**Experiment No 3: Dynamic Programming**

**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) O/1 Knapsack

**Theory:**

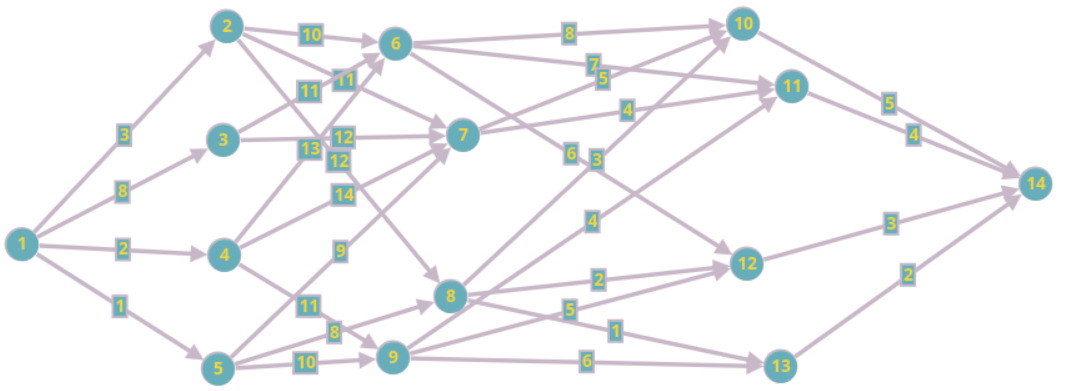
* Dynamic Programming is an algorithm design method that can be used when the solution to a problem can be viewed as the result of a sequence of decisions.
* Dynamic programming, like Divide and Conquer method, solves problems by combining the solutions to subproblems. Dynamic Programming applies when 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.

Date:

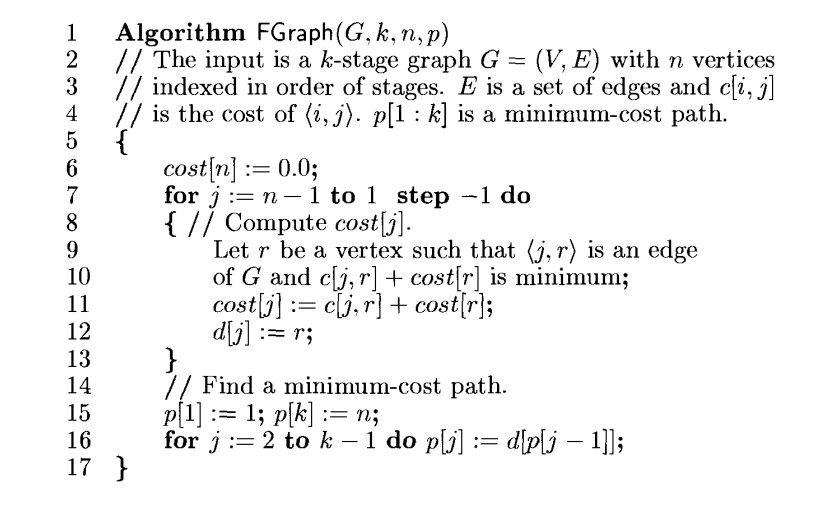
**i)Multistage Graph (Forward and Backward Approach)**

