**Raahat Arora (Week 2)**

**230957216**

**Roll no - 72**

**Q1-**

#include <iostream>

using namespace std;

int Knapsack(int knapsack, int w[], int v[], int n){

if (n == 0 || knapsack == 0)

return 0;

else if(w[n - 1] > knapsack)

return Knapsack(knapsack, w, v, n - 1);

else

return max(Knapsack(knapsack, w, v, n - 1), v[n - 1] + Knapsack(knapsack - w[n - 1], w, v, n - 1));

}

int main(){

int knapsack, n;

cout << "Enter the number of items: ";

cin >> n;

int w[n];

int v[n];

cout << "Enter the knapsack capacity: ";

cin >> knapsack;

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

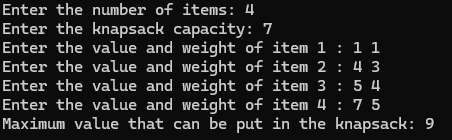
cout << "Enter the value and weight of item " << (i + 1) << " : ";

cin >> v[i] >> w[i];

}

cout << "Maximum value that can be put in the knapsack: " << Knapsack(knapsack, w, v, n) << endl;

}



**Q2-**

#include <iostream>

#include <climits>

using namespace std;

int calculateCost(int costMatrix[4][4], int assignment[4]) {

int totalCost = 0;

for (int worker = 0; worker < 4; ++worker) {

totalCost += costMatrix[worker][assignment[worker]];

}

return totalCost;

}

void generatePermutations(int costMatrix[4][4], int assignment[4], bool used[4], int currentJob, int &minCost, int bestAssignment[4]) {

if (currentJob == 4) {

int currentCost = calculateCost(costMatrix, assignment);

if (currentCost < minCost) {

minCost = currentCost;

for (int i = 0; i < 4; ++i) {

bestAssignment[i] = assignment[i];

}

}

return;

}

for (int job = 0; job < 4; ++job) {

if (!used[job]) {

used[job] = true;

assignment[currentJob] = job;

generatePermutations(costMatrix, assignment, used, currentJob + 1, minCost, bestAssignment);

used[job] = false;

}

}

}

int main() {

int costMatrix[4][4] = {

{9, 2, 7, 8},

{6, 4, 3, 7},

{3, 8, 5, 6},

{7, 6, 4, 3}

};

int assignment[4];

bool used[4] = {false};

int minCost = INT\_MAX;

int bestAssignment[4];

generatePermutations(costMatrix, assignment, used, 0, minCost, bestAssignment);

cout << "Optimal Assignment: ";

for (int i = 0; i < 4; ++i) {

cout << "Worker " << i + 1 << " -> Job " << bestAssignment[i] + 1 << ", ";

}

cout << endl;

cout << "Minimum Cost: " << minCost << endl;

return 0;

}



**Q3-**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

class Graph {

public:

int V;

vector<vector<int>> adjList;

Graph(int V) {

this->V = V;

adjList.resize(V);

}

void addEdge(int u, int v) {

adjList[u].push\_back(v);

adjList[v].push\_back(u);

}

void BFS(int start) {

vector<bool> visited(V, false);

queue<int> q;

vector<int> bfsOrder;

visited[start] = true;

q.push(start);

while (!q.empty()) {

int node = q.front();

q.pop();

bfsOrder.push\_back(node);

for (int neighbor : adjList[node]) {

if (!visited[neighbor]) {

visited[neighbor] = true;

q.push(neighbor);

}

}

}

cout << "BFS Order: ";

for (int i : bfsOrder) {

cout << i << " ";

}

cout << endl;

}

};

int main() {

Graph g(6);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(1, 4);

g.addEdge(2, 5);

cout << "BFS starting from vertex 0:" << endl;

g.BFS(0);

return 0;

}



**Q4-**

#include <iostream>

#include <vector>

#include <stack>

using namespace std;

class Graph {

public:

int V;

vector<vector<int>> adjList;

Graph(int V) {

this->V = V;

adjList.resize(V);

}

void addEdge(int u, int v) {

adjList[u].push\_back(v);

adjList[v].push\_back(u);

}

void DFS(int start) {

vector<bool> visited(V, false);

stack<int> stk;

vector<int> pushOrder;

vector<int> popOrder;

stk.push(start);

visited[start] = true;

pushOrder.push\_back(start);

while (!stk.empty()) {

int node = stk.top();

stk.pop();

popOrder.push\_back(node);

for (int neighbor : adjList[node]) {

if (!visited[neighbor]) {

visited[neighbor] = true;

stk.push(neighbor);

pushOrder.push\_back(neighbor);

}

}

}

cout << "Push Order: ";

for (int i : pushOrder) {

cout << i << " ";

}

cout << endl;

cout << "Pop Order: ";

for (int i : popOrder) {

cout << i << " ";

}

cout << endl;

}

};

int main() {

Graph g(6);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(1, 4);

g.addEdge(2, 5);

cout << "DFS starting from vertex 0:" << endl;

g.DFS(0);

return 0;

}

