



ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY & SCIENCES

Autonomous status accorded by UGC and Andhra University

Approved by AICTE, Permanently Affiliated to Andhra University

Accredited by NBA (IT,CSE,EEE,ECE, and Mech) & accredited by NAAC with "A" Grade

Sangivalasa - 531162, Bheemunipatnam (Mandal), Visakhapatnam (Dist.)

Phone: 08933 - 225083, 225084, 226131, Fax: 08933-226395

Email: principal@anits.edu.in

COLLEGE CODE - ANIL



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DATA STRUCTURES LAB USING C (R-19 CSE-218) MANUAL

Year & Semester: 2/4 B-TECH- SEM-I

VISION:

Our vision is to emerge as a world class Computer Science and Engineering department through excellent teaching and a strong research environment that responds swiftly to the challenges of changing computer science technology and addresses technological needs of the stakeholders.

MISSION:

To enable our students to master the fundamental principles of computing and to develop in them the skills needed to solve practical problems using contemporary computer-based technologies and practices to cultivate a community of professionals who will serve the public as resources on state-of- the-art computing science and information technology.

PROGRAM OUTCOMES (POs):

Graduate Attribute1:	Engineering Knowledge
PO-A	An ability to apply the knowledge of basic engineering sciences, humanities, communication and computing concept in modeling and designing computer based systems.
Graduate Attribute2:	Problem Analysis
PO-B	An ability to identify, analyze the problems in different domains and define the requirements appropriate to the solution.
Graduate Attribute3:	Design/Development of Solution
PO-C	An ability to design, implement & test a computer based system, component or process that meet functional constraints such as public health and safety, cultural, societal and environmental considerations.
Graduate Attribute4:	Conduct Investigations of Complex Problems
PO-D	An ability to apply computing knowledge to conduct experiments and solve complex problems, to analyze and interpret the results obtained within specified timeframe and financial constraints consistently.
Graduate Attribute5:	Modern Tool Usage
PO-E	An ability to apply or create modern techniques and tools to solve engineering problems that demonstrate cognition of limitations involved in design choices.
Graduate Attribute6:	The Engineer and Society
PO-F	An ability to apply contextual reason and assess the local and global impact of professional engineering practices on individuals, organizations and society.
Graduate Attribute7:	Environment and Sustainability
PO-G	An ability to assess the impact of engineering practices on societal and environmental sustainability.
Graduate Attribute8:	Ethics
PO-H	Ability to apply professional ethical practices and transform into good responsible citizen with social concern.
Graduate Attribute9:	Individual and Team Work
PO-I	Acquire capacity to understand and solve problems pertaining to various fields of engineering and be able to function effectively as an individual and as a member or leader in a team.

Graduate Attribute10:	Communication
PO-J	An ability to communicate effectively with range of audiences in both oral and written forms through technical papers, seminars, presentations, assignments, project reports etc.
Graduate Attribute11:	Project Management and Finance
PO-K	An ability to apply the knowledge of engineering, management and financial principles to develop and critically assess projects and their outcomes in multidisciplinary areas.
Graduate Attribute12:	Life-long Learning
PO-L	An ability to recognize the need and prepare oneself for lifelong self learning to be abreast with rapidly changing technology.

PROGRAM SPECIFIC OUTCOMES (PSOs):

1	Programming and software Development skills: Ability to acquire programming efficiency to analyze, design and develop optimal solutions, apply standard practices in software project development to deliver quality software product.
2	Computer Science Specific Skills: Ability to formulate, simulate and use knowledge in various domains like data engineering, image processing and information and network security, artificial intelligence etc., and provide solutions to new ideas and innovations.

Course Objectives:

1. The course is designed to develop skills to design and analyse simple linear and non-linear data structures.
2. It strengthens the ability of the students to identify and apply the suitable data structure for the given real-world problem.
3. It enables them to gain knowledge in practical applications of data structures.

Course Outcomes of the Lab:

CO1	Implement the techniques for searching and sorting (quick and merge)
CO2	Implement of stack and queue and Linked list data structures and their applications.
CO3	Implement operations like insertion, deletion, search and traversing mechanism on binary search tree
CO4	Apply BFS and DFS algorithms to implement graph traversal.

CO-PO Mapping:

Mapping		PO												PSO	
		1	2	3	4	5	6	7	8	9	10	11	12	1	2
CO	1	2	2	2	1		1		1	1	1		1	1	
	2	1	2	2	1		1		1	1	1		1	1	
	3	2	2	2	1		1		1	1	1		1	2	1
	4	2	2	2	1		1		1	1	1		1	2	

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND
SCIENCES

A Laboratory
Manual For
Data Structures (CSE 218)

Semester – I



Prepared by

1. Mrs.S.V.S.S.Lakshmi, Assistant Professor, CSE
2. Mrs. B. Siva Jyothi, Assistant Professor, CSE
3. Mr. P. Krishnanjaneyulu, Assistant Professor, CSE

b) Quadratic probing: Quadratic Probing we look for i^2 'th slot in i 'th iteration, let store k keys into an array of size S at the location computed using a hash function, $\text{hash}(x)$ where $k \leq n$ and k takes values from $[1 \text{ to } m]$, $m > n$.

Constraints: let $\text{hash}(x)$ be the slot index computed using hash function.

If slot $\text{hash}(x) \% S$ is full, then we try $(\text{hash}(x) + 1^2) \% S$

If $(\text{hash}(x) + 1^2) \% S$ is also full, then we try $(\text{hash}(x) + 2^2) \% S$

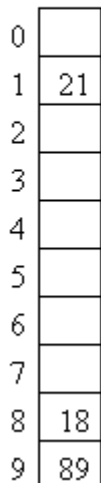
If $(\text{hash}(x) + 2^2) \% S$ is also full, then we try $(\text{hash}(x) + 3^2) \% S$

Sample Test Case:

Insert
18, 89, 21

Insert
58

Insert
68

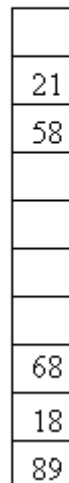


For **58**:

- $H = \text{hash}(58, 10) = 8$
- Probe sequence:
 - $i = 0, (8+0) \% 10 = 8$
 - $i = 1, (8+1) \% 10 = 9$
 - $i = 2, (8+4) \% 10 = 2$

For **68**:

- $H = \text{hash}(68, 10) = 8$
- Probe sequence:
 - $i = 0, (8+0) \% 10 = 8$
 - $i = 1, (8+1) \% 10 = 9$
 - $i = 2, (8+4) \% 10 = 2$
 - $i = 3, (8+9) \% 10 = 7$

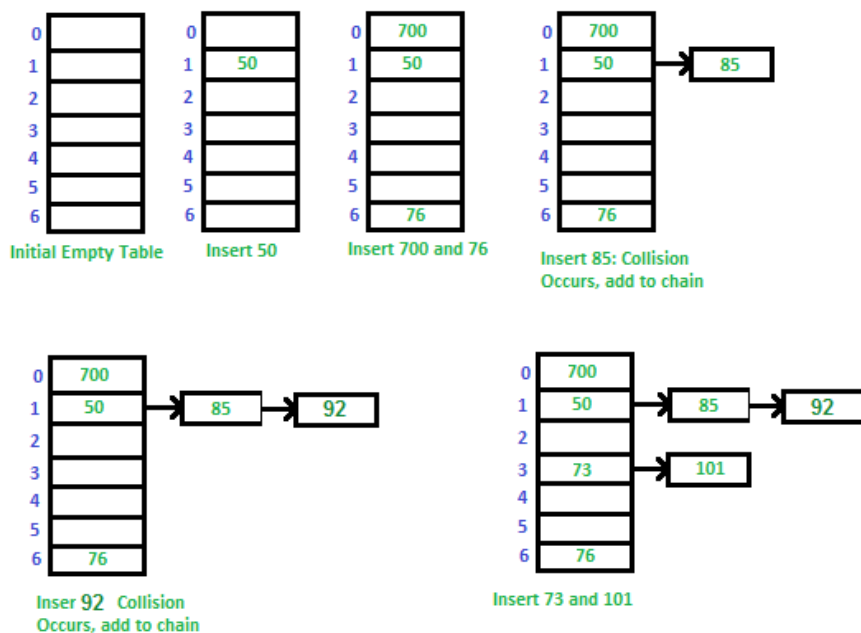


c) Separate Chaining: The idea is to make each cell of hash table points to a linked list of records that have same hash function value.

Let us store K keys into hash table of size S , where $k \leq n$ and k takes values from $[1 \text{ to } m]$, $m > n$.

Sample Test Case:

Let us consider a simple hash function as "key mod 7" and sequence of keys as 50, 700, 76, 85, 92, 73, 101.

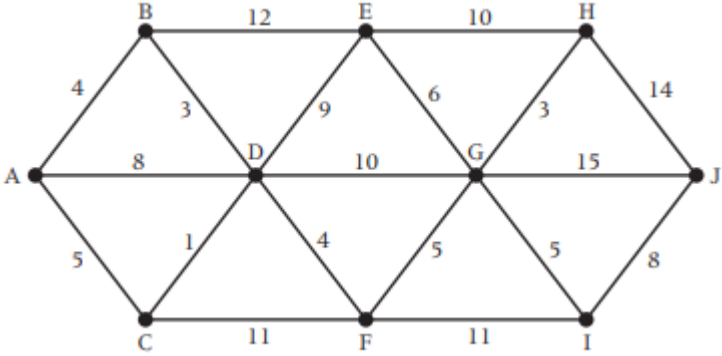


5. Design, Develop and Implement a menu driven Program in C for the following.

a) Operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)

	<ol style="list-style-type: none"> 1. Push an Element on to Stack 2. Pop an Element from Stack 3. Demonstrate Overflow and Underflow situations on Stack 4. Display the status of Stack 5. Exit 	
	<p>b) Operations on QUEUE of Characters (Array Implementation of Queue with maximum size MAX)</p> <ol style="list-style-type: none"> 1. Insert an Element on to QUEUE 2. Delete an Element from QUEUE 3. Demonstrate Overflow and Underflow situations on QUEUE 4. Display the status of QUEUE 5. Exit <p>Note: Support the program with appropriate functions for each of the above operations</p>	CO2
6.	<p>Design, Develop and Implement a C program to do the following using a singly linked list.</p> <p>a) Stack- In single linked list store the information in the form of nodes .Create nodes using dynamic memory allocation method. All the single linked list operations perform based on Stack operations LIFO (last in first out). A stack contains a top pointer. Which is “head” of the stack where pushing and popping items happens at the head of the list. first node have null in link field and second node link have first node address in link field and so on and last node address in “top” pointer. Stack Operations:</p> <ol style="list-style-type: none"> 1. push() : Insert the element into linked list nothing but which is the top node of Stack. 2. pop() : Return top element from the Stack and move the top pointer to the second node of linked list or Stack. 3. peek(): Return the top element. 4. display(): Print all element of Stack. 	CO2
	<p>b) Queue- All the single linked list operations perform based on queue operations FIFO (First in first out). In a Queue data structure, we maintain two pointers, <i>front</i> and <i>rear</i>. The <i>front</i> points the first item of queue and <i>rear</i> points to last item.</p> <ol style="list-style-type: none"> 1. enqueue() This operation adds a new node after <i>rear</i> and moves <i>rear</i> to the next node. 2. dequeue() This operation removes the front node and moves <i>front</i> to the next node. 3. Display() Display all elements of the queue. <p>Note: Sample node information: Student Data with the fields: <i>USN, Name, Branch, Sem, PhNo</i>.</p>	CO2
7.	<p>7. Design, Develop and Implement a Program in C for the following</p> <p>a) Converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.</p>	CO2
	<p>b) Evaluation of postfix expression with single digit operands and operators: +, -, *, /, %, ^</p>	CO2
8.	<p>Design, Develop and Implement a menu driven Program in C for the following :</p>	CO2

	<p>a) Circular Queue</p> <ol style="list-style-type: none"> 1. Insert an Element on to Circular QUEUE 2. Delete an Element from Circular QUEUE 3. Demonstrate <i>Overflow</i> and <i>Underflow</i> situations on Circular QUEUE 4. Display the status of Circular QUEUE 5. Exit 	
	<p>b) Priority Queue</p> <ol style="list-style-type: none"> 1. Insert an Element on to Priority QUEUE 2. Delete an Element with highest priority from Priority QUEUE 3. Demonstrate <i>Overflow</i> and <i>Underflow</i> situations on Priority QUEUE 4. Display the status of Priority QUEUE 5. Exit <p>Support the program with appropriate functions for each of the above operations</p>	CO2
9.	<p>Design, Develop and Implement a menu driven C program to Perform Operations on dequeue (double ended queue) using circular array.</p> <ol style="list-style-type: none"> 1. insertFront(): Adds an item at the front of Deque. 2. insertRear(): Adds an item at the rear of Deque. 3. deleteFront(): Deletes an item from front of Deque 4. deleteRear(): Deletes an item from rear of Deque 5. getFront(): Gets the front item from queue 6. getRear(): Gets the last item from queue 7. isEmpty(): Checks whether Deque is empty or not 8. isFull(): Checks whether Deque is full or not <p>Support the program with appropriate functions for each of the above operations.</p>	CO2
10.	<p>Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers</p> <ol style="list-style-type: none"> 1. Create a BST of N Integers: 13, 3, 4, 12, 14, 10, 5, 1, 8, 2, 7, 9, 11, 6, 18 2. Traverse the BST(either inorder, preorder or postorder) 3. Search the BST for a given element (KEY) and report the appropriate message 4. Exit 	CO3
11.	<p>Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers</p> <ol style="list-style-type: none"> 1. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2 2. Traverse the BST in Inorder, Preorder and Post Order using non-recursive functions 3. exit 	CO3
12.	<p>Design, Develop and Implement a Program in C for the following operations on Graph(G) of Cities</p> <ol style="list-style-type: none"> 1. Create a Graph of N cities using Adjacency Matrix. 2. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method 	CO4
13.	<p>Design, Develop and Implement a C Program to the problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph using Warshall's Algorithm. The Graph is represented as Adjacency Matrix, and the Matrix denotes the weight of the edges (if it exists) else INF (1e7).</p> <p>Input: The first line of input contains an integer T denoting the no of test cases. Then T test cases follow. The first line of each test case contains an integer V denoting the size of the adjacency matrix. The next V lines contain V space separated values of the matrix (graph). All input will be integer type.</p>	CO4

	<p>Output: For each test case output will be V*V space separated integers where the i-jth integer denote the shortest distance of ith vertex from jth vertex. For INT_MAX integers output INF.</p> <p>Constraints: $1 \leq T \leq 20$ $1 \leq V \leq 100$ $1 \leq \text{graph}[][] \leq 500$</p>	
14.	<p>Design, Develop and Implement a C Program to Find the shortest distance from A to J on the network below using Dijkstra's Algorithm.</p> 	CO4

LIST OF INDUSTRY RELEVANT SKILLS:

After completing the course Student able to

- Develop Algorithm for the given scenario.
- Develop Software.
- Store and Organize the Data efficiently.

