BRACT’s

Vishwakarma Institute of Information Technology, Pune

**Practical Implementation Sheet**

| **Department:** IT | **Semester:** IV | **Academic Year:** 2024-25 | **Practical No: 8** |
| --- | --- | --- | --- |
| **Class/ Division/ Batch: SY (B)- B3** | | **Roll no: 70** | |
| **Course:** Data Structures and Analysis of Algorithms | | **Name of Student**: Anushka Kadam | |

**Aim:** Branch and Bound:

a) Write a program to solve the travelling salesman problem. Print the path and the cost.

b) Implement branch and bound for the 0/1 Knapsack problem.

**Code: a) Write a program to solve the travelling salesman problem. Print the path and the cost.**

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

int tsp(vector<vector<int>>& graph, vector<bool>& visited, int pos, int n, int count, int cost, int start, vector<int>& path, vector<int>& bestPath, int& minCost)

{

if (count == n && graph[pos][start])

{

cost += graph[pos][start];

if (cost < minCost)

{

minCost = cost;

bestPath = path;

bestPath.push\_back(start);

}

return cost;

}

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

{

if (!visited[i] && graph[pos][i])

{

visited[i] = true;

path.push\_back(i);

tsp(graph, visited, i, n, count + 1, cost + graph[pos][i], start, path, bestPath, minCost);

visited[i] = false;

path.pop\_back();

}

}

return minCost;

}

int main()

{

int n;

cout << "Enter number of cities: ";

cin >> n;

vector<vector<int>> graph(n, vector<int>(n));

cout << "Enter cost matrix:\n";

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

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

cin >> graph[i][j];

vector<bool> visited(n, false);

vector<int> path, bestPath;

int minCost = INT\_MAX;

int start = 0;

visited[start] = true;

path.push\_back(start);

tsp(graph, visited, start, n, 1, 0, start, path, bestPath, minCost);

cout << "\nMinimum Cost: " << minCost << endl;

cout << "Path: ";

for (int city : bestPath)

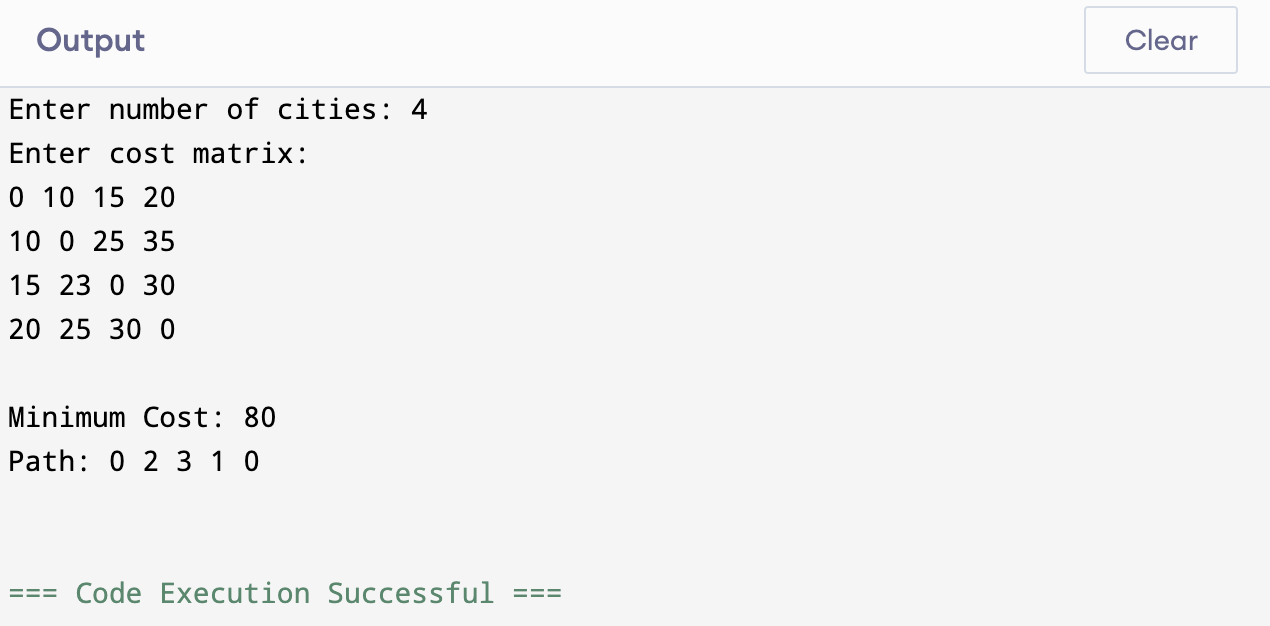
cout << city << " ";

cout << endl;

return 0;

