Savitribai Phule Pune University

सावित्रीबाई फुले पुणे विद्यापीठ

# WORKBOOK

*S. Y. B. Sc. (Computer Science) Semester IV*

**CS -243**

Practical course on

CS 241 (Data Structures and Algorithms II)

&

CS 242 (Computer Networks I)

| | | | |
|---|---|---|---|
| **Student Name:** | | | |
| **College:** | | | |
| **Roll No:** | | **Exam Seat No:** | |
| **Year:** | | **Division:** | |

**Coordinators and Editors**

| | |
|---|---|
| **Dr. Manisha Bharambe** | Abasaheb Garware College, Pune. |
| **Dr. Poonam Ponde** | Nowrosjee Wadia College, Pune. |

**Prepared by**

| | |
|---|---|
| Sagar Pravin Chitte | Maratha Vidya Prasarak Samaj's, Commerce, Management and Computer Science (CMCS) College, Nashik. |
| Neeta Nandgude | Vidya Pratisthan's Arts, Science and Commerce College, Baramati. |
| Swati Jadhav | Maharashtra Education Society's Abasaheb Garware College, Pune. |
| Chitra Alavani | Kaveri College of Arts, Science and Commerce |
| Deepashree Mehendale | D. Y. Patil College, Pune. |

# Assignment Completion Sheet

| Sr. No | Assignment Name | Marks (Out of 10) | Signature |
|---|---|---|---|
| **Data Structures and Algorithms II** | | | |
| 1 | Binary Search Tree and Traversals | | |
| 2 | Binary Tree Applications | | |
| 3 | Graph as Adjacency Matrix | | |
| 4 | Graph as Adjacency List | | |
| 5 | Graph Applications - I | | |
| 6 | Graph Applications - II | | |
| 7 | Hash Table – I | | |
| 8 | Hash Table – II | | |
| | Total out of 80 | | |
| | a. Total out of 10 (Data Structures and Algorithms II) | | |
| **Computer Networks - I** | | | |
| 1 | Study of Networking Commands | | |
| 2 | Study of Network IP | | |
| | Total out of 10 | | |
| | b. Total out of 5 (Computer Networks) | | |
| | **Total (Out of 15) (a+b) :** | | |

**This is to certify that Mr/Ms ________________________________________________**

**University Exam Seat Number _________ has successfully completed the course work for**

**Computer Science Lab Course CS 243 and has scored _________ / 15.**

**Instructor** **Head**

**Internal Examiner** **External Examiner**

Date: Date:

# Introduction

**Teaching Scheme : 4 hrs 20 mins / week  Batch Size : 12**

**About the workbook**

This workbook is intended to be used by S. Y. B. Sc (Computer Science) students for the Data structures and Algorithms using C Lab course and Computer Networks in Semester IV.

Workbook is divided in two sections.

1. First section contains assignments on Data Structure and Algorithms II.
2. Second section contains assignments on Computer Networks I.

Data structures and Algorithm is an important core subject of computer science curriculum, and hands-on laboratory experience is critical to the understanding of theoretical concepts studied as part of this course. Study of any programming language is incomplete without hands on experience of implementing solutions using programming paradigms and verifying them in the lab. This workbook provides rich set of problems covering the advanced algorithms as well as numerous computing problems demonstrating the applicability and importance of various data structures and related algorithms and computer networks.

The objectives of this book are

- Defining clearly the scope of the course
- Bringing uniformity in the way the course is conducted across different colleges
- Continuous assessment of the course
- Bring variation and variety in experiments carried out by different students in a batch
- Providing ready reference for students while working in the lab
- Catering to the need of slow paced as well as fast paced learners

**How to use this workbook**

The Data Structures and Algorithms practical syllabus is divided into eight assignments. Each assignment has problems divided into three sets A, B and C.

- Set A consists of simple and basic programs.

- Set B is used to demonstrate small variations on the implementations carried out in set A to improve its applicability.

- Set C consists of advanced programs and applications. Students should spend additional time either at home or in the Lab and solve these problems so that they get a deeper understanding of the subject.

**Instructions to the students**

Please read the following instructions carefully and follow them.

- Students are expected to carry workbook during every practical.

- Students should prepare oneself beforehand for the Assignment by reading the relevant material.

- Instructor will specify which problems to solve in the lab during the allotted slot and student should complete them and get verified by the instructor. However student should spend additional hours in Lab and at home to cover as many problems as possible given in this work book.

- Students will be assessed for each exercise on a scale from 0 to 5

| | |
|---|---|
| ➢ Not done | 0 |
| ➢ Incomplete | 1 |
| ➢ Late Complete | 2 |
| ➢ Needs improvement | 3 |
| ➢ Complete | 4 |
| ➢ Well Done | 5 |

**Instruction to the Practical In-Charge**

- Explain the assignment and related concepts in around ten minutes using white board if required or by demonstrating the software.

- Choose appropriate problems to be solved by students. Set A is mandatory. Choose problems from set B depending on time availability. Discuss set C with students and encourage them to solve the problems by spending additional time in lab or at home.

- Make sure that students follow the instruction as given above.

- You should evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.

- The value should also be entered on assignment completion page of the respective Lab course.

**Instructions to the Lab administrator and Exam guidelines**

- You have to ensure appropriate hardware and software is made available to each student.

- Do not provide Internet facility in Computer Lab while examination

- Do not provide pen drive facility in Computer Lab while examination.

The **operating system and software requirements** are as given below:

- Operating system: Linux
- Editor: Any linux based editor like vi, gedit etc.
- Compiler: cc or gcc

## Section I

# Data Structures and Algorithms II

# Assignment 1: Binary Search Tree and Traversals

**Definition:**

A binary search tree is a binary tree, which is either empty or in which each node contains a key that satisfies following conditions:

1) For every node X, in the tree the values of all the keys in its left subtree are smaller than the key value in X.

