one end, called the top.

Data Structures(16CAU301) Unit I BCA(2016-2019 Batch)

R.NITHYA Department of CA,CS,IT KAHE 3

- The last element to be inserted into the stack will be the first to be removed.
- For this reason stacks are sometimes referred to as

Last in First out (LIFO) lists.

```
procedure PUSH(stack, top, item ,n)
begin
if top = n then call stack_overfolw() top top+1
stack[top] item
end PUSH
```

4

After

pushing 10 4

After

pushing 20 4

3 3 3

2 2 2

Top

1 Top 1 10 1

Top 0

After

pushing 30 4

3

Top 30

20 2 20

10 1 10

```
procedure POP(stack, top, item)
begin
if top = 0 then call stack_underfolw() item
stack[top]
top top-1
end POP
```

Before

poping 4

After

poping 30 4

After

poping 20 4

After
poping 10 4
Data Structures(16CAU301) Unit I BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE 4
3 3 3 3
Top 30
2
20
2
20
2 2
Top
1 10 1 10 Top 1 10 1
Applications of Stack

- Procedure or function calls
- Storing the return address -When a subroutine or a function calls itself recursively, the location(address) of the instruction at which it can later resume needs.
- Local data storage -A subroutine frequently needs memory space for storing the values of local variables, the variables that are known only within the active subroutine.
- Parameter passing -Subroutines a function call often require that values for parameters be supplied to them by the code which calls them .

Infix to postfix conversion and postfix expression evaluation

- The order of precedence is of arithmetic operators is
- Exponentiation
- Multiplication/division *, /
- Addition/subtraction +, -

Postfix:

- Parenthesis the infix expression based on hierarchy of operators
- While writing the postfix form ignore the open parenthesis
- Replace the close parenthesis with the corresponding operator
- ((a + ((b * c) / d)) - (e ^ f))

Prefix:

- Parenthesis the infix expression based on hierarchy of operators
- While writing the prefix form ignore the close parenthesis
- Replace the open parenthesis with the corresponding operator.

Enable | Enlighten | Enrich
KARPAGAM
ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)

Karpagam Academy of Higher Edu
Department of Computer Applicati
Subject : Data Struc
Class: II BCA **Sub**
Objective Type Ques

UNIT I

S.No	QUESTION	OPT 1
1	The logical or mathematical model of a particular data organization is called as_____	Data Structure
2	An algorithms _____is measured in terms of computing time ad space consumed by it.	performance
3	Which of the following is not structured data type?	Arrays
4	Which of the following is a valid non - linear data structure.	Stacks
5	Combining elements of two _____data structure into one is called Merging	Similar
6	Data structures are classified as _____ data type.	User Defined
7	_____are the commonl used ordered list.	Graphs
8	Data structure can be classified as _____data type based on relationship with complex data element.	Linear & Non Linear
9	A data structure whose elements forms a sequence of ordered list is called as _____ data structure.	Non Linear
10	A data structure which represents hierarchical relationship between the elements are called as _____data structure.	Linear
11	A data structure, which is not composed of other data structure, is called as _____data structure.	Linear
12	Data structures, which are constructed from one or more primitive data structure, are called as _____data structure.	Non Primitive

13	_____is the term that refers to the kinds of data that variables may hold in a programming language.	data type
14	The _____model of a particular data organization is called as Data Structure.	software Engineering
15	The design approach where the main tast is decomposed into subtasks and each subtask is further decomposed into simpler solutions is called _____	top down approach
16	_____is a sequence of instructions to accomplish a particular task	Data Strucuture
17	_____criteria of an algorithm ensures that the algorithm terminate after a particular number of steps.	effectiveness
18	An algorithm must produce _____ output(s)	many
19	_____criteria of an algorithm ensures that the algorithm must be feasible.	effectiveness
20	_____criteria of an algorithm ensures that each step of the algorithm must be clear and unambiguous.	effectiveness
21	The time factor whrn determining the efficiency of the algorithm is measured by	counting micro seconds
22	Which of the following data strucutre is linear data structure	Trees
23	The operation of processing each element in a list is called	sorting
24	Finding the location of the element with a given value is	Traversal
25	Which of the following data structure are indexed structures?	Linear array
26	Size of the int data type is	2 byte
27	Each array declaration need not give, implicitely or explicitely the information about	name of array
28	Which of the following data strucutre cannot store the non-homogeneous data elements?	Arrays
29	Which of the following data strucutre can only store the homogeneous data elements?	Union

30	Which of the following data strucutre is linear type?	Strings
31	An algorithm that directly calls itself is called	Sub algorithm
32	What term is used to describe O(N) algorithm?	constant
33	Which of these is the correct Big O expression for 1 + 2 + 3 + …. + n ?	O(log n)
34	Which of these is the correct Big O expression for 35n + 6?	O(log n)
35	Find out the complexity of x = 3*y + 2; z=z+1;	O(log n)
36	Representation of data structure in memory is known as:	recursive
37	If the address of A[1][1] and A[2][1] are 1000 and 1010 respectively and each element occupies 2 bytes then the array has been stored in _____order.	row major
38	When the maximum entries of (m*n) matrix are zeros then it is called as _____.	Transpose matrix
39	A matrix of the form (row, col, n) is otherwise known as _____.	Transpose matrix
40	A list of finite number of homogeneous data elements are called as _____	Stacks
41	No of elements in an array is called the _____of an array.	Structure
42	The size or length of an array = _____.	UB – LB + 1
43	Searching is the Process of finding the _____ of the element with the given value or a record with the given key.	Place
44	Length of an array is defined as _____of elements in it.	Structure
45	_____is a set of pairs, index and value.	stack
46		
47	Sum of terms of the form ax^e is called _____.	Array
48	_____is a collection of data and links.	Links
49	Each item in a node is called a_____.	Field
50	The elements in the list are stored in a one dimensional array called a _____	Value

51	Data movement and displacing the pointers of the Queue are tedious proplems in _____ representation of a Queue.	Array
52	Solving different parts of a program directly and combining these pieces into a complete program is called _____	top down approach
53	The number of times a statement in a program is executed is called its _____	Piori estimate
54	_____means the computing time of the algorithm is constant	O(log n)
55	_____means the computing time of the algorithm is linear	O(log n)
56	Algorithms with time complexity $O(n^2)$ are called _____	linear
57	For large data set algorithms with complexity greater than _____are impractical	O(log n)
58	Which of the following case does not exist in complexity theory?	Best case
59	Two main measures for the efficiency of the algorithm are	Processor and memory

cation
ons
tures
ject code: 18CAU301
stions

OPT 2	OPT 3	OPT 4	ANSWER
Software Engineering	Data Mining	Data Ware Housing	Data Structure
effectiveness	finiteness	definiteness	performance
Union.	Queue	Linked list.	Union.
Trees	Queues	Linked list.	Trees
Dissimilar	Even	Un Even	Similar
Abstract	Primitive & Non Primitive	predefined only	Primitive & Non Primitive
Trees	Stack and Queues	List	Stack and Queues
Linear	Non Linear	None of the above	Linear & Non Linear
Linear.	Primitive	Non Primitive	Linear.
Primitive.	Non Linear	Non Primitive	Non Linear
Non Primitive	Non Linear	Primitive	Primitive
Primitive.	Non Linear	Linear	Non Primitive

data structure	data Object	data	data type
logical or mathematical	Data Mining	Data Ware Housing	logical or mathematical
bottom up approach	hierarchical approach	merging approach	top down approach
Algorithm	Ordered List	Queue	Algorithm
finiteness	definiteness	particular	finiteness
only one	atleast one	zero or more	atleast one
finiteness	definiteness	infinite	effectiveness
finiteness	definiteness	infinite	definiteness
counting the number of key operatiuons	counting the number of statements	counting the kilobyte of the algorithm	counting the number of key operatiuons
Graphs	Arrays	Union	Arrays
merging	Inserting	Traversal	Traversal
Search	Sort	Merging	Search
Linked list	Stack and Queues	queue	Linear array
4 byte	compiler dependent	varies all the time	compiler dependent
data type of array	first data from the set to be stored	index set of array	first data from the set to be stored
Records	Pointers	Union	Arrays
Arrays	Pointers	Strucutres	Arrays

Trees	Graph	B Tree	Strings
Recursion	polish notation	traversal algorithm	Recursion
linear	logarithmic	quadratic	linear
O(n log n)	O(n)	O(n2)	O(n2)
O(n log n)	O(n)	O(n2)	O(n)
O(n log n)	O(n)	O(1)	O(1)
abstract data type	storage structure	file structure	abstract data type
column major	matrix major	tuple major	row major
Sparse Matrix	Inverse Matrix	tridiagonal matrix	Sparse Matrix
Inverse Matrix	Sparse Matrix	Diagonal matrix	Sparse Matrix
Records	Arrays	Linked list.	Arrays
Height	Width	Length.	Length.
LB + 1	UB - LB	UB – 1	UB – LB + 1
Location	Value	Operand	Location
Height	Size	Number	Number
queue	Arrays	Set	Arrays
Matrix	Expression	Polynomial	Polynomial
Node	List	Item	Node
Data item	Pointer	Data	Field
List	Data	Link	Data

Linked	Circular	lenear list	Array
bottom up approach	hierarchical approach	merging approach	bottom up approach
Posteriori estimate	Frequency count	Program count	Frequency count
O(n)	O(n log n)	O(1)	O(1)
O(n)	O(n log n)	O(1)	O(n)
exponential	quadratic	cubic	quadratic
O(n)	O(n log n)	O(1)	O(n log n)
Worst case	Average case	Null case	Null case
Complexity and capacity	Time and space	Data and space	Time and space

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY
SUBJECT NAME: DATA STRUCTURES
SUBJECT CODE: 16CAU301
UNIT 2

2.Linked List Introduction
2.1 1nsertion and Deletion in singlely linked ordered list
Single linked Ordered list consists of sequence nodes. Each node consists of a data item and a pointer to a nextnode. A
pointer called start point to the first node in the ordered linked list. In a empty list start is Null
If (start = Null) then // empty list
while ((currnode <> Null) and (item > INFO(currnode)) do // insertion in middle or end of list prevnode currnode
currnode LINK(currnode)
end
X
end
end
end SLINKLISTADD
Empty list GETNODE(X) INFO(X) item & LINK(X) Null
start=Null X
Anandi null
X
start X
Anandi
null
start
Insertion at the beginning of the list. Insert Anand in the list
start Anandi
null
GETNODE(X) INFO(X) item
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
X X Anand
Establishing link for newly created node X
LINK(X) start
start Anandi null
X Anand
Making start to point the new node x which now the first node in the list
start X
Anandi null

X
start
Anand
Insertion at the middle or end of the list. Insert Gopi in the list
After traversing the list the location for inserting Gopi is before the node current in other words after the previous node
prevnode
currnode
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
start Anand Babu Hari
Node X is created and info Gopi is stored
X Gopi
Establishing link for the newly created node X
prevnode
currnode
star
t
Anand Babu Hari
null
LINK(X) currnode
X Gopi
Prevnode LINK made to point newly created node X
prevnode
currnode
start Anand Babu Hari
null
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
Gopi
X
Procedure SLINKLISTDELETE(start, item) begin
If (item = INFO(start)) then // Deleting first node
X start
start LINK(start)
RET(X)
return
else
start Anand Babu Hari
null
X
start Babu Hari
null
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE

Deleting a node in the middle or at the end of the list
prevnode
start Anand Babu Gopi
ll
X
prevnode
start Anand Babu Gopi
ll
X
prevnode
start Anand Babu
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
2.2 1nsertion and Deletion in Doubly linked ordered list
Double linked ordered list consists of sequence nodes. Each node consists of a data item and two pointers RLINK to the
previous node and LLINK to the next node. A pointer called start point to the first node in the ordered linked list. In a
empty list start is Null
procedure DLINKLISTADD(start, item) begin
GETNODE(X) INFO(X) item
If (start = Null) then // empty list RLINK(X) LLINK(X) Null
start X return
else if (item < INFO(start)) then // at front of the list
RLINK(X) start
LLINK(X) Null
LLINK(start) X start X
return
else
currnode start
while ((RLINK(currnode) <> Null) and (item > INFO(RLINK(currnode))) do // insertion in middle of list
currnode RLINK(currnode)
end
RLINK(X) RLINK(currnode) LLINK(X) currnode; LLINK(RLINK(X)) X; RLINK(currnode) X
end
return
end end
end DLINKLISTADD
Inserting in empty list
As X is the one and only node in the list it's RLINK and LLINK
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
Create new node i.e. X Store Anandi in the new node X are marked Null. i.e. predecessor and successor nodes are absent
Inserting at the beginning of the list

start Babu
null Babu null
start
null
Arun
null
X
Arun
start
null
X null
Anand Anandi
null
Insertion in the middle or at the end of the list
CurrNode
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
start
nul
Arun Babu Gopu Ram
nul
RLINK(X) RLINK(currnode)
X Guru
tart
nul
Arun Babu Gopu Ram
nul
LLINK(X) currnode
Guru
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
X
nul
Arun Babu Gopu Ram
nul
start
LLINK(RLINK(X)) X
X
Guru
nul
Arun Babu Gopu Ram
start
RLINK(currnode) X
Guru
X

procedure DLINKLISTDELETE(start, item) begin
If (item = INFO(start)) then // Deleting first node
X start
start RLINK(start) LLINK(start) Null
RET(X)
return
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
else
currnode RLINK(start)
while ((currnode <> Null) and (item <> INFO(currnode)) do // insertion in middle or end of list currnode
RLINK(currnode)
end
if currnode = Null then
print("Item not found") else
RLINK(LLINK(currnode)) RLINK(currnode) If RLLINK(currnode) <> Null LLINK(RLINK(currnode))
LLINK(currnode)
end
RET(currnode)
end
return
end
end DLINKLISTADD
t nul
Arun Babu Gopu Ram
x
nul
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
start nul
Arun Babu Gopu Ram
x
nul
Deleting from the middle of the list
currnode
start nul
Arun Babu Gopu Ram
nul
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
RLINK(LLINK(currnode)) RLINK(currnode)
start nul
Arun Babu Gopu Ram
nul
nul

Arun Babu Gopu Ram
LLINK(RLINK(currnode)) LLINK(
nul
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
start nul Anand
RET(currnode)
nul
Babu Ramu
nul
2.3 1nsertion and Deletion in Linked stack
Linked stack consists of sequence nodes. Each node consists of a data item and a pointer to a next node. A pointer called
top point to the top node in the stack. In an empty stack top Null. The following illustrates PUSH and POP functions of
stack.
procedure PUSH(top, item) begin
GETNODE(X) INFO(X) item LINK(X) top
top X
end PUSH
empty stack: if top = Null
inserting Anand in stack
Statement illustration
GETNODE(X) top Null X
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
INFO(X) item top Null X
Anand
LINK(X) top top Null X
Anand Null
Top X
Anand Null
top
procedure POP(top, item)
begin
X top
item INFO(top) top LINK(top) RET(X)
end POP
Statement illustration
X top top
Anand Babu Null
item INFO(top)
top
Anand Babu Null
item Anand

top LINK(top)
Anand Babu Null
Anand
top
RET(X) top
Babu Null
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
2.4 1nsertion and Deletion in Linked Queue
Linked queue consists of sequence nodes. Each node consists of a data item and a pointer to a next node. A pointer called
rear and front point to the first and last node in the queue respectively. In a empty queue rear Null and front Null. The
following illustrates PUSH and POP functions of stack.
procedure ADDQ(rear, front, item)
begin
GETNODE(X) INFO(X) item
If (front = NULL) then
front X; LINK(X) Null
else
LINK(X) LINK(rear)
end
rear X
end ADDQ
Statement illustration
GETNODE(X) front rear Null X
INFO(X) item front rear Null X
Anand
If front = NULL
Anand Null
front X rear NullX
LINK(X) Null front
rear X x
Anand Null
rear
front
procedure DELETEQ(rear, front, item)
begin
If (front = NULL) then
call QUEUE_EMPTY()
else
X rear
item INFO(rear)
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE

If (front = rear) then
front rear NULL
end
front LINK(front)
end RE(X)
end DELETEQ
Statement illustration
X
Anand Babu Null
X front front rear
item INFO(front)
front
Anand Babu Null
rear
item Anand
front LINK(front)
Anand Babu Null
rear
front item Anand
RET(X) front
Babu Null
rear
2.5 Polynomial Addition
In general, we want to represent the polynomial
A(x) = amxem + ... + a1xe1
where the ai are non-zero coefficients with exponents ei such that em > em-1 > ... > e2 > e1 >= 0. Each term will be
represented by a node. A node will be of fixed size having 3 fields which represent the coefficient and exponent of a term
plus a pointer to the next term
For instance, the polynomial A= 13x3 + 12x2 + 1 would be stored as
A 13 3 12 2 1 0
Null
while B = 8x4 - 3x3 + 10x2 would look like
B 8 4 -13 3 12 2 Null
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
In order to add two polynomials together we examine their terms starting at the nodes pointed to by A and B. Two
pointers p and q are used to move along the terms of A and B. If the exponents of two terms are equal, then the
coefficients are added and a new term created for the result. If the exponent of the current term in A is less than the
exponent of the current term of B, then a duplicate of the term of B is created and attached to C. The pointer q is

advanced to the next term. Similar action is taken on A if EXP (p) > EXP(q).
Each time a new node is generated its COEF and EXP fields are set and it is appended to the end of the list C. In order to
avoid having to search for the last node in C each time a new node is added, we keep a pointer d which points to the
current last node in C. The complete addition algorithm is specified by the procedure PADD. PADD makes use of a
subroutine ATTACH which creates a new node and appends it to the end of C. To make things work out neatly, C is initially
given a single node with no values which is deleted at the end of the algorithm. Though this is somewhat inelegant, it
avoids more computation.

```
procedure ATTACH(C,E,d)
//create a new term with COEF = C and EXP = E and attach it to the node pointed at by d//
call GETNODE(I) EXP(I) E COEF(I) C LINK(d) I end ATTACH
procedure PADD(A,B,C)
//polynomials A and B represented as singly linked lists are summed to form the new list named C// p A; q B
call GETNODE(C); d C
while ((p<> 0) and (q<> 0)) do case
: EXP(p) = EXP(q): x COEF(p) + COEF(q)
if (x<> 0) then call ATTACH(x, EXP(p),d)
p LINK(p); q LINK(q)
: EXP(p) < EXP(q): call ATTACH(COEF(q),EXP(q),d)
q LINK(q)
: else: call ATTACH(COEF(p),EXP(p),d)
p LINK(p)
end
end
while (p<> 0) do
call ATTACH(COEF(p),EXP(p),d)
p LINK(p) end
wile (q <>0) do
call ATTACH(COEF(q),EXP(q),d) q LINK(q)
end
LINK(d) 0; t C; C LINK(C)
call RET(t ) end PAD
```

2.6 Sparse Matrix

A basic node structure called MATRIX_ELEMENT as depicted in the below figure is required to represent sparse matrices.
The V, R and C fields of one of those nodes contain the value, row, and column indices, respectively, of one matrix
element. The fields LEFT and UP are pointers to the next element in a circular list containing matrix elements for a row or

Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE

column. LEFT points to the node with the next smallest column subscript, and UP points to the node with the next smallest
row subscript.
LEFT UP
V R C
A circular list represent each row and column. A column's list can share nodes with one or more of the row's list. Each row
and column list has a head node such that more efficient insertion and deletion algorithms can be implemented. The head
node of each row list contains 0 in the C field. The head node of each column list has 0 in the R field. The row head nodes
are pointed to by respective elements in the array of pointers AROW. Elements of ACOL point to the column head nodes.
A row or column without nonzero elements is represented by a head node whose LEFT and UP field points to itself.
In scanning a circular list we encounter matrix elements in order of decreasing row or column subscripts. This is used to
simplify the insertion of new nodes to the structure. We assume that new nodes being added to a matrix are usually
ordered by ascending-row subscript and ascending-column subscript. A new node is inserted following the head node all
the time and no searching of the list is necessary.
Algorithm for constructing a multilinked structure representing a matrix is given below. It is assumed that input records for
the algorithm consist of row, column, and nonzero matrix-element values in arbitrary order. Algorithm
CONSTRUCT_MATIX. It is required to form a multilinked representation of a matrix using the MATRIX-ELEMENT node
structure. The matrix dimensions M and N, representing the number of rows and columns are known before execution of
algorithm. Arrays AROW and ACOL contain pointers to the head nodes of the circular lists. X and Y are used as auxiliary
pointers. A row index, column index, and value of a matrix element are read into variables ROW, COLUMN, and VALUE.
1. Initialize matrix structures
for l 1,2,.....M
ACROW[l] MATRIX_ELEMENT C(AROW[l]) 0
LEFT(AROW[l] AROW[l] end
for l 1,2.....N
ACOL[l] MATRIX_ELEMENT R(ACOL[l] 0
UP(ACOL[l]VACOL[l] end
loop
read(ROW, COLUMN, VALUE)
P MATRIX_ELEMENT
R(P) ROW
C(P) COLUMN V(P) VALUE

Q AROW[R(P)] while C(P)<C(LEFT(Q))
Q LEFT(Q)
LEFT(P) LEFT(Q) LEFT(Q) P
End
Q ACOL[C(P)]
while R(P)<R(UP(Q))
Q UP(Q) UP(P) UP(Q)
UP(Q) P
end forever
representation of 3X3 Sparse matrix 0 22 0 by linked list
Data structures(16cau301)unitII BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE
0 0 0
33 0
1 2 3
ACOL
AROW LEFT LEFT LEFT
1
0 0
0 0 0 0
UP
22 1 2
0 0

Enable | Enlighten | Enrich
KARPAGAM
ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)

Karpagam Academy of Higher Edu
Department of Computer Applicati
Subject : Data Struc
Class: II BCA **Sub**
Objective Type Ques
UNIT II

1	The data field of the _____node usually donot conatain any information.	first
2	A_____is a linked list in which last node of the list points to the first node in the list.	Linked list
3	A_____in which each node has two pointers, a forward link and a Backward link.	Doubly linked circular list
4	In sparse matrices each nonzero term was represented by a node with _____fields.	Five
5	T(i)_____	Top of the i^{th} stack
6	F(i)_____	Front of the $(i + 1)^{th}$ Queue
7	R(i)_____	Rear of the $(i + 1)^{th}$ Queue
8	_____list allows traversing in only one direction.	Singly linked list
9	_____ allows traversing in both direction.	Singly linked list
10	The best application of Doubly Linked list in computers is _____	Job scheduling in Time sharing environment
11	The computing time for manipulating the list is _____for sequential Representation	Less then
12	In singly linked list ,each node has _____field.	One
13	In linked list ,each node has fields namely_____	Link, Value
14	In Doubly linked list ,each node has at least _____field.	One