GUIDELINES TO TEACHERS

- Faculty should verify Students Id cards and Dress code before enter into Lab
- Faculty must verify the observations before assigning the system.
- Faculty must take the attendance starting and ending of the lab time period.
- Faculty should follow rubrics while evaluating the student work.

Sessional marks : 50 marks

- Daily Evaluation (Includes Record, Observation & regular performance) – 25
- Attendance – 5 marks
- Internal Exam – 20 marks (Write-up – 10, Execution – 5, Viva- 5)

External Exam (50 marks)

- Viva voce – 10 marks
- Write up + Execution – 40 marks

INSTRUCTIONS TO STUDENTS:

- Students should come to Lab with proper uniform and Id –card.
- Student must bring Observation and Record to the Lab.
- Students should Lab related components smoothly
- Students should not carry other items into lab.
- Student always need to login to the allocated system only.
- Student should not open any other applications unnecessarily.
- Students should not corrupt the others works in the system.

GUIDELINES TO LAB PROGRAMMERS:

- Lab Programmers must verify All the Systems whether they are working properly or not.
- Lab Programmers should switch on the power connections and AC 10 minutes before the Lab.

LAB RUBRICS

Key Performance Criteria(KPC) (25 pts)	4-Very Good	3-Good	2-Fair	1-Need to improve
Problem Statement (2)	Detail understanding of the problem (2)	Understanding of the problem (2)	Basic understanding of the problem (1)	Partial understanding of the problem (1)
Experimental Procedure/ algorithm/ flow chart/ analysis (4)	The procedure is explained and well designed the problem with appropriate analysis (4)	The procedure is explained and designed the problem with analysis (3)	Missing some experimental procedure with partial analysis (2)	Missing major experimental details and analysis (1)
Implementation (4)	Implement Optimal solution with appropriate results for all the inputs	Implement solution with correct results for most of the inputs	implement solution with the correct answers for some inputs and results wrong answers for some cases	Implement Solution does not produce the appropriate results for the given inputs
Test Case verification (2)	Produces correct output for all possible test cases(2)	Produces correct output for most of the test cases (2)	Produces correct output for some of the test cases (1)	Produces Wrong output for most of the test cases (1)
Code of conduct (courtesy, safety, behavioral aspects, ethics etc.)(3)	While conducting the procedure, the student is in proper dress code, always respectful of others and leaves the area clean.(3)	While conducting the procedure, the student is in proper dress code, many times respectful of others and leaves the area clean only after being reminded.(2)	While conducting the procedure, the student is in partial dress code, sometimes respectful of others and leaves the area clean only after being reminded.(2)	While conducting the procedure, the student is not in proper dress code , not respectful of others and leaves the area messy even after being reminded.(1)
Viva voce / oral presentation(5)	In depth knowledge on the concept and answered all the questions(5)	Good knowledge on the concept and answered all the questions(4)	Basic knowledge on the concept and answered some of the questions(3)	With basic knowledge on the concept and answered few questions(2)

Presentation of record / documentation(5)	Presented the content effectively and Submitted on time (5)	Presented the content and Submitted on time (4)	Presented the incomplete content and Submitted. (3)	Presented the wrong content and submitted delay.(2)
--	---	---	---	---

PRACTICAL 1:

Write a program to sort the given array of N elements using divide and conquer method (merge sort and quick sort algorithms)

1. Practical significance:

A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison of the elements. The comparison is used to decide the new order of element in the respective data structure.

2. Relevant Program Outcomes :

PO1, PO 2, PO3 - Moderate,
PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

Able to sort the list of elements of any size in Ascending/Descending Order

4. Prerequisites:

- Basic knowledge about problem solving
- Require programming knowledge through C language

5. Resources required:

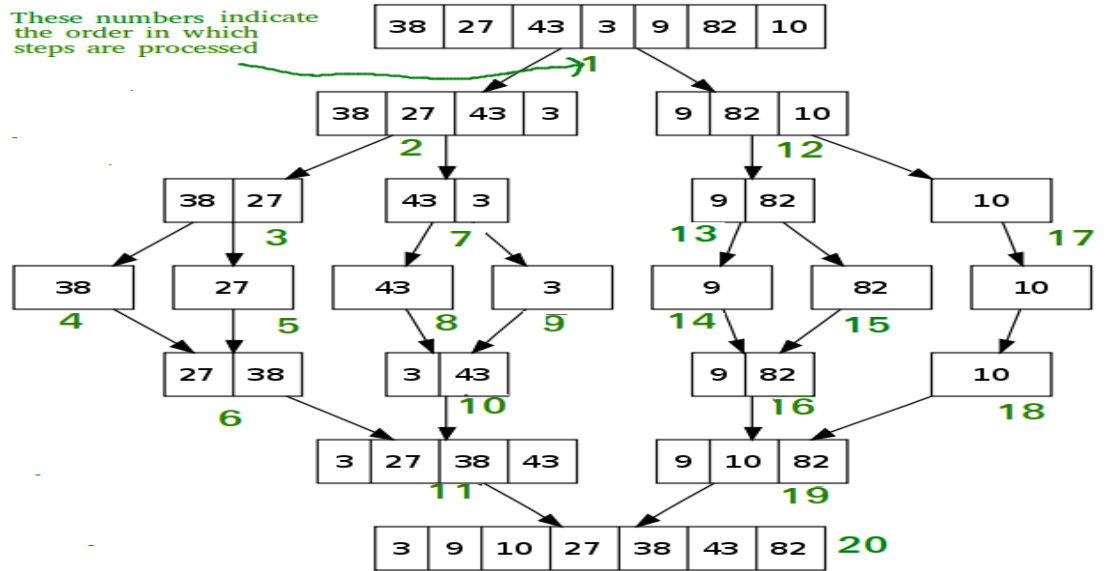
S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	1. Processor – 2GHz 2. RAM – 4GB 3. Hard-Drive Space – 20GB 4. VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX /LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

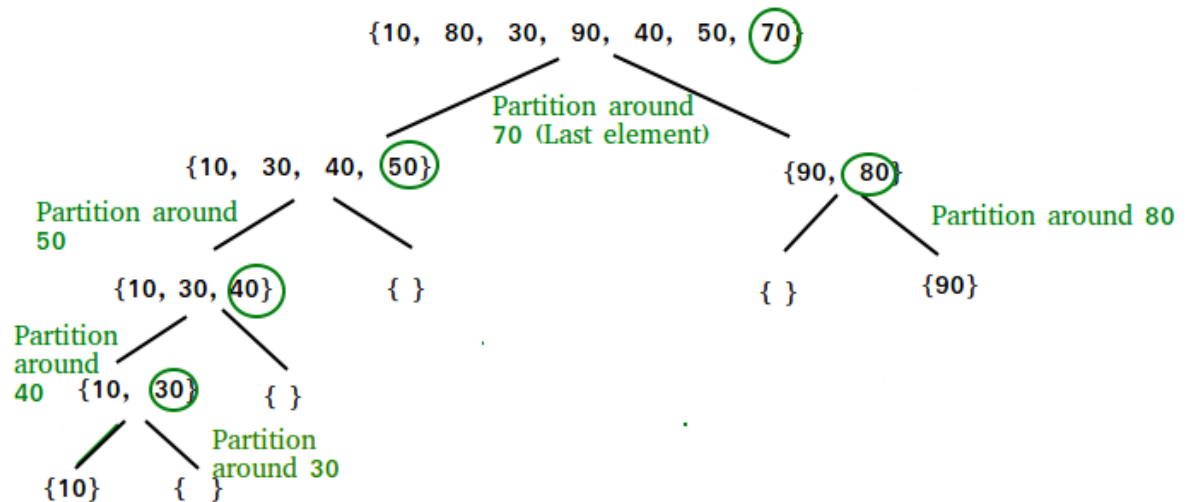
7. Algorithm/circuit/Diagram/Description:

a) **Merge sort** is one of the most efficient sorting algorithms. It works on the principle of Divide and Conquer. Merge sort repeatedly breaks down a list into several sublists until each sublist consists of a single element and merging those sublists in a manner that results into a sorted list.



b) **Quick Sort** is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.



8. Test cases:

Sample Input array: 87, 36,9, 12,24, 5, 78, 567, 456, 34, 96, 45, 39, and 89,123

9. Sample output:

Sample Output array:5,9, 12, 24, 34, 36, 39, 45, 78, 87, 89, 96, 123, 456, and

567

10. Practical Related Questions:

1. What is their time complexity?

Algorithm	Time Complexity		
	Best	Average	Worst
Selection Sort	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$
Bubble Sort	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
Insertion Sort	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
Heap Sort	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n \log(n))$
Quick Sort	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n^2)$
Merge Sort	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n \log(n))$
Bucket Sort	$\Omega(n+k)$	$\theta(n+k)$	$O(n^2)$
Radix Sort	$\Omega(nk)$	$\theta(nk)$	$O(nk)$

11. Exercise Questions:

1. Find the largest element from the given list of elements
2. Find the Smallest element from the given list of elements

PRACTICAL 2:

Write a C Program to search whether an item K present in an array of N elements (Using Linear and binary Search algorithms).

1. Practical significance:

Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored.

2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,

PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

Able to find the element or position of the element using Linear or Binary search in the group of elements.

4. Prerequisites:

- Basic knowledge about problem solving
- Require programming knowledge through C language

5. Resources required:

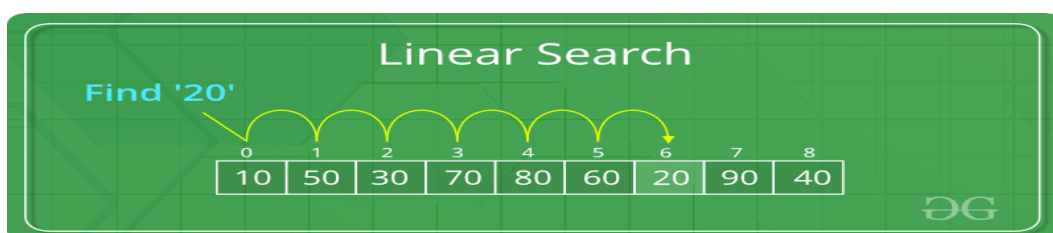
S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX /LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

7. Algorithm/circuit/Diagram/Description:

a) Linear Search



- Sequential search starts at the beginning of the list and checks every element of the list.
- It is a basic and simple search algorithm.

- Sequential search compares the element with all the other elements given in the list.
If the element is matched, it returns the value index, else it returns -1.

b) Binary Search



Steps of binary search algorithm:

- Select the element in the middle of the array.
- Compare the selected element to the searched element, if it is equal to the searched element, terminate.
- If the searched element is larger than the selected element, repeat the search operation in the major part of the selected element.
- If the searched element is smaller than the selected element, repeat the search in the smaller part of the selected element.
- Repeat the steps until the smallest index in the search space is less than or equal to the largest index.

8. Test cases:

Sample Input array: 45, 78, 123, 48, 34, 89, 67, 54, and 74, 543

Search Item: 34

Search Item: 343

9. Sample output:

Output: Key Found

Output: Key Not Found

10. Practical Related Questions:

- What is their time complexity?
- Tell the Differences between Linear Search & Binary Search?
- Which searching is a Good Technique? Why ?

11. Exercise Questions:

- Implement the Both Searching techniques in One program using Switch Case.

PRACTICAL 3:

Write a C program to store k keys into an array of size n at the location computed using a hash function, $loc = key \% n$, where $k \leq n$ and k takes values from $[1 \text{ to } m]$, $m > n$.

1. Practical significance:

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.

2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,

PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

Able to compute index for the given key value

4. Prerequisites:

- Basic knowledge about problem solving
- Require programming knowledge through C language

5. Resources required:

S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX /LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

7. Algorithm/circuit/Diagram/Description:

Step1: Read the size of the hash table – n

Step2: Read number of key values – k where $k < n$ and k takes values from [1 to m], $m > n$.

Step3: Read each key value

Step4: Compute the index location to store key value with the given hash function [$loc = key \% n$]

Step5: Store $a[loc] = key$

Step6: print the index values of each key.

8. Test cases:

Input 1	Input2
10	10
5	6
11	10
12	12
13	31
14	40
15	56
	67

9. Sample output:

Output1:

```
0 located at 0 index
11 located at 1 index
12 located at 2 index
13 located at 3 index
14 located at 4 index
15 located at 5 index
0 located at 6 index
0 located at 7 index
0 located at 8 index
0 located at 9 index
```

Output2:

```
40 located at 0 index
31 located at 1 index
12 located at 2 index
0 located at 3 index
0 located at 4 index
0 located at 5 index
56 located at 6 index
67 located at 7 index
0 located at 8 index
0 located at 9 index
```

10. Practical Related Questions:

1. What is n represented for?
2. Is there any size constraint on k ?
3. How to Compute hash function?
4. What is key value?
5. What is loc ?