2) For every node X, in the tree the values of all the keys in its right subtree are larger than the key value in X.

**Representation of Binary Tree:**

**1. Static Representation of Binary Tree:**

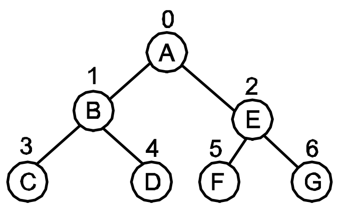
One of the way to represent binary tree using array is to store nodes level by level, starting from the zero level where the root is present. Such representation needs sequential numbering of the nodes starting with nodes on level zero, then those on level 1 and so on. Already, we have defined complete tree. A complete binary tree of height h has $(2^{h+1} - 1)$ nodes in it. The nodes can be stored in one dimensional array, TREE, with the node numbered at location TREE(i). A array of size $(2^{h+1} - 1)$ or $2^d - 1$ (where d = no. of levels) is required.

The root node is stored in the first memory location as the first element in the array. Following rules can be used to decide the location of any ith node of a tree:

For any node with index i, $1 \leq i \leq n$;

a) PARENT (i) = $\left\lfloor \frac{i}{2} \right\rfloor$ if $i \neq 1$
   If i = 1 then it is root which has no parent
b) LCHILD (i) = 2 * i, if $2i \leq n$
   If 2i > n, then i has no left child
c) RCHILD(i) = 2i + 1, if $2i + 1 \leq n$
If (2i + 1) > n, then i has no right child

Example: Consider the given Binary tree:

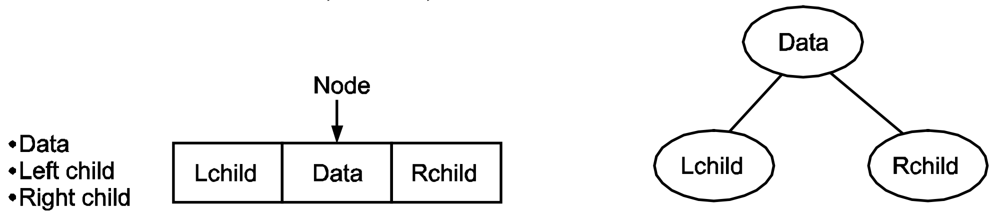


The representation of the above binary tree using array is as followed:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G |

**2. Linked Representation of Binary Tree:**

Another way to represent a binary tree is linked list, which is more memory efficient than the array representation. All nodes should be allocated dynamically. Each node with data and link fields. The root pointer points to the tree in memory.
Each node consists of three fields, Lchild, Data and Rchild.



Linked representation Binary tree

**Binary Search Tree (BST)**:

It is a binary tree with the property that the value in a node is greater than any value in a node's left subtree and less than any value in the node's right subtree. The Operations on Binary Search Tree are:

| | |
|---|---|
| init (T) | creates an empty Binary search tree by initializing T to NULL |
| insert (T, x) | inserts the value x in the proper position in the Binary search tree |
| search (T, x) | searches if the value x is present in the search tree |
| inorder (T) | displays the node using inorder traversal of binary search tree |
| postorder (T) | displays the node using postorder traversal of binary search tree |
| preorder (T) | displays the node using preorder traversal of binary search tree |
| delete(T, x) | Deletes node x from binary search tree |

Consider the structure of a node of a BST*:*

```
typedef struct BST
{
        struct BST * Lchild;
        int data;
        struct BST * Rchild;
}BSTnode;
```

**Insert (T,x):** Consider T is root node and x is element to be inserted in BST

**Algorithm Steps:**

```
1.  t = root, flag = false
2.  while (t ≠ null) & (flag = false) do
    Case 1:   key < t -> data
              t1 = t;
              t = t ->Lchild;
              Case 2:    key > t -> data
                         t1 = t
                         t = t ->Rchild
              Case 3:    t -> data = x
                         flag = true
                         display "item already exist"
                         break
                      end case
              End while
        3.    If (t = null) then
                   new = getnode (node)       //create node
                   new -> data = x
                   new ->Lchild = null        //initialize a node
                   new ->Rchild = null
                   if (t1 -> data <x) then    //insert at right
                        t1 ->Rchild = new
                   else t1 ->Lchild = new     //insert at left
              endif
4.            stop
```

**Search(T,x):** Consider T is a root node and x is an element to be searched in BST.

**Algorithm steps:**

```
        1.    Initialize t = root, flag = false
        2.    while (t ≠ null) and (flag = false) do
                   Case 1:  t->data = key
                         flag = true          //successful search
                   Case 2:  Key < t -> data
                         t = t ->Lchild       // goto left subtree
                   Case 3:  Key > t -> data
                         t = t ->Rchild       //goto right subtree
                   End case
              End while
        3.    If (flag = true) then
                      display "Key is found at node", t
              else
                      display "Key is not exist"
              end if;
        4.    Stop
```

**Traversal Methods (Recursive Algorithm):**

Consider T is root node of BST which is passed as argument to a function when function is called from main() function.

**inorder(T):**

**Algorithm steps:**

```
   1.  If T is not empty then
           inorder (T->Lchild)
```

```
            display T->data
            inorder(T->Rchild)
    2. stop
```

**preorder(T):**

**Algorithm steps:**

```
    1.If T is not empty then
            display T->data
            preorder (T->Lchild)
            preorder(T->Rchild)
    2. stop
```

**postorder(T):**

**Algorithm steps:**

```
    1.  If T is not empty then
            postorder (T->Lchild)
            postorder(T->Rchild)
            display T->data
    2. stop
```

**Delete (T,x):** Consider T is root node and x is an element to be deleted from BST**.** There are three cases with respective to node to be deleted:

1. x is a leaf node.
2. x has one child.
3. x has both child nodes.