15	In Doubly linked list ,each node has fields namely_____	Link, Data1, Data2
16	The doubly linked list is said to be empty if it conatins _____	no nodes at all.
17	In Linked representation of Sparse Matrix, DOWN field used to link to the next nonzero element in the same _____	Row
18	In Linked representation of Sparse Matrix, RIGHT field used to link to the next nonzero element in the same _____	Row
19	The time complexity of the MREAD algorithm that reads a sparse matrix of n rows, n columns and r nonzero terms is _____	O(max {n, m, r})
20	A _____is a set of characters is called a string.	Array
21	Adding a new element into a data structure called	Merging
22	The Process of finding the location of the element with the given value or a record with the given key is _____.	Merging
23	Arranging the elements of a data structure in some type of order is called _____.	Merging
24	What is the index number of the last element of an array with 29 elements?	29
25	The memory address of the first element of an array is called	Floor address
26	Two dimensional array are also called as	Table arrays
27	Arrays are best data structure	for relatively permanent collection of data
28	In a singly linkedlist how many fields are there?	1
29	Name the fields in circular linked list	Data and link
30	To implement Sparse matrix dynamically, the following data structure is used	Stacks
31	How many fields are there in doubly linked list?	1
32	In the last node of the circular linked list the link field contains	null

33	Name the fields in doubly linked list	Data and link
34	How many fields are there in circular doubly linked list?	1
35	In the last node of the circular doubly linked list the link field contains	null
36	What member function places a new node at the end of the linked list?	addNode()

cation
ons
tures
ject code: 18CAU301
stions

head	tail	last	head
Singly linked circular list	Circular list	Insertion node	Singly linked circular list
Circular list	Singly linked circular list	Linked list	Doubly linked circular list
Six	Three	Four	Three
Top of the (i + 1)th stack	Top of the (i th stack	Top of the (i - 2)th stack	Top of the ith stack
Front of the ith Queue	Front of the (i th Queue	Front of the (i -2)th Queue	Front of the ith Queue
Rear of the ith Queue	1) th Queue	Rear of the (i - 2)th Queue	Rear of the ith Queue
Doubly linked list	Circular Doubly Linked List	Ordered List	Singly linked list
Doubly linked list	Circular Singly Linked List	Circular Queue	Doubly linked list
Processing Procedure calls	Dynamic Storage Management	Evaluating postfix expressions	Dynamic Storage Management
Greater than	Less then equal	Greater than equal	Less then
Two	Three	Five	Two
Link, Link	Data, Link	Data, Data	Data, Link
Two	Three	Five	Three

Data and Link	Only Llink and Rlink	Llink, Data, Rlink	Llink, Data, Rlink
nodes with data fields empty.	only a head node.	a node with its link fields points to null	only a head node.
List	Column	Diagonal	Column
Matrix	Column	Diagonal	Row
O(m * n * r)	O(m + n + r)	O(max {n, m})	O(m + n + r)
String	Heap	List	String
Insertion	Searching	Sorting	Insertion
Insertion	Searching	Sorting	Searching
Insertion	Searching	Sorting	Sorting
28	0	25	28
Foundation address	First address	Base address	Base address
Matrix arrays	both a and b	Special array	both a and b
for data are constantly changing	both a and b	none of the above	for relatively permanent collection of data
2	3	4	2
2Data and link	Data and 2link	2Data and 2link	2Data and link
linked list	Trees	Graphs	linked list
2	3	4	3
pointer data item	pointer to next node	pointer to first node	pointer to first no

2Data and 2link	Data and 2link	2Data and 2link	Data and 2link
2	3	4	3
pointer data item	pointer to next node	pointer to first node	pointer to first node
appendNode()	lastNode()	newNode()	appendNode()

d

37	The largest element of an array index is called its	lower bound
38		ABCD
39	Consider the usual implementation of parentheses	1
	Assume that the operators +,-, X are left ^ associative and ^ is right associative. The order	^ ^ abc X+ def -
41	The memory address of the first element of an array is called	floor address
42	What is the minimum number of stacks of size n required to implement a queue of size n?	1
43	The situation when in a linked list START=NULL is	underflow
44	A linear collection of data elements where the linear node is given by means of pointer is called	linked list
45	Which of the following operations is performed more efficiently by doubly linked list than by singly linked list?	Deleting a node whose location in given
46	How many nodes in a tree have no ancestors.	0

e

range	upper bound	subscript	upper bound
DCBA	DCAB	ABDC	DCBA
2	3	4	3
Λ Λ abc X+ de f -	Λ	Λ	Λ Λ abc X+ def -
base address	first address	foundation address	base address
3	2	4	2
overflow	housefull	saturated	underflow
node list	primitive list	queue	linked list
Searching of an unsorted list for a given item	Inverting a node after the node with given location	Traversing a list to process each node	Deleting a node whose location in given
1	2	n	1

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY
SUBJECT NAME: DATA STRUCTURES
SUBJECT CODE: 18CAU301

Inorder traversal of above tree is : 3 + 7 * 8 - 6 The following table traces how the INORDER works for the binary tree above

Call of IN ORDER
Value of root
Action Main *
1 +
2 3
3 Null Print
3
3 Null Print
+
4 7
5 Null Print
7
5 Null Print
***
6 -
7 8
8 Null Print
8
8 Null Print -
9 6
10 Null Print
6
10 Null

Preorder Traversal: 1. Visit the node (root or subroot) 2. Traverse the left subtree 3. Traverse the right subtree
Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)

R.NITHYA Department of CA,CS,IT KAHE 6 / 19 Preorder traversal of above tree is : * + 3 - 8 6 The following table traces how the PREORDER works for tree above

Call of
INORDER
Value
of
root
Action Main * Print
*** 1 + Print**
+ 2 3 Print
3 3 Null 3 Null 4 7 Print
7 5 Null 5 Null 6 - Print - 7 8 Print
8 8 Null 8 Null 9 6 Print
6 10 Null

Postorder Traversal: 1. Traverse the left subtree 2. Traverse the right subtree3. Visit the node (root or subroot) Postorder traversal of above tree is : 3 7 + 6 8 - * The following table traces how the PREORDER works for the tree above Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)

Call of
INOR
DER
Value
of
root
Action Main * Print
*** 1 + Print**
+ 2 3 Print
3 3 Null 3 Null 4 7 Print
7 5 Null 5 Null 6 - Print - 7 8 Print
8 8 Null 8 Null 9 6 Print
6 10 Null 10 Null Threaded Binary Tree

- The traversal algorithms described use stack which consumes large storage space
- The majority of pointers **(LLINK and RLINK)** in any binary tree are **Null. Threaded representation avert the need for a stack, making use of pointer fields which would otherwise have the value Null**. The most common convention for threaded representation is: if the left subtree is empty, **LLINK** points to the in-order predecessor if the right subtree is empty, **RLINK** points to the in-order successor. But any program examining the tree must be able to distinguish between a branch and a thread. So we introduce two additional fields into each node

LBIT and **RBIT** obviously need be only one-bit fields. Because of word-length considerations, these fields will often not make any difference to the amount of storage used per node; the convention adopted, for instance, might be to use negated pointer values to indicate threads. The convention used in these notes is: if **LBIT = 0, LLINK** points to the left child; if **LBIT = 1, LLINK** points to the in-order predecessor; if **RBIT = 0, RLINK** points to the right child; if **RBIT = 1, RLINK** points to the in-order successor.

LBIT LCHI
LD
DAT
A
RCH
ILD

RBIT In the representation of a threaded binary tree, it is convenient to use a special node HEAD — the "list" head — which is always present, even for an empty tree is: Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)

0 - 1 For any node X in a binary tree, if **RBIT(X) = 0** then the inorder successor of **X** is **RCHILD(X)** is by definition of threads. If **RBIT(X) = 1** then the inorder successor of X is obtained by following a path of left child links from the right child of X until a node with **LBIT(X) = 0.** The algorithm **INSUC** finds the inorder successor of any node X in a threaded binary tree.

Procedure INSUC(X) S → RCHILD(X)
If RBIT[X] = 1 **then**
While LBIT[X] = 1 **do S → LCHILD(X) return(S) end INSUC The procedure INSUC** finds the inorder successor of an arbitrary node in the threaded binary tree without using an additional stack. Since the tree is the left subtree of the head node and because of the choice of **RBIT(X) = 0 for the head node, the inorder sequence of the nodes for the tree T** is obtained by the procedure **TINORDER**

Procedure TINORDER(T) HEAD → T
loop T → INSUC(T)

If T = HEAD **then** return Print (**DATA(T)**)
forever end INSUC Heap sort A *heap* is defined to be a complete binary tree with the property that the value of each node is at least as large as the value of its children nodes (if they exist) (i.e., $K\,j/2\,Kj$ for 1 j/2 < j n). This implies that the root of the heap has the largest key in the tree. Heap sort may be regarded as a two stage method. First the tree representing the file is converted into a heap. In the second stage the output sequence is generated in decreasing order by successively outputting the root and restructuring the remaining tree into a heap. Essential to any algorithm for Heap Sort is a subalgorithm that takes a binary tree T whose left and right subtrees satisfy the heap property but whose root may not and adjusts T so that the entire binary tree satisfies the heap property. Algorithm ADJUST does this.
procedure *ADJUST* (*i,n*) //Adjust the binary tree with root *i* to satisfy the heap property. The left and right subtrees of *i*, i.e., with roots 2*i and 2i*+ 1, already satisfy the heap property. The nodes of the trees contain records, *R*, with keys *K*. No node has index greater than *n*//
R Ri; *K Ki*; *j* 2*i*
while *j n* **do**
if *j* < *n* **and** *Kj* < *Kj*+1 **then** *j j* + 1 //find max of left and right child// //compare max. child with *K*. If *K* is max. then done//
if *K Kj* **then exit**
R j*/2* Rj; j *2*j *//move* Rj *up the tree//*
end
R j*/2* R
end *ADJUST Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)*
R.NITHYA Department of CA,CS,IT KAHE 9 / 19
Analysis of Algorithm Adjust If the depth of the tree with root i is k, then the while loop is executed at most *k* times. Hence the computing time of the algorithm is *O*(*k*). The heap sort algorithm may now be stated.
procedure *HSORT* (*R,n*) //The file *R* = (*R*1, ...,*Rn*) is sorted into nondecreasing order of the key *K*//
for *i n*/2 **to** 1 **by** -1 **do** //convert *R* into a heap//
call *ADJUST* (*i,n*)
end
for *i n* - 1 **to** 1 **by** -1 **do** //sort *R*//
T Ri+1; *Ri*+1 *R*1; *R*1 *T* //interchange *R*1 and *Ri*+1//
call *ADJUST* (1,*i*) //recreate heap//
80 21 73 55 42 85 33 79 90 84 93 97 Constructing Heap
Data added to the Heap is Heap tree
80
21
Data added to the Heap is Heap tree
73
Data added to the Heap is Heap tree
83 Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE 10 / 19
Data added to the Heap is Heap tree
79
90 Second Phase 90 85 80 79 42 73 21 55 Array representation of above tree Swap A[1] and A[8] and reconstruct the heap with the data from A[1] to A[7] 55 85 80 79 42 73 21 90 Heap reconstruction Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE 11 / 19 85 79 80 55 42 73 21 Array representation of above tree Swap A[1] and A[7] and reconstruct the heap with the data from A[1] to A[6] 21 79 80 55 42 73 85 90 Heap reconstruction 80 79 73 55 42 21 Array representation of above tree After swaping A[1] and

A[6] and reconstruct the heap with the data from A[1] to A[5] 21 79 73 55 42 80 85 90 Heap reconstruction 79 55 73 21 42 Array representation of above tree After swaping A[1] and A[5] and reconstruct the heap with the data from A[1] to A[4] Heap reconstruction Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)

 73 55 42 21 Array representation of above tree After swaping A[1] and A[4] and reconstruct the heap with the data from A[1] to A[3] 21 55 42 73 79 80 85 90 55 21 42 Array representation of above tree After swaping A[1] and A[3] and reconstruct the heap with the data from A[1] to A[2] 42 21 55 73 79 80 85 90 42 21 Array representation of above tree After swaping A[1] and A[2] 21 21 55 73 79 80 85 90 B-Tree A B-tree is a balanced tree scheme in which balance is achieved by permitting the nodes to have multiple keys and more than two children. In Balanced Tree height of paths from root to leaf are same. Given a search-key K, nearly same access time for different K values. B Tree A B-tree of order m ($m \geq 3$) is a balanced tree that satisfies the following properties: – each node contains at most m - 1 elements – each node contains at least -1 elements, – The root may contain a single element Each non leaf node containing j elements has j +1 children Each node has a structure of the form: Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)

 P_0 K_I P_0 K_2 P_3 K_3 P_4 $P_0(K_1, R_1)P_1(P_2, R_2)P_2 \ldots P_{j-1}(K_j, r_j)P_j$ where j is the number of the elements in the node In each node $K_1, \ldots, K_j$ are ordered key values, that is, $K_1 < K_2 < \ldots < K_j$ There are j + 1 references to children nodes $p_0, \ldots, p_j$ and j references to the data file $r_1, \ldots, r_j$

Insertion in B tree The updates always start from the leaf nodes and the tree grows or shrinks from the bottom of the tree The insertion requires first of all a search operation to verify whether the key value is already present in the tree The insertion is always performed on a leaf node – two cases can arise: – If the leaf node is not full, the key value is inserted and the updated leaf node is re-written Before inserting 43. After a search it is found that 43 should be inserted in node 4th node in level 2

P After inserting 43

P – If the leaf node is full, a splitting process starts that can propagate to next level and, in the worst case, can reach the root inserting 33 After a search it is found that 33 should be inserted in node 3rd node in level 2. Let it be P. i.e. P address of the 3rd node. The node P is already full. Hence

- Order as sequence the m entries that would be created with the new key value and
- retain 1 to -1 in the current node.
- Create a new node P′and move +1 to m to the new node.
- Store key and P′ in Q the parent of P.
- if space is not available in Q initiate a propagation of a key from Q to its parent. Ordering of keys in node P with the new key 33: 30 31 **32** 33 34 Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)

 m=5 i.e. m is the number of elements is The node P is already full. Create new node P′**.** Move elements from +1 to m in P to P′**.**

P When element in location and P′ enters node Q the elements in Q is ordered with the new element 33 and P′**.** After insertion of 32 and in Q the resultant B tree is shown below. P P′

P P′ When 45 is inserted a new root node is created as illustrated below. After a search it is found that 45 should be inserted in node 4th node in level 2. Let it be P.

P Ordering of keys in node P with the new key 33: 41 42 43 44 45

Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)

 m=5 i.e. m is the number of elements. is The node P is already full. Create new node P′**.** Move elements from +1 to m in P to P′**. When 43 and P′ to Q the parent P, keys in node Q with the new key 43: 15 25 33 40 43 m=5 i.e. m is the number of elements. is The node P is already full. Create new node P′.** Move elements from +1 to m in P to P′**.**

P P′ Create new root node and add P and P′ as its children

Deletion in B tree Deletion of an element from a leaf node – there are two cases: – If the leaf node is not in

underflow (that is, it has at least elements after the deletion), the key is deleted and the updated leaf node is rewritten – If the leaf node is in underflow, a concatenation process or redistribution process is started Deleting 46. After a search it is found that 46 is in node 4th node in level 3. Let it be P. Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)

 After deletion the leaf node P contains the minimum number of elements 2.

Concatenation The concatenation of two adjacent nodes P and P' is possible if the two nodes contain on the overall less than m - 1 keys

- A node with less -1 than elements, that is, in **underflow**, is combined with
  - an adjacent node with at most -1 keys Deleting 22 Initial situation: – node P in underflow with elements: $p_0 k_1 p_1 k_2 p_2 \ldots k_e p_e$ (e= -2) – node P' adjacent to the right of P with elements : $p_0 k_{e+1} p_{e+1} \ldots$ – node Q, father of P and P', with elements . . . $k_{t-1} p_{t-1} k_t p_t k_{t+1} p_{t+1}$ . . .where p_{t-1} is a pointer to P and p_t is a pointer to P'

Q

P P′ The concatenation of two adjacent nodes results in the following situation: – node P with elements: $p_0 k_1 p_1 k_2 p_2 \ldots k_e p_e k t p'_0 k_{e+1} p_{e+1} \ldots$ – node Q with elements: . . . $k_{t-1} p_{t-1} k_{t+1} p_{t+1}$ . . .where p_{t-1} is a pointer to P

Q

P_{t-1} P_t

P P′

Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)

 Now Q becomes P and its right adjacent is P**'.** Parent of P is Q.

Q

P P′

Redistribution

- If two adjacent nodes cannot be concatenated, then the elements of the two nodes can be Redistributed
- The redistribution operation involves also the father node because one of its elements must be modified; however the number of its elements is not modified and therefore there is no propagation to the next level Initial situation: – node P in underflow with elements: $p_0 k_1 p_1 k_2 p_2 \ldots k_j p_j$ (j = -2) – node P' adjacent to P on the right with elements: $p_0' k_1 p_1' k_2 p_2' \ldots k_e p_e'$ – node Q, father of P and P', with elements : . . . $k_{t-1} p_{t-1} k_t p_t k_{t+1} p_{t+1}$ . . . where p_{t-1} is a pointer to P' and pt a pointer to P To redistribute the elements in the two nodes: – Consider the list of key values $k_1 k_2 \ldots k_j k_t k'_1 k'_2 \ldots k'_e$ – The first key values are left in P' – In the father node the key k_t is replaced by the key value of position +1 Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)

 – The remaining key values are inserted in P Consider the list of key values 21 25 30 31 32 since j=1 (i.e. I element) and e=3 (i.e. 3 element)

Q

P P' Trie indexing An index structure that is particularly useful when key values are of varying size is the trie.

A *trie* is a tree of degree *m* 2 in which the branching at any level is determined not by the entire key value but by only a portion of it. The trie contains two types of nodes.

- The first type we shall call a *branch node* (represented by rectangle) and
- the second an *information node* (represented by oval) In the trie of figure below each branch node contains 27 link fields. All characters in the key values are assumed to be one of the 26 letters of the alphabet. A blank is used to terminate a key value. Thus, LINK(*T*,*i*) points to a subtrie containing all key values beginning with the *i*-th letter (*T* is the root of the trie. When a subtrie contains only one key value, it is replaced by a node of type information. This node contains the key value, together with other relevant information such as the address of the record with this key value, etc. Key is stored here Address of record is stored here Searching a trie for a key value **X** requires breaking up **X** into its constituent characters and following the branching patterns determined by these characters. The algorithm **TRIE** assumes that **P = 0** is not a branch node and that **KEY(***P***)** is the key value represented in **P** if **P** is an information node. Insertion Let us consider the trie of figure 1. Now insert **SURESH**. A search for ' **SURESH**' in *T*

leads to the node , where

LINK(, 'S') = 0. Hence **SUNDAR** is not in T and may be inserted here (see figure 2). Next, X = bluejay and a search of T leads us to the information node . A comparison indicates

Figure 1: Empty Trie tree a b c d e f g h i j k l m n o p q r s t u v w x y z Data Structure(16CAU301) Unit III BCA(2016-2019 Batch)

Figure 2: Trie tree after inserting SURESH Now insert SURIYA A search for ' SURIYA' in T leads to the node , where **LINK(, 'S') ≠ 0**. A comparison indicates that **KEY**

(SURESH) ≠ SURIIYA. Both **KEY(SURESH)** and **SURIIYA** will form a subtrie of . The two values **KEY(SURESH) ≠ SURIIYA** are sampled until the sampling results in two different values. The happens when the 5th letter of KEY( ) and X are compared. The resulting trie after insertion is in figure 3. Deletion From the trie of figure 4 let us first delete '**SURIYA**.' To do this we just set LINK(,'O') = 0. No other changes need be made. Next, let us delete 'SURESH' If after deletion only one key value is then that node may be deleted and the key may be propagated up one level. The propagation may end when at a node where more than on key value is present. At this junction we cannot be moved up any more levels and we set the corresponding LINK filed to point the key (information node for the key). In order to facilitate deletions from tries, it is useful to add a COUNT field in each branch node. This count field will at all times give us the number of information nodes in the subtree for which it is the root. A B C D E F G H I J K L M N O P Q R S T U V W X Y Z A……U……Z A……R……Z A… E…… ……I……. After deleting key value SURIYA the node has only one key ‘SURAJ’. Hence the branch node is deleted and the key SURAJ is propagated up one levd one more level. As the node up also contain only one key value the key SURAJ is propagated one more level. At this junction the propagation stops because the already has one key value ‘ZAHIR; and inserting SURAJ make the number of keys in that node as 2.

