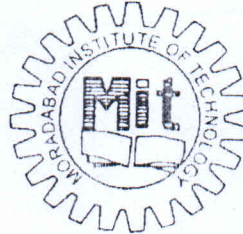


MORADABAD INSTITUTE OF TECHNOLOGY




IN PURSUIT OF EXCELLENCE

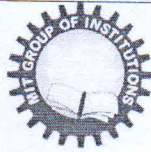
MANUAL OF Data Structure using C Lab (KCS351) Session: 2019-20

Prepared By

Manoj Kr Singh
Assistant Prof.
CSE Dept.
MIT Moradabad

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MORADABAD INSTITUTE OF TECHNOLOGY, MORADABAD -244001


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001



In Pursuit of Excellence

Index

SESSION-2019-2020

SEM- 3rd

| <i>Content</i> | Page No. |
|--|----------|
| Index | 2 |
| Vision & Mission of Institute | 3 |
| Vision & Mission of Department | 4 |
| Program Educational Objectives | 5 |
| Program Outcomes | 6 |
| Program Specific Outcomes | 8 |
| Course Evaluation Scheme | 9 |
| Course Outcomes of Practical | 10 |
| CO -PO Mapping | 11 |
| CO-PSO Mapping | 12 |
| List of Programs as per University | 13 |
| List of Programs with enhancement by the Faculty | 14 |
| Programs Theory and Pseudocode | 15-80 |



In Pursuit of Excellence

Vision & Mission of Institute

SESSION-2019-2020

SEM- 3rd


Vision of Institute

To develop industry ready professionals with values and ethics for global needs.

Mission of Institute

1. To impart education through outcome based pedagogic principles.
2. To provide conducive environment for personality development, training and entrepreneurial skills.
3. To induct high professional ethics and accountability towards society in students.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

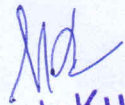
| | | |
|---|---|----------------------|
|  | Vision & Mission of Department | SESSION-2019-2020 |
| | | SEM- 3 rd |

Vision of Department

To develop globally recognized computer science and engineering graduates with ethical values for need of software industries.

Mission of Department

1. To impart knowledge through well-defined instructional objectives in the field of computer science and engineering.
2. To provide learning ambiance for skills, innovation, leadership and overall personality development.
3. To inculcate professional ethics, teamwork and responsiveness towards society.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001



In Pursuit of Excellence

Program Education Objectives

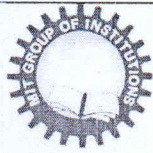
SESSION-2019-2020

SEM- 3rd

Program Education Objectives

- PEO 1:** The graduates will have entrepreneurial and employable skills in software industries, by adapting themselves in the corporate world by utilizing the defined instructional objectives learnt in the program.
- PEO 2:** The graduates will engage in skill enhancement, that would help to work in their own area of interest, individually or in a team.
- PEO 3:** The graduates will demonstrate ownership and responsiveness towards the profession and the society.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001



In Pursuit of Excellence

Program Outcomes

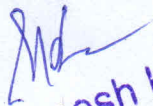
SESSION-2019-2020

SEM- 3rd

Program Outcomes

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communications:** Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001



In Pursuit of Excellence

Program Specific Outcomes


SESSION-2019-2020

SEM- 3rd

After completing their graduation, students of Computer Science and Engineering will be able to do-

PSO 1: Comprehend the core subjects of CSE and apply them to resolve domain specific tribulations.

PSO 2: Extrapolate the fundamental concepts in engineering and to apply latest technology with programming language skills to develop, test, implement and maintain software products.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001



In Pursuit of Excellence

Course Evaluation Scheme

SESSION-2019-2020

SEM- 3rd


AKTU, LUCKNOW, U.P
Study and Evaluation Scheme B.Tech in
Computer Science & Engineering [Effective
from the session 2019-20]

B.TECH (COMPUTER SCIENCE AND ENGINEERING)

SEMESTER- III

| Sl. No. | Subject Codes | Subject | Periods | | | Evaluation Scheme | | | | End Semester | | Total | Credit | |
|---------|----------------------|--|---------|---|---|-------------------|----|-------|----|--------------|-----|-------|--------|---|
| | | | L | T | P | CT | TA | Total | PS | TE | PE | | | |
| 1 | KOE031-38/ KAS302 | Engineering Science - Course/Maths IV | 3 | 1 | 0 | 30 | 20 | 50 | | | 100 | 150 | 4 | |
| 2 | KAS301/ KVE 301 | Technical Communication/Universal Human values | 2 | 1 | 0 | 30 | 20 | 50 | | | 100 | 150 | 3 | |
| | | | 3 | 0 | 0 | | | | | | | | | |
| 3 | KCS301 | Data Structure | 3 | 1 | 0 | 30 | 20 | 50 | | | 100 | 150 | 4 | |
| 4 | KCS302 | Computer Organization and Architecture | 3 | 1 | 0 | 30 | 20 | 50 | | | 100 | 150 | 4 | |
| 5 | KCS303 | Discrete Structures & Theory of Logic | 3 | 0 | 0 | 30 | 20 | 50 | | | 100 | 150 | 3 | |
| 6 | KCS351 | Data Structures Using C Lab | 0 | 0 | 2 | | | | | | 25 | 25 | 50 | 1 |
| 7 | KCS352 | Computer Organization Lab | 0 | 0 | 2 | | | | | | 25 | 25 | 50 | 1 |
| 8 | KCS353 | Discrete Structure & Logic Lab | 0 | 0 | 2 | | | | | | 25 | 25 | 50 | 1 |
| 9 | KCS354 | Mini Project or Internship Assessment* | 0 | 0 | 2 | | | 50 | | | | 50 | 1 | |
| 10 | KNC301/ KNC302 | Computer System Security/Python Programming | 2 | 0 | 0 | 15 | 10 | 25 | | | 50 | | 0 | |
| 11 | | MOOCs (Essential for Honors Degree) | | | | | | | | | | | | |
| | | Total | | | | | | | | | | 950 | 22 | |

Dr. Somesh Kumar
Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

| | | |
|---|------------------------|----------------------|
|  | Course Outcomes | SESSION-2019-2020 |
| | | SEM- 3 rd |

COURSE OUTCOMES

At the end of course, Students will be able to:

| Course Code | CO | Course Outcomes (COs) | Cognitive Levels |
|---------------|-----|---|------------------|
| KCS351 | CO1 | KCS351.1 implement matrix and perform operation like insertion, deletion, searching, traversing by using array | Apply |
| | CO2 | KCS351.2 implement various searching techniques | Apply |
| | CO3 | KCS351.3 implement various sorting techniques using recursive and non -recursive method | Apply |
| | CO4 | KCS351.4 implement linear data structure stack, queue, linked list | Apply |
| | CO5 | KCS351.5 implement non-linear data structures tree and graph | Apply |


Dr. Somesh Kumar
 Prof. & Head, CSE
 Moradabad Institute of Technology
 Moradabad-244001



In Pursuit of Excellence

CO-PO Mapping

SESSION-2019-2020

SEM- 3rd

CO-PO Mapping

| Course Code | CO | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|------------------|--------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|
| KCS351 | CO1 KCS351.1 | 3 | | | | 2 | | | | 1 | 1 | | |
| | CO2 KCS351.2 | 3 | | | | 2 | | | | 1 | 1 | | |
| | CO3 KCS351.3 | 3 | | | | 2 | | | | 1 | 1 | | |
| | CO4 KCS351.4 | 3 | | | | 2 | | | | 1 | 1 | | |
| | CO5 KCS351.5 | 3 | | | | 2 | | | | 1 | 1 | | |
| Mapping Strength | KCS351 | 3 | | | | 2 | | | | 1 | 1 | | |

High: 3 Average: 2 Low: 1


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001



In Pursuit of Excellence


CO-PSO Mapping

SESSION-2019-2020

SEM- 3rd

CO-PSO Mapping

| Course Code | CO | | PSO1 | PSO2 |
|------------------|--------|----------|------|------|
| | | | | |
| KCS351 | CO1 | KCS351.1 | 3 | 3 |
| | CO2 | KCS351.2 | 3 | 3 |
| | CO3 | KCS351.3 | 3 | 3 |
| | CO4 | KCS351.4 | 3 | 3 |
| | CO5 | KCS351.5 | 3 | 3 |
| Mapping Strength | KCS351 | | 3 | 3 |


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001



In Pursuit of Excellence

List of Programs as per University

SESSION-2019-2020

SEM- 3rd

Data Structure using C Lab (KCS351)

Write C Programs to illustrate the concept of the following:

1. Sorting Algorithms-Non-Recursive.
2. Sorting Algorithms-Recursive.
3. Searching Algorithm.
4. Implementation of Stack using Array.
5. Implementation of Queue using Array.
6. Implementation of Circular Queue using Array.
7. Implementation of Stack using Linked List.
8. Implementation of Queue using Linked List.
9. Implementation of Circular Queue using Linked List.
10. Implementation of Tree Structures, Binary Tree, Tree Traversal, Binary Search Tree, insertion and Deletion in BST.
11. Graph Implementation, BFS, DFS, Minimum cost spanning tree, shortest path algorithm.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001



