School Of Computer Science

Gujarat University

Certificate

Roll No: **30** Seat No: _____

This is to certify that Mr. /~~Ms.~~ **Rathod Ajinkya** student of MCA Semester **-** IV, has duly completed his / ~~her~~ term work for the semester ending in May 2021, in the subject of **Design & Analysis of Algorithms (DAA)** towards partial fulfillment of his / ~~her~~ Degree of Masters in Computer Applications.

May 26, 2021
Date of Submission: Internal Faculty

Head of Department

Department Of Computer Science
Rollwala Computer Centre
Gujarat University

MCA – IV

Subject: - Design and Analysis of Algorithms (DAA)

Name: - Ajinkya Rathod

Roll No.: - 30 **Exam Seat No.: -**_____

Sr. No.	Contents	Date	Signature
1	Assignment 1 (Asymptotic Notations)	26-05-2021	
2	Assignment 2 (Recurrence Relations)	26-05-2021	
3	Assignment 3 (Sorting & Searching)	26-05-2021	
4	Assignment 4 (Graphs & Algorithms)	26-05-2021	
5	Assignment 5 (Exercises)	26-05-2021	

D E P A R T M E N T O F C O M P U T E R S C I E N C E
R O L L W A L A C O M P U T E R C E N T R E
G U J A R A T U N I V E R S I T Y
M.C.A. – IV

R O L L NO : 30
N A M E : Ajinkya Rathod
S U B J E C T : Design & Analysis of Algorithms (DAA)

NO.	TITLE	DATE	SIGN
1	**Practical Assignment**	**26-05-2021**	
2	**Hands on Classwork**	**26-05-2021**	
	- **Insertion Sort**		
	- **Merge Sort**		
	- **Heap Sort**		
	- **Activity Selection**		
	- **Longest Common Subsequence**		
	- **Matrix Multiplication**		
	- **Matrix Chain Multiplication**		
	- **Huffman Coding**		
	- **Naive String Matching**		
	- **Rabin Karp String Matching**		
	- **Travelling Salesman Problem**		
	- **Graphs (BFS, DFS, Dijkstra, Kruskal's, Prims, Bellman Ford)**		

PAGE NO.
DATE :

"Jai Swaminarayan"

Name : Ajinkya Rathod

D A A

Assgn 1

PAGE NO.
DATE :

Q1. What is algorithm?

Ans) It is any well defined computational procedure that takes some value or set values as input and produces some value or set of value as output

→ It is sequence of computational steps that transform the input into output.

→ We also require for solving well specified computational problem.

Q2

Ans) Asymptotic notation

It uses primarily to describe the running time of algorithm.

The notations are used to represent the function in simple form or belong to class of function.

Q3 Explain Asymptotic Notation.

Ans Big O
Big Omega
Theta

(1) O Big oh notation

$f(n) = O(g(n))$ ∃ +ve const

$\therefore f(n) \leq C * g(n)$

$\therefore f(n) = O(g(n))$

(a) Ω

$f(n) \geq c * g(n) \quad \forall n \geq n_0$

$\therefore f(n) = \Omega(g(n))$

(b) Θ

$f(n) = \Theta(g(n))$

$\frac{c_1 * g(n)}{1.0} \leq f(n)$

$f(n) = \Theta(g(n))$

Q4 $\frac{n^3}{1000} - 100n^2 - 100n + 3$

$f(x) \leq c * g(n)$

$\frac{n^3}{100} - 100n^2 - 100n + 3 \leq n$

$= 76.998\%$

$f(n) = O(g(n)) = O(n)$

$f(n) \leq c * g(n)$

$\frac{n^3}{1000} - 100n^2 - 1000n + 3 \geq 1 + n$

Q: Modify any algo to have good worst running time?

A: Merge Sort.

Q: Explain divide and conquer method in detail

A: Many useful algorithms are recursive in structure to solve a given problem they call themselves

→ It follows divide and conquer approach.

→ They break the problem into several sub-problem that are similar to original problem

(1) Divide

(2) Conquer

(3) Combine

Q Time Complexity of Merge & Insertion sort

Insertion Sort	
Best	$O(n)$
Avg.	$O(n^2)$
Worst	$O(n^2)$

Merge

Best — $O(N \log N)$

Avg. — $O(N \log N)$

Worst — $O(N \log N)$

Q Define the term Recursion

- When algorithm contains a recursive call to itself, then its running time is described using recursive or recurrence equation.

A recursion is a type of mechanism that defines an function in terms of itself on smaller inputs.

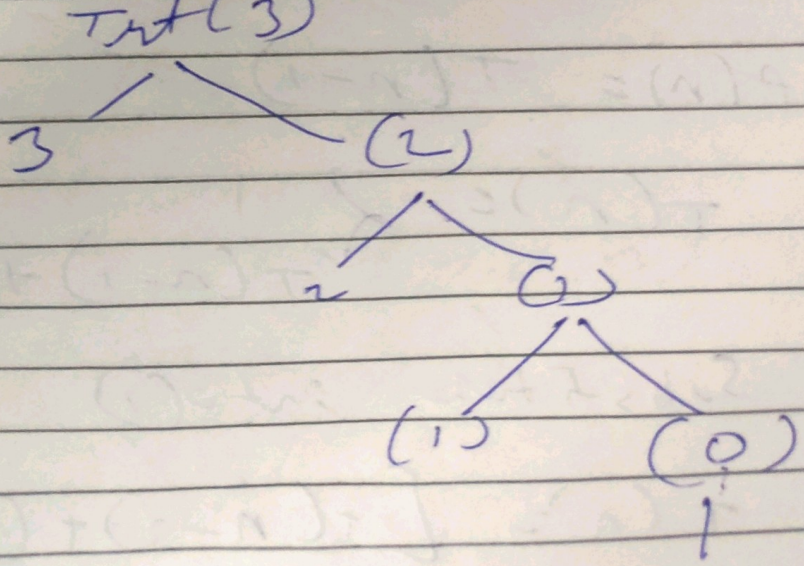
Q Diffrnt Method of Recursive relation

Ans

```
void fun(int n) {
    if (n>0) {
        printf("%d", n);
        fun(n-1);
    }
}
```

fun n=3

Tree fun(n=)



3+1=4 times called function

PAGE NO.
DATE :

Assignment - 2

DAA

Q10 Analyse the for loop

```
for(i=0; i<n; i++) {
    pf("%d", n) -> n
}
```

n+1 times condition is checked and n times pf is executed

Q

2. $P(n) = T(n-1)$

$$T(n) = \begin{cases} 1 \\ T(n-1) + n \end{cases}$$

Substitute into (1)

$$T(n) = [T(n-2) + (n-1)] + n$$

$$T(n) = T[(n-2) + (n-1)] + n$$

Teacher's Signature.....

$T(n) = 1 + 2 + \cdots + (n-1) + n$

$= \frac{1+n+1}{2}$

$= O(n^2)$

Q2 $f(n) = (n/2) + 1$

$a = 2 \quad b = 2 \quad k = 0$

$k = 0$, we can apply master theorem

```
Test (int n)
{
  if (n > 1)
  {
    printf ("%d", n);
    Test (n/2);
  }
}
```

PAGE NO.
DATE :

$T(n) = 2T(n/2 + n^2)$ ---(1)

$T(n) = 2T(n/2) + n$ ---(2)

$T(n/2 + n/2)$ --(3)

substitute (3) in (2)

$T(n) = 2^2[2 + n/8 + n/4] + 2n$

So, $T(n) = 2^3 + (n/2^3) + 3n$ -(3)

Assume $T(1) = 1$

let $n/2 = 1$ $n = 2$ ...(6)

$\log 2^n = \log 1 = 1 \cdot \log 1$

$= 1 \times 1$

$= (1)$

$\log_2 n = ?$ ... (7)

PAGE NO.
DATE :

Put i= (6) and (2) :- (5)

$= \Theta(n \log n)$

So, $t(n) = O(n \log_2 n^n) \approx \Theta(n \log_2 n^n) \approx \Omega(n \log_2 n^n)$

Q. Use Recursive tree method to get asymptotic upper bound for recurrence

$t(n) = 2[2 + (n-2) + 1] + 1$

$= 2[2 + (n-2) + 2] + 1$

$t(n) = 2^2 + (n-2) + 2 + 1$

$t(n) = 2^2 T(n-1) + 1$

$t(n) = 2^2 T(0) + 1 + 2 + 2^2 \cdots$

$\cdots 2^{k-1}$

Teacher's Signature.....