**Algorithm Steps:**

```
1.  t = root, flag = false
2.  while (t ≠ null) and (flag = false) do
            Case 1: key < t->data
                        parent = t
                        t = t ->Lchild
            Case 2: key > t -> data
                        parent = t
                        t = t ->Rchild
            Case 3: t -> data = key
                         flag = true
            end case
            end while
3.  If flag = false
    Then display "item not exist".
    exit.
    /* case 1 if node has no child */
4.  If (t ->Lchild= null) and (t ->Rchild= null) then
      if (parent ->Lchild= t) then
            parent ->Lchild = null       //set pointer to its parent when node is left child
        else
            parent ->Rchild = null
```

```
5.  /* case 2 if node contains one child */
        If (parent ->Lchild = t) then                     //when node contains only one child
            if (t ->Lchild = null) then                                 //if node is left child
                parent ->Lchild = t ->Rchild
            else
                parent ->Lchild= t ->Lchild
            endif
        else
            if (parent ->Rchild = t) then
                if (t ->Lchild = null) then
                    parent ->Rchild = t ->Rchild
                else
                    parent ->Rchild = t ->Lchild
                endif
            endif
    endif
6.  /* Case 3 if node contains both child */
        t1 = succ(t)               //find inorder successor of the node
        key1 = t1 -> data
        Delete(key1) //delete inorder successor
        t -> data = key            //replace data with the data of an order successor
7.  Stop.
```

Other operations on a binary search tree include counting leaf and non leaf nodes in the tree.
**count (T)(Recursive Definition):**This will give total number of nodes of BST

**Algorithm Steps:**

```
1. counter=0
2. If (T≠NULL)  then
            counter=counter+1
            count (T->Lchild)
            count(T->Rchild)
2. stop
```

Similarly, conditions for checking leaf node and non-leaf can be written and respective counters can be incremented.

## <u>Set A</u>

a) Implement a Binary search tree (BST) library (btree.h) with operations – create, search, insert, inorder, preorder and postorder. Write a menu driven program that performs the above operations.

b) Write a program which uses binary search tree library and counts the total nodes and total leaf nodes in the tree.
int count(T) – returns the total number of nodes from BST
int countLeaf(T) – returns the total number of leaf nodes from BST

**Set B**

a) Write a C program which uses Binary search tree library and implements following function with recursion:
T copy(T) – create another BST which is exact copy of BST which is passed as parameter.
int compare(T1, T2) – compares two binary search trees and returns 1 if they are equal and 0 otherwise.

**Set C**

a) Write a C program which uses Binary search tree library and implements following two functions:
int sumodd(T) – returnssum of all odd numbers from BST
intsumeven(T) – returnssum of all even numbers from BST
mirror(T) – converts given tree into its mirror image.

b) Write a function to delete an element from BST.

c) What modifications are required in search function to count the number of comparisons required?

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2:Late Complete | |
| 3: Needs Improvement | | 4: Complete | | 5: Well Done | |

**Practical In-charge**

**Date:**

## Assignment 2: Binary Tree Applications

The level order traversal **levelwisedisplay (T)** traverses the tree levelwise starting from the root.

Here queue data structure is required to store node address. The variables total represents total number of nodes of BST, variable level represents number of level in BST and variable count represents total number of node in each level.

**Algorithm Steps:**

```
1. temp = root
2. add(Q, temp)     //insert temp in queue
3. add(Q, NULL)//insert NULL in queue
4. if queue is not empty then
   temp = delete(Q)
   else goto 6
5. if (temp≠NULL) then
        total=total+1
        cnt=cnt+1
       display t -> data
       if (temp ->Lchild≠NULL) then
               add(Q, T->Lchild)
       if (T->Rchild≠NULL) then
               add(Q, T->Rchild)
               goto step 4
   else
       if queue is not empty then
               display  "\n" ( newline character )
            display cnt
        cnt = 0
            level=level+1
add(Q,NULL)
      else goto step4
6. Stop
```

The Binary tree is widely used in a variety of applications.

It is used in a sorting method called Heap Sort. This sorting method uses a special type of binary tree called "Heap". A max-heap is a full or complete binary tree such that each parent has a value greater than its children.

A set of n elements can be considered as a binary tree implemented using array. This tree is converted into a Heap and then sorted.

**Algorithm Heapsort**

```
Step 1.   Start
Step 2. Accept n elements in array A.
Step 3. Convert array A into a heap
Step 4. last = n – 1, top = 0.
Step 5. Interchange A[top] and A[last]
Step 6. last = last – 1
Step 7. Heapify (A, top, last), i.e. recreate heap
Step 8. If last > 0
            goto Step 5
Step 9. Stop
```

**Algorithm Heapify(A, top, last)**

```
1.   Start
2.   key = A[top]
3.   j = 2∗top +1  i.e. j is at the left child
4.   If A[j] < A[j+1]  i.e. j points to the larger child
          j = j + 1
5.   If key < A[j]  i.e if parent < child
          Interchange A[top] and A[j]
          Heapify (A,j,n)
6.   Stop
```

**Set A**

a) Write a C program which uses Binary search tree library and displays nodes at each level, count of node at each level and total levels in the tree.

**Set B**

a) Write a program to sort n randomly generated elements using Heapsort method.

**<u>Set C</u>**

a) Which data structure will be required to display nodes of BST depth wise?
b) Write a C program to displays nodes of BST depth wise.
c) Write a C program to compare two binary search trees (node data wise comparison).
d) How to implement mirror() and copy() functions without recursion?
e) How to convert singly linked list to binary search tree?

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2:Late Complete | |
| 3: Needs Improvement | | 4: Complete | | 5: Well Done | |

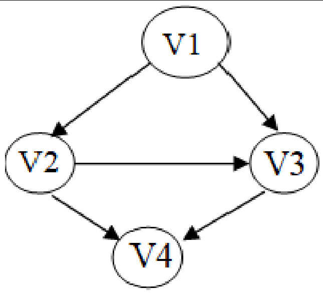
**Practical In-charge**
**Date:**

## Assignment 3: Graph as Adjacency Matrix

A graph consists of a set of vertices and a set of edges. We can write a graph as G=(V,E), where V is a set of nodes (vertices) and E is a set of edges (arcs).