In Pursuit of Excellence


**List of Programs with
enhancement by the Faculty
Member**

SESSION-2019-2020

SEM- 3rd

Data Structure using C Lab (KCS351)

| S. No. | Program Name | Page No. |
|--------|---|----------|
| 1 | Program to implement Matrix Multiplication. (Beyond Syllabus) | 15 |
| 2 | Program to implement Linear Search. | 17 |
| 3 | Program to implement Binary Search. | 19 |
| 4 | Program to implement Bubble Sort. | 22 |
| 5 | Program to implement Insertion Sort. | 24 |
| 6 | Program to implement Selection Sort. | 26 |
| 7 | Program to implement Quick Sort (Recursive). | 29 |
| 8 | Program to implement Merge Sort (Recursive). | 32 |
| 9 | Program to implement Stack using Array. | 35 |
| 10 | Program to implement queue using Array. | 39 |
| 11 | Program to implement circular queue using Array. | 44 |
| 12 | Program to implement Stack using Linked List. | 49 |
| 13 | Program to implement queue using Linked List. | 53 |
| 14 | Program to implement priority queue using Linked List. (Beyond Syllabus) | 58 |
| 15 | Program to implement solution of Tower of Hanoi problem using recursion. (Beyond Syllabus) | 61 |
| 16 | Program to construct Binary Search Tree and its inorder, preorder and postorder traversal. | 63 |
| 17 | Program to implement BFS (Depth First Search) graph traversal algorithm. | 66 |
| 18 | Program to implement DFS (Depth First Search) graph traversal algorithm. | 70 |
| 19 | Program to implement Kuruskal's algorithm to find minimum spanning tree of a given graph. | 74 |
| 20 | WAP to implement warshall's algorithm to find all pair shortest path of a graph. | 78 |


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -1

Program Name: Program to implement Matrix Multiplication.

Theory:

Array plays a very important role in C language. When we store the various numbers of the same data type then we have to use the concept of Array. So, we can say that array is the collection of same data type.

Array is classified into following categories: -

1. Single dimension array
2. Double Dimension Array

Double Dimension Array: - Double Dimension array is used to store the same data type numbers in matrix form. The syntax for defining the single dimensional array is as follows: -


<Data type><array name> [size of row] [size of column];

For example, if we define `int a [3][3];` This means a is an array of 3 Rows and columns. The first element is stored at the position of a [0] [0] and the last element is stored at the position of a [2] [2].

Algorithm for Matrix Multiplication

MatMul (a,b,m,n,p)

1. for(i=1 to m)
2. for(j = 1 to p)
3. `c[i][j] =0;`
4. for(k= 1to n)
5. `c[i][j] = c[i][j]+a[i][k]*b[k][j]`
6. exit


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. What do you mean by an Array?
2. How to create an Array?
3. Advantages and disadvantages of Array?
4. Can we declare array size as a negative number?
5. Is there any difference between `int [] a` and `int a []`?
6. What is the two-dimensional array?



Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -2

Program Name: Program to implement Linear Search.

Theory:

Searching is the process of finding the location of given element in the linear array. The search is said to be successful if the given element is found i.e. the element does exist in the array; otherwise unsuccessful.

There are two approaches to search operation:

- Linear Search
- Binary Search

Linear Search: Given no information about the array a , the only way to search for given element $item$ is to compare $item$ with each element of a one by one. This method, which traverses a sequentially to locate $item$ is called linear search or sequential search.


Algorithm for Linear Search:

Linear (A, ITEM, N, LOC)

1. [INSERT ITEM AT THE END OF THE A] SET $A[N+1]=ITEM$
2. SET $LOC:=1$.
3. [SEARCH FOR ITEM]
 REPEAT WHILE $A[LOC] \neq ITEM$.
 SET $LOC:=LOC+1$.
 [END OF LOOP]
 [SUCCESSFUL?] IF $LOC:=N+1$, THEN SET $LOC:=0$.

Viva Voce Questions

1. Where is linear searching used?
2. What is the best case for linear search?
3. What is the worst case for linear search?
4. What are the various applications of linear search?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -3

Program Name: Program to implement Binary Search.

Theory:

Suppose the elements of the array are sorted in ascending order (if the elements are numbers) or dictionary order (if the elements are strings). The best searching algorithm, called binary search, is used to find the location of the given element. We do use this approach in our daily life. For example, suppose we want to find the meaning of the term modem in a computer dictionary. Obviously, we don't search page by page. We open the dictionary in the middle (roughly) to determine which half contains the term being sought. Then for the subsequent search one half is discarded, and we search in the other half. This process is continued till either we have located the required term or that term is missing from the dictionary, which will be indicated by the fact that at the end we will be left with only one page.

Algorithm for Binary Search

BINARY (DATA, LB, UB, ITEM, LOC)

Here data is a sorted array with lower and upper bounds LB, UB respectively and ITEM is given item of information. The variable BEG, END and MID denotes respectively, the beginning, end and middle location LOC of ITEM in DATA or set LOC:=NULL.

1. [Initialize segment variables.]
Set BEG: =LB, END: =UB, MID: INT ((BEG+END)/2).
2. Repeat step 3 & 4 while BEG<=END and DATA [MID]!=ITEM.
3. IF ITEM < DATA[MID], then
Set END: =MID-1.
ELSE:
Set BEG: =MID+1.
[End of If structure.]
4. Set MID: =INT ((BEG+END)/2).
[End of Step 2 loop.]
5. IF DATA[MID] :=ITEM, then:

Set LOC: = MID.

ELSE:

Set LOC: = NULL.


[End of If structure.]

6. Exit.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. What is Binary Search ?
2. What are the advantages of Binary Search over Linear Search.
3. Explain why complexity of Binary Search is $O(\log n)$?
4. What is the limitation of binary search.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -4

Program Name: Program to implement Bubble Sort.

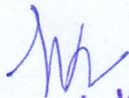
Theory:

In Bubble sort, each element of the array is compared with its adjacent element. The algorithm processes the list in passes. A list with n elements requires $n-1$ passes for sorting. Consider an array A of n elements whose elements are to be sorted by using Bubble sort. The algorithm processes like following.

1. In Pass 1, $A[0]$ is compared with $A[1]$, $A[1]$ is compared with $A[2]$, $A[2]$ is compared with $A[3]$ and so on. At the end of pass 1, the largest element of the list is placed at the highest index of the list.
2. In Pass 2, $A[0]$ is compared with $A[1]$, $A[1]$ is compared with $A[2]$ and so on. At the end of Pass 2 the second largest element of the list is placed at the second highest index of the list.
3. In pass $n-1$, $A[0]$ is compared with $A[1]$, $A[1]$ is compared with $A[2]$ and so on. At the end of this pass. The smallest element of the list is placed at the first index of the list.


Algorithm or Pseudocode of Bubble Sort

- **Step 1:** Repeat Step 2 For $i = 0$ to $N-1$
- **Step 2:** Repeat For $J = i + 1$ to $N - 1$
- **Step 3:** IF $A[J] > A[i]$
SWAP $A[J]$ and $A[i]$
[END OF INNER LOOP]
[END OF OUTER LOOP]
- **Step 4:** EXIT


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. What is the best case complexity of bubble sort?
2. What is the average case complexity of bubble sort?
3. What is the worst case complexity of bubble sort?
4. What are number of passes required by bubble sort?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -5

Program Name: Program to implement Insertion Sort.

Theory:

Insertion sort is the simple sorting algorithm which is commonly used in the daily lives while ordering a deck of cards. In this algorithm, we insert each element onto its proper place in the sorted array. This is less efficient than the other sort algorithms like quick sort, merge sort, etc.

Consider an array A whose elements are to be sorted. Initially, A[0] is the only element on the sorted set. In pass 1, A[1] is placed at its proper index in the array.

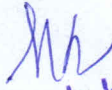
In pass 2, A[2] is placed at its proper index in the array. Likewise, in pass n-1, A[n-1] is placed at its proper index into the array.

To insert an element A[k] to its proper index, we must compare it with all other elements i.e. A[k-1], A[k-2], and so on until we find an element A[j] such that, $A[j] \leq A[k]$.

All the elements from A[k-1] to A[j] need to be shifted and A[k] will be moved to A[j+1].


Algorithm or Pseudocode of Insertion Sort

- **Step 1:** Repeat Steps 2 to 5 for K = 1 to N-1
- **Step 2:** SET TEMP = ARR[K]
- **Step 3:** SET J = K - 1
- **Step 4:** Repeat while TEMP <= ARR[J]
SET ARR[J + 1] = ARR[J]
SET J = J - 1
[END OF INNER LOOP]
- **Step 5:** SET ARR[J + 1] = TEMP
[END OF LOOP]
- **Step 6:** EXIT


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. What is the best case complexity of insertion sort?
2. What is the average case complexity of insertion sort?
3. What is the worst case complexity of insertion sort?
4. What are number of passes required by insertion sort?



Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -6

Program Name: Program to implement Selection Sort.

Theory:

In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array.

