**Aim:** Write a program to implement Fractional knapsack using Greedy algorithm and 0/1 knapsack using dynamic programming. Show that Greedy strategy does not necessarily yield an optima lsolution over a dynamic programming approach.

# Fractional Knapsack Code:

#include <iostream> using namespace std; #include <bits/stdc++.h> typedef struct

{

int v; int w; float d;

} Item;

void input(Item items[], int sizeOfItems)

{

cout << "Enter total " << sizeOfItems << " items value and weight" << endl; for (int i = 0; i < sizeOfItems; i++)

{

cout << "Enter: " << i + 1 << " profit\t"; cin >> items[i].v;

cout << "Enter: " << i + 1 << " weight\t"; cin >> items[i].w;

}

}

void display(Item items[], int sizeOfItems)

{

int i;

cout << endl

<< "Profits: ";

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

{

cout << items[i].v << "\t";

}

cout << endl

<< "weight: ";

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

{

cout << items[i].w << "\t";

}

cout << endl;

}

bool compare(Item i1, Item i2)

{

return (i1.d > i2.d);

}

float knapsack(Item items[], int sizeOfItems, int w)

{

int i, j;

float totvalue = 0, totweight = 0; for (i = 0; i < sizeOfItems; i++)

{

items[i].d = (float)items[i].v / items[i].w;

}

sort(items, items + sizeOfItems, compare); for (i = 0; i < sizeOfItems; i++)

{

if (totweight + items[i].w <= w)

{

totvalue += items[i].v; totweight += items[i].w;

}

else

{

int wt = w - totweight; totvalue += (wt \* items[i].d);

totweight += wt; break;

}

}

cout << "Total weight in bag: " << totweight << endl; return totvalue;

}

int main()

{

int w; int n;

cout << "Enter number of items: "; cin >> n;

Item items[n];

cout << "\nEnter data: "; input(items, n); display(items, n);

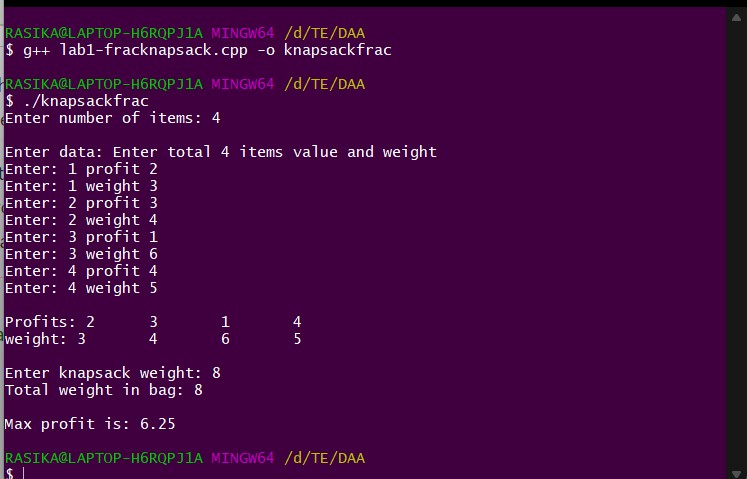
cout << "\nEnter knapsack weight: "; cin >> w;

float mxval = knapsack(items, n, w);

cout << "\nMax profit is: " << mxval << endl;

}

**Output:**



# 0/1 Knapsack Code:

#include <bits/stdc++.h> using namespace std; int max(int a, int b)

{

return (a > b) ? a : b;

}

int knapSack(int M, int wt[], int val[], int n)

{

int K[n + 1][M + 1];

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

{

for(int w = 0; w <= M; w++)

{

if (i == 0 || w == 0) K[i][w] = 0;

else if (wt[i - 1] <= w) K[i][w] = max(val[i - 1] +

K[i - 1][w - wt[i - 1]], K[i - 1][w]);

else

K[i][w] = K[i - 1][w];

}

}

return K[n][M];

}

int main()

{

int n;

cout << "Enter size: "; cin >> n;

int val[n]; int wt[n];

cout << "\nEnter Profit\tWeight:\n"; for (int i = 0; i < n; i++)

{

cin >> val[i] >> wt[i]; cout << endl;