The one way of representing graphs is adjacency matrix representation.

In adjacency matrix representation of a Graph with n vertices and e edges, a two dimensional n x n array , say a , is used , with the property that a[i,j] equals 1 if there is an edge from i to j and a[i,j] equals 0 if there is no edge from i to j.

| Graph | Adjacency Matrix |
|---|---|
|  | $A44= \begin{array}{c\|cccc} & V1 & V2 & V3 & V4 \\ \hline V1 & 0 & 1 & 1 & 0 \\ V2 & 0 & 0 & 1 & 1 \\ V3 & 0 & 0 & 0 & 1 \\ V4 & 0 & 0 & 0 & 0 \end{array}$ |

The usual operations on graph are:

Indegree(i) – returns the indegree (the number of edges ending on) of the ith vertex
Outdegree(i) – returns the outdegree(the number of edges moving out) of the ith vertex)
displayAdjMatrix – displays the adjacency matrix for the graph

**Set A**

a) Write a C program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix.

b) Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement functions to print indegree, outdegree and total degree of all vertices of graph.

**Set B**

**a)** Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement function to traverse the graph using Breadth First Search (BFS) traversal.

b) Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement function to traverse the graph using Depth First Search (BFS) traversal.

**<u>Set C</u>**

a) Which data structure is used to implement Breadth First Search?
b) Where the new node is appended in Depth first search of OPEN list?
c) What is simple graph?
d) Which data structure is used to implement adjacency matrix method?

**Assignment Evaluation**

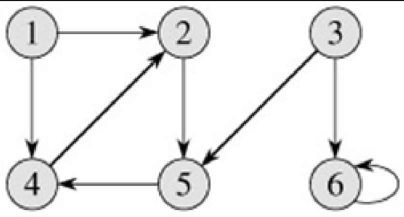
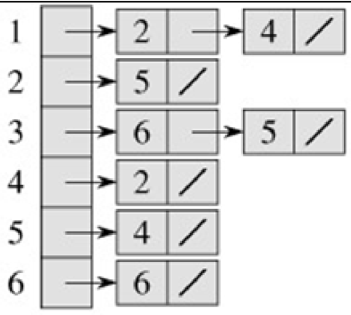
| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2:Late Complete | |
| 3: Needs Improvement | | 4: Complete | | 5: Well Done | |

**Practical In-charge**

**Date:**

## Assignment 4: Graph as Adjacency List

Another way of representing graphs is adjacency list representation. In adjacency list representation of a graph with n vertices and e edges, there are n linked lists, one list for each vertex in the graph.

| Graph | Adjacency Matrix |
|---|---|
|  |  |

To implement graph as an adjacency list, we create an array of lists. The usual operations on graph are:

Indegree(i) – returns the indegree (the number of edges ending on) of the ith vertex
Outdegree(i) – returns the outdegree(the number of edges moving out) of the ith vertex)
displayAdjList – displays the adjacency list for the graph

**<u>Set A</u>**

a) Write a C program that accepts the vertices and edges of a graph. Create adjacency list and display the adjacency list.

b) Write a C program that accepts the vertices and edges of a graph. Create adjacency list. Implement functions to print indegree, outdegree and total degree of all vertex of graph.

**<u>Set B</u>**

**a)** Write a C program that accepts the vertices and edges of a graph and store it as an adjacency list. Implement function to traverse the graph using Breadth First Search (BFS) traversal.

b) Write a C program that accepts the vertices and edges of a graph and store it as an adjacency list. Implement function to traverse the graph using Depth First Search (BFS) traversal.

**<u>Set C</u>**

a) Which data structure is used to implement Depth First Search?
b) Where the new node is appended in Breadth-first search of OPEN list?
c) What is complete graph?

d) Which data structure is used to implement adjacency list method?

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2:Late Complete | |
| 3: Needs Improvement | | 4: Complete | | 5: Well Done | |

**Practical In-charge**

**Date:**

## Assignment 5: Graph Applications – I

**Topological Sorting:**

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that if there is an edge directed towards vertex $v_j$ from vertex $v_i$, then $v_i$ comes before $v_j$. Topological sorting for a graph is not possible if the graph is not a DAG. There can be more than one topological sorting for a graph.

For topological sort to perform we need to find adjacency matrix. Example: The **topological sort** algorithm takes a directed graph and returns an array of the nodes where each node appears *before* all the nodes it points to.

**Algorithm**

```
1.  Begin
2.  Create a graph
3.  Find in degree of each vertex
4.  Insert vertices of in degree 0 in queue Q
5.  While queue is not empty and all vertices are not covered
      ▪ Add vertex v to sorted array Topo_sort
      ▪ Delete all outgoing edges from vertex v
6.  If all vertices are covered then display Topo_sort array
    Else display message "Graph contains cycle, no topological ordering is possible".
7.  End
```

**Minimum Spanning Tree**

A spanning tree is a sub-graph of an undirected connected graph, which includes all the vertices of the graph with a minimum possible number of edges. An example is a cable company wanting to lay line to multiple neighbourhoods; by minimizing the amount of cable laid, the cable company will save money.

A **tree** has one path joins any two vertices. A **spanning tree** of a graph is a tree that:

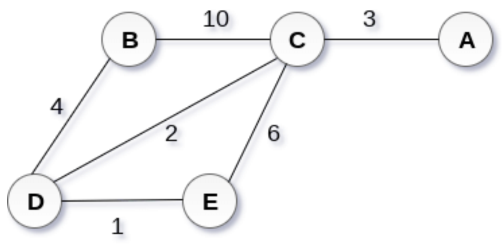
- Contains all the original graph's vertices.
- Reaches out to (spans) all vertices.
- Is acyclic means the graph doesn't have any nodes which loop back to itself.

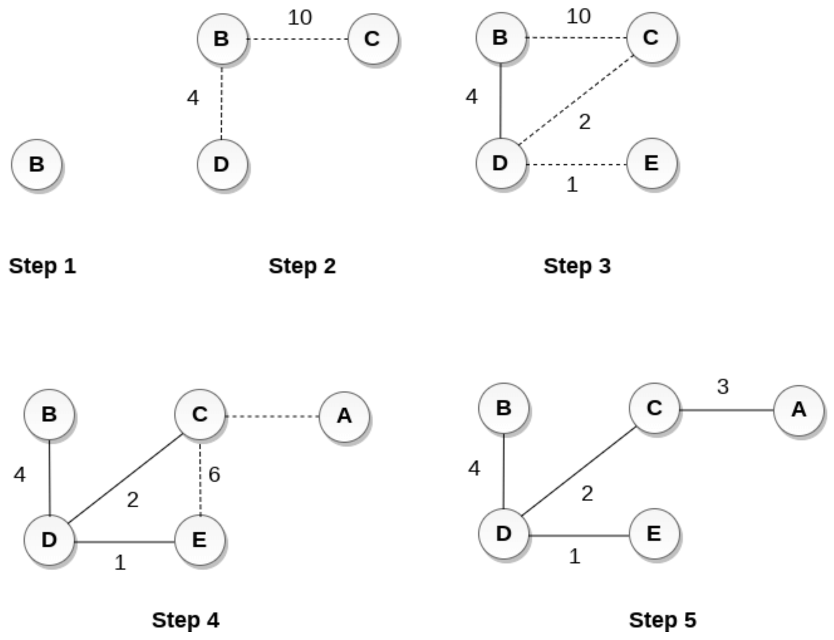
The minimum spanning tree from a graph is found using the following algorithms:

A. Prim's Algorithm
B. Kruskal's Algorithm

**A. Prims *Algorithm***

***Example:***





**Algorithm:**

1) Start
**2)** Create a set *mstSet* that keeps track of vertices already included in MST.
**3)** Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign
  key value as 0 for the first vertex so that it is picked first.
**4)** While mstSet doesn't include all vertices
    **a)** Pick a vertex *u* which is not there in *mstSet* and has minimum key value.
    **b)** Include *u* to mstSet.
    **c)** Update key value of all adjacent vertices of *u*.

To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u-v* is less than the previous key value of *v*, update the key value as weight of *u-v*
*5) End*

**B. Kruskal's Algorithm**

1. Create a priority queue Q that contains all the edges of the graph.
2. Repeat Steps 4 and 5 while Q is NOT EMPTY
3. Remove an edge from Q
4. IF the edge obtained in Step 4 connects two different trees, then Add it to the forest (for combining two trees into one tree).

```
   ELSE
   Discard the edge
5. END
```

**Set A**

a) Write a C program for the implementation of Topological sorting.

b) Write a C program for the Implementation of Prim's Minimum spanning tree algorithm.

**Set B**

a) Write a C program for the Implementation of Kruskal's Minimum spanning tree algorithm.

**Set C**

a) A graph may not have an edge from a vertex back to itself(self edges or self loops).
Given an adjacency matrix representation of a graph, how to know if there are self edges?

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2:Late Complete | |
| 3: Needs Improvement | | 4: Complete | | 5: Well Done | |

**Practical In-charge**

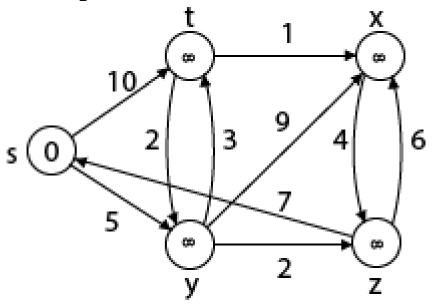
**Date:**

## Assignment 6: Graph Applications – II

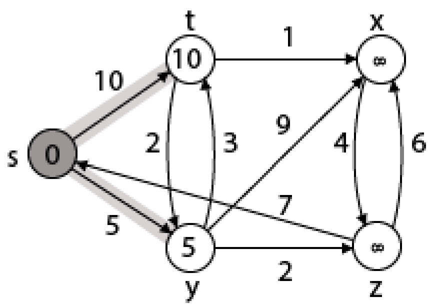
### Dijkstra’s algorithm

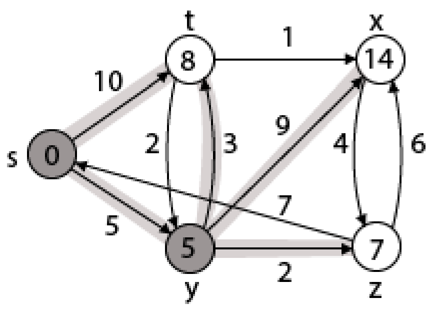
Dijkstra algorithm is a greedy algorithm. It finds a shortest path tree for a weighted undirected graph. Dijkstra’s algorithm finds the solution for the single source shortest path problems only when all the edge-weights are non-negative.

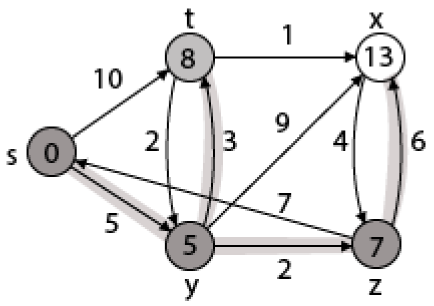
Example:

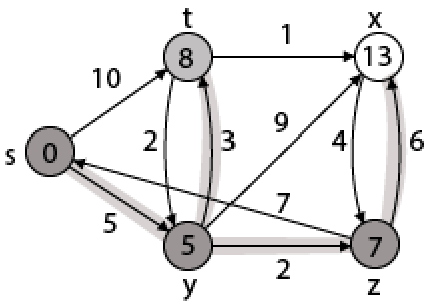


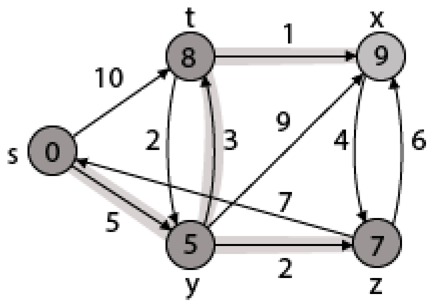
All vertices are scanned one by one to find out adjacent vertices. Calculate the distance of each adjacent to the source vertices. A stack is maintained, which contains those vertices which are selected after computation of shortest distance. Now find the adjacent of s that are t and y. Find shortest distances of t and y. It is found that y is with shortest distance.

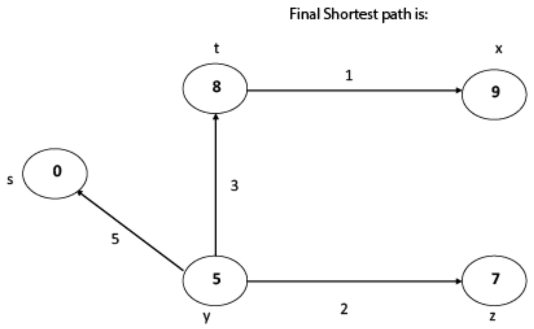












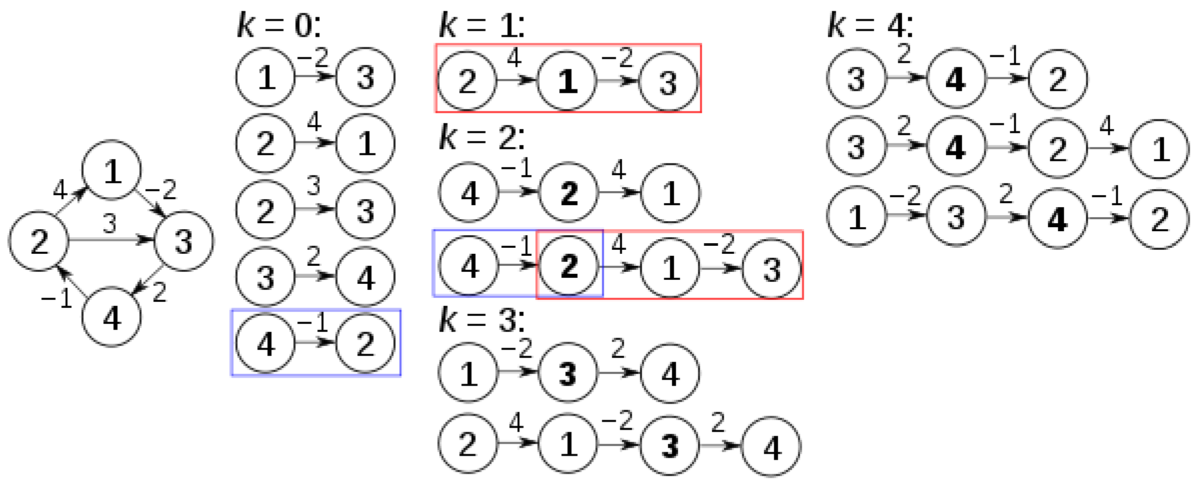
1. Create a set shortPath to store vertices that come in the way of the shortest path tree.
2. Initialize all distance values as INFINITE and assign distance values as 0 for source vertex so that it is picked first.
3. Loop until all vertices of the graph are in the shortPath.
   - 3.1 : Take a new vertex that is not visited and is nearest.
   - 3.2 : Add this vertex to shortPath.
   - 3.3 : For all adjacent vertices of this vertex update distances. Now check every adjacent of V, if sum of distance of u and weight of edge is else the update it.

**Floyd-Warshall Algorithm:**

This algorithm is used 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).This algorithm follows the dynamic programming approach to find the shortest paths.

The solution matrix is initialized same as the input graph matrix. Then update the solution matrix by considering all vertices as an intermediate vertex. The idea is to one by one pick all vertices and update all shortest paths which include the picked vertex as an intermediate vertex in the shortest path. When vertex k is selected as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices. For every pair (i, j) of source and destination vertices respectively, there are two possible cases.

- k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.
- k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j].



**Input** – The cost matrix of given Graph.

**Output --** Matrix to for shortest path between any vertices to any vertex.

```
Begin
   for k := 0 to n, do
      fori := 0 to n, do
         for j := 0 to n, do
            if cost[i,k] + cost[k,j] < cost[i,j], then
               cost[i,j] := cost[i,k] + cost[k,j]
         done
      done
   done
   display the current cost matrix
End
```

## Set A

a) Write a C program for the implementation of Dijkstra's shortest path algorithm for finding shortest path from a given source vertex using adjacency cost matrix.

## Set B

a) Write a C program for the implementation of Floyd Warshall's algorithm for finding all pairs shortest path using adjacency cost matrix.

## Set C

a) A graph can be used to show relationships between people. For example following list of vertices represent people and list of edges represent the friendships like:

People={ George, Jim, Jean, Frank, Fred, John, Susan}
Friendship= { (George, Jean),(Frank, Fred), (George, John),(Jim, Fred), (Jim, Frank)}

1. Find all friends of John
2. Find all friends of Susan

3. Find all friends of Jim.

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2:Late Complete | |
| 3: Needs Improvement | | 4: Complete | | 5: Well Done | |

**Practical In-charge**

**Date:**

## Assignment 7: Hash Table-I

A hash table or hash map is a data structure that efficiently stores and retrieves data from memory. Hash table is a data structure that represents data in the form of key-value pairs. Each key is mapped to a value in the hash table. The keys are used for indexing the values/data. A similar approach is applied by an associative array. A hash table uses a hash function to compute an index, also called a hash code, into an array of buckets or slots, from which the desired value can be found. Ideally, the hash function will assign each key to a unique bucket, but most hash table designs employ an imperfect hash function, which might cause hash collisions where the hash function generates the same index for more than one key.