Enable | Enlighten | Enrich
KARPAGAM
ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)

Karpagam Academy of Higher Edu
Department of Computer Applicati
Subject : Data Struc
Class: II BCA **Sub**
Objective Type Ques

UNIT III

S.No	QUESTION	OPT 1
1	To add an item into the queue,	FRONT is incremented by one
2	In Queue FRONTis incremented then, the operation performed on it is _____.	DelQ
3	Which of the following is a valid linear data structure.	Stacks
4	A _____is a linear list in which elements can be inserted and deleted at both ends but not at the Middle	Queue
5	A _____is a collection of elements such that each element has been assigned a priority.	Priority Queue
6	A _____is made up of Operators and Operands.	Stack
7	A _____is a procedure or function which calls itself.	Stack
8	An example for application of stack is _____.	Time sharing computer system
9	An example for application of queue is _____.	Stack of coins
10	_____is efficient solution to avoid data movement in array representation of Queue.	Circular Queue
11	What is the strategy of Stack?	LILO
12	In queue we can add elements at _____.	Top
13	In queue we can delete elements at _____.	Front
14	In Stack we can add elements at _____.	Bottom
15	In Stack we can delete elements at _____	Front

16	When Top = Bottom in stack, the total no of element in the stack is	1
17	When FRONT = REAR in queue, the total no of element in the queue is	0
18	In Stack the TOP is decremeted by one after every ____operation.	AddQ
19	In Stack the TOP is incremeted by one before every ____operation.	AddQ
20	data structure used for recursion is	stack
21	The data structure required to check whether an expression contains balanced parenthesis is?	stack
22	removing the element from the top of the stack is called the	push
23	What data structure would you mostly likely see in a non recursive implementation of a recursive algorithm?	stack
24	which data structure is used to implement the queue most efficiently	array

cation
ons
tures
ject code: 18CAU301
stions

OPT 2	OPT 3	OPT 4	ANSWER
FRONT is decremented by one	REAR is decremented by one	REAR is incremeted by one	REAR is incremeted by one
Pop	Push	AddQ	DelQ
Records	Trees	Graphs	Stacks
DeQueue	Enqueue	Priority Queue	DeQueue
De Queue	Circular Queue	En Queue	Priority Queue
Expression	Linked list	Queue	Expression
Recursion	Queue	Tree	Recursion
Waiting Audience	Processing of subroutines	space sharing system	Processing of subroutines
Stack of bills	Processing of subroutines	Job Scheduling in TimeSharing computers	Job Scheduling in TimeSharing computers
Dequeue	Enqueue	queue	Circular Queue
FIFO	FILO	LIFO	LIFO
Bottom	Front	Rear	Rear
Bottom	Top	Rear	Front
Top	Front	Rear	Top
Rear	Top	Bottom	Top

2	3	0	0
1	2	3	0
Pop	Push	DelQ	Pop
Pop	Push	DelQ	Push
queue	array	linked list	stack
queue	array	linked list	stack
pop	delete	remove	pop
queue	array	linked list	stack
linked list	structure	union	linked list

25	The process of accessing data stored in a serial access memory is similar to manipulating data on a ?	stack
26	which data structure is used to implement the double ended queue?	doubly linked list
27	what is the time complexity of inserting n elements in a queue?	O(n)
	The postfix form of A*B+C/D is?	*AB/CD+
29	what is the time complexity of deleting n elements in a double ended queue?	O(n)
30	what is the data strucutre used to implement circular queue?	circular linked list
31	which data structure allows deleting data element from front and inserting at rear?	stack
32	The prefix form of A-B/ (C * D E) is?	-/* ACBDE
33	The prefix form of an infix expression p + q - r * t is?	+ pq - *rt
34	The result of evaluating the postfix expression 5, 4, 6, +, *, 4, 9, 3, /, +, * is?	600
35	Infinite recursion leads to	Overflow of run- time stack
36	What is the result of the following operation Top (Push (S, X))	X
37	Convert the following infix expressions into its ^	(A B D G + E F - / +)

queue	array	linked list	stack
structure	Trees	graphs	doubly linked list
O(n log n)	O(1)	O(n2)	O(n)
AB*CD/+	A*BC+/D		AB*CD/+
O(n log n)	O(1)	O(n2)	O(n)
singly linked list	double linked list	multilist	circular linked list
queue	dequeue	binary search tree	queue
-ABCD* DE	-A/B*C DE	-A/BC* DE	-A/B*C DE
- +pqr * t	- +pq * rt	- + * pqrt	- +pq * rt
350	650	588	350
Underflow of registers usage	Overflow of I/O cycles	Underflow of run-time stack	Overflow of run-time stack
null stack	S	0	X
(A B D + E F - / G +)	(A B D + E F/- G +)	AB+D^+EF/-G+	(A B D + E F - / G +)

38	What would be returned by the following recursive function after we call test (0, 3) int test (int a, int b)
	{
	if (a==b) return (1); else if (a>b) return(0);
	else return (a+test(a+1, b));
	}
		1

39	what is bounded queue?	queue with fixed size of storage
40	what is the data strucutre used to convert a bounded queue to circular queue?	array
	In a growing array what the amortized type of time complexity of all dequeue operations?	constant time
42	…… is very useful in situation when data have to stored and then retrieved in reverse order.	stack
43	Which data structure allows deleting data elements from and inserting at rear?	stack
44	Which of the following data structure is linear type?	stack
45	To represent hierarchical relationship between elements, Which data structure is suitable?	Dequeue
46	The term "push" and "pop" is related to the	stack
47	A data structure where elements can be added or removed at either end but not in the middle	stack
48	The postfix form of A ^ B * C - D + E/ F/ (G + H),	AB^C*D- EF/GH+/+
49	Which of the following statement(s) about stack data structure is/are NOT correct?	Stack data structure can be implemented using linked list
50	Which of the following is not an inherent application of stack?	Reversing a string

51	Consider the following operation performed on a stack of size 5. Push(1); Pop(); Push(2); Push(3); Pop(); Push(4); Pop(); Pop(); Push(5); After the completion of all operation, the no of element present on stack are	1
52	The type of expression in which operator succeeds its operands is?	Infix Expression
53	Which of the following application generally use a stack?	Parenthesis balancing program
54	Consider the linked list implementation of a stack. Which of the following node is considered as Top of the stack?	First node
55	The prefix of (A+B)*(C-D)/E*F	/+-AB*CD
56	Convert the following Infix expression to Postfix form using a stack x + y * z + (p * q + r) * s	xyz*+pq*r+s*+
57	One can convert an infix expression to a postfix expression using a	stack
58	A linear list of elements in which deletion can be done from one end (front) and insertion can take place only at the other end (rear) is known as a	stack
59	Which of the following types of expressions do not require precedence rules for evaluation?	fully parenthesised infix expression

60	What is the postfix form of the following prefix expression -A/B*C$DE	ABCDE$*/-
61	The smallest element of an array's index is called its	lower bound
62	In a circular linked list	components are all linked together in some sequential manner
63	The data structure required to evaluate a postfix expression is	stack
64	What data structure would you mostly likely see in a nonrecursive implementation of a recursive algorithm?	stack
65	What is the strategy of Queue?	LILO
66	Postfix notation is also known as	polish notation
67	give the postfix notation of a+b	a+b
68	Inserting element in a stack is known as	insertion
69	deleting element from a stack is known as	insertion
70	elements are removed from which position of the stack	middle
71	If the pushing consumes all of the space allocated for the call stack, an error called a	stack full
72	convert infix expression A * (B + C) / D to postfix expression	ABCD/+*
73	convert infix expression A * (B + C / D) to postfix expression	ABCD/+*
74	convert infix expression A * (B + C / D) to postfix expression	/ * A+BCD
75	elements are added at which position of the stack	bootom
76	The postfix form of the expression (A+ B)*(C*D-E)*F / G is?	AB+ CD*E - FG /**

2	3	4	4
queue with fixed size of elements	queue with fixed size of reallocation	queue with variable size	queue with fixed size of elements
Stack of bills	queue	tree	array
linear time	exponential time	quadratic time	constant time
queue	list	linked list	stack
queue	list	linked list	queue
tree	graph	binary tree	stack
Prioriy queue	Graph	Tree	Tree
queue	Trees	Graphs	stack
queue	Trees	deque	deque
AB^CD-EP/GH+/+*	ABCDEFGH+//+-*^	AB^D+EFGH +//*+	AB^C*D-EF/GH+/+
New node can only be added at the top of the stack	Stack is the FIFO data structure	The last node at the bottom of the stack has a NULL link	Stack is the FIFO data structure
Evaluation of postfix expression	Implementation of recursion	Job scheduling	Job scheduling

2	3	4	1
pre fix Expression	postfix Expression	Expression	postfix Expression
Recursive program	Factorial	Fibonacci	Parenthesis Balancing program
Last node	Any node	Middle node	First node
/*+-ABCD*EF	*/*+AB- CDEF	**AB+CD/E F	*/*+AB-CDEF
xyz*+pq*r+s+*	xyz+*pq*r+s *+	xyz+*pq*r+s +*	xyz*+pq*r+s*+
queue	Trees	deque	stack
queue	Trees	deque	queue
postfix expression	partially parenthesised infix expression	Prefix expression	fully parenthesised infix expression

A-BCDE$*/-	ABC$ED*/-	A-BCDE$*/	ABCDE$*/-
upper bound.	range	extraction	lower bound
there is no beginning and no end.	components are arranged hierarchically .	) forward and backward traversal within the list is permitted.	there is no beginning and no end.
queue	list	linked list	stack
queue	list	linked list	stack
FIFO	FILO	LIFO	FIFO
suffix notation	infix notation	default notation	suffix notation
ab+	ab	#NAME?	ab+
push	pop	addition	push
push	pop	addition	pop
top	bottom	front	top
null stack	stack empty	stack overflow	stack overflow
ABC+D/*	ABC+*D/	AB*C+D/	ABC+*D/
ABC+D/*	ABC+*D/	AB*C+D/	ABCD/+*
*A+B/CD	*+AB/CD	*+ABCD/	*A+B/CD
top	front	rear	top
AB + CD* E - F **G /	AB + CD* E *F *G /	AB + CDE * - * F *G /	AB+ CD*E - FG /**

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY
SUBJECT NAME: DATA STRUCTURES
SUBJECT CODE: 18CAU301
UNIT –IV

A graph is a mathematical structure consisting of a set of vertices (also called nodes) and a set of edges . An edge is a pair of vertices . Graphs are used to model real-world systems such as the Internet (each node represents a router and each edge represents a connection between routers); airline connections (each node is an airport and each edge is a flight); or a city road network (each node represents an intersection and each edge represents a block). The wireframe drawings in computer graphics are another example of graphs. A graph G0 = (V 0;E0) is called a subgraph of G = (V;E) if V' _ V and E0 _ E \ _V 02 _. A graph may be either undirected or directed. An undirected edge models is a "two-way" or "duplex" connection between its endpoints, v1 v2 v3 The pairs (v1, v2) and (v2, v1) represent the same. A directed edge is a one-way connection, and is typically drawn as an arrow. v1 V2 v3 edge is represented by a directed pair (v1, v2). v1 is the tail and v2 the head of the edge. Therefore <v2, v1> and <v1, v2> represent two different edges. The number of edges with one endpoint on a given vertex is called that vertex's degree. In a directed graph, the number of edges that point to a given vertex is called its in-degree, and the number that point from it is called its
out-degree. Node in degree out degree
A 01 01
B 00 02
C 02 00 We may also want to associate some cost or weight to the traversal of an edge. When we add this information, the graph is called weighted. Data Structures(16CAU301) Unit IV BCA(2016-2019 Batch) R.NITHYA Department of CA,CS,IT KAHE 2 / 11 The number of distinct unordered pairs (vi,vj) with vi vj in a graph with n vertices is n(n - 1)/2. This is the maximum number of edges in any n vertex undirected graph. An n vertex undirected graph with exactly n(n - 1)/2 edges is said to be complete. A path from vertex vp to vertex vq in graph G is a sequence of vertices vp,vi1,vi2, ...,vin,vq such that (vp,vi1),(vi1,vi2), ...,(vin,vq) are edges in E(G). If G' is directed then the path consists of <vp,vi1>,<vi,vi2>, ...,<vin,vq>, edges in E(G'). The length of a path is the number of edges on it. A simple path is a path in which all vertices except possibly the first and last are distinct. A cycle is a simple path in which the first and last vertices are the same. Simple path: H-A-B Closed path or walk contains repeated vertices. Eg, B-D-E-F-D-C-B. Length of the path is 6 Cycle contains no repeated vertex. Eg. D-H-G An undirected graph is said to be connected if for every pair of distinct vertices vi, vi in V(G) there is a path from vi to vj in G. A connected component or simply a component of an undirected graph is a maximal connected subgraph. A directed graph G is said to be strongly connected if for every pair of distinct vertices vi, vj in V(G) there is a directed path from vi to vj and also from vj to vi. In other words Strongly connected directed graph has a path from all vertices to all vertices. Data Structures(16CAU301) Unit IV BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE 3 / 11 A G is strong connected as all vertices can be reached from all other vertices. H is not, as we cannot reach any vertex from a.
Representation of graph. Let G = (V,E) be a graph with n vertices, n 1. The adjacency matrix of G is a 2-dimensional n n array, say A, with the property that A(i,j) = 1 iff the edge (vi,vj) (<vi,vj> for a directed

graph) is in E(G). A(i,j) = 0 if there is no such edge in G. 1 2 3 4 1 0 1 1 1 2 1 0 1 1 3 1 1 0 1 4 1 1 1 0 The adjacency matrix for an undirected graph is symmetric as the edge (vi,vj) is in E(G) iff the edge (vj,vi) is also in E(G). The adjacency matrix for a directed graph need not be symmetric. From the adjacency matrix, one may readily determine if there is an edge connecting any two vertices I and j. For an undirected graph the degree of any Data Structures(16CAU301) Unit IV BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE 4 / 11 vertex i is its row For a directed graph the row sum is the out-degree while the column sum is the in-degree..
Adjacency Lists In this representation the n rows of the adjacency matrix are represented as n linked lists. There is one list for each vertex in G. The nodes in list i represent the vertices that are adjacent from vertex i. Each node has at least two fields: VERTEX and LINK. The VERTEX fields contain the indices of the vertices adjacent to vertex i The degree of any vertex in an undirected graph may be determined by just counting the number of nodes in its adjacency list. The out-degree of any vertex may be determined by counting the number of nodes on its adjacency list. Determining the in-degree of a vertex is a little more complex. In case there is a need to repeatedly access all vertices adjacent to another vertex.
Adjacency Multilist In the adjacency list representation of an undirected graph each (vi,vj) is represented by two entries, one on the list for vi and other on the list for vj. But in adjacency multilist structure for each edge there will be exactly one node, but this node will be in two lists. i.e., the adjacency lists of the two nodes it is incident to.
Depth First Search Depth first search of an undirected graph proceeds as follows. The start vertex v is visited. Next an unvisited vertex w adjacent to v is selected and a depth first search from w initiated. When a vertex u is reached such that all its adjacent vertices have been visited, we back up to the last vertex visited which has an unvisited vertex w adjacent to it and initiate a depth first search from w. The search terminates when no unvisited vertex can be reached from any of the visited one. This procedure is best described recursively as in

```
procedure DFS(v) //Given an undirected graph G = (V,E) with n vertices and an array VlSlTED(n) initially set to zero, this algorithm visits all vertices reachable from v. G and VISITED are global.//
VISITED (v) ←1
for each vertex w adjacent to v do
if VISlTED(w) = 0 then call DFS(w)
end
end DFS
```

Breadth First Search Starting at vertex v and marking it as visited, breadth first search differs from depth first search in that all unvisited vertices adjacent to v are visited next. Then unvisited vertices adjacent to these vertices are visited and so on. A Data Structures(16CAU301) Unit IV BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE 5 / 11 breadth first search beginning at vertex v1 of the graph in figure 6.13(a) would first visit v1 and then v 2 and v 3. Next vertices v 4, v 5, v 6 and v 7 will be visited and finally v 8.

```
procedure BFS(v) //A breadth first search of G is carried out beginning at vertex v. All vertices visited are marked as VISlTED(i)= 1. The graph G and array VISITED are global and VISITED is initialized to zero.//
VISITED (v) ←1
initialize Q to be empty //Q is a queue//
loop
for all vertices w adjacent to v do
if VISITED(w) = 0 then
call ADDQ(w,Q); VISITED(w) ←1
```

end
if Q is empty then return
call DELETEQ(v,Q)
forever
end BFS
Content of queue Q Node deleted from Q Adjacent Node(s) not yet
and visited visited & hence added to Q
Q →
Q → ←
Q→ ← ←
Content of queue Q Node deleted from Q Adjacent Node(s) not yet
and visited visited & hence added to Q
Q→ ← ←← ←
Q→ ← ←← ←
Q→ ← ←←
Q→ ← Data Structures(16CAU301) Unit IV BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE 6 / 11
Q→ Q=Null A spanning tree T of a connected, undirected graph G is a tree composed of all the vertices and some (or perhaps all) of the edges of G. Informally, a spanning tree of G is a selection of edges of G that form a tree spanning every vertex. That is, every vertex lies in the tree, but no cycles (or loops) are formed. A spanning tree of a connected graph G can also be defined as a maximal set of edges of G that contains no cycle, or as a minimal set of edges that connect all vertices. Few spanning trees formed for the graph given above is listed below A minimum spanning tree (MST) is a spanning tree with weight less than or equal to the weight of every other spanning tree.
Kruskal Algorithm In this approach a minimum cost spanning tree, T, is built edge by edge. Edges are considered for inclusion in T in nondecreasing order of their costs. An edge is included in T if it does not form a cycle with the edges already in T. Since T is connected and has n>0 vertices, exacty n-1 edges will be selected for inclusion in T. As an example consider the graph given below.
while T contains less than n - 1 edges and E not empty do
choose an edge (v,w) from E of lowest cost;
delete (v,w) from E;
if (v,w) does not create a cycle in T
then add (v,w) to T
else discard (v,w)
end
if T contains fewer than n - 1 edges then print ('no spanning tree')
Single Source all Destinations shortest path In this problem we are given a directed graph G = (V,E), a weighting function w(e) for the edges of G and a source vertex v
o. The problem is to determine the shortest paths from vo to all the remaining vertices of G. It is assumed that all the weights are positive. The numbers on the edges are the weights. If we attempt to devise an algorithm which generates the shortest paths in nondecreasing order, then we can make several observations. Let Data Structures(16CAU301) Unit IV BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE 7 / 11
S denote the set of vertices (including vo) to which the shortest paths have already been found. For w not in S, let

DIST(w) be the length of the shortest path starting from vo going through only those vertices which are in S and ending at w. We observe that: (i) If the next shortest path is to vertex u, then the path begins at vo, ends at u and goes through only those vertices which are in S. (ii) The destination of the next path generated must be that vertex u which has the minimum distance, DIST(u), among all vertices not in S. This follows from the definition of DIST and observation (i). In case there are several vertices not in S with the same DIST, then any of these may be selected. (iii) Having selected a vertex u as in (ii) and generated the shortest vo to u path, vertex u becomes a member of S. At this point the length of the shortest paths starting at vo, going through vertices only in S and ending at a vertex w not in S may decrease. I.e., the value of DIST(w) may change. If it does change, then it must be due to a shorter path starting at vo going to u and then to w. The intermediate vertices on the vo to u path and the u to w path must all be in S. Further, the vo to u path must be the shortest such path, otherwise DIST(w) is not defined properly. Also, the u to w path can be chosen so as to not contain any intermediate vertices. Therefore, we may conclude that if DIST(w) is to change (i.e., decrease), then it is because of a path from vo to u to w where the path from vo to u is the shortest such path and the path from u to w is the edge <u,w>. The length of
this path is DIST(u) + length (<u,w>). In the algorithm SHORTEST_PATH it is assumed that the n vertices of G are numbered 1 through n. The set S is maintained as a bit array with S(i) = 0 if vertex i is not in S and S(i) = 1 if it is. It is assumed that the graph itself is represented by its cost adjacency matrix with COST(i,j) being the weight of the edge <i,j>. COST(i,j) will be set to some large number, +∞ , in case the edge <i,j> is not in E(G). For i= j, COST(i,j) may be set to any non-negative number without affecting the outcome of the algorithm.

procedure SHORTEST-PATH (v,COST,DIST,n) //DIST(j), 1 j n is set to the length of the shortest path from vertex v to vertex j in a digraph G with n vertices.
DIST(v) is set to zero. G is represented by its cost adjacency matrix, COST(n,n)//
declare S (1: n)
for i 1 to n do //initialize set S to empty//
S(i) ←0; DIST(i) ←COST(v,i)
end
S(v)← 1; DIST(v)← 0; num ←2 //put vertex v in set S//
while num < n do //determine n - 1 paths from vertex v//
choose u: DIST(u) ← min {DIST(w)}
(w)=0
S(u)← 1; num num + 1 //put vertex u in set S//
for all w with S(w) = 0 do //update distances//
DIST(w)← min {DIST(w),DIST(u) + COST(u,w)}
end
end
end SHORTEST-PATH Consider the 8 vertex digraph given below The cost matrix of the above 8 vertex digraph 7 ∞ Min(DIST(7), DIST(6)+COST (6,7)) i.e.Min(∞,250+900) As Min(∞,1150) is 1150 DIST(7) is changed to 1150 8 ∞ Min(DIST(8), DIST(6)+COST (6,8)) As Min(∞, 1650) is 1650 DIST(8) is changed to 1650 Data Structures(16CAU301) Unit IV BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE 8 / 11 i.e.Min(∞,250+1400) 7 8 1650 Min(DIST(8), DIST(7)+COST (7,8)) Min(1650,1150+ 1000) As Min(1650,2150) IS 1650 DIST(8) remains unchanged 4 3 ∞ Min(DIST(3), DIST(4)+COST (4,3)) Min(∞,1250+1200) As Min(∞, 2450) is 2450 DIST(3) is changed to 2450 8 1 ∞ Min(DIST(1), DIST(8)+COST (8,1)) Min(∞,1650+1700) As Min(∞, 3350) is DIST(1) is changed to 3350 3 1 3350 Min(DIST(1), DIST(3)+COST (3,1)) Min(3350 ,2450+1000) As Min(3350<3550

) is 3350 DIST(1) remains unchanged 2 ∞ Min(DIST(2), DIST(3)+COST (3,2)) Min(∞,2450+800) As Min(∞, 3250) is DIST(2) is changed to 3250

All Pairs shortest path The all pairs shortest path problem calls for finding the shortest paths between all pairs of vertices vi,vj, i≠ j. The graph G is represented by its cost adjacency matrix with COST(i.i)= 0 and COST(i,j) = +∞ in case edge <i,j>, i≠ j is not in G. Define Ak (i,j) to be the cost of the shortest path from i to j going through no intermediate Data Structures(16CAU301) Unit IV BCA(2016-2019 Batch) R.NITHYA Department of CA,CS,IT KAHE 9 / 11 vertex of index greater than k. Then, An(i,j) will be the cost of the shortest i to j path in G since G contains no vertex with index greater than n. Ao(i,j) is just COST(i,j) since the only i to j paths allowed can have no intermediate vertices on them. The basic idea in the all pairs algorithm is to successively generate the matrices A0, A1, A2, ...,An. If we have already generated Ak-1, then we may generate Ak by realizing that for any pair of vertices i,j either (i) the shortest path from i to j going through no vertex with index greater than k does not go through the vertex with index k and so its cost is Ak-1(i,j); or (ii) the shortest such path does go through vertex k. Such a path consists of a path from i to k and another one from k to j. These paths must be the shortest paths from i to k and from k to j going through no vertex with index greater than k - 1, and so their costs are Ak-1(i,k) and Ak-1(k,j). Thus, we obtain the following formulas for Ak (i,j):

Ak(i,j) = min {Ak-1(i,j), Ak-1(i,k) + Ak-1 (k,j)} k ≥1 ----------(1) and

Ao(i,j) = COST(i,j).

procedure ALL_COSTS(COST,A,n) // COST(n,n) is the cost adjacency matrix of a graph with n vertices; A(i,j) is the cost of the shortest path between vertices vi,vj. COST(i,i) = 0, 1 i n//
for i 1 to n do
for j 1 to n do A (i,j) ←COST (i,j) //copy COST into A//
end
end
for k 1 to n do //for a path with highest vertex index k//
for i 1 to n do //for all possible pairs of vertices//
for j 1 to n do A (i,j) ←min {A (i,j),A(i,k) + A(k,j)}
end
end
end
end ALL_COSTS Source vertex i Destinatio n vertex j Intermedia te vertex k Equation 1 A1 (i,j)= Min{ A1 (i,j), A1 (i,j)+ A1 (i,j)} Action 1 1 1 A1 (1,1)= Min{ A0(1,1), A0(1,1)+ A0 (1,1) } A1 (1,1)= Min{0,0+ 0} 2 1 A1 (1,2)= Min{ A0(1,2), A0(1,1)+ A0 (1,2)} A1 (1,2)= Min{ 4,0+ 4} 3 1 A1 (1,3)= Min{ A0(1,3), A0(1,1)+ A0 (1,3) A1 (1,3)= Min{11, 0+ 11} 2 1 1 A1 (2,1)= Min{ A0(2,1), A0(2,1)+ A0 (1,1) } A1 (2,1)= Min{ 6, 6 + 0} 2 1 A1 (2,2)= Min{ A0(2,2), A0(2,1)+ A0 (1,2)} A1 (2,2)= Min{ 0, 6 + 0} 3 1 A1 (2,3)= Min{ A0(2,3), A0(2,1)+ A0 (1,3)} A1 (2,3)= Min{ 6 , 0 + 11} Data Structures(16CAU301) Unit IV BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE 10 / 11 3 1 1 A1 (3,1)= Min{ A0(3,1), A0(3,1)+ A0 (1,1)} A1 (3,1)= Min{ 3, 3+0} 2 1 A1 (3,2)= Min{ A0(3,2), A0(3,1)+ A0 (1,2)} A1 (3,2)= Min{ +∞, 3+ 4} Old value of A1 (3,2)= +∞ New value of A1 (3,2) =7 3 1 A1 (3,3)= Min{ A0(3,3), A0(3,1)+ A0 (1,3)} A1 (3,3)= Min{ 0, 3 + 11} Source vertex i Destinatio n vertex j Intermedia te vertex k Equation 1 A1 (i,j)= Min{ A1 (i,j), A1 (i,j)+ A1 (i,j)} Action 1 1 2 A2 (1,1)= Min{ A1(1,1), A1(1,2)+ A1 (2,1) } A2 (1,1)= Min{0,4+ 6} 2 2 A2 (1,2)= Min{ A1(1,2), A1(1,2)+ A1 (2,2)} A2 (1,2)= Min{ 4,4+ 0} 3 2 A2 (1,3)= Min{ A1(1,3), A1(1,2)+ A1 (2,3) A2 (1,3)= Min{11, 4+ 2} Old value of A1 (1,3)= 11 New value of A1 (1,3) =6 2 1 2 A2 (2,1)= Min{