PAGE NO.
DATE :

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

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

$= 2^{n+1}$

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

$f(n) = O(2^n)$

Q2 $f(n) = n + \lceil n/2 \rceil + c(n)$ where $c(n)$ is constant can provide asymptotic bound.

Ans $T(n) = n^2 \left[n + n/2 \right] + \left\lceil n/2 \right\rceil + n$

$T(n) = n + 2n + 2^2 n + n^3 \, T(n/3)$

$2n + 2n + 2^n n + \ldots + 2^{k-1} n^k$

$+ (n/2^k)$

PAGE NO.
DATE :

$n \log n = n^2$ ...(2)

$n\left[a\frac{(\log) n - 1}{a-1}\right] + n^2 + (1)$

$= n(n-1) + n^2 + (1 \Rightarrow O(n^2)$

PAGE NO.
DATE :

Assignment - 3

Q) Insertion sort on (31, 41, 59, 26, 41, 58)

[31, 41, 59, 26, 41, 58]
 0 1 2 3 4 5

a[j+1] = key
a[0] = 41

i j
[31, 41, 59, 26, 41, 58]
 0 1 2 3 4 5

a[j+1] = key

j = i

i j
[31, 41, 59, 59, 41, 58]
 0 1 2 3 4 5

j = 2 key = 4

i ≥ 0

a[j+1] = key

a[3] = 41

[26, 31, 41, 41, 59, 58]
 0 1 2 3 4

j = 3

j != 0 and a[j] > key

a[j+1] = key

a[4] = 58

[26, 31, 41, 41, 58, 59]

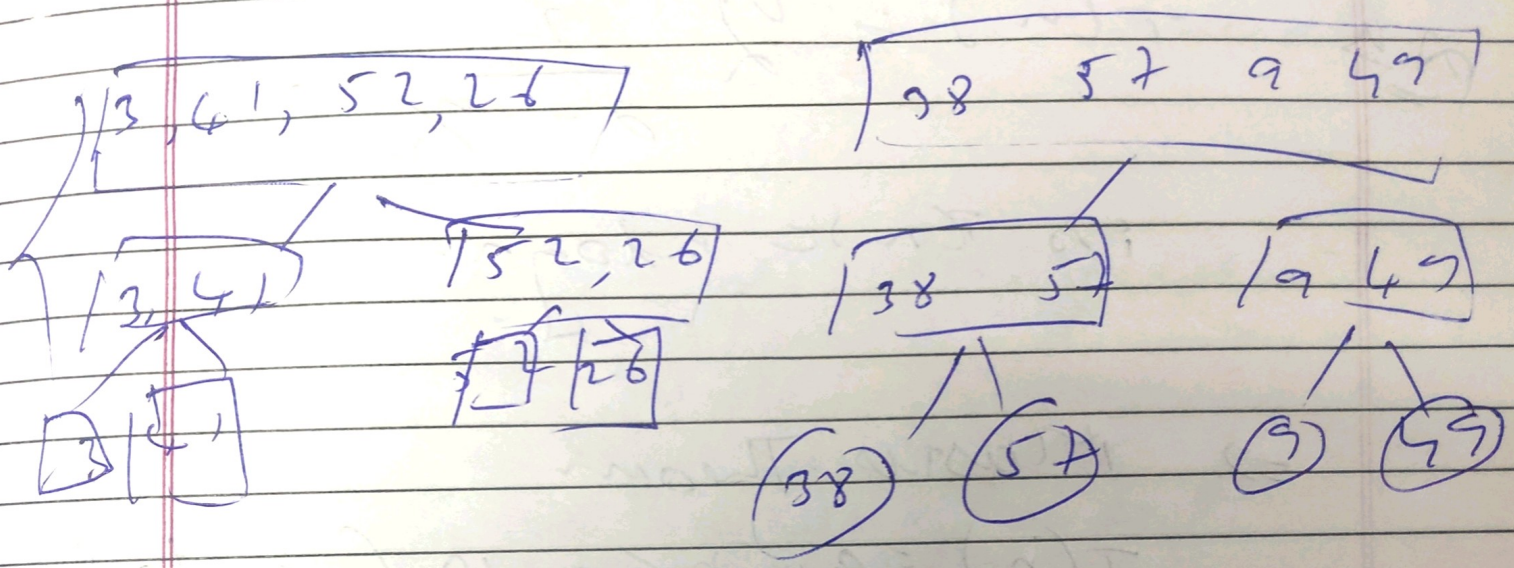
Q. Insertion Sort

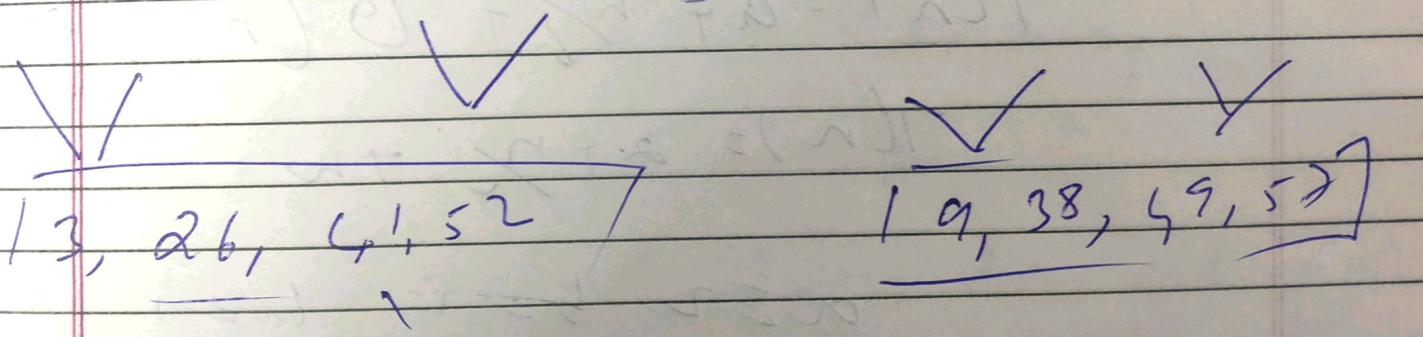
```
insertion_sort(int a[], int n)

int i, j, key;
for(int i = 1; i < n; i++) {
    key = a[i];
    j = i - 1;
    while(j >= 0 && a[j] < key)
    {
        a[j+1] = a[j];
        j--;
    }
    a[j+1] = key;
}
```

Q5 Merge Sort ex.

[3, 41, 52, 26, 38, 57, 9, 49]





[3, 9, 26, 38, 41, 49, 52, 57]

Q. Mathematical induction to show that which is exact order of recurrence solution of recurrence

$$T(n) = \begin{cases} 2 & \\ 2T(n/2) + n & \end{cases}$$

A ⇒ is $T(n) = n \log n$.

⇒ 1st Master Theorem

$T(n) = aT(n/b) + \Theta(n^k \log^p n)$

$T(n) = 2T(n/2) + n$

$a = 2 \quad b = 2 \quad k = 1 \quad p = 0$

$a > 1 \quad b > 1 \quad k \geq 0$

& $p > -1$

$T(n) = \Theta(n^{\log_b a}, \log^{p+1} n)$

$= \Theta(n \log_2 2 \log n)$

$= \Theta(n \log n)$

Teacher's Signature.....

Q7 Observe while loop of ins. sort. Can we use binary search to improve overall insertion sort?

Ans This loop has 2 purposes:

→ A linear search to scan through the sorted subarray to find proper position for key

→ This shifting of elements has a(n) time, due to which average case and worst case running time of insertion sort will be $O(n^2)$.

PAGE NO.
DATE :

Assignment - 4

PAGE NO.
DATE :

Q1 Define and explain following giving suitable examples:

(i) Graph

A set of items connected by edges. Each item is called a vertex.

Formally a graph is set of vertices and binary relation between vertices adjacency.

Graphs can be of 2 types

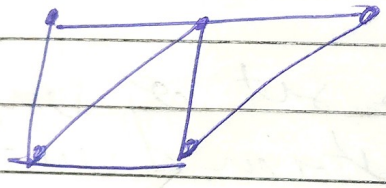
(i) Directed Graph.

A directed graph is a pair (V, E) where V is a finite set and E is binary relation on V.

