

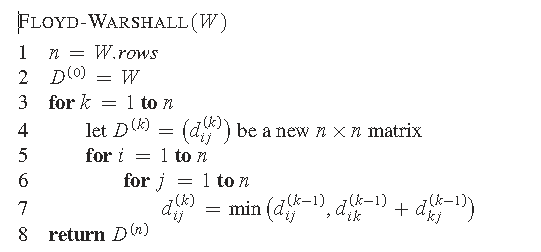
**Batch: B-4 Roll No.: 16010422234 Name: Chandana Ramesh Galgali**

**Experiment No.: 05**

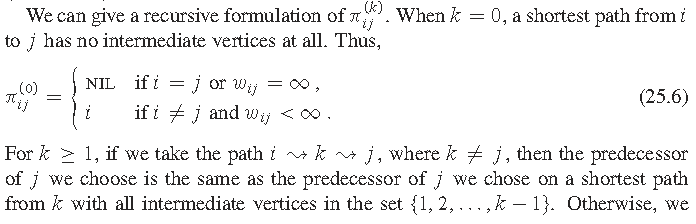
**Aim:** To Implement All pair shortest path Floyd-Warshall Algorithm using Dynamic programming approach and analyse its time Complexity.

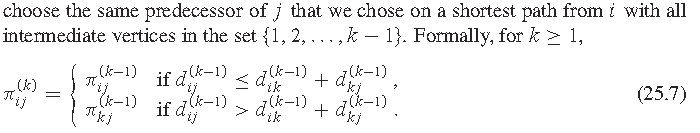


**Algorithm of Floyd-Warshall Algorithm:**



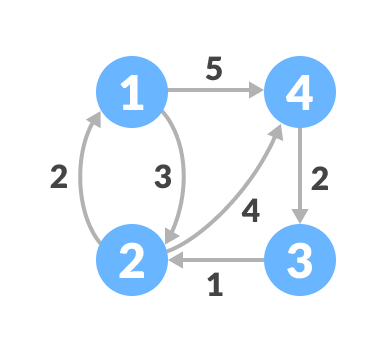
**Constructing Shortest Path:**





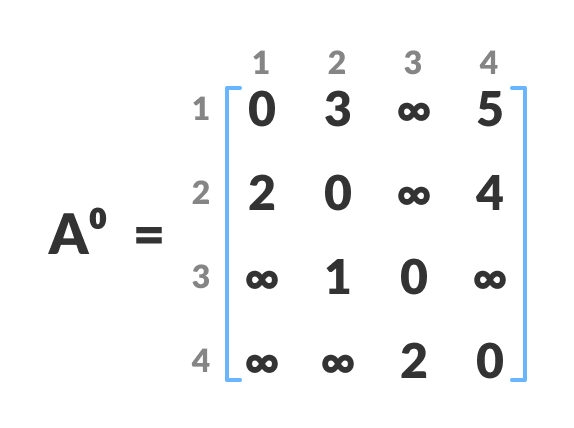
**Working of Floyd-Warshall Algorithm:**

Let the given graph be:



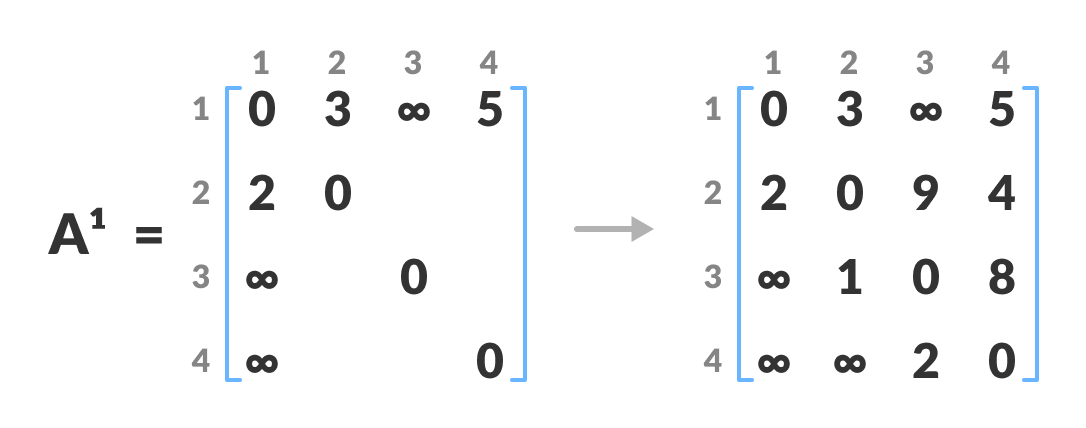
*Initial graph*

Follow the steps below to find the shortest path between all the pairs of vertices.

1. Create a matrix A0 of dimension n\*n where n is the number of vertices. The row and the column are indexed as i and j respectively. i and j are the vertices of the graph.  
   Each cell A[i][j] is filled with the distance from the ith vertex to the jth vertex. If there is no path from ith vertex to the jth vertex, the cell is left as infinity.

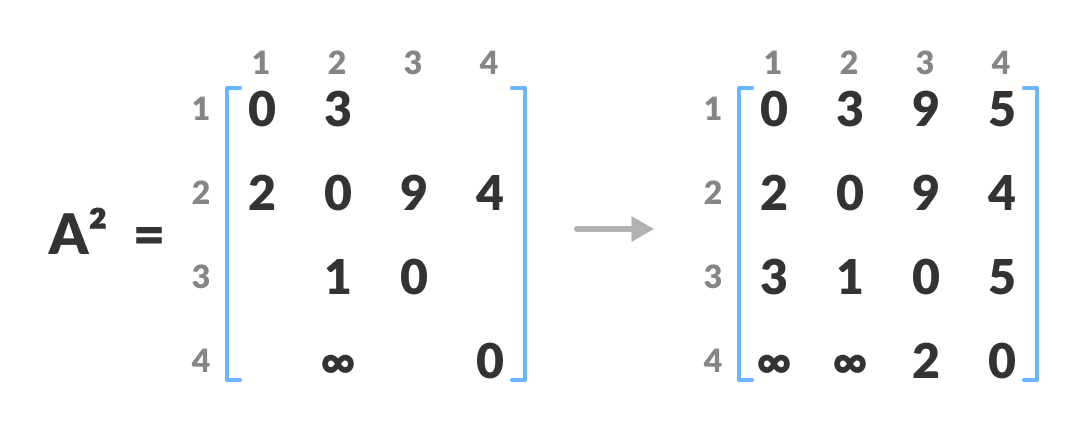
*Fill each cell with the distance between ith and jth vertex*

1. Now, create a matrix A1 using matrix A0. The elements in the first column and the first row are left as they are. The remaining cells are filled in the following way.  
   Let k be the intermediate vertex in the shortest path from source to destination. In this step, k is the first vertex. A[i][j] is filled with (A[i][k] + A[k][j]) if (A[i][j] > A[i][k] + A[k][j]).  
   That is, if the direct distance from the source to the destination is greater than the path through the vertex k, then the cell is filled with A[i][k] + A[k][j].  
   In this step, k is vertex 1.

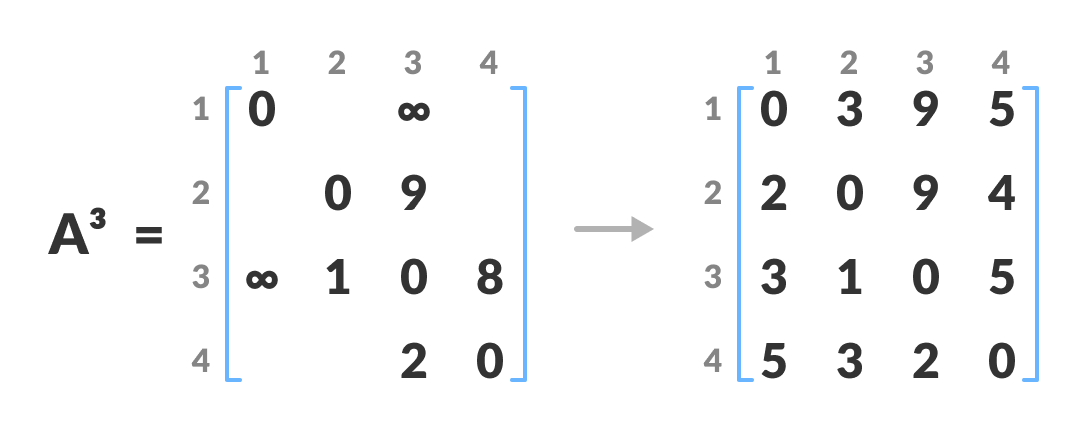
We calculate the distance from source vertex to destination vertex through this vertex k.