A1(2,1), A1(2,2)+ A1 (2,1) } A2 (2,1)= Min{ 0, 0 + 6} 2 2 A2 (2,2)= Min{ A1(2,2), A1(2,2)+ A1 (2,2)} A2 (2,2)= Min{ 0, 0 + 0} 3 2 A2 (2,3)= Min{ A1(2,3), A1(2,2)+ A1 (2,3)} A2 (2,3)= Min{ 2 , 0 + 2} 3 1 2 A2 (3,1)= Min{ A1(3,1), A1(3,2)+ A1 (2,1)} A2 (3,1)= Min{ 3, 7+6} 2 2 A2 (3,2)= Min{ A1(3,2), A1(3,2)+ A1 (2,2)} A2 (3,2)= Min{ 7, 7+ 0} 3 2 A2 (3,3)= Min{ A1(3,3), A1(3,2)+ A1 (2,3)} A2 (3,3)= Min{ 0, 7+ 2} Source vertex i Destinatio n vertex j Intermedia te vertex k Equation 1 A1 (i,j)= Min{ A1 (i,j), A1 (i,j)+ A1 (i,j)} Action 1 1 3 A3 (1,1)= Min{ A2(1,1), A2(1,3)+ A2 (3,1) } A3 (1,1)= Min{0,6+ 3} 2 3 A3 (1,2)= Min{ A2(1,2), A2(1,3)+ A2 (3,2)} A3 (1,2)= Min{ 4,6+ 7} Data Structures(16CAU301) Unit IV BCA(2016-2019 Batch)
R.NITHYA Department of CA,CS,IT KAHE 11 / 11 3 3 A3 (1,3)= Min{ A2(1,3), A2(1,3)+ A2 (3,3) A3 (1,3)= Min{6, 6+ 0} 2 1 3 A3 (2,1)= Min{ A2(2,1), A2(2,3)+ A2 (3,1) } A3 (2,1)= Min{ 6, 2 + 3} Old value of A1 (2,1)= 6 New value of A1 (2,1) =5 2 3 A3 (2,2)= Min{ A2(2,2), A2(2,3)+ A2 (3,2)} A3 (2,2)= Min{ 0, 6 + 7} 3 3 A3 (2,3)= Min{ A2(2,3), A2(2,3)+ A2 (3,3)} A3 (2,3)= Min{ 2 , 2 + 0} 3 1 3 A3 (3,1)= Min{ A2(3,1), A2(3,3)+ A2 (3,1)} A3 (3,1)= Min{ 3, 3+0} 2 3 A3 (3,2)= Min{ A2(3,2), A2(3,3)+ A2 (3,2)} A3 (3,2)= Min{ +∞, 0+ 7} 3 3 A3 (3,3)= Min{ A2(3,3), A2(3,3)+ A2 (3,3)} A3 (3,3)= Min{ 0, 0 + 0}

Enable | Enlighten | Enrich
KARPAGAM
ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)

Karpagam Academy of Higher Edu
Department of Computer Applicati
Subject : Data Struc
Class: II BCA **Sub**
Objective Type Ques
UNIT IV

S.No	QUESTION	OPT 1
1	Which of the following operations is performed more efficiently by doubly linked list than by singly linked list	Deleting a node whose location is given.
2	Nodes that have degree zero are called _____.	end node
3	A binary tree with all its left branches supressed is called a _____	balanced tree
4	All node except the leaf nodes are called_____.	terminal node
5	The roots of the subtrees of a node X, are the _____of X.	Parent
6	X is a root then X is the _____of its children.	sub tree
7	The children of the same parent are called _____.	sibiling
8	_____of a node are all the nodes along the path form the root to that node.	Degree
9	The _____of a tree is defined to be a maximum level of any node in the tree.	weight
10	A_____is a set of n ≥ 0 disjoint trees	Group
11	A tree with any node having at most two branches is called a _____.	branched tree
12	A _____of depth k is a binary tree of depth k having 2^K-1 nodes.	full binary tree
13	Data structure represents the hierarchical relationships between individual data item is known as _____.	Root
14	Node at the highest level of the tree is known as _____.	Child
15	The root of the tree is the _____of all nodes in the tree.	Child

16	_____is a subset of a tree that is itself a tree.	Branch
17	A node with no children is called _____.	Root Node
18	In a tree structure a link between parent and child is called _____	Branch
19	Height – balanced trees are also referred as as _____trees.	AVL trees
20	Visiting each node in a tree exactly once is called _____	searching
21	In_____traversal ,the current node is visited before the subtrees.	PreOrder
22	In_____traversal ,the node is visited between the subtrees.	PreOrder
23	In_____traversal ,the node is visited after the subtrees.	PreOrder
24	Inorder traversal is also sometimes called_____	Symmetric Order
25	Postorder traversal is also sometimes called_____	Symmetric Order
26	One can determine whether a Binary tree is a Binary Search Tree by traversing it in	Preorder
27	Nodes of any level are numbered from _____	Left to right
28	In Threaded Binary Tree ,LCHILD(P) is a normal pointer When LBIT(P) = _____	1
29	In Threaded Binary Tree ,LCHILD(P) is a Thread When LBIT(P) = _____	1
30	In Threaded Binary Tree ,RCHILD(P) is a normal pointer When RBIT(P) = _____	2
31	In Threaded Binary Tree ,RCHILD(P) is a Thread When LBIT(P) = _____	1
32	Which of these searching algorithm uses the Divide and Conquere technique for sorting	Linear search
33	_____algorithm can be used only with sorted lists.	Linear search
34	_____search involves comparision of the element to be found with every elements in a list.	Linear search
35	Binary search algorithm in a list of n elements takes only _____time.	$O(\log_2 n)$
36	_____is used for decision making in eight coin problem.	trees

37	The Linear search algorithm in a list of n element takes _____time to compare in worst case.	constant
38	In _____search method the search begins by examining the record in the middle of the file.	sequential
39	A binary tree with external nodes added is an ---- ---------- binary tree	extended
40	The search technique for searching a sorted file that requires increased amount of space is	indexed sequential search
41	If hl and hr are the heights of the left and right subtrees of a tree respectively and if |hl-hr|<=1 then this tree is called _____	extended binary tree
42	If hl and hr are the heights of the left and right subtrees of a tree respectively then |hl-hr| is called its _____	Average height
43	For an AVL Tree the balance factor is =_____	0
44	In a binary search tree all values of the left subtree is _____than the root's value	smaller
45	In a binary search tree all values of the right subtree is _____than the root's value	smaller
46	In BST, we can search for a value in	O(log n)
47	In tree construction, which is the suitable efficient data structure?	array
48	In a balance binary tree, the height of two subtrees of every node cannot differ by more than?	4
49	The number of possible binary trees with 3 nodes is	15
50	the number of possible binary trees with 4 nodes is	14
51	the order of binary search algorithm is	n
52	A connected graph T without any cycle is called	Tree
53	A binary tree with 10 nodes has _____null branches	12
54	binary search algorithm can be applied to	sorted binary trees
55	If aaa, bbb and ccc are the elements of a lexically ordered binary tree, then in pre order traversal which node will be traversed first?	aaa

56	In an array representation of binary tree the right child of the root will be at location of	1
57	The maximum number of nodes in a binary tree of depth 5 is	31
58	A complete binary tree with n leaf nodes has	n+1 nodes
59	In a binary tree, certain null entries are replaced by special pointers which point to nodes higher in tree for efficiency. These special pointers are called	leaf
60	A binary tree whose every node has either zero or two children is called	complete binary tree
61	when cinverting binary tree into extended binary tree, all the original nodes in binary tree are	internal nodes on extended binary tree
62	The inorder traversal will yield a sorting list of elements of tree in	binary tree
63	Which of the following traversal technique lists the nodes of a binary search tree in ascending order?	postorder
64	Which of the following need not be a binary tree?	B-Tree
65	A binary tree can be converted in to its mirror image by traversing it in	postorder
66	The prefix form of A-B/ (C * D ^ E) is,	-/*^ACBDE
67	Consider that n elements are to be sorted. What is the worst case time complexity of Bubble sort?	O(1)
68	Which of these searching algorithm uses the Divide and Conquere technique for sorting	Linear search
69	_____are genealogical charts which are used to present the data	Graphs
70	A ___is a finite set of one or more nodes, with one root node and remaining form the disjoint sets forming the subtrees.	tree
71	A _____is a graph without any cycle.	tree
72	In binary trees there is no node with a degree greater than _____	zero

73	Which of this is true for a binary tree.	It may be empty
74	Overflow condition in linked list may occur when attempting to_____	Create a node when free space pool is empty.
75	The Number of subtrees of a node is called its _____.	leaf

cation
ons
tures
ject code: 18CAU301
stions

OPT 2	OPT 3	OPT 4	ANSWER
Searching of an unsorted list for a given item.	Inserting a new node after node whose location is given.	Traversing the list to process each node.	Deleting a node whose location is given.
leaf nodes	subtree	root node	leaf nodes
left sub tree	full binary tree	right skewed tree	right skewed tree
percent node	non terminal	children node	non terminal
Children	Sibling	sub tree	Children
Parent	Sibilings	subordinate	Parent
leaf	child	subtree	sibiling
sub tree	Ancestors	parent	Ancestors
length	breath	height	height
forest	Branch	sub tree	forest
sub tree	binary tree	forest	binary tree
half binary tree	sub tree	n branch tree	full binary tree
Node	Tree	Address	Tree
Root	Sibiling	Parent	Root
Parent	Ancestor	Head	Ancestor

Root	Leaf	Subtree	Subtree
Branch	Leaf Node	Null tree	Leaf Node
Root	Leaf	Subtree	Branch
Binary Trees	Subtree	Branch Tree	AVL trees
travering	walk through	path	travering
PostOrder	Inorder	End Order	PreOrder
PostOrder	Inorder	End Order	Inorder
PostOrder	Inorder	End Order	PostOrder
End Order	PreOrder	PostOrder	Symmetric Order
End Order	PreOrder	PostOrder	End Order
Inorder	Postorder	Any order	Inorder
Right to Left	Top to Bottom	Bottom to Top	Left to right
2	3	0	1
2	3	0	0
1	3	0	1
2	0	4	0
Binary search	fibonacci search	m-way search	Binary search
Binary search	insertion sort	merge sort	Binary search
Binary search	fibonacci search	m-way search	Linear search
O(n)	$O(n^3)$	$O(n^2)$	$O(\log_2 n)$
graphs	linked lists	array	trees

linear	quadratic	exponential	constant
fibonacci	binary	non- sequential	binary
expanded	internal	external	extended
interpolation search	sequential search	tree search	indexed sequential search
binary search tree	skewed tree	height balanced tree	height balanced tree
minimal depth	Maximum levels	Balance factor	Balance factor
2	3	4	0
larger	equal	multiples	smaller
larger	equal	multiples	larger
O(n)	O(n2)	O(1)	O(log n)
linked list	stack	queue	linked list
8	5	3	8
10	8	5	5
10	15	12	14
log n	n log n	1	log n
stack	queue	graph	tree
11	21	22	21
sorted graph	pointer array	sorted linked list	sorted linked list
bbb	ccc	cannot be determined	aaa

2	3	0	3
16	32	15	31
2n-1 nodes	2n+1 nodes	n(n-1)/2 nodes	2n-1 nodes
path	branch	thread	thread
extended binary tree	binary search tree	all of above	extended binary tree
external nodes on extended binary tree	vanished on extended binary tree	vanished on b tree	internal nodes on extended binary tree
binary search tree	heaps	AVL tree	binary search tree
inorder	preorder	insertion	inorder
AVL tree	heaps	Search tree	B-Tree
inorder	preorder	insertion	preorder
-ABCD*^DE	-A/B*C^DE	-A/BC*^DE	-A/B*C^DE
O(log2n)	O(n)	O(n2)	O(n2)
Binary search	fibonacci search	Factorial	Binary search
Pedigree and lineal chart	Line , bar chart	pie chart	Pedigree and lineal chart
graph	list	set	tree
path	set	list	tree
one	two	three	two

The degree of all nodes must be <=1	It contains a leaf node	The degree of all nodes must be <2	It may be empty
Traverse the nodes when free space pool is empty.	Create a node when linked list is empty.	Search for node	Create a node when free space pool is empty.
terminal	children	degree	degree

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY
SUBJECT NAME: DATA STRUCTURES
SUBJECT CODE: 18CAU301
UNIT V

HASHING

Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data.

Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from. Hashing

Hashing is a technique to convert a range of key values into a range of indexes of an array. We're going to use modulo operator to get a range of key values. Consider an example of hash table of size 20, and the following items are to be stored. Item are in the (key,value) format.

(1,20)
(2,70)
(42,80)
(4,25)
(12,44)
(14,32)
(17,11)
(13,78)
(37,98) Sr. No. Key Hash Array Index
1 1 1 % 20 = 1 1
2 2 2 % 20 = 2 2
3 42 42 % 20 = 2 2
4 4 4 % 20 = 4 4
5 12 12 % 20 = 12 12
6 14 14 % 20 = 14 14
7 17 17 % 20 = 17 17
8 13 13 % 20 = 13 13
9 37 37 % 20 = 17 17 Linear Probing

As we can see, it may happen that the hashing technique is used to create an already used index of the array. In

such a case, we can search the next empty location in the array by looking into the next cell until we find an empty
cell. This technique is called linear probing. Sr. No. Key Hash Array Index After Linear Probing, Array Index
1 1 1 % 20 = 1 1 1
2 2 2 % 20 = 2 2 2
3 42 42 % 20 = 2 2 3
4 4 4 % 20 = 4 4 4
5 12 12 % 20 = 12 12 12
6 14 14 % 20 = 14 14 14
7 17 17 % 20 = 17 17 17
8 13 13 % 20 = 13 13 13
9 37 37 % 20 = 17 17 18 Basic Operations
Following are the basic primary operations of a hash table. Search − Searches an element in a hash table. Insert − inserts an element in a hash table. delete − Deletes an element from a hash table. DataItem
Define a data item having some data and key, based on which the search is to be conducted in a hash table.

```
struct DataItem {
int data;
int key;
}; Hash Method
```

Define a hashing method to compute the hash code of the key of the data item.

```
int hashCode(int key){
return key % SIZE;
} Search Operation
```

Whenever an element is to be searched, compute the hash code of the key passed and locate the element using that
hash code as index in the array. Use linear probing to get the element ahead if the element is not found at the computed hash code. Example

```
struct DataItem *search(int key) {
//get the hash
int hashIndex = hashCode(key);
//move in array until an empty
while(hashArray[hashIndex] != NULL) {
if(hashArray[hashIndex]->key == key)
return hashArray[hashIndex];
//go to next cell
++hashIndex;
//wrap around the table
hashIndex %= SIZE;
}
return NULL;
} Insert Operation
```

Whenever an element is to be inserted, compute the hash code of the key passed and locate the index using that
hash code as an index in the array. Use linear probing for empty location, if an element is found at the

computed
hash code. Example

```
void insert(int key,int data) {
struct DataItem *item = (struct DataItem*) malloc(sizeof(struct DataItem));
item->data = data;
item->key = key;
//get the hash
int hashIndex = hashCode(key);
//move in array until an empty or deleted cell
while(hashArray[hashIndex] != NULL && hashArray[hashIndex]->key != -1) {
//go to next cell
++hashIndex;
//wrap around the table
hashIndex %= SIZE;
}
hashArray[hashIndex] = item;
}
```

Delete Operation

Whenever an element is to be deleted, compute the hash code of the key passed and locate the index using that
hash code as an index in the array. Use linear probing to get the element ahead if an element is not found at the
computed hash code. When found, store a dummy item there to keep the performance of the hash table intact. Example

```
struct DataItem* delete(struct DataItem* item) {
int key = item->key;
//get the hash
int hashIndex = hashCode(key);
//move in array until an empty
while(hashArray[hashIndex] !=NULL) {
if(hashArray[hashIndex]->key == key) {
struct DataItem* temp = hashArray[hashIndex];
//assign a dummy item at deleted position
hashArray[hashIndex] = dummyItem;
return temp;
}
//go to next cell
++hashIndex;
//wrap around the table
```

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY

UNIT I

PART-B

[EACH QUESTION CARRIES 2 MARKS]

1. Define data.
2. Define algorithm.
3. Define data type in data structures.
4. Define array.
5. What is meant by sparse matrix?
6. What is meant by single dimensional array?
7. What is meant by multi dimensional array?
8. Define stack.
9. What is meant by stack representation?
10. How user define an array using stack representation?
11. What is meant by linked representation of an array?
12. How to represent the postfix and prefix expression?

PART – C

[EACH QUESTION CARRIES 6 MARKS]

1.What is meant by algorithm complexity? Explain their types.
2. Explain in detail structured programming constructs
3. Explain about algorithm complexity and algorithm analysis
4. Explain the difference between abstract data type, data type and data structures
5. Define Data Structure .Explain the description of various Data Structures.
6. Explain the different approaches to design an algorithm.
7. Define Data Structure .Explain the description of various Data Structures.
8. Explain the different approaches to design an algorithm.
9. Define abstract data type, atomic type and data type with example.
10. Explain about the structures approach programming in detail

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY

UNIT II

PART – B
[EACH QUESTION CARRIES 2 MRAKS]

1. Define linked list.
2. What is meant by circular linked list?
3. What is double linked list?
4. What is single linked list?
5. What is meant by queue?
6. Write down the types of queue.
7. Define de-queue in linked list.
8. How to represent the linked list using array.
9. What is meant by double ended queue?
10. What is priority queue/

PART-C
[EACH QUESTION CARRIES 6 MARKS]

1. Define Array. Explain their types and their operations with example
2. Discuss Polynomial Manipulation with example.
3. Write note on Sparse Matrix and how to represent them in memory
4. Define double linked list. Explain the operations on double linked list with
5. Write a note on Circular linked lists and its operation.
6. Explain a note on linked list vs arrays
7. Explain the operations of linear linked list with example
8. Write the basic operations of a doubly linked list.
9. Explain linear array operations with example
10. Explain the operations of doubly linked list with example

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY

UNIT III

PART-B

[EACH QUESTION CARRIES 2 MARKS]

1. Define tree in data structures.
2. Write the tree representation using array.
3. Define binary tree.
4. Define threaded binary tree.
5. What is meant by node in a tree.?
6. What is meant by root of a tree?
7. What is child of a tree?
8. Write the representation of a binary tree using the linked list element.

Part – c

[each question carries 6 marks]

1. Write an algorithm to convert infix to postfix expression and explain it with example
2. Write an algorithm for Queue operations
3. Write a C program to implement stack operations
4. Explain in detail about queue operations
5. Explain Circular queue and its implementation of operation on circular queue
6. Define Stack Explain the Representation of stack using linked list
7. How stack is used in converting an infix expression into postfix expression. Explainwith an example
8. Explain how queues can be implemented using Linked list
9. Explain various operations on AVL trees.
10. Explain operations of queue using linked list

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY

UNIT IV

PART – B
[EACH QUESTION CARRIES 2 MARKS]

1. Define sorting technique.
2. Define heap sorting technique.
3. Define selection sortingtechnique.
4. Define searchingtechnique.
5. Define linear searchtechnique.
6. Define binary searchtechnique.
7. Define shell sort technique.

PART – C
[EACH QUESTION CARRIES 6 MARKS]

1. List the common operations on binary and binary search tree. Write an algorithm with example. Explain in detail operations on AVL tree with example
2. Explain detail about tree traversal algorithm with suitable example.
3. Write an algorithm to perform sequential search
4. Define Trees and its types. Explain any tree in detail with its operation.
5. Write an algorithm and program for selection sort
6. Write an algorithm and program for binary search.
7. Explain about threaded binary tree and B-Tree.
8. What is a binary tree? When a binary tree becomes a binary search tree?
9. Write an algorithm to perform indexed sequential search

Karpagam Academy of Higher Education
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
DEPARTMENT OF COMPUTER SCIENCE, APPLICATIONS AND INFORMATION TECHNOLOGY

SUBJECT NAME: DATA STRUCTURES CLASS :II BCA
SUBJECT CODE: 16CAU301 SEMESTER: III

UNIT V
POSSIBLE SIX MARKS

1. Explain Insertion sort with example.
2. Write note on representation of graphs
3. Explain quick sort with example
4. Write an algorithm for finding shortest paths
5. Write an algorithm for heap sort with example
    a. Write an algorithm for Heap sort with example
    b. Write an algorithm for merge sort with example
6. Explain about minimum spanning tree with example
7. Write an algorithm for selection and bucket sort with example.

Reg No. ……

[18CAU301]

Karpagam Academy Of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 64 021
BCA Degree Examination
(For the candidates admitted from 2018 onwards)
THIRD SEMESTER
First Internal Exam July 2019
DATA STRUCTURES

Time: 2 Hours **Maximum: 50 Marks**
Date&Session: **CLASS: II BCA(A&B)**

Part – A (20 X 1 = 20 Marks)

Answer ALL Questions

1.The tool useful for specifying the logical properties of datatype is_____
a)abstract data type b)datatype c)data structure d)algorithm

2.The group of items identified by its own identifier is known as_____
a)member of structure b)structure c)record d)field

3.An order collection of items is called_____
a)stack b)array c)list d)queue

4.The operation to perform add items in stack using_____
a)add element b)pop c)pop the element d)pushdown

5.The operation to perform deleting items in stack using_____
a)pushdownlist b)list the element c)delete item d)push

6.When the stack has zero elements then it can be called as_____
a)stack zero b)zero element c)no element d)stack empty

7.The operator "+"between the operand A and B is called_____
a)infix b)precedence c)add the operand d)combine operand

8.An ordered collection of items from the front is called_____
a)queue b)double link c)priority queue d)descending queue

9._____is any data representation and its associated operations.
a)data structure b)datatype c)data declaration d)object.