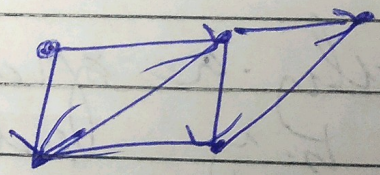
PAGE NO.
DATE :

① Undirected Graph

In a undirected graph, $G = (V, E)$ the edge set E consists of unordered pair of vertices.



Undirected

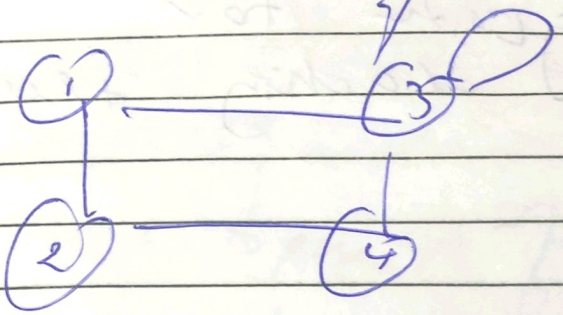


Directed

PAGE NO.
DATE :

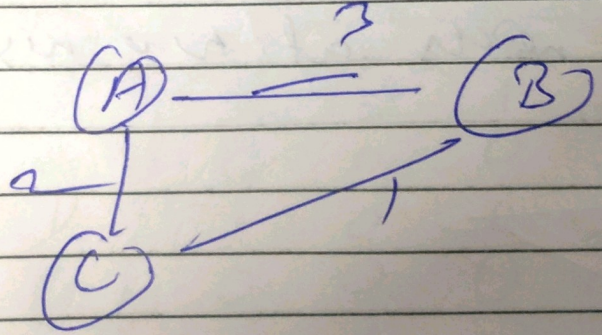
Loop

Loop is edge that connects to itself.



Q Weighted Graph

Ans A graph where each edge has numeric value called weight



(vi) Depth first

Depth first traversal of graph

→ It is an algorithm for traversing or searching tree or graph.

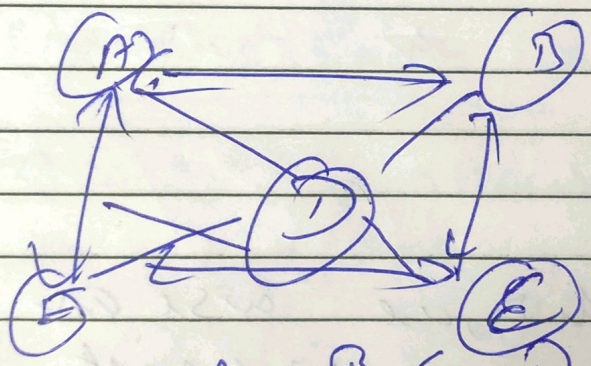
Breadth First Traversal

This an algorithm or searching or graph. It starts at the root and explored all of neighbour nodes at present depth before moving to nodes at the next depth levell.

PAGE NO.
DATE :

iv) Adjacency Matrix.

Ans) It is a sequential representation used to represent a finite graph. The elements of matrix indicate whether pairs of vertices are adjacent or not in a graph.



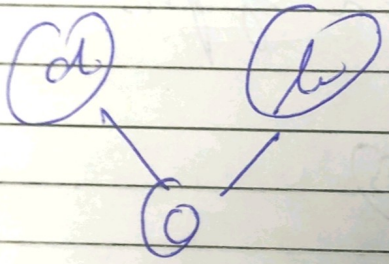
	A	B	C	D	E
A	0	1	0	1	1
B	1	0	1	1	0
C	0	0	0	1	1
D	1	0	1	0	1
E	0	0	0	0	0

PAGE NO.
DATE :

Adjacency list :

It is list of unordered list used to represent a finite graph.

Each unordered list within an adjacency list describes the set of neighbors of particular vertex in the graph.



In adjacency list, we use an array of a list to represent the graph.

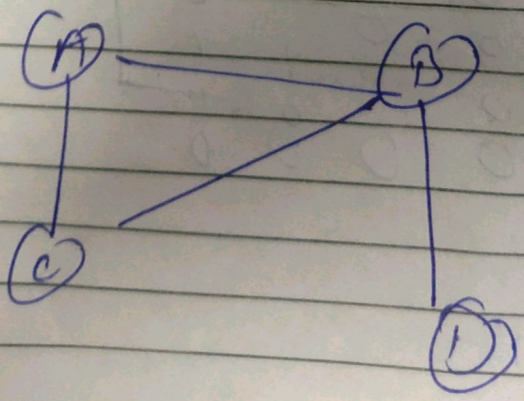
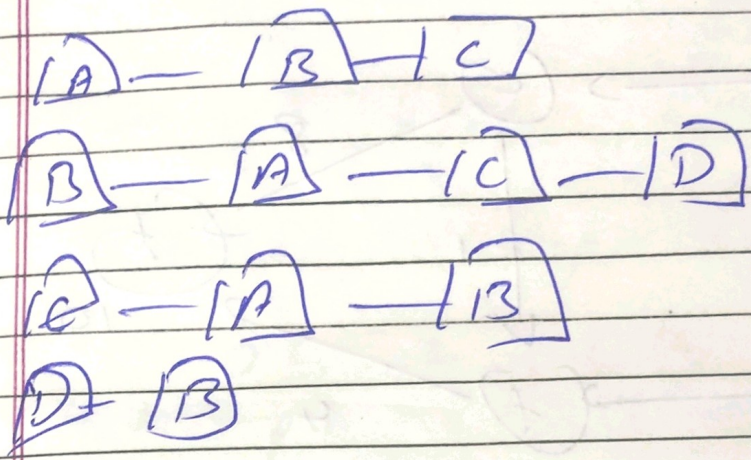


Fig Graph

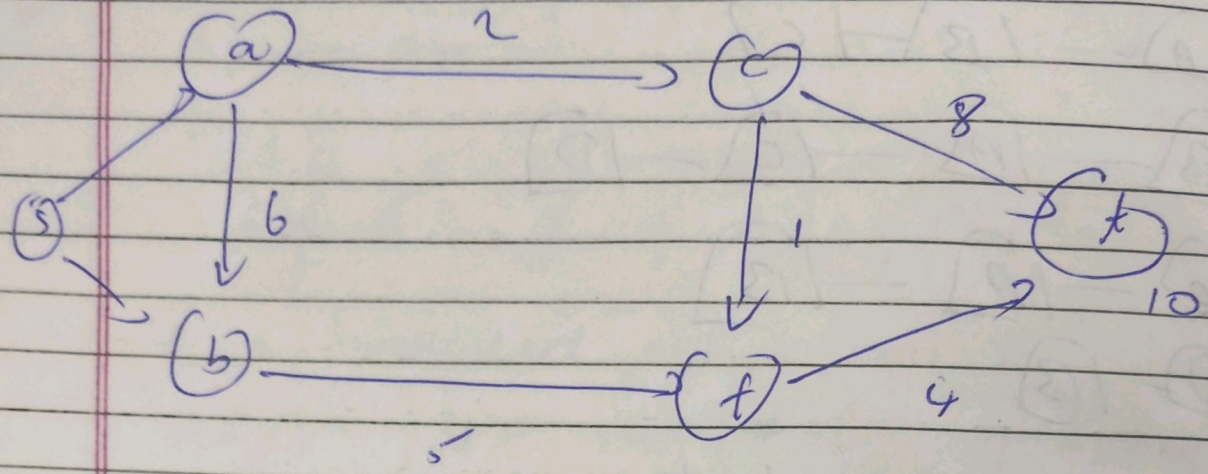


Adjacey list

Q Shortest Path Problem

It is problem of finding a path between two of vertices in graph such that sum of weights of its constituent edges is minized.

$$\delta(u,v) = \begin{cases} \min\{w(u,v) : \text{if this is path} \\ \infty \text{ otherwise.} \end{cases}$$



optimal solution

One of the key indicators that dynamic programming and greedy method might apply.

⇒ Dijkstra is greedy algorithm and Floyd Warshall algorithm is dynamic programming language.

Teacher's Signature.....

* Lemma

Given a directed graph
$w: E \to R$ let $p = \langle v_0, v, v_1, \ldots v_k \rangle$
such $0 \le i < j \le k$
let p_{ij} be a path of the vertex v_i to
vertex v_j of

Q) What is edge relaxation.