**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):**

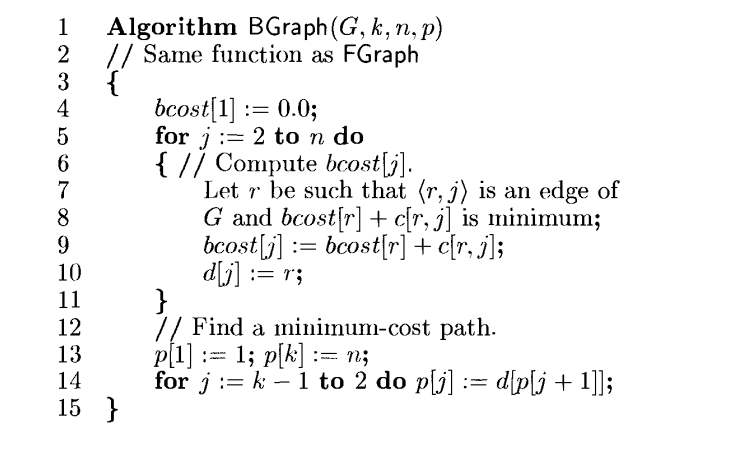


**Time Complexity (Forward Approach):** O(N2)

**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:**

#include <stdio.h>

#include <limits.h>

#define sint(x) scanf("%d", &x)

#define inf INT\_MAX

#define N 15

void fillinf(int a[][N], int n)

{

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

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

a[i][j] = inf;

}

void getmat(int a[][N], int n)

{

int w, x, y;

// printf("Enter -1 -1 to exit: ");

for (int i = 0; i < n \* (n - 1); i++)

{

sint(x);

sint(y);

if (x == -1 && y == -1)

return;

printf("Enter weight: ");

sint(w);

a[--x][--y] = w;

}

}

int min\_(int a, int b)

{

return (a = a < b ? a : b);

}

int fgraph(int a[][N], int n)

{

int dist[N]; // dist[i] will store the min distance from node i to the destination ie. node n-1

dist[n - 1] = 0;

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

{

dist[i] = inf;

// now check in front

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

{

if (a[i][j] == inf)

continue;

else

dist[i] = min\_(dist[i], a[i][j] + dist[j]);

}

}

return dist[0];

}

int bgraph(int a[][N], int n)

{

int dist[N];

dist[0] = 0;

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

{

dist[i] = inf;

for (int j = i; j >= 0; j--)

{

if (a[j][i] == inf)

continue;

else

{

dist[i] = min\_(dist[i], dist[j] + a[j][i]);

}

}

}

return dist[n - 1];

}

int main(int argc, char const \*argv[])

{

int n, a[N][N];

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

sint(n);

fillinf(a, n);

getmat(a, n);

int mincost = fgraph(a, n);

printf("The shortest path from 1 to %d using the forward approach is: %d\n", n, mincost);

mincost = 0;

mincost = bgraph(a, n);

printf("The shortest path from 1 to %d using the backward approach is: %d\n", n, mincost);

return 0;

}

**Output:**

C:\P Jeevesh Naidu\college\second year\lV sem\madf codes>cd "c:\P Jeevesh Naidu\college\second year\lV sem\madf codes\" && gcc mg.c -o mg && "c:\P Jeevesh Naidu\college\second year\lV sem\madf codes\"mg

Enter the number of vertices: 14

Enter the weights as u v w(enter -1 -1 to to stop taking input)