10.Which one of the following is a LIFO list?
a)Stack b)Queue c)Linked list d)Circular list

11.The items may be inserted at one end of queue which is called
a)front b)rear c)top d)avail

12. The Complexity of linear search

a)O(n) b)O(log n) c)O(n log n) d)O(n^2)

13.In linear list each node has pointers to point to the predecessor and successors then node is called_____

a)singly linked list b)circular linked list c)doubly linked list d)linear linked list

14. A linear collection of data elements where the linear node is given by means of pointer is called

a) linked list b) node list c) primitive list d)array

15.The expression AB/C-DE*+AC*- is called

a)postfix b)prefix c)infix d)expn

16.A BST is traversed in the following order recursively: Right, root, left . The output sequence will be in

a) Ascending order b) Descending order c) Bitomic sequence d) No specific order

17.The expression AB/C-DE*+AC*- is called

a)postfix b)prefix c)infix d)expn

18.Stacks and queues are

a)primitive data structure b)non-primitive data structure
c)non-linear data structure d)data types

19. Which one is not Divide and Conquer algorithm?

a)2-way merge sort b)quick sort c)selection sort d)insertion sort

20. Stack organization is

a)FIFO b)LILO c)LIFO d)FILO

Part – B(3X 2= 6 Marks)
Answer ALL Questions

21. Define Array with suitable example.
22. List out the applications of Stack.
23. Define Linked List.

Part – C(3X 8= 24 Marks)
Answer ALL Questions

24.(a)Explain arrays in c with example.

(OR)

(b) Discuss about Singly Linked List with Suitable example.

25.(a) Explain the stack in c with example

(OR)

(b) Discuss the representation of queue in c with example

26.(a) Explain the data structures in C Programming Language.

(OR)

(b) Write a C program to perform linked list(Any 2 Operations).

_____

Reg No. ……

[18CAU301]

KARPAGAM UNIVERSITY
Karpagam Academy Of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 64 021
BCA Degree Examination
(For the candidates admitted from 2018 onwards)
THIRD SEMESTER
First Internal Exam July 2019
DATA STRUCTURES

Time: 2 Hours **Maximum: 50 Marks**
Date&Session: **CLASS: II BCA(A&B)**

Part – A (20 X 1 = 20 Marks)

Answer ALL Questions

1.The tool useful for specifying the logical properties of datatype is_____
a)abstract data type b)datatype c)data structure d)algorithm

2.The group of items identified by its own identifier is known as_____
a)member of structure **b)structure** c)record d)field

3.An order collection of items is called_____
a)stack b)array c)list d)queue

4.The operation to perform add items in stack using_____
a)add element **b)pop** c)pop the element d)pushdown

5.The operation to perform deleting items in stack using_____
a)pushdownlist b)list the element c)delete item **d)push**

6.When the stack has zero elements then it can be called as_____
a)stack zero b)zero element c)no element d)stack empty

7.The operator "+"between the operand Aand B is called_____
a)infix b)precedence c)add the operand **d)combine operand**

8.An ordered collection of items from the front is called_____
a)queue b)double link c)priority queue d)descending queue

9._____is any data representation and its associated operations.
a)data structure **b)datatype** c)data declaration d)object.

10.Which one of the following is a LIFO list?
a)Stack b)Queue c)Linked list d)Circular list

11.The items may be inserted at one end of queue which is called
a)front **b)rear** c)top d)avail

12. The Complexity of linear search

a)O(n) b)O(log n) c)O(n log n) d)O(n^2)

13.In linear list each node has pointers to point to the predecessor and successors then node is called_____

a)singly linked list b)circular linked list c)doubly linked list **d)linear linked list**

14. A linear collection of data elements where the linear node is given by means of pointer is called

a) linked list b) node list c) primitive list d)array

15.The expression AB/C-DE*+AC*- is called

a)postfix b)prefix c)infix **d)expn**

16.A BST is traversed in the following order recursively: Right, root, left . The output sequence will be in

a) Ascending order b) Descending order c) Bitomic sequence d) No specific order

17.The expression AB-B+CE*+A-D*- is called

a)postfix b)prefix c)infix **d)expn**

18.Stacks and queues are

a)primitive data structure b)non-primitive data structure

c)non-linear data structure d)data types

19. Which one is not Divide and Conquer algorithm?

a)2-way merge sort b)quick sort c)selection sort d)insertion sort

20. Stack organization is

a)FIFO b)LILO **c)LIFO** d)FILO

Part – B(3X 2= 6 Marks)
Answer ALL Questions

21. Define Array with suitable example.

An **ARRAY** is a data structure that contains a group of elements. Typically these elements are all of the same data type, such as an integer or string. Arrays are commonly used in computer programs to organize data so that a related set of values can be easily sorted or searched.

22. List out the applications of Stack.

Applications of Stack

Expression Evaluation

Stack is used to evaluate prefix, postfix and infix expressions.

Expression Conversion

An expression can be represented in prefix, postfix or infix notation. Stack can be used to convert one form of expression to another.

Syntax Parsing
Many compilers use a stack for parsing the syntax of expressions, program blocks etc. before translating into low level code.

Backtracking
Suppose we are finding a path for solving maze problem. We choose a path and after following it we realize that it is wrong. Now we need to go back to the beginning of the path to start with new path. This can be done with the help of stack.

Parenthesis Checking
Stack is used to check the proper opening and closing of parenthesis.

String Reversal
Stack is used to reverse a string. We push the characters of string one by one into stack and then pop character from stack.

Function Call
Stack is used to keep information about the active functions or subroutines.

23. Define Linked List.

A **Linked list** is a linear collection of data elements, whose order is not given by their physical placement in memory. Instead, each element points to the next. ... In its most basic form, each node contains: data, and a reference (in other words, a **link**) to the next node in the sequence.

Part – C(3X 8= 24 Marks)
Answer ALL Questions

24.(a)Explain arrays in c with example.

An **Array** is a group (or collection) of same data types. For example an int array holds the elements of int
types while a float array holds the elements of float types.

How to declare Array in C

int num[35]; /* An integer array of 35 elements */
char ch[10]; /* An array of characters for 10 elements */
Similarly an array can be of any data type such as double, float, short etc.

How to access element of an array in C

int mydata[20]; mydata[0] /* first element of array mydata*/ mydata[19] /* last (20th) element of array mydata*/

Example of Array In C programming to find out the average of 4 integers

```
#include <stdio.h>
int main()
 {
int avg = 0;
int sum =0;
int x=0;
/* Array- declaration – length 4*/
```

```
int num[4];
/* We are using a for loop to traverse through the array
* while storing the entered values in the array
*/
for (x=0; x<4;x++) { printf("Enter number %d \n", (x+1)); scanf("%d", &num[x]); }
for (x=0; x<4;x++) { sum = sum+num[x]; } avg = sum/4; printf("Average of entered number is:
%d", avg);
return 0;
}
```

Output:

Enter number 1
10
Enter number 2
10
Enter number 3
20
Enter number 4
40
Average of entered number is: 20

(b) Discuss about Singly Linked List with Suitable example.

A **linked list** is a way to store a collection of elements. Like an array these can be character or integers. Each element in a linked list is stored in the form of a **node**.

Node:

A node is a collection of two sub-elements or parts. A **data** part that stores the element and a **next** part that stores the link to the next node.

Linked List:

A linked list is formed when many such nodes are linked together to form a chain. Each node points to the next node present in the order. The first node is always used as a reference to traverse the list and is called **HEAD**. The last node points to **NULL**.

Declaring a Linked list :

In C language, a linked list can be implemented using structure and pointers .

```
struct LinkedList{
   int data;
   struct LinkedList *next;
 };
```

The above definition is used to create every node in the list. The **data** field stores the element and the **next** is a pointer to store the address of the next node.

Noticed something unusual with next?

In place of a data type, **struct LinkedList** is written before next. That's because its a **self-referencing pointer**. It means a pointer that points to whatever it is a part of. Here **next** is a part of a node and it will point to the next node.

Creating a Node:

Let's define a data type of struct LinkedListto make code cleaner.

```
typedef struct LinkedList *node; //Define node as pointer of data type struct LinkedList

node createNode(){
   node temp; // declare a node
   temp = (node)malloc(sizeof(struct LinkedList)); // allocate memory using malloc()
   temp->next = NULL;// make next point to NULL
   return temp;//return the new node
}
```

typedef is used to define a data type in C.

malloc() is used to dynamically allocate a single block of memory in C, it is available in the header file stdlib.h.

sizeof() is used to determine size in bytes of an element in C. Here it is used to determine size of each node and sent as a parameter to malloc.

The above code will create a node with data as value and next pointing to NULL.

Let's see how to **add a node to the linked list**:

```
node addNode(node head, int value){
   node temp,p;// declare two nodes temp and p
   temp = createNode();//createNode will return a new node with data = value and next pointing
to NULL.
   temp->data = value; // add element's value to data part of node
   if(head == NULL){
      head = temp;     //when linked list is empty
   }
   else{
```

```
      p  = head;//assign head to p
      while(p->next != NULL){
         p = p->next;//traverse the list until p is the last node.The last node always points to
NULL.
      }
      p->next = temp;//Point the previous last node to the new node created.
   }
   return head;
}
```

Here the new node will always be added after the last node. This is known as **inserting a node at the rear end**.

This type of linked list is known as **simple or singly linked list**. A simple linked list can be traversed in only one direction from **head** to the last node.

The last node is checked by the condition :

```
p->next = NULL;
```

Here -> is used to access **next** sub element of node p. **NULL** denotes no node exists after the current node , i.e. its the end of the list.

Traversing the list:

The linked list can be traversed in a while loop by using the **head** node as a starting reference:

```
node p;
p = head;
while(p != NULL){
   p = p->next;
}
```

25.(a) Explain the stack in c with example

A **STACK** is a data structure which is used to store data in a particular order.
Two operations that can be
performed on a Stack are:
Push operation which inserts an element into the stack.
Pop operation which removes the last element that was added into the stack.
It follows **Last In First Out(LIFO) Order**.

```
#include <stdio.h>
#define MAXSIZE 5
struct stack
{
int stk[MAXSIZE];
```

```
int top;
};
typedef struct stack STACK;
STACK s;
void push(void);
int pop(void);
void display(void);
void main ()
{
int choice;
int option = 1;
s.top = -1;
printf ("STACK OPERATION\n");
while (option)
{
printf ("----------------------------------------\n");
printf (" 1 --> PUSH \n");
printf (" 2 --> POP \n");
printf (" 3 --> DISPLAY \n");
printf (" 4 --> EXIT \n");
printf ("----------------------------------------\n");
printf ("Enter your choice\n");
scanf ("%d", &choice);
switch (choice)
{
case 1:
push();
break;
case 2:
pop();
break;
case 3:
display();
break;
case 4:
return;
}
fflush (stdin);
printf ("Do you want to continue(Type 0 or 1)?\n");
scanf ("%d", &option);
}
}
/* Function to add an element to the stack */
void push ()
{
int num;
if (s.top == (MAXSIZE - 1))
{
printf ("Stack is Full\n");
return;
}
else
```

```
{
printf ("Enter the element to be pushed\n");
scanf ("%d", &num);
s.top = s.top + 1;
s.stk[s.top] = num;
}
return;
}
/* Function to delete an element from the stack */
int pop ()
{
int num;
if (s.top == - 1)
{
printf ("Stack is Empty\n");
return (s.top);
}
else
{
num = s.stk[s.top];
printf ("poped element is = %dn", s.stk[s.top]);
s.top = s.top - 1;
}
return(num);
}
/* Function to display the status of the stack */
void display ()
{
int i;
if (s.top == -1)
{
printf ("Stack is empty\n");
return;
}
else
{
printf ("\n The status of the stack is \n");
for (i = s.top; i >= 0; i--)
{
printf ("%d\n", s.stk[i]);
}
}
printf ("\n");
}
```

OUTPUT

STACK OPERATION

----

1 --> PUSH

2 --> POP

3 --> DISPLAY

4 --> EXIT

Enter your choice
1
Enter the element to be pushed
34
Do you want to continue(Type 0 or 1)?
0
$ a.out
STACK OPERATION
----
1 --> PUSH
2 --> POP
3 --> DISPLAY
4 --> EXIT

Enter your choice
1
Enter the element to be pushed
34
Do you want to continue(Type 0 or 1)?
1
----
1 --> PUSH
2 --> POP
3 --> DISPLAY
4 --> EXIT

Enter your choice
2
poped element is = 34

(b) Discuss the representation of queue in c with example

```
        #include <stdio.h>
#define MAX 50
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
int choice;
while (1)
{
printf("1.Insert element to queue \n");
printf("2.Delete element from queue \n");
printf("3.Display all elements of queue \n");
printf("4.Quit \n");
printf("Enter your choice : ");
scanf("%d", &choice);
switch (choice)
{
case 1:
insert();
```

```
break;
case 2:
delete();
break;
case 3:
display();
break;
case 4:
exit(1);
default:
printf("Wrong choice \n");
} /*End of switch*/
} /*End of while*/
} /*End of main()*/
insert()
{
int add_item;
if (rear == MAX - 1)
printf("Queue Overflow \n");
else
{
if (front == - 1)
/*If queue is initially empty */
front = 0;
printf("Inset the element in queue : ");
scanf("%d", &add_item);
rear = rear + 1;
queue_array[rear] = add_item;
}
} /*End of insert()*/
delete()
{
if (front == - 1 || front > rear)
{
printf("Queue Underflow \n");
return ;
}
else
{
printf("Element deleted from queue is : %d\n", queue_array[front]);
front = front + 1;
}
} /*End of delete() */
display()
{
int i;
if (front == - 1)
printf("Queue is empty \n");
else
{
printf("Queue is : \n");
for (i = front; i <= rear; i++)
```

```
printf("%d ", queue_array[i]);
printf("\n");
}
}
```

OUTPUT

1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 10
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 15

26.(a) Explain the data structures in C Programming Language.

A **Data Structure** is a specialized format for organizing and storing data. General data structure types include the array, the file, the record, the table, the tree, and so on. Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways.

In computer programming, a data structure may be selected or designed to store data for the purpose of working on it with various algorithms.

Data structures are used to store data in a computer in an organized form. In C language Different types of

DATA STRUCTURES ARE; ARRAY, STACK, QUEUE, LINKED LIST, TREE.

- Array: Array is collection of similar data type, you can insert and deleted element form array without follow any order.
- Stack: Stack work on the basis of Last-In-First-Out (LIFO). Last entered element removed first.
- Queue: Queue work on the basis of First-In-First-Out (FIFO). First entered element removed first.
- Linked List: Linked list is the collection of node, Here you can insert and delete data in any order.
- Tree: Stores data in a non linear form with one root node and sub nodes.

Algorithm

An algorithm is a finite set instruction, which is written for solve any problem. Algorithm is not the complete code or

program

Array in Data Structure

An Array is a collection of similar data type value in a single variable. An array is a derived data type in C, which is

constructed from fundamental data type of C language.

Insert element in array at specific position.

(b) Write a C program to perform linked list(Any 2 Operations).

C program to implement linked list

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *start = NULL;
void insert_at_begin(int);
void insert_at_end(int);
void traverse();
void delete_from_begin();
void delete_from_end();
int count = 0;
int main ()
{
int input, data;
for (;;)
{
printf("1. Insert an element at beginning of linked list.\n");
printf("2. Insert an element at end of linked list.\n");
printf("3. Traverse linked list.\n");
printf("4. Delete element from beginning.\n");
printf("5. Delete element from end.\n");
printf("6. Exit\n");
scanf("%d", &input);
if (input == 1)
{
printf("Enter value of element\n");
scanf("%d", &data);
insert_at_begin(data);
}
else if (input == 2)
{
printf("Enter value of element\n");
scanf("%d", &data);
insert_at_end(data);
}
else if (input == 3)
traverse();
else if (input == 4)
delete_from_begin();
else if (input == 5)
delete_from_end();
else if (input == 6)
break;
else
printf("Please enter valid input.\n");
}
return 0;
```

```
}
void insert_at_begin(int x)
 {
struct node *t;
t = (struct node*)malloc(sizeof(struct node));
count++;
if (start == NULL)
{
start = t;
start->data = x;
start->next = NULL;
return;
}
t->data = x;
t->next = start;
start = t;
}
void insert_at_end(int x)
{
struct node *t, *temp;
t = (struct node*)malloc(sizeof(struct node));
count++;
if (start == NULL) {
start = t;
start->data = x;
start->next = NULL;
return;
}
temp = start;
while (temp->next != NULL)
temp = temp->next;
temp->next = t;
t->data    = x;
t->next    = NULL;
}
void traverse()
{
struct node *t;
t = start;
if (t == NULL)
{
printf("Linked list is empty.\n");
return;
}
printf("There are %d elements in linked list.\n", count);
while (t->next != NULL) {
printf("%d\n", t->data);
t = t->next;
}
printf("%d\n", t->data);
}
void delete_from_begin()
```