→ The edge relaxation is operation to calculate the leading cost of vertex wise.

FOR edge form vertex v to v,
$d[v] \; v(v,v) < dv$ is
satisfied, update $d(v)$ to
$d[v] + w(v,v)$.

PAGE NO.
DATE :

7.) DAG shortest path algorithm

by Relaxing the edge of weighted directed acyclic graph

$G = (V, E)$ according to topological sort of its vertices, we can compute shortest path form single source in $\Theta(V+E)$ time

1.) Topologically sort the vertices of G

2.) Initialize - Single Source (G, s)

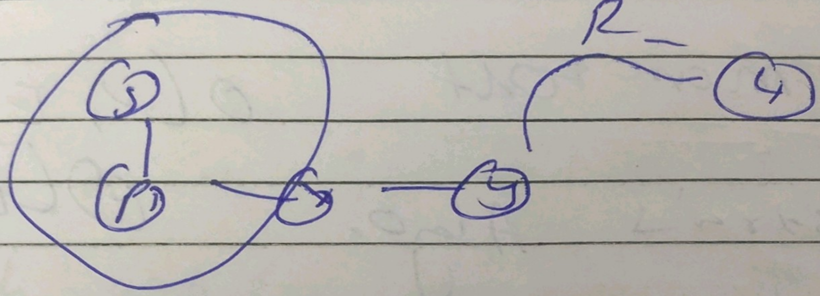
2.) For each vertex u taken in topologically sorted order.

4. For each adj. $v \in G.Adj[u]$

i. RELAX (u, v, w).

Dijkstra algorithm

On important algo given an on-
weighted directed source, j terminates
with $S(s,v)$ for all vertices.



$S \neq 0$ all

<u>proof all</u>

→ Begining of while loop v is added to
S,

so $v.d = \delta(s,v)$

→ $v \neq s$, $S \neq 0$ just v is added to S.

Q5 Execution Effect of Follow single source shortest path algorithm

→ Bell man FORD $O(V, E)$

→ Dijkstra's Algo. $O(E \log V)$

→ Time Complexity $O(E + (V) \log$

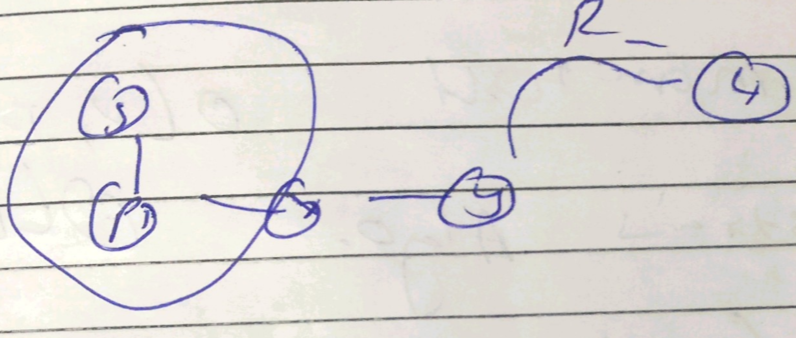
Q6 Red and Black Tree. state Advantages

Ans It is a Binary search tree with one extra bit of storage per node. Its colors can be either RED or black. By constraining the node colors on any simple path roots

PAGE NO.
DATE :

Dijkstra algorithm

On input Dijkstra given an un-weighted vertex source s, terminates with S(s,0) for all vertices.



S ≠ 0 all

proof all

-> Begining of while loop v is added to S,

so vid = δ(s,v)

-> v ≠ s, S ≠ 0 just v is added to S.

Q5 Execution Effect of Follow input tree structure path algorithm

→ Bell man FORT O(V, E)

→ Dijkstra's Algo. O(E log V)

→ Time Complexity O((E + (V) log)

Q6 Red and Black Tree. state Advantage

Ans → It is a Binary search tree with one extra bit of storage per node. Its colours can be either RED or black. By constraining the node colors on any simple path roots

PAGE NO.
DATE :

Advantages

→ Red black tree are useful when we need insertion and deletion relatively frequent

→ They have relatively constant variety of services

PAGE NO.
DATE :

Assignment - 5

DAA

PAGE NO.
DATE :

(Q1) Greedy choice Property

→ It is a property that works well on optimization problems.

→ Greedy choice property is global optimal can be reached by selecting the local optimal.

(Q2) Explain sequence of steps to be carried out while designing a greedy algorithm.

(1) Determine optimal substructure of problem

(2) Develop recursive solution

(3) Show if we make greedy choice, only one subproblem remains

Teacher's Signature.....

PAGE NO.
DATE :

- Develop recursive algorithm
- Convert to iterative algorithm

03 show the problem of finding optimal prefix code using HUFFMAN prefix code

→ we show that the problem of determining optimal prefix code

- exhibits greedy choice property
- exhibits optimal substructure property.

PAGE NO.
DATE :

Correctness of HUFFMANS

- Tree T represents an arbitrary optimal code.

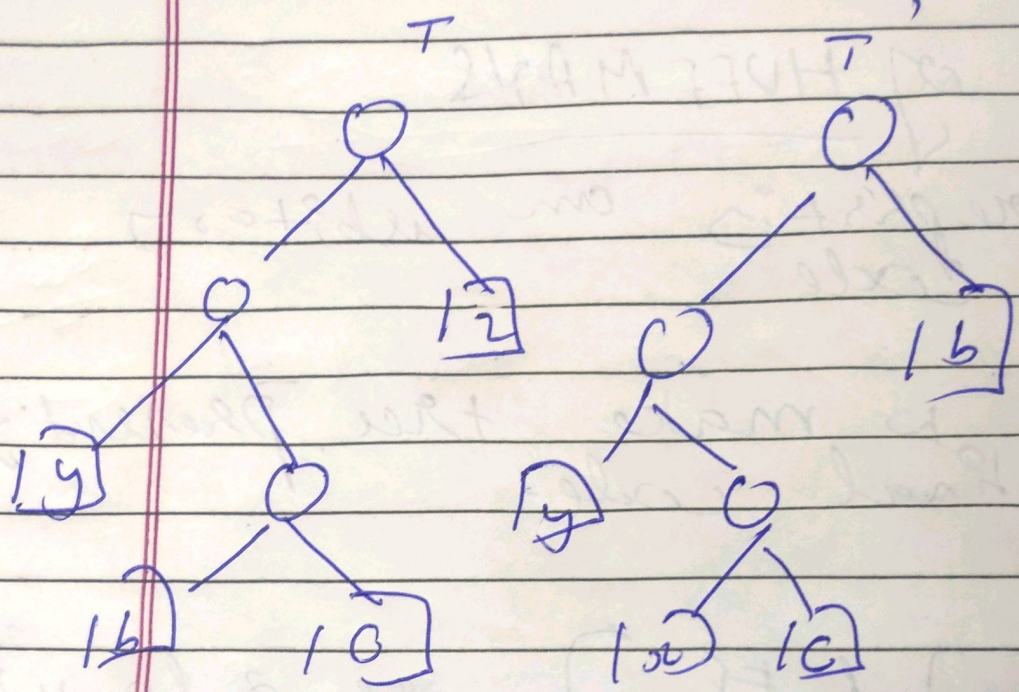
Modify T to make tree represents another optimal code.

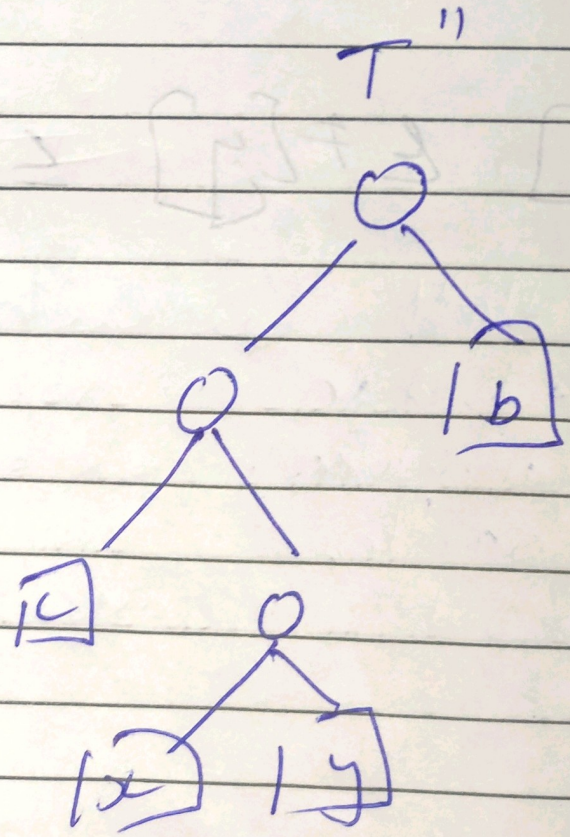
- Since, F[x] & F[y] are 2 lowest leaf freqy.

$$F[x] \leq F[b] \leq F[y] \leq F[c]$$

$T \Rightarrow T'$

$T' \Rightarrow T$


T
T'
z
y
b
a
b
c


T''
b
c
x
y

Difference in cost between T and T' is $B(T) = B(T') = f[b] - f[x]$

$f[a] - f[x] \geq 0$ and $d + [b] \geq d(T)$

$B[T') \leq B(T)$

Since T is optimal,

$B[T') = B(T) = T'$ is also optimal.

Lemma 2

Consider a character x & y that appears as sibling leaves in optimal T and let z be their parent.

$f[z] = f[(x)] + f[y]$

Q. Exercise

16.2)

Ans

⇒ If we have x of freq = b. freq

we then a is used to count frequency

Exercise

16.1.)

→ Assume that the inputs have been sorted as in equation 16.1. Compare how much of your solution is like that of greedy activity selection

DATE :

16.1.3

Ans As a counter example to optimality of greedily selecting the shortest.

Suppose our activity times are {(1, 9), (8, 11), (10, 20)}

Then picking the shortest first, we have to eliminate the other two, while if we picked the other two instead, we would have 2 tasks, not one.

- Suppose our activity time is {(1, 10), (2, 3), (4, 5)} we only have single activity (1, 10) optimal solution would be to pick two other activities

Q-6 State and prove overlapping suffix lemma.

→ Suppose that x, y and z are strings such that x ⊐ z and y ⊐ z.

→ If |x| ≤ |y|, then x ⊐ y

→ If |x| ≥ |y|, then y ⊐ x.

→ If |x| = |y| then x = y.

Naive String Matcher.

n = T. length
m = p. length

for s = 0 to n - m

if P[1...m] = T[s+1...s+m]

print "pattern with shift" s;

Q. Exercises

222.-1

Working modulo $q = 11$, how many spurious hits does Robin Karp matcher encounter in text

$T = 3141592653589793$ when looking for pattern $P = 26$?

Ans $|2, 15, 52, 92| = 3$

-2 How would you extend the method for searching a string of any one out of k patterns ? Start all k patterns have same length

Ans Truncation.

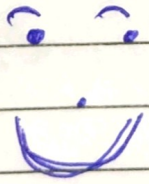
Q32.2-3

Show how to extend the Rabin Karp method to handle the problem of looking for a given $m \times m$ pattern in an $n \times n$ array.

Ans Calculate the hashes in each column just like the Rabin Karp in 1-Dimension.

Now, treat the hashes in each row as characters and hashing again.

