|  |  |  |
| --- | --- | --- |
| 16 | Given a (directed/undirected) graph, design an algorithm and implement it using a program to find if a path exists between two given vertices or not. (Hint: use DFS) |  |
| 17 | Given a graph, design an algorithm and implement it using a program to find if a graph is bipartite or not. (Hint: use BFS) |  |
| 18 | Given a directed graph, design an algorithm and implement it using a program to find whether cycle exists in the graph or not. |  |
| 19 | After end term examination, Akshay wants to party with his friends. All his friends are living as paying guest and it has been decided to first gather at Akshay’s house and then move towards party location. The problem is that no one knows the exact address of his house in the city. Akshay as a computer science wizard knows how to apply his theory subjects in his real life and came up with an amazing idea to help his friends. He draws a graph by looking in to location of his house and his friends’ location (as a node in the graph) on a map. He wishes to find out shortest distance and path covering that distance from each of his friend’s location to his house and then whatsapp them this path so that they can reach his house in minimum time. Akshay has developed the program that implements Dijkstra’s algorithm but not sure about correctness of results. Can you also implement the same algorithm and verify the correctness of Akshay’s results? |  |
| 20 | Design an algorithm and implement it using a program to solve previous question's problem using Bellman- Ford's shortest path algorithm. |  |
| 21 | Given a directed graph with two vertices ( source and destination). Design an algorithm and implement it using a program to find the weight of the shortest path from source to destination with exactly k edges on the path. |  |
| 22 | Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm which will find minimum cost required to connect these cities. (use Prim's algorithm) |  |
| 23 | Implement the previous problem using Kruskal's algorithm. |  |

|  |  |  |
| --- | --- | --- |
| 24 | Assume that same road construction project is given to another person. The amount he will earn from this project is directly proportional to the budget of the project. This person is greedy, so he decided to maximize the budget by constructing those roads who have highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project. |  |
| 25 | Given a graph, Design an algorithm and implement it using a program to implement FloydWarshall all pair shortest path algorithm. |  |
| 26 | Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. You have to Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight w and has maximum value. You can take fractions of items,i.e. the items can be broken into smaller pieces so that you have to carry only a fraction xi of item i, where 0 ≤xi≤ 1. |  |
| 27 | Given an array of elements. Assume arr[i] represents the size of file i. Write an algorithm and a program to merge all these files into single file with minimum computation. For given two files  A and B with sizes m and n, computation cost of merging them is O(m+n). (Hint: use greedy approach) |  |
| 28 | Given a list of activities with their starting time and finishing time. Your goal is to select maximum number of activities that can be performed by a single person such that selected activities must be non-conflicting. Any activity is said to be non-conflicting if starting time of an activity is greater than or equal to the finishing time of the other activity. Assume that a person can only work on a single activity at a time. |  |
| 29 | Given a long list of tasks. Each task takes specific time to accomplish it and each task has a deadline associated with it. You have to design an algorithm and implement it using a program to find maximum number of tasks that can be completed without crossing their deadlines and also find list of selected tasks. |  |
| 30 | Given an unsorted array of elements, design an algorithm and implement it using a program to find whether majority element exists or not. Also find median of the array. A majority element is an element that appears more than n/2 times, where n is the size of array. |  |
| 31 | Given a sequence of matrices, write an algorithm to find most efficient way to multiply these matrices together. To find the optimal solution, you need to find the order in which these matrices should be multiplied. |  |
| 32 | Given a set of available types of coins. Let suppose you have infinite supply of each type of coin. For a given value N, you have to Design an algorithm and implement it using a program to find number of ways in which these coins can be added to make sum value equals to N. |  |
| 33 | Given a set of elements, you have to partition the set into two subsets such that the sum of elements in both subsets is same. Design an algorithm and implement it using a program to solve this problem. |  |

|  |  |  |
| --- | --- | --- |
| 34 | Given two sequences, Design an algorithm and implement it using a program to find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. |  |
| 35 | Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight <= w and has maximum value. Here, you cannot break an item i.e. either pick the complete item or don't pick it. (0-1 property). |  |
| 36 | Given a string of characters, design an algorithm and implement it using a program to print all possible permutations of the string in lexicographic order. |  |
| 37 | Given an array of characters, you have to find distinct characters from this array. Design an algorithm and implement it using a program to solve this problem using hashing. (Time  Complexity = O(n)) |  |
| 38 | Given an array of integers of size n, design an algorithm and write a program to check whether this array contains duplicate within a small window of size k < n. |  |
| 39 | Given an array of nonnegative integers, Design an algorithm and implement it using a program to find two pairs (a,b) and (c,d) such that a\*b = c\*d, where a, b, c and d are distinct elements of array. |  |
| 40 | Given a number n, write an algorithm and a program to find nth ugly number. Ugly numbers are those numbers whose only prime factors are 2, 3 or 5. The sequence 1, 2, 3, 4, 5, 6, 8, 9,  10, 12, 15, 16, 18, 20, 24,. is sequence of ugly numbers. |  |
| 41 | Given a directed graph, write an algorithm and a program to find mother vertex in a graph. A mother vertex is a vertex v such that there exists a path from v to all other vertices of the graph. |  |

# Program 15

**Problem Statement:** You have been given two sorted integer arrays of size m and n. Design an algorithm and implement it using a program to find a list of elements which are common to both. **Algorithm:**

printIntersection(arr1[] , arr2[], m(size of arr1) , n(size of arr2) )

{ i = 0 j = 0

WHILE ( i < m and j < n)

{

IF ( arr1[i] < arr2[j] )

i++

ELSE IF( arr2[j] < arr1[i] ) j++

ELSE

{

print arr2[j] i++ j++

}

}

}

**Complexity Analysis: Time Complexity:**

Worst Case: When data is skewed and range is large

Best Case: When all elements(characters) are same𝑂(𝑚 + 𝑛) T(n) = [ Linear] Average Case: N & K equally dominantΩ(𝑚 + 𝑛) T(n)= [ Linear] T(n) = θ(𝑚 + 𝑛) [ Linear]

**Space Complexity:**

Worst Case = Best Case = Average case = θ(1)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

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

void printIntersection(int arr1[], int arr2[], int m, int n)

{ int i = 0, j = 0; while(i < m && j < n){

if(arr1[i] < arr2[j]) i++; else if(arr2[j] < arr1[i]) j++; else{

cout << arr2[j] << " "; i++; j++;

}

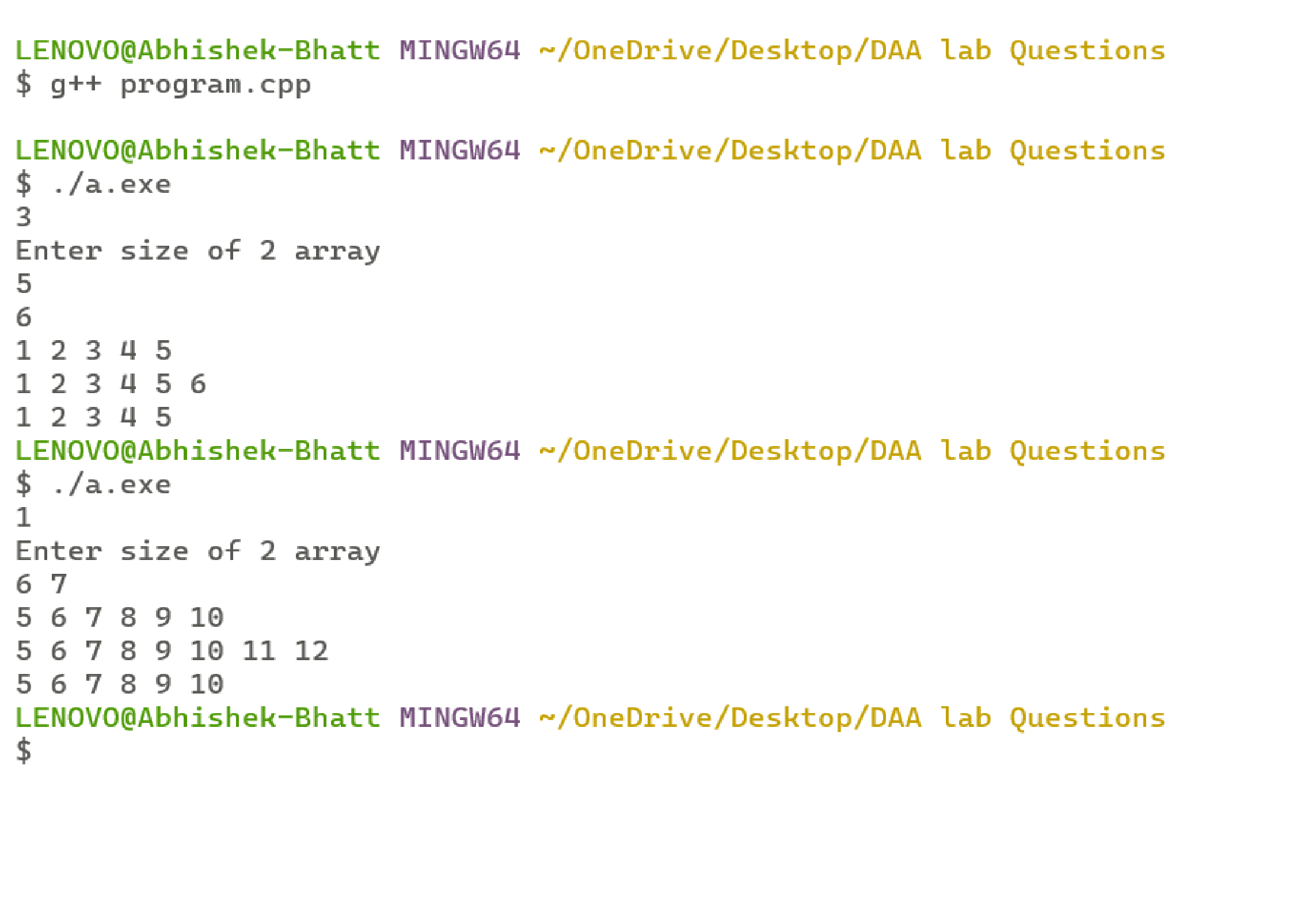
} } void inputArray(int arr[], int x)