```
{
struct node *t;
int n;
if (start == NULL)
{
printf("Linked list is already empty.\n");
return;
}
n = start->data;
t = start->next;
free(start);
start = t;
count--;
printf("%d deleted from beginning successfully.\n", n);
}
void delete_from_end()
{
struct node *t, *u;
int n;
if (start == NULL)
{
printf("Linked list is already empty.\n");
return;
}
count--;
if (start->next == NULL)
{
n = start->data;
free(start);
start = NULL;
printf("%d deleted from end successfully.\n", n);
return;
}
t = start;
while (t->next != NULL)
{
u = t;
t = t->next;
}
n = t->data;
u->next = NULL;
free(t);
printf("%d deleted from end successfully.\n", n);
}
```

_____

Reg No. ……

[18CAU301]

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
BCA Degree Examination
(For the candidates admitted from 2018 onwards)
Third Semester
Second Internal Exam
DATA STRUCTURES

Time: 2 Hours **Maximum: 50 Marks**
Date: 28.08.19 **Class: II BCA(A&B)**

Part – A (20 X 1 = 20 Marks)
Answer ALL Questions

1. Which item may be deleted at one end is called_____of the queue

a. Stack b. Front c. Rear d. List

2.Queue organization is _____

a. LIFO b. FIFO c. LILO d. FILO

3. _____priority queue is similar but allows deletion of only the largest item

a. Ascending b. Descending c. Similar d. Abstract

4. Each item in the list is called _____

a. node b. file c. field d. data

5. This_____is used to signal the end of the list

a. Empty List b. Null Pointer c. Node d. Linked List

6. List is a dynamic _____

a. Dynamic b. Stack c. Queue d. Data Structure

7. _____hold actual element on the list

a. Information b. Next Address c. Data d. Abstract

8.____field contains the address of the next node in the last

a. Information b. Next Address c. Data d. Abstract

9. Self address of the next item is called ____

a. Node b. Stack c. Queue d. Linked List

10. A_____is a finite set of element

a. Binary Tree b. Tree c. Linked List d. Queue

11. The first subset contains a single element called_____of the tree

a. Insertion b. Deletion c. Root d. Recursive

12. Two nodes are_____if they are left and right sons of the same father

a. Root b. Brothers c. Leaf d. Depth

13. The_____of the binary tree is the maximum level of any leaf in the tree

a. Root b. Brothers c. Leaf d. Depth

14. Binary tree of depth d is an almost_____binary tree

a. Threaded b. Complete c. Height Balance d. Traversals

15. To traverse a non-empty binary tree in _____

a. Preorder b. In order c. Post Order d .Depth Second Order

16. Both the linked array representation and the dynamic node representation are implementation of an abstract___

a. Array Representation b. Queue Representation c .Tree Representation d .Linked Representation

17. _____is used to link a node to its left or right sub-tree

a. Array b. Thread c. Stack d. Node

18. A_____is a finite non-empty set of commands

a. Binary Tree b. Tree c. Linked List d. Queue

19. A node with no sub-trees is a _____

a. Root b. Brothers c. Leaf d. Depth

20. An_____is define as a tree in which the sub-trees of each node from an ordered set

a. Ordered tree b. In ordered tree c. Deletion tree d. Insertion tree

Part-B[3*2=6 Marks]
Answer all the questions

21. Define Priority Queue.

22. Mention the two fields of Linked List.

23. List out the Terminologies of TREE.

Part-C[3*8=24 Marks]
Answer all the questions

24. a) Explain Linked List with an Example.

OR

b) Write about the Inserting and Removing node from a List.

25.a)Explain Queue with example.

OR

b) Write a C program for Implementing Queue Operations (any2).

26.a)Write a C program for Binary Search Tree(any2).

OR

b) Discuss the representation of tree using c with example.

Reg No. ……

[18CAU301]

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021
BCA Degree Examination
(For the candidates admitted from 2018onwards)
Third Semester
Second Internal Exam
DATA STRUCTURES

Time: 2 Hours **Maximum: 50 Marks**
Date: **Class: II BCA(A&B)**

Part – A (20 X 1 = 20 Marks)
Answer ALL Questions

1. Which item may be deleted at one end is called_____of the queue
a. stack b. front **c. rear** d. list
2.Queue organization is _____
a. LIFO **b. FIFO** c. LILO d. FILO
3. _____priority queue is similar but allows deletion of only the largest item
a. ascending b. descending c. similar d. abstract
4. Each item in the list is called _____
a. node b. file c. field d. data
5. This_____is used to signal the end of the list
a. empty list **b. null pointer** c. node d. linked list
6. List is a dynamic _____
a. dynamic b. stack **c. queue** d. data structure
7. _____hold actual element on the list
a. information b. next address c. data d. abstract
8. ____field contains the address of the next node in the last
a. information **b. next address** c. data d. abstract
9. Self address of the next item is called ____
a. node b. stack **c. queue** d. linked list
10. A_____is a finite set of element
a. binary tree b. tree c. linked list d. queue
11. The first subset contains a single element called_____of the tree
a. insertion b. deletion **c. root** d. recursive
12. Two nodes are_____if they are left and right sons of the same father
a. root **b. brothers** c. leaf d. depth
13. The_____of the binary tree is the maximum level of any leaf in the tree
a. root b. brothers c. leaf **d. depth**
14. Binary tree of depth d is an almost_____binary tree
a. threaded b. complete **c. height balance** d. traversals
15. To traverse a non-empty binary tree in _____
a. preorder b. inorder **c.post order** d .depth second order
16. Both the linked array representation and the dynamic node representation are implementation of an abstract _____
a. array representation b. queue representation c .tree representation d .linked representation
17. _____is used to link a node to its left or right sub-tree
a. array b. thread c. stack **d. node**
18. A_____is a finite non-empty set of commands
a. binary tree b. tree c.linked list **d. queue**
19. A node with no sub-trees is a _____
a. root b. brothers **c. leaf** d. depth

20. An_____is define as a tree in which the sub-trees of each node from an ordered set
a. ordered tree b. inordered tree c. deletion tree d. insertion tree

Part-B [3*2=6 Marks]
Answer all the questions

21. Define Priority Queue.

- A **priority queue** is an abstract data type which is like a regular **queue** or stack data structure, but where additionally each element has a "**priority**" associated with it.
- In a **priority queue**, an element with high **priority** is served before an element with low **priority**.

22.Mention the two fields of LinkedList.

- A doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes.
- Each node contains two fields, called links, that are references to the previous and to the next node in the sequence of nodes.

23.List out the Applications of TREE.

a.Manipulate hierarchical data.

b.Make information easy to search (see tree traversal).
c.Manipulate sorted lists of data.
d.As a workflow for compositing digital images for visual effects.
e.Router algorithms

Part-B [3*2=6 Marks]
Answer all the questions

Explain Linked List with an Example.

A linked list is a dynamic data structure.

The number of nodes in a list is not fixed and can grow and shrink on demand.

Any application which has to deal with an unknown number of objects will need to use a linked list.

Types of Linked Lists

A **singly linked list** is described above

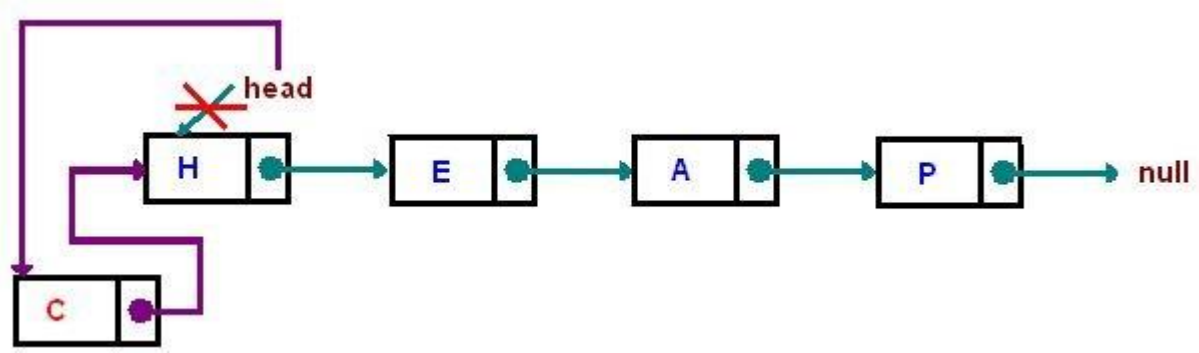
A **doubly linked list** is a list that has two references, one to the next node and another to previous node.

Another important type of a linked list is called a **circular linked list** where last node of the list points back to the first node (or the head) of the list.

Linked List Operations

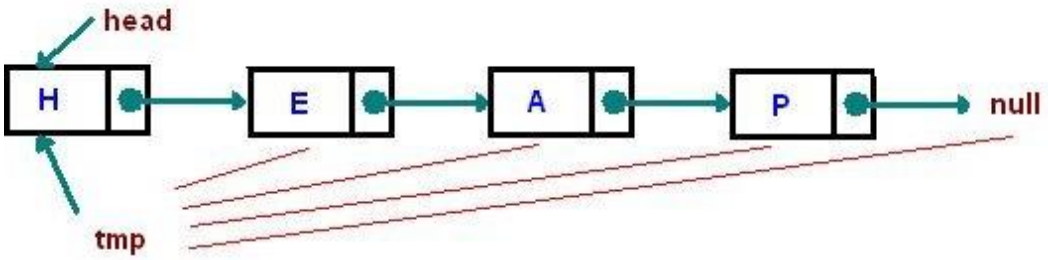
addFirst

The method creates a node and prepends it at the beginning of the list.



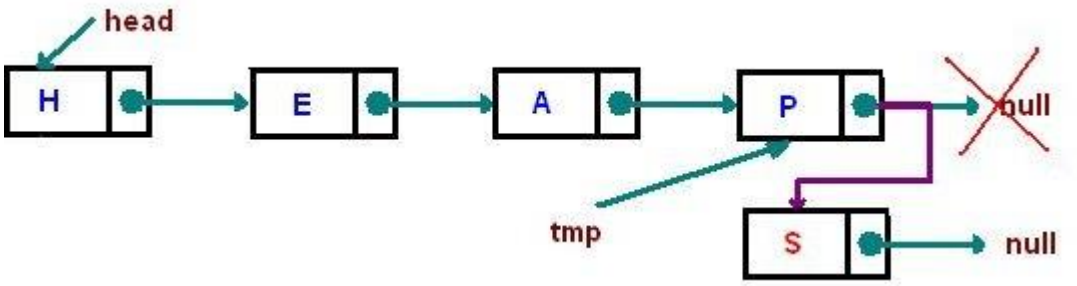
Traversing

Start with the head and access each node until you reach null. Do not change the head reference.



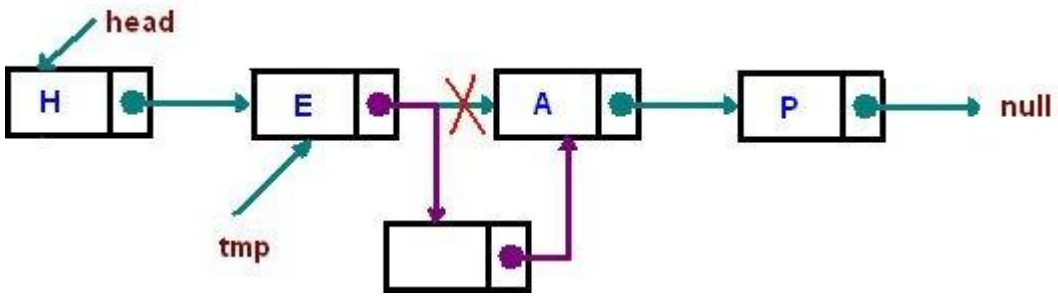
addLast

The method appends the node to the end of the list. This requires traversing, but make sure you stop at the last node



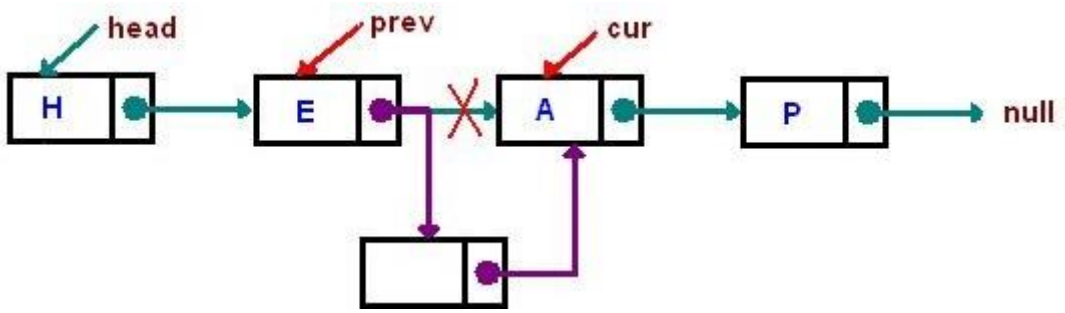
Inserting "after"

Find a node containing "key" and insert a new node after it. In the picture below, we insert a new node after "e":



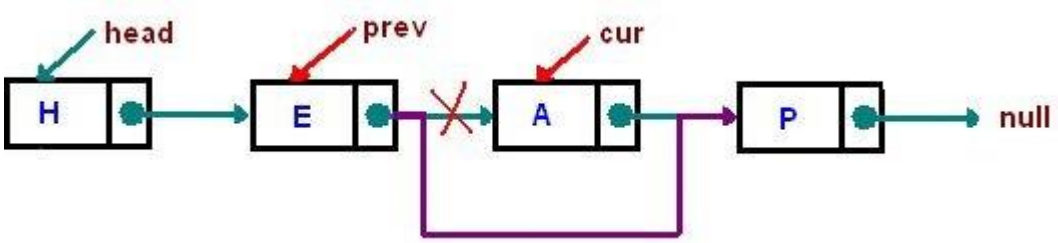
Inserting "before"

Find a node containing "key" and insert a new node before that node. In the picture below, we insert a new node before "a":



Deletion

Find a node containing "key" and delete it. In the picture below we delete a node containing "A"



The algorithm is similar to insert "before" algorithm. It is convinient to use two references prev and cur. When we move along the list we shift these two references, keeping prev one step before cur. We continue until cur reaches the node which we need to delete. There are three exceptional cases, we need to take care of:

1. list is empty
2. delete the head node
3. node is not in the list

Write about the iserting and removing node from a list.

Insertion In Linked list

There are three situation for inserting element in list.

1. Insertion at the front of list.
2. Insertion in the middle of the list.
3. Insertion at the end of the list.

Procedure For Inserting an element to linked list

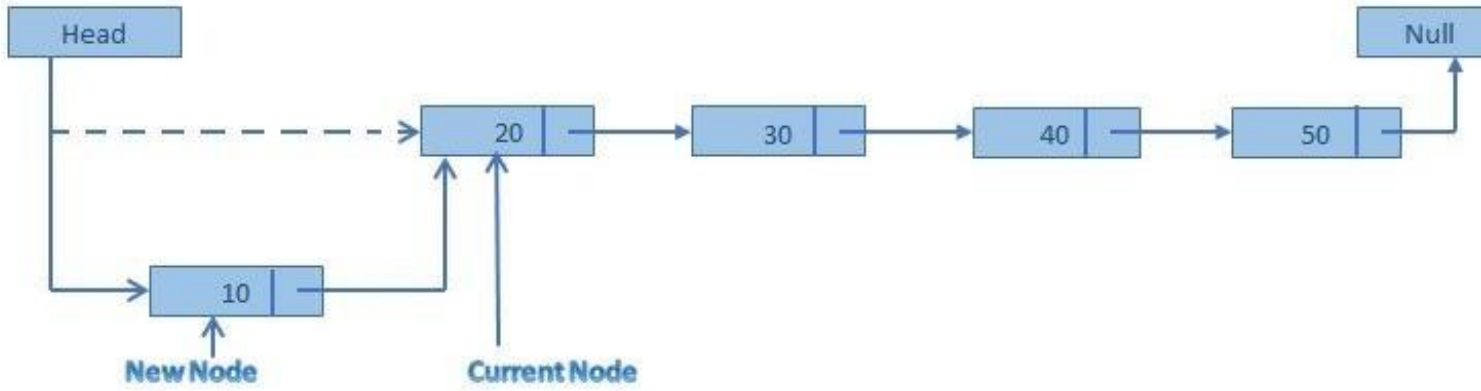
Step-1: Get the value for NEW node to be added to the list and its position.
Step-2: Create a NEW, empty node by calling malloc(). If malloc() returns no error then go to step-3 or else say "Memory shortage".
Step-3: insert the data value inside the NEW node's data field.
Step-4: Add this NEW node at the desired position (pointed by the "location") in the LIST.
Step-5: Go to step-1 till you have more values to be added to the LIST.

Insertion Node In Linked List

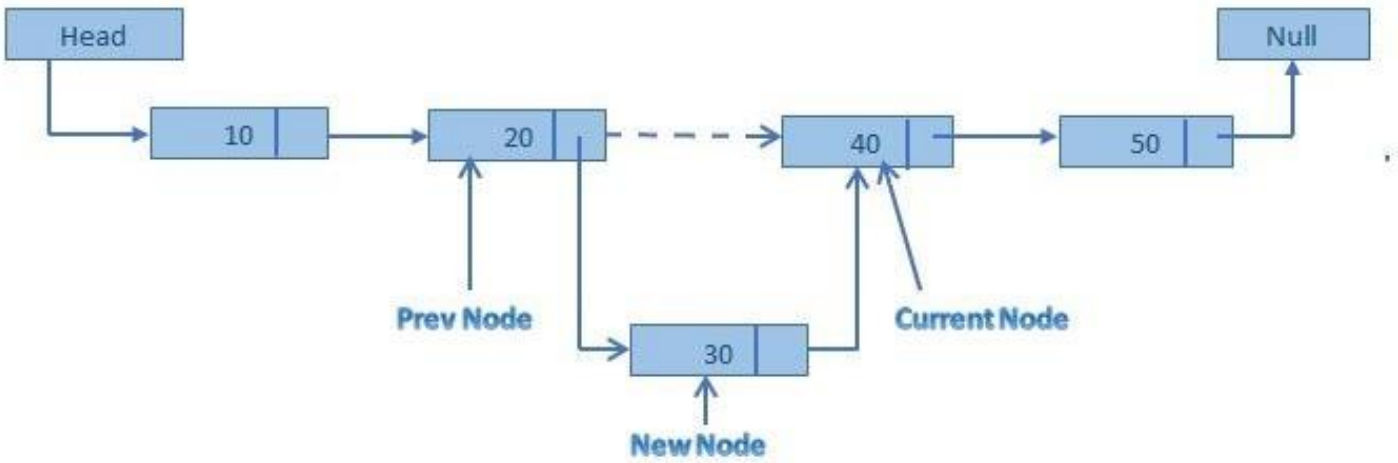
```
void insert(node *ptr, int data)
{
    /* Iterate through the list till we encounter the last node.*/
```

```
        while(ptr->next!=NULL)
        {
                ptr = ptr -> next;
        }

        /* Allocate memory for the new node and put data in it.*/

        ptr->next = (node *)malloc(sizeof(node));
        ptr = ptr->next;
        ptr->data = data;
        ptr->next = NULL;
}
```

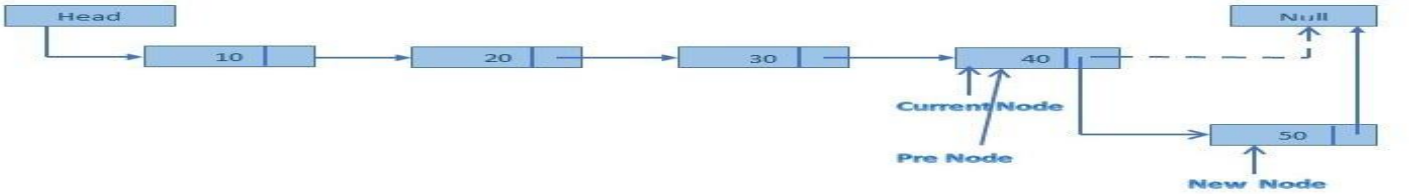
Insertion at the front of list



Insertion Node in given location Linked List



Insertion at the end of the list.

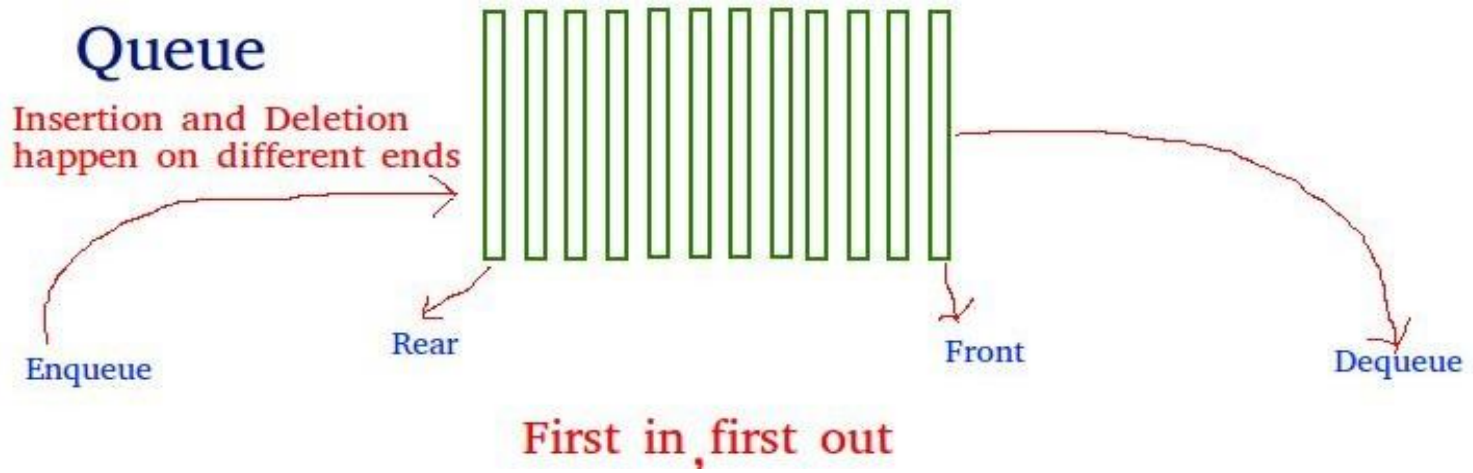


Explain Queue with example.

- The order is **F**irst **I**n **F**irst **O**ut (FIFO).
- A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.
  The difference between stacks and queues is in removing.
- In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

Operations on Queue:
Mainly the following four basic operations are performed on queue:

Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.
Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.
Front: Get the front item from queue.
Rear: Get the last item from queue.



Applications of Queue:
Queue is used when things don't have to be processed immediatly, but have to be processed in **F**irst **I**n**F**irst **O**ut order like Breadth First Search. This property of Queue makes it also useful in following kind of scenarios.

1) When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
2) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

Array implementation Of Queue
For implementing queue, we need to keep track of two indices, front and rear. We enqueue an item at the rear and

dequeue an item from front. If we simply increment front and rear indices, then there may be problems, front may reach end of the array. The solution to this problem is to increase front and rear in circular manner.

```
// C program for array implementation of queue

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

 // A structure to represent a queue

struct Queue

{

   int front, rear, size;

   unsigned  capacity;

   int* array;

};


// function to create a queue of given capacity.

// It initializes size of queue as 0

struct Queue* createQueue(unsigned capacity)

{

   struct Queue* queue = (struct Queue*) malloc(sizeof(struct Queue));

   queue->capacity = capacity;

   queue->front = queue->size = 0;

   queue->rear = capacity - 1; // This is important, see the enqueue

   queue->array = (int*) malloc(queue->capacity * sizeof(int));

   return queue;

}

 // Queue is full when size becomes equal to the capacity

int isFull(struct Queue* queue)

{ return (queue->size == queue->capacity); }

 // Queue is empty when size is 0

int isEmpty(struct Queue* queue)
```

```
{ return (queue->size == 0); }
 // Function to add an item to the queue.
// It changes rear and size
void enqueue(struct Queue* queue, int item)
{
    if (isFull(queue))
        return;
    queue->rear = (queue->rear + 1)%queue->capacity;
    queue->array[queue->rear] = item;
    queue->size = queue->size + 1;
    printf("%d enqueued to queue\n", item);
}
 // Function to remove an item fromqueue.
// It changes front and size
int dequeue(struct Queue* queue)
{
    if (isEmpty(queue))
        return INT_MIN;
    int item = queue->array[queue->front];
    queue->front = (queue->front + 1)%queue->capacity;
    queue->size = queue->size - 1;
    return item;
}
 // Function to get front of queue
int front(struct Queue* queue)
{
    if (isEmpty(queue))
        return INT_MIN;
    return queue->array[queue->front];
```

```
}
 // Function to get rear of queue
int rear(struct Queue* queue)
{
   if (isEmpty(queue))
      return INT_MIN;
   return queue->array[queue->rear];
}
 // Driver program to test above functions./
int main()
{
   struct Queue* queue = createQueue(1000);
    enqueue(queue, 10);
   enqueue(queue, 20);
   enqueue(queue, 30);
   enqueue(queue, 40);
    printf("%d dequeued from queue\n", dequeue(queue));
    printf("Front item is %d\n", front(queue));
   printf("Rear item is %d\n", rear(queue));
    return 0;
}
```

Output:

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
10 dequeued from queue
Front item is 20
Rear item is 40
```

24.b) Discuss the representation of tree using c with example.

A tree data structure can be defined recursively (locally) as a collection of nodes(starting at a root node), where each node is a data structure consisting of a value, together with a list of references to nodes (the "children"), with the constraints that no reference is duplicated, and none points to the root.

Alternatively, a tree can be defined abstractly as a whole (globally) as an ordered tree, with a value assigned to each node. Both these perspectives are useful: while a tree can be analyzed mathematically as a whole, when actually represented as a data structure it is usually represented and worked with separately by node (rather than as a set of nodes and an adjacency list of edges between nodes

Root

The top node in a tree.

Child

A node directly connected to another node when moving away from the Root.

Parent

The converse notion of a *child*.

Siblings

A group of nodes with the same parent.

Descendant

A node reachable by repeated proceeding from parent to child.

Ancestor

A node reachable by repeated proceeding from child to parent.

Leaf

(less commonly called External node)

A node with no children.

Branch

Internal node

A node with at least one child.

Degree

The number of subtrees of a node.

Edge

The connection between one node and another.

Path

A sequence of nodes and edges connecting a node with a descendant.

Level

The level of a node is defined by 1 + (the number of connections between the node and the root).

Height of node

The height of a node is the number of edges on the longest path between that node and a leaf.

Height of tree

The height of a tree is the height of its root node.

Depth

The depth of a node is the number of edges from the tree's root node to the node.

Write a program to perform binary tree using c.

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct btnode
{
    int value;
    struct btnode *l;
    struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;

void delete1();
void insert();
void delete();
void inorder(struct btnode *t);
void create();
void search(struct btnode *t);
void preorder(struct btnode *t);
void postorder(struct btnode *t);
void search1(struct btnode *t,int data);
int smallest(struct btnode *t);
int largest(struct btnode *t);

int flag = 1;

void main()
{
    int ch;

    printf("\nOPERATIONS ---");
    printf("\n1 - Insert an element into tree\n");
    printf("2 - Delete an element from the tree\n");
    printf("3 - Inorder Traversal\n");
    printf("4 - Preorder Traversal\n");
    printf("5 - Postorder Traversal\n");
    printf("6 - Exit\n");
    while(1)
    {
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            insert();
            break;
        case 2:
            delete();
            break;
        case 3:
            inorder(root);
            break;
        case 4:
            preorder(root);
            break;
        case 5:
            postorder(root);
            break;
```

```
        case 6:
            exit(0);
        default :
            printf("Wrong choice, Please enter correct choice ");
            break;
        }
    }
}

/* To insert a node in the tree */
void insert()
{
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}

/* To create a node */
void create()
{
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}

/* Function to search the appropriate position to insert the new node */
void search(struct btnode *t)
{
    if ((temp->value > t->value) && (t->r  != NULL))       /* value more than root node value insert at right
*/
        search(t->r);
    else if ((temp->value > t->value) && (t->r == NULL))
        t->r = temp;
     else if ((temp->value < t->value)  && (t->l != NULL))     /* value less than root node value insert at left
*/
        search(t->l);
    else if ((temp->value < t->value) && (t->l == NULL))
        t->l = temp;
}

/* recursive function to perform inorder traversal of tree */
void inorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
```

```
    }
    if (t->l != NULL)
        inorder(t->l);
    printf("%d -> ", t->value);
    if (t->r != NULL)
        inorder(t->r);
}

/* To check for the deleted node */
void delete()
{
    int data;

    if (root == NULL)
    {
        printf("No elements in a tree to delete");
        return;
    }
    printf("Enter the data to be deleted : ");
    scanf("%d", &data);
    t1 = root;
    t2 = root;
    search1(root, data);
}

/* To find the preorder traversal */
void preorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    printf("%d -> ", t->value);
    if (t->l != NULL)
        preorder(t->l);
    if (t->r != NULL)
        preorder(t->r);
}

/* To find the postorder traversal */
void postorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display ");
        return;
    }
    if (t->l != NULL)
        postorder(t->l);
    if (t->r != NULL)
        postorder(t->r);
    printf("%d -> ", t->value);
```

```
}

/* Search for the appropriate position to insert the new node */
void search1(struct btnode *t, int data)
{
    if ((data>t->value))
    {
        t1 = t;
        search1(t->r, data);
    }
    else if ((data < t->value))
    {
        t1 = t;
        search1(t->l, data);
    }
    else if ((data==t->value))
    {
        delete1(t);
    }
}

/* To delete a node */
void delete1(struct btnode *t)
{
    int k;

    /* To delete leaf node */
    if ((t->l == NULL) && (t->r == NULL))
    {
        if (t1->l == t)
        {
            t1->l = NULL;
        }
        else
        {
            t1->r = NULL;
        }
        t = NULL;
        free(t);
        return;
    }

    /* To delete node having one left hand child */
    else if ((t->r == NULL))
    {
        if (t1 == t)
        {
            root = t->l;
            t1 = root;
        }
        else if (t1->l == t)
        {
            t1->l = t->l;
```

```
        }
        else
        {
            t1->r = t->l;
        }
        t = NULL;
        free(t);
        return;
    }

    /* To delete node having right hand child */
    else if (t->l == NULL)
    {
        if (t1 == t)
        {
            root = t->r;
            t1 = root;
        }
        else if (t1->r == t)
            t1->r = t->r;
        else
            t1->l = t->r;
        t == NULL;
        free(t);
        return;
    }

    /* To delete node having two child */
    else if ((t->l != NULL) && (t->r != NULL))
    {
        t2 = root;
        if (t->r != NULL)
        {
            k = smallest(t->r);
            flag = 1;
        }
        else
        {
            k =largest(t->l);
            flag = 2;
        }
        search1(root, k);
        t->value = k;
    }

}

/* To find the smallest element in the right sub tree */
int smallest(struct btnode *t)
{
    t2 = t;
    if (t->l != NULL)
```

```
    {
        t2 = t;
        return(smallest(t->l));
    }
    else
        return (t->value);
}

/* To find the largest element in the left sub tree */
int largest(struct btnode *t)
{
    if (t->r != NULL)
    {
        t2 = t;
        return(largest(t->r));
    }
    else
        return(t->value);
}
```

```
$ cc tree43.c
$ a.out
OPERATIONS ---
1 - Insert an element into tree
2 - Delete an element from the tree
3 - Inorder Traversal
4 - Preorder Traversal
5 - Postorder Traversal
6 - Exit
 Enter your choice : 1
Enter data of node to be inserted : 40
 Enter your choice : 1
Enter data of node to be inserted : 20
 Enter your choice : 1
Enter data of node to be inserted : 10
Enter your choice : 1
Enter data of node to be inserted : 30
Enter your choice : 1
Enter data of node to be inserted : 60

Enter your choice : 1
Enter data of node to be inserted : 80
 Enter your choice : 1
Enter data of node to be inserted : 90
 Enter your choice : 3
10 -> 20 -> 30 -> 40 -> 60 -> 80 -> 90 ->
            40
           /\
          / \
         20    60
        / \    \
       10 30 80
                \
```

Write a program for binary search tree using c.