}

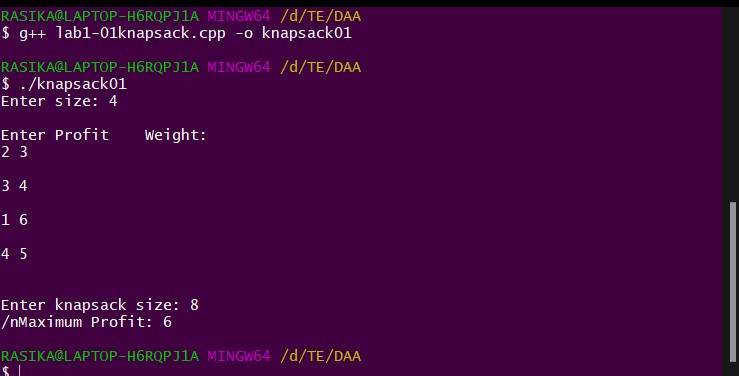
int M;

cout << "\nEnter knapsack size: "; cin >> M;

cout << "/n" << "Maximum Profit: " << knapSack(M, wt, val, n) << endl; return 0;

}

# Output:



**Aim:** Write a program to implement Bellman-Ford Algorithm using Dynamic Programming and verify the time complexity

**Code:**

#include <bits/stdc++.h> using namespace std;

// Each edge has 3 values source, destination and weight...

// structure to store source,destination and weight struct edge

{

int source, destination, weight;

};

int Bell\_Ford(int vertices, int edges, int src, vector<edge> &edge, vector<int> &dist)

{

int source, dest, weight, i, j; i = src;

dist[i - 1] = 0;

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

{

for (j = 0; j < edges; j++)

{

source = edge[j].source; dest = edge[j].destination; weight = edge[j].weight;

// if we can get a smaller value of dist[d] using this edge, replace it if (dist[source] != INT\_MAX && dist[source] + weight < dist[dest])

{

dist[dest] = dist[source] + weight;

}

}

}

// loop to detect negative cycle... for (j = 0; j < edges; j++)

{

source = edge[i].source; dest = edge[i].destination; weight = edge[i].weight;

if (dist[source] != INT\_MAX && dist[source] + weight < dist[dest])

{

return 0; // negative cycle...

}

}

return 1; // no negative cycle...

}

int main()

{

int vertices, edges; int source, dest, w; int src;

cout << "Enter the number of Vertices: "; cin >> vertices;

cout << "Enter the number of Edges: "; cin >> edges;

vector<edge> edge(edges);

cout << "Enter source, destination and weight of each Edge(space seprated): " << endl; for (int i = 0; i < edges; i++)

{

cin >> source >> dest >> edge[i].weight; edge[i].source = source - 1; edge[i].destination = dest - 1;

}

cout << "Enter source Vertex: "; cin >> src;

vector<int> dist(vertices, INT\_MAX);

int result = Bell\_Ford(vertices, edges, src, edge, dist); cout << endl;

if (result)

{

cout << "\nVertex\t\tDistance" << endl; for (int i = 0; i < vertices; i++)

{

cout << i + 1 << "\t\t" << dist[i] << endl;

}

}

else

{

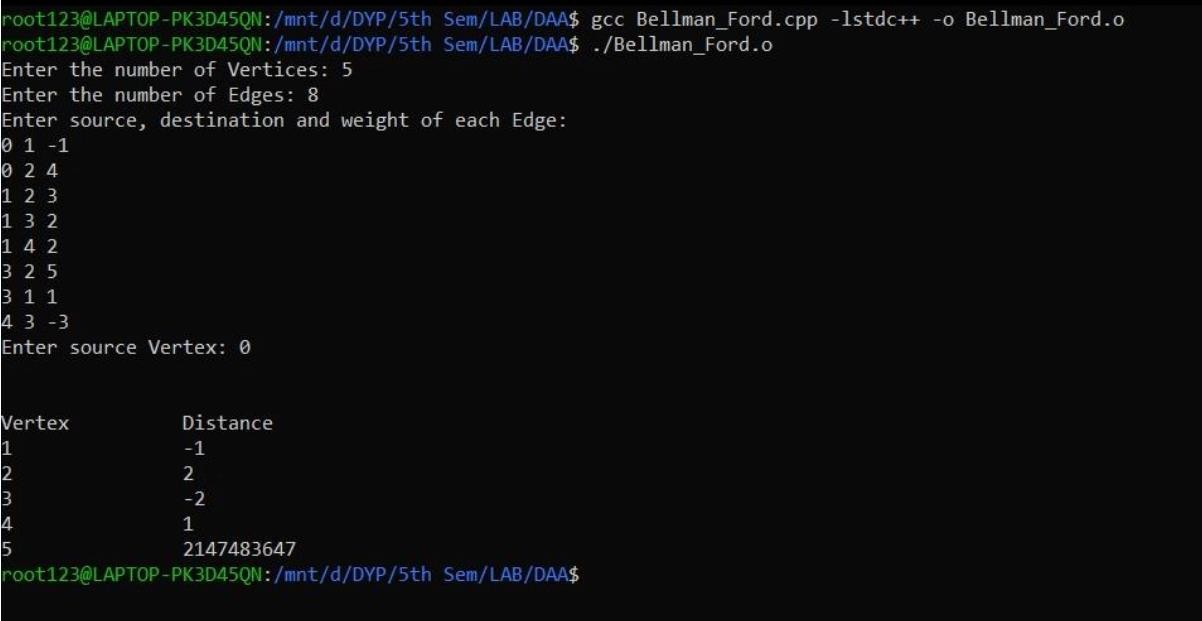
cout << "\nSorry, negative weight Cycle in Graph...! ";