11. Exercise Questions:

1. What is hashing?
2. What is the use of hashing?
3. What is the drawback you identified in hashing?

PRACTICAL 4 (A):

Design, Develop and Implement a C program to handle the collisions using the Linear Probing collision resolution Technique

1. Practical significance:

Linear probing: In linear probing, we linearly probe for next slot, let store k keys into an array of size S at the location computed using a hash function, $\text{hash}(x)$ where $k \leq n$ and k takes values from $[1 \text{ to } m]$, $m > n$.

Constraints: If slot $\text{hash}(x) \% S$ is full, then we try $(\text{hash}(x) + 1) \% S$, If $(\text{hash}(x) + 1) \% S$ is also full, then we try $(\text{hash}(x) + 2) \% S$, If $(\text{hash}(x) + 2) \% S$ is also full, then we try $(\text{hash}(x) + 3) \% S$

2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,

PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

Able to compute index for the given key value by applying Linear Probing Technique.

4. Prerequisites:

- Basic knowledge about problem solving
- Require programming knowledge through C language

5. Resources required:

S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX /LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

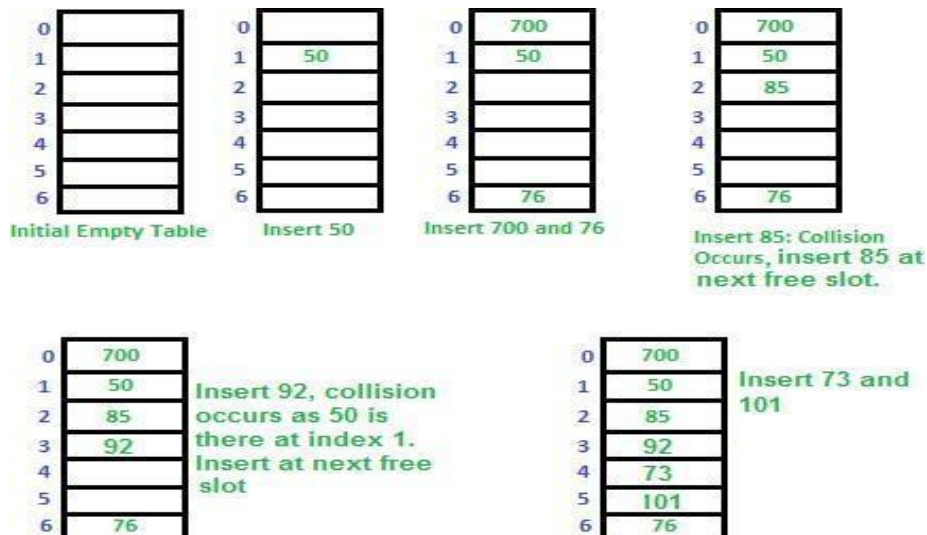
7. Algorithm/circuit/Diagram/Description:

1. Read the size of the hash table – n
2. Read number of key values – k where $k < n$ and k takes values from [1 to m], $m > n$.
3. Read each key value
4. Compute the index location to store key value with the given hash function [$loc = key \% n$]
5. if loc is not free, until a[loc] equals to 0
loc := loc+1
6. Store a[loc] = key
7. print the index values of each key.

8. Test cases:

Sample Test Case:

Let us consider a simple hash function as —key mod 7 and sequence of keys as 50, 700, 76, 85, 92, 73, 101



9. Sample output:

```
key 700 is located at location 0
key 50 is located at location 1
key 85 is located at location 2
key 92 is located at location 3
key 73 is located at location 4
key 101 is located at location 5
key 76 is located at location 6
```

10. Practical Related Questions:

1. If hash(x) is full, what to do?
2. How to check if hash(x) +1 is free or not?
3. How to get the next free index value?

11.Exercise Questions:

1. What is collision?
2. What is linear probing?
3. What is worst case time complexity in linear probing for search key in hash table?

PRACTICAL 4(b):

Design, Develop and Implement a C program to handle the collisions using the Quadratic Probing collision resolution Technique

1. Practical significance:

Quadratic probing: Quadratic Probing we look for i^2 th slot in i th iteration, let store k keys into an array of size S at the location computed using a hash function, $\text{hash}(x)$ where $k \leq n$ and k takes values from $[1$ to $m]$, $m > n$.

Constraints: let $\text{hash}(x)$ be the slot index computed using hash function.

If slot $\text{hash}(x) \% S$ is full, then we try $(\text{hash}(x) + 1*1) \% S$

If $(\text{hash}(x) + 1*1) \% S$ is also full, then we try $(\text{hash}(x) + 2*2) \% S$

If $(\text{hash}(x) + 2*2) \% S$ is also full, then we try $(\text{hash}(x) + 3*3) \% S$

2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,

PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

Able to compute index for the given key value by applying Quadratic Probing Technique

4. Prerequisites:

- Basic knowledge about problem solving
- Require programming knowledge through C language

5. Resources required:

S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX / LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

7. Algorithm/circuit/Diagram/Description:

1. Read the size of the hash table – n
2. Read number of key values – k where $k < n$ and k takes values from [1 to m], $m > n$.
3. Read each key value
4. Compute the index location to store key value with the given hash function [$loc = key \% n$]
5. if loc is not free, until $a[loc]$ equals to 0
 $loc := loc + i * i$
6. Store $a[loc] = key$
7. print the index values of each key.

8. Test cases:

Insert 18, 89, 21 Insert 58

0		
1	21	21
2		58
3		
4		
5		
6		
7		
8	18	18
9	89	89

For **58**:

- $H = \text{hash}(58, 10) = 8$
- Probe sequence:
 $i = 0, (8+0) \% 10 = 8$
 $i = 1, (8+1) \% 10 = 9$
 $i = 2, (8+4) \% 10 = 2$

Insert 68

21	
58	
68	
18	
89	

For **68**:

- $H = \text{hash}(68, 10) = 8$
- Probe sequence:
 $i = 0, (8+0) \% 10 = 8$
 $i = 1, (8+1) \% 10 = 9$
 $i = 2, (8+4) \% 10 = 2$
 $i = 3, (8+9) \% 10 = 7$

9. Sample output:

```
5
18
key 18 is located at location 8
89
key 89 is located at location 9
21
key 21 is located at location 1
58
key 58 is located at location 3
68
key 68 is located at location 2
```

10. Practical Related Questions:

1. Explain the functioning of $(h(x) + i^2) \% s$
2. What is $h(x)$?

11. Exercise Questions:

1. What is the drawback of linear probing?
2. What is quadratic probing?
3. Compare the worst case time complexity of linear probing with quadratic probing?

PRACTICAL 4(c):

Design, Develop and Implement a C program to handle the collisions using the Separate Chaining collision resolution Technique

1. Practical significance:

Separate Chaining: The idea is to make each cell of hash table points to a linked list of records that have same hash function value.

Let us store K keys into hash table of size S , where $k \leq n$ and k takes values from $[1 \text{ to } m]$, $m > n$.

2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,

PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

Able to compute index for the given key value by applying Separate Chaining Technique.

4. Prerequisites:

- Basic knowledge about problem solving
- Require programming knowledge through C language

5. Resources required:

S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX /LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

7. Algorithm/circuit/Diagram/Description:

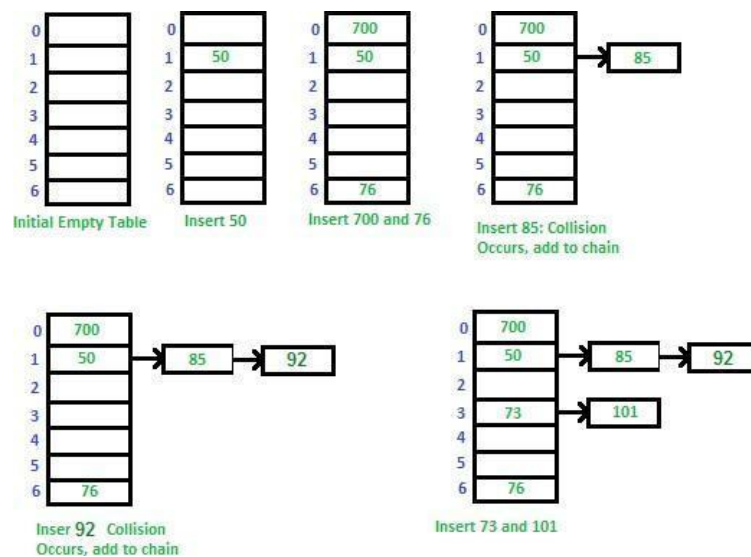
1. Read the size of the hash table – n
2. Read number of key values – k where $k < n$ and k takes values from $[1 \text{ to } m]$, $m > n$.
3. allocate memory for new node and read key value

4. Compute the index location to store key value with the given hash function [$loc = key \% n$]
5. if location is free, add the new node to that location
6. Else add the new node as next node to the node in that location.
7. print the hash table locations with the chain of nodes.

8. Test cases:

Sample Test Case:

Let us consider a simple hash function as —**key mod 7** and sequence of keys as 50, 700, 76, 85, 92, 73, 101.



9. Sample output:

```
chain[0]-->700 -->NULL
chain[1]-->50 -->85 -->92 -->NULL
chain[2]-->NULL
chain[3]-->73 -->101 -->NULL
chain[4]-->NULL
chain[5]-->NULL
chain[6]-->76 -->NULL
```

10. Practical Related Questions:

1. How to allocate memory for new node?
2. How to add node to the hash table location?
3. How to print the chain of nodes in the given location?

11. Exercise Questions:

1. What is the data structure used to implement separate chaining?
2. What is time complexity for inserting an element using this technique?

PRACTICAL 5(a):

Design, Develop and Implement a menu driven Program in C for stack.

Operations on **STACK** of Integers (Array Implementation of Stack with maximum size **MAX**)

1. **Push** an Element on to Stack
2. **Pop** an Element from Stack
3. Demonstrate **Overflow** and **Underflow** situations on Stack
4. Display the status of Stack
5. Exit

1. Practical significance:

Stack is one of the Linear Data Structure. We can insert the elements using push function and delete the elements using pop function. The Variable top is used to maintain the position of the topmost element in the stack. We can do any operations on stack though top variable only.

2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,

PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

Able to perform the operations on Stack.

4. Prerequisites:

- Basic knowledge about problem solving
- Require programming knowledge through C language

5. Resources required:

S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX /LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

7. Algorithm/circuit/Diagram/Description:

Algorithm for Push Operation

- Step 1** – Checks if the stack is full.
- Step 2** – If the stack is full, produces an error and exit.
- Step 3** – If the stack is not full, increments **top** to point next empty space.
- Step 4** – Adds data element to the stack location, where **top** is pointing.
- Step 5** – Returns success.

Algorithm for Pop Operation

- Step 1** – Checks if the stack is empty.
- Step 2** – If the stack is empty, produces an error and exit.
- Step 3** – If the stack is not empty, accesses the data element at which **top** is pointing.
- Step 4** – Decreases the value of **top** by 1.
- Step 5** – Returns success.

8. Test cases:

Sample Input :

```
push(2)
push(3)
pop()
push(4)
pop()
```

9. Sample output:

```
3, 4
```

10. Practical Related Questions:

1. Define Stack.
2. Tell the Differences between Array & Linked List
3. What is Overflow?
4. What is Underflow?
5. What are the Application of Stack?

11. Exercise Questions:

Implement a program to find the Value of the Top after performing 5 PUSH operations and 2 POP operations continuously in stack.

PRACTICAL 5(b):

Design, develop and Implement a menu driven Program in C for the following.

1. Insert an Element on to QUEUE
2. Delete an Element from QUEUE
3. Demonstrate *Overflow* and *Underflow* situations on QUEUE
4. Display the status of QUEUE
5. Exit

Note: Support the program with appropriate functions for each of the above operations

Operations on QUEUE of Characters (Array Implementation of Queue with maximum size MAX)

1. Practical significance:

Queue Using Arrays

A queue data structure can be implemented using one dimensional array. The queue implemented using array stores only fixed number of data values.

The implementation of queue data structure using array is very simple. Just define a one dimensional array of specific size and insert or delete the values into that array by using **FIFO (First In First Out) principle** with the help of variables 'front' and 'rear'. Initially both 'front' and 'rear' are set to -1.

Whenever, we want to insert a new value into the queue, increment 'rear' value by one and then insert at that position. Whenever we want to delete a value from the queue, then delete the element which is at 'front' position and increment 'front' value by one.

Queue Operations using Array

We can Perform the following operations on Queue

- 1.enQueue()
- 2.deQueue()
- 3.Display()

2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,
PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

1. Able to perform ENQUEUE operation
2. Able to perform DEQUEUE operation
3. Able to display QUEUE data items.

4. Prerequisites:

- Basic knowledge about problem solving
- Require programming knowledge through C language

5. Resources required:

S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX /LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

7. Algorithm/circuit/Diagram/Description:

Follow the below steps to create an empty queue.

Step 1 - Include all the **header files** which are used in the program and define a constant '**SIZE**' with specific value.

Step 2 - Declare all the **user defined functions** which are used in queue implementation.

Step 3 - Create a one dimensional array with above defined SIZE (**int queue[SIZE]**).

Step 4 - Define two integer variables '**front**' and '**rear**' and initialize both with '**-1**'. (**int front = -1, rear = -1**).

Step 5 - Then implement main method by displaying menu of operations list and make

suitable function calls to perform operation selected by the user on queue.

1. enQueue(value) - Inserting value into the queue

In a queue data structure, enQueue() is a function used to insert a new element into the queue. In a queue, the new element is always inserted at **rear** position. The enQueue() function takes one integer value as a parameter and inserts that value into the queue. We can use the following steps to insert an element into the queue...

Step 1 - Check whether **queue** is **FULL**. (**rear == SIZE-1**)

Step 2 - If it is **FULL**, then display "**Queue is FULL!!! Insertion is not possible!!!**" and terminate the function.

Step 3 - If it is **NOT FULL**, then increment **rear** value by one (**rear++**) and set **queue[rear] = value**.