{ for(int i = 0; i < x ; i++) cin >> arr[i];

} int main ()

{ int T; cin >> T; while(T--){ cout << "Enter size of 2 array" << endl; int m, n; cin >> m >> n; int arr1[m], arr2[n]; inputArray(arr1, m); inputArray(arr2, n); printIntersection (arr1, arr2, m, n); return 0; } return 0;}

# Output



# Program 16

**Problem Statement:** Given a (directed/undirected) graph, design an algorithm and implement it using a program to find if a path exists between two given vertices or not. (Hint: use DFS)

**Algorithm:**

DFS(G, u)

u.visited = true for each v ∈ G.Adj[u]

if v.visited == false

DFS(G,v)

init() {

For each u ∈ G

u.visited = false

For each u ∈ G

DFS(G, u)

}

**Complexity Analysis: Time Complexity:**

O(V + E), where V is the number of nodes and E is the number of edges

**Space Complexity:**

Worst Case = Best Case = Average case = 𝑂(𝑉)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

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

void printIntersection(int arr1[], int arr2[], int m, int n)

{ int i = 0, j = 0; while(i < m && j < n){

if(arr1[i] < arr2[j]) i++; else if(arr2[j] < arr1[i]) j++; else{

cout << arr2[j] << " "; i++; j++;

}

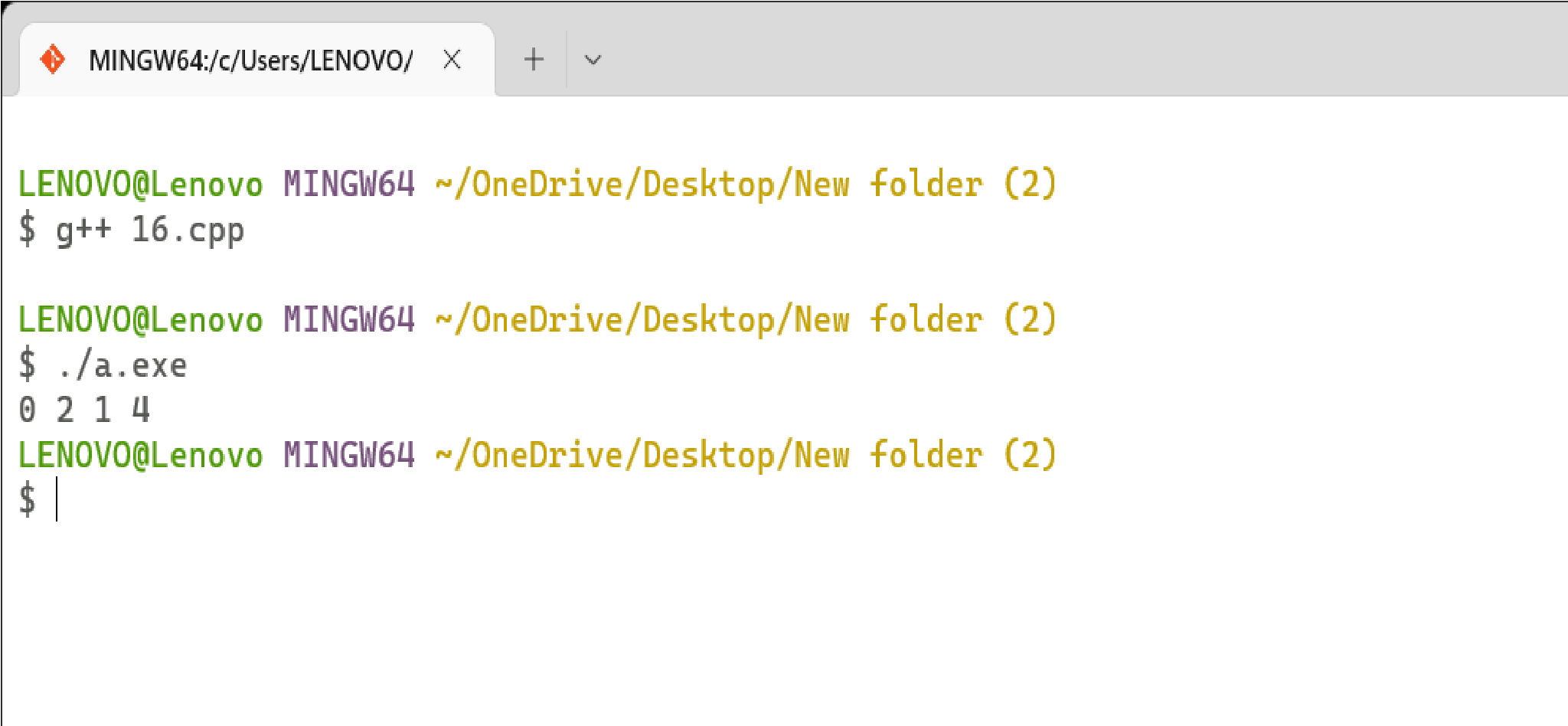
} } void inputArray(int arr[], int x)

{ for(int i = 0; i < x ; i++) cin >> arr[i];

} int main ()

{ int T; cin >> T; while(T--){ cout << "Enter size of 2 array" << endl; int m, n; cin >> m >> n; int arr1[m], arr2[n]; inputArray(arr1, m); inputArray(arr2, n); printIntersection (arr1, arr2, m, n); return 0; } return 0;}

# Output



# Program 17

**Problem Statement:** Given a graph, design an algorithm and implement it using a program to find if a graph is bipartite or not. (Hint: use BFS)

**Algorithm:**

Procedure BFS (G, s)

G is the graph and s is the source node begin

let q be queue to store nodes

q.enqueue(s) //insert source node in the queue mark s as visited.

while (q is not empty)

//remove the element from the queue whose adjacent nodes are to be processed n = q.dequeue( )

//processing all the adjacent nodes of n for all neighbors m of n in Graph G if w is not visited

q.enqueue (m) //Stores m in Q to in turn visit its adjacent nodes mark m as visited.

End

**Complexity Analysis:**

**Time Complexity:**

O(V + E), where V is the number of nodes and E is the number of edges **Space Complexity:**

Worst Case = Best Case = Average case = 𝑂(𝑉)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249 Section: C**

**\*/**

#include <iostream> #include <list> using namespace std;

class Graph {

int numVertices; list<int>\* adjLists; bool\* visited;

public: Graph(int vertices); void addEdge(int src, int dest); void BFS(int startVertex);

};

Graph::Graph(int vertices) { numVertices = vertices; adjLists = new list<int>[vertices];

}

void Graph::addEdge(int src, int dest) {

adjLists[src].push\_back(dest); adjLists[dest].push\_back(src);

}

void Graph::BFS(int startVertex) { visited = new bool[numVertices]; for (int i = 0; i < numVertices; i++) visited[i] = false; list<int> queue;

visited[startVertex] = true; queue.push\_back(startVertex);

list<int>::iterator i; while (!queue.empty()) { int currVertex = queue.front(); cout << "Visited " << currVertex << " "; queue.pop\_front();

for (i = adjLists[currVertex].begin(); i != adjLists[currVertex].end(); ++i) {

int adjVertex = \*i; if (!visited[adjVertex]) {

visited[adjVertex] = true; queue.push\_back(adjVertex);

}

}

}

} int main() { Graph g(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

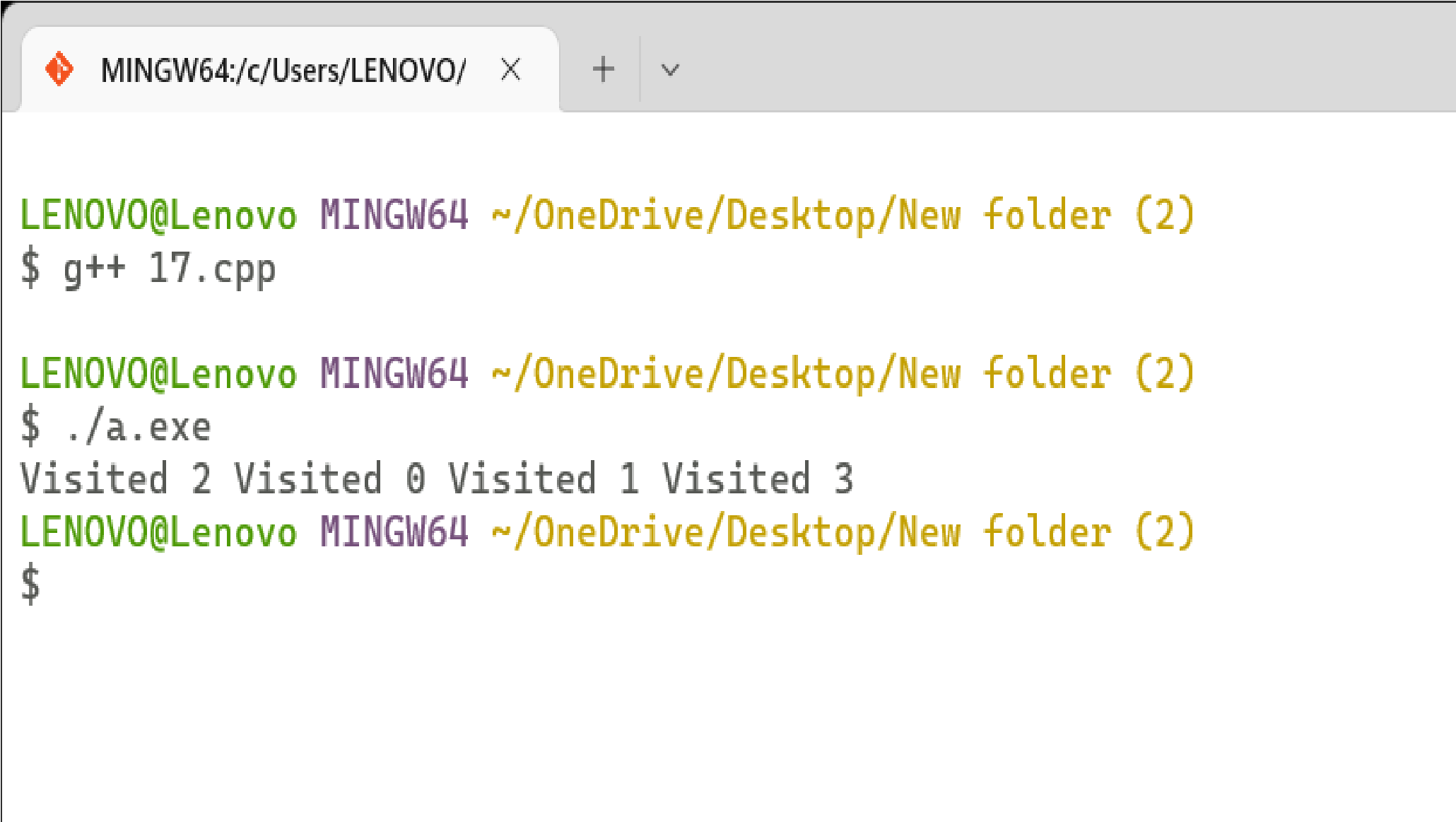
g.addEdge(2, 3);

g.addEdge(3, 3);

g.BFS(2); return 0;

}

# Output



# Program 18

**Problem Statement:** Given a directed graph, design an algorithm and implement it using a program to find whether cycle exists in the graph or not.

**Algorithm:**

DFS-iterative (G, s):

Where G is graph and s is source vertex let S be stack

S.push( s ) //Inserting s in stack mark s as visited.

while ( S is not empty):

//Pop a vertex from stack to visit next v = S.top( )

S.pop( )

//Push all the neighbours of v in stack that are not visited for all neighbours w of v in Graph G:

if w is not visited :

S.push( w )

//mark w as visited

DFS-recursive(G, s): //mark s as visited for all neighbours w of s in Graph G:

if w is not visited:

DFS-recursive(G, w)

**Complexity Analysis:**

**Time Complexity:**

O(V + E), where V is the number of nodes and E is the number of edges **Space Complexity:**

Worst Case = Best Case = Average case = 𝑂(𝑉)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include<iostream>

#include<set> #define NODE 5 using namespace std; int graph[NODE][NODE] =

{{0, 1, 1, 0,0},

{1, 0, 1, 1,1}, {1, 1, 0, 1,0},

{0, 1, 1, 0,1},

{0, 1, 0, 1,0}

};

bool dfs(int curr, set<int>&wSet, set<int>&gSet, set<int>&bSet) {

wSet.erase(wSet.find(curr)); gSet.insert(curr);

for(int v = 0; v < NODE; v++) { if(graph[curr][v] != 0) { if(bSet.find(v) != bSet.end()) continue;

if(gSet.find(v) != gSet.end()) return true; if(dfs(v, wSet, gSet, bSet)) return true;

}

}

gSet.erase(gSet.find(curr)); bSet.insert(curr); return false;

}

bool hasCycle() {

set<int> wSet, gSet, bSet; for(int i = 0; i<NODE; i++) wSet.insert(i);

while(wSet.size() > 0) { for(int current = 0; current < NODE; current++) { if(wSet.find(current) != wSet.end()) if(dfs(current, wSet, gSet, bSet)) return true;

}

} return false;

}

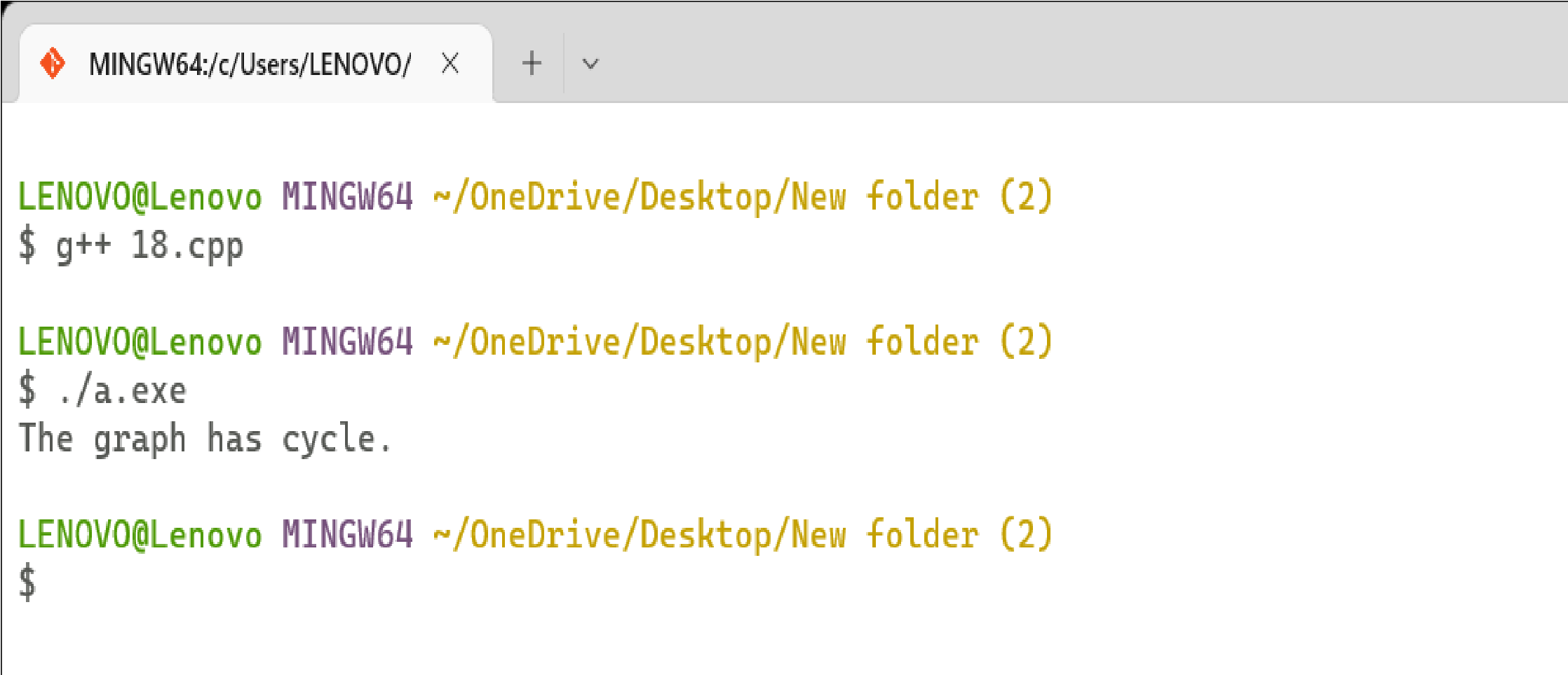
int main() {

bool res; res = hasCycle(); if(res) cout << "The graph has a cycle." << endl;

else cout << "The graph has no cycle." << endl;

}

# Output



# Program 19

**Problem Statement:** After end term examination, Akshay wants to party with his friends. All his friends are living as paying guest and it has been decided to first gather at Akshay’s house and then move towards party location. The problem is that no one knows the exact address of his house in the city. Akshay as a computer science wizard knows how to apply his theory subjects in his real life and came up with an amazing idea to help his friends. He draws a graph by looking in to location of his house and his friends’ location (as a node in the graph) on a map. He wishes to find out shortest distance andpath covering that distance from each of his friend’s location to his house and then whatsapp them this path so that they can reach his house in minimum time. Akshay has developed the program that implements Dijkstra’s algorithm but not sure about correctness of results. Can you also implement the same algorithm and verify the correctness of Akshay’s results? (Hint: Print shortestpath and distance from friends’ location to Akshay’s house)