}

return 0;

}

**Output:**



**Aim:** Write a recursive program to find the solution of placing n queens on the chessboard so that no two queens attack each other using Backtracking.

## Code:

#include<stdio.h> #include<conio.h> #include<math.h> int a[30],count=0; int place(int pos) {

int i;

for (i=1;i<pos;i++) {

if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos)))) return 0;

}

return 1;

}

void print\_sol(int n) { int i,j; count++;

printf("\n\nSolution #%d:\n",count); for (i=1;i<=n;i++) {

for (j=1;j<=n;j++) {

if(a[i]==j) printf("Q\t"); else printf("\*\t");

}

printf("\n");

}

}

void queen(int n) {

int k=1; a[k]=0;

while(k!=0) {

a[k]=a[k]+1;

while((a[k]<=n)&&!place(k)) a[k]++;

if(a[k]<=n) {

if(k==n) print\_sol(n); else {

k++; a[k]=0;

}

} else k--;

}

}

void main() {

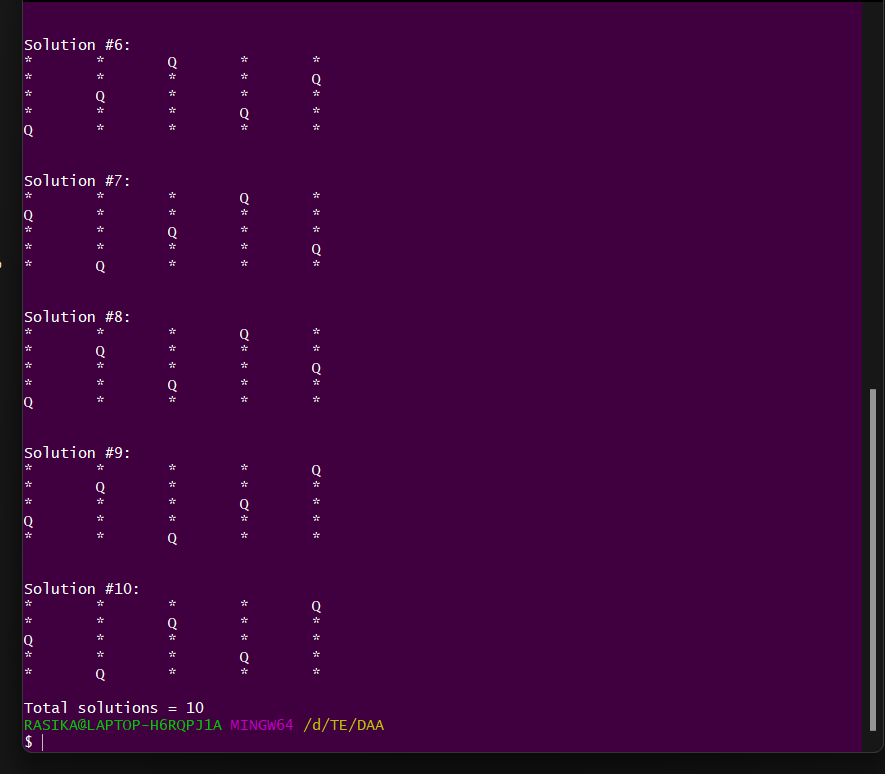
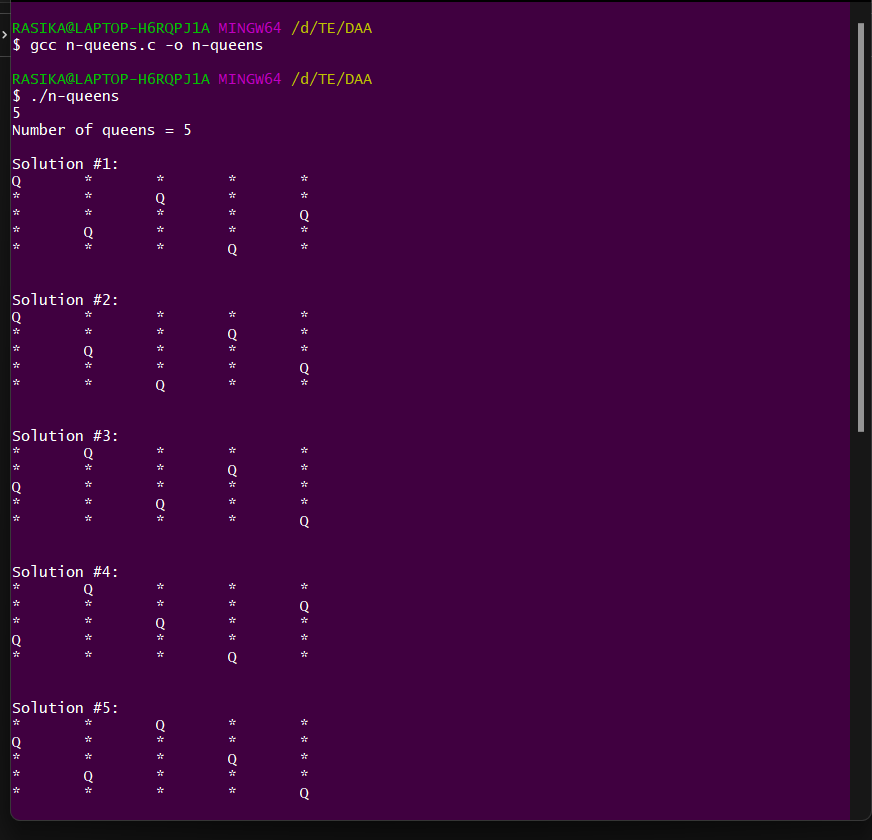
int i,n;; scanf("%d",&n);

