Name- PRAVEER RAJ

Sec.- Maths and Computing

Roll- 1

Reg- 230957002

1) Write a program to implement Knapsack problem using brute-force design technique and analyze its time efficiency. Knapsack Problem: Given n items of known weights w1, w2 ... wn values v1, v2, ... vn and a knapsack of capacity B, find the most valuable subset of items that fit into the knapsack.

--------------------------------------------------------------------------------------------------------------------------------------

#include <iostream>

using namespace std;

int knapSack(int W, int wt[], int val[], int n) {

if (n == 0 || W == 0)

return 0;

if (wt[n - 1] > W)

return knapSack(W, wt, val, n - 1);

return max(knapSack(W, wt, val, n - 1),

val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1));

}

int main() {

int profit[] = {60, 100, 222, 103};

int weight[] = {10, 20, 30, 50};

int W = 50;

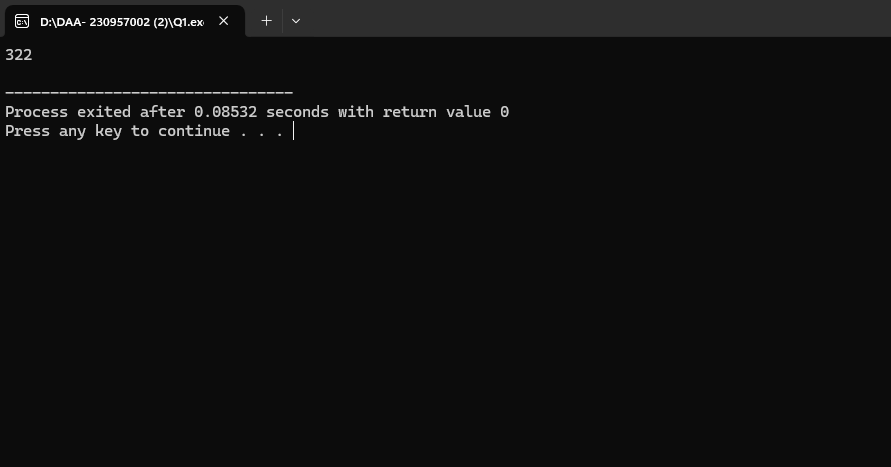
int n = sizeof(profit) / sizeof(profit[0]);

cout << knapSack(W, weight, profit, n) << endl;

return 0;

}

***OUTPUT--***



2) Write a program for assignment problem by brute-force technique. Analyze its time efficiency

#include <iostream>

#include <algorithm>

using namespace std;

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

int totalCost = 0;

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

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

}

return totalCost;

}

int assignmentProblemBruteForce(int costMatrix[][4], int n) {

int assignment[n];

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

assignment[i] = i;

}

int minCost = 1000000;

do {

int currentCost = calculateCost(costMatrix, assignment, n);

minCost = min(minCost, currentCost);

} while (next\_permutation(assignment, assignment + n));

return minCost;

}

int main() {

int costMatrix[4][4] = {

{10, 2, 8, 12},

{4, 5, 7, 9},

{6, 3, 9, 11},

{12, 6, 4, 10}

};

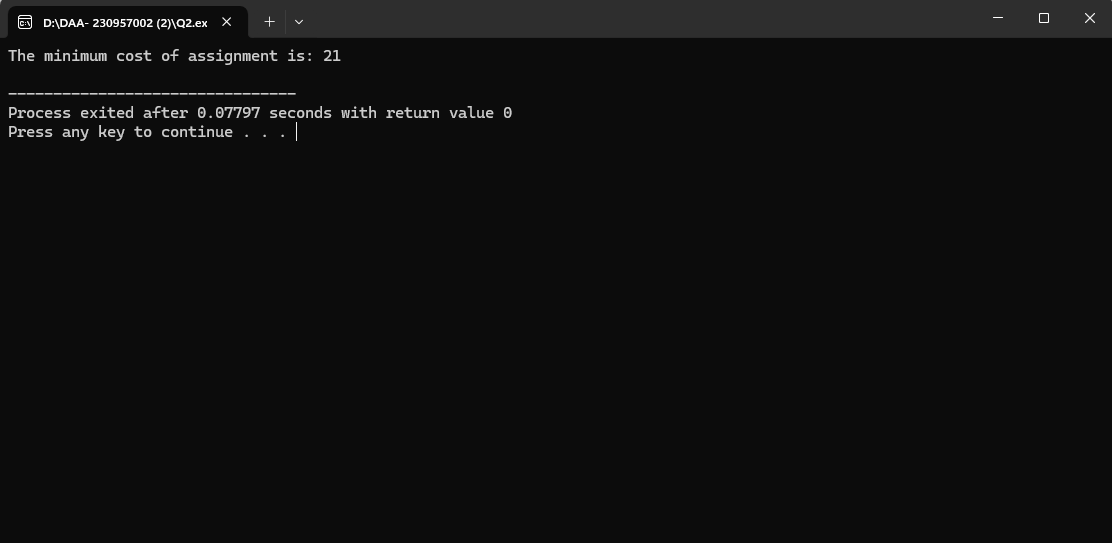
int n = 4;

cout << "The minimum cost of assignment is: " << assignmentProblemBruteForce(costMatrix, n) << endl;

return 0;

}

***OUTPUT—***

******

3) Write a program for depth-first search of a graph. Identify the push and pop order of vertices.

#include <iostream>

#include <stack>

using namespace std;

class Graph {

public:

int V;

int \*\*adj;

Graph(int V) {

this->V = V;

adj = new int\*[V];

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

adj[i] = new int[V]();

}

}

void addEdge(int u, int v) {

adj[u][v] = 1;

}

void dfs(int start) {

bool \*visited = new bool[V]();

stack<int> s;

int pushIdx = 0, popIdx = 0;

int \*pushOrder = new int[V], \*popOrder = new int[V];

s.push(start);

visited[start] = true;

pushOrder[pushIdx++] = start;

while (!s.empty()) {

int node = s.top();

s.pop();

popOrder[popIdx++] = node;

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

if (adj[node][i] == 1 && !visited[i]) {

s.push(i);

visited[i] = true;

pushOrder[pushIdx++] = i;

}

}

}

for (int i = 0; i < pushIdx; i++) cout << pushOrder[i] << " ";

cout << "\n";

for (int i = 0; i < popIdx; i++) cout << popOrder[i] << " ";

cout << "\n";

delete[] visited;

delete[] pushOrder;

delete[] popOrder;

}

};

int main() {

Graph g(6);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(1, 4);

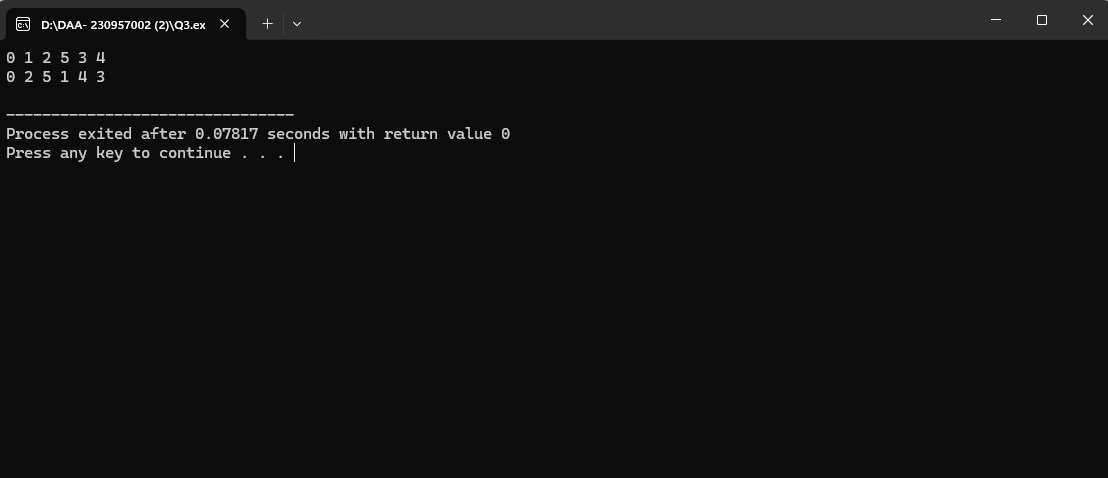
g.addEdge(2, 5);

g.dfs(0);

return 0;

}

***OUTPUT—***

******

4) Write a program for breadth-first search of a graph.

#include <iostream>

#include <queue>

using namespace std;

class Graph {

public:

int V;

int \*\*adj;

Graph(int V) {

this->V = V;

adj = new int\*[V];

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

adj[i] = new int[V]();

}

}

void addEdge(int u, int v) {

adj[u][v] = 1;

adj[v][u] = 1; // Assuming an undirected graph

}

void bfs(int start) {

bool \*visited = new bool[V]();

queue<int> q;

visited[start] = true;

q.push(start);

while (!q.empty()) {

int node = q.front();

q.pop();

cout << node << " ";

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

if (adj[node][i] == 1 && !visited[i]) {

q.push(i);

visited[i] = true;

}

}

}

cout << endl;

delete[] visited;

}

};

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);

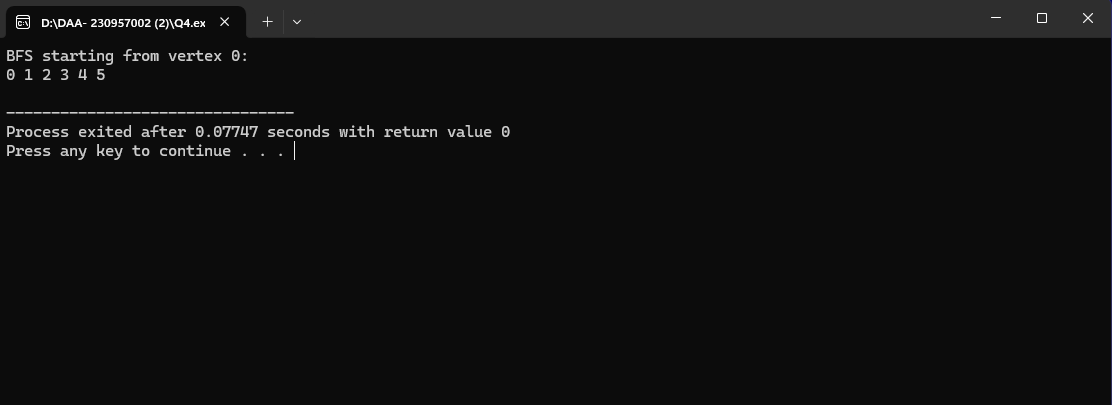
cout << "BFS starting from vertex 0:" << endl;

g.bfs(0);

return 0;

}

***OUTPUT—***

******