**Experiment No. 3: Dynamic Programming**

**Date:**

**Aim:** To implement the following Algorithms using the Dynamic Programming approach

i) Multistage Graph using Forward and Backward approach

ii) All Pairs shortest path

iii) Single Source Shortest Path

iv) Optimal Binary Search Tree

v) 0/1 Knapsack

**Theory:**

* Dynamic Programming is an algorithm design method that can be used when the solution to a problem can be viewed as the result of a sequence of decisions.
* Dynamic programming, like Divide and Conquer method, solves problems by combining the solutions to subproblems. Dynamic Programming applies when the subproblems overlap. Ie. when the subproblems share subproblems.
* A dynamic programming algorithm solves each subproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time it solves each subproblem.

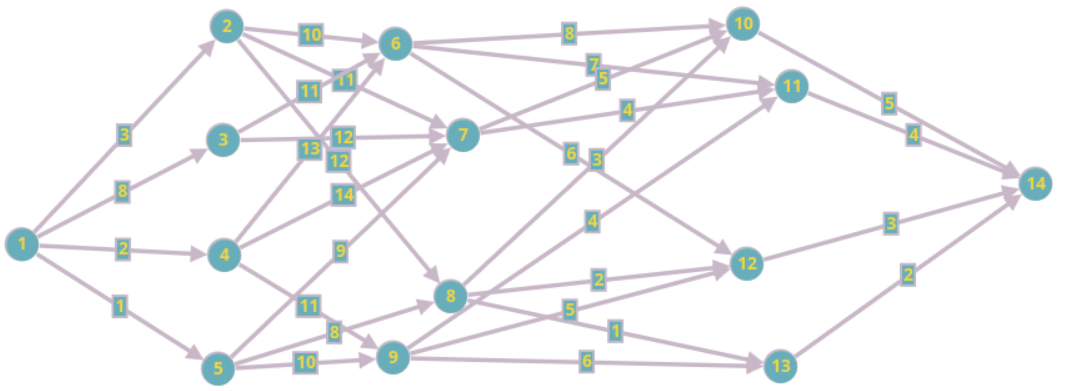
* We typically apply dynamic programming to optimization problems. Such problems can have many possible solutions. Each solution has a value, and we wish to find a solution with the optimal (minimum or maximum) value.
* We call such a solution an optimal solution to the problem, as opposed to the optimal solution, since there may be several solutions that achieve the optimal value.

**a)Multistage Graph (Forward and Backward Approach)**

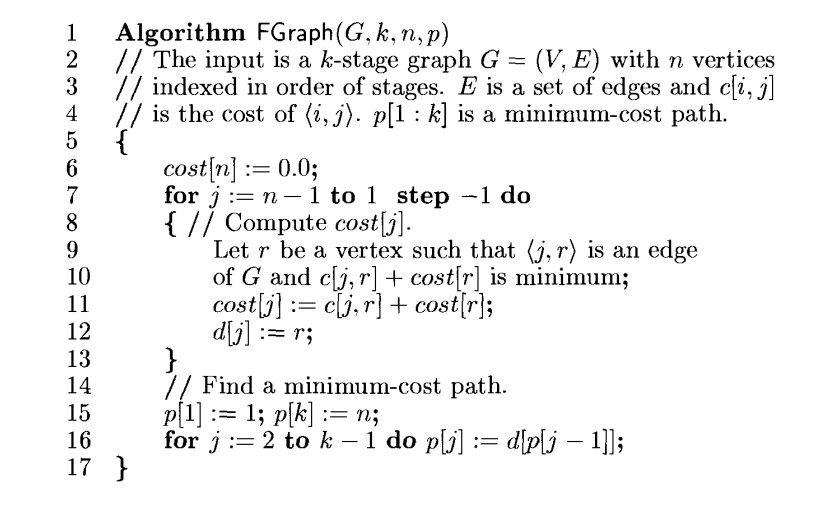
**Date:**

**Problem Statement:**

Find the optimum path from source vertex (1) to destination vertex (14) in the given Multistage graph using forward and backward approach (Dynamic Programming)



**Algorithm (Forward Approach):**



**Time Complexity:**

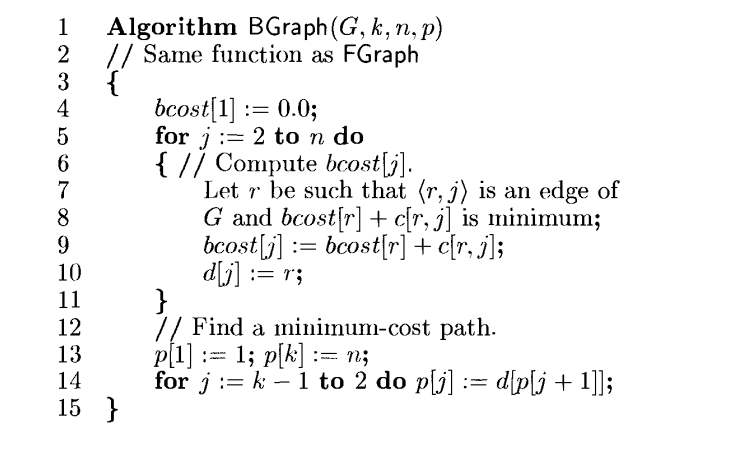
(Forward Approach): O(N2)

**Space Complexity:**

(Forward Approach): O(N)

(Where N is the number of nodes in the Multistage graph)

**Algorithm (Backward Approach):**



**Time Complexity:**

(Backward Approach): O(N2)

**Space Complexity:**

(Backward Approach): O(N)

(Where N is the number of nodes in the Multistage graph)

**Code(FGraph):**

#include <stdio.h>

int adj[100][100], C[100][100], p[100], cost[100], d[100];

// void accept\_graph(FILE \*fp, int n, int k)

void accept\_graph(int n, int k)

{

int max\_edges = n \* (n - 1);

int origin, destin, edgecost;

printf("Enter the edges of the graph and their respective cost.\n");

for (int i = 0; i < max\_edges; i++)

{

printf("Enter the edge,( 0 0 randval) to quit :");

scanf("%d %d %d", &origin, &destin, &edgecost);

// fscanf(fp, "%d %d %d", &origin, &destin, &edgecost);

if ((origin == 0) && (destin == 0))

break;

if (origin > n || destin > n || origin <= 0 || destin <= 0)

{

printf("Invalid edge.\n");

i--;

}

else

{

adj[origin - 1][destin - 1] = 1;

C[origin - 1][destin - 1] = edgecost;

}

}

}

void init\_cost(int n)

{

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

{

for (int j = 0; j < n; j++)

C[i][j] = \_\_INT32\_MAX\_\_;

}

}

void fgraph(int n, int k, int v)