1 2 3

Enter weight: 1 3 8

Enter weight: 1 4 2

Enter weight: 1 5 1

Enter weight: 2 6 10

Enter weight: 2 7 11

Enter weight: 2 8 12

Enter weight: 3 6 11

Enter weight: 3 7 12

Enter weight: 4 6 13

Enter weight: 4 7 14

Enter weight: 4 9 11

Enter weight: 5 7 9

Enter weight: 5 8 8

Enter weight: 5 9 10

Enter weight: 6 10 8

Enter weight: 6 11 7

Enter weight: 6 12 6

Enter weight: 7 11 4

Enter weight: 8 10 3

Enter weight: 8 12 2

Enter weight: 8 13 1

Enter weight: 9 11 4

Enter weight: 9 12 5

Enter weight: 9 13 6

Enter weight: 10 14 5

Enter weight: 11 14 4

Enter weight: 12 14 3

Enter weight: 13 14 2

Enter weight: -1 -1

The shortest path from 1 to 14 using the forward approach is: 12

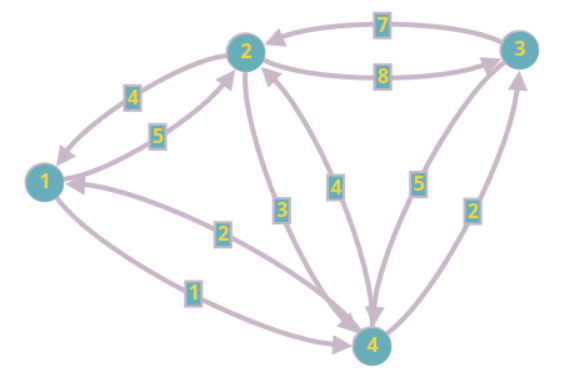
The shortest path from 1 to 14 using the backward approach is: 12

Date:

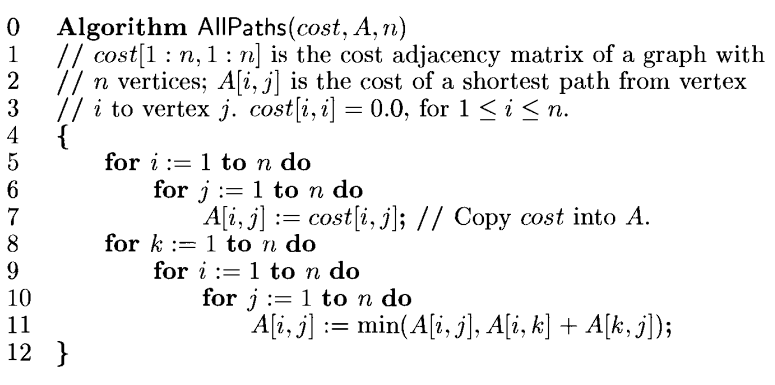
**ii) All Pairs Shortest Path**

**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>

#include <limits.h>

#define sint(x) scanf("%d", &x);

#define N 30

#define inf 99999

void fillinf(int a[][N], int n)

{

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

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

a[i][j] = inf;

}

void fillz(int a[][N], int n)

{

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

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

if (i == j)

a[i][j] = 0;

}

void getmat(int a[][N], int n)

{

int w, x, y;

// printf("Enter -1 -1 to exit: ");

for (int i = 0; i < n \* (n - 1); i++)

{

sint(x);

sint(y);

if (x == -1 && y == -1)

return;

// printf("Enter weight: ");

sint(w);

a[--x][--y] = w;

}

}

int \_min(int a, int b)

{

return a < b ? a : b;

}

void allpair(int cost[][N], int a[][N], int n)

{

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

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

cost[i][j] = a[i][j];

fillz(cost, n);

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

{

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

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

if (cost[i][k] + cost[k][j] < cost[i][j])

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

}

}

void showmat(int a[][N], int n)

{

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

{

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

{

if (a[i][j] == inf)

printf("inf ");

else

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

}

printf("\n");

}

}

int main(int argc, char const \*argv[])

{

int cost[N][N], a[N][N], n;

sint(n);

fillinf(a, n);

getmat(a, n);

allpair(cost, a, n);

printf("The shortest path matrix is\n");

showmat(cost, n);

return 0;}

**Output:**

C:\P Jeevesh Naidu\college\second year\lV sem\madf codes>cd "c:\P Jeevesh Naidu\college\second year\lV sem\madf codes\" && gcc apsp.c -o apsp && "c:\P Jeevesh Naidu\college\second year\lV sem\madf codes\"apsp

Enter the number of vertices - 4

Enter the weights as u v w(enter -1 -1 to to stop taking input)