Operations to be performed

a. Initialize the Hash Bucket
b. Insert elements in the hash table
c. Search elements from the hash table
d. Delete an element from the hash table
e. Display the contents of hash table

### Algorithm

1. Create hash table (array of max size)
2. Take a value to be stored in hash table as input.
3. Apply appropriate has function to generate a key(index) based on the value
4. Using the generated index, access the data located in that array index.
5. In case of absence of data, insert the data item (value) into it and increment the size of hash table.
6. In case the data exists, probe through the subsequent elements (looping back if necessary) for free space to insert new data item.
   Note: This probing will continue until we reach the same element again (from where we began probing)
7. To display all the elements of hash table, element at each index is accessed (via for loop).
8. To remove a key from hash table, we will first calculate its index and delete it if key matches, else probe through elements until we find key or an empty space where not a single data has been entered (means data does not exist in the hash table).
9. Exit

**<u>Set A</u>**

a) Write a program to implement various types of hash functions which are used to place the data in a hash table

a. Division Method
b. Mid Square Method
c. Digit Folding Method

Accept n values from the user and display appropriate message in case of collision for each of the above functions.

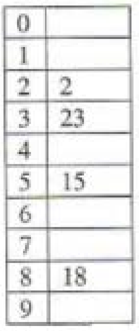
**<u>Set B</u>**

a) Write a menu driven program to implement hash table using array (insert, search, delete, display). Use any of the above-mentioned hash functions. In case of collision apply linear probing.

b) Write a menu driven program to implement hash table using array (insert, search, delete, display). Use any of the above-mentioned hash functions. In case of collision apply quadratic probing.

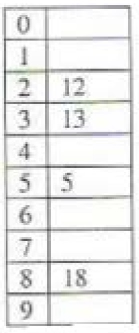
**Set C**

a) Calculate the time complexity of hash table with Linear Probing
b) The keys 12, 18, 13, 2, 3, 23, 5 and 15 are inserted into an initially empty hash table of length 10 using open addressing with hash function h(k) = k mod 10 and linear probing. What is the resultant hash table?



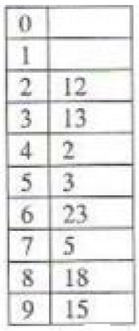
| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 2 |
| 3 | 23 |
| 4 | |
| 5 | 15 |
| 6 | |
| 7 | |
| 8 | 18 |
| 9 | |

(A)



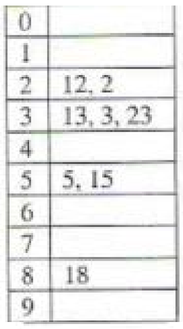
| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 12 |
| 3 | 13 |
| 4 | |
| 5 | 5 |
| 6 | |
| 7 | |
| 8 | 18 |
| 9 | |

(B)



| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 12 |
| 3 | 13 |
| 4 | 2 |
| 5 | 3 |
| 6 | 23 |
| 7 | 5 |
| 8 | 18 |
| 9 | 15 |

(C)



| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 12, 2 |
| 3 | 13, 3, 23 |
| 4 | |
| 5 | 5, 15 |
| 6 | |
| 7 | |
| 8 | 18 |
| 9 | |

(D)

Which data structure is appropriate for simple chaining? Why?

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2:Late Complete | |
| 3: Needs Improvement | | 4: Complete | | 5: Well Done | |

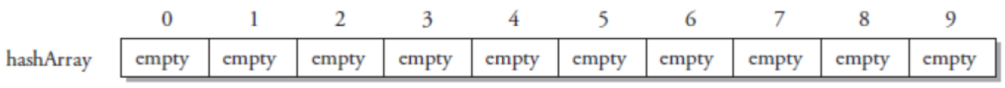
**Practical In-charge**

**Date:**

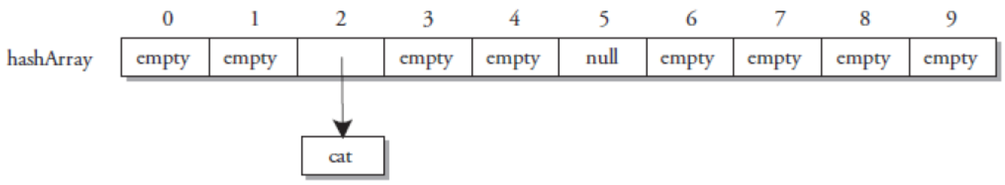
## Assignment 8: Hash Table-II

Implementing hash table using Linked list

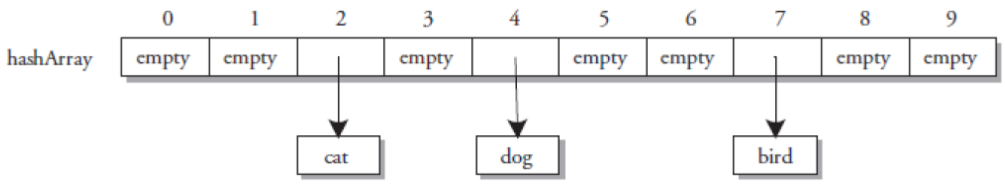
1. Existing hash table initialized with 10 empty linked lists