{

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

{

cost[i] = \_\_INT32\_MAX\_\_;

d[i] = 0;

}

cost[n - 1] = 0;

for (int j = n - 1; j >= 0; j--)

{

int min = \_\_INT32\_MAX\_\_, minr;

for (int r = j + 1; r < n; r++)

{

if (adj[j][r] == 1 && (C[j][r] + cost[r]) < min)

{

min = C[j][r] + cost[r];

minr = r;

}

}

if (min != \_\_INT32\_MAX\_\_)

cost[j] = min;

d[j] = minr;

}

p[0] = v - 1;

p[k - 1] = n - 1;

for (int j = 1; j < k - 1; j++)

p[j] = d[p[j - 1]];

}

int main()

{

// FILE \*fp;

// fp = fopen("graph3.txt", "r+");

int n, k;

printf("Enter the number of vertices of the graph, and the stages.\n");

scanf("%d%d", &n, &k);

init\_cost(n);

accept\_graph(n, k);

// accept\_graph(fp, n, k);

// fclose(fp);

int v;

printf("Enter the source vertex of the graph.\n");

scanf("%d", &v);

fgraph(n, k, v);

printf("VERTEX COST CHOICE VERTEX\n");

printf("%2d %2d \n", n, cost[n - 1]);

for (int i = n - 2; i >= 0; i--)

{

printf("%2d %2d %2d", i + 1, cost[i], d[i] + 1);

printf("\n");

}

printf("The chosen minimun cost path is: ");

for (int i = 0; i < k; i++)

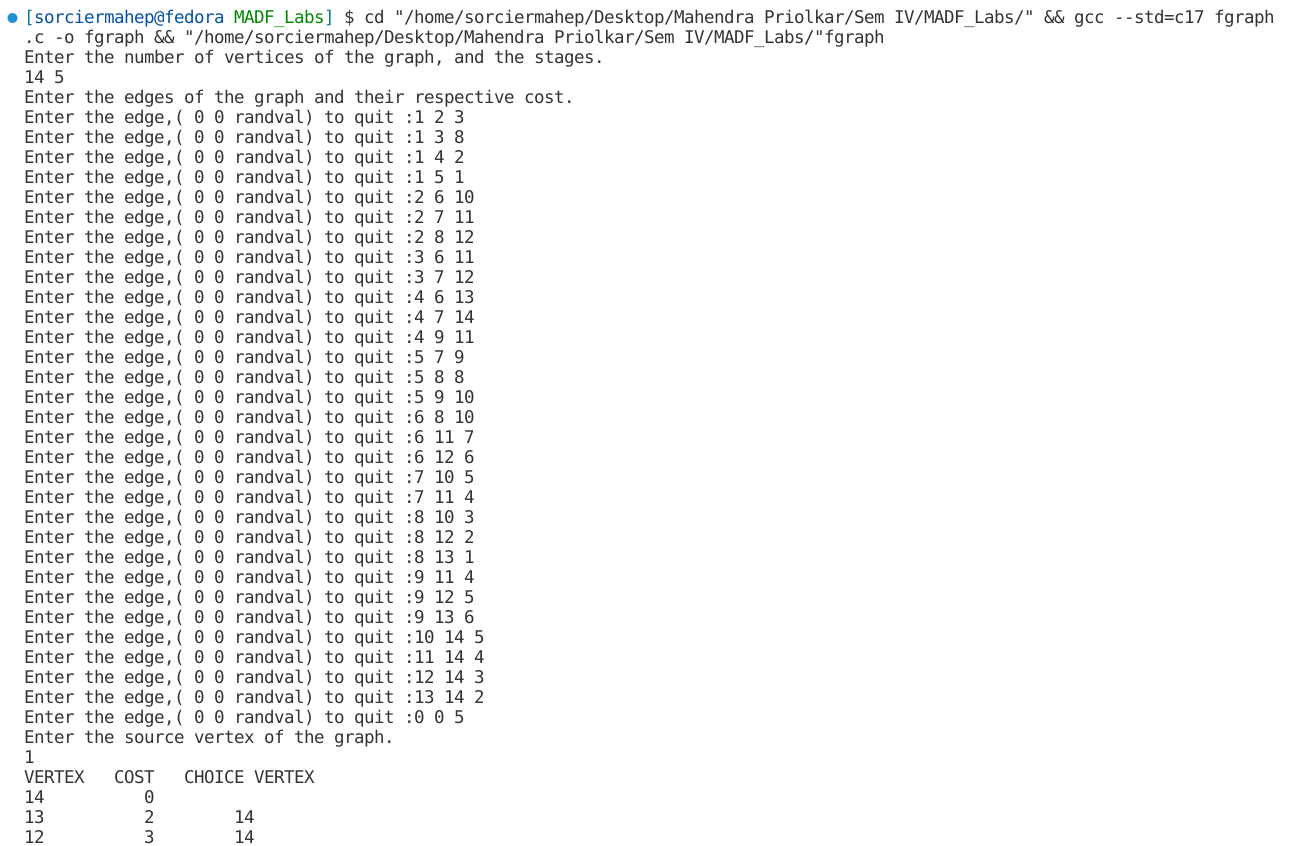
printf("%d ", p[i] + 1);

printf("\n");

return 0;

}

**Output:**

****

**Code(BGraph):**

#include <stdio.h>

int adj[100][100], C[100][100], p[100], bcost[100], d[100];

// void accept\_graph(FILE \*fp, int n, int k)

void accept\_graph(int n, int k)

{

int max\_edges = n \* (n - 1);

int origin, destin, edgecost;

printf("Enter the edges of the graph and their respective cost.\n");

for (int i = 0; i < max\_edges; i++)

{

printf("Enter the edge,( 0 0 randval) to quit :");

scanf("%d %d %d", &origin, &destin, &edgecost);

// fscanf(fp, "%d %d %d", &origin, &destin, &edgecost);

if ((origin == 0) && (destin == 0))

break;

if (origin > n || destin > n || origin <= 0 || destin <= 0)

{

printf("Invalid edge.\n");

i--;

}

else

{

adj[origin - 1][destin - 1] = 1;

C[origin - 1][destin - 1] = edgecost;

}

}

}

void init\_cost(int n)

{

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

{

for (int j = 0; j < n; j++)

C[i][j] = \_\_INT32\_MAX\_\_;

}

}

void bgraph(int n, int k, int v)

{

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

{

bcost[i] = \_\_INT32\_MAX\_\_;

d[i] = 0;

}

bcost[0] = 0;

for (int j = 0; j < n; j++)

{

int min = \_\_INT32\_MAX\_\_, minr;

for (int r = j - 1; r >= 0; r--)

{

if (adj[r][j] == 1 && (C[r][j] + bcost[r]) < min)

{

min = C[r][j] + bcost[r];

minr = r;

}

}

if (min != \_\_INT32\_MAX\_\_)

bcost[j] = min;

d[j] = minr;

}

p[0] = v - 1;

p[k - 1] = n - 1;

for (int j = k - 2; j > 0; j--)

p[j] = d[p[j + 1]];

}

int main()

{

// FILE \*fp;

// fp = fopen("graph3.txt", "r+");

int n, k;

printf("Enter the number of vertices of the graph, and the stages.\n");

scanf("%d%d", &n, &k);

init\_cost(n);

accept\_graph(n, k);

// accept\_graph(fp, n, k);

// fclose(fp);

int v;

printf("Enter the source vertex of the graph.\n");

scanf("%d", &v);

bgraph(n, k, v);

printf("VERTEX COST CHOICE VERTEX\n");

printf("%2d %2d \n", 1, bcost[1 - 1]);

for (int i = 1; i < n; i++)

{

printf("%2d %2d %2d", i + 1, bcost[i], d[i] + 1);

printf("\n");

}

printf("The chosen minimun cost path is: ");

for (int i = 0; i < k; i++)

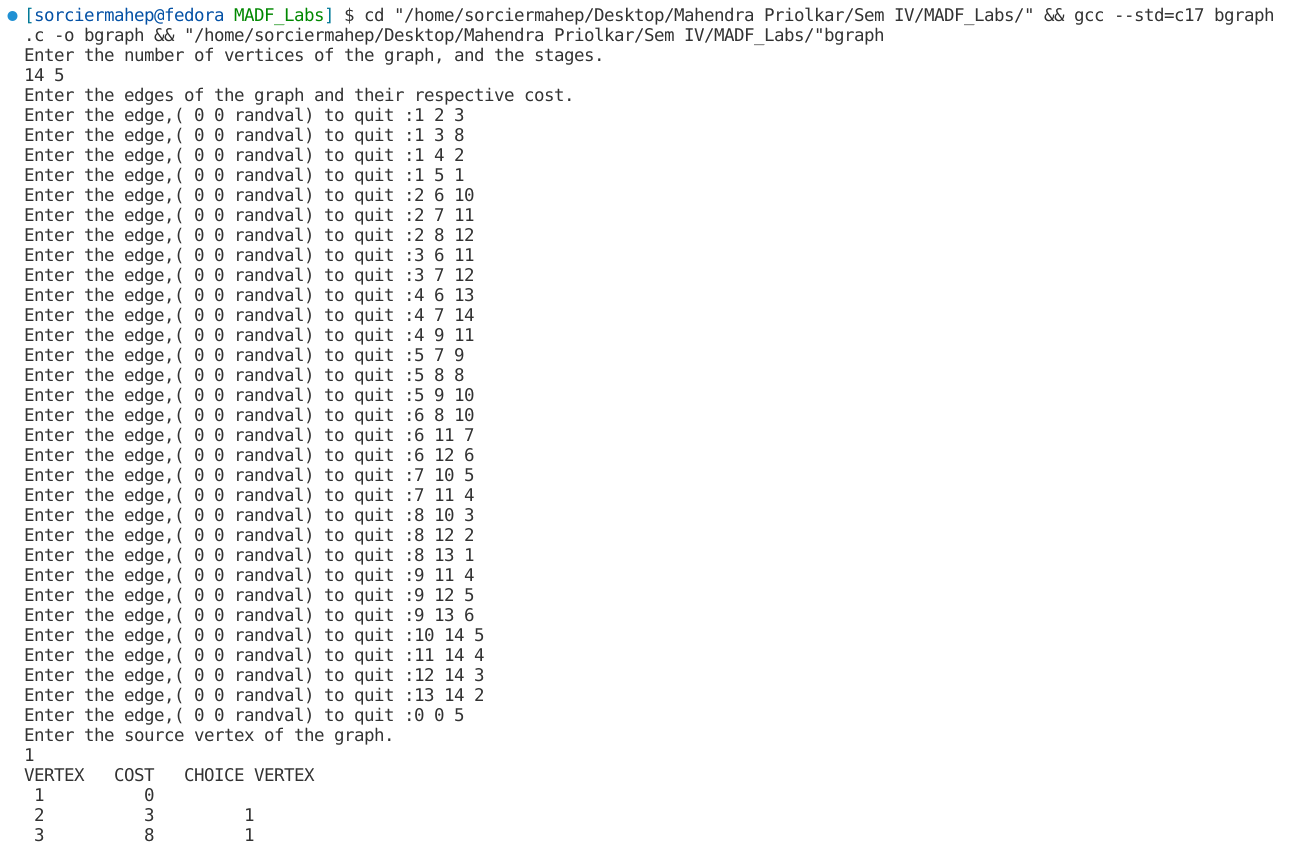
printf("%d ", p[i] + 1);

printf("\n");

return 0;

}

**Output:**

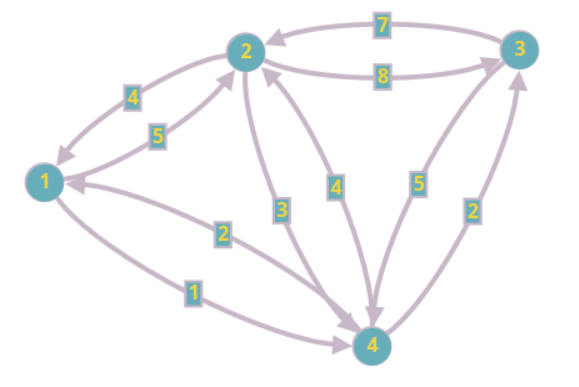


**b)All Pairs Shortest Path**

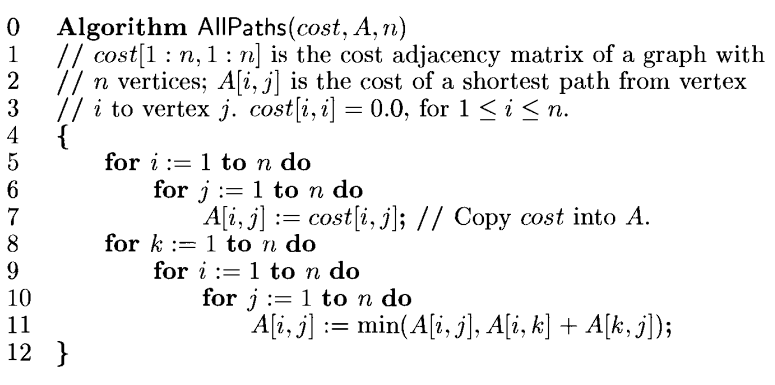
**Date:**

**Problem Statement:**

Find the shortest path between every pair of vertices in the given graph.



**Algorithm:**



**Time Complexity:**

O(N3)

**Space Complexity:**

O(N2)

**Code:**

#include <stdio.h>

int adj[100][100], cost[100][100], A[100][100];

// void accept\_graph(FILE \*fp, int n)

void accept\_graph(int n)

{

int max\_edges = n \* (n - 1);

int origin, destin, edgecost;

printf("Enter the edges of the graph and their respective cost.\n");

for (int i = 0; i < max\_edges; i++)

{

printf("Enter the edge,( 0 0 randval) to quit :");

scanf("%d %d %d", &origin, &destin, &edgecost);

// fscanf(fp, "%d %d %d", &origin, &destin, &edgecost);

if ((origin == 0) && (destin == 0))

break;

if (origin > n || destin > n || origin <= 0 || destin <= 0)

{

printf("Invalid edge.\n");

i--;

}

else

{

adj[origin - 1][destin - 1] = 1;

cost[origin - 1][destin - 1] = edgecost;

}

}

}

void init\_cost(int n)

{

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

{

for (int j = 0; j < n; j++)

{

if (i == j)

cost[i][j] = 0;

else

cost[i][j] = \_\_INT32\_MAX\_\_;

}

}

}

void printm(int n, int mat[][100])

{

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

{

for (int j = 0; j < n; j++)

{

if (mat[i][j] >= \_\_INT32\_MAX\_\_)

printf("∞ ");

else

printf("%d ", mat[i][j]);

}

printf("\n");

}

printf("\n");

}

void allpair(int n)

{

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

for (int j = 0; j < n; j++)

A[i][j] = cost[i][j];

printm(n, A);

for (int k = 0; k < n; k++)

{

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

{

for (int j = 0; j < n; j++)

{

if (adj[i][k] == 1 && adj[k][j] == 1)

A[i][j] = A[i][j] < A[i][k] + A[k][j] ? A[i][j] : A[i][k] + A[k][j];

}

}

if (k != n - 1)

printm(n, A);

}

}

int main()

{

// FILE \*fp;

// fp = fopen("graph4.txt", "r+");

int n;

printf("Enter the number of vertices of the graph.\n");

scanf("%d", &n);

init\_cost(n);

accept\_graph(n);

// accept\_graph(fp, n);

// fclose(fp);

allpair(n);

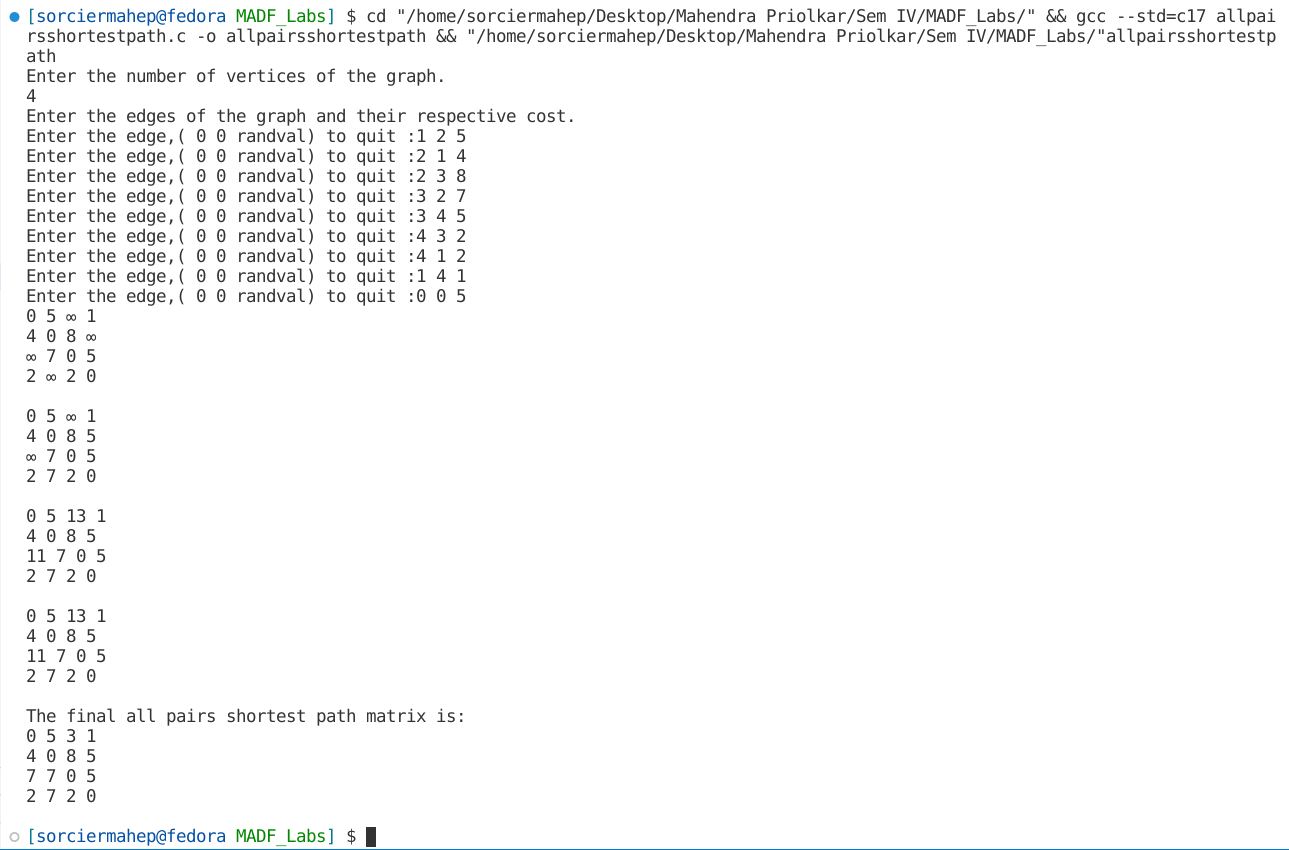
printf("The final all pairs shortest path matrix is: \n");

printm(n, A);

return 0;