First, find the smallest element of the array and place it on the first position. Then, find the second smallest element of the array and place it on the second position. The process continues until we get the sorted array.

The array with n elements is sorted by using $n-1$ pass of selection sort algorithm.

- In 1st pass, smallest element of the array is to be found along with its index pos . then, swap $A[0]$ and $A[pos]$. Thus $A[0]$ is sorted, we now have $n-1$ elements which are to be sorted.
- In 2nd pas, position pos of the smallest element present in the sub-array $A[n-1]$ is found. Then, swap, $A[1]$ and $A[pos]$. Thus $A[0]$ and $A[1]$ are sorted, we now left with $n-2$ unsorted elements.
- In $n-1$ th pass, position pos of the smaller element between $A[n-1]$ and $A[n-2]$ is to be found. Then, swap, $A[pos]$ and $A[n-1]$.

Therefore, by following the above explained process, the elements $A[0]$, $A[1]$, $A[2]$,....., $A[n-1]$ are sorted.

Algorithm or Pseudocode of Selection Sort

SELECTION SORT(ARR, N)

- **Step 1:** Repeat Steps 2 and 3 for $K = 1$ to $N-1$
- **Step 2:** CALL SMALLEST(ARR, K, N, POS)

- **Step 3:** SWAP A[K] with ARR[POS]
[END OF LOOP]
- **Step 4:** EXIT

SMALLEST (ARR, K, N, POS)

- **Step 1:** [INITIALIZE] SET SMALL = ARR[K]
- **Step 2:** [INITIALIZE] SET POS = K
- **Step 3:** Repeat for J = K+1 to N -1
IF SMALL > ARR[J]
SET SMALL = ARR[J]
SET POS = J
[END OF IF]
[END OF LOOP]
- **Step 4:** RETURN POS


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. What is the best case complexity of selection sort?
2. What is the average case complexity of selection sort?
3. What is the worst case complexity of selection sort?
4. What are number of passes required by selection sort?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -7

Program Name: Program to implement Quick Sort (Recursive).

Theory:

Quick sort is the widely used sorting algorithm that makes $n \log n$ comparisons in average case for sorting of an array of n elements. This algorithm follows divide and conquer approach. The algorithm processes the array in the following way.

1. Set the first index of the array to left and loc variable. Set the last index of the array to right variable. i.e. $left = 0$, $loc = 0$, $end = n - 1$, where n is the length of the array.
2. Start from the right of the array and scan the complete array from right to beginning comparing each element of the array with the element pointed by loc.

Ensure that, $a[loc]$ is less than $a[right]$.

1. If this is the case, then continue with the comparison until right becomes equal to the loc.
2. If $a[loc] > a[right]$, then swap the two values. And go to step 3.
3. Set, $loc = right$
4. Start from element pointed by left and compare each element in its way with the element pointed by the variable loc. Ensure that $a[loc] > a[left]$
5. if this is the case, then continue with the comparison until loc becomes equal to left.
6. $[loc] < a[right]$, then swap the two values and go to step 2.
7. Set, $loc = left$.

Algorithm or Pseudocode of Quicksort

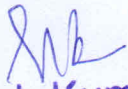
PARTITION (ARR, BEG, END, LOC)

- **Step 1:** [INITIALIZE] SET $LEFT = BEG$, $RIGHT = END$, $LOC = BEG$, $FLAG = 0$
- **Step 2:** Repeat Steps 3 to 6 while $FLAG = 0$
- **Step 3:** Repeat while $ARR[LOC] \leq ARR[RIGHT]$
AND $LOC \neq RIGHT$

- SET RIGHT = RIGHT - 1
 [END OF LOOP]
- **Step 4:** IF LOC = RIGHT
 SET FLAG = 1
 ELSE IF ARR[LOC] > ARR[RIGHT]
 SWAP ARR[LOC] with ARR[RIGHT]
 SET LOC = RIGHT
 [END OF IF]
 - **Step 5:** IF FLAG = 0
 Repeat while ARR[LOC] >= ARR[LEFT] AND LOC != LEFT
 SET LEFT = LEFT + 1
 [END OF LOOP]
 - **Step 6:** IF LOC = LEFT
 SET FLAG = 1
 ELSE IF ARR[LOC] < ARR[LEFT]
 SWAP ARR[LOC] with ARR[LEFT]
 SET LOC = LEFT
 [END OF IF]
 [END OF IF]
 - **Step 7:** [END OF LOOP]
 - **Step 8:** END

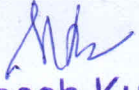
QUICK_SORT (ARR, BEG, END)

- **Step 1:** IF (BEG < END)
 CALL PARTITION (ARR, BEG, END, LOC)
 CALL QUICKSORT(ARR, BEG, LOC - 1)
 CALL QUICKSORT(ARR, LOC + 1, END)
 [END OF IF]
- **Step 2:** END


Dr. Somesh Kumar
 Prof. & Head, CSE
 Moradabad Institute of Technology
 Moradabad-244001

Viva Voce Questions

1. What is the best case complexity of quick sort?
2. What is the average case complexity of quick sort?
3. What is the worst case complexity of quick sort?
4. Is Quicksort faster than bubble sort?



Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -8

Program Name: Program to implement Merge Sort (Recursive).

Theory:

Merge sort is the algorithm which follows divide and conquer approach. Consider an array A of n number of elements. The algorithm processes the elements in 3 steps.

1. If A Contains 0 or 1 elements then it is already sorted, otherwise, Divide A into two sub-array of equal number of elements.
2. Conquer means sort the two sub-arrays recursively using the merge sort.
3. Combine the sub-arrays to form a single final sorted array maintaining the ordering of the array.

The main idea behind merge sort is that, the short list takes less time to be sorted.

Algorithm or Pseudocode of Mergesort

- **Step 1:** [INITIALIZE] SET I = BEG, J = MID + 1, INDEX = 0
- **Step 2:** Repeat while (I <= MID) AND (J <= END)
 - IF ARR[I] < ARR[J]
 - SET TEMP[INDEX] = ARR[I]
 - SET I = I + 1
 - ELSE
 - SET TEMP[INDEX] = ARR[J]
 - SET J = J + 1
 - [END OF IF]
 - SET INDEX = INDEX + 1
 - [END OF LOOP]
- Step 3: [Copy the remaining elements of right sub-array, if any]
 - IF I > MID
 - Repeat while J <= END
 - SET TEMP[INDEX] = ARR[J]

SET INDEX = INDEX + 1, SET J = J + 1

[END OF LOOP]

[Copy the remaining elements of
left sub-array, if any]

ELSE

Repeat while I <= MID

SET TEMP[INDEX] = ARR[I]

SET INDEX = INDEX + 1, SET I = I + 1

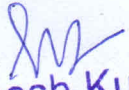
[END OF LOOP]

[END OF IF]

- **Step 4:** [Copy the contents of TEMP back to ARR] SET K = 0
- **Step 5:** Repeat while K < INDEX
SET ARR[K] = TEMP[K]
SET K = K + 1
[END OF LOOP]
- **Step 6:** Exit

MERGE_SORT(ARR, BEG, END)

- **Step 1:** IF BEG < END
SET MID = (BEG + END)/2
CALL MERGE_SORT (ARR, BEG, MID)
CALL MERGE_SORT (ARR, MID + 1, END)
MERGE (ARR, BEG, MID, END)
[END OF IF]
- **Step 2:** END


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. What is the average case complexity for merge sort algorithm?
2. What is the worst case complexity for merge sort algorithm?
3. What is the best case complexity for merge sort algorithm?
4. What are the total numbers of passes in merge sort of 'n' numbers?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -9

Program Name: Program to implement Stack using Array.

Theory:

A Stack is a linear data structure that follows the **LIFO (Last-In-First-Out)** principle. Stack has one end, whereas the Queue has two ends (**front and rear**). It contains only one pointer **top pointer** pointing to the topmost element of the stack. Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack. In other words, *a stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.*

Key points related to stack

- It is called as stack because it behaves like a real-world stack, piles of books, etc.
- A Stack is an abstract data type with a pre-defined capacity, which means that it can store the elements of a limited size.
- It is a data structure that follows some order to insert and delete the elements, and that order can be LIFO or FILO.

Standard Stack Operations

The following are some common operations implemented on the stack:

- **push():** When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
- **pop():** When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.
- **isEmpty():** It determines whether the stack is empty or not.
- **isFull():** It determines whether the stack is full or not.'
- **peek():** It returns the element at the given position.
- **count():** It returns the total number of elements available in a stack.
- **change():** It changes the element at the given position.

- **display():** It prints all the elements available in the stack.

Array implementation of Stack

In array implementation, the stack is formed by using the array. All the operations regarding the stack are performed using arrays.

Adding an element onto the stack (push operation)

Adding an element into the top of the stack is referred to as push operation. Push operation involves following two steps.

1. Increment the variable Top so that it can now refer to the next memory location.
2. Add element at the position of incremented top. This is referred to as adding new element at the top of the stack.

Stack is overflown when we try to insert an element into a completely filled stack therefore, our main function must always avoid stack overflow condition.