}

**Output:**

****

**Code: b) Implement branch and bound for the 0/1 Knapsack problem**

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

using namespace std;

struct Item

{

int weight, value;

double ratio;

};

bool cmp(Item a, Item b)

{

return a.ratio > b.ratio;

}

struct Node

{

int level, profit, weight;

double bound;

};

double bound(Node u, int n, int W, vector<Item>& items)

{

if (u.weight >= W) return 0;

double profit\_bound = u.profit;

int j = u.level + 1;

int totweight = u.weight;

while ((j < n) && (totweight + items[j].weight <= W))

{

totweight += items[j].weight;

profit\_bound += items[j].value;

j++;

}

if (j < n)

profit\_bound += (W - totweight) \* items[j].ratio;

return profit\_bound;

}

int knapsack(int W, vector<Item>& items, int n)

{

sort(items.begin(), items.end(), cmp);

queue<Node> Q;

Node u, v;

u.level = -1;

u.profit = u.weight = 0;

u.bound = bound(u, n, W, items);

Q.push(u);

int maxProfit = 0;

while (!Q.empty())

{

u = Q.front();

Q.pop();

if (u.level == n - 1)

continue;

v.level = u.level + 1;

v.weight = u.weight + items[v.level].weight;

v.profit = u.profit + items[v.level].value;

if (v.weight <= W && v.profit > maxProfit)

maxProfit = v.profit;

v.bound = bound(v, n, W, items);

if (v.bound > maxProfit)

Q.push(v);

v.weight = u.weight;

v.profit = u.profit;

v.bound = bound(v, n, W, items);

if (v.bound > maxProfit)

Q.push(v);

}

return maxProfit;

}

int main()

{

int n, W;

cout << "Enter number of items: ";

cin >> n;

vector<Item> items(n);

cout << "Enter weights and values of items:\n";

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

{

cin >> items[i].weight >> items[i].value;

items[i].ratio = (double)items[i].value / items[i].weight;

}

cout << "Enter maximum capacity of knapsack: ";

cin >> W;

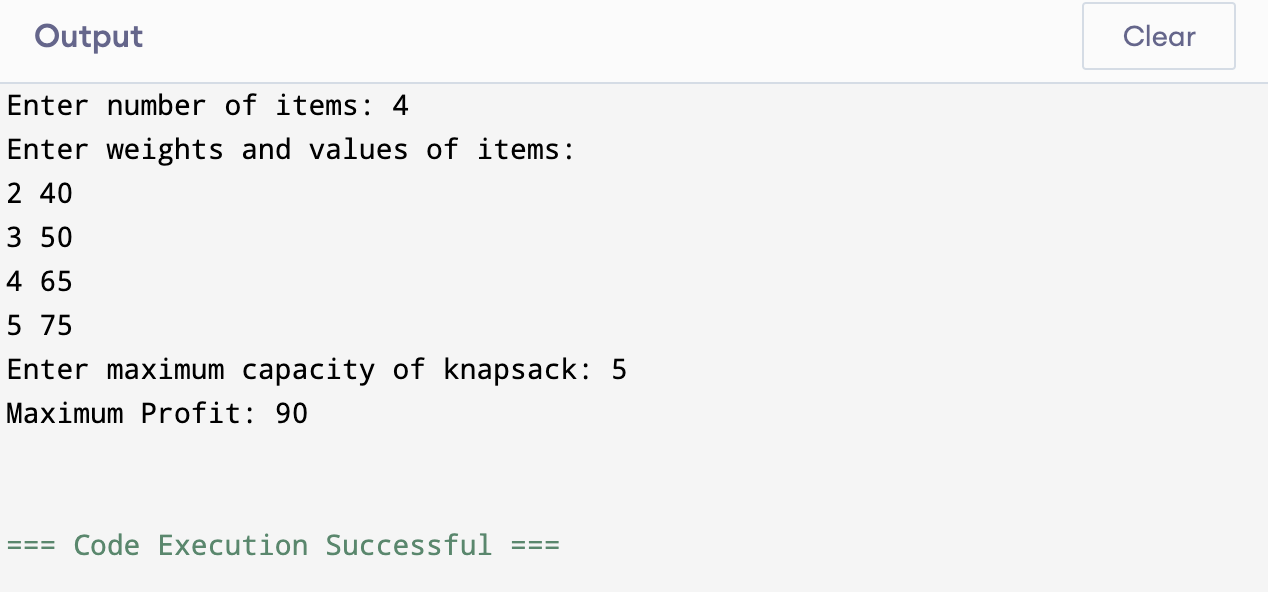
int maxProfit = knapsack(W, items, n);

cout << "Maximum Profit: " << maxProfit << endl;

return 0;

}

**OUTPUT:**

****