1 2 5

1 4 1

2 1 4

2 3 8

2 4 3

3 2 7

3 4 5

4 1 2

4 2 4

4 3 2

-1 -1

The shortest path matrix is

0 5 3 1

4 0 5 3

7 7 0 5

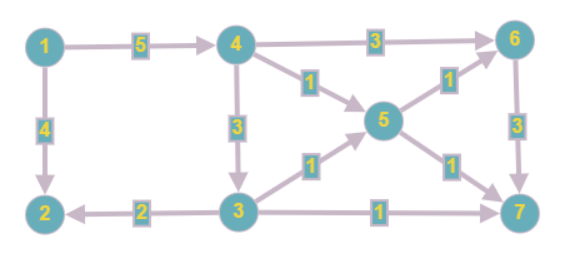
2 4 2 0

Date:

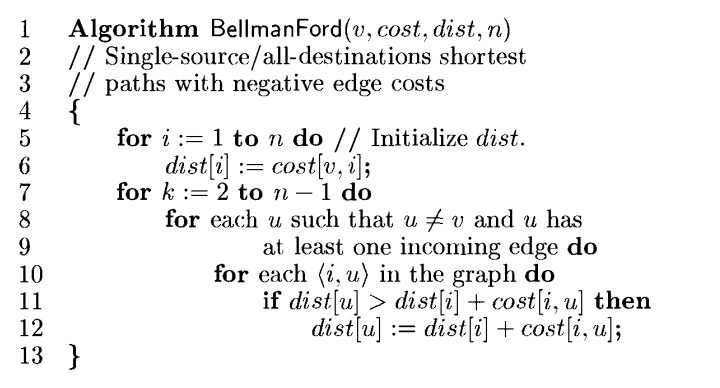
**iii) Single Source Shortest Path Algorithm**

**Problem Statement:**

Find the shortest path to each vertex from a single source vertex for the given graph.



**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 <limits.h>

#define N 50

#define inf 99999

#define sint(x) scanf("%d", &x);

void fillinf(int a[][N], int n)

{

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

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

a[i][j] = inf;

}

void getmat(int a[][N], int n)

{

int w, x, y;

// printf("Enter -1 -1 to exit: ");

for (int i = 0; i < n \* (n - 1); i++)

{

sint(x);

sint(y);

if (x == -1 && y == -1)

return;

// printf("Enter weight: ");

sint(w);

a[--x][--y] = w;

}

}

int \_min(int a, int b)

{

return a < b ? a : b;

}

void showmat(int a[N], int n)

{

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

{

if (a[i] == inf)

printf("inf\t");

else

printf("%d\t", a[i]);

}

}

int incoming(int u, int a[][N], int v)

{

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

if (a[i][u] != inf)

return 1;

return 0;

}

void show2dmat(int a[][N], int n)

{

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

{

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

{

if (a[i][j] == inf)

printf("inf ");

else

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

}

printf("\n");

}

}

void bellman(int src, int cost[][N], int v)

{

int dist[N];

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

dist[i] = cost[src][i]; // init dist arr

dist[src] = 0; // dist to src is zero

// printf("The cost matrix\n");

// show2dmat(cost, v);

// printf("\n");

printf("k = 1: ");

showmat(dist, v);

printf("\n");

for (int k = 2; k <= v - 1; k++) // this for loop is for the times

{

// printf("entered the k for loop\nk = %d\n", k);

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

{

// printf("entered the u for loop\nu = %d\n", u);

if (u != src && incoming(u, cost, v) == 1)

{

// printf("entered the if condition inside thee u for loop\n");

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

{

// printf("entered the i for loop\ni = %d\n", i);

if (cost[i][u] != inf)

{

// printf("entered the if condition inside the i for loop\n");

if (dist[u] > dist[i] + cost[i][u])

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

// printf("k=%d u=%d i=%d\n", k, u, i);

}

// showmat(dist, v);

// printf("\n");

}

}

// showmat(dist, v);

// printf("\n");

}

printf("k = %d: ", k);

showmat(dist, v);

printf("\n");

}

// showmat(dist, v);

}

int main(int argc, char const \*argv[])

{

int n, a[N][N], src;

sint(n);

fillinf(a, n);

getmat(a, n);

sint(src);

bellman(--src, a, n);

return 0;