```
#include <stdio.h>
#include <stdlib.h>

struct btnode
{
    int value;
    struct btnode *l;
    struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;

void delete1();
void insert();
void delete();
void inorder(struct btnode *t);
void create();
void search(struct btnode *t);
void preorder(struct btnode *t);
void postorder(struct btnode *t);
void search1(struct btnode *t,int data);
int smallest(struct btnode *t);
int largest(struct btnode *t);

int flag = 1;

void main()
{
    int ch;

    printf("\nOPERATIONS ---");
    printf("\n1 - Insert an element into tree\n");
    printf("2 - Delete an element from the tree\n");
    printf("3 - Inorder Traversal\n");
    printf("4 - Preorder Traversal\n");
    printf("5 - Postorder Traversal\n");
    printf("6 - Exit\n");
    while(1)
    {
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            insert();
            break;
        case 2:
            delete();
            break;
        case 3:
            inorder(root);
            break;
```

```
        case 4:
            preorder(root);
            break;
        case 5:
            postorder(root);
            break;
        case 6:
            exit(0);
        default :
            printf("Wrong choice, Please enter correct choice ");
            break;
        }
    }
}

/* To insert a node in the tree */
void insert()
{
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}

/* To create a node */
void create()
{
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}

/* Function to search the appropriate position to insert the new node */
void search(struct btnode *t)
{
    if ((temp->value > t->value) && (t->r  != NULL))        /* value more than root node value insert at  right
*/
        search(t->r);
    else if ((temp->value > t->value) && (t->r == NULL))
        t->r = temp;
     else if ((temp->value < t->value)  && (t->l != NULL))      /* value less than root node value insert at  left
*/
        search(t->l);
    else if ((temp->value < t->value) && (t->l == NULL))
        t->l = temp;
}

/* recursive function to perform inorder traversal of tree */
```

```
void inorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    if (t->l != NULL)
        inorder(t->l);
    printf("%d -> ", t->value);
    if (t->r != NULL)
        inorder(t->r);
}

/* To check for the deleted node */
void delete()
{
    int data;

    if (root == NULL)
    {
        printf("No elements in a tree to delete");
        return;
    }
    printf("Enter the data to be deleted : ");
    scanf("%d", &data);
    t1 = root;
    t2 = root;
    search1(root, data);
}

/* To find the preorder traversal */
void preorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    printf("%d -> ", t->value);
    if (t->l != NULL)
        preorder(t->l);
    if (t->r != NULL)
        preorder(t->r);
}

/* To find the postorder traversal */
void postorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display ");
        return;
```

```
    }
    if (t->l != NULL)
        postorder(t->l);
    if (t->r != NULL)
        postorder(t->r);
    printf("%d -> ", t->value);
}

/* Search for the appropriate position to insert the new node */
void search1(struct btnode *t, int data)
{
    if ((data>t->value))
    {
        t1 = t;
        search1(t->r, data);
    }
    else if ((data < t->value))
    {
        t1 = t;
        search1(t->l, data);
    }
    else if ((data==t->value))
    {
        delete1(t);
    }
}

/* To delete a node */
void delete1(struct btnode *t)
{
    int k;

    /* To delete leaf node */
    if ((t->l == NULL) && (t->r == NULL))
    {
        if (t1->l == t)
        {
            t1->l = NULL;
        }
        else
        {
            t1->r = NULL;
        }
        t = NULL;
        free(t);
        return;
    }

    /* To delete node having one left hand child */
    else if ((t->r == NULL))
    {
        if (t1 == t)
        {
```

```
            root = t->l;
            t1 = root;
        }
        else if (t1->l == t)
        {
            t1->l = t->l;

        }
        else
        {
            t1->r = t->l;
        }
        t = NULL;
        free(t);
        return;
    }

    /* To delete node having right hand child */
    else if (t->l == NULL)
    {
        if (t1 == t)
        {
            root = t->r;
            t1 = root;
        }
        else if (t1->r == t)
            t1->r = t->r;
        else
            t1->l = t->r;
        t == NULL;
        free(t);
        return;
    }

    /* To delete node having two child */
    else if ((t->l != NULL) && (t->r != NULL))
    {
        t2 = root;
        if (t->r != NULL)
        {
            k = smallest(t->r);
            flag = 1;
        }
        else
        {
            k =largest(t->l);
            flag = 2;
        }
        search1(root, k);
        t->value = k;
    }

}
```

```
/* To find the smallest element in the right sub tree */
int smallest(struct btnode *t)
{
    t2 = t;
    if (t->l != NULL)
    {
        t2 = t;
        return(smallest(t->l));
    }
    else
        return (t->value);
}

/* To find the largest element in the left sub tree */
int largest(struct btnode *t)
{
    if (t->r != NULL)
    {
        t2 = t;
        return(largest(t->r));
    }
    else
        return(t->value);
}
```

OUTPUT:

OPERATIONS ---

1 - Insert an element into **tree**

2 - Delete an element from the **tree**

3 - Inorder Traversal

4 - Preorder Traversal

5 - Postorder Traversal

6 - Exit

Enter your choice : 1

Enter data of node to be inserted : 40

Enter your choice : 1

Enter data of node to be inserted : 20

Enter your choice : 1

Enter data of node to be inserted : 10

Enter your choice : 1

Enter data of node to be inserted : 30

Enter your choice : 1

Enter data of node to be inserted : 60

Enter your choice : 1

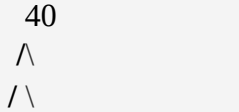
Enter data of node to be inserted : 80

Enter your choice : 1

Enter data of node to be inserted : 90

Enter your choice : 3

10 -> 20 -> 30 -> 40 -> 60 -> 80 -> 90 ->



```
  20    60
 / \    \
10 30 80
        \
        90
```

Reg.No ----------

[18CAU301]

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore – 641 021

BCA DEGREE EXAMINATION

(For the candidates admitted in 2018 onwards)

THIRD SEMESTER

III- INTERNAL TEST

DATA STRUCTURES

Duration:2 hours **Maximum Marks:50**

Date: **Class: II BCA(A&B)**

PART-A [20 * 1 = 20 Marks]

Answer ALL the questions

1.In a Hash table the address of the identifier x is obtained by applying

a) sequence of comparisons b)binary searching c)arithmetic function d)collision

2.The partitions of the hash table are called _____

a)Nodes b)Buckets c)Roots d)Fields

3.The arithmetic functions used for Hashing is called _____

a)Logical operations b)Rehashing c)Mapping function d)Hashing function

4. Each bucket of Hash table is said to have several _____

a) sequence of comparisons b)binary searching c)arithmetic function d)collision

5. If the names are_____in the symbol table, searching is easy.

a)sorted b)short c)bold d)upper case

6._____allocation is not desirable for dynamic tables, where insertions and deletions are allowed.

a)Linear b)Sequential c)Dynamic d)None

7.A search in a hash table with n identifiers may take------ time

a)O(n) b)O(1) c)O(2) d)O(2n)

8._____data structure is used to implement symbol tables

a) directed graphs b)binary search trees c)circular queue d)None

9.A_____occurs when two non_identical identifiers are hashed in the same bucket.

a)collision b)contraction c)expansion d)Extraction

10.A hashing function f transforms an identifier x into a_____in the hash table

a)symbol name b)bucket address c)link field d)slot number

11. When a new identifier I is mapped or hashed by the function f into a full bucket then___occurs

a)underflow b)overflow c)collision d)rehashing

12. In External sorting data are storedin _____

a) RAM memory b)Cache memory c)secondary storage devices d)Buffers

13. _____ techniques are used for sorting large files

14. a)Topological sort b)External sorting c)Linear Sorting d)Heap sort

15. 14.In_____a k-way merging uses only k+1 tapes

a)Internal sorting b)Polyphase merging c)Linking d)Hashing

15. Before merging the next phase is necessary to_____the output tapes

a)replace b)rewind c)remove d)None

16. To reduce the rewind time it is overlapped with read/write on other tapes. This modification need _____

a)double the number of tapes (2k tapes) b)only two tapes

c)one additional tape (k+2 tapes) d)k+1 tapes

17. In hash table, if the identifier x has an equal chance of hashing into any of the buckets,

this function is called as _____

a)Equal hash function b)uniform hash function c)Linear hashing function d)unequal Hashing function

18. Each head node is smaller than the other nodes because it has to retain

a)only a link b) only a link and a record c)only two link d)only the record

19.Each chain in the hash tables will have a

a)tail node b)link node c)head node d)null node

20.Folding of identifiers from end to end to obtain a hashing function is called _____

a)Shift folding b)boundary folding c)expanded folding d)end to end folding

PART-B [3*2=6Marks]
Answer ALL the questions

21.What is meant by Preorder & Post Order search?

22.Define selection Sort.

23.What is meant by Hashing?

PART-C [3* 8= 24Marks]
Answer ALL the questions

24.A) Write an algorithm for Queue operations.

OR

B) List the common operations on binary and binary search tree. Write an algorithm with example.

25.A) Explain Insertion sort with example.

OR

B) Explain quick sort with example.

26.A) Write an algorithm for heap sort with example.

OR

B) Write an algorithm for merge sort with example.

Reg.No ----------

[18CAU301]

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore – 641 021

BCA DEGREE EXAMINATION

(For the candidates admitted in 2018 onwards)

Third Semester

III- INTERNAL TEST

DATA STRUCTURES

Duration:2 hours **Maximum Marks:50**

Date: **Class: II BCA(A&B)**

PART-A [20 * 1 = 20 Marks]

Answer ALL the questions

1.In a Hash table the address of the identifier x is obtained by applying

a) sequence of comparisons **b)binary searching** c)arithmetic function d)collision

2.The partitions of the hash table are called _____

a)Nodes b)Buckets c)Roots **d)Fields**

3.The arithmetic functions used for Hashing is called _____

a)Logical operations b)Rehashing c)Mapping function **d)Hashing function**

4. Each bucket of Hash table is said to have several _____

a) sequence of comparisons b)binary searching c)arithmetic function d)collision

5. If the names are_____in the symbol table, searching is easy.

a)sorted b)short c)bold d)upper case

6._____allocation is not desirable for dynamic tables, where insertions and deletions are allowed.

a)Linear **b)Sequential** c)Dynamic d)None

7.A search in a hash table with n identifiers may take ------ time

a)O(n) b)O(1) c)O(2) d)O(2n)

8._____data structure is used to implement symbol tables

a) directed graphs b)binary search trees c)circular queue d)None

9.A_____occurs when two non_identical identifiers are hashed in the same bucket.

a)collision b)contraction c)expansion d)Extraction

10.A hashing function f transforms an identifier x into a_____in the hash table

a)symbol name **b)bucket address** c)link field d)slot number

11. When a new identifier I is mapped or hashed by the function f into a full bucket then___occurs

a)underflow b)overflow c)collision **d)rehashing**

12. In External sorting data are stored in _____

a) RAM memory b)Cache memory c)secondary storage devices **d)Buffers**

13. _____ techniques are used for sorting large files

14. a)Topological sort b)External sorting c)Linear Sorting **d)Heap sort**

15. 14.In_____a k-way merging uses only k+1 tapes

a)Internal sorting b)Polyphase merging c)Linking d)Hashing

15. Before merging the next phase is necessary to_____the output tapes

a)replace **b)rewind** c)remove d)None

16. To reduce the rewind time it is overlapped with read/write on other tapes. This modification need _____

a)double the number of tapes (2k tapes) b)only two tapes

c)one additional tape (k+2 tapes) d)k+1 tapes

17. In hash table, if the identifier x has an equal chance of hashing into any of the buckets,

this function is called as _____

a)Equal hash function b)uniform hash function **c)Linear hashing function** d)unequal Hashing function

18. Each head node is smaller than the other nodes because it has to retain

a)only a link b) only a link and a record c)only two link d)only the record

19.Each chain in the hash tables will have a

a)tail node b)link node **c)head node** d)null node

20. Folding of identifiers from end to end to obtain a hashing function is called _____

21. a)Shift folding **b)boundary folding** c)expanded folding d)end to end folding

PART-B [3*2=6Marks]

Answer ALL the questions

22. What is meant by Preorder & PostOrder search?

Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on of an expression tree.

Postorder traversal is used to delete the tree. Please see the question for deletion of tree for details. Postorder traversal is also useful to get the postfix expression of an expression tree.

23. Define selection Sort.

The Selection sort algorithm is based on the idea of finding the minimum or maximum element in an unsorted array and then putting it in its correct position in a sorted array.

24. What is meant by Root & Node of aTree?

A data structure accessed beginning at the **root node**. Each **node** is either a leaf or an internal **node**. An internal **node** has one or more child **nodes** and is called the parent of its child **nodes**.

PART-C [3* 8= 24Marks]

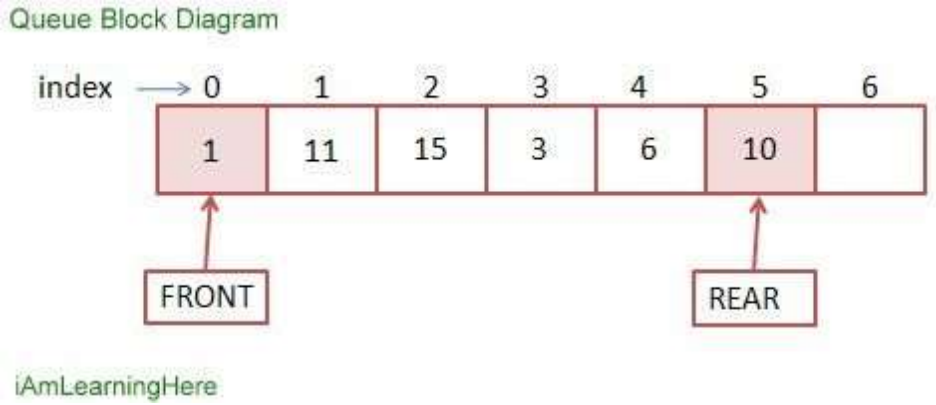
Answer ALL the questions

rite an algorithm for Queue operations.

Queue is ordered collection of homogeneous data elements in which insertion and deletion operation take place at two end . insertion allowed from starting of queue called **FRONT** point and deletion allowed from **REAR** end only

- insertion operation is called **ENQUEUE**
- deletion operation is called **DEQUEUE**

Block Diagram of Queue



Conditions in Queue

- FRONT < 0 (Queue is Empty)
- REAR = Size of Queue (Queue is Full)
- FRONT < REAR (Queue contains at least one element)
- No of elements in queue is : (REAR - FRONT) + 1
- **Algorithm for ENQUEUE (insert element in Queue)**
- ***Input*** : An element say ITEM that has to be inserted.
  Output : ITEM is at the REAR of the Queue.
  Data structure : Que is an array representation of queue structure with two pointer FRONT and REAR.

Steps:

```
1. If ( REAR = size ) then //Queue is full
2.       print "Queue is full"
3.       Exit
4. Else
5.       If ( FRONT = 0 ) and ( REAR = 0 ) then //Queue is empty
6.             FRONT = 1
```

```
7.      End if
8.      REAR = REAR + 1 // increment REAR
9.      Que[ REAR ] = ITEM
10. End if
11. Stop
```

- **Algorithm for DEQUEUE (delete element from Queue)**
- ***Input*** : A que with elements. FRONT and REAR are two pointer of queue .
  Output : The deleted element is stored in ITEM.
  Data structure : Que is an array representation of queue structure..

Steps:

```
1.  If ( FRONT = 0 ) then
2.      print "Queue is empty"
3.      Exit
4.  Else
5.      ITEM = Que [ FRONT ]
6.      If  ( FRONT  =  REAR )
7.          REAR = 0
8.          FRONT = 0
9.      Else
10.         FRONT = FRONT + 1
11.     End if
12. End if
13. Stop
```

b. List the common operations on binary and binary search tree. Write an algorithm with example.

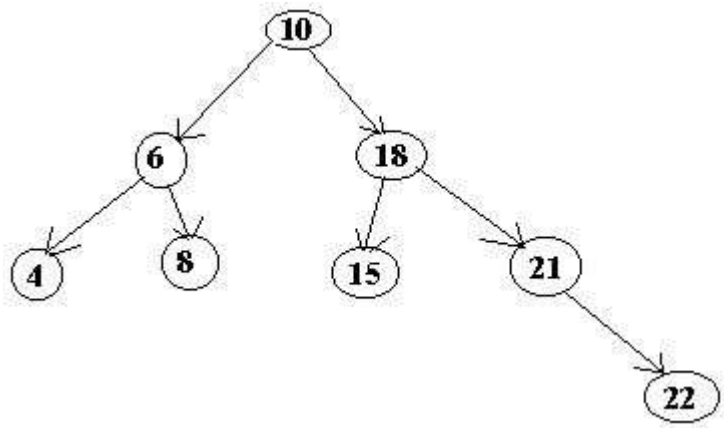
Operations on Binary Search Tree's

In the previous lesson, we considered a particular kind of a binary tree called a Binary Search Tree (BST). A binary tree is a binary search tree (BST) if and only if an inorder traversal of the binary tree results in a sorted sequence. The idea of a binary search tree is that data is stored according to an order, so that it can be retrieved very efficiently.

A BST is a binary tree of nodes ordered in the following way:

1. Each node contains one key (also unique)
2. The keys in the left subtree are < (less) than the key in its parent node
3. The keys in theright subtree > (greater) than the key in its parent node
4. Duplicate node keys are not allowed.

Here is an example of a BST



Exercise : Draw the binary tree which would be created by inserting the following numbers in the order given

50 30 25 75 82 28 63 70 4 43 74 35

If the BST is built in a "balanced" fashion, then BST provides log time access to each element. Consider an arbitrary BST of the height k. The total possible number of nodes is given by

k+1

$2 - 1$

In order to find a particular node we need to perform one comparison on each level, or maximum of(k+1) total. Now, assume that we know the number of nodes and we want to figure out the number of comparisons. We have to solve the following equation with respect to k:

Assume that we have a "balanced" tree with n nodes. If the maximum number of comparisons to find an entry is (k+1), where k is the height, we have

$$2^{K+1} - 1 = n$$

we obtain

$$k = \log_2(n+1) - 1 = O(\log_2 n)$$

This means, that a "balanced" BST with *n* nodes has a maximum order of log(n) levels, and thus it takes at most log(n) comparisons to find a particular node. This is the most important fact you need to know about BSTs. But building a BST as a balanced tree is not a trivial task. If the data is randomly distributed, then we can expect that a tree can be "almost" balanced, or there is a good probability that it would be. However, if the data already has a pattern, then just naïve insertion into a BST will result in unbalanced trees. For example, if we just insert the data 1, 2, 3, 4, 5 into a BST in the order they come, we will end up with a tree that looks like this:

Binary search trees work well for many applications (one of them is a dictionary or help browser). But they can be limiting because of their bad worst-case performance height = O(# nodes). Imagine a binary search tree created from a list that is already sorted.

Clearly, the tree will grow to the right or to the left. A binary search tree with this worst-case structure is no more efficient than a regular linked list. A great care needs to be taken in order to keep the tree as balanced as possible. There are many techniques for balancing a tree including AVL trees, and Splay Trees. We will discuss AVL trees in the next lesson. Splay Trees will be discussed in advanced data structure courses like 15-211.

BST OPERATIONS

There are a number of operations on BST's that are important to understand. We will discuss some of the basic operations such as how to insert a node into a BST, how to delete a node from a BST and how to search for a node in a BST.

Inserting a node

A naïve algorithm for inserting a node into a BST is that, we start from the root node, if the node to insert is less than the root, we go to left child, and otherwise we go to the right child of the root. We continue this process (each node is a root for some sub tree) until we find a null pointer (or leaf node) where we cannot go any further. We then insert

the node as a left or right child of the leaf node based on node is less or greater than the leaf node. We note that a new node is always inserted as a leaf node. A recursive algorithm for inserting a node into a BST is as follows. Assume we insert a node N to tree T. if the tree is empty, the we return new node N as the tree. Otherwise, the problem of inserting is reduced to inserting the node N to left of right sub trees of T, depending on N is less or greater than T. A definition is as follows.

Insert(N, T) = N if T is empty
= insert(N, T.left) if N < T
= insert(N, T.right) if N > T

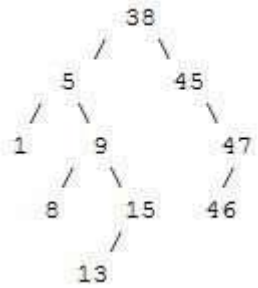
Searching for a node

Searching for a node is similar to inserting a node. We start from root, and then go left or right until we find (or not find the node). A recursive definition of search is as follows. If the node is equal to root, then we return true. If the root is null, then we return false. Otherwise we recursively solve the problem for T.left or T.right, depending on N < T or N > T. A recursive definition is as follows.

Search should return a true or false, depending on the node is found or not.
Search(N, T) = false if T is empty
= true if T = N
= search(N, T.left) if N < T
= search(N, T.right) if N > T

Deleting a node

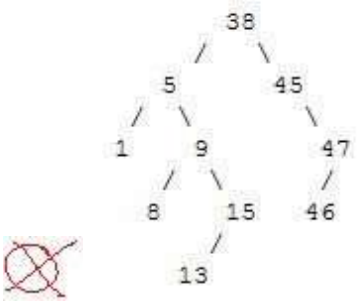
A BST is a connected structure. That is, all nodes in a tree are connected to some other node. For example, each node has a parent, unless node is the root. Therefore deleting a node could affect all sub trees of that node. For example, deleting node 5 from the tree



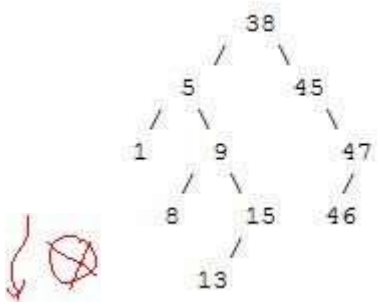
could result in losing sub trees that are rooted at 1 and 9. Hence we need to be careful about deleting nodes from a tree. The best way to deal with deletion seems to be considering special cases. What if the node to delete is a leaf node? What if the node is a node with just one child? What if the node is an internal node (with two children). The latter case is the hardest to resolve. But we will find a way to handle this situation as well.

Case 1 : The node to delete is a leaf node

This is a very easy case. Just delete the node. We are done

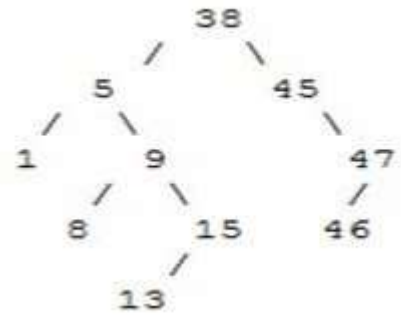


Case 2 : The node to delete is a node with one child.
This is also not too bad. If the node to be deleted is a left child of the parent, then we connect the left pointer of the parent (of the deleted node) to the single child. Otherwise if the node to be deleted is a right child of the parent, then we connect the right pointer of the parent (of the deleted node) to single child.

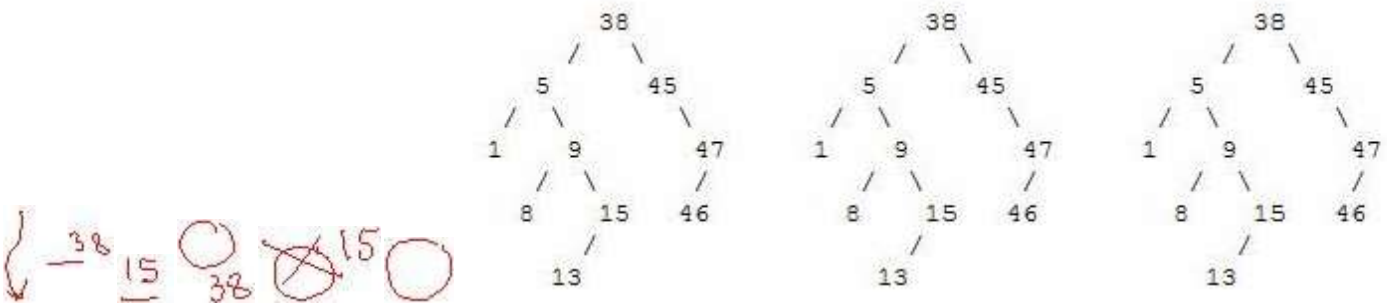


Case 3: The node to delete is a node with two children
This is a difficult case as we need to deal with two sub trees. But we find an easy way to handle it. First we find a replacement node (from leaf node or nodes with one child) for the node to be deleted. We need to do this while maintaining the BST order property. Then we swap leaf node or node with one child with the node to be deleted (swap the data) and delete the leaf node or node with one child (case 1 or case 2)

Next problem is finding a replacement leaf node for the node to be deleted. We can easily find this as follows. If the node to be deleted is N, the find the largest node in the left sub tree of N or the smallest node in the right sub tree of N. These are two candidates that can replace the node to be deleted without losing the order property. For example, consider the following tree and suppose we need to delete the root 38.



Then we find the largest node in the left sub tree (15) or smallest node in the right sub tree (45) and replace the root with that node and then delete that node. The following set of images demonstrates this process.



25. a. Explain Insertion sort with example.
An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, **insertion sort**.

The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array). This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$, where **n** is the number of items.

How Insertion Sort Works?

We take an unsorted array for our example.

Insertion sort compares the first two elements.

It finds that both 14 and 33 are already in ascending order. For now, 14 is in sorted sub-list.

Insertion sort moves ahead and compares 33 with 27.

And finds that 33 is not in the correct position.

It swaps 33 with 27. It also checks with all the elements of sorted sub-list. Here we see that the sorted sub-list has only one element 14, and 27 is greater than 14. Hence, the sorted sub-list remains sorted after swapping.

By now we have 14 and 27 in the sorted sub-list. Next, it compares 33 with 10.

These values are not in a sorted order.

So we swap them.

However, swapping makes 27 and 10 unsorted.

Hence, we swap them too.

Again we find 14 and 10 in an unsorted order.

We swap them again. By the end of third iteration, we have a sorted sub-list of 4 items.

This process goes on until all the unsorted values are covered in a sorted sub-list. Now we shall see some programming aspects of insertion sort.

b. Explain quick sort with example

quicksort uses divide-and-conquer, and so it's a recursive algorithm. The way that quicksort uses divide-and-conquer is a little different from how merge sort does. In merge sort, the divide step does hardly anything, and all the real work happens in the combine step. Quicksort is the opposite: all the real work happens in the divide step.

Divide by choosing any element in the subarray array[p..r]. Call this element the **pivot**. Rearrange the elements in array[p..r] so that all other elements in array[p..r] that are less than or equal to the pivot are to its left and all elements in array[p..r] are to the pivot's right.

Conquer by recursively sorting the subarrays array[p..q-1] (all elements to the left of the pivot, which must be less than or equal to the pivot) and array[q+1..r] (all elements to the right of the pivot, which must be greater than the pivot).

26.a. Write an algorithm for heap sort with example

Heap Sort

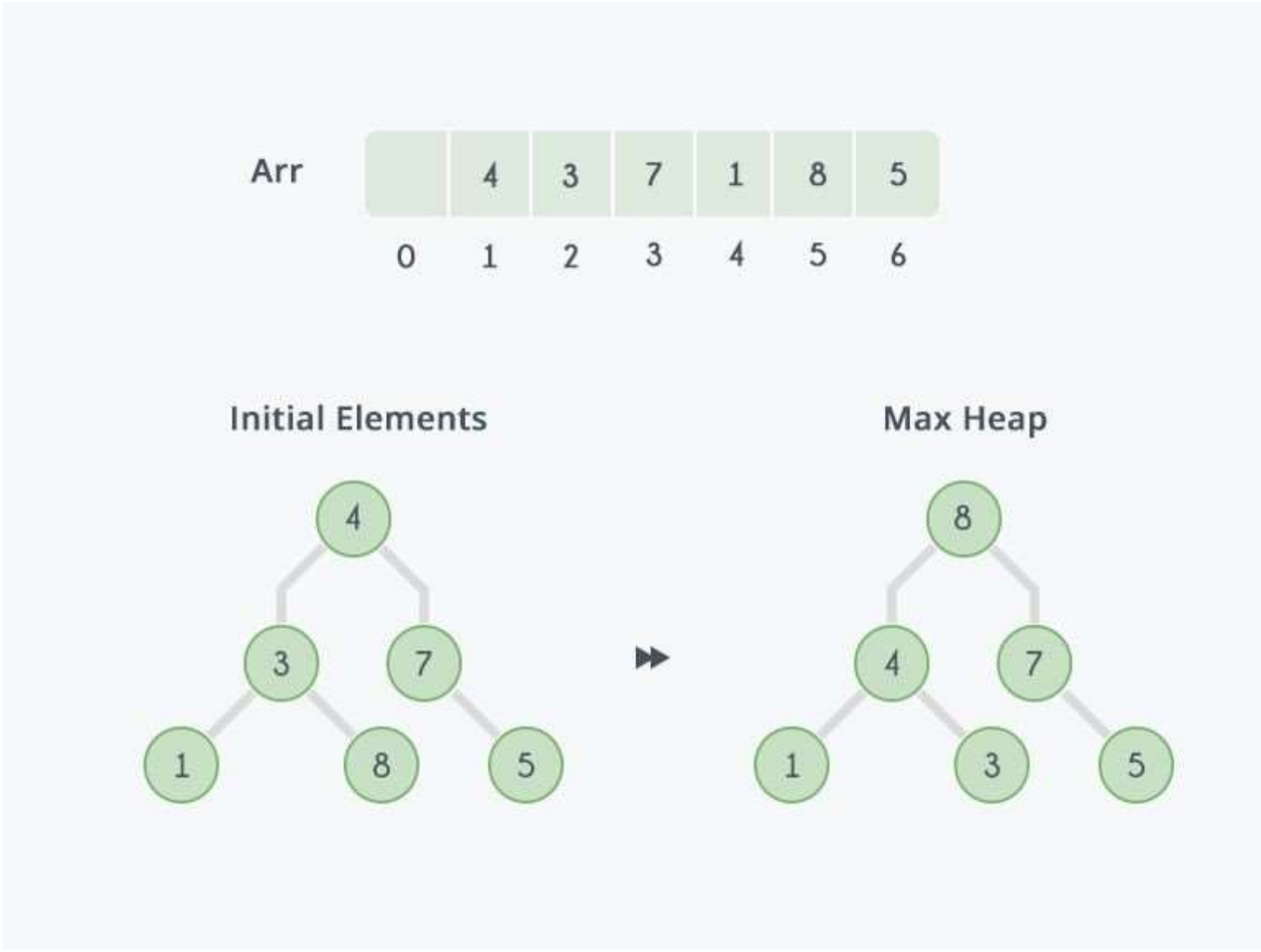
Heaps can be used in sorting an array. In max-heaps, maximum element will always be at the root. Heap Sort uses this property of heap to sort the array.

Complexity:

max_heapify has complexity O(logN)O(logN), build_maxheap has complexity O(N)O(N) and we run max_heapify N−1N−1 times in heap_sort function, therefore complexity of heap_sort function is O(NlogN)O(NlogN).

Example:

In the diagram below,initially there is an unsorted array Arr having 6 elements and then max-heap will be built.



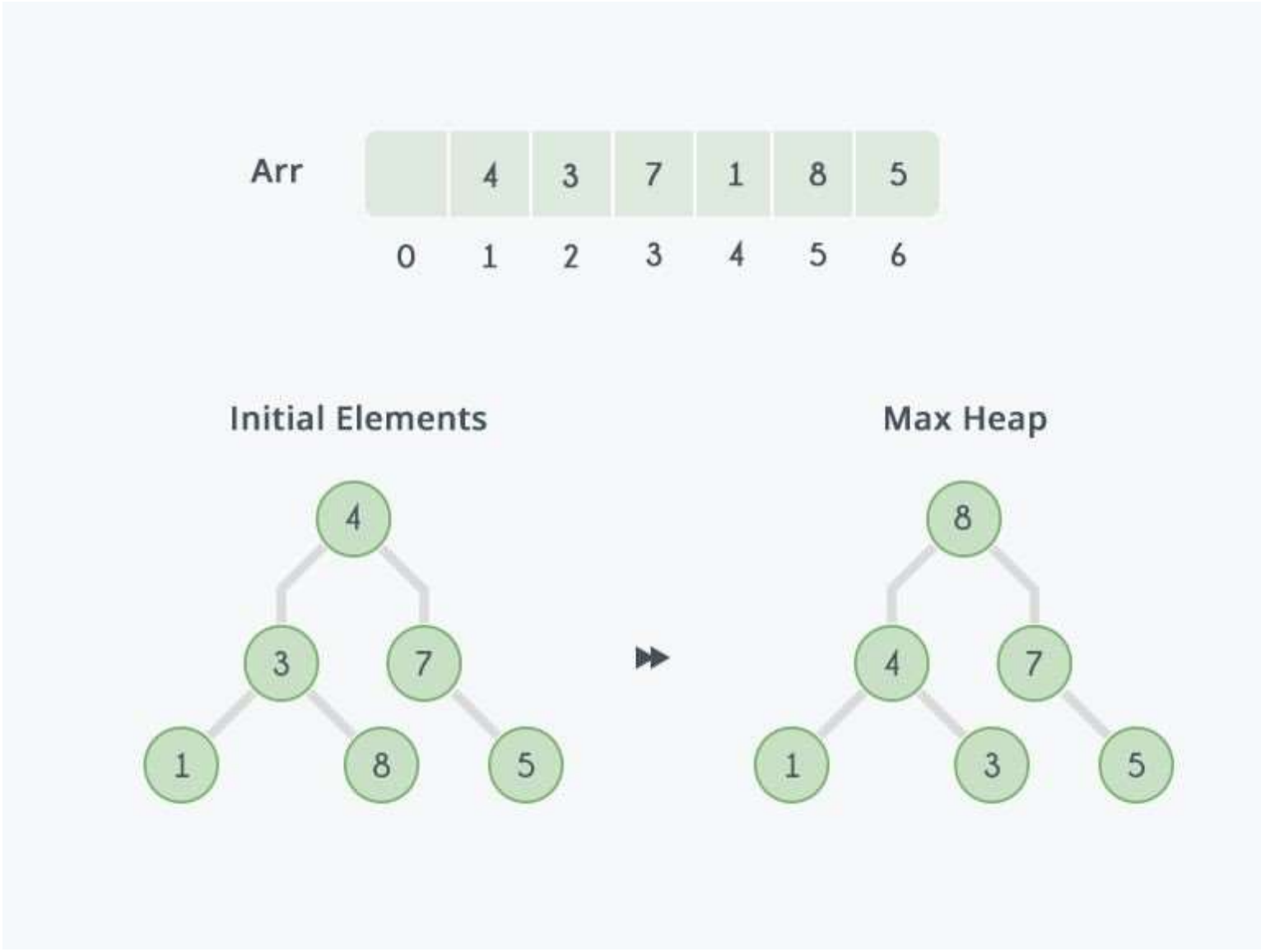
After building max-heap, the elements in the ar

Complexity:

max_heapify has complexity O(logN)O(logN), build_maxheap has complexity O(N)O(N) and we run max_heapify N−1N−1 times in heap_sort function, therefore complexity of heap_sort function is O(NlogN)O(NlogN).

Example:

In the diagram below,initially there is an unsorted array Arr having 6 elements and then max-heap will be built.



After building max-heap, the elements in the ar

b. Write an algorithm for merge sort with example

MERGE SORT:

Divide the unsorted list into *n* sublists, each containing 1 element (a list of 1 element is considered sorted).

Repeatedly merge sublists to produce new sorted sublists until there is only 1 sublist remaining. This will be the sorted list.

Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being O(n log n), it is one of the most respected algorithms.

Merge sort first divides the array into equal halves and then combines them in a sorted manner.

How Merge Sort Works?

To understand merge sort, we take an unsorted array as the following −

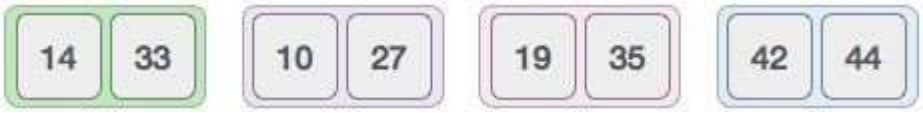
We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.

This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

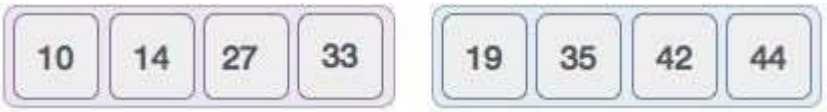
We further divide these arrays and we achieve atomic value which can no more be divided.

Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.

We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.



In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.



After the final merging, the list should look like this −

Reg. No.....

[11CAU302]

KARPAGAM UNIVERSITY

(Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2011 onwards)

BCA DEGREE EXAMINATION, APRIL 2015

Third Semester

COMPUTER APPLICATIONS

DATA STRUCTURES AND ALGORITHMS

Time: 3 hours Maximum : 100 marks

PART – A (15 x 2 = 30 Marks)
Answer ALL the Questions

1. List and explain the two major phases of performance evaluation of an algorithm.
2. How a postfix expression can be evaluated?
3. What is shell sort?
4. How will you represent a polynomial in a linked list?
5. Why linked stacks and queues are needed?
6. What is a head node?
7. What is a Tree?
8. What are the three types of binary tree traversals?
9. What is tree indexing?
10. What is adjacency list?
11. What is a connected and strongly connected component?
12. What is a minimum cost spanning tree?
13. Differentiate static tree table and dynamic tree table.
14. What is a hashing function?
15. What is an AVL tree?

PART B (5 X 14= 70 Marks)
Answer ALL the Questions

16. a. Explain the operations associated with stack, its structure and its implementation.

Or

b. With an example explain the following,
i. Selection Sort ii. 2-Way Merge Sort

17. a. How will you perform insertion and deletion in linked list?

Or

b. With neat diagrams explain the dynamic storage management.

18. a. Explain in detail the three types of binary tree traversal.

Or

b. Discuss about B Trees.

19. a. Explain in detail the three most commonly used representations of graph.

Or

b. Discuss about spanning trees and the stages in Kruskal's Algorithm which leads to minimum cost spanning tree.

20. a. Write in detail about balanced merge sort.

Or

b. Explain the height balanced binary trees in detail.

Reg. No.....

[13CAU302]

KARPAGAM UNIVERSITY

Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2013 onwards)

BCA DEGREE EXAMINATION, NOVEMBER 2015

Third Semester

COMPUTER APPLICATIONS

DATA STRUCTURES AND ALGORITHMS

Time: 3 hours Maximum : 60 marks

PART – A (10 x 2 = 20 Marks)
Answer any TEN Questions

1. List the criteria upon which an algorithm must be analyzed.
2. What is an array? How it can be represented?
3. What is sorting and what are its two broad categories?
4. Compare singly linked list and doubly linked list.
5. Give the procedures used to allocate and release a node.
6. What is the node structure for a sparse matrix representation?
7. How will you find the degree of a node in a tree?
8. Define a skewed tree.
9. Give any two properties of binary search.
10. How is a graph represented?
11. What is a network in a graph?
12. What is a spanning tree?
13. What is a dynamic tree table?
14. Define concept of loading factor.
15. What is a bucket?

PART B (5 X 8= 40 Marks)
Answer ALL the Questions

16. a. Explain the operations associated with queue, its structure and its implementation.

Or

b. With an example explain the following,
i. Selection Sort ii. 2-Way Merge Sort

17. a. Discuss in detail the technique involved in sorting with disks.

Or

b. Explain polyphase merge on tapes with example.

18. a. What is a threaded binary tree? Explain it.

Or

b. Discuss about B Trees.

19. a. Given an undirected graph G = (V, E) and a vertex v in V (G), explain the two ways of visiting all vertices in G that are reachable from V.

Or

b. Explain the following shortest path algorithms
i. Dijkstra's shortest path algorithm ii. All Pairs shortest path algorithm

20. **Compulsory : -**

If your application involves more insertions and deletions of records which data structure you would prefer, array or linked list? Why? Write algorithm to perform insertion and deletion operation.

Reg. No.....

[15CAU301/15CSU301]

KARPAGAM UNIVERSITY

Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2015 onwards)

BCA., B.Sc., DEGREE EXAMINATION, NOVEMBER 2016

Third Semester

COMPUTER APPLICATIONS/COMPUTER SCIENCE

DATA STRUCTURES AND ALGORITHMS

Time: 3 hours

Maximum : 60 marks

PART – A (20 x 1 = 20 Marks) (30 Minutes)
(Question Nos. 1 to 20 Online Examinations)

PART B (5 x 8 = 40 Marks) (2 ½ Hours)
Answer ALL the Questions

21. a. How are the arrays represented? Explain.

Or

b. Discuss QUEUE structure with its operations.

22. a. Write and illustrate the procedure to create, insert and delete nodes in a singly linked list.

Or

b. Explain ALLOCATE procedure in dynamic storage management.

23. a. Elucidate the binary tree traversal procedures.

Or

b. Explain the following : (i) Heap Sort (ii) Binary search

24. a. What are all the ways to represent the graphs?

Or

b. Illustrate the Kruskal algorithm with an example.

25. a. Elucidate polyphase merge sort with an example.

Or

b. Discuss Division and Folding hash functions.

Reg. No.....

[14CAU302]

KARPAGAM UNIVERSITY

Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2014onwards)

BCA DEGREE EXAMINATION, JANUARY 2017

Third Semester

COMPUTER APPLICATIONS

DATA STRUCTURES AND ALGORITHMS

Time: 3 hours Maximum : 60 marks

PART – A (20 x 1 = 20 Marks)
Answer ALL the Questions

1._____ is a sequence of instructions to accomplish a particular task
a. Data Strucuture b. Algorithm c. Ordered List d. Queue

2. _____ criteria of an algorithm ensures that the algorithm terminate after a particular number of steps.
a. effectiveness b. finiteness c. definiteness d. recursive

3. An algorithm must produce _____ output(s)
a. many b. only one c. atleast one d. zero or more

4. _____ criteria of an algorithm ensure that the algorithm must be feasible.
a. effectiveness b. output c. finiteness d. input

5. In Linked List, a _____is a collection of data and link fields.
a. Link b. Node c. Stack d. Data

6. Each item in a node is called a_____.
a. Field b. Data item c. Pointer d. Data

7. The Link field contains the _____ of the next node it points
a. name b. type c. size d. address

8. Data movement and displacing the pointers of the Queue are tedious problems in _____ representation of a Queue.
a. Array b. Linked c. Circular d. Matrix

9. A data structure whose elements forms a sequence of ordered list is called as _____ data structure.
a. Non Linear b. Linear. c. Primitive d. Non Primitive

10. A data structure which represents (non-sequenctial) hierarchical relationship between the elements are called as _____ data structure.
a. Linear b. Primitive. c. Non Linear d. Non Primitive

11. Which of the following is a valid non - linear data structure?
a. Stacks b. Trees c. Queues d. Linked list.

12. _____ are genealogical charts.
a. Stack and Queue b. Pedigree and lineal chart c. Line and bar chart
d. Flow charts

13. In a undirected graph G two vertices v1 and v2 are said to be _____if there is a path in G from v1 to v2.
a. connected b. adjacent c. neighbours d. incident

14. The maximum number of edges in an undirected graph with n veritices is _____
a. n(n-1)/2 b. n/2 c. n-1/2 d. n(n-1)

15. The maximum number of edges in an directed graph with n veritices is _____
a. n(n-1)/3 b. n/3 c. n-1/3 d. n(n-1)

16. In a Graph G if there are n vertices the adjacencies list then consists of _____ nodes.
a. n/2 b. 2n c. n-1 d. n

17. Every binary search tree wth n nodes has _____ sqare node (external nodes).
a. n/2 b. n+1 c. n-1 d. 2^n

18. In a Hash table the address of the identifier x is obtained by applying
a. sequence of comparisions b. binary searching c. arithmetic function
d. collision

19. The partitions of the hash table are called _____
a. Nodes b. Buckets c. Roots d. Fields