The End



PRACTICAL

```
/*
================================================================
================
 *
 * Roll No: 30
 *
 * File:      ActivitySelection.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ========================================================== */

import java.util.*;
import java.lang.*;
import java.io.*;

public class ActivitySelection {
    public static void main(String[] args) {
        int start[] = { 1, 4, 3, 5 };
        int finish[] = { 2, 9, 5, 7 };
        int totalProcesses = start.length;
        sortArray(start, finish, totalProcesses);
        printMaxActivities(start, finish, totalProcesses);
    }

    public static void sortArray(int start[], int finish[], int
totalProcesses) {
        for (int i = 0; i < totalProcesses - 1; i++) {
            for (int j = 0; j < totalProcesses - i - 1; j++) {
                if (finish[j] > finish[j + 1]) {
                    int tempFinish = finish[j];
                    int tempStart = start[j];
                    finish[j] = finish[j + 1];
                    start[j] = start[j + 1];
                    finish[j + 1] = tempFinish;
                    start[j + 1] = tempStart;
```

```
                }
            }
        }
    }

    public static void printMaxActivities(int start[], int
finish[], int totalProcesses) {
        int i, j, count = 0;

        System.out.println("Following activities are selected :
");
        i = 0;
        System.out.println(++count + " : (Start : " + start[i] +
", End : " + finish[i] + ")");
        for (j = 1; j < totalProcesses; j++) {
            if (start[j] >= finish[i]) {
                System.out.println(++count + " : (Start : " +
start[j] + ", End : " + finish[j] + ")");
                i = j;
            }
        }
    }
} /* ============================================================
 *
 * Roll No: 30
 *
 * File:      GraphBFS.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

public class GraphBFS
{

    private Queue<Node> queue;
    static ArrayList<Node> nodes=new ArrayList<Node>();
    static class Node
    {
```

```
    int data;
    boolean visited;
    List<Node> neighbours;

    Node(int data)
    {
        this.data=data;
        this.neighbours=new ArrayList<>();

    }
    public void addneighbours(Node neighbourNode)
    {
        this.neighbours.add(neighbourNode);
    }
    public List<Node> getNeighbours() {
        return neighbours;
    }
    public void setNeighbours(List<Node> neighbours) {
        this.neighbours = neighbours;
    }
}

public GraphBFS()
{
    queue = new LinkedList<Node>();
}

public void bfs(Node node)
{
    queue.add(node);
    node.visited=true;
    while (!queue.isEmpty())
    {

        Node element=queue.remove();
        System.out.print(element.data + " ");
        List<Node> neighbours=element.getNeighbours();
        for (int i = 0; i < neighbours.size(); i++) {
            Node n=neighbours.get(i);
            if(n!=null && !n.visited)
            {
                queue.add(n);
                n.visited=true;
```

```
                }
            }

        }
    }

    public static void main(String arg[])
    {

        Node node40 =new Node(40);
        Node node10 =new Node(10);
        Node node20 =new Node(20);
        Node node30 =new Node(30);
        Node node60 =new Node(60);
        Node node50 =new Node(50);
        Node node70 =new Node(70);

        node40.addneighbours(node10);
        node40.addneighbours(node20);
        node10.addneighbours(node30);
        node20.addneighbours(node10);
        node20.addneighbours(node30);
        node20.addneighbours(node60);
        node20.addneighbours(node50);
        node30.addneighbours(node60);
        node60.addneighbours(node70);
        node50.addneighbours(node70);
        System.out.println("The BFS traversal of the graph is ");
        GraphBFS bfsExample = new GraphBFS();
        bfsExample.bfs(node40);

    }
} /* ============================================================
 *
 * Roll No: 30
 *
 * File:      GraphDFS.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 import java.util.ArrayList;
```

```java
import java.util.List;
import java.util.Stack;

public class GraphDFS
{

    static class Node
    {
        int data;
        boolean visited;
        List<Node> neighbours;

        Node(int data)
        {
            this.data=data;
            this.neighbours=new ArrayList<>();

        }
        public void addneighbours(Node neighbourNode)
        {
            this.neighbours.add(neighbourNode);
        }
        public List<Node> getNeighbours() {
            return neighbours;
        }
        public void setNeighbours(List<Node> neighbours) {
            this.neighbours = neighbours;
        }
    }

    // Recursive DFS
    public  void dfs(Node node)
    {
        System.out.print(node.data + " ");
        List<Node> neighbours=node.getNeighbours();
        node.visited=true;
        for (int i = 0; i < neighbours.size(); i++) {
            Node n=neighbours.get(i);
            if(n!=null && !n.visited)
            {
                dfs(n);
            }
        }
```

```
}

// Iterative DFS using stack
public  void dfsUsingStack(Node node)
{
    Stack<Node> stack=new  Stack<Node>();
    stack.add(node);
    while (!stack.isEmpty())
    {
        Node element=stack.pop();
        if(!element.visited)
        {
            System.out.print(element.data + " ");
            element.visited=true;
        }

        List<Node> neighbours=element.getNeighbours();
        for (int i = 0; i < neighbours.size(); i++) {
            Node n=neighbours.get(i);
            if(n!=null && !n.visited)
            {
                stack.add(n);
            }
        }
    }
}

public static void main(String arg[])
{

    Node node40 =new Node(40);
    Node node10 =new Node(10);
    Node node20 =new Node(20);
    Node node30 =new Node(30);
    Node node60 =new Node(60);
    Node node50 =new Node(50);
    Node node70 =new Node(70);

    node40.addneighbours(node10);
    node40.addneighbours(node20);
    node10.addneighbours(node30);
    node20.addneighbours(node10);
    node20.addneighbours(node30);
```

```
        node20.addneighbours(node60);
        node20.addneighbours(node50);
        node30.addneighbours(node60);
        node60.addneighbours(node70);
        node50.addneighbours(node70);

        GraphDFS dfsExample = new GraphDFS();

        System.out.println("The DFS traversal of the graph using
stack ");
        dfsExample.dfsUsingStack(node40);

        System.out.println();

        // Resetting the visited flag for nodes
        node40.visited=false;
        node10.visited=false;
        node20.visited=false;
        node30.visited=false;
        node60.visited=false;
        node50.visited=false;
        node70.visited=false;

        System.out.println("The DFS traversal of the graph using
recursion ");
        dfsExample.dfs(node40);
    }
} /* ============================================================
 *
 * Roll No: 30
 *
 * File:      Graph.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 import java.util.*;

class Graph<T> {
    private Map<T, List<T>> map = new HashMap<>();

    public void addNewVertex(T s) {
        map.put(s, new LinkedList<T>());
```

```
    }

    public void addNewEdge(T source, T destination, boolean
bidirectional) {
        //
        if (!map.containsKey(source))
            addNewVertex(source);
        if (!map.containsKey(destination))
            addNewVertex(destination);
        map.get(source).add(destination);
        if (bidirectional == true) {
            map.get(destination).add(source);
        }
    }

    // the method counts the number of vertices
    public void countVertices() {
        System.out.println("Total number of vertices: " +
map.keySet().size());
    }

    // the method counts the number of edges
    public void countEdges(boolean bidirection) {
        // variable to store number of edges
        int count = 0;
        for (T v : map.keySet()) {
            count = count + map.get(v).size();
        }
        if (bidirection == true) {
            count = count / 2;
        }
        System.out.println("Total number of edges: " + count);
    }

    // checks a graph has vertex or not
    public void containsVertex(T s) {
        if (map.containsKey(s)) {
            System.out.println("The graph contains " + s + " as a
vertex.");
        } else {
            System.out.println("The graph does not contain " + s +
" as a vertex.");
        }
```

```
    }

    // checks a graph has edge or not
    // where s and d are the two parameters that represent
source(vertex) and
    // destination (vertex)
    public void containsEdge(T s, T d) {
        if (map.get(s).contains(d)) {
            System.out.println("The graph has an edge between " +
s + " and " + d + ".");
        } else {
            System.out.println("There is no edge between " + s + "
and " + d + ".");
        }
    }

    // prints the adjacencyS list of each vertex
    // here we have overridden the toString() method of the
StringBuilder class
    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();
        // foreach loop that iterates over the keys
        for (T v : map.keySet()) {
            builder.append(v.toString() + ": ");
            // foreach loop for getting the vertices
            for (T w : map.get(v)) {
                builder.append(w.toString() + " ");
            }
            builder.append("\n");
        }
        return (builder.toString());
    }
}

// creating a class in which we have implemented the driver code
public class GraphImplementation {
    public static void main(String args[]) {
        // creating an object of the Graph class
        Graph graph = new Graph();
        // adding edges to the graph
        graph.addNewEdge(0, 1, true);
        graph.addNewEdge(0, 4, true);
```

```
        graph.addNewEdge(1, 2, true);
        graph.addNewEdge(1, 3, false);
        graph.addNewEdge(1, 4, true);
        graph.addNewEdge(2, 3, true);
        graph.addNewEdge(2, 4, true);
        graph.addNewEdge(3, 0, true);
        graph.addNewEdge(2, 0, true);
        // prints the adjacency matrix that represents the graph
        System.out.println("Adjacency List for the graph:\n" +
graph.toString());
        // counts the number of vertices in the graph
        graph.countVertices();
        // counts the number of edges in the graph
        graph.countEdges(true);
        // checks whether an edge is present or not between the
two specified vertices
        graph.containsEdge(3, 4);
        graph.containsEdge(2, 4);
        // checks whether vertex is present or not
        graph.containsVertex(3);
        graph.containsVertex(5);
    }
} /* ============================================================
 *
 * Roll No: 30
 *
 * File:      HeapSort.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 import java.util.*;

public class HeapSort {
    public static void main(String[] args) {
        int arr[] = new int[10];
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter size of array : ");
        int size = sc.nextInt();

        System.out.println("Enter elements of array: ");
        for (int i = 0; i < size; i++) {
```

```
            arr[i] = sc.nextInt();
        }
        s.heap(arr, size);
        System.out.print("\nApplying Heap Sort Algorithm....\
nAfter sorting...");
        s.display(arr, size);
    }

    public static void heap(int arr[], int size) {
        for (int i = size / 2 - 1; i >= 0; i--)
            heapify(arr, size, i);

        for (int i = size - 1; i > 0; i--) {