2. deQueue() - Deleting a value from the Queue

In a queue data structure, deQueue() is a function used to delete an element from the queue. In a queue, the element is always deleted from **front** position. The deQueue() function does not take any value as parameter. We can use the following steps to delete an element from the queue...

Step 1 - Check whether **queue** is **EMPTY**. (**front == rear**)

Step 2 - If it is **EMPTY**, then display "**Queue is EMPTY!!! Deletion is not possible!!!**" and terminate the function.

Step 3 - If it is **NOT EMPTY**, then increment the **front** value by one (**front ++**). Then display **queue[front]** as deleted element. Then check whether both **front** and **rear** are equal (**front == rear**), if it **TRUE**, then set both **front** and **rear** to **-1** (**front = rear = -1**).

3.Display() - Displays the elements of a Queue

We can use the following steps to display the elements of a queue

Step 1 - Check whether **queue** is **EMPTY**. (**front == rear**)

Step 2 - If it is **EMPTY**, then display "**Queue is EMPTY!!!!**" and terminate the function.

Step 3 - If it is **NOT EMPTY**, then define an integer variable **'i'** and set **'i = front+1'**.

Step 4 - Display **'queue[i]'** value and increment **'i'** value by one (**i++**). Repeat the same until **'i'** value reaches to **rear** (**i <= rear**).

8. Test cases:

9. Sample output:

```
C:\Users\Abdurrasheed\Documents\codes\clan
enter 1 to INSERT an element in que
enter 2 to DELETE an element from que
enter 3 to DISPLAY all elements of que
enter 4 to quit

enter an option
1
enter an element to insert
1

enter an option
2
1

enter an option
3
1

queue is empty (UNDERFLOW)

enter an option
1
enter an element to insert
1

enter an option
2
1

enter an option
3
1

enter an option
1
enter an element to insert
1

enter an option
1
enter an element to insert
3

enter an option
1
enter an element to insert
4

enter an option
1
enter an element to insert
5

enter an option
1
enter an element to insert
6

enter an option
1
queue is full (OVERFLOW)

enter an option
3
1 3 4 5 6

enter an option
4
```

10. Practical Related Questions:

1. What is enqueue()?
2. What is dequeue()?
3. What is the use of disp()?

11. Exercise Questions:

1. Define Stack.
2. Tell the Differences between Array & Linked List
3. What is Overflow?
4. What is Underflow?
5. What are the Application of Stack?

PRACTICAL 6(a):

Design, Develop and Implement a C program to do the following using a singly linked list.CO2

a) Stack- In single linked list store the information in the form of nodes .Create nodes using dynamic memory allocation method. All the single linked list operations perform based on Stack operations LIFO(last in first out).

A stack contains a top pointer. Which is “head” of the stack where pushing and popping items happens at the head of the list. first node have null in link field and second node link have first node address in link field and so on and last node address in “top” pointer.

Stack Operations:

1. push() : Insert the element into linked list nothing but which is the top node of Stack.
2. pop() : Return top element from the Stack and move the top pointer to the second node of linked list or Stack.
3. peek(): Return the top element.
4. display(): Print all element of Stack.

1. Practical significance:

Stack is one of the Linear Data Structure. We can insert the elements using push function and delete the elements using pop function. The Variable top is used to maintain the position of the topmost element in the stack. We can do any operations on stack though top variable only.

2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,

PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

Able to implement the stack operation using single linked list.

4. Prerequisites:

- Basic knowledge about problem solving
- Require programming knowledge through C language

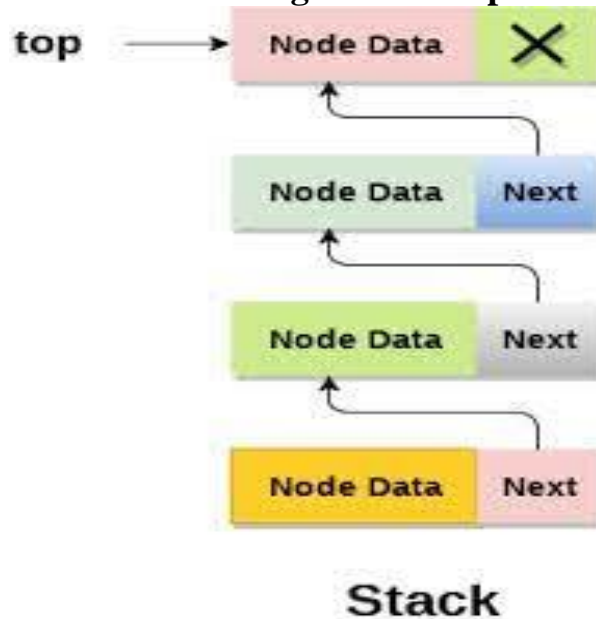
5. Resources required:

S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX /LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

7. Algorithm/circuit/Diagram/Description:



push(value) - Inserting an element into the Stack

We can use the following steps to insert a new node into the stack...

- Step 1 - Create a newNode with given value.
- Step 2 - Check whether stack is Empty ($top == NULL$)
- Step 3 - If it is Empty, then set $newNode \rightarrow next = NULL$.
- Step 4 - If it is Not Empty, then set $newNode \rightarrow next = top$.
- Step 5 - Finally, set $top = newNode$.

pop() - Deleting an Element from a Stack

We can use the following steps to delete a node from the stack...

- Step 1 - Check whether stack is Empty ($top == NULL$).
- Step 2 - If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function
- Step 3 - If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.
- Step 4 - Then set $top = top \rightarrow next$.
- Step 5 - Finally, delete 'temp'. ($free(temp)$).

8. Test cases:

- i)
 - push(11);
 - push(22);
 - push(33);
 - push(44);
- ii)
 - pop();
 - pop();

9. Sample output:

- i) 44->33->22->11->
- ii) 22->11->

10. Practical Related Questions:

1. Define Stack.
2. Tell the Differences between Array & Linked List
3. What is Overflow?
4. What is Underflow?
5. What are the Application of Stack?

11. Exercise Questions:

Implement a C program to find out the Total elements of a stack using Single Linked List.

PRACTICAL 6(b):

Design, Develop and Implement a C program to do the following using a singly linked list

Queue- All the single linked list operations perform based on queue operations FIFO (First in first out). In a Queue data structure, we maintain two pointers, *front* and *rear*. The *front* points the first item of queue and *rear* points to last item.

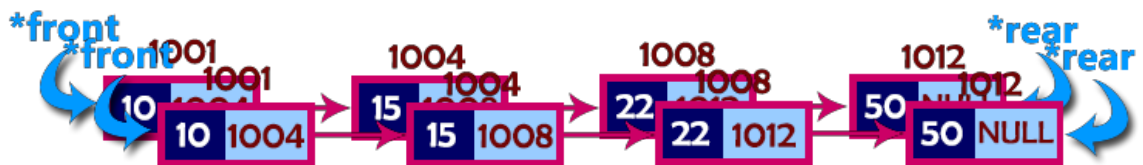
1. **enQueue()** This operation adds a new node after *rear* and moves *rear* to the next node.
2. **deQueue()** This operation removes the front node and moves *front* to the next node.
3. **Display()** Display all elements of the queue.

1. Practical significance:

The major problem with the queue implemented using an array is, It will work for an only fixed number of data values. That means, the amount of data must be specified at the beginning itself. Queue using an array is not suitable when we don't know the size of data which we are going to use.

A queue data structure can be implemented using a linked list data structure. The queue which is implemented using a linked list can work for an unlimited number of values. That means, queue using linked list can work for the variable size of data (No need to fix the size at the beginning of the implementation).

The Queue implemented using linked list can organize as many data values as we want. In linked list implementation of a queue, the last inserted node is always pointed by '**rear**' and the first node is always pointed by '**front**'.



In above example, the last inserted node is 50 and it is pointed by '**rear**' and the first inserted node is 10 and it is pointed by '**front**'. The order of elements inserted is 10, 15, 22 and 50.

Queue Operations using Array

We can Perform the following operations on Queue

1. enQueue()
2. deQueue()
3. Display()

2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,
PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

Able to implement the queue operations using single linked list.

4. Prerequisites:

- Basic knowledge about problem solving

- Require programming knowledge through C language

5. Resources required:

S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX / LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

7. Algorithm/circuit/Diagram/Description:

To implement queue using linked list, we need to set the following things before implementing actual

Step 1 - Include all the header files which are used in the program. And declare all the user defined functions.

Step 2 - Define a 'Node' structure with two members data and next.

Step 3 - Define two Node pointers 'front' and 'rear' and set both to NULL.

Step 4 - Implement the main method by displaying Menu of list of operations and make suitable function calls in the main method to perform user selected operation.

1. enQueue(value) - Inserting an element into the Queue

We can use the following steps to insert a new node into the queue...

Step 1 - Create a newNode with given value and set 'newNode → next' to NULL.

Step 2 - Check whether queue is Empty (rear == NULL)

Step 3 - If it is Empty then, set front = newNode and rear = newNode.

Step 4 - If it is Not Empty then, set rear → next = newNode and rear = newNode.

2. deQueue() - Deleting an Element from Queue

We can use the following steps to delete a node from the queue...

Step 1 - Check whether queue is Empty (front == NULL).

Step 2 - If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function

Step 3 - If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.

Step 4 - Then set 'front = front → next' and delete 'temp' (free(temp)).

3. **Display() - Displaying the elements of Queue**

We can use the following steps to display the elements (nodes) of a queue...

Step 1 - Check whether queue is Empty (front == NULL).

Step 2 - If it is Empty then, display 'Queue is Empty!!!' and terminate the function.

Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with front.

Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until

'temp' reaches to 'rear' (temp → next != NULL). Step 5 - Finally! Display 'temp → data ---> NULL'

8. **Test cases:**

9. **Sample output:**

```

C:\Users\Abdurrasheed\Documents\codes\clan
enter 1 to INSERT an element in que
enter 2 to DELETE an element from que
enter 3 to DISPLAY all elements of que
enter 4 to quit

enter an option
1
enter an element to insert
1

enter an option
2
1

enter an option
2
queue is empty (UNDERFLOW)

enter an option
1
enter an element to insert
1

enter an option
2
1

enter an option
1
enter an element to insert
1

enter an option
1
enter an element to insert
3

enter an option
1
enter an element to insert
4

enter an option
1
enter an element to insert
5

enter an option
1
enter an element to insert
6

enter an option
1
queue is full (OVERFLOW)

enter an option
3
1 3 4 5 6

enter an option
4

```

10. Practical Related Questions:

1. What is enqueue()?
2. What is dequeue()?
3. How to allocate memory for a node?

11. Exercise Questions:

1. How to traverse the queue?
2. From which end the insertion is done?
3. What is difference in queue implementation using array and linked list?

PRACTICAL 7:

Design, Develop and Implement a Program in C for the following CO2

- a) Converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, %(Remainder), ^(Power) and alphanumeric operands.
- b) Evaluation of postfix expression with single digit operands and operators: +, -, *, /, %, ^

1. Practical significance:

There are 3 types of expressions: Infix, Prefix, Postfix .Expression is a combination of Operators & Operands. By using stack we can convert the expression from one to another & Evaluate the expression very easily.

2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,

PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

Able to Convert one expression to another & Evaluate the expression using Stack data structure.

4. Prerequisites:

- i) Stacks
- ii) Infix ,Prefix, Postfix Expressions
- iii) Operator Precedence

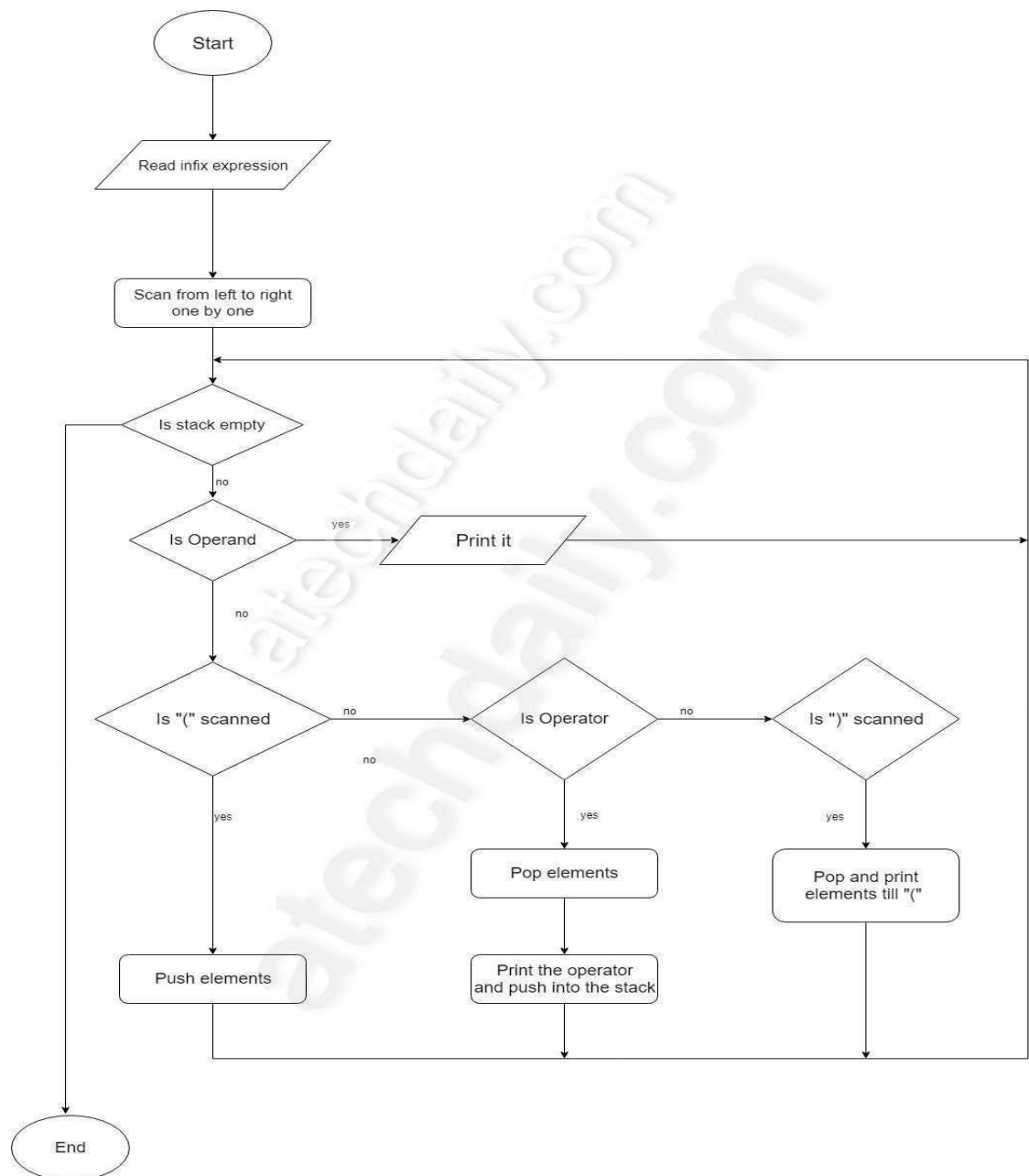
5. Resources required:

S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX /LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

7. Algorithm/circuit/Diagram/Description:



Protected with free version of Watermarkly. Full version doesn't put this mark.

b) Evaluation of Postfix expression

1. While reading the expression from left to right, push the element in the stack if it is an operand.
2. Pop the two operands from the stack, if the element is an operator and then evaluate it.
3. Push back the result of the evaluation. Repeat it till the end of the expression.