printf("Number of queens = %d", n); queen(n);

printf("\nTotal solutions = %d",count);

}

**Output:**



**Aim:** Write a program to solve travelling salesman problem and to print the path and the cost using dynamic programming

## Code:

#include <stdio.h>

int ary[10][10], completed[10], n, cost = 0; void takeInput()

{

int i, j;

printf("Enter the number of villages: "); scanf("%d", &n);

printf("\nEnter the Cost Matrix\n"); for (i = 0; i < n; i++)

{

printf("\nEnter Elements of Row: %d\n", i + 1); for (j = 0; j < n; j++)

scanf("%d", &ary[i][j]); completed[i] = 0;

}

printf("\n\nThe cost list is:"); for (i = 0; i < n; i++)

{

printf("\n");

for (j = 0; j < n; j++) printf("\t%d", ary[i][j]);

}

}

int least(int c)

{

int i, nc = 999;

int min = 999, kmin;

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

{

if ((ary[c][i] != 0) && (completed[i] == 0))

if (ary[c][i] + ary[i][c] < min)

{

min = ary[i][0] + ary[c][i]; kmin = ary[c][i];

nc = i;

}

}

if (min != 999) cost += kmin;

return nc;

}

void mincost(int city)

{

int i, ncity; completed[city] = 1; printf("%d--->", city + 1); ncity = least(city);

if (ncity == 999)

{

ncity = 0;

printf("%d", ncity + 1); cost += ary[city][ncity]; return;

}

mincost(ncity);

}

int main()

{

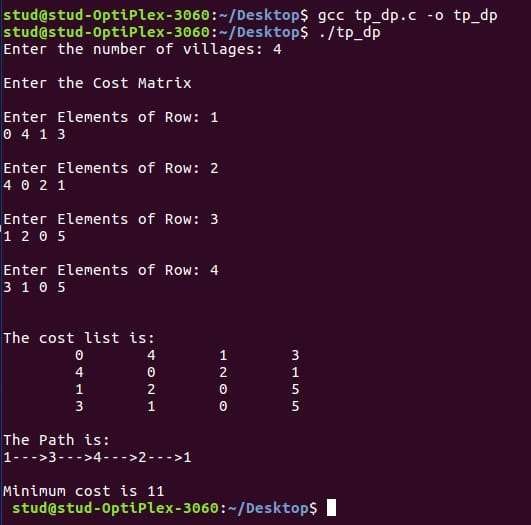
takeInput();

printf("\n\nThe Path is:\n");

mincost(0); // passing 0 because starting vertex printf("\n\nMinimum cost is %d\n ", cost); return 0;

}

## Output:



**Aim:** Write a program to solve travelling salesman problem and to print the path and the cost using branch and bound.

## Code:

#include <stdio.h> #include <conio.h> int main()