Algorithm or pseudocode for push operation

```
begin
  if top = n then stack full
    top = top + 1
    stack (top): = item;
  end
```

Deletion of an element from a stack (Pop operation)

Deletion of an element from the top of the stack is called pop operation. The value of the variable top will be incremented by 1 whenever an item is deleted from the stack. The top most element of the stack is stored in an another variable and then the top is decremented by 1. the operation returns the deleted value that was stored in another variable as the result. The underflow condition occurs when we try to delete an element from an already empty stack.

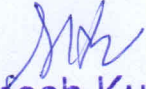
Algorithm or pseudocode for pop operation

```
begin  
if top = -1 then stack empty;  
item = stack(top);  
top = top - 1;  
end;
```


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. What is Stack and where it can be used?
2. What is the condition of full in stack?
3. What are various operation on stack?
4. What are the applications of stack?
5. Which type of data structure is stack?
6. What are overflow and underflow conditions in stack?

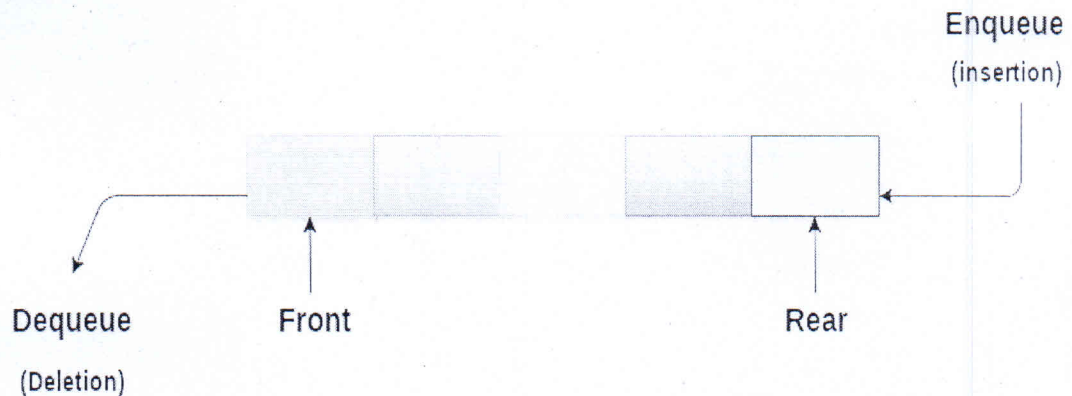

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -10

Program Name: Program to implement queue using Array.

Theory:

1. A queue can be defined as an ordered list which enables insert operations to be performed at one end called **REAR** and delete operations to be performed at another end called **FRONT**.
2. Queue is referred to be as First In First Out list.
3. For example, people waiting in line for a rail ticket form a queue.



Applications of Queue

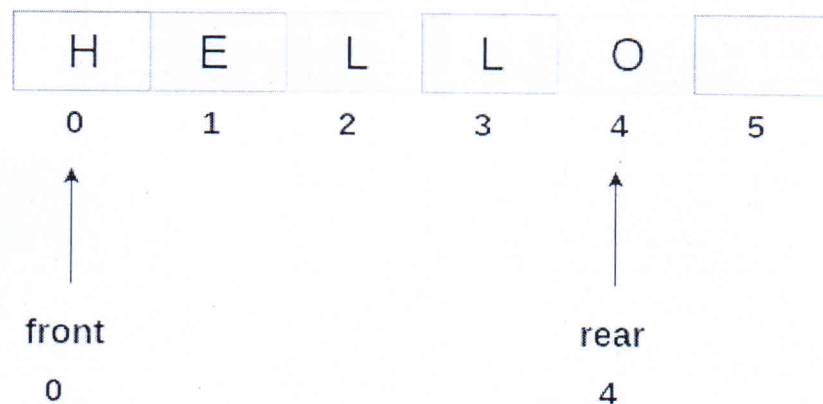
Due to the fact that queue performs actions on first in first out basis which is quite fair for the ordering of actions. There are various applications of queues discussed as below.

1. Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.

2. Queues are used in asynchronous transfer of data (where data is not being transferred at the same rate between two processes) for eg. pipes, file IO, sockets.
3. Queues are used as buffers in most of the applications like MP3 media player, CD player, etc.
4. Queue are used to maintain the play list in media players in order to add and remove the songs from the play-list.
5. Queues are used in operating systems for handling interrupts.

Array representation of Queue

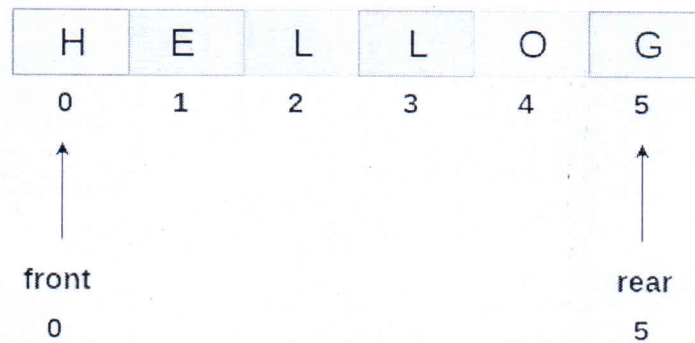
We can easily represent queue by using linear arrays. There are two variables i.e. front and rear, that are implemented in the case of every queue. Front and rear variables point to the position from where insertions and deletions are performed in a queue. Initially, the value of front and queue is -1 which represents an empty queue. Array representation of a queue containing 5 elements along with the respective values of front and rear, is shown in the following figure.



Queue

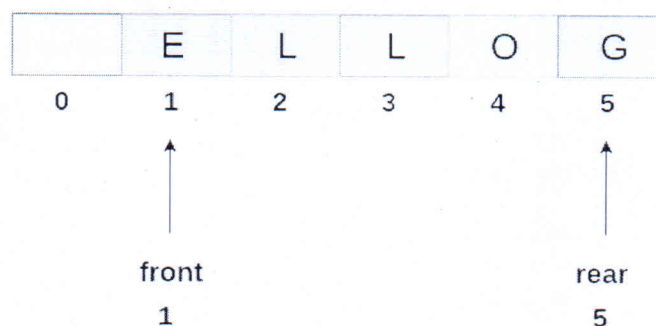
The above figure shows the queue of characters forming the English word "HELLO". Since, No deletion is performed in the queue till now, therefore the value of front remains -1. However, the value of rear increases by one every time an insertion is performed in the queue. After inserting

an element into the queue shown in the above figure, the queue will look something like following. The value of rear will become 5 while the value of front remains same.



Queue after inserting an element


After deleting an element, the value of front will increase from -1 to 0. however, the queue will look something like following.



Queue after deleting an element


Algorithm or pseudocode to insert any element in a queue

- **Step 1:** IF REAR = MAX - 1
Write OVERFLOW
Go to step
[END OF IF]
- **Step 2:** IF FRONT = -1 and REAR = -1
SET FRONT = REAR = 0
ELSE
SET REAR = REAR + 1
[END OF IF]
- **Step 3:** Set QUEUE[REAR] = NUM
- **Step 4:** EXIT


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. What is Queue?
2. List some Queue real-life applications
3. What are some types of Queue?
4. What is the condition of full and empty if queue is implemented using array?
5. What is the limitation of simple queue?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

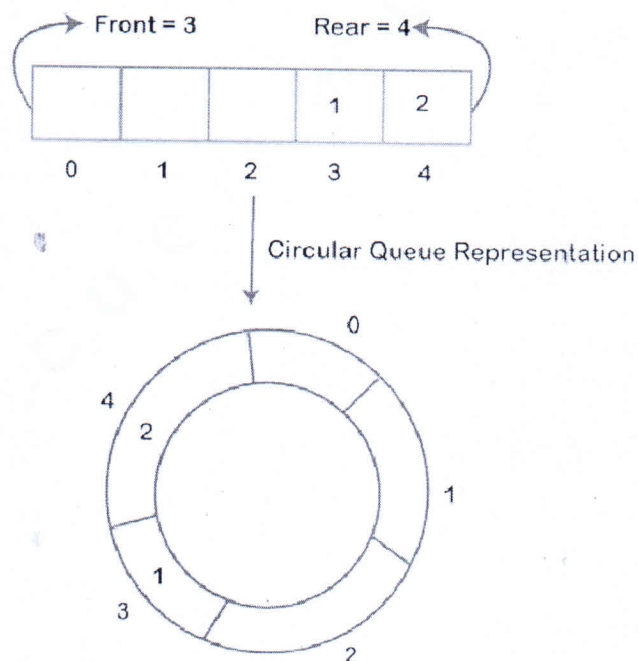
Program No. -11

Program Name: Program to implement circular queue using Array.

Theory:

A circular queue is similar to a linear queue as it is also based on the FIFO (First In First Out) principle except that the last position is connected to the first position in a circular queue that forms a circle. It is also known as a *Ring Buffer*.

There was one limitation in the array implementation of Queue. If the rear reaches to the end position of the Queue then there might be possibility that some vacant spaces are left in the beginning which cannot be utilized. So, to overcome such limitations, the concept of the circular queue was introduced.