8. Test cases:

i) $a+b*(c^d-e)^{(f+g*h)}-i$

ii) $2\ 3\ 1\ * + 9 -$

9. Sample output:

i) $abcd^e-fgh^{*+^{*+i}-}$

ii) -4

10. Practical Related Questions:

1. List out the Types of Expressions.
2. Explain Infix , Prefix and Postfix expressions.
3. Evaluate the given expression .
4. Covert the given expression from Infix to prefix/Postfix

11. Exercise Questions:

1. Develop a C program for conversion of Infix to Prefix
2. Develop a C program for Evaluating a Prefix expression.

PRACTICAL 8(a):

Design, Develop and Implement a menu driven Program in C for the following :

Circular Queue

1. Insert an Element on to Circular QUEUE
2. Delete an Element from Circular QUEUE
3. Demonstrate *Overflow* and *Underflow* situations on Circular QUEUE
4. Display the status of Circular QUEUE
5. Exit

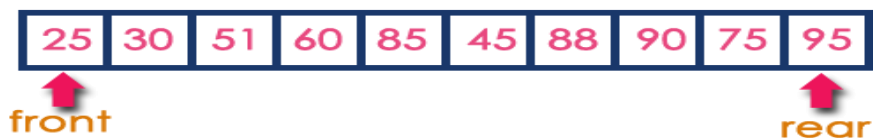
Support the program with appropriate functions for each of the above operations.

1. Practical significance:

In a normal Queue Data Structure, we can insert elements until queue becomes full. But once the queue becomes full, we can not insert the next element until all the elements are deleted from the

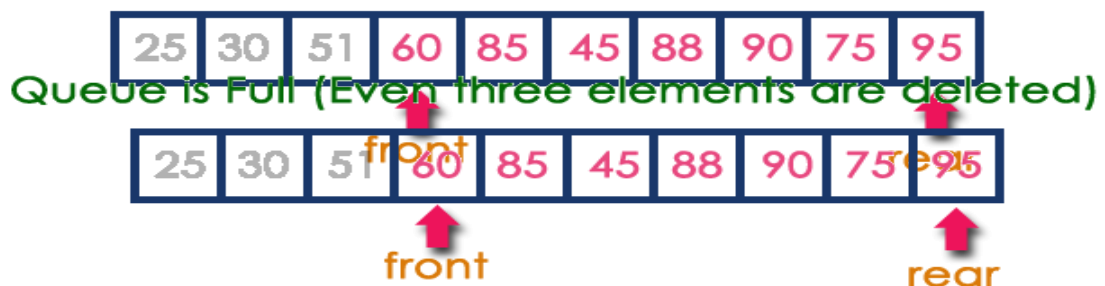
The queue after inserting all the elements into it is as follows...

Queue is Full



Now consider the following situation after deleting three elements from the queue...

Queue is Full (Even three elements are deleted)



This situation also says that Queue is Full and we cannot insert the new element because

'rear' is still at last position. In the above situation, even though we have empty positions in the

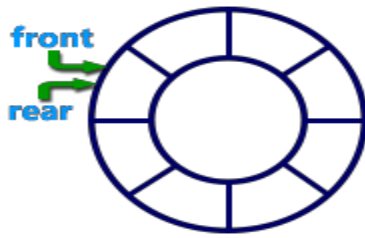
queue we can not make use of them to insert the new element. This is the major problem in a normal queue data structure. To overcome this problem we use a circular queue data structure.

Circular Queue

A Circular Queue can be defined as follows...

A circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle.

Graphical representation of a circular queue is as follows...



2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,
PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

Able to implement circular queue.

4. Prerequisites:

- Basic knowledge about problem solving
- Require programming knowledge through C language

5. Resources required:

S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX /LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

7. Algorithm/circuit/Diagram/Description:

Implementation of Circular Queue

To implement a circular queue data structure using an array, we first perform the following steps before we implement actual operations.

Step 1 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.

Step 2 - Declare all user defined functions used in circular queue implementation.

Step 3 - Create a one dimensional array with above defined SIZE (int cQueue[SIZE])

- Step 4 - Define two integer variables 'front' and 'rear' and initialize both with '-1'. (int front = -1, rear = -1)
- Step 5 - Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on circular queue.

1. enqueue(value) - Inserting value into the Circular Queue

In a circular queue, enqueue() is a function which is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at rear position. The enqueue() function takes one integer value as parameter and inserts that value into the circular queue. We can use the following steps to insert an element into the circular queue...

Step 1 - Check whether queue is FULL.

`((rear == SIZE-1 && front == 0) || (front == rear+1))`

Step 2 - If it is FULL, then display "Queue is FULL!!! Insertion is not possible!!!" and terminate the function.

Step 3 - If it is NOT FULL, then check `rear == SIZE - 1 && front != 0` if it is TRUE, then set `rear = -1`.

Step 4 - Increment rear value by one (`rear++`), set `queue[rear] = value` and check

`'front == -1'` if it is TRUE, then set `front = 0`.

2.deQueue() - Deleting a value from the Circular Queue

In a circular queue, deQueue() is a function used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position. The deQueue() function doesn't take any value as a parameter. We can use the following steps to delete an element from the circular queue...

Step 1 - Check whether queue is EMPTY. (`front == -1 && rear == -1`)

Step 2 - If it is EMPTY, then display "Queue is EMPTY!!! Deletion is not possible!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then display `queue[front]` as deleted element and increment the front value by one (`front ++`). Then check whether `front == SIZE`, if it is TRUE, then set `front = 0`. Then check whether both front –

1 and rear are equal (`front -1 == rear`), if it TRUE, then set both front and rear to '-1' (`front = rear = -1`).

3.Display() - Displays the elements of a Circular Queue

We can use the following steps to display the elements of a circular queue...

Step 1 - Check whether queue is EMPTY. (`front == -1`)

Step 2 - If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then define an integer variable 'i' and set `'i = front'`.

Step 4 - Check whether `'front <= rear'`, if it is TRUE, then display `'queue[i]'` value and
and
increment 'i' value by one (`i++`). Repeat the same until `'i <= rear'` becomes FALSE.

Step 5 - If `'front <= rear'` is FALSE, then display `'queue[i]'` value and increment 'i'

value by one (i++). Repeat the same until 'i <= SIZE - 1' becomes FALSE.
Step 6 - Set i to 0.

Step 7 - Again display 'cQueue[i]' value and increment i value by one (i++).
Repeat the same until 'i <= rear' becomes FALSE.

8. Test cases:

```
Circular Queue:
```

- 1. Insert
- 2. Delete
- 3. Display
- 0. Exit

9. Sample output:

```
Circular Queue:
```

- 1. Insert
- 2. Delete
- 3. Display
- 0. Exit

```
Enter Choice 0-3? : 1
```

```
Enter number: 10
```

```
Circular Queue:
```

- 1. Insert
- 2. Delete
- 3. Display
- 0. Exit

```
Enter Choice 0-3? : 1
```

```
Enter number: 20
```

```
Circular Queue:
```

- 1. Insert
- 2. Delete
- 3. Display
- 0. Exit

```
Enter Choice 0-3? : 2
```

```
10 deleted
```

```
Circular Queue:
```

- 1. Insert
- 2. Delete
- 3. Display
- 0. Exit

```
Enter Choice 0-3? : 1
```

```
Enter number: 60
```

```
Circular Queue:
```

- 1. Insert
- 2. Delete
- 3. Display
- 0. Exit

```
Enter Choice 0-3? : 3
```

```
20 30 40 50 60
```

10. Practical Related Questions:

1. How to increment Front and Rear in circular queue?
2. How to check queue full or not in circular queue?

11.Exercise Questions:

1. Mention the applications of circular queue?
2. What is the drawback of the simple queue?
3. How to overcome simple queue drawback in circular queue?

PRACTICAL 8(b):

Design, Develop and Implement a menu driven Program in C for the following :

Priority Queue

1. Insert an Element on to Priority QUEUE
2. Delete an Element with highest priority from Priority QUEUE
3. Demonstrate *Overflow* and *Underflow* situations on Priority QUEUE
4. Display the status of Priority QUEUE
5. Exit

Support the program with appropriate functions for each of the above operations

1. Practical significance:

Priority queue is a linear data structure. It is having a list of items in which each item has associated priority. It works on a principle add an element to the queue with an associated priority and remove the element from the queue that has the highest priority. In general different items may have different priorities. In this queue highest or the lowest priority item are inserted in random order. It is possible to delete an element from a priority queue in order of their priorities starting with the highest priority.

Types of Priority Queue:

Min Priority Queue: In min priority Queue minimum number of value gets the highest priority and lowest number of element gets the highest priority.

Max Priority Queue: Max priority Queue is the opposite of min priority Queue in it maximum number value gets the highest priority and minimum number of value gets the minimum priority.

Insertion Operation:

- Ask the data from the user.
- If front is equal to 0 and rear is equal to n-1 then Queue is full.
- Else if front is equal to the -1 then queue is empty so we have initialize front and rear with 0.
- Insert the data entered by user in Queue using rear.
- If rear is equal to n-1 and front is not equal to 0 then there is empty space in queue before the front for using that space we will shift the elements of the queue with the help of front and rear.
- Insert the data in the queue before at the position where given priority is greater then priority of the element in queue.

Deletion Operation:

- Remove the element and the priority from the front of the queue.
- Increase front with 1.

Display Operation:

- Using loop take the starting point from the front of the queue and ending point from the rear of the queue and print the data.

2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,
PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

Able to implement Priority Queue.

4. Prerequisites:

- Basic knowledge about problem solving
- Require programming knowledge through C language
- Knowledge about implementation of queue.

5. Resources required:

S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX / LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

7. Algorithm/circuit/Diagram/Description:

Enqueue()

1. IF((Front == 0)&&(Rear == N-1))
2. PRINT "Overflow Condition"
3. Else
4. IF(Front == -1)
5. Front = Rear = 0
6. Queue[Rear] = Data
7. Priority[Rear] = Priority
8. ELSE IF(Rear == N-1)
9. FOR i=Front;i<=Rear;i++)
10. FOR(i=Front;i<=Rear;i++)
11. Q[i-Front] = Q[i]
12. Pr[i-Front] = Pr[i]
13. Rear = Rear-Front
14. Front = 0
15. FOR(i = r;i>f;i-)

16. IF(p>Pr[i])
17. Q[i+1] = Q[i] Pr[i+1] = Pr[i]
18. ELSE
19. Q[i+1] = data Pr[i+1] = p
20. Rear++

Dequeue()

1. IF(Front == -1)
2. PRINT "Queue Under flow condition"
3. ELSE
4. PRINT"Q[f],Pr[f]"
5. IF(Front==Rear)
6. Front = Rear = -1
7. ELSE
8. FRONT++

Print()

1. FOR(i=Front;i<=Rear;i++)
2. PRINT(Q[i],Pr[i])

8. Test cases:

- 1 - Insert
- 2 - Delete
- 3 - Display
- 4 - Exit

9. Sample output:

```

Enter your choice : 1
Enter value to be inserted : 10
Enter your choice : 1
Enter value to be inserted : 20
Enter your choice : 1
Enter value to be inserted : 40
Enter your choice : 1
Enter value to be inserted : 30
Enter your choice : 3
40 30 20 10
Enter your choice : 2
Enter your choice : 3
40 30 20 10
Enter your choice : 2
Enter value to delete : 40
Enter your choice : 3
30 20 10
Enter your choice : 0
Choice is incorrect, Enter a correct choice

```

10. Practical Related Questions:

1. Is the implemented priority queue is min priority or max priority?
2. Are you arranging the data according priority during insertion time or deletion time?

11. Exercise Questions:

1. What are the applications of priority queue?
2. Which data structure can be used to implement priority queue?

PRACTICAL 9:

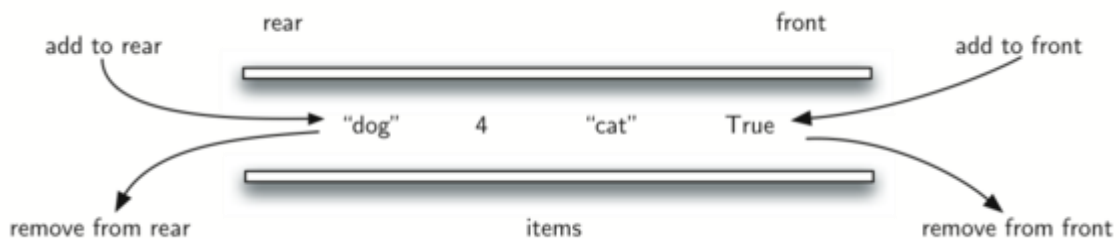
Design, Develop and Implement a menu driven C program to Perform Operations on dequeue (double ended queue) using array.

- a) insertFront(): Adds an item at the front of Deque.
- b) insertRear(): Adds an item at the rear of Deque.
- c) deleteFront(): Deletes an item from front of Deque
- d) deleteRear(): Deletes an item from rear of Deque
- e) getFront(): Gets the front item from queue
- f) getRear(): Gets the last item from queue
- g) isEmpty(): Checks whether Deque is empty or not
- h) isFull(): Checks whether Deque is full or not

Support the program with appropriate functions for each of the above operations

1. Practical significance:

A **deque**, also known as a double-ended queue, is an ordered collection of items similar to the queue. It has two ends, a front and a rear, and the items remain positioned in the collection. What makes a deque different is the unrestrictive nature of adding and removing items. New items can be added at either the front or the rear. Likewise, existing items can be removed from either end. In a sense, this hybrid linear structure provides all the capabilities of stacks and queues in a single data structure.



A deque of Python data objects

The Deque Abstract Data Type

The deque abstract data type is defined by the following structure and operations. A deque is structured, as described above, as an ordered collection of items where items are added and removed from either end, either front or rear. The deque operations are given below.

- Deque() creates a new deque that is empty. It needs no parameters and returns an empty deque.
- add_front(item) adds a new item to the front of the deque. It needs the item and returns nothing.
- add_rear(item) adds a new item to the rear of the deque. It needs the item and returns nothing.
- remove_front() removes the front item from the deque. It needs no parameters and returns the item. The deque is modified.
- remove_rear() removes the rear item from the deque. It needs no parameters and returns the item. The deque is modified.
- is_empty() tests to see whether the deque is empty. It needs no parameters and returns a boolean value.
- size() returns the number of items in the deque. It needs no parameters and returns an integer.

2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,
PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

Able to implement Double ended queue.

4. Prerequisites:

- Basic knowledge about problem solving
- Require programming knowledge through C language

5. Resources required:

S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX /LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

7. Algorithm/circuit/Diagram/Description:

1. Add at Rear:

```
Add_rear( element)
{
    If(rear==max)
        QUEUE FULL
    If(front==0 and rear =0)
    { Inserting First element
      Front=rear=1;
      Queue[rear]=element
    }
    Rear++;
    Queue[rear]=element;
}
```


2. Add at Front:

```
Add_front(element)
{
    If(front==1)
        FRONT FULL
    If(front==0)
        { Inserting First element in queue
          Front=rear=1;
          Queue[front]=element;
        }
    Front --;
    Queue[front]=element;
}
```

3. Delete at Front:

```
delete_front()
{
    If(front==0 and rear==0 )
        QUEUE EMPTY
    If(front==rear)
        {
            Temp = arr[front]
            Front=rear =0
            return temp;
        }
    Temp= queue[front]
    Front++;
    return temp;
}
```

4. Delete at Rear:

```
Delete_rear()
{
    If(rear==0)
        QUEUE EMPTY
    If(front==rear)
        {
            Temp = arr[rear]
            Front=rear =0
            return temp;
        }
    Temp=queue[rear]
    Rear--
    Return temp
}
```

8. Test cases:

1. Insert at the front end
2. Insert at the rear end
3. Delete from front end

4. Delete from rear end

5. Display

6. Quit

9. Sample output:

```
1.Insert at the front end 2.Insert at the rear end 3.Delete from front end 4.Delete from rear end 5.Display 6.Quit
Enter your choice : 1
```

```
Input the element for adding in queue : 10
```

```
front = 0, rear =0
```

```
Queue elements :
10
```

```
1.Insert at the front end 2.Insert at the rear end 3.Delete from front end 4.Delete from rear end 5.Display 6.Quit
Enter your choice : 2
```

```
Input the element for adding in queue : 20
```

```
front = 0, rear =1
```

```
Queue elements :
10 20
```

```
1.Insert at the front end 2.Insert at the rear end 3.Delete from front end 4.Delete from rear end 5.Display 6.Quit
Enter your choice : 1
```

```
Input the element for adding in queue : 30
```

```
front = 6, rear =1
```

```
Queue elements :
30 10 20
```

```
1.Insert at the front end 2.Insert at the rear end 3.Delete from front end 4.Delete from rear end 5.Display 6.Quit
Enter your choice : 2
```

```
Input the element for adding in queue : 40
```

```
front = 6, rear =2
```

```
Queue elements :
30 10 20 40
```

```
1.Insert at the front end 2.Insert at the rear end 3.Delete from front end 4.Delete from rear end 5.Display 6.Quit
Enter your choice : 3
```

```
Element deleted from front end is : 30
```

```
front = 0, rear =2
```

```
Queue elements :
10 20 40
```

```
1.Insert at the front end 2.Insert at the rear end 3.Delete from front end 4.Delete from rear end 5.Display 6.Quit
Enter your choice : 4
```

```
Element deleted from rear end is : 40
```

```
front = 0, rear =1
```

```
Queue elements :
10 20
```

```
1.Insert at the front end 2.Insert at the rear end 3.Delete from front end 4.Delete from rear end 5.Display 6.Quit
Enter your choice : 6
```

10. Practical Related Questions:

1. Explain Add_rear(element) function?
2. Explain Add_front(element) function?
3. Explain Delete_rear() function?
4. Explain Delete_front() function?

11. Exercise Questions:

1. What is deque?
2. What are the applications of deque?

PRACTICAL :

1. Practical significance:

2. Relevant Program Outcomes :

PO 1, PO 2, PO3 - Moderate,

PO4, PO 6, PO 8, PO 9, PO 10, PO 12- Weak

3. Competency and practical skills:

4. Prerequisites:

- Basic knowledge about problem solving
- Require programming knowledge through C language

5. Resources required:

S.No	Name of the Resource	Broad Specification(Approx.)
1	Computer System	Processor – 2GHz RAM – 4GB Hard-Drive Space – 20GB VGA with 1024×768 screen resolution (exact hardware requirement will depend upon the distribution that we choose to work with)
2	Operating System	UNIX /LINUX / UBUNTU
3.	Utilities	GCC Compiler

6. Precautions:

- Check whether the computer is getting proper power or not.
- Ensure the key-board, mouse and monitor are properly working.
- Ensure that there are no power fluctuations while using the system.
- Do not disturb electrical and network connections.

7. Algorithm/circuit/Diagram/Description:

8. Test cases:

9. Sample output:

10. Practical Related Questions:

11. Exercise Questions:

Experiment 10.

Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers.

CO3

- a. Create a BST of N Integers: 13, 3, 4, 12, 14, 10, 5, 1, 8, 2, 7, 9, 11, 6, 18.
- b. Traverse the BST(either inorder, preorder or postorder).
- c. Search the BST for a given element (KEY) and report the appropriate message.
- d. Exit.

1. Practical significance:

- a. The binary search tree is considered as efficient data structure in compare to arrays and linked lists. In searching process, it removes half sub-tree at every step. Searching for an element in a binary search tree takes $O(\log_2 n)$ time. In worst case, the time it takes to search an element is $O(n)$.
- b. It also speed up the insertion and deletion operations as compare to that in array and linked list.
- c. Searching become very efficient in a binary search tree since, we get a hint at each step, about which sub-tree contains the desired element.
- d. Student can perform various operations like insertion, deletion, search, traverse, update and etc.

2. Relevant Program Outcomes – PO- A, B, C, D, H, I.

- a. Student able to know that the significance of Binary Search Tree (BST).
- b. Various operations performed on BST like Create, Insert, Update, Traverse and Delete.
- c. Applications of BST in the real world.

3. Competency and practical skills:

- a. The student will examine the Algorithm and prepare step by step code to solve the problem.
- b. Generating a function which bounds the algorithm's computing time (a priori analysis) and Space.
- c. Using asymptotic notation to determine the order of magnitude of the frequency of execution of statements
- d. Apply mathematical and logical skills to BST Operations in efficient manner.
- e. Practical skills are required like. Programming language efficiency, Error solving Capability, Must know about the Test cases, whether the output of the code is relevant to the Test case.

4. Prerequisites:

- a. Critical thinking.
- b. Computer / Technology Usage.
- c. Analysis required on Algorithm selection and complexity in terms of Time and Space.
- d. Knowledge on Programming Language (C).
- e. Ability to Apply Mathematics relevant to the code logic.

5. Resources required:

During execution of an experiment,

- a. A Computer with good configuration in terms of CPU, Memory and other peripherals.
- b. Operating system preferably open source.
- c. Desired tool installed in the system (GCC, Turbo C).
- d. Lab Manuel, Observation book.

6. Precautions:

- a. Students while entering in to the lab, must know the rules, precautions includes Do's and Don'ts.
- b. Be cautious with Power cables and switches on and off.
- c. Proper turn on and turn off the system to avoid abnormal termination / close.
- d. All the equipment in the lab handled with Care.

7. Algorithm/circuit/Diagram/Description:

Description:

- Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers.
- It is called a binary tree because each tree node has a maximum of two children.
- It is called a search tree because it can be used to search for the presence of a number in $O(\log(n))$ time.
- **Binary Search Tree** is a node-based binary tree data structure which has the following properties:
 - The left subtree of a node contains only nodes with keys lesser than the node's key.
 - The right subtree of a node contains only nodes with keys greater than the node's key.
 - The left and right subtree each must also be a binary search tree.

Algorithm: Create/Insert

1. Create a new BST node and assign values to it.
2. insert(node, key)
 - i) If root == NULL,
return the new node to the calling function.
 - ii) if root->data < key
call the insert function with root->right and assign the return value in root->right.
root->right = insert(root->right,key)
 - iii) if root->data > key
call the insert function with root->left and assign the return value in root->left.
root->left = insert(root->left,key)
3. Finally, return the original root pointer to the calling function.

Algorithm for Pre Order Traversal

- Visit or print the root.
- Traverse the left subtree.
- Traverse the right subtree.

//preorder traversal of binary search tree

```
void preorder(struct node *root)
{
    if(root == NULL)
        return;

    //visit the root
    printf("%d ",root->key);

    //traverse the left subtree
    preorder(root->left);

    //traverse the right subtree
    preorder(root->right);
}
```

Algorithm for In- order Traversal

- Traverse the left subtree.
- Visit or print the root.
- Traverse the right subtree.

```
void inorder(struct node *root)
{
    if(root == NULL)
        return;

    //traverse the left subtree
    inorder(root->left);

    //visit the root
    printf("%d ",root->key);

    //traverse the right subtree
    inorder(root->right);
}
```

Algorithm for Postorder traversal is one of the depth first tree traversal methods.

Postorder : Left - Right – Root

- Traverse the left subtree.
- Traverse the right subtree.
- Visit or print the root.

```
//postorder traversal of binary search tree
void postorder(struct node *root)
{
    if(root == NULL)
        return;

    //traverse the left subtree
    postorder(root->left);

    //traverse the right subtree
    postorder(root->right);

    //visit the root
    printf("%d ",root->key);
}
```

Algorithm for Search Operation:

```
If root == NULL
return NULL;
If number == root->data
return root->data;
If number < root->data
return search(root->left)
If number > root->data
return search(root->right)
```

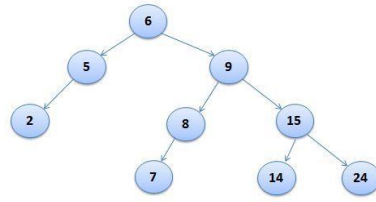
8. Test cases:

- a. Create a BST of N Integers: 13, 3, 4, 12, 14, 10, 5, 1, 8, 2, 7, 9, 11, 6, 18 or 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
- b. Traverse the BST (either in order, preorder or post order).
- c. Search the BST for a given element (10 and 22) and report the appropriate message.

9. Sample output:

- a. Create a BST on the given data.

Final Binary Search Tree
(skip 8, 5, 2 as they are duplicates)



- b. Perform Traversal Techniques on the Tree.
- c. Search an item.

Practical / Viva Questions

1. List 3 Differences between Binary Tree and BST.
2. Give the procedure how to insert a key in BST.
3. Give the procedure how to delete a key from BST.
4. Give the procedure how to traverse a BST.
5. What are the Differences between BST and AVL Trees?

Experiment 11

Design, develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers.

CO3

- a. **Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2.**
- b. **Traverse the BST in Inorder, Preorder and Post Order using non-recursive functions.**
- c. **exit**

2. Practical significance:

- a. The binary search tree is considered as efficient data structure in compare to arrays and linked lists. In searching process, it removes half sub-tree at every step. Searching for an element in a binary search tree takes $O(\log_2 n)$ time. In worst case, the time it takes to search an element is $O(n)$.
- b. It also speed up the insertion and deletion operations as compare to that in array and linked list.
- c. Searching become very efficient in a binary search tree since, we get a hint at each step, about which sub-tree contains the desired element.
- d. Student can perform various operations like insertion, deletion, search, traverse, update and etc.

3. Relevant Program Outcomes – PO- A, B, C, D, H, I.

- a. Student able to know that the significance of Binary Search Tree (BST).
- b. Various operations performed on BST like Create, Insert, Update, Traverse and Delete.
- c. Applications of BST in the real world.

4. Competency and practical skills

- a. The student will examine the Algorithm and prepare step by step code to solve the problem.
- b. Generating a function which bounds the algorithm's computing time (a priori analysis) and Space.
- c. Using asymptotic notation to determine the order of magnitude of the frequency of execution of statements
- d. Apply mathematical and logical skills to BST Operations in efficient manner.
- e. Practical skills are required like. Programming language efficiency, Error solving Capability, Must know about the Test cases, whether the output of the code is relevant to the Test case.

5. Prerequisites

- a. Critical thinking.
- b. Computer / Technology Usage.
- c. Analysis required on Algorithm selection and complexity in terms of Time and Space.
- d. Knowledge on Programming Language (C).
- e. Ability to Apply Mathematics relevant to the code logic.