{

int cost[20][20], min, l, m, sr[20], sc[20], flag[20][20], i, j, k, rf[20], cf[20], n; int nrz[20], ncz[20], cn, a, noz, nrz1[20], ncz1[20], counter = 0; printf("\n\nEnter the total number of assignments:");

scanf("%d", &n);

printf("\nEnter the cost matrix\n"); for (i = 0; i < n; i++)

{

printf("\n");

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

{

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

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

}

}

printf("\n\n"); printf("Cost matrix:\n"); for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++) printf("\t%d\t", cost[i][j]);

printf("\n");

}

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

{

min = cost[i][0]; for (j = 0; j < n; j++)

{

if (min > cost[i][j])

min = cost[i][j];

}

for (j = 0; j < n; j++) cost[i][j] = cost[i][j] - min;

}

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

{

min = cost[0][i]; for (j = 0; j < n; j++)

{

if (min > cost[j][i])

min = cost[j][i];

}

for (j = 0; j < n; j++) cost[j][i] = cost[j][i] - min;

}

printf("\n\n");

printf("Cost matrix after row & column operation:\n"); for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++) printf("\t%d\t", cost[i][j]);

printf("\n");

}

repeat x;

a = 0;

noz = 0, min = 1000; for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++) flag[i][j] = 0;

}

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

{

cn = 0;

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

{

if (cost[i][j] == 0)

{

cn++; flag[i][j] = 1;

}

}

nrz[i] = cn;

noz = noz + cn;

}

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

{

cn = 0;

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

{

if (cost[j][i] == 0)

{

cn++; flag[j][i] = 1;

}

}

ncz[i] = cn; noz = noz + cn;

}

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

{

nrz1[i] = nrz[i];

ncz1[i] = ncz[i];

}

k = 0;

while (nrz[k] != 0 || ncz[k] != 0)

{

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

{

cn = 0;

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

{

if (flag[i][j] == 1) cn++;

nrz[i] = cn;

}

if (nrz[i] == 1)

{

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

{

if (flag[i][j] == 1)

{

flag[i][j] = 2;

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

{

if (flag[k][j] == 1)

flag[k][j] = 0;

}

}

}

}

}

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

{

cn = 0;

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

{

if (flag[j][i] == 1) cn++;

ncz[i] = cn;

}

if (ncz[i] == 1)

{

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

{

if (flag[j][i] == 1)

{

flag[j][i] = 2;

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

{

if (flag[j][k] == 1)

flag[j][k] = 0;

}

}

}

}

} k++;

}

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

{

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

{

if (flag[i][j] == 2) a++;

}

}

if (a == n)

{

printf("\nAssignments completed in order!!\n"); for (i = 0; i < n; i++)

{

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

{

if (flag[i][j] == 2) printf(

"%d->%d", i + 1, j + 1);

}

printf(

"\n");

}

getch();

exit(0);

}

else

{

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

{

rf[i] = 0, sr[i] = 0;

cf[i] = 0, sc[i] = 0;

}

for (k = n; (k > 0 && noz != 0); k--)

{

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

{

m = 0;

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

{

if ((flag[i][j] == 4) && (cost[i][j] == 0)) m++;

}

sr[i] = m;

}

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

{

if (nrz1[i] == k && nrz1[i] != sr[i])

{

rf[i] = 1;

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

{

if (cost[i][j] == 0)

flag[i][j] = 4;

}

noz = noz - k;

}

}

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

{

l = 0;

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

{

if ((flag[j][i] == 4) && (cost[j][i] == 0)) l++;

}

sc[i] = l;

}

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

{

if (ncz1[i] == k && ncz1[i] != sc[i])

{

cf[i] = 1;

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

{

if (cost[j][i] == 0)

flag[j][i] = 4;

}

noz = noz - k;

}

}

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

{

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

{

if (flag[i][j] != 3)

{

if (rf[i] == 1 && cf[j] == 1)

{

flag[i][j] = 3;

if (cost[i][j] == 0) noz = noz + 1;

}

}

}

}

}

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

{

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

{

if (rf[i] != 1 && cf[j] != 1)

{

if (min > cost[i][j])

min = cost[i][j];

}

}

}

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

{

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

{

if (rf[i] != 1 && cf[j] != 1)

cost[i][j] = cost[i][j] - min;

}

}

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

{

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

{

if (flag[i][j] == 3)

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

}

}

}

printf("\n\n"); if (counter < 10)

{

counter = counter + 1; printf("\n\nIntermediate Matrix:\n"); for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++) printf("\t%d\t", cost[i][j]);

printf("\n");

}

}

else

{

printf("\n\nOptimal solution to given problem is not possible"); getch();

return 0;

}

goto repeat x;

}

## Output:

