TOPIC:

—------------------------------------------------------------1.BFS

2.DFS

3.PRIM’S

4.0/1 KNAPSACK

5.FRACTIONAL KNAPSACK

—------------------------------------------------------------

Name: Md. Samiul Alam

ID: 0322320105101032

BATCH: 23rd

SEMESTER: 3rd

Solve —----------------------------------------------------

// BFS - Breadth First Search

#include <iostream>

#include <queue>

using namespace std;

void BFS(int graph[][100], int visited[], int n, int startVertex) {

queue<int> q;

visited[startVertex] = 1;

q.push(startVertex);

while (!q.empty()) {

int node = q.front();

q.pop();

cout << node << " ";

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

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

visited[i] = 1;

q.push(i);

}

}

}

}

int main() {

int n, e;

cout << "Enter number of vertices: ";

cin >> n;

cout << "Enter number of edges: ";

cin >> e;

int graph[100][100] = {0};

cout << "Enter edges:\n";

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

int u, v;

cin >> u >> v;

graph[u][v] = 1;

graph[v][u] = 1; // For undirected graph

}

int visited[100] = {0};

cout << "BFS traversal starting from vertex 0:\n";

BFS(graph, visited, n, 0);

return 0;

}

**Input:-**

Enter number of vertices: 5

Enter number of edges: 6

Enter edges:

0 1

0 2

1 3

1 4

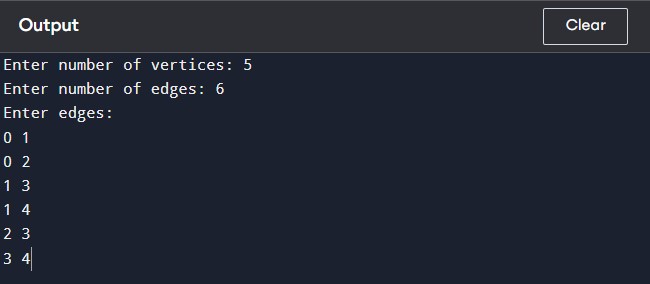
2 3

3 4

**Output:-**

BFS traversal starting from vertex 0:

0 1 2 3 4





// DFS - Depth First Search

#include <iostream>

using namespace std;

void DFS(int graph[][100], int visited[], int v, int startVertex) {

visited[startVertex] = 1;

cout << startVertex << " ";

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

if(graph[startVertex][i] == 1 && !visited[i]) {

DFS(graph, visited, v, i);

}

}

}

int main() {

int v, e;

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

cin >> v;

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

cin >> e;

int graph[100][100] = {0};

cout << "Enter edges: \n";

for(int I = 0; I <e; i++) {

int u, v;

cin >> u >> v;

graph[u][v] = 1;

graph[v][u]=1; //For undirected graph

}

int visited[100] = {0};

cout << "DFS traversal start from the vertex 0: ";

DFS(graph, visited, v, 0);

return 0;

}

**Input:-**

Enter number of vertices: 5

Enter number of edges: 6

Enter edges:

0 1

0 2

1 3

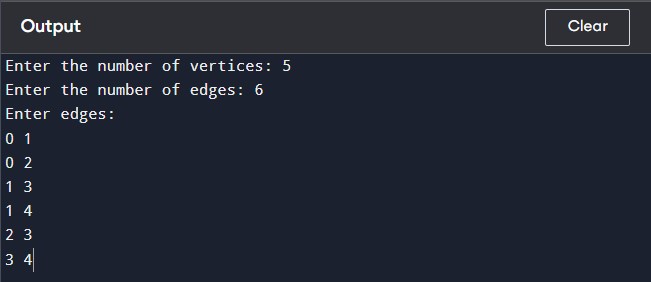
1 4

2 3

3 4

**Output:-**

DFS traversal start from the vertex 0: 0 1 3 4 2





// PRIM’S

#include <iostream>

#include <limits.h>

using namespace std;

int parent[100], key[100];

bool mstSet[100];

int minKey(int key[], bool mstSet[], int V) {

int min = INT\_MAX, min\_index;

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

if (mstSet[v] == false && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

void printMST(int parent[], int graph[100][100], int V) {

cout << "Edge \tWeight\n";

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

cout << parent[i] << " - " << i << "\t" << graph[i][parent[i]] << endl;

}

void primMST(int graph[100][100], int V) {

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

key[i] = INT\_MAX, mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; count++) {

int u = minKey(key, mstSet, V);

mstSet[u] = true;

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

if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v];

}

printMST(parent, graph, V);

}

int main() {

int V;

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

cin >> V;

int graph[100][100];

cout << "Enter the adjacency matrix:\n";

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

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

cin >> graph[i][j];

}

}

primMST(graph, V);

return 0;

}

**Input:-**

Enter the number of vertices: 5

Enter the adjacency matrix:

0 2 0 6 0

2 0 3 8 5

0 3 0 0 7

6 8 0 0 9

0 5 7 9 0

**Output:-**

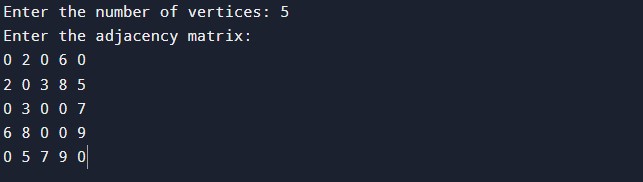
Edge Weight

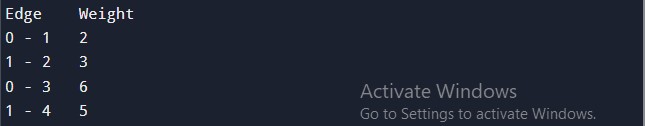
0 - 1 2

1 - 2 3

0 - 3 6

1 - 4 5





// Knapsack

#include <iostream>

using namespace std;

double knapsack(int c, int n, int value[], int weight[]) {

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

// build table K[][] bottom up manner

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

for(int w=0; w<=c; w++) {

if(i == 0 || w == 0) {

K[i][w]=0;

} else if(weight[i-1] <= w) {

K[i][w] = max(value[i-1]+K[i-1][w-weight[i-1]], K[i-1][w]);

} else {

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

}

}

}

return K[n][c];

}

int main() {

int c, n, i;

cout << "Enter the maximum capacity(weight) : ";

cin >> c;

cout << "Enter the total item numbers: ";

cin >> n;

int value[n], weight[n];

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

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

cin >> value[i];

}

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

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

cin >> weight[i];

}

double maxValue = knapsack(c, n, value, weight);

cout << "Maximum Value = " << maxValue;

return 0;

}

**Input:-**

Enter the maximum capacity(weight): 50

Enter the total item numbers: 3

Enter the value of items:

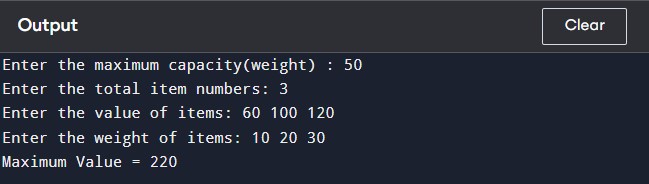
60 100 120

Enter the weight of items:

10 20 30

**Output:-**

Maximum Value = 220



// Partial Knapsack

#include <iostream>

#include <algorithm>

using namespace std;

struct Item {

int value, weight;

};

bool compare(Item a, Item b) {

double r1 = (double)a.value / a.weight;

double r2 = (double)b.value / b.weight;

return r1 > r2;

}

double fractionalKnapsack(int capacity, Item items[], int total) {

sort(items, items+total, compare);

double totalValue = 0.0;

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

if(items[i].weight <= capacity) {

capacity -= items[i].weight;

totalValue += items[i].value;

} else {

totalValue += items[i].value \* ((double) capacity / items[i].weight);

break;

}

}

return totalValue;

}

int main() {

int total, capacity;

cout << "Enter the maximum capacity(weight) of items: ";

cin >> capacity;

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

cin >> total;

Item items[total];

cout << "Enter the items value: ";

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

cin >> items[i].value;

}

cout << "Enter the items weight: ";

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

cin >> items[i].weight;

}

double maxValue = fractionalKnapsack(capacity, items, total);

cout << "Maxvalue = " << maxValue;

return 0;

}

**Input:-**

Enter the maximum capacity(weight) of items: 50

Enter the total number of items: 3

Enter the items value:

60 100 120

Enter the items weight:

10 20 30

**Output:-**

Maxvalue = 240