**Algorithm:**

function dijkstra(G, S) for each vertex V in G distance[V] <- infinite previous[V] <- NULL

If V != S, add V to Priority Queue Q distance[S] <- 0

while Q IS NOT EMPTY U <- Extract MIN from Q for each unvisited neighbour V of U

tempDistance <- distance[U] + edge\_weight(U, V) if tempDistance < distance[V] distance[V] <- tempDistance previous[V] <- U

return distance[], previous[]

}

**Complexity Analysis:**

**Time Complexity:**O(E Log V)where V is the number of nodes and E is the number of edge **Space Complexity:** 𝑂(𝑉)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include<bits/stdc++.h> using namespace std; int main(){ int n,m,source; cin >> n >> m;

vector<pair<int,int> > g[n+1]; // 1-indexed adjacency list for of graph int a,b,wt;

for(int i = 0; i<m ; i++){ cin >> a >> b >> wt; g[a].push\_back(make\_pair(b,wt)); g[b].push\_back(make\_pair(a,wt));

}

cin >> source;

priority\_queue<pair<int,int>,vector<pair<int,int> >,greater<pair<int,int> > > pq; In pair => (dist,from) vector<int> distTo(n+1,INT\_MAX); distTo[source] = 0;

pq.push(make\_pair(0,source)); // (dist,from) while( !pq.empty() ){ int dist = pq.top().first; int prev = pq.top().second; pq.pop();

vector<pair<int,int> >::iterator it;

for( it = g[prev].begin() ; it != g[prev].end() ; it++){ int next = it->first;

int nextDist = it->second;

if( distTo[next] > distTo[prev] + nextDist){ distTo[next] = distTo[prev] + nextDist; pq.push(make\_pair(distTo[next], next));

} }

}

cout << "The distances from source, " << source << ", are : \n"; for(int i = 1 ; i<=n ; i++) cout << distTo[i] << " "; cout << "\n";

return 0;}

72

**Output**

# Program 20

**Problem Statement:** Design an algorithm and implement it using a program to solve previoquestion's problem using Bellman- Ford's shortest path algorithm.

**Algorithm:**

function bellmanFord(G, S) for each vertex V in G distance[V] <- infinite previous[V] <- NULL

distance[S] <- 0

for each vertex V in G for each edge (U,V) in G

tempDistance <- distance[U] + edge\_weight(U, V) if tempDistance < distance[V] distance[V] <- tempDistance previous[V] <- U

for each edge (U,V) in G

If distance[U] + edge\_weight(U, V) < distance[V}

Error: Negative Cycle Exists return distance[], previous[]

**Complexity Analysis:**

**Time Complexity**

Best Case Complexity: O(E)

Average Case Complexity: O(VE)

Worst Case Complexity: O(VE)

**Space Complexity: O(V).**

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include <bits/stdc++.h> struct Edge {

int u; //start vertex of the edge int v; //end vertex of the edge int w; //w of the edge (u,v)

};

struct Graph {

int V; // Total number of vertices in the graph int E; // Total number of edges in the graph struct Edge\* edge; // Array of edges

};

struct Graph\* createGraph(int V, int E) {

struct Graph\* graph = new Graph; graph->V = V; // Total Vertices graph->E = E; // Total edges graph->edge = new Edge[E]; return graph;

}

void printArr(int arr[], int size) { int i; for (i = 0; i < size; i++) printf("%d ", arr[i]);

printf("\n");

}

void BellmanFord(struct Graph\* graph, int u) {

int V = graph->V; int E = graph->E; int dist[V]; for (int i = 0; i < V; i++) dist[i] = INT\_MAX;

dist[u] = 0; for (int i = 1; i <= V - 1; i++) { for (int j = 0; j < E; j++) { int u = graph->edge[j].u; int v = graph->edge[j].v; int w = graph->edge[j].w;

if (dist[u] != INT\_MAX && dist[u] + w < dist[v]) dist[v] = dist[u] + w;

} }

for (int i = 0; i < E; i++) { int u = graph->edge[i].u; int v = graph->edge[i].v; int w = graph->edge[i].w;

if (dist[u] != INT\_MAX && dist[u] + w < dist[v]) { printf("Graph contains negative w cycle"); return;

} } printArr(dist, V); return; }

int main() {

int V = 5; // Total vertices int E = 8; // Total edges struct Graph\* graph = createGraph(V, E); graph->edge[0].u = 0; graph->edge[0].v = 1; graph->edge[0].w = 5;

//edge 0 --> 2 graph->edge[1].u = 0; graph->edge[1].v = 2; graph->edge[1].w = 4;

//edge 1 --> 3 graph->edge[2].u = 1; graph->edge[2].v = 3; graph->edge[2].w = 3;

//edge 2 --> 1 graph->edge[3].u = 2; graph->edge[3].v = 1; graph->edge[3].w = 6;

//edge 3 --> 2 graph->edge[4].u = 3; graph->edge[4].v = 2; graph->edge[4].w = 2;

BellmanFord(graph, 0); //0 is the source vertex return 0;

}

77

**Output**

# Program 21

**Problem Statement:** Given a directed graph with two vertices ( source and destination). Design an algorithm and implement it using a program to find the weight of the shortest path from source to destination with exactly k edges on the path.

**Algorithm:**

n = no of vertices

A = matrix of dimension n\*n for k = 1 to n for i = 1 to n for j = 1 to n

Ak[i, j] = min (Ak-1[i, j], Ak-1[i, k] + Ak-1[k, j]) return A

**Complexity Analysis:**

**Time Complexity** O(n3)

**Space Complexity:** O(n2).

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include <iostream> using namespace std; #define nV 4

#define INF 999

void printMatrix(int matrix[][nV]); void floydWarshall(int graph[][nV]) {

int matrix[nV][nV], i, j, k; for (i = 0; i < nV; i++) for (j = 0; j < nV; j++) matrix[i][j] = graph[i][j];

for (k = 0; k < nV; k++) { for (i = 0; i < nV; i++) { for (j = 0; j < nV; j++) { if (matrix[i][k] + matrix[k][j] < matrix[i][j]) matrix[i][j] = matrix[i][k] + matrix[k][j];

}

}

} printMatrix(matrix);

}

void printMatrix(int matrix[][nV]) { for (int i = 0; i < nV; i++) { for (int j = 0; j < nV; j++) {

if (matrix[i][j] == INF) printf("%4s", "INF"); else printf("%4d", matrix[i][j]);

} printf("\n");

}

} int main() {

int graph[nV][nV] = {{0, 3, INF, 5},

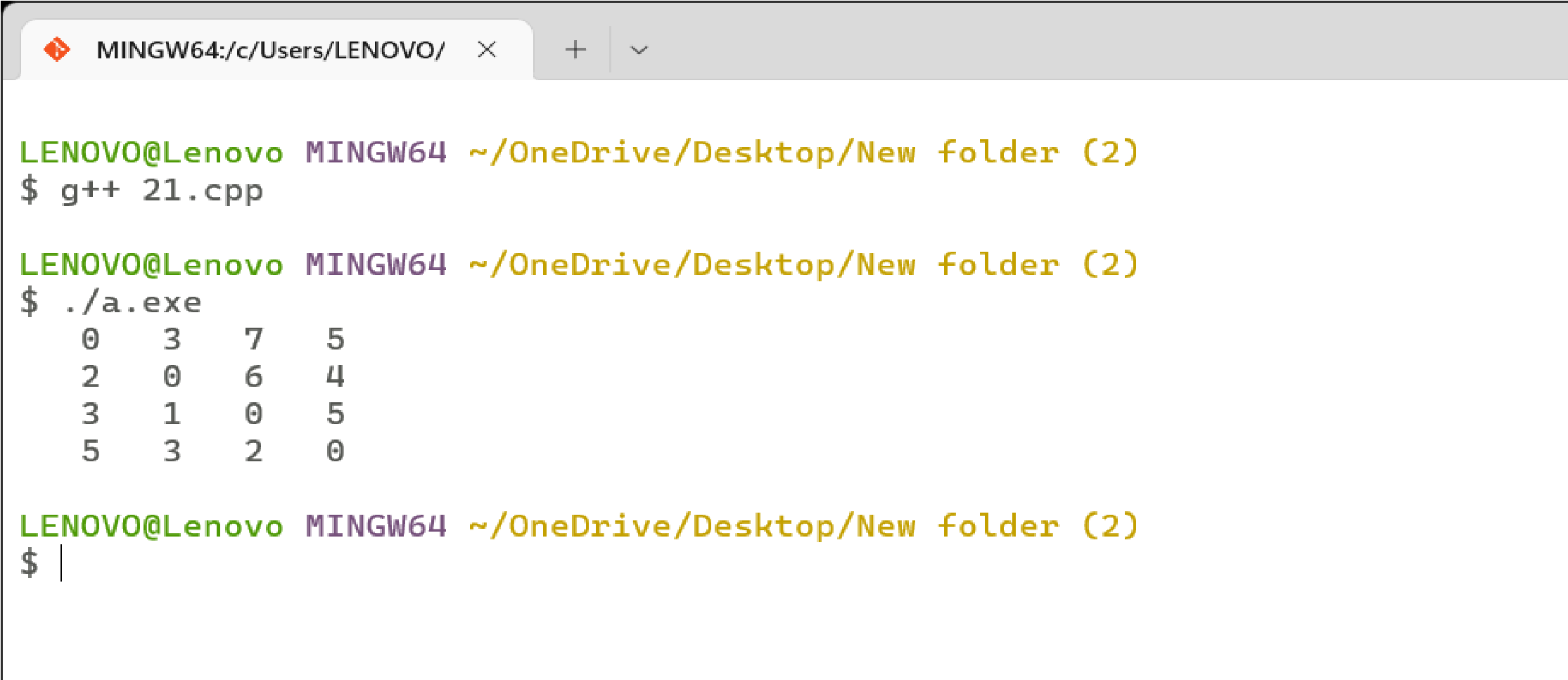
{2, 0, INF, 4},

{INF, 1, 0, INF},

{INF, INF, 2, 0}}; floydWarshall(graph);