6. Resources required

During execution of an experiment,

- a. A Computer with good configuration in terms of CPU, Memory and other peripherals.
- b. Operating system preferably open source.
- c. Desired tool installed in the system (GCC, Turbo C).
- d. Lab Manuel, Observation book.

7. Precautions

- a. Students while entering in to the lab, must know the rules, precautions includes Do's and Don'ts.
- b. Be cautious with Power cables and switches on and off.
- c. Proper turn on and turn off the system to avoid abnormal termination / close.
- d. All the equipment in the lab handled with Care.

8. Algorithm/circuit/Diagram/Description:

Create a BST

1. Create a new BST node and assign values to it.
2. Insert (node, key)
 - i) If root == NULL,
 - return the new node to the calling function.
 - ii) if root=>data < key
 - call the insert function with root=>right and assign the return value in root=>right.
 - root->right = insert(root=>right, key)
 - iii) if root=>data > key
 - call the insert function with root->left and assign the return value in root=>left.
 - root=>left = insert(root=>left, key)
3. Finally, return the original root pointer to the calling function.

1. Recursive :- Inorder : Left - Root – Right

- a. Algorithm:
 - i. Traverse the left subtree.
 - ii. Visit or print the root.
 - iii. Traverse the right subtree.

2. Recursive :- Preorder : Root - Left – Right

- a. Algorithm
 - i. Visit or print the root.

- ii. Traverse the left subtree.
- iii. Traverse the right subtree.

3. Recursive :- Postorder : Left - Right – Root

a. Algorithm

- i. Traverse the left subtree.
- ii. Traverse the right subtree.
- iii. Visit or print the root.

Inorder Tree Traversal Non - Recursion:

- 1) Create an empty stack S.
- 2) Initialize current node as root.
- 3) Push the current node to S and set current = current->left until current is NULL.
- 4) If current is NULL and stack is not empty, then
 - a. Pop the top item from stack.
 - b. Print the popped item, set current = popped_item->right
 - c. Go to step 3.
- 5) If current is NULL and stack is empty, then we are done.

Preorder Traversal Non – Recursive:

- 1) Create an empty stack nodeStack and push root node to stack.
- 2) Do following while nodeStack is not empty.
 - a. Pop an item from stack and print it.
 - b. Push right child of popped item to stack.
 - c. Push left child of popped item to stack.

Right child is pushed before left child to make sure that left subtree is processed first.

Postorder Traversal Non – Recursive:

- 1) Create an empty stack.
- 2) Do following while root is not NULL
 - a. Push root's right child and then root to stack.
 - b. Set root as root's left child.
- 3) Pop an item from stack and set it as root.
 - a. If the popped item has a right child and the right child is at top of stack, then remove the right child from stack, push the root back and set root as root's right child.
 - b. Else print root's data and set root as NULL.

Repeat steps 2.1 and 2.2 while stack is not empty.

Operations to be performed on BST:

- a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2.
- b. Traverse the BST in Inorder, Preorder and Post Order using non-recursive functions.
- c. exit.

Practical / Viva Questions:

1. Explain the difference between Recursive and Non-recursive Traversal Techniques performed on BST.
2. Advantage of BST over Binary Tree.
3. Difference between Tree and Graph?
4. What is Degenerated Binary Tree?
5. Applications of Trees.

Experiment 12

Design, Develop and Implement a Program in C for the following operations on Graph(G) of Cities.

CO-4

- 1. Create a Graph of N cities using Adjacency Matrix.**
- 2. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method.**

1. Practical significance:

Adjacency Matrix In graph theory, computer science, an adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph. In the special case of a finite simple graph, the adjacency matrix is a (0, 1)-matrix with zeros on its diagonal.

A graph $G = (V, E)$ where $v = \{0, 1, 2, \dots, n-1\}$ can be represented using two dimensional integer array of size $n \times n$.

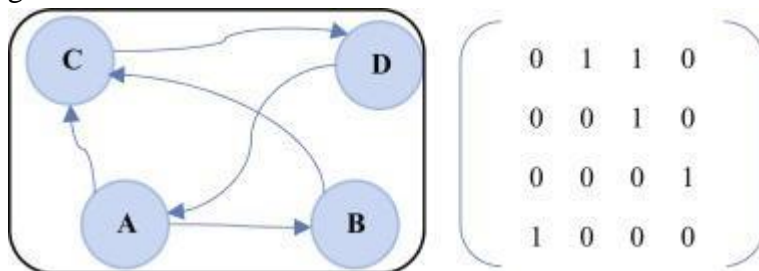
$a[20][20]$ can be used to store a graph with 20 vertices.

$a[i][j] = 1$, indicates presence of edge between two vertices i and j .

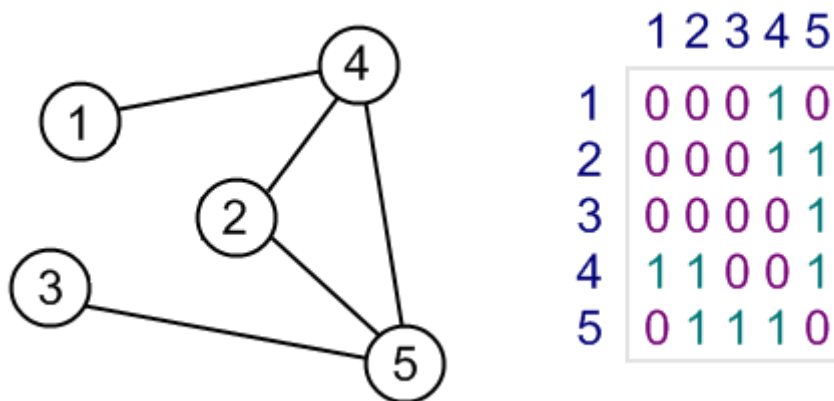
$a[i][j] = 0$, indicates absence of edge between two vertices i and j .

- A graph is represented using square matrix.
- Adjacency matrix of an undirected graph is always a symmetric matrix, i.e. an edge (i, j) implies the edge (j, i) .
- Adjacency matrix of a directed graph is never symmetric, $adj[i][j] = 1$ indicates a directed edge from vertex i to vertex j .

An example of adjacency matrix representation of an undirected and directed graph is given below:



1 .Directed Graph and Its Matrix



2. Undirected Graph

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root and explores the neighbor nodes first, before moving to the next level neighbors. Breadth First Search algorithm(BFS) traverses a graph in a breadthwards motion and uses a queue to remember to get the next vertex to start a search when a dead end occurs in any iteration.

It employs following rules.

- Rule 1 – Visit adjacent unvisited vertex. Mark it visited. Display it. Insert it in a queue.
- Rule 2 – If no adjacent vertex found, remove the first vertex from queue.
- Rule 3 – Repeat Rule 1 and Rule 2 until queue is empty.

DFS Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

Depth-first search, or DFS, is a way to traverse the graph. Initially it allows visiting vertices of the graph only, but there are hundreds of algorithms for graphs, which are based on DFS. Therefore, understanding the principles of depth-first search is quite important to move ahead into the graph theory. The principle of the algorithm is quite simple: to go forward (in depth) while there is such possibility, otherwise to backtrack.

3. Relevant Program Outcomes – PO-A, B, C, D, E, H and I.

4. Competency and practical skills

- a. The student will examine the Algorithm and prepare step by step code to solve the problem.
- b. Generating a function which bounds the algorithm's computing time (a priori analysis) and Space.
- c. Using asymptotic notation to determine the order of magnitude of the frequency of execution of statements
- d. Apply mathematical and logical skills to solve Graph and its operations in efficient manner.
- e. Practical skills are required like. Programming language efficiency, Error solving Capability, Must know about the Test cases, whether the output of the code is relevant to the Test case.

5. Prerequisites

- a. Critical thinking.
- b. Computer / Technology Usage.
- c. Analysis required on Algorithm selection and complexity in terms of Time and Space.
- d. Knowledge on Programming Language (C).

- e. Ability to Apply Mathematics relevant to the code logic.

6. Resources required

During execution of an experiment,

- a. A Computer with good configuration in terms of CPU, Memory and other peripherals.
- b. Operating system preferably open source.
- c. Desired tool installed in the system (GCC, TurboC).
- d. Lab Manuel, Observation book.

7. Precautions

- a. Students while entering in to the lab, must know the rules, precautions includes Do's and Don'ts.
- b. Be cautious with Power cables and switches on and off.
- c. Proper turn on and turn off the system to avoid abnormal termination / close.
- d. All the equipment in the lab handled with Care.

8. Algorithm/circuit/Diagram/Description:

DFS:

Algorithmic Steps

Step 1: Push the root node in the Stack.

Step 2: Loop until stack is empty.

Step 3: Peek the node of the stack.

Step 4: If the node has unvisited child nodes, get the unvisited child node, mark it as traversed and push it on stack.

Step 5: If the node does not have any unvisited child nodes, pop the node from the stack.

DFS Pseudocode: - Non – Recursive:

- 1) DFS-iterative (G, s)
 - a. let S be stack
 - b. S.push(s)
 - c. mark s as visited.
 - d. Print s
 - e. while (S is not empty):
 - i. v = S.top()
 - ii. S.pop()
 - iii. for all neighbours w of v in Graph G:
 - 1. if w is not visited :
 - a. S.push(w).
 - b. mark w as visited

DFS Pseudocode :- Recursive:

- 1) DFS-recursive(G, s):
 - a. mark s as visited
 - b. Print s
 - c. for all neighbours w of s in Graph G:
 - i. if w is not visited:
 - 1. DFS-recursive(G, w)

BFS

Algorithmic Steps

Step 1: Push the root node in the Queue.

Step 2: Loop until the queue is empty.

Step 3: Remove the node from the Queue.

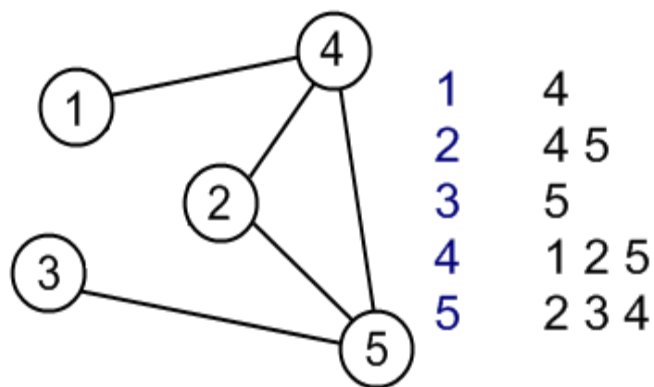
Step 4: If the removed node has unvisited child nodes, mark them

as visited and insert the unvisited children in the queue.

BFS Pseudocode:

- 1) BFS (G, s)
 - a. let Q be queue.
 - b. Q.enqueue(s)
 - c. mark s as visited.
 - d. while (Q is not empty)
 - i. v = Q.dequeue()
 - ii. print v
 - iii. for all neighbours w of v in Graph G
 1. if w is not visited
 - a. Q.enqueue(w).
 - b. Mark w as visited.

9. Sample input and Output:



Graph and Adjacency List

SAMPLE OUTPUT:

1. Create Graph 2.BFS 3.Check graph connected or not (DFS) 4.Exit
Enter your choice: 1
Enter the number of vertices of the digraph: 4
Enter the adjacency matrix of the graph:
0 0 1 1
0 0 0 0
0 1 0 0
0 1 0 0

1. Create Graph 2.BFS 3.Check graph connected or not (DFS) 4.Exit
Enter your choice: 2
Enter the source vertex to find other nodes reachable or not: 1
3
4
2

1. Create Graph 2.BFS 3.Check graph connected or not (DFS) 4.Exit
Enter your choice: 3
Enter the source vertex to find the connectivity: 1
1 -> 3
3 -> 2
1 -> 4
Graph is Connected

1. Create Graph 2.BFS 3.Check graph connected or not (DFS) 4.Exit
Enter your choice: 4

10 .Practical / Viva Questions:

1. What is Transitive Closure of a Graph?
2. What are the Differences between Graph and Tree?
3. What are the Differences between DFS, BFS?
4. What is Panning and Minimum Spanning Tree?
5. What are the Applications of Graphs?
6. What is Adjacency Matrix?

Experiment 13

Design, Develop and Implement a C Program to the problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph using Warshall's Algorithm. The Graph is represented as Adjacency Matrix, and the Matrix denotes the weight of the edges (if it exists) else INF (1e7). CO4

1. Practical significance:

Floyd Warshall Algorithm-

Floyd Warshall Algorithm is a famous algorithm.

- It is used to solve All Pairs Shortest Path Problem.
- It computes the shortest path between every pair of vertices of the given graph.
- Floyd Warshall Algorithm is an example of dynamic programming approach.

Advantages-

Floyd Warshall Algorithm has the following main advantages-

- It is extremely simple.
- It is easy to implement.

Time Complexity-

Floyd Warshall Algorithm consists of three loops over all the nodes.

- The inner most loop consists of only constant complexity operations.
- Hence, the asymptotic complexity of Floyd Warshall algorithm is $O(n^3)$.
- Here, n is the number of nodes in the given graph.

When Floyd Warshall Algorithm Is Used?

Floyd Warshall Algorithm is best suited for dense graphs.

- This is because its complexity depends only on the number of vertices in the given graph.
- For sparse graphs, Johnson's Algorithm is more suitable.

2. Relevant Program Outcomes – PO- A, B, C, D, H, I.

3. Competency and practical skills:

- a. The student will examine the Algorithm and prepare step by step code to solve the problem.
- b. Generating a function which bounds the algorithm's computing time (a priori analysis) and Space.
- c. Using asymptotic notation to determine the order of magnitude of the frequency of execution of statements
- d. Apply mathematical and logical skills to compute the matrix in efficient manner.
- e. Practical skills are required like. Programming language efficiency, Error solving Capability, Must know about the Test cases, whether the output of the code is relevant to the Test case.

4. Prerequisites:

- a. Critical thinking.
- b. Computer / Technology Usage.
- c. Analysis required on Algorithm selection and complexity in terms of Time and Space.
- d. Knowledge on Programming Language (C).
- e. Ability to Apply Mathematics relevant to the code logic.