}

**Output:**

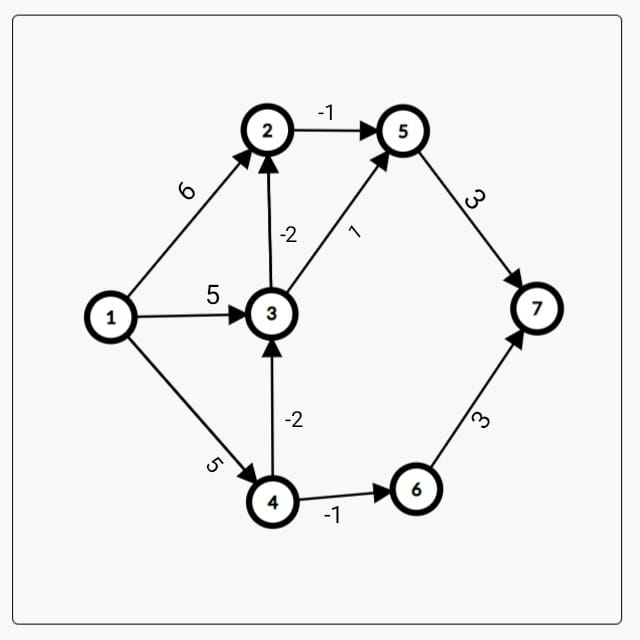


**c)Single Source Shortest Path Algorithm(Bellman-Ford)**

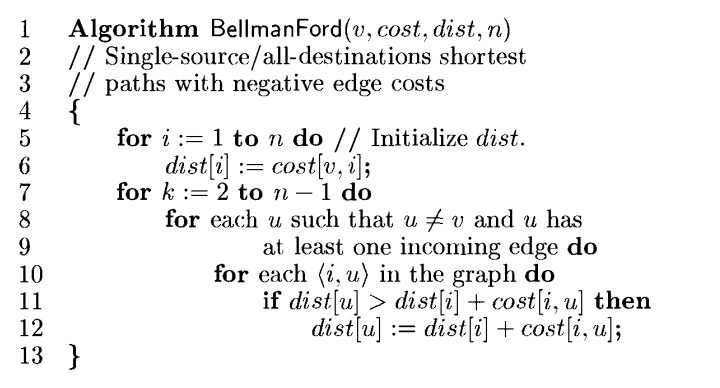
**Date:**

**Problem Statement:**

Find the shortest path from source vertex (1) to all the other nodes using Bellman-Ford algorithm.



**Algorithm:**



**Time Complexity:**

O(NE)

**Space Complexity:**

O(N)

Where N is the number of vertices in the graph and E the edges.

**Code:**

#include <stdio.h>

#include <stdbool.h>

int adj[100][100], cost[100][100], dist[100], newdist[100];

// void accept\_graph(FILE \*fp, int n)

void accept\_graph(int n)

{

int max\_edges = n \* (n - 1);

int origin, destin, edgecost;

printf("Enter the edges of the graph and their respective cost.\n");

for (int i = 0; i < max\_edges; i++)

{

printf("Enter the edge,( 0 0 randval) to quit :");

scanf("%d %d %d", &origin, &destin, &edgecost);

// fscanf(fp, "%d %d %d", &origin, &destin, &edgecost);

if ((origin == 0) && (destin == 0))

break;

if (origin > n || destin > n || origin <= 0 || destin <= 0)

{

printf("Invalid edge.\n");

i--;

}

else

{

adj[origin - 1][destin - 1] = 1;

cost[origin - 1][destin - 1] = edgecost;

}

}

}

void init\_cost(int n, int v)

{

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

{

for (int j = 0; j < n; j++)

{

if (i == j)

cost[i][j] = 0;

else

cost[i][j] = \_\_INT32\_MAX\_\_;

}

}

}

bool incoming\_check(int u, int n)

{

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

{

if (adj[i][u] == 1)

return true;

}

return false;

}

void printarr(int arr[], int n)

{

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

{

if (arr[i] == \_\_INT32\_MAX\_\_)

printf(" ∞ ");

else

printf("%2d ", arr[i]);

}

printf("\n");

}

void recpathprint(int parent[], int j)

{

if (parent[j] == -1)

{

printf("%d ", j + 1);

return;

}

recpathprint(parent, parent[j]);

printf("%d ", j + 1);

}

void allpathprint(int parent[], int n, int v)

{

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

{

if (i == v - 1)

continue;

else

{

printf("%d ", v);

recpathprint(parent, i);

printf("\n");

}

}

printf("\n");

}

void bellmanford(int n, int v)

{

int parent[n];

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

{

dist[i] = newdist[i] = cost[v - 1][i];

parent[i] = -1;

}

printarr(dist, n);

for (int k = 1; k < n - 1; k++)

{

for (int u = 0; u < n; u++)

{

int l, i, m;

if (u != v - 1 && incoming\_check(u, n) == true)

{

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

{

if (newdist[u] > dist[i] + cost[i][u] && adj[i][u] == 1 && dist[i] != \_\_INT32\_MAX\_\_)

{

newdist[u] = dist[i] + cost[i][u];

parent[u] = i;

}

}

}

}

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

dist[i] = newdist[i];

printarr(dist, n);

}

printf("The shortest paths are: \n");

allpathprint(parent, n, v);

}

int main()

{

// FILE \*fp;

// fp = fopen("graph5.txt", "r+");

int n;

printf("Enter the number of vertices of the graph.\n");

scanf("%d", &n);

int v;

printf("Enter the source vertex of the graph.\n");

scanf("%d", &v);

init\_cost(n, v);

accept\_graph(n);

// accept\_graph(fp, n);

// fclose(fp);

bellmanford(n, v);

printf("The distances of source vertex to other vertices are:\nVERTICES: ");

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

printf("%2d ", i + 1);