}

# Output



# Program 22

**Problem Statement:** Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm which will find the minimum cost required to connect these cities. (use Prim's algorithm)

**Algorithm:**

Prim(G, r) for each u in V[G] key[u] = infinity π[u] = NIL key[r] = 0

Q =V(G) while Q is not empty u= EXTRACT-MIN(Q)

for each v in Adj[u] if v EQ and w(u, v) < key[v]

DECREASE-KEY(Q, key[v], w(u, v)) π[V] = u

return {(v, π[v]):v € V - {r}}

**Complexity Analysis:**

**Time Complexity** O(E Log V)

**Space Complexity:**O(V+E)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include <cstring> #include <iostream> using namespace std;

#define INF 9999999 #define V 5

int G[V][V] = {

{0, 9, 75, 0, 0},

{9, 0, 95, 19, 42},

{75, 95, 0, 51, 66},

{0, 19, 51, 0, 31},

{0, 42, 66, 31, 0}}; int main() {

int no\_edge; // number of edge int selected[V];

memset(selected, false, sizeof(selected)); no\_edge = 0; selected[0] = true;

int x; // row number int y; // col number cout << "Edge"<< " “<< "Weight"; cout << endl; while (no\_edge < V - 1) { int min = INF; x = 0;

y = 0;

for (int i = 0; i < V; i++) { if (selected[i]) { for (int j = 0; j < V; j++) { if (!selected[j] && G[i][j]) { if (min > G[i][j]) { min = G[i][j]; x = i; y = j;

} } } }

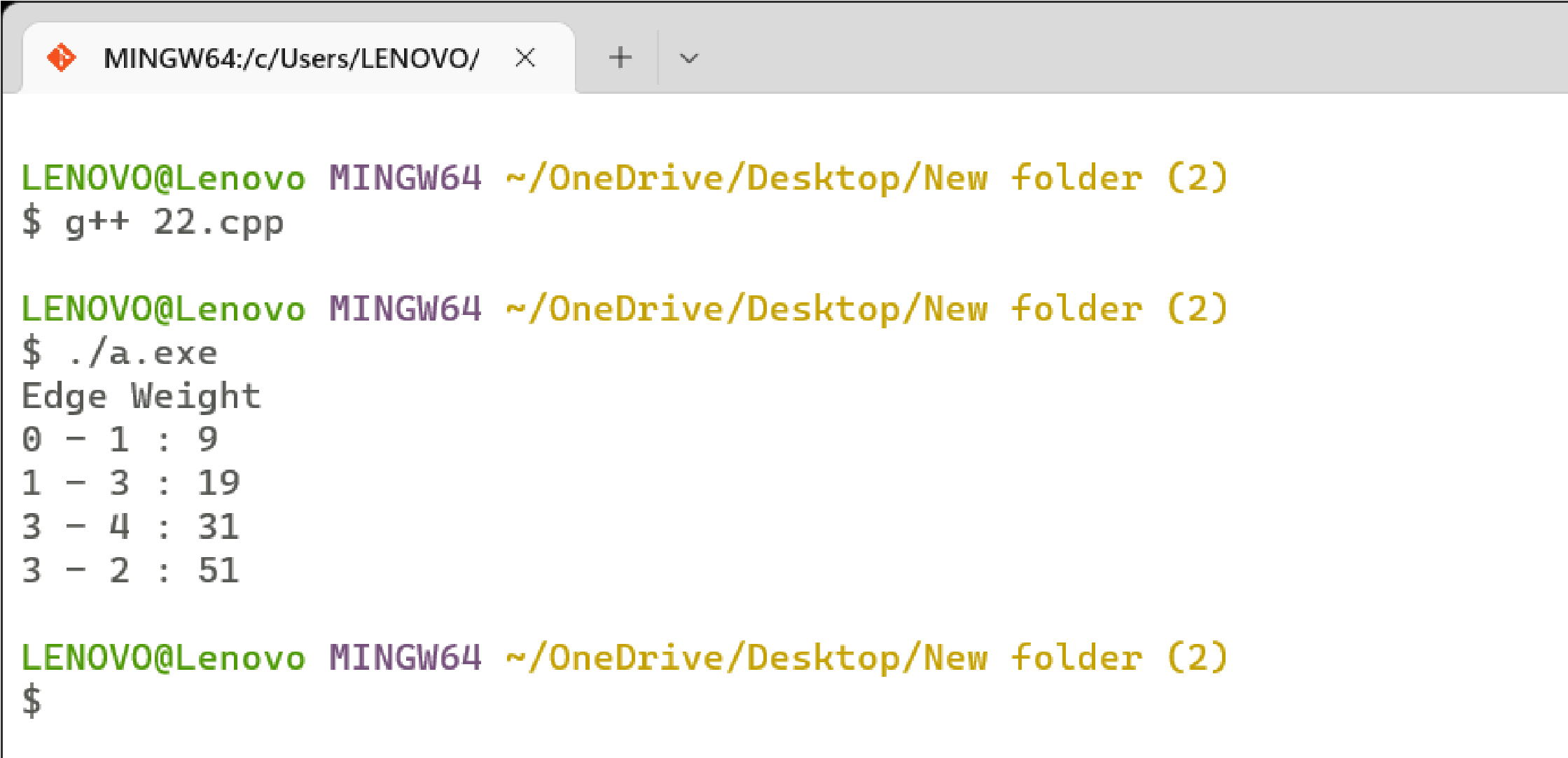
}

cout << x << " - " << y << " : " << G[x][y]; cout << endl; selected[y] = true; no\_edge++;

} return 0;

}

# Output



# Program 23

**Problem Statement:** Implement the previous problem using Kruskal's algorithm.

**Algorithm:**

KRUSKAL(G):

A = ∅

∈

For each vertex v G.V:

MAKE-SET(v)

For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):

if FIND-SET(u) ≠ FIND-SET(v):

A = A ∪ {(u, v)}

UNION(u, v) return A

**Complexity Analysis:**

**Time Complexity** O(E Log E)

**Space Complexity:**O(LogE)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include <algorithm>

#include <iostream> #include <vector> using namespace std; #define edge pair<int, int> class Graph {

private:

vector<pair<int, edge> > G; // graph vector<pair<int, edge> > T; // mst int \*parent;

int V; // number of vertices/nodes in graph

public: Graph(int V);

void AddWeightedEdge(int u, int v, int w); int find\_set(int i); void union\_set(int u, int v); void kruskal(); void print();

};

Graph::Graph(int V) { parent = new int[V]; for (int i = 0; i < V; i++) parent[i] = i;

G.clear();

T.clear();

}

void Graph::AddWeightedEdge(int u, int v, int w) {

G.push\_back(make\_pair(w, edge(u, v)));

}

int Graph::find\_set(int i) {

if (i == parent[i]) return i;

else return find\_set(parent[i]);

}

void Graph::union\_set(int u, int v) {

parent[u] = parent[v];

}

void Graph::kruskal() {

int i, uRep, vRep;

sort(G.begin(), G.end()); // increasing weight for (i = 0; i < G.size(); i++) { uRep = find\_set(G[i].second.first); vRep = find\_set(G[i].second.second); if (uRep != vRep) {

T.push\_back(G[i]); // add to tree union\_set(uRep, vRep);

}

}

}

void Graph::print() {

cout << "Edge :"

<< " Weight" << endl;

for (int i = 0; i < T.size(); i++) { cout << T[i].second.first << " - " << T[i].second.second << " : " << T[i].first;

cout << endl;

}

} int main() { Graph g(6);

g.AddWeightedEdge(0, 1, 4);

g.AddWeightedEdge(0, 2, 4);

g.AddWeightedEdge(1, 2, 2);

g.AddWeightedEdge(1, 0, 4);

g.AddWeightedEdge(2, 0, 4);

g.AddWeightedEdge(2, 1, 2);

g.AddWeightedEdge(2, 3, 3);

g.AddWeightedEdge(2, 5, 2);

g.AddWeightedEdge(2, 4, 4);

g.AddWeightedEdge(3, 2, 3);

g.AddWeightedEdge(3, 4, 3);

g.AddWeightedEdge(4, 2, 4);

g.AddWeightedEdge(4, 3, 3);

g.AddWeightedEdge(5, 2, 2);

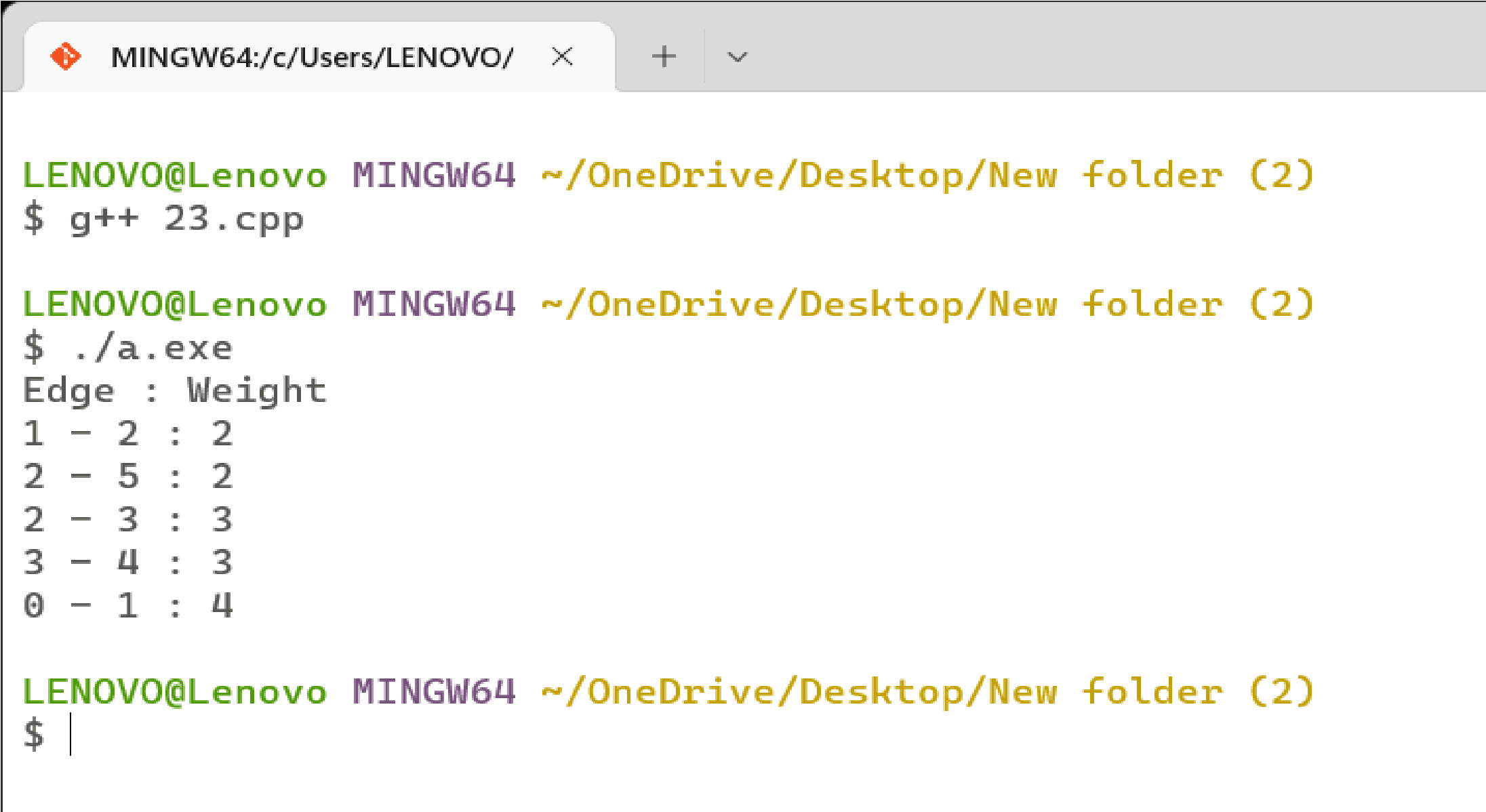
g.AddWeightedEdge(5, 4, 3);

g.kruskal();

g.print(); return 0;

}

# Output



# Program 24

**Problem Statement:** Assume that same road construction project is given to another person.

The amount he will earn from this project is directly proportional to the budget of the project. This person is greedy, so he decided to maximize the budget by constructing those roads who have highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project.

**Algorithm:**

Step1: Start with a weighted graph

Step2: Choose a starting vertex and assign infinity path values to all other devices

Step3: Go to each vertex and update its path length

Step4: If the path length of the adjacent vertex is lesser than new path length, don't update it

Step5: Avoid updating path lengths of already visited vertices

Step6: After each iteration, we pick the unvisited vertex with the least path length. So we choose 5 before 7

Step7: Notice how the rightmost vertex has its path length updated twice

Step8: Repeat until all the vertices have been visited

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include<bits/stdc++.h> using namespace std; int main (){

int n = 5, m = 6, source = 1;

vector < pair < int, int >>g[n + 1]; // assuming 1 based indexing of graph

// Constructing the graph g[1].push\_back ( {2, 2}); g[1].push\_back ({ 4, 1}); g[2].push\_back ( {1, 2}); g[2].push\_back ( {5,5}); g[2].push\_back ( {3,4}); g[2].push\_back ( { 2,4}); g[2].push\_back ( {4,3}); g[3].push\_back ({5, 1}); g[4].push\_back ({1, 1}); g[4].push\_back ({3, 3}); g[5].push\_back ({2, 5}); g[5].push\_back ({3, 1});

priority\_queue < pair < int, int >, vector < pair < int, int >>,greater < pair < int, int >>>pq; vector < int >distTo (n + 1, INT\_MAX); distTo[source] = 0;

pq.push (make\_pair (0, source)); // (dist,source)

while (!pq.empty ()){

int dist = pq.top ().first; int prev = pq.top ().second; pq.pop ();

vector < pair < int, int >>::iterator it; for (it = g[prev].begin (); it != g[prev].end (); it++)

{ int next = it->first; int nextDist = it->second; if (distTo[next] > distTo[prev] + nextDist){ distTo[next] = distTo[prev] + nextDist;

pq.push (make\_pair (distTo[next], next));

}

}

}

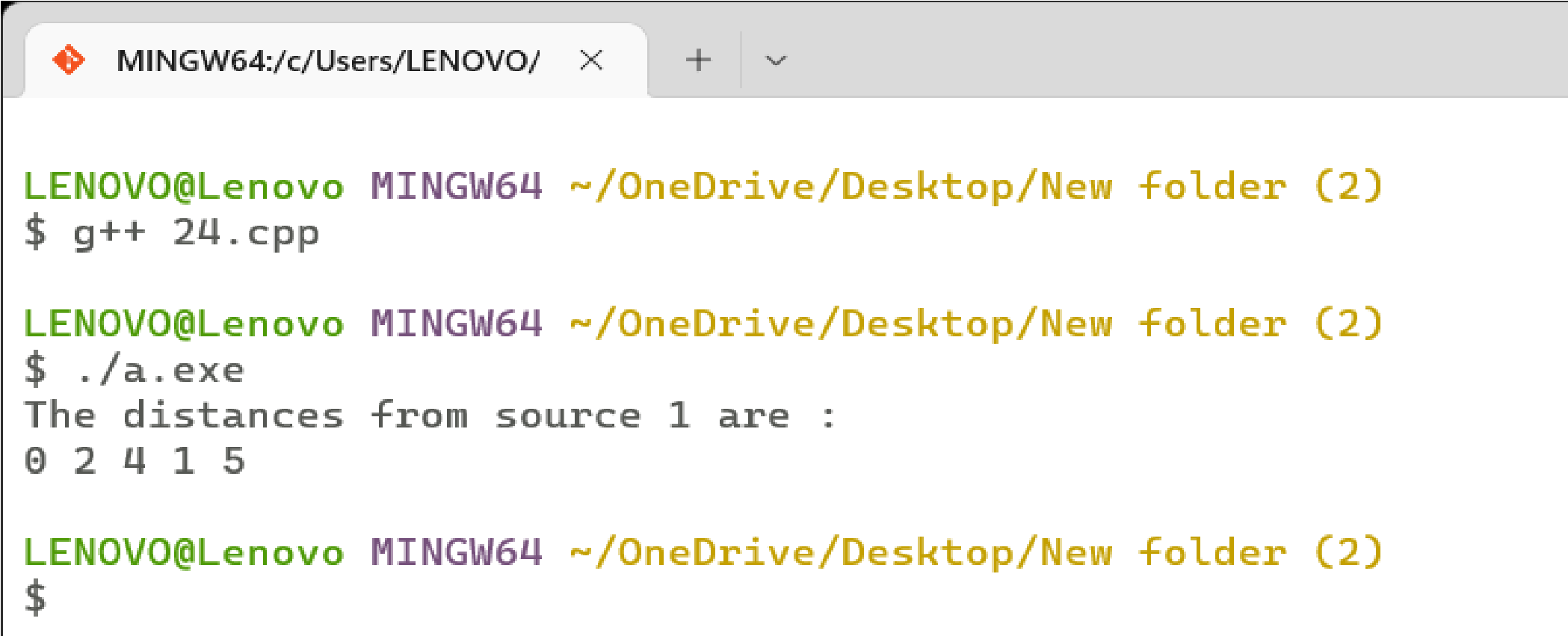
cout << "The distances from source " << source << " are : \n"; for (int i = 1; i <= n; i++)

cout << distTo[i] << " ";

cout << "\n"; return 0;