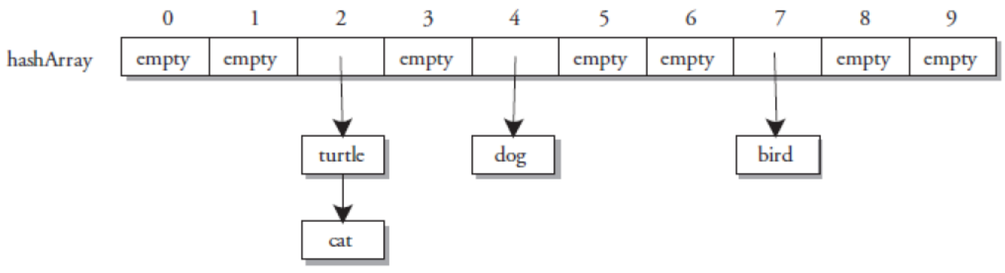
2. After adding "cat" with hash of 2



3. After adding "dog" with hash of 4 and "bird" with hash of 7



4. After adding "turtle" with hash of 2 – collision and chained to linked list with "cat"



**Algorithm | Insert data into the separate chain**

```
1. Declare an array of a linked list with the hash table size.
2. Initialize an array of a linked list to NULL.
3. Find hash key.
4. If chain[key] == NULL
     Make chain[key] points to the key node.
5. Otherwise(collision),
     Insert the key node at the end of the chain[key].
```

**Algorithm Searching a value from the hash table**

```
1. Get the value
2. Compute the hash key.
3. Search the value in the entire chain. i.e. chain[key].
```

4. If found, print "Search Found"
5. Otherwise, print "Search Not Found"

**<u>Set A</u>**

a) Implement hash table using singly linked lists. Write a menu driven program to perform operations on the hash table (insert, search, delete, display). Select appropriate hashing function. In case of collision, use separate chaining.

**<u>Set B</u>**

a) Implement hash table using doubly linked lists. Write a menu driven program to perform operations on the hash table (insert, search, delete, display). Select appropriate hashing function. In case of collision, use separate chaining.

**<u>Set C</u>**

a) What the pseudo code to find all the pairs of two integers in an unsorted array that sum up to a given S using hash tables.
Example:
if the array is [3, 5, 2, -4, 8, 11] and the sum is 7,
program should return [[11, -4], [2, 5]] since11 + -4 = 7 and 2 + 5 = 7.

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2:Late Complete | |
| 3: Needs Improvement | | 4: Complete | | 5: Well Done | |

**Practical In-charge**

**Date:**

## Section II

# Computer Networks - I

## Assignment 1: Study of Networking Commands

**Execute the following commands**

1. hostname: - Displays the machine host name

______________________________________________

2. hostname –d: - displays the domain name of the machine

______________________________________________

3. hostname –f: - displays the fully qualified host and domain name

______________________________________________

4. hostname –i:-displays the IP address for the current machine

______________________________________________

______________________________________________

5. ping: -This command is used to test connectivity between two nodes. You can ping host name or ip address. Using command ping <ip-address> or ping www.google.com

______________________________________________

______________________________________________

6. netstat: - (Network Statistic) command displays interfaces, connection information, routing table information.

______________________________________________

______________________________________________

7. netstat –a: - This command is used to list all ports.

______________________________________________

______________________________________________

8. netstat –at: - This command will display all TCP ports.

______________________________________________

______________________________________________

9. netstat –au: - This command will lists all UDP ports

______________________________________________

10. netstat –l: - This command will lists sockets in listening mode

11. netstat –lt:- This command will lists all sockets of TCP ports in listening mode.

12. netstat –lu: - This command will lists all sockets of UDP ports in listening mode.

13. netstat –s :- This command shows the statistics for each protocol

14. netstat – st:- This command will show the statistics for TCP protocol

15. netstat –su:- This command will show the statistics for UDP protocol

16. traceroute**:** -traceroute is a network troubleshooting utility which shows number of hops taken to reach destination also determine packets traveling path. For example: -traceroute google.com

17. ifconfig: -ifconfig is used for displaying network interface information. ifconfig (interface configurator) command is use to initialize an interface, assign IP Address to interface and

enable or disable interface on demand. With this command you can view IP Address and Hardware / MAC address assign to interface and also MTU (Maximum transmission unit) size.

______________________________________________________________________________

______________________________________________________________________________

18. ifconfig –a:- This command will show all network configuration and settings.

______________________________________________________________________________

______________________________________________________________________________

**19.** nslookup
If you know the IP address it will display hostname. To find all the IP addresses for a given domain name, the command nslookup is used. You must have a connection to the internet for this utility to be useful.

______________________________________________________________________________

______________________________________________________________________________

**20**. nmap
Network mapper tool to discover hosts and services on a computer network.
$ nmap<ip-address>

______________________________________________________________________________

______________________________________________________________________________

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2:Late Complete | |
| 3: Needs Improvement | | 4: Complete | | 5: Well Done | |

**Practical In-charge**

**Date:**

## Assignment 2: Study of Network IP

1. What is an IP address?

2. What is the use of IP address?

3.How a host determines its IP address?

4. How is an IP address represented?

5. What are the components of IP address?

6. How to determine the class of IP address?

7. Determine to which IP address classes the following addresses belong:

| IP Address | Class |
|---|---|
| 201.20.30.40 | |
| 67.39.81.52 | |
| 191.116.92.3 | |
| 238.24.182.85 | |
| 28.47.18.167 | |

8. Determine the network and host numbers of the IP address:

| IP address | Network id | Host id |
|---|---|---|
| 201.20.30.40 | | |
| 67.39.81.52 | | |
| 191.116.92.3 | | |
| 238.24.182.85 | | |
| 28.47.18.167 | | |

9. According to the provided IP addresses determine the address class to which it belongs, the number of network in that class and the number of the host in that network.

| IP address | Class | Network in the class | Host in the network |
|---|---|---|---|
| 181.37.137.75 | | | |
| 92.33.154.222 | | | |
| 220.108.179.78 | | | |
| 242.24.18.40 | | | |

10. Why we develop sub netting and how to calculate subnet mask and how to identify subnet address.

11. Why we develop super netting and how to calculate supernet mask and how to identify supernet address.

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2:Late Complete | |
| 3: Needs Improvement | | 4: Complete | | 5: Well Done | |

**Practical In-charge**
**Date:**