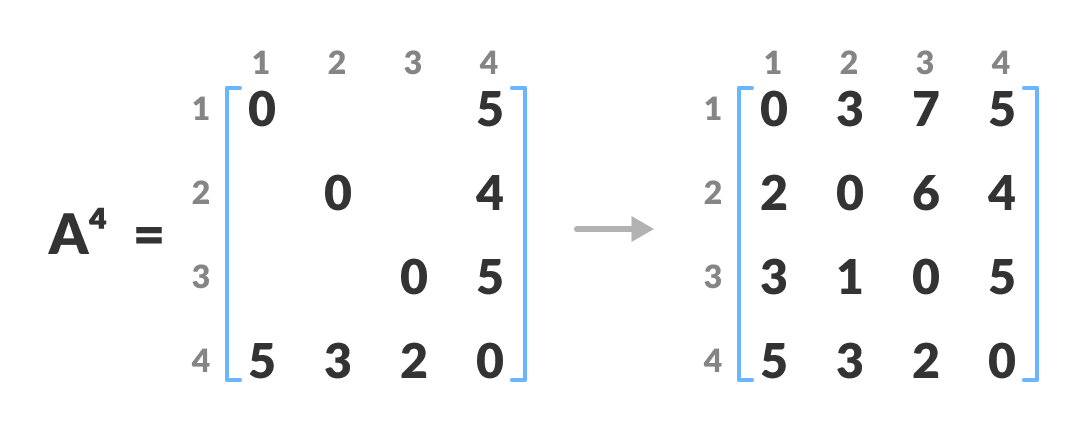
*Calculate the distance from the source vertex to destination vertex through this vertex k*

For example: For A1[2, 4], the direct distance from vertex 2 to 4 is 4 and the sum of the distance from vertex 2 to 4 through vertex (ie. from vertex 2 to 1 and from vertex 1 to 4) is 7. Since 4 < 7, A0[2, 4] is filled with 4.

1. Similarly, A2 is created using A1. The elements in the second column and the second row are left as they are.  
   In this step, k is the second vertex (i.e. vertex 2). The remaining steps are the same as in step 2.

*Calculate the distance from the source vertex to destination vertex through this vertex 2*

1. Similarly, A3 and A4 is also created.

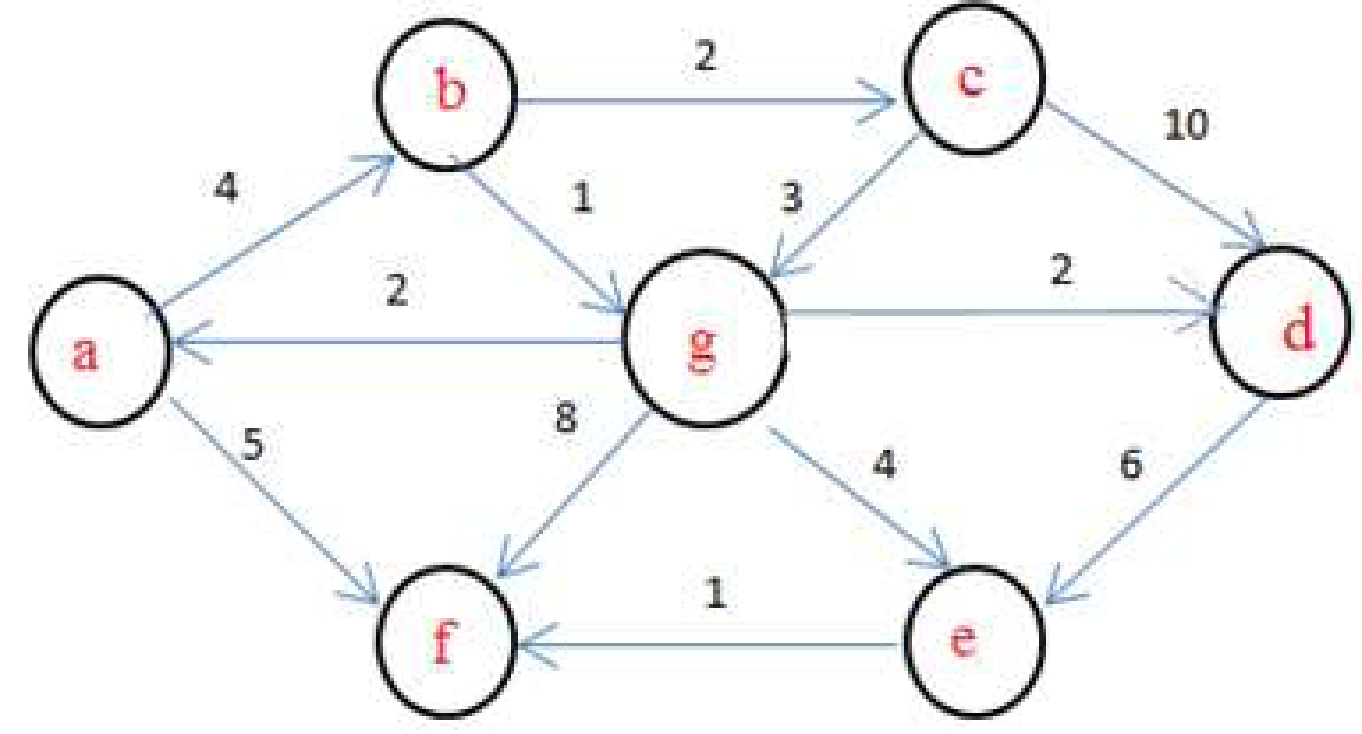
*Calculate the distance from the source vertex to destination vertex through this vertex 3* 

*Calculate the distance from the source vertex to destination vertex through this vertex 4*

1. A4 gives the shortest path between each pair of vertices.

**Problem Statement**

Find Shortest Path for each source to all destinations using Floyd-Warshall Algorithm for the following graph.

****

**Solution:**

[[0, 4, 6, 11, 6, 5, 9],

['INF', 0, 2, 7, 2, 1, 5],

['INF', 6, 0, 5, 7, 7, 3],

['INF', 18, 17, 0, 6, 7, 15],

['INF', 12, 11, 11, 0, 1, 9],

['INF', 11, 10, 10, 1, 0, 8],

['INF', 3, 2, 2, 4, 4, 0]]

**Derivation of Floyd-Warshall Algorithm:**

Time complexity Analysis

The time complexity is derived from the three nested loops used in the algorithm, each going through all vertices:

For a graph with vertices, the first loop runs times for choosing an intermediate vertex .

The second loop runs times for picking the starting vertex .

The third loop runs times for picking the ending vertex .

The overall time complexity is since the total number of operations is proportional to .