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            heapify(arr, i, 0);
        }
    }

    public static void heapify(int arr[], int size, int i) {
        int largest = i;
        int l = 2 * i + 1;
        int r = 2 * i + 2;

        if (l < size && arr[l] > arr[largest])
            largest = l;

        if (r < size && arr[r] > arr[largest])
            largest = r;

        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;

            heapify(arr, size, largest);
        }
    }

    public static void display(int arr[], int size) {
        System.out.println("\nArray: ");
```

```
        for (int i = 0; i < size; i++) {
            System.out.print(" " + arr[i]);
        }
        System.out.print("\n");
    }
}
 /* ============================================================
 *
 * Roll No: 30
 *
 * File:      Huffman.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 import java.util.PriorityQueue;
import java.util.Scanner;
import java.util.Comparator;

// node class is the basic structure
// of each node present in the Huffman - tree.
class HuffmanNode {

    int data;
    char c;

    HuffmanNode left;
    HuffmanNode right;
}

// comparator class helps to compare the node
// on the basis of one of its attribute.
// Here we will be compared
// on the basis of data values of the nodes.
class MyComparator implements Comparator<HuffmanNode> {
    public int compare(HuffmanNode x, HuffmanNode y)
    {

        return x.data - y.data;
    }
}

public class Huffman{
```

```java
// recursive function to print the
// huffman-code through the tree traversal.
// Here s is the huffman - code generated.
public static void printCode(HuffmanNode root, String s)
{

    // base case; if the left and right are null
    // then its a leaf node and we print
    // the code s generated by traversing the tree.
    if (root.left
            == null
        && root.right
               == null
        && Character.isLetter(root.c)) {

        // c is the character in the node
        System.out.println(root.c + ":" + s);

        return;
    }

    // if we go to left then add "0" to the code.
    // if we go to the right add"1" to the code.

    // recursive calls for left and
    // right sub-tree of the generated tree.
    printCode(root.left, s + "0");
    printCode(root.right, s + "1");
}

// main function
public static void main(String[] args)
{

    Scanner s = new Scanner(System.in);

    // number of characters.
    int n = 6;
    char[] charArray = { 'a', 'b', 'c', 'd', 'e', 'f' };
    int[] charfreq = { 5, 9, 12, 13, 16, 45 };

    // creating a priority queue q.
```

```
        // makes a min-priority queue(min-heap).
        PriorityQueue<HuffmanNode> q
            = new PriorityQueue<HuffmanNode>(n, new
MyComparator());

        for (int i = 0; i < n; i++) {

            // creating a Huffman node object
            // and add it to the priority queue.
            HuffmanNode hn = new HuffmanNode();

            hn.c = charArray[i];
            hn.data = charfreq[i];

            hn.left = null;
            hn.right = null;

            // add functions adds
            // the huffman node to the queue.
            q.add(hn);
        }

        // create a root node
        HuffmanNode root = null;

        // Here we will extract the two minimum value
        // from the heap each time until
        // its size reduces to 1, extract until
        // all the nodes are extracted.
        while (q.size() > 1) {

            // first min extract.
            HuffmanNode x = q.peek();
            q.poll();

            // second min extarct.
            HuffmanNode y = q.peek();
            q.poll();

            // new node f which is equal
            HuffmanNode f = new HuffmanNode();

            // to the sum of the frequency of the two nodes
```

```
            // assigning values to the f node.
            f.data = x.data + y.data;
            f.c = '-';

            // first extracted node as left child.
            f.left = x;

            // second extracted node as the right child.
            f.right = y;

            // marking the f node as the root node.
            root = f;

            // add this node to the priority-queue.
            q.add(f);
        }

        // print the codes by traversing the tree
        printCode(root, "");
    }
} /* ============================================================
 *
 * Roll No: 30
 *
 * File:      InsertionSort.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 import java.util.*;

public class InsertionSort {
    public static void main(String[] args) {
        int arr[] = new int[10];
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter size of array : ");
        int size = sc.nextInt();

        System.out.println("Enter elements of array: ");
        for (int i = 0; i < size; i++) {
            arr[i] = sc.nextInt();
        }
```

```
        s.insertionsort(arr, size);
        System.out.print("\nApplying Insertion Sort Algorithm....\
nAfter sorting...");
        s.display(arr, size);
    }

    public static void insertionsort(int arr[], int size) {

        for (int i = 1; i < size; ++i) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }

    }

    public static void display(int arr[], int size) {
        System.out.println("\nArray: ");
        for (int i = 0; i < size; i++) {
            System.out.print(" " + arr[i]);
        }
        System.out.print("\n");
    }
}
 /* ============================================================
 *
 * Roll No: 30
 *
 * File:      LongestCommonSubsequence.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 public class LongestCommonSubsequence{

    /* Returns length of LCS for X[0..m-1], Y[0..n-1] */
    int lcs(char[] X, char[] Y, int m, int n)
    {
        if (m == 0 || n == 0)
```

```
            return 0;
        if (X[m - 1] == Y[n - 1])
            return 1 + lcs(X, Y, m - 1, n - 1);
        else
            return max(lcs(X, Y, m, n - 1), lcs(X, Y, m - 1, n));
    }

    /* Utility function to get max of 2 integers */
    int max(int a, int b)
    {
        return (a > b) ? a : b;
    }

    public static void main(String[] args)
    {
        LongestCommonSubsequence lcs = new
LongestCommonSubsequence();
        String s1 = "AGGTAB";
        String s2 = "GXTXAYB";

        char[] X = s1.toCharArray();
        char[] Y = s2.toCharArray();
        int m = X.length;
        int n = Y.length;

        System.out.println("Length of LCS is"
                           + " " + lcs.lcs(X, Y, m, n));
    }
} /* ============================================================
 *
 * Roll No: 30
 *
 * File:      MatrixChainMultiplication.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 //MatrixChainMultiplication program
public class MatrixChainMultiplication {
    // Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
    static int MatrixChainOrder(int p[], int n)
    {
        /* For simplicity of the program, one extra row and one
```

```
        extra column are allocated in m[][].  0th row and 0th
        column of m[][] are not used */
        int m[][] = new int[n][n];

        int i, j, k, L, q;

        /* m[i, j] = Minimum number of scalar multiplications
needed
        to compute the matrix A[i]A[i+1]...A[j] = A[i..j] where
        dimension of A[i] is p[i-1] x p[i] */

        // cost is zero when multiplying one matrix.
        for (i = 1; i < n; i++)
            m[i][i] = 0;

        // L is chain length.
        for (L = 2; L < n; L++) {
            for (i = 1; i < n - L + 1; i++) {
                j = i + L - 1;
                if (j == n)
                    continue;
                m[i][j] = Integer.MAX_VALUE;
                for (k = i; k <= j - 1; k++) {
                    // q = cost/scalar multiplications
                    q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] *
p[j];

                    if (q < m[i][j])
                        m[i][j] = q;
                }
            }
        }

        return m[1][n - 1];
    }

    // Driver program to test above function
    public static void main(String args[])
    {
        int arr[] = new int[] { 1, 2, 3, 4 };
        int size = arr.length;

        System.out.println("Minimum number of multiplications is "
                           + MatrixChainOrder(arr, size));
```

```
    }
} /* ============================================================
 *
 * Roll No: 30
 *
 * File:      MatrixMul.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 import java.util.*;
class MatrixMul
{
        public static void main(String args[])
        {
                int r1, r2,c1,c2,i,j,k,sum;
                Scanner in = new Scanner(System.in);

                System.out.println("Enter the number of rows of
matrix1");
                r1 = in.nextInt();

                System.out.println("Enter the number columns of
matrix 1");
                c1 = in.nextInt();
                System.out.println("Enter the number of rows of
matrix2");
                r2 = in.nextInt();

                System.out.println("Enter the number of columns of
matrix 2");
                c2 = in.nextInt();

                if(c1==r2)
                {

                int mat1[][] = new int[r1][c1];
                int mat2[][] = new int[r2][c2];
                int res[][] = new int[r1][c2];

                System.out.println("Enter the elements of
matrix1");
```

```
                for ( i= 0 ; i < r1 ; i++ )
                {

                for ( j= 0 ; j < c1 ;j++ )
                mat1[i][j] = in.nextInt();

                }
                System.out.println("Enter the elements of
matrix2");

                for ( i= 0 ; i < r2 ; i++ )
                {

                for ( j= 0 ; j < c2 ;j++ )
                mat2[i][j] = in.nextInt();

                }

                System.out.println("\n\noutput matrix:-");
                for ( i= 0 ; i < r1 ; i++ )

                for ( j= 0 ; j <c2;j++)
                {
                sum=0;
                for ( k= 0 ; k <r2;k++ )
                {
                sum +=mat1[i][k]*mat2[k][j] ;

                }
                res[i][j]=sum;
                }
                for ( i= 0 ; i < r1; i++ )
                {
                for ( j=0 ; j < c2;j++ )
                System.out.print(res[i][j]+" ");

                System.out.println();
                }
                }
                else
                System.out.print("multipication does not exist ");
        }
```

```java
} /* ============================================================
 *
 * Roll No: 30
 *
 * File:      MergeSort.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 import java.util.*;

public class MergeSort {
    public static void main(String[] args) {
        int arr[] = new int[10];
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter size of array : ");
        int size = sc.nextInt();

        System.out.println("Enter elements of array: ");
        for (int i = 0; i < size; i++) {
            arr[i] = sc.nextInt();
        }
        s.mergesort(arr, 0, size - 1);
        System.out.print("\nApplying Merge Sort Algorithm....\
nAfter sorting...");
        s.display(arr, size);
    }

    public static void mergesort(int arr[], int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
            mergesort(arr, l, m);
            mergesort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    public static void display(int arr[], int size) {
        System.out.println("\nArray: ");
        for (int i = 0; i < size; i++) {
            System.out.print(" " + arr[i]);