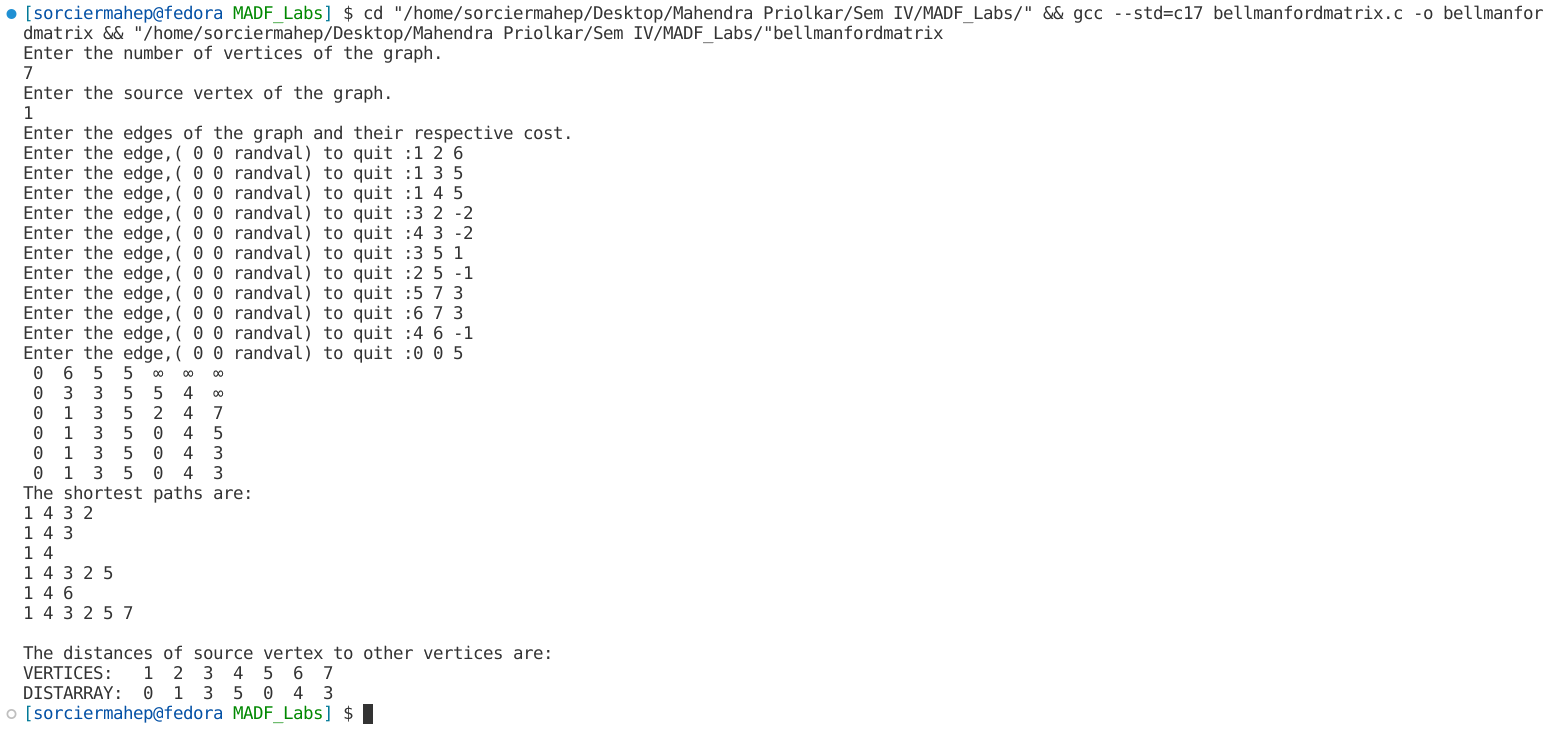
printf("\nDISTARRAY: ");

printarr(dist, n);

return 0;

}

**Output:**



**d)Optimal Binary Search Tree**

**Date:**

**Problem Statement:**

Construct an optimal Binary Search Tree for the following set of data.

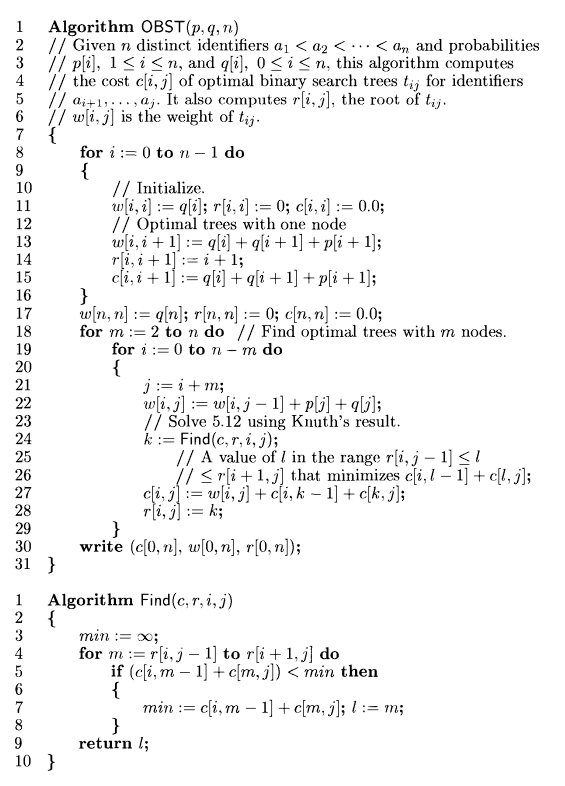
N = 5

(a1. . . a5) = {Apr, Mar, May, Oct, Sept}

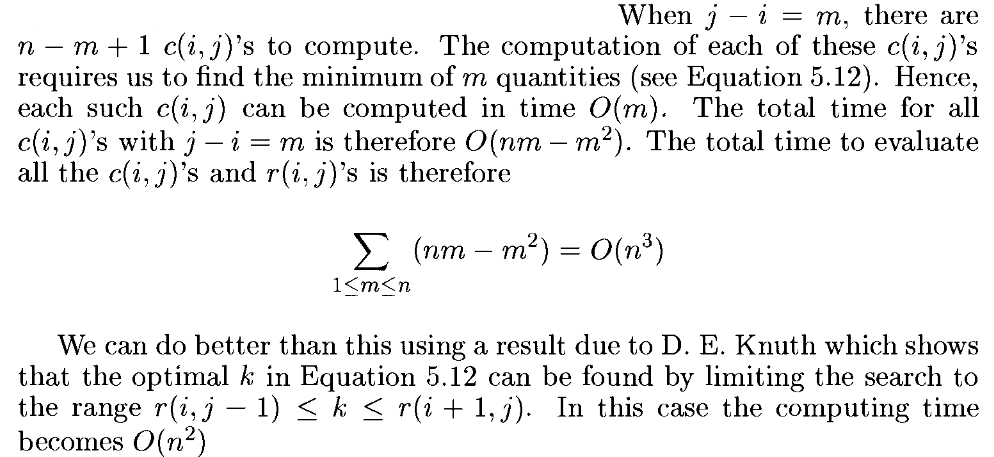
(p1. . . p5) = (3, 4, 3, 2, 4)

(q0. . . q5) = (4, 4, 5, 4, 5, 4)

**Algorithm:**



**Time Complexity:**



**Space Complexity:**

O(N2)

**Code:**

#include <stdio.h>

#include <string.h>

#include <math.h>

#define MAX 100

char iden[MAX][MAX];

int p[MAX], q[MAX];

int printlchild(int r[MAX][MAX], int i, int j, int level)

{

if (r[i][r[i][j] - 1] != 0)

{

if (level == 0)

printf("%d = %s ", r[i][r[i][j] - 1], iden[r[i][r[i][j] - 1]]);

return 1;

}

return 0;

}

int printrchild(int r[MAX][MAX], int i, int j, int level)

{

if (r[r[i][j]][j] != 0)

{

if (level == 0)

printf("%d = %s ", r[r[i][j]][j], iden[r[r[i][j]][j]]);

return 1;

}

return 0;

}

void printchild(int r[MAX][MAX], int i, int j, int n, int level)

{

int a, b;

a = printlchild(r, i, j, level);

b = printrchild(r, i, j, level);

if (a != 0 && level > 0)

printchild(r, i, r[i][j] - 1, n, level - 1);

if (b != 0 && level > 0)

printchild(r, r[i][j], j, n, level - 1);