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Program(s) of Floyd-Warshall Algorithm:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define INF 99999

#define MAX\_INPUT\_LEN 100

void printSolution(int \*\*dist, int V);

void floydWarshall(int \*\*graph, int V);

int main() {

int V, \*\*graph;

char input[MAX\_INPUT\_LEN];

printf("Enter the number of vertices in the graph: ");

scanf("%d", &V);

graph = (int \*\*)malloc(V \* sizeof(int \*));

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

graph[i] = (int \*)malloc(V \* sizeof(int));

}

printf("Enter the adjacency matrix (use 'INF' for infinity):\n");

for (int i = 1; i <= V; i++) {

for (int j = 1; j <= V; j++) {

printf("Enter the weight of the edge from vertex %d to vertex %d (or 'INF' for no direct edge): ", i, j);

scanf("%s", input);

if (strcmp(input, "INF") == 0) {

graph[i-1][j-1] = INF;

} else {

graph[i-1][j-1] = atoi(input);

}

}

}

floydWarshall(graph, V);

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

free(graph[i]);

}

free(graph);

return 0;

}

void floydWarshall(int \*\*graph, int V) {

int \*\*dist = (int \*\*)malloc(V \* sizeof(int \*));

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

dist[i] = (int \*)malloc(V \* sizeof(int));

for (int j = 0; j < V; j++) {

dist[i][j] = graph[i][j];

}

}

for (int k = 0; k < V; k++) {

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

for (int j = 0; j < V; j++) {

if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) {

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

}

printSolution(dist, V);

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

free(dist[i]);

}

free(dist);

}

void printSolution(int \*\*dist, int V) {

printf("The shortest distances between every pair of vertices:\n");

for (int i = 1; i <= V; i++) {

for (int j = 1; j <= V; j++) {

if (dist[i-1][j-1] == INF) {

printf("%7s", "INF");

} else {

printf("%7d", dist[i-1][j-1]);

}

}

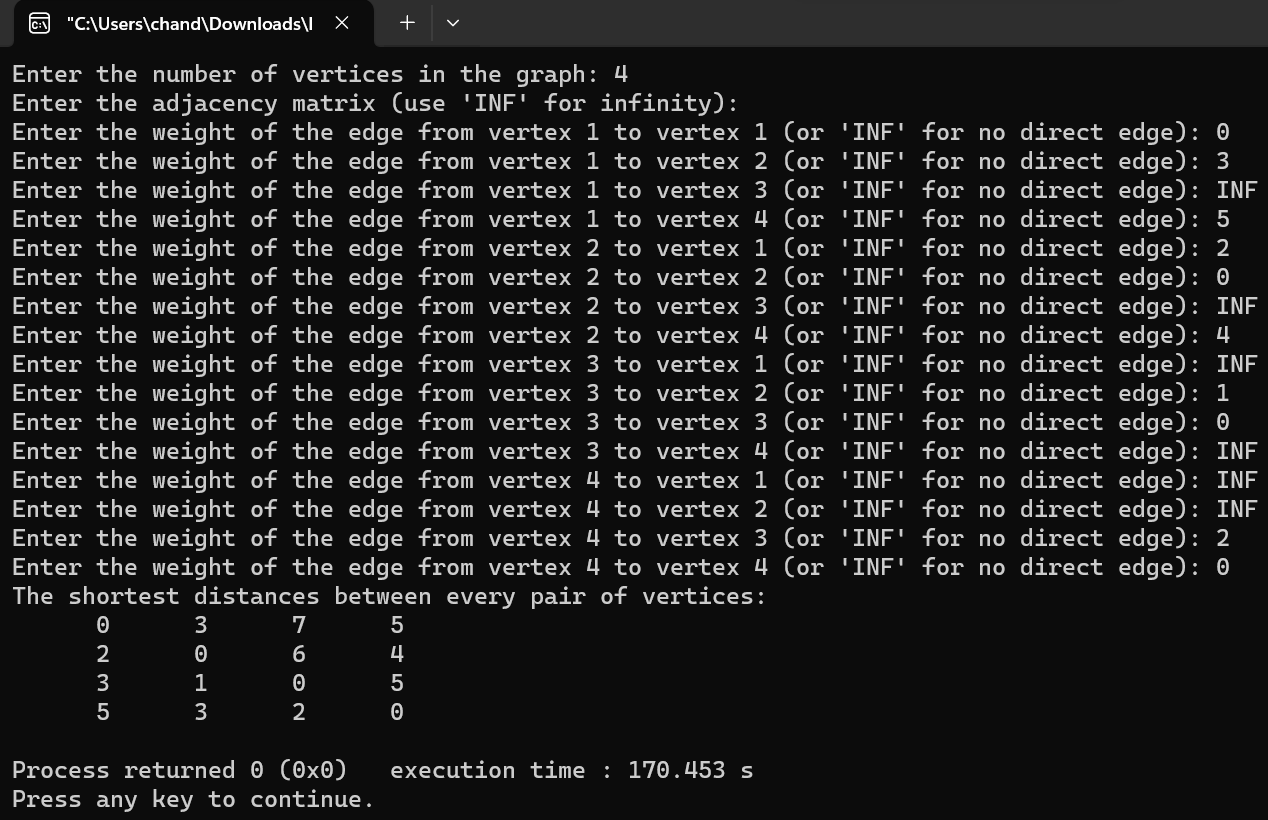
printf("\n");

}

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Output(o) of Floyd-Warshall Algorithm:**

****



**Post Lab Questions:**

**Explain a dynamic programming approach for the Floyd-Warshall algorithm and write the various applications of it.**

**Ans:** The Floyd-Warshall algorithm is a classic example of dynamic programming, used to find the shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles). The beauty of the Floyd-Warshall algorithm lies in its simplicity and efficiency in computing the shortest paths between all pairs of vertices in a graph.

Dynamic Programming Approach:

The dynamic programming approach of the Floyd-Warshall algorithm iteratively improves the solution by considering all possible paths between each pair of vertices and efficiently updating the shortest paths using a bottom-up approach. The key idea is to incrementally consider intermediate vertices through which a shortest path might pass.

Algorithm Steps:

1. Initialization: Start with a matrix of distances between each pair of vertices. Initially, this matrix is just the adjacency matrix of the graph, where the entry at `dist[i][j]` is the direct distance from `i` to `j` (if there is an edge), or infinity (if there is no direct edge).

2. Iterative Update: For each vertex `k`, consider it as an intermediate vertex in the paths between all pairs of vertices `(i, j)`. For every pair of vertices `(i, j)`, check if a path from `i` to `j` passing through `k` is shorter than the current known shortest path. If so, update the shortest path to this new value. This step is repeated for all vertices `k`.

The key relation for the update is:

dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])

This formula is applied for each pair `(i, j)` for each intermediate vertex `k`.

3. Completion: After considering all vertices as intermediate points, the matrix `dist` will contain the lengths of the shortest paths between all pairs of vertices.

Applications of the Floyd-Warshall Algorithm:

The Floyd-Warshall algorithm's ability to compute shortest paths between all pairs of vertices in a weighted graph makes it versatile for various applications, including:

1. Network Routing: In telecommunications, finding the shortest paths can optimize the routing of messages through a network, minimizing latency or cost.

2. Geographical Mapping: In geographic information systems (GIS), calculating the shortest routes between various locations on a map.

3. Social Network Analysis: Determining the shortest paths can help analyze the degrees of separation between individuals in a social network.

4. Game Development: In pathfinding algorithms for NPCs (non-player characters), to find the shortest path through a game's map.

5. Robotics and Path Planning: In robotics, for calculating the shortest route a robot should take to navigate between points in a space filled with obstacles.

6. Internet Routing Protocols: Algorithms like Floyd-Warshall influence the design of protocols for routing internet traffic to ensure efficient data transmission.

7. Arbitrage Opportunities in Finance: Finding negative cycles in the graph representing currency exchange rates to exploit arbitrage opportunities.

The Floyd-Warshall algorithm, through its dynamic programming approach, offers a robust method for solving a wide range of real-world problems that require efficient computation of shortest paths in various domains.



**Conclusion: (Based on the observations)**

The experiment with the Floyd-Warshall algorithm underscores its significance in the realm of graph algorithms, offering a powerful tool for solving all-pair shortest path problems with a dynamic programming approach.



**Outcome: Implement Greedy and Dynamic Programming algorithms**



**References:**

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.