As we can see in the above image, the rear is at the last position of the Queue and front is pointing somewhere rather than the 0th position. In the above array, there are only two elements and other three positions are empty. The rear is at the last position of the Queue; if we try to insert the element then it will show that there are no empty spaces in the Queue.

There is one solution to avoid such wastage of memory space by shifting both the elements at the left and adjust the front and rear end accordingly. It is not a practically good approach because shifting all the elements will consume lots of time. The efficient approach to avoid the wastage of the memory is to use the circular queue data structure.

Operations on Circular Queue

The following are the operations that can be performed on a circular queue:

- **Front:** It is used to get the front element from the Queue.
- **Rear:** It is used to get the rear element from the Queue.
- **enQueue(value):** This function is used to insert the new value in the Queue. The new element is always inserted from the rear end.
- **deQueue():** This function deletes an element from the Queue. The deletion in a Queue always takes place from the front end.

Applications of Circular Queue

The circular Queue can be used in the following scenarios:

- **Memory management:** The circular queue provides memory management. As we have already seen that in linear queue, the memory is not managed very efficiently. But in case of a circular queue, the memory is managed efficiently by placing the elements in a location which is unused.
- **CPU Scheduling:** The operating system also uses the circular queue to insert the processes and then execute them.
- **Traffic system:** In a computer-control traffic system, traffic light is one of the best examples of the circular queue. Each light of traffic light gets ON one by one after every jinterval of time. Like red light gets ON for one minute then yellow light for one minute and then green light. After green light, the red light gets ON.

Enqueue operation

The steps of enqueue operation are given below:

- First, we will check whether the Queue is full or not.
- Initially the front and rear are set to -1. When we insert the first element in a Queue, front and rear both are set to 0.
- When we insert a new element, the rear gets incremented, i.e., $rear=rear+1$.

Scenarios for inserting an element

There are two scenarios in which queue is not full:

- If $rear \neq \text{max} - 1$, then rear will be incremented to mod (maxsize) and the new value will be inserted at the rear end of the queue.
- If $front \neq 0$ and $rear = \text{max} - 1$, it means that queue is not full, then set the value of rear to 0 and insert the new element there.

There are two cases in which the element cannot be inserted:

- When $front == 0$ && $rear = \text{max}-1$, which means that front is at the first position of the Queue and rear is at the last position of the Queue.
- $front == rear + 1$;

Algorithm to insert an element in a circular queue

```
Step 1: IF (REAR+1)%MAX = FRONT
Write " OVERFLOW "
Goto step 4
[End OF IF]
```

```
Step 2: IF FRONT = -1 and REAR = -1
SET FRONT = REAR = 0
ELSE IF REAR = MAX - 1 and FRONT != 0
SET REAR = 0
ELSE
SET REAR = (REAR + 1) % MAX
[END OF IF]
```

Step 3: SET QUEUE[REAR] = VAL

Step 4: EXIT

Dequeue Operation

The steps of dequeue operation are given below:

- First, we check whether the Queue is empty or not. If the queue is empty, we cannot perform the dequeue operation.
- When the element is deleted, the value of front gets decremented by 1.
- If there is only one element left which is to be deleted, then the front and rear are reset to -1.

Algorithm to delete an element from the circular queue

Step 1: IF FRONT = -1
Write " UNDERFLOW "
Goto Step 4
[END of IF]

Step 2: SET VAL = QUEUE[FRONT]


Step 3: IF FRONT = REAR
SET FRONT = REAR = -1
ELSE
IF FRONT = MAX -1
SET FRONT = 0
ELSE
SET FRONT = FRONT + 1
[END of IF]
[END OF IF]

Step 4: EXIT


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. What are benefits of Circular Queue over simple queue?
2. What is the condition of full and empty if circular queue is implemented using array?
3. What are the applications of circular queue?
4. How does circular queue work?

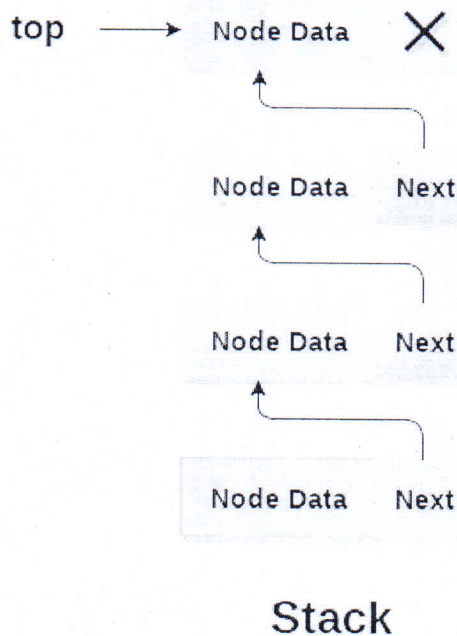

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -12

Program Name: Program to implement Stack using Linked List.

Theory: Instead of using array, we can also use linked list to implement stack. Linked list allocates the memory dynamically. However, time complexity in both the scenario is same for all the operations i.e. push, pop and peek.

In linked list implementation of stack, the nodes are maintained non-contiguously in the memory. Each node contains a pointer to its immediate successor node in the stack. Stack is said to be overflowed if the space left in the memory heap is not enough to create a node.

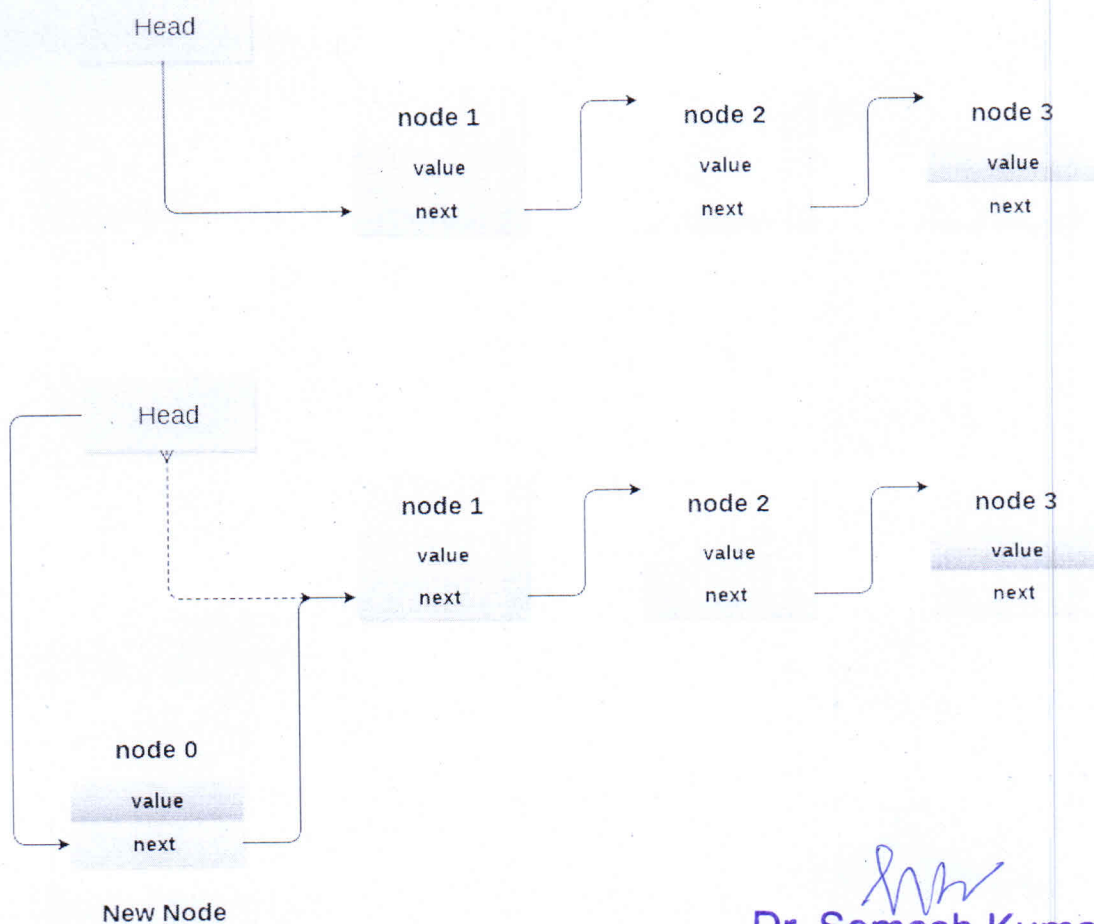


The top most node in the stack always contains null in its address field. Let's discuss the way in which, each operation is performed in linked list implementation of stack.

Adding a node to the stack (Push operation)

Adding a node to the stack is referred to as **push** operation. Pushing an element to a stack in linked list implementation is different from that of an array implementation. In order to push an element onto the stack, the following steps are involved.

1. Create a node first and allocate memory to it.
2. If the list is empty then the item is to be pushed as the start node of the list. This includes assigning value to the data part of the node and assign null to the address part of the node.
3. If there are some nodes in the list already, then we have to add the new element in the beginning of the list (to not violate the property of the stack). For this purpose, assign the address of the starting element to the address field of the new node and make the new node, the starting node of the list.