}

void levelorder(int r[MAX][MAX], int i, int j, int n)

{

for (int level = 0; level < n - 1; level++)

{

printchild(r, i, j, n, level);

printf("\n");

}

}

void OBST(int w[MAX][MAX], int c[MAX][MAX], int r[MAX][MAX], int n)

{

int temp = 0, min, min1;

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

{

w[i][i] = q[i];

r[i][i] = 0;

c[i][i] = 0;

w[i][i + 1] = q[i] + q[i + 1] + p[i + 1];

r[i][i + 1] = i + 1;

c[i][i + 1] = q[i] + q[i + 1] + p[i + 1];

printf("%d %d\t%d\t%d\t%d\n", i, i, w[i][i], c[i][i], r[i][i]);

}

w[n][n] = q[n];

r[n][n] = 0;

c[n][n] = 0;

printf("%d %d\t%d\t%d\t%d\n", n, n, w[n][n], c[n][n], r[n][n]);

for (int k = 1; k <= n; k++)

{

for (int i = 0, j = k; j <= n; i++, j++) // Since table is of upper left triangular format

{

if (i != j && j - i != 1)

{

w[i][j] = p[j] + q[j] + w[i][j - 1];

min = \_\_INT32\_MAX\_\_;

for (int l = i + 1; l <= j; l++)

{

min1 = c[i][l - 1] + c[l][j];

if (min1 < min)

{

min = min1;

temp = l;

}

}

c[i][j] = min + w[i][j];

r[i][j] = temp;

}

printf("%d %d\t%d\t%d\t%d\n", i, j, w[i][j], c[i][j], r[i][j]);

}

}

printf("The minimum cost is %d.\n", c[0][n]);

printf("The level order traversal is:\n");

printf("Root = %d = %s.\n", r[0][n], iden[r[0][n]]);

levelorder(r, 0, n, n);

}

int main()

{

int w[MAX][MAX], c[MAX][MAX], r[MAX][MAX];

int n;

int i, j, k, b;

printf("Enter the number of elements: \n");

scanf("%d", &n);

printf("Enter %d identifiers.\n", n);

for (i = 1; i <= n; i++)

scanf("%s", iden[i]);

printf("Enter %d success probabilities.\n", n);

for (i = 1; i <= n; i++)

scanf("%d", &p[i]);

printf("Enter %d failure probabilities.\n", n + 1);

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

scanf("%d", &q[i]);

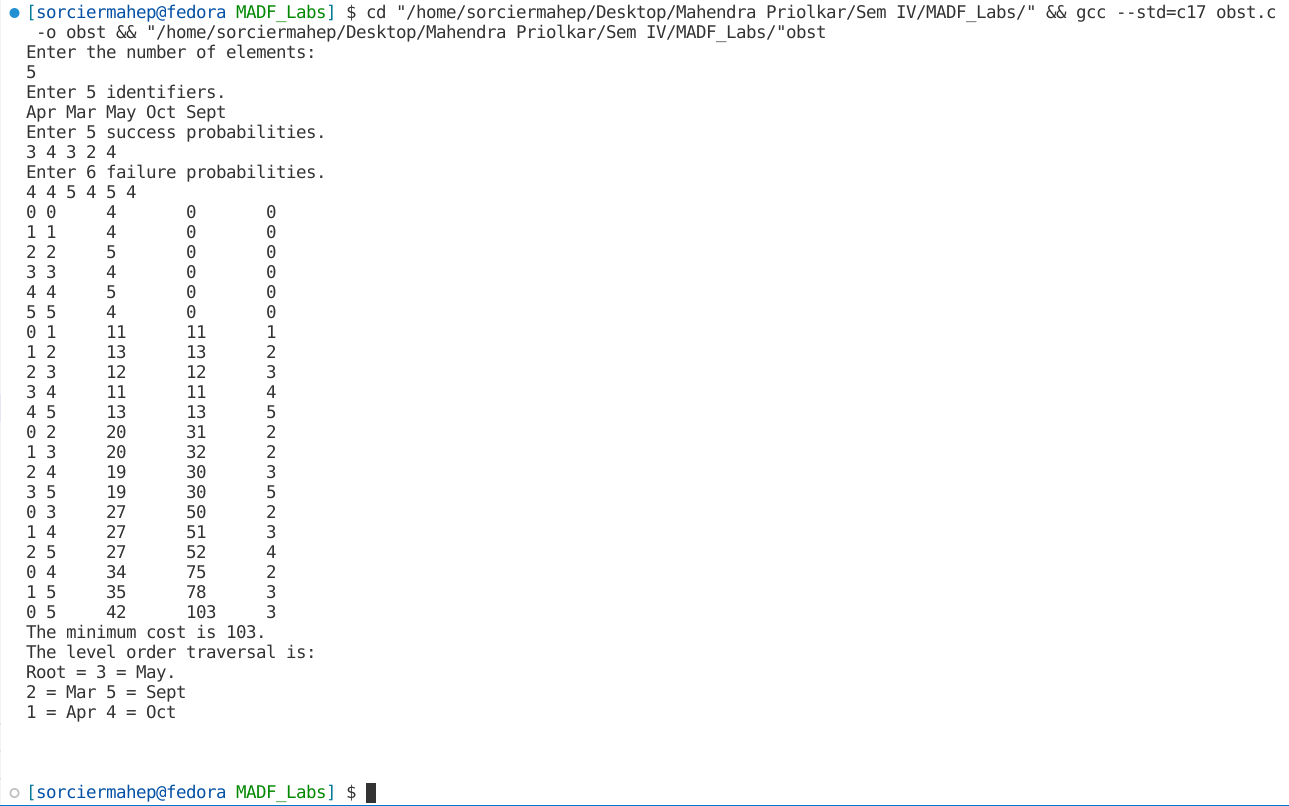
OBST(w, c, r, n);

printf("\n");

return 0;

}

**Output:**

****

**e)0/1 Knapsack**

**Date:**

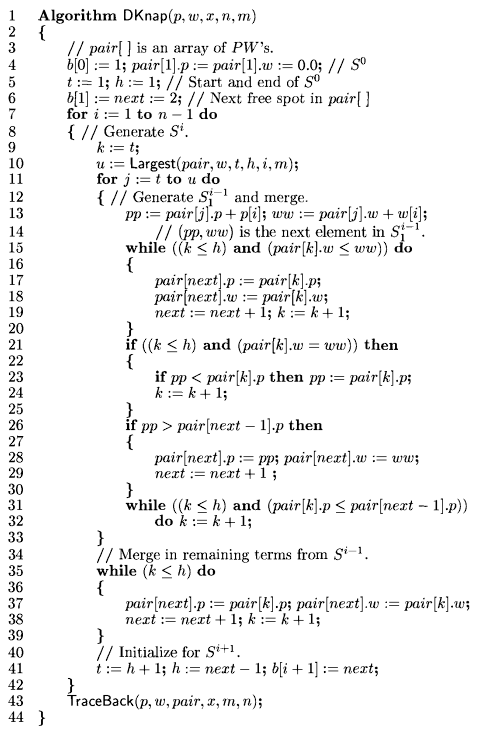
**Problem Statement:**

Solve the 0/1 knapsack problem for the knapsack instance n = 7, m = 15

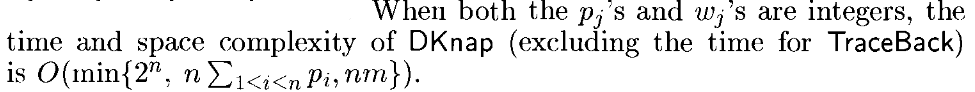
(w1 . . . w7) = (3, 2, 3, 4, 5, 2, 3)

(p1 . . . p7) = (15, 14, 16, 21, 17, 14, 13)

**Algorithm:**



**Time Complexity and Space Complexity:**



**Code:**

#include <stdio.h>

#define MAX 1000

struct knap

{

int p;

int w;

int obj[MAX];

};

void PurgeKnap(struct knap S[MAX], int size)

{

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

if ((S[i].p < S[j].p && S[i].w >= S[j].w) || (S[i].p == S[j].p && S[i].w > S[j].w))

S[i].p = S[i].w = -1;

else if ((S[j].p < S[i].p && S[j].w >= S[i].w) || (S[i].p == S[j].p && S[i].w < S[j].w))

S[j].p = S[j].w = -1;

}

for (int j = i + 1; j < size; j++)

{

if (S[i].p == S[j].p && S[i].w == S[j].w)

S[j].p = S[j].w = -1;

}

}

}

void DKnap(int m, int n, int p[n], int w[n])

{

struct knap S[MAX];

S[0].w = 0, S[0].p = 0;

printf("(%d,%d)\n", S[0].p, S[0].w);

int i, j, k, l;

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

S[0].obj[i] = 0;

int size = 1;

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

{

for (j = 0, k = size; j < size; j++, k++)

{

if (S[j].w == -1)

{

continue;

}

S[k].w = S[j].w + w[i];

if (S[k].w > m)

{

S[k].w = S[k].p = -1;

continue;

}

S[k].p = S[j].p + p[i];

for (l = 0; l < n; l++)

S[k].obj[l] = S[j].obj[l];

S[k].obj[i] += 1;

}

printf("\n");

size = size \* 2;

PurgeKnap(S, size);

for (int i = 0; i < size; i++)

{

if (S[i].w != -1 && S[i].p != -1)

printf("(%d,%d) ", S[i].p, S[i].w);

}

printf("\n");

}

printf("\n");

int maxp = -1, maxw = -1, index;

for (i = 0; i < size; i++)

{

if (S[i].w < maxw)

continue;

else if (S[i].w > maxw)

{

maxw = S[i].w, maxp = S[i].p;

index = i;

}

else

{

if (S[i].p > maxp)

{

maxp = S[i].p, index = i;

}

}

}

printf("\n\nMax profit: %d\nBag filled: %d\nObjects chosen: ", maxp, maxw);

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

printf("x[%d] ", i + 1);

printf(" : ");

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

{

if (S[index].obj[i] != 1)

printf("%d ", 0);

else

printf("%d ", S[index].obj[i]);

}

}

int main()

{

int n, m, i;

printf("Enter number of elements:");

scanf("%d", &n);

printf("Enter max weight capacity of knapsack bag:");

scanf("%d", &m);

int p[n], w[n];

printf("Enter profits :");

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

scanf("%d", &p[i]);

printf("Enter weights :");

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

scanf("%d", &w[i]);

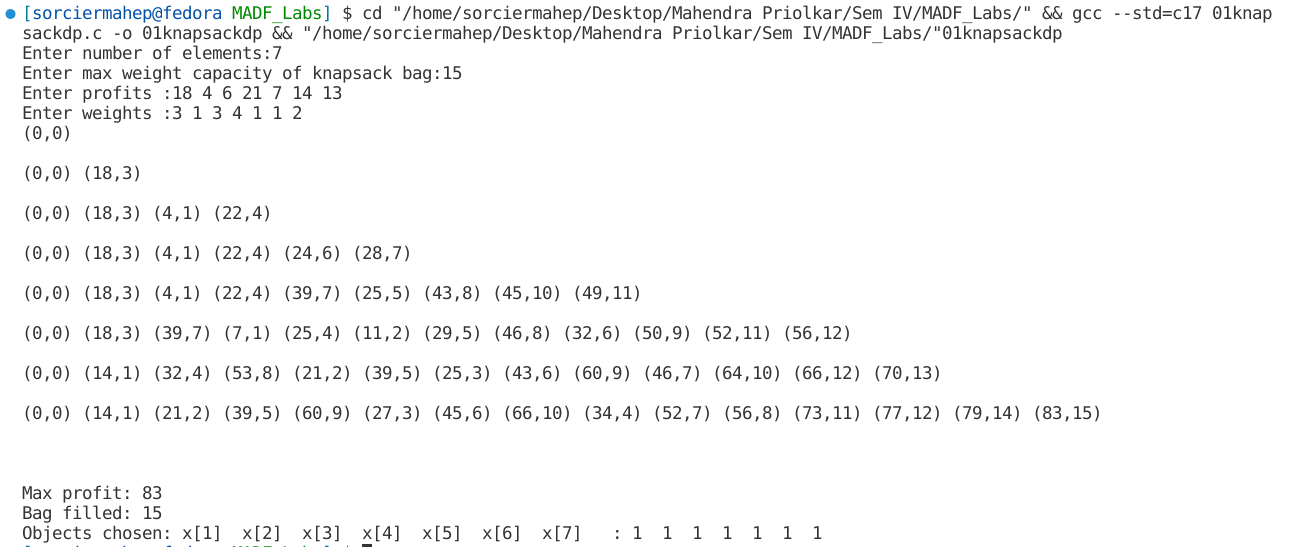
DKnap(m, n, p, w);

printf("\n");

return 0;

}

**Output:**



**Conclusion:** Several Optimization problems were studied and implemented using the Dynamic Programming Algorithm. Multistage Graphs using Forward and Backward Approach, Optimal Binary Search Tree, All Pair Shortest Path (Floyd Warshall Algorithm), Single Source Shortest Path (Bellman Ford Algorithm) and 0/1 Knapsack problem, were implemented using the Dynamic Programming Approach.