}

# Output



# Program 25

**Problem Statement:** Given a graph, Design an algorithm and implement it using a program to implement Floyd-Warshall all pair shortest path algorithm

**Algorithm:**

n = no of vertices

A = matrix of dimension n\*n for k = 1 to n

for i = 1 to n for j = 1 to n

Ak[i, j] = min (Ak-1[i, j], Ak-1[i, k] + Ak-1[k, j]) return A

**Complexity Analysis:**

**Time Complexity** O(n3)

**Space Complexity:**O(n2)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include <iostream> using namespace std; #define nV 4

#define INF 999 void printMatrix(int matrix[][nV]); void floydWarshall(int graph[][nV]) {

int matrix[nV][nV], i, j, k; for (i = 0; i < nV; i++) for (j = 0; j < nV; j++) matrix[i][j] = graph[i][j];

for (k = 0; k < nV; k++) { for (i = 0; i < nV; i++) { for (j = 0; j < nV; j++) { if (matrix[i][k] + matrix[k][j] < matrix[i][j]) matrix[i][j] = matrix[i][k] + matrix[k][j];

}}

} printMatrix(matrix);

} void printMatrix(int matrix[][nV]) { for (int i = 0; i < nV; i++) { for (int j = 0; j < nV; j++) { if (matrix[i][j] == INF) printf("%4s", "INF"); else printf("%4d", matrix[i][j]);

} printf("\n");}

} int main() {

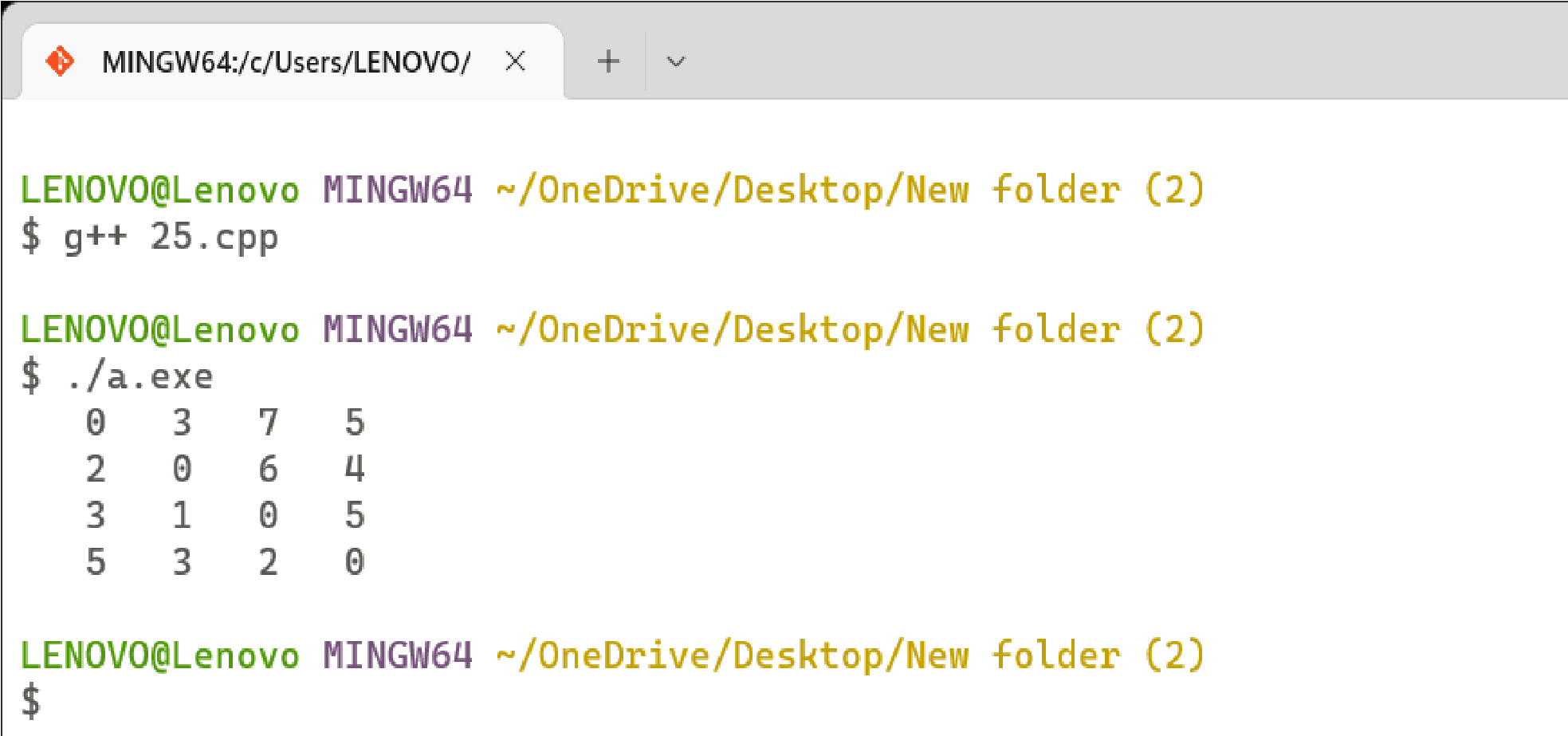
int graph[nV][nV] = {{0, 3, INF, 5},

{2, 0, INF, 4},

{INF, 1, 0, INF},

{INF, INF, 2, 0}}; floydWarshall(graph);}

# Output



# Program 26

**Problem Statement:** Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. You have to Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight w and has maximum value. You can take fractions of items,i.e. the items can be broken into smaller pieces so that you have to carry only a fraction xi of item i, where 0 ≤xi≤ 1.

**Algorithm:**

FRACTIONAL\_KNAPSACK(X, V, W, M)

X: An array of n items

V: An array of profit associated with each item

W: An array of weight associated with each item

M: Capacity of knapsack

SW: Weight of selected items SP: Profit of selected items

S ← Φ // Set of selected items, initially empty

SW ← 0 // weight of selected items SP ← 0 // profit of selected items i ← 1 while i ≤ n do

if (SW + w[i]) ≤ M then

S ← S X[i]

SW ← SW + W[i]

SP ← SP + V[i] else frac ← (M - SW) / W[i]

S ← S X[i] \* frac // Add fraction of item X[i]

|  |  |
| --- | --- |
| SP ← SP + V[i] \* frac | // Add fraction of profit |
| SW ← SW + W[i] \* frac | // Add fraction of weight |

end

i ← i + 1 end

**Complexity Analysis:**

**Time Complexity** O(N\*W)

**Space Complexity:**( O(N\*W))

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include <bits/stdc++.h> using namespace std; struct Item { int value; int weight;

};

class Solution {

public:

bool static comp(Item a, Item b) { double r1 = (double) a.value / (double) a.weight; double r2 = (double) b.value / (double) b.weight; return r1 > r2;

} double fractionalKnapsack(int W, Item arr[], int n) { sort(arr, arr + n, comp); int curWeight = 0; double finalvalue = 0.0; for (int i = 0; i < n; i++) { if (curWeight + arr[i].weight <= W) { curWeight += arr[i].weight; finalvalue += arr[i].value; cout<<i<< " "<<arr[i].weight<<endl;

} else { int remain = W - curWeight; finalvalue += (arr[i].value / (double) arr[i].weight) \* (double) remain; break;

}

}

return finalvalue;

}

};

int main() {

int n,weight; cin>>n>>weight;

Item arr[n]; for(int i=0;i<n;i++)