Deleting a node from the stack (POP operation)

Deleting a node from the top of stack is referred to as **pop** operation. Deleting a node from the linked list implementation of stack is different from that in the array implementation. In order to pop an element from the stack, we need to follow the following steps :

1. **Check for the underflow condition:** The underflow condition occurs when we try to pop from an already empty stack. The stack will be empty if the head pointer of the list points to null.
2. **Adjust the head pointer accordingly:** In stack, the elements are popped only from one end, therefore, the value stored in the head pointer must be deleted and the node must be freed. The next node of the head node now becomes the head node.

Display the nodes (Traversing)


Displaying all the nodes of a stack needs traversing all the nodes of the linked list organized in the form of stack. For this purpose, we need to follow the following steps.

1. Copy the head pointer into a temporary pointer.
2. Move the temporary pointer through all the nodes of the list and print the value field attached to every node.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. How stack is implemented using Linked List?
2. Compare Array based vs Linked List stack implementations.
3. What is the condition for overflow and underflow in linked list implementation of stack?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -13

Program Name: Program to implement queue using Linked List.

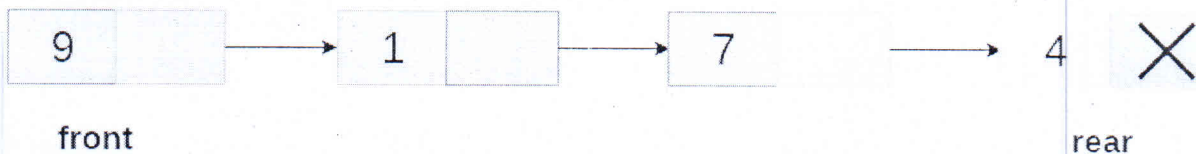
Theory: The array implementation of queue can not be used for the large scale applications where the queues are implemented. One of the alternative of array implementation is linked list implementation of queue.

In a linked queue, each node of the queue consists of two parts i.e. data part and the link part. Each element of the queue points to its immediate next element in the memory.

In the linked queue, there are two pointers maintained in the memory i.e. front pointer and rear pointer. The front pointer contains the address of the starting element of the queue while the rear pointer contains the address of the last element of the queue.

Insertion and deletions are performed at rear and front end respectively. If front and rear both are NULL, it indicates that the queue is empty.

The linked representation of queue is shown in the following figure.



Linked Queue

Operation on Linked Queue

There are two basic operations which can be implemented on the linked queues. The operations are Insertion and Deletion.

Insert operation

The insert operation appends the queue by adding an element to the end of the queue. The new element will be the last element of the queue.

Firstly, allocate the memory for the new node ptr by using the following statement.

```
Ptr = (struct node *) malloc (sizeof(struct node));
```

There can be two scenarios of inserting this new node ptr into the linked queue.

In the first scenario, we insert an element into an empty queue. In this case, the condition, **front == NULL**, becomes true. Now, the new element will be added as the only element of the queue and the next pointer of front and rear pointer both, will point to NULL.

```
1. ptr -> data = item;
2.     if(front == NULL)
3.     {
4.         front = ptr;
5.         rear = ptr;
6.         front -> next = NULL;
7.         rear -> next = NULL;
8.     }
```

In the second case, the queue contains more than one element. The condition **front = NULL** becomes false. In this scenario, we need to update the end pointer rear so that the next pointer of rear will point to the new node ptr. Since, this is a linked queue, hence we also need to make the rear pointer point to the newly added node ptr. We also need to make the next pointer of rear point to NULL.

```
1. rear -> next = ptr;
2.     rear = ptr;
3.     rear->next = NULL;
```

In this way, the element is inserted into the queue.

Algorithm to insert an element from the circular queue

- **Step 1:** Allocate the space for the new node PTR
- **Step 2:** SET PTR -> DATA = VAL
- **Step 3:** IF FRONT = NULL
SET FRONT = REAR = PTR
SET FRONT -> NEXT = REAR -> NEXT = NULL
ELSE
SET REAR -> NEXT = PTR
SET REAR = PTR
SET REAR -> NEXT = NULL
[END OF IF]
- **Step 4:** END

Delete operation


Deletion operation removes the element that is first inserted among all the queue elements. Firstly, we need to check either the list is empty or not. The condition `front == NULL` becomes true if the list is empty, in this case, we simply write underflow on the console and make exit. Otherwise, we will delete the element that is pointed by the pointer front. For this purpose, copy the node pointed by the front pointer into the pointer ptr. Now, shift the front pointer, point to its next node and free the node pointed by the node ptr. This is done by using the following statements.

1. `ptr = front;`
2. `front = front -> next;`
3. `free(ptr);`


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Algorithm to delete an element from the circular queue

- **Step 1:** IF FRONT = NULL
Write " Underflow "
Go to Step 5
[END OF IF]
- **Step 2:** SET PTR = FRONT
- **Step 3:** SET FRONT = FRONT -> NEXT
- **Step 4:** FREE PTR
- **Step 5:** END


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. How queue is implemented using Linked List?
2. Compare Array based vs Linked List queue implementations.
3. What is the condition for full and empty in linked list implementation of queue?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

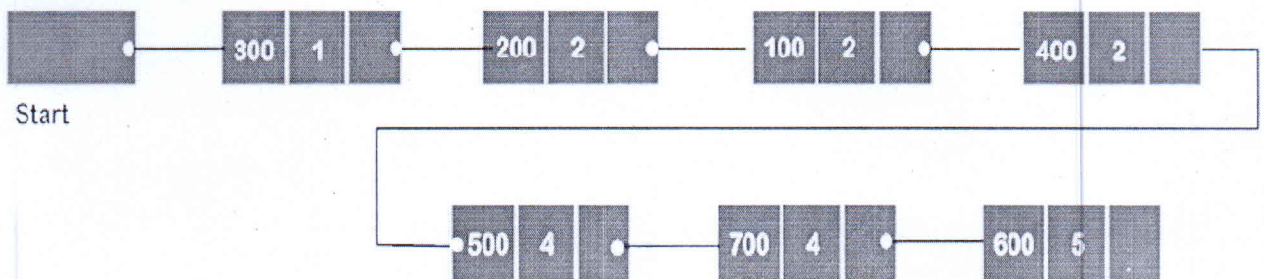
Program No. -14

Program Name: Program to implement priority queue using Linked List.

Theory: A priority queue is an abstract data type that behaves similarly to the normal queue except that each element has some priority, i.e., the element with the highest priority would come first in a priority queue. The priority of the elements in a priority queue will determine the order in which elements are removed from the priority queue.

The priority queue supports only comparable elements, which means that the elements are either arranged in an ascending or descending order.

Example: start is a pointer that contains the address of first node.



Characteristics of a Priority queue

A priority queue is an extension of a queue that contains the following characteristics:

- Every element in a priority queue has some priority associated with it.
- An element with the higher priority will be deleted before the deletion of the lesser priority.
- If two elements in a priority queue have the same priority, they will be arranged using the FIFO principle.

Algorithm to insert an element in priority queue

INSERT (HEAD, DATA, PRIORITY)

Step 1: Create new node with DATA and PRIORITY

Step 2: Check if HEAD has lower priority. If true follow Steps 3-4 and end. Else goto Step 5.

Step 3: NEW -> NEXT = HEAD

Step 4: HEAD = NEW

Step 5: Set TEMP to head of the list

Step 6: While TEMP -> NEXT != NULL and TEMP -> NEXT -> PRIORITY > PRIORITY

Step 7: TEMP = TEMP -> NEXT

[END OF LOOP]

Step 8: NEW -> NEXT = TEMP -> NEXT

Step 9: TEMP -> NEXT = NEW

Step 10: End


Algorithm to delete an element in priority queue

DELETE (HEAD)

Step 1: Set the head of the list to the next node in the list. HEAD = HEAD -> NEXT.


Step 2: Free the node at the head of the list

Step 3: End


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. What is priority queue?
2. What are priority queues used for?
3. What is difference between queue and priority queue?
4. How is data stored in priority queue?
5. What are the applications of priority queue?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -15

Program Name: Program to implement solution of Tower of Hanoi problem using recursion.