}

**Output:**

C:\P Jeevesh Naidu\college\second year\lV sem\madf codes>cd "c:\P Jeevesh Naidu\college\second year\lV sem\madf codes\" && gcc bf.c -o bf && "c:\P Jeevesh Naidu\college\second year\lV sem\madf codes\"bf

Enter the number of vertices - 7

Enter the weights as u v w(enter -1 -1 to to stop taking input)

1 2 4

1 4 5

2 4 6

3 2 2

3 5 -4

3 7 -4

4 3 3

4 5 -1

4 6 -3

5 6 -2

5 7 -1

6 7 3

-1 -1

Enter the source vertex - 1

k = 1: 0 4 inf 5 inf inf inf

k = 2: 0 4 8 5 4 2 3

k = 3: 0 4 8 5 4 2 3

k = 4: 0 4 8 5 4 2 3

k = 5: 0 4 8 5 4 2 3

k = 6: 0 4 8 5 4 2 3

Date:

**iv) Optimal Binary Search Tree**

**Problem Statement:**

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

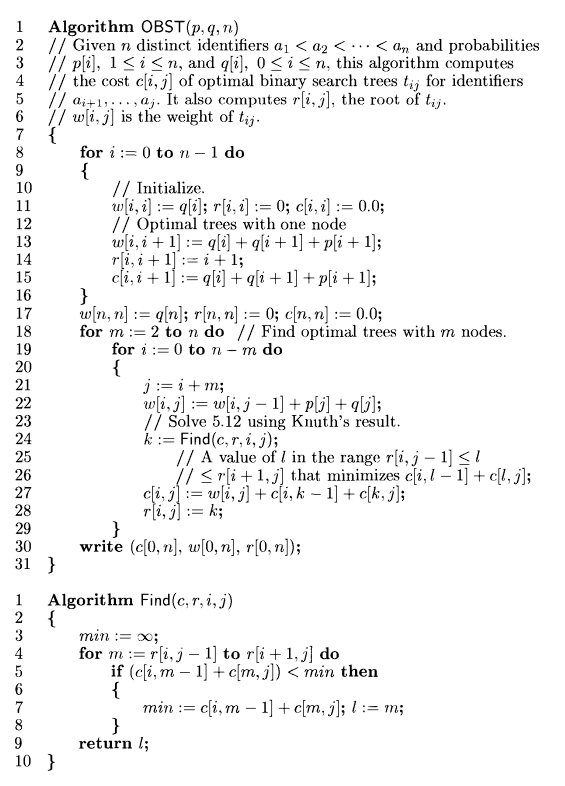
N = 5

(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:** O(N2)

**Space Complexity:** O(N2)

**Code:**

#include <stdio.h>

#include <limits.h>

#include <string.h>

#define sint(x) scanf("%d", &x);

#define N 20

#define inf INT\_MAX

#define new printf("\n")

struct key

{

int w, r, c;

};

void get0(int a[N], int n)

{

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

sint(a[i]);

}

void get1(int a[N], int n)

{

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

sint(a[i]);

}

void get1s(int a[N], int n)

{

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

gets(a[i]);

}

int find(struct key g[][N], int i, int j)

{

int min = inf, l;

for (int m = g[i][j - 1].r; m <= g[i + 1][j].r; m++)

{

if ((g[i][m - 1].c + g[m][j].c) < min)

{

min = g[i][m - 1].c + g[m][j].c;

l = m;

}

}

return l;

}

void obst(struct key g[][N], int q[N], int p[N], int n, char \*a[N])

{

int k;

printf("\tw\tc\tr");

new, new;

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

{

// init

g[i][i].w = q[i], g[i][i].r = g[i][i].c = 0;

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

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

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

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

}

new;

for (int i = 0, j = i + 1; i <= n - 1; i++, j++)