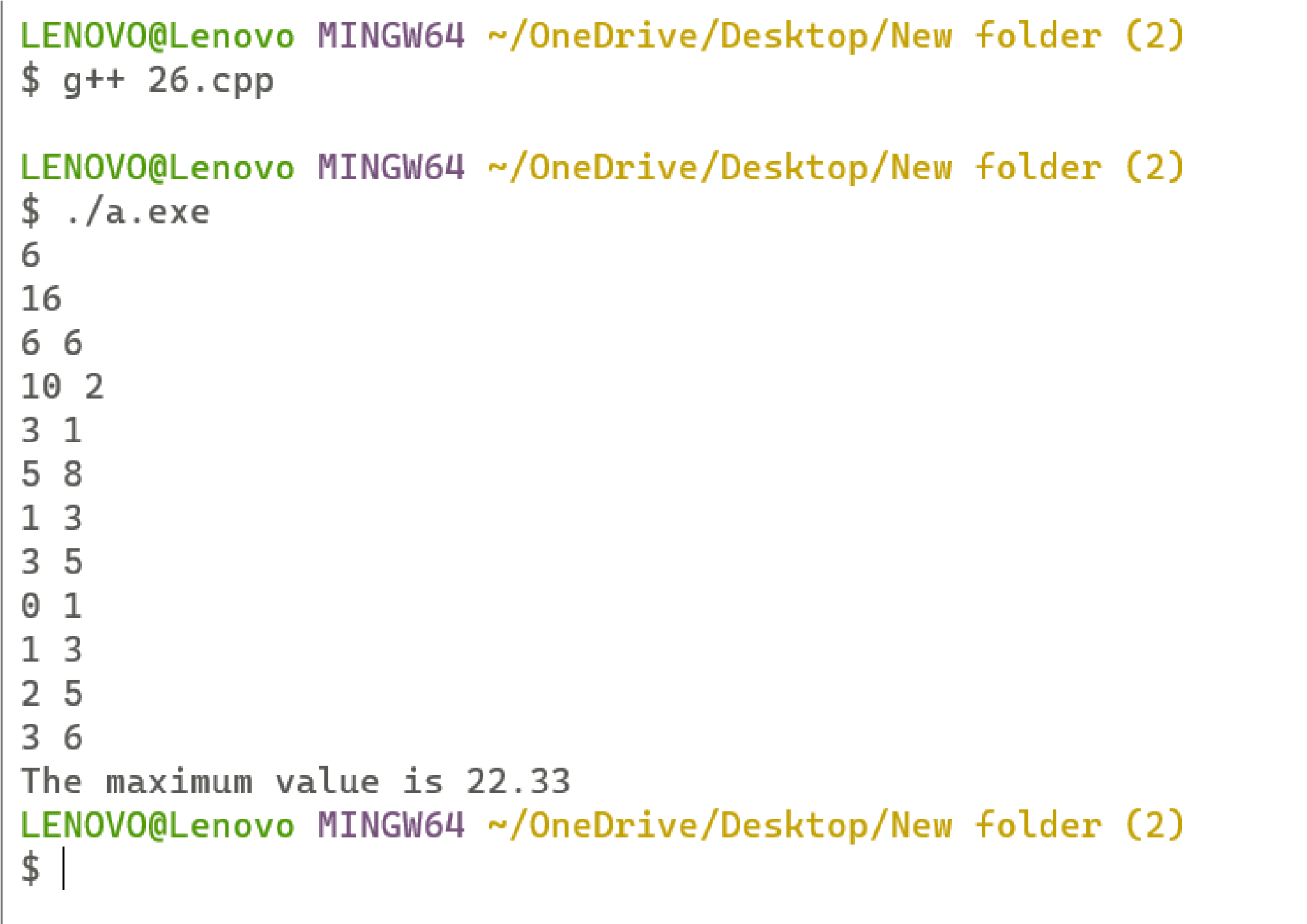
{ cin>>arr[i].weight; cin>>arr[i].value;

}

Solution obj; double ans = obj.fractionalKnapsack(weight, arr, n); cout << "The maximum value is " << setprecision(2) << fixed << ans; return 0;

}

# Output



# Program 27

**Problem Statement:** Given an array of elements. Assume arr[i] represents the size of file i. Write an algorithm and a program to merge all these files into a single file with minimum computation. For given two files A and B with sizes m and n, the computation cost of merging them is O(m+n). (Hint: use greedy approach) **Algorithm:**

Algorithm Tree(n)

//list is a global vector of n single node {

For i=1 to i= n-1 do

{

// get a new tree node

Pt: new treenode;

// merge two trees with smallest length

(Pt = lchild) = least(list);

(Pt = rchild) = least(list);

(Pt =weight) = ((Pt = lchild) = weight) = ((Pt = rchild) = weight);

Insert (list , Pt);

}

// tree left in list

Return least(list);

}

**Complexity Analysis:**

**Time Complexity:**

**Space Complexity:**

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include <bits/stdc++.h> using namespace std; int minComputation(int size, int files[])

{

priority\_queue<int, vector<int>, greater<int> > pq; for (int i = 0; i < size; i++) { pq.push(files[i]);

} int count = 0; while (pq.size() > 1) { int first\_smallest = pq.top(); pq.pop(); int second\_smallest = pq.top(); pq.pop(); int temp = first\_smallest + second\_smallest; count += temp; pq.push(temp);

}

return count;

} int main()

{ int n ;

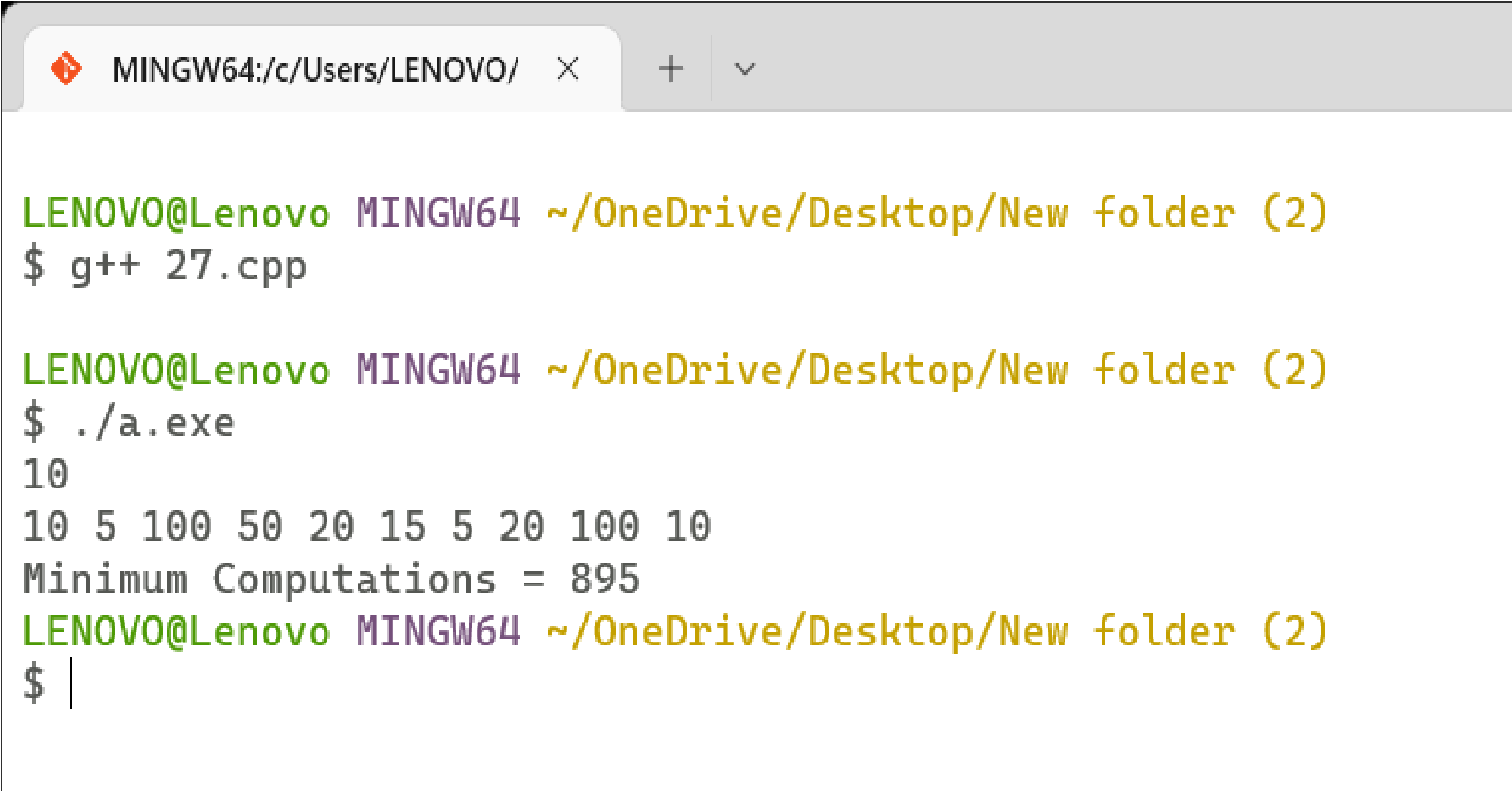
cin>>n;

int files[n]; for(int i=0;i<n;i++){ cin>>files[i];

}

cout << "Minimum Computations = " << minComputation(n, files); return 0;}

# Output



# Program 28

**Problem Statement:**Given a list of activities with their starting time and finishing time. Your goal is to select the maximum number of activities that can be performed by a single person such that selected activities must be non-conflicting. Any activity is said to be non-conflicting if the starting time of an activity is greater than or equal to the finishing time of the other activity.

Assume that a person can only work on a single activity at a time.

**Algorithm:**

GREEDY- ACTIVITY SELECTOR (s, f) n ← length [s] A ← {1} j ← 1.

for i ← 2 to n do if si ≥ fi

then A ← A  {i}

j ← i return A

**Complexity Analysis:**

**Time Complexity** O(NLogN)

**Space Complexity:**O(1)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

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

struct Activitiy

{ int start, finish;

};

bool activityCompare(Activitiy s1, Activitiy s2)

{ return (s1.finish < s2.finish);

} void printMaxActivities(Activitiy arr[], int n)

{

sort(arr, arr+n, activityCompare); cout << "Following activities are selected n"; int i = 0; cout << "(" << arr[i].start << ", " << arr[i].finish << "), "; for (int j = 1; j < n; j++)

{ if (arr[j].start >= arr[i].finish)

{ cout << "(" << arr[j].start << ", "<< arr[j].finish << "), "; i = j; }