Theory: Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

Algorithm or Pseudocode for Tower of Hanoi solution using recursion

START

Procedure Hanoi(disk, source, dest, aux)

IF disk == 1, THEN

 move disk from source to dest

ELSE

 Hanoi(disk - 1, source, aux, dest) // Step 1

 move disk from source to dest // Step 2

 Hanoi(disk - 1, aux, dest, source) // Step 3

END IF


END Procedure

STOP


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. What is Tower of Hanoi problem?
2. What is the complexity of recursive solution of Tower of Hanoi problem?
3. What is the optimal data structure used to solve Tower of Hanoi problem?
4. What is the number of moves required to solve Tower of Hanoi problem for n disks?
5. Can Tower of hanoi problem solved iteratively?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

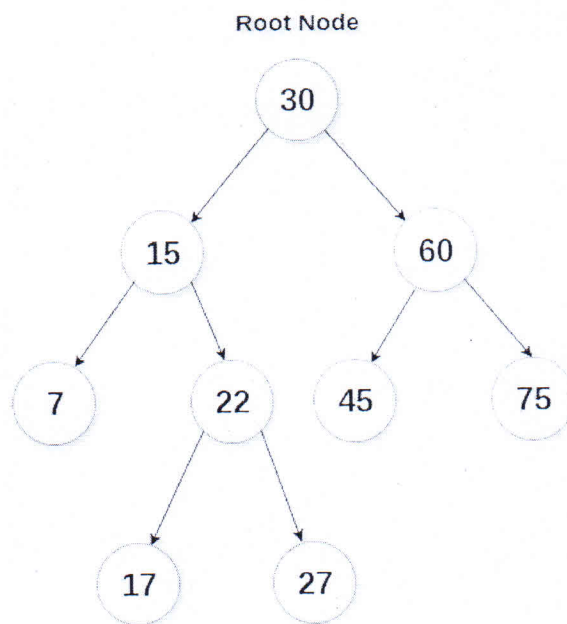
Program No. -16

Program Name: Program to construct Binary Search Tree and its inorder, preorder and postorder traversal.

Theory:

1. Binary Search tree can be defined as a class of binary trees, in which the nodes are arranged in a specific order. This is also called ordered binary tree.
2. In a binary search tree, the value of all the nodes in the left sub-tree is less than the value of the root.
3. Similarly, value of all the nodes in the right sub-tree is greater than or equal to the value of the root.
4. This rule will be recursively applied to all the left and right sub-trees of the root.

Example:



Binary Search Tree

Algorithm or Pseudocode for insertion in binary search tree

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)
4. Finally, return the original root pointer to the calling function.

Algorithm for inorder traversal of binary search tree

1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Algorithm for preorder traversal of binary search tree


1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder (left-subtree)
3. Traverse the right subtree, i.e., call Preorder (right-subtree)

Algorithm for postorder traversal of binary search tree

1. Traverse the left subtree, i.e., call Postorder (left-subtree)
2. Traverse the right subtree, i.e., call Postorder (right-subtree)
3. Visit the root.

Viva Voce Questions

1. Define Binary Tree.
2. What is Binary Search Tree?
3. Explain the difference between Binary Tree and Binary Search Tree with an example?
4. Why do we want to use Binary Search Tree?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -17

Program Name: Program to implement BFS (Breadth First Search) graph traversal algorithm.

Theory:

Graph is another important non-linear data structure. This data structure is used to represent relationship between pairs of elements which are not necessarily hierarchical in nature. A graph is defined as

“Graph G is an ordered set (V, E) where $V(G)$ represents the set of elements, called vertices and $E(G)$ represents the edges between these vertices.

Representation of Graphs: There are two ways to represent a graph $G=(V,E)$:

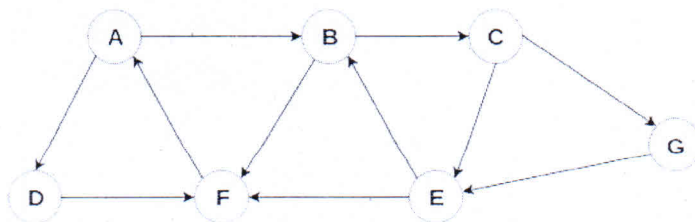
- As an adjacency matrix
- As an adjacency lists

Breadth First Search (BFS) Algorithm

Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighboring nodes. Then, it selects the nearest node and explore all the unexplored nodes. The algorithm follows the same process for each of the nearest node until it finds the goal.

Example

Consider the graph G shown in the following image, calculate the minimum path p from node A to node E . Given that each edge has a length of 1.



Adjacency Lists

A : B, D
B : C, F
C : E, G
G : E
E : B, F
F : A
D : F

Solution:

Minimum Path P can be found by applying breadth first search algorithm that will begin at node A and will end at E. the algorithm uses two queues, namely QUEUE 1 and QUEUE 2.

QUEUE 1 holds all the nodes that are to be processed while QUEUE 2 holds all the nodes that are processed and deleted from QUEUE 1.

Let's start examining the graph from Node A:

1. Add A to QUEUE1 and NULL to QUEUE2.

1. QUEUE1 = {A}
2. QUEUE2 = {NULL}

2. Delete the Node A from QUEUE1 and insert all its neighbors. Insert Node A into QUEUE2

1. QUEUE1 = {B, D}
2. QUEUE2 = {A}

3. Delete the node B from QUEUE1 and insert all its neighbors. Insert node B into QUEUE2.

1. QUEUE1 = {D, C, F}
2. QUEUE2 = {A, B}

4. Delete the node D from QUEUE1 and insert all its neighbors. Since F is the only neighbor of it which has been inserted, we will not insert it again. Insert node D into QUEUE2.

1. QUEUE1 = {C, F}
2. QUEUE2 = {A, B, D}

5. Delete the node C from QUEUE1 and insert all its neighbors. Add node C to QUEUE2.

1. QUEUE1 = {F, E, G}
2. QUEUE2 = {A, B, D, C}

6. Remove F from QUEUE1 and add all its neighbors. Since all of its neighbors has already been added, we will not add them again. Add node F to QUEUE2.

1. QUEUE1 = {E, G}
2. QUEUE2 = {A, B, D, C, F}

7. Remove E from QUEUE1, all of E's neighbors has already been added to QUEUE1 therefore we will not add them again. All the nodes are visited and the target node i.e. E is encountered into QUEUE2.


1. QUEUE1 = {G}
2. QUEUE2 = {A, B, D, C, F, E}

Now, backtrack from E to A, using the nodes available in QUEUE2.

The minimum path will be **A → B → C → E**.

Algorithm or Pseudocode of BFS

- **Step 1:** SET STATUS = 1 (ready state)
for each node in G
- **Step 2:** Enqueue the starting node A
and set its STATUS = 2
(waiting state)
- **Step 3:** Repeat Steps 4 and 5 until
QUEUE is empty
- **Step 4:** Dequeue a node N. Process it
and set its STATUS = 3
(processed state).
- **Step 5:** Enqueue all the neighbors of
N that are in the ready state
(whose STATUS = 1) and set
their STATUS = 2
(waiting state)
[END OF LOOP]
- **Step 6:** EXIT


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

1. What is the use of Breadth first search?
2. Which data structure is used to implement BFS?
3. Can BFS find shortest path?

Program No. -18

Program Name: Program to implement DFS (Depth First Search) graph traversal algorithm.

Theory:

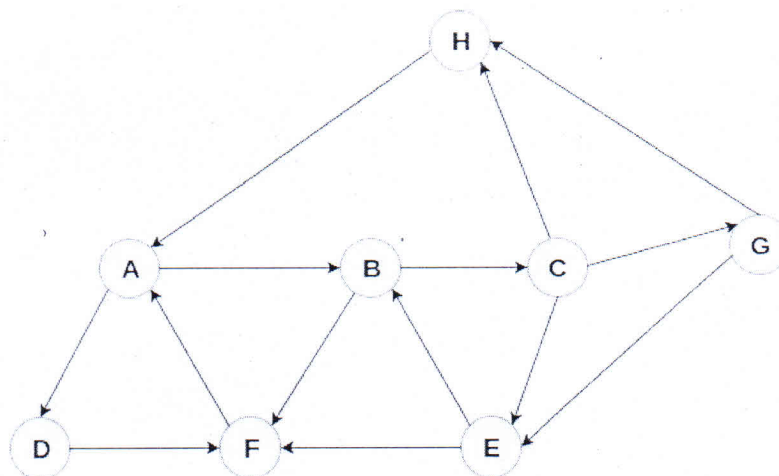
Depth First Search (DFS) Algorithm

Depth first search (DFS) algorithm starts with the initial node of the graph G , and then goes to deeper and deeper until we find the goal node or the node which has no children. The algorithm, then backtracks from the dead end towards the most recent node that is yet to be completely unexplored.

The data structure which is being used in DFS is stack. The process is similar to BFS algorithm. In DFS, the edges that leads to an unvisited node are called discovery edges while the edges that leads to an already visited node are called block edges.

Example :

Consider the graph G along with its adjacency list, given in the figure below. Calculate the order to print all the nodes of the graph starting from node H , by using depth first search (DFS) algorithm.



Adjacency Lists

```
A : B, D
B : C, F
C : E, G, H
G : E, H
E : B, F
F : A
D : F
H : A
```

Solution:

Push H onto the stack

1. STACK: H

POP the top element of the stack i.e. H, print it and push all the neighbors of H onto the stack that are in ready state.

1. Print H
2. STACK: A

Pop the top element of the stack i.e. A, print it and push all the neighbors of A onto the stack that are in ready state.

1. Print A
2. Stack: B, D

Pop the top element of the stack i.e. D, print it and push all the neighbours of D onto the stack that are in ready state.

1. Print D
2. Stack: B, F

Pop the top element of the stack i.e. F, print it and push all the neighbours of F onto the stack that are in ready state.