{

// init

// g[i][j].w = q[i], g[i][i].r = g[i][i].c = 0;

g[i][j].w = q[i] + q[j] + p[j];

g[i][j].r = i + 1;

g[i][j].c = q[i] + q[j] + p[j];

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

}

new;

g[n][n].w = q[n], g[n][n].r = g[n][n].c = 0;

for (int m = 2; m <= n; m++)

{

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

{

int j = i + m;

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

k = find(g, i, j);

g[i][j].c = g[i][j].w + g[i][k - 1].c + g[k][j].c;

g[i][j].r = k;

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

}

new;

}

printf("\nc[%d][%d] = %d\nw[%d][%d] = %d\nr[%d][%d] = %d\nMinimum cost of the BST is: %d\nroot: %s\n", 0, n, g[0][n].c, 0, n, g[0][n].w, 0, n, g[0][n].r, g[0][n].c, a[g[0][n].r - 1]);

}

int main(int argc, char const \*argv[])

{

int n;

char \*a[N] = {"Apr", "Mar", "May", "Oct", "Sept"};

int p[N], q[N];

struct key g[N][N];

printf("Enter the number of identifiers: ");

sint(n);

// printf("Enter the identifiers: ");

// get1s(a, n);

printf("Enter the probabilities for a succesfull search: ");

get1(p, n);

printf("Enter the probablities for an unsuccesfull search: ");

get0(q, n);

obst(g, q, p, n, a);

return 0;

}

**Output:**

C:\P Jeevesh Naidu\college\second year\lV sem\madf codes>cd "c:\P Jeevesh Naidu\college\second year\lV sem\madf codes\" && gcc obst.c -o obst && "c:\P Jeevesh Naidu\college\second year\lV sem\madf codes\"obst

Enter the number of identifiers: 5

Enter the probabilities for a successful search: 3

4

3

2

4

Enter the probabilities for an unsuccessful search: 4

4

5

w c r

00 4 0 0

11 4 0 0

22 5 0 0

33 4 0 0

44 5 0 0

55 4 0 0

01 11 11 1

12 13 13 2

23 12 12 3

34 11 11 4

45 13 13 5

02 20 31 2

13 20 32 2

24 19 30 3

35 19 30 5

03 27 50 2

14 27 51 3

25 27 52 4

04 34 75 2

15 35 78 3

05 42 103 3

c[0][5] = 103

w[0][5] = 42

root: May

Date:

**v) O/1 Knapsack**

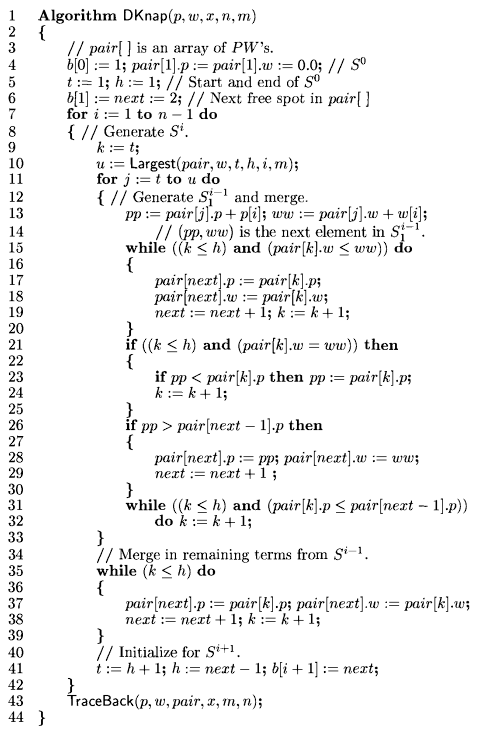
**Problem Statement:**

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

(w1 . . . w7) = (3, 1, 3, 4, 1, 1, 2)

(p1 . . . p7) = (18, 4, 6, 21, 7, 14, 13)

**Algorithm:**



**Time Complexity:**

**Space Complexity:**

**Code:**

#include <stdio.h>

#include <limits.h>

#define sint(x) scanf("%d", &x)

#define N 200

#define obj 100

int count = 0; // this is used to print the subsets

int size;

struct dp

{

int p, w, o[obj];

// here p = profit, w = weight and o array holds zeroes and ones, indicating whether the eobject has been included in the knapsack or not

};

void get(int a[N], int n)

{

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

sint(a[i]);

}

void purrrrrge(struct dp s[N])

// (=｀ω´=)...