5. Resources required:

During execution of an experiment,

- a. A Computer with good configuration in terms of CPU, Memory and other peripherals.

- b. Operating system preferably open source.
- c. Desired tool installed in the system (GCC, Turbo C).
- d. Lab Manual, Observation book.

6. Precautions:

- a. Students while entering in to the lab, must know the rules, precautions includes Do's and Don'ts.
- b. Be cautious with Power cables and switches on and off.
- c. Proper turn on and turn off the system to avoid abnormal termination / close.
- d. All the equipment in the lab handled with Care.

7. Algorithm/circuit/Diagram/Description:

```

Create a |V| x |V| matrix           // It represents the distance between every pair of
vertices as given
For each cell (i,j) in M do-
if i == j
M[ i ][ j ] = 0           // For all diagonal elements, value = 0
if (i , j) is an edge in E
M[ i ][ j ] = weight(i,j)   // If there exists a direct edge between the vertices, value =
weight of edge
else
M[ i ][ j ] = infinity      // If there is no direct edge between
the vertices, value = ∞
for k from 1 to |V|
for i from 1 to |V|
for j from 1 to |V|
if M[ i ][ j ] > M[ i ][ k ] + M[ k ][ j ]
M[ i ][ j ] = M[ i ][ k ] + M[ k ][ j ]

```

Or

Floyd – Warshall Algorithm:

- 1) Let V = number of vertices in the graph
- 2) let dist be $V \times V$ matrix of minimum distances initialized to INFINITY.
- 3) for each vertex v
 - a) $\text{dist}[v][v] = 0$
- 4) for each edge (u, v)
 - a) $\text{dist}[u][v] = \text{weight}(u, v)$
- 5) for k from 0 to $|V| - 1$
 - a) for i from 0 to $|V| - 1$
 - i) for j from 0 to $|V| - 1$
 - 1) if $(\text{dist}[i][k] + \text{dist}[k][j])$ is less than $\text{dist}[i][j]$ then
 - a) $\text{dist}[i][j] = \text{dist}[i][k] + \text{dist}[k][j]$

Time Complexity-

- Floyd Warshall Algorithm consists of three loops over all the nodes.
- The inner most loop consists of only constant complexity operations.
- Hence, the asymptotic complexity of Floyd Warshall algorithm is $O(n^3)$.
- Here, n is the number of nodes in the given graph.

When Floyd Warshall Algorithm Is Used?

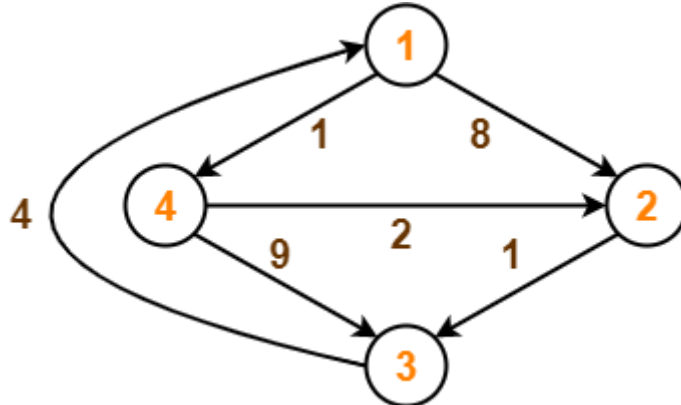
- Floyd Warshall Algorithm is best suited for dense graphs.
- This is because its complexity depends only on the number of vertices in the given graph.
- For sparse graphs, Johnson's Algorithm is more suitable.

8. Sample Input and Output

Floyd Warshall Algorithm-

Problem

Consider the following directed weighted graph-



Using Floyd Warshall Algorithm, find the shortest path distance between every pair of vertices.

Solution-

Step-01:

- Remove all the self loops and parallel edges (keeping the lowest weight edge) from the graph.
- In the given graph, there are neither self edges nor parallel edges.

Step-02:

- Write the initial distance matrix.
- It represents the distance between every pair of vertices in the form of given weights.
- For diagonal elements (representing self-loops), distance value = 0.
- For vertices having a direct edge between them, distance value = weight of that edge.
- For vertices having no direct edge between them, distance value = ∞ .

Initial distance matrix for the given graph is-

$$D_0 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{matrix}$$

Step-03:

Using Floyd Warshall Algorithm, write the following 4 matrices-

$$D_1 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{matrix}$$

$$D_2 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix} \end{matrix}$$

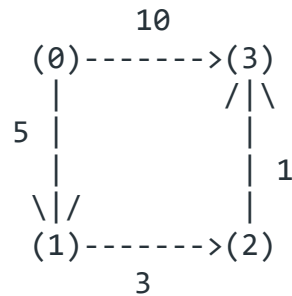
$$D_3 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{matrix}$$

$$D_4 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{matrix}$$

9. Input:

```
graph[][] = { {0, 5, INF, 10},
              {INF, 0, 3, INF},
              {INF, INF, 0, 1},
              {INF, INF, INF, 0} }
```

which represents the following graph



Note that the value of $\text{graph}[i][j]$ is 0 if i is equal to j

And $\text{graph}[i][j]$ is INF (infinite) if there is no edge from vertex i to j .

Output:

Shortest distance matrix

0	5	8	9
INF	0	3	4
INF	INF	0	1
INF	INF	INF	0

And

Input: The first line of input contains an integer **T** denoting the no of test cases. Then **T** test cases follow. The first line of each test case contains an integer **V** denoting the size of the adjacency matrix. The next **V** lines contain **V** space separated values of the matrix (graph). All input will be integer type.

Output: For each test case output will be $V \times V$ space separated integers where the i - j th integer denote the shortest distance of i th vertex from j th vertex. For INT_MAX integers output INF.

Constraints: $1 \leq T \leq 20$

$1 \leq V \leq 100$

$1 \leq \text{graph}[][] \leq 500$

10. Practical / Viva Questions:

1. What is Adjacency Matrix?
2. What is the Time and Space complexity of this Algorithm?
3. Applications of Floyd Warshall Algorithm?
4. What are the Advantages and Disadvantages of this Algorithm?
5. Why this Algorithms works on Directed Graphs?

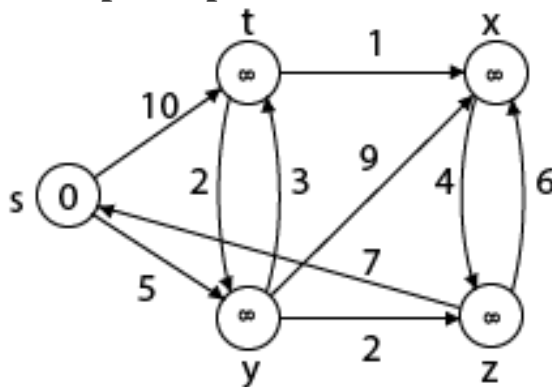
Experiment 14

Design, develop and Implement a C Program to find the shortest distance from A to J on the network below using Dijkstra's Algorithm. CO-4.

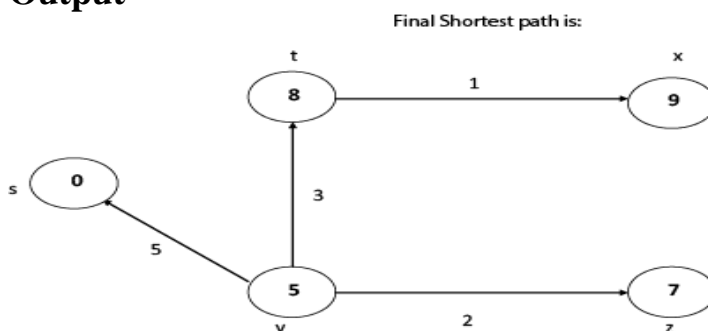
1. Practical significance:

- It is a greedy algorithm that solves the single-source shortest path problem for a directed graph $G = (V, E)$ with nonnegative edge weights, i.e., $w(u, v) \geq 0$ for each edge $(u, v) \in E$.
- Dijkstra's algorithm solves the single-source shortest-paths problem on a directed weighted graph $G = (V, E)$, where all the edges are non-negative (i.e., $w(u, v) \geq 0$ for each edge $(u, v) \in E$).
- In the following algorithm, we will use one function *Extract-Min()*, which extracts the node with the smallest key.
- Dijkstra's Algorithm maintains a set S of vertices whose final shortest - path weights from the source s have already been determined. That's for all vertices $v \in S$; we have $d[v] = \delta(s, v)$. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest - path estimate, inserts u into S and relaxes all edges leaving u .

Example: Input



Output



2. Relevant Program Outcomes – PO- A, B, C, D, H, I.

3. Competency and practical skills:

- The student will examine the Algorithm and prepare step by step code to solve the problem.
- Generating a function which bounds the algorithm's computing time (a priori analysis) and Space.
- Using asymptotic notation to determine the order of magnitude of the frequency of execution of statements
- Apply mathematical and logical skills to compute the matrix in efficient manner.

- Practical skills are required like. Programming language efficiency, Error solving Capability, Must know about the Test cases, whether the output of the code is relevant to the Test case.

4. Prerequisites:

- Critical thinking.
- Computer / Technology Usage.
- Analysis required on Algorithm selection and complexity in terms of Time and Space.
- Knowledge on Programming Language (C).
- Ability to Apply Mathematics relevant to the code logic.

5. Resources required:

During execution of an experiment,

- A Computer with good configuration in terms of CPU, Memory and other peripherals.
- Operating system preferably open source.
- Desired tool installed in the system (GCC, Turbo C).
- Lab Manuel, Observation book.

6. Precautions:

- Students while entering in to the lab, must know the rules, precautions includes Do's and Don'ts.
- Be cautious with Power cables and switches on and off.
- Proper turn on and turn off the system to avoid abnormal termination / close.
- All the equipment in the lab handled with Care.

7. Algorithm/circuit/Diagram/Description:

Algorithm: Dijkstra's-Algorithm (G, w, s)

```

for each vertex  $v \in G.V$ 
     $v.d := \infty$ 
     $v.\Pi := NIL$ 
 $s.d := 0$ 
 $S := \Phi$ 
 $Q := G.V$ 
while  $Q \neq \Phi$ 
     $u := \text{Extract-Min}(Q)$ 
     $S := S \cup \{u\}$ 
    for each vertex  $v \in G.adj[u]$ 
        if  $v.d > u.d + w(u, v)$ 
             $v.d := u.d + w(u, v)$ 
             $v.\Pi := u$ 

```

Analysis

- The complexity of this algorithm is fully dependent on the implementation of Extract-Min function. If extract min function is implemented using linear search, the complexity of this algorithm is $O(V^2 + E)$.
- In this algorithm, if we use min-heap on which *Extract-Min()* function works to return the node from Q with the smallest key, the complexity of this algorithm can be reduced further.

Disadvantage of Dijkstra's Algorithm:

1. It does a blind search, so wastes a lot of time while processing.
2. It can't handle negative edges.
3. It leads to the acyclic graph and most often cannot obtain the right shortest path.
4. We need to keep track of vertices that have been visited.

Or

Dijkstra Algorithm – pseudocode:

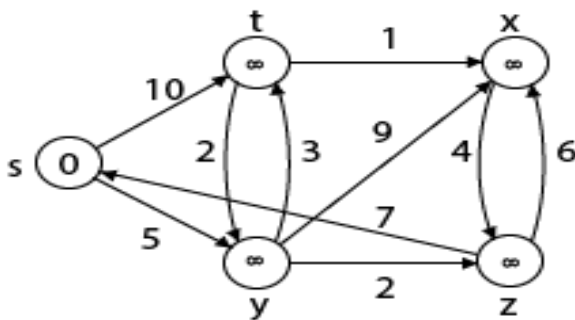
1: function Dijkstra(Graph, source):

```

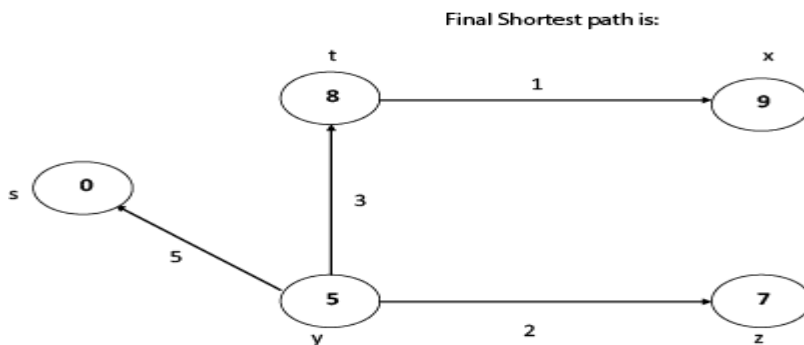
2: for each vertex v in Graph: // Initialization
3:     dist[v] := infinity // initial distance from source to vertex v set infinite
4:     previous[v] := undefined // Previous node in optimal path from source
5: dist[source] := 0 // Distance from source to source
6: Q := the set of all nodes in Graph // all nodes in the graph are unoptimized - are in Q
7: while Q is not empty: // main loop
8:     u := node in Q with smallest dist[ ]
9:     remove u from Q
10:    for each neighbor v of u: // where v has not yet been removed from Q.
11:        alt := dist[u] + dist_between(u, v)
12:        if alt < dist[v] // Relax (u,v)
13:            dist[v] := alt
14:            previous[v] := u
15:    return previous[ ]

```

8. Test Case:



9. Output:



Thus we get all shortest path vertex as

Weight from s to y is 5

Weight from s to z is 7

Weight from s to t is 8

Weight from s to x is 9

These are the shortest distance from the source-'s' in the given graph.

10. Practical / Viva Questions:

1. What are the Differences between Dijkstra and Warshall's Algorithms?
2. What is the Time and Space Complexity of this Algorithm?
3. What are the Applications of this Algorithm?
4. What are the differences between DFS & BFS?

5. What are the Differences between Linear and Non- Linear DS?
6. What are the Diff. between Tree and Graph?