1. Print F
2. Stack: B

Pop the top of the stack i.e. B and push all the neighbours

1. Print B
2. Stack: C

Pop the top of the stack i.e. C and push all the neighbours.

1. Print C
2. Stack: E, G

Pop the top of the stack i.e. G and push all its neighbours.

1. Print G

2. Stack : E

Pop the top of the stack i.e. E and push all its neighbors'.

1. Print E

2. Stack :

Hence, the stack now becomes empty and all the nodes of the graph have been traversed.

The printing sequence of the graph will be :

$H \rightarrow A \rightarrow D \rightarrow F \rightarrow B \rightarrow C \rightarrow G \rightarrow E$

Algorithm or Pseudocode of DFS

This algorithm executes a depth –first search on Graph G beginning at the starting node A.

STEP 1: initialize all nodes to the ready state (STATUS: =1)

STEP 2: Push the starting node An in QUEUE. And change its status to
The waiting state. (STATUS: =2).

STEP 3: Repeat Steps 4 and 5 until QUEUE is empty:

STEP 4: Pop the Top node N of STACK. Process N and change the status
of N to the processed state. (STATUS: =3)

STEP 5 Push onto STACK all the neighbors of N that are still in the ready
state (STATUS=1), and change their status to the waiting
state (STATUS+2).


[End of Step 3 loop]

STEP 6: exit.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Viva Voce Questions

4. What is the use of depth first search?
5. Which data structure is used to implement DFS?
6. Which is better depth first or breadth first?
7. Why BFS takes more memory than DFS?
8. Can DFS find shortest path?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -19

Program Name: Program to implement Kruskal's algorithm to find minimum spanning tree of a given graph.

Theory:

The minimum spanning tree (MST) of a graph defines the cheapest subset of edges that keeps the graph in one connected component. Telephone companies are particularly interested in minimum spanning trees, because the minimum spanning tree of a set of sites defines the wiring scheme that connects the sites using as little wire as possible. It is the mother of all network design problems.

Minimum spanning trees prove important for several reasons:

- They can be computed quickly and easily, and they create a sparse subgraph that reflects a lot about the original graph.
- They provide a way to identify clusters in sets of points. Deleting the long edges from a minimum spanning tree leaves connected components that define natural clusters in the data set, as shown in the output figure above.
- They can be used to give approximate solutions to hard problems such as Steiner tree and traveling salesman.
- As an educational tool, minimum spanning tree algorithms provide graphic evidence that greedy algorithms can give provably optimal solutions.

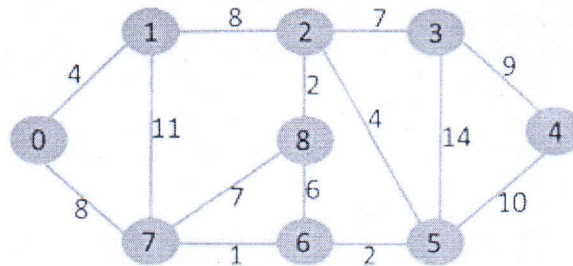
Two classical algorithms efficiently construct minimum spanning trees

1. Prim's and
2. Kruskal's.

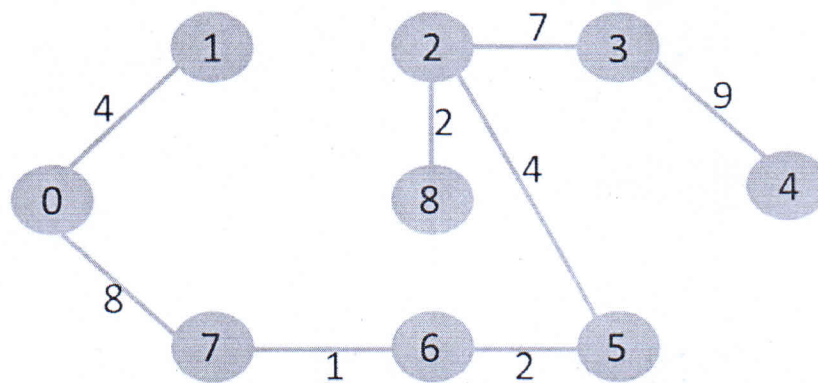
Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

If the graph is not connected, then it finds a *minimum spanning forest* (a minimum spanning tree for each connected component). Kruskal's algorithm is an example of a greedy algorithm.

Example:



Minimum spanning Tree of above graph is given below:



Algorithm or Pseudocode of Kruskal's algorithm

Kruskal(G)

1. **for each** vertex v in G do
2. Define an elementary cluster $C(v) \leftarrow \{v\}$.
3. Initialize a priority queue Q to contain all edges in G , using the weights as keys.
4. Define a tree $T \leftarrow \emptyset$ // T will ultimately contain the edges of the MST

```

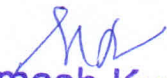
5  while  $T$  has fewer than  $n-1$  edges do    //  $n$  is total number of vertices

7   $(u,v) \leftarrow Q.\text{removeMin}()$            // edge  $u,v$  is the minimum weighted route from/to  $v$ 

    /* prevent cycles in  $T$ . add  $u,v$  only if  $T$  does not already contain an edge consisting of  $u$ 
    and  $v$ . Note that the cluster contains more than one vertex only if an edge containing a pair of the
    vertices has been added to the tree. */


8  Let  $C(v)$  be the cluster containing  $v$ , and let  $C(u)$  be the cluster containing  $u$ .
9  if  $C(v) \neq C(u)$  then
10     Add edge  $(v,u)$  to  $T$ .
11     Merge  $C(v)$  and  $C(u)$  into one cluster, that is, union  $C(v)$  and
         $C(u)$ .
12 return tree  $T$ 

```


Dr. Somesh Kumar
 Prof. & Head, CSE
 Moradabad Institute of Technology
 Moradabad-244001

Viva Voce Questions

1. What is the time complexity of Kruskal algorithm?
2. What is the use of Kruskal algorithm?
3. How is Kruskal algorithm implemented?
4. What is the difference between Prim and Kruskal algorithm?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. -20

Program Name: Program to implement warshall's algorithm to find all pair shortest path of a graph.

Theory:

In graph theory, the shortest path problem is the problem of finding a path between two vertices such that the sum of the weights of its constituent edges is minimized.

The most important algorithms for solving this problem are:

- Dijkstra's algorithm — solves single source problem if all edge weights are greater than or equal to zero. Without worsening the run time, this algorithm can in fact compute the shortest paths from a given start point s to *all other* nodes.
- Bellman-Ford algorithm — solves single source problem if edge weights may be negative.
- A* search algorithm solves for single source shortest paths using heuristics to try to speed up the search
- Floyd-Warshall algorithm — solves all pairs shortest paths.
- Johnson's algorithm — solves all pairs shortest paths, may be faster than Floyd-Warshall on sparse graphs.

Warshall's Algorithm is an **algorithm** for finding the shortest path between all the pairs of vertices in a weighted graph. This **algorithm** works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative).

Let the vertices of G be $V = \{1, 2, \dots, n\}$ and consider a subset $\{1, 2, \dots, k\}$ of vertices for some k . For any pair of vertices $i, j \in V$, considered all paths from i to j whose intermediate vertices are all drawn from $\{1, 2, \dots, k\}$, and let p be a minimum weight path from amongst them.

The Floyd-Warshall algorithm exploits a link between path p and shortest paths from i to j with all intermediate vertices in the set $\{1, 2, \dots, k-1\}$. The link depends on whether or not k is an intermediate vertex of path p .

If k is not an intermediate vertex of path p , then all intermediate vertices of path p are in the set $\{1, 2, \dots, k-1\}$. Thus, the shortest path from vertex i to vertex j with all intermediate vertices in the set $\{1, 2, \dots, k-1\}$ is also the shortest path i to j with all intermediate vertices in the set $\{1, 2, \dots, k\}$.

If k is an intermediate vertex of path p , then we break p down into $i \rightarrow k \rightarrow j$.

Let $d_{ij}^{(k)}$ be the weight of the shortest path from vertex i to vertex j with all intermediate vertices in the set $\{1, 2, \dots, k\}$.


A recursive definition is given by

$$d_{ij}^{(k)} = \begin{cases} W_{ij} & \text{if } k=0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1 \end{cases}$$

Algorithm or Pseudocode of Warshall's algorithm

Create a distance matrix, D , that will describe the distances between vertices. W is a weight matrix of given graph.

1. $n \leftarrow \text{rows}[W]$.
2. $D^0 \leftarrow W$
3. for $k \leftarrow 1$ to n
4. do for $i \leftarrow 1$ to n
5. do for $j \leftarrow 1$ to n
6. do $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
7. return $D^{(n)}$


Dr. Somesh Kumar
 Prof. & Head, CSE
 Moradabad Institute of Technology
 Moradabad-244001

Viva Voce Questions

1. What is warshall's algorithm?
2. What is the use of warshall's algorithm?
3. Which algorithm solves all pair shortest path?
4. What is the difference between warshall's algorithm and dijkstra's algorithm?
5. What is the time complexity of warshall's algorithm?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001