{

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[i].w = -1;

else if (s[j].p < s[i].p && s[j].w >= s[i].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 knap(int m, int n, int p[N], int w[N])

{

struct dp s[N];

s[0].w = s[0].p = 0; // this represents an empty knapsack ie. no profit, no weight

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

s[0].o[i] = 0;

size = 1;

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

{

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

// j represents the index of the entry in the prev row that is being copied

// k represents the idex of the new entry that is beign created

{

// check if the entry in the prev row is valid

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

continue;

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

// check if the weight exceeds the knapsack cap

if (s[k].w > m)

{

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

continue;

}

// if it does not exceed, we proceed to update the profits

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

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

s[k].o[z] = s[j].o[z];

s[k].o[i] += 1;

}

purrrrrge(s);

printf("S^%d = { ", count);

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

{

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

continue;

else

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

}

printf("}");

printf("\n");

size \*= 2, count++;

}

purrrrrge(s);

int mw = -1, mp = -1, ind; // mp = max profit, mw = max weight, ind = index

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

{

if (s[i].w < mw)

continue;

else if (s[i].w > mw)

mw = s[i].w, mp = s[i].p, ind = i;

else

{

if (s[i].p > mp)

mp = s[i].p, ind = i;

}

}

printf("Max Profit: %d\nKnapsack filled at: %d\nObjects: ( ", mp, mw);

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

{

if (s[ind].o[i] != 1)

printf("0 ");

else

printf("%d ", s[ind].o[i]);

}

printf(")\n");

}

int main(int argc, char const \*argv[])

{

int m, n; // m = max knapsack size, n = number of items

printf("Enter the knapsack capacity: ");

sint(m);

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

sint(n);

int p[n], w[n]; // p is the array holding the profits and w the weights

printf("Enter the profit for each item\n");

get(p, n);

printf("Enter the weights for each item\n");

get(w, n);

knap(m, n, p, w);

return 0;

}

**Output:**

C:\P Jeevesh Naidu\college\second year\lV sem\madf codes>cd "c:\P Jeevesh Naidu\college\second year\lV sem\madf codes\" && gcc 1ks.c -o 1ks && "c:\P Jeevesh Naidu\college\second year\lV sem\madf codes\"1ks

Enter the knapsack capacity: 15

Enter the number of elements: 7

Enter the profit for each item

18 4 6 21 7 14 13

Enter the weights for each item

3 1 3 4 1 1 2

S^0 = { (0, 0) }

S^1 = { (0, 0) (18, 3) }

S^2 = { (0, 0) (18, 3) (4, 1) (22, 4) }

S^3 = { (0, 0) (18, 3) (4, 1) (22, 4) (24, 6) (28, 7) }

S^4 = { (0, 0) (18, 3) (4, 1) (22, 4) (39, 7) (25, 5) (43, 8) (45, 10) (49, 11) }

S^5 = { (0, 0) (18, 3) (39, 7) (7, 1) (25, 4) (11, 2) (29, 5) (46, 8) (32, 6) (50, 9) (52, 11) (56, 12) }

S^6 = { (0, 0) (14, 1) (32, 4) (53, 8) (21, 2) (39, 5) (25, 3) (43, 6) (60, 9) (46, 7) (64, 10) (66, 12) (70, 13) }

Max Profit: 83

Knapsack filled at: 15

Objects: ( 1 1 1 1 1 1 1 )

**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.