        }
```

```
        System.out.print("\n");
    }
}
 /* ============================================================
 *
 * Roll No: 30
 *
 * File:      NaiveStringMatching.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 import java.util.Scanner;

class NaiveStringMatching {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter Your String : ");
        String fullString = sc.nextLine();
        System.out.print("Enter Your String Pattern : ");
        String pattern = sc.nextLine();
        int n = fullString.length();
        int m = pattern.length();

        PatternMatcher patternMatcher = new PatternMatcher();
        int index = patternMatcher.searchPattern(fullString,
pattern); // calling pattern matching function
        System.out.println("n = " + n);
        System.out.println("m = " + m);
        System.out.println("n-m = " + (n - m));
        System.out.println("Pattern Found at index : " + index);
    }
} /* ============================================================
 *
 * Roll No: 30
 *
 * File:      PatternMatcher.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 import java.lang.String;
```

```java
public class PatternMatcher {

    // searchpattern function will return index only if
patternString matches else
    // return -1
    public int searchPattern(String fullString, String
patternString) {
        int index = -1, i, j;
        int n = fullString.length();
        int m = patternString.length();

        for (i = 0; i <= (n - m); i++) {
            for (j = 0; j < m; j++) {
                if (fullString.charAt(i - j) !=
patternString.charAt(j))
                    break;
            }
            if (j == m && index > 0)
                return index;
        }
        return index;
    }
}
 /* ============================================================
 *
 * Roll No: 30
 *
 * File:      rabinkarp_algo.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 public class rabinkarp_algo{
    // d is the number of characters in the input alphabet
    public final static int d = 256;

    /* pat -> pattern
        txt -> text
        q -> A prime number
    */
    static void search(String pat, String txt, int q)
    {
        int M = pat.length();
```

```
        int N = txt.length();
        int i, j;
        int p = 0; // hash value for pattern
        int t = 0; // hash value for txt
        int h = 1;

        // The value of h would be "pow(d, M-1)%q"
        for (i = 0; i < M - 1; i++)
            h = (h * d) % q;

        // Calculate the hash value of pattern and first
        // window of text
        for (i = 0; i < M; i++) {
            p = (d * p + pat.charAt(i)) % q;
            t = (d * t + txt.charAt(i)) % q;
        }

        // Slide the pattern over text one by one
        for (i = 0; i <= N - M; i++) {

            // Check the hash values of current window of text
            // and pattern. If the hash values match then only
            // check for characters on by one
            if (p == t) {
                /* Check for characters one by one */
                for (j = 0; j < M; j++) {
                    if (txt.charAt(i + j) != pat.charAt(j))
                        break;
                }

                // if p == t and pat[0...M-1] = txt[i,
i+1, ...i+M-1]
                if (j == M)
                    System.out.println("Pattern found at index " +
i);
            }

            // Calculate hash value for next window of text:
Remove
            // leading digit, add trailing digit
            if (i < N - M) {
                t = (d * (t - txt.charAt(i) * h) + txt.charAt(i +
M)) % q;
```

```
                // We might get negative value of t, converting it
                // to positive
                if (t < 0)
                    t = (t + q);
            }
        }
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        String txt = "AJ INK AJ";
        String pat = "AJ";
        int q = 101; // A prime number
        search(pat, txt, q);
    }
} /* ============================================================
 *
 * Roll No: 30
 *
 * File:      TravSale.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 class TravSale {

        static int tsp(int[][] graph, boolean[] v, int currPos,
int n, int count, int cost, int ans) {

                // System.out.println("count: "+count);

                // if last node is reached...get the min cost
                if (count == n && graph[currPos][0] > 0) {
                        // System.out.println("Final node:
"+graph[currPos][0]);
                        ans = Math.min(ans, cost + graph[currPos]
[0]);
                        return ans;
                }

                for (int i = 0; i < n; i++) {
```

```
                        /*
                         * System.out.println("currPos:
"+currPos); System.out.println("i: "+i);
                         * System.out.println("else value:
"+graph[currPos][i]);
                         */

                        if (v[i] == false && graph[currPos][i] >
0) {
                                v[i] = true; // mark node as
visited

                                ans = tsp(graph, v, i, n, count +
1, cost + graph[currPos][i], ans);

                                // System.out.println("\nans:
"+ans+"\n");

                                // System.out.println("node "+i+"
is visited: "+v[i]);

                                v[i] = false; // mark ith node as
unvisited
                        }

                }

                return ans;
        }

        public static void main(String[] args) {

                int n = 4; // number of nodes A,B,C,D

                // matrix
                int[][] graph = { { 0, 16, 11, 6 }, { 8, 0, 13, 16
}, { 4, 7, 0, 9 }, { 5, 12, 2, 0 } };

                // check node visited or not
                boolean[] v = new boolean[n];

                // fix starting node
```

```
                v[0] = true;

                // store total cost
                int ans = Integer.MAX_VALUE;

                // Find minimum weight of graph
                ans = tsp(graph, v, 0, n, 1, 0, ans);

                // print the total cost
                System.out.println("\nTotal minimum cost to visit
every node: " + ans);
        }
}

 /* ============================================================
 *
 * Roll No: 30
 *
 * File:      QuickSort.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

import java.util.*;

public class QuickSort {
    public static void main(String[] args) {
        int arr[] = new int[10];
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter size of array : ");
        int size = sc.nextInt();

        System.out.println("Enter elements of array: ");
        for (int i = 0; i < size; i++) {
            arr[i] = sc.nextInt();
        }
        s.quicksort(arr, 0, size - 1);
        System.out.print("\nApplying Quick Sort Algorithm....\
nAfter sorting...");
        s.display(arr, size);
    }
```

```
    public static void quicksort(int arr[], int low, int high) {

        if (low < high) {
            int pi = partition(arr, low, high);

            quicksort(arr, low, pi - 1);
            quicksort(arr, pi + 1, high);
        }
    }

    public static void display(int arr[], int size) {
        System.out.println("\nArray: ");
        for (int i = 0; i < size; i++) {
            System.out.print(" " + arr[i]);
        }
        System.out.print("\n");
    }
}

 /* ============================================================
 *
 * Roll No: 30
 *
 * File:      BellmanGraph.java
 * Copyright: by Ajinkya Rathod(ajinzrathod)
 *
 * ============================================================ */

 import java.util.*;
import java.lang.*;
import java.io.*;

// A class to represent a connected, directed and weighted graph
class BellmanGraph {
    // A class to represent a weighted edge in graph
    class Edge {
        int src, dest, weight;

        Edge() {
            src = dest = weight = 0;
        }
    };
```

```
    int V, E;
    Edge edge[];

    // Creates a graph with V vertices and E edges
    BellmanGraph(int v, int e) {
        V = v;
        E = e;
        edge = new Edge[e];
        for (int i = 0; i < e; ++i)
            edge[i] = new Edge();
    }

    // The main function that finds shortest distances from src
    // to all other vertices using Bellman-Ford algorithm. The
    // function also detects negative weight cycle
    void BellmanFord(Graph graph, int src) {
        int V = graph.V, E = graph.E;
        int dist[] = new int[V];

        // Step 1: Initialize distances from src to all other
        // vertices as INFINITE
        for (int i = 0; i < V; ++i)
            dist[i] = Integer.MAX_VALUE;
        dist[src] = 0;

        // Step 2: Relax all edges |V| - 1 times. A simple
        // shortest path from src to any other vertex can
        // have at-most |V| - 1 edges
        for (int i = 1; i < V; ++i) {
            for (int j = 0; j < E; ++j) {
                int u = graph.edge[j].src;
                int v = graph.edge[j].dest;
                int weight = graph.edge[j].weight;
                if (dist[u] != Integer.MAX_VALUE && dist[u] +
weight < dist[v])
                    dist[v] = dist[u] + weight;
            }
        }

        // Step 3: check for negative-weight cycles. The above
        // step guarantees shortest distances if graph doesn't
        // contain negative weight cycle. If we get a shorter
        // path, then there is a cycle.
```

```
        for (int j = 0; j < E; ++j) {
            int u = graph.edge[j].src;
            int v = graph.edge[j].dest;
            int weight = graph.edge[j].weight;
            if (dist[u] != Integer.MAX_VALUE && dist[u] + weight <
dist[v]) {
                System.out.println("Graph contains negative weight
cycle");
                return;
            }
        }
        printArr(dist, V);
    }

    // A utility function used to print the solution
    void printArr(int dist[], int V) {
        System.out.println("Vertex Distance from Source");
        for (int i = 0; i < V; ++i)
            System.out.println(i + "\t\t" + dist[i]);
    }

    // Driver method to test above function
    public static void main(String[] args) {
        int V = 5; // Number of vertices in graph
        int E = 8; // Number of edges in graph

        BellmanGraph graph = new BellmanGraph(V, E);

        // add edge 0-1 (or A-B in above figure)
        graph.edge[0].src = 0;
        graph.edge[0].dest = 1;
        graph.edge[0].weight = -1;

        // add edge 0-2 (or A-C in above figure)
        graph.edge[1].src = 0;
        graph.edge[1].dest = 2;
        graph.edge[1].weight = 4;

        // add edge 1-2 (or B-C in above figure)
        graph.edge[2].src = 1;
        graph.edge[2].dest = 2;
        graph.edge[2].weight = 3;
```

```
        // add edge 1-3 (or B-D in above figure)
        graph.edge[3].src = 1;
        graph.edge[3].dest = 3;
        graph.edge[3].weight = 2;

        // add edge 1-4 (or A-E in above figure)
        graph.edge[4].src = 1;
        graph.edge[4].dest = 4;
        graph.edge[4].weight = 2;

        // add edge 3-2 (or D-C in above figure)
        graph.edge[5].src = 3;
        graph.edge[5].dest = 2;
        graph.edge[5].weight = 5;

        // add edge 3-1 (or D-B in above figure)
        graph.edge[6].src = 3;
        graph.edge[6].dest = 1;
        graph.edge[6].weight = 1;

        // add edge 4-3 (or E-D in above figure)
        graph.edge[7].src = 4;
        graph.edge[7].dest = 3;
        graph.edge[7].weight = -3;

        graph.BellmanFord(graph, 0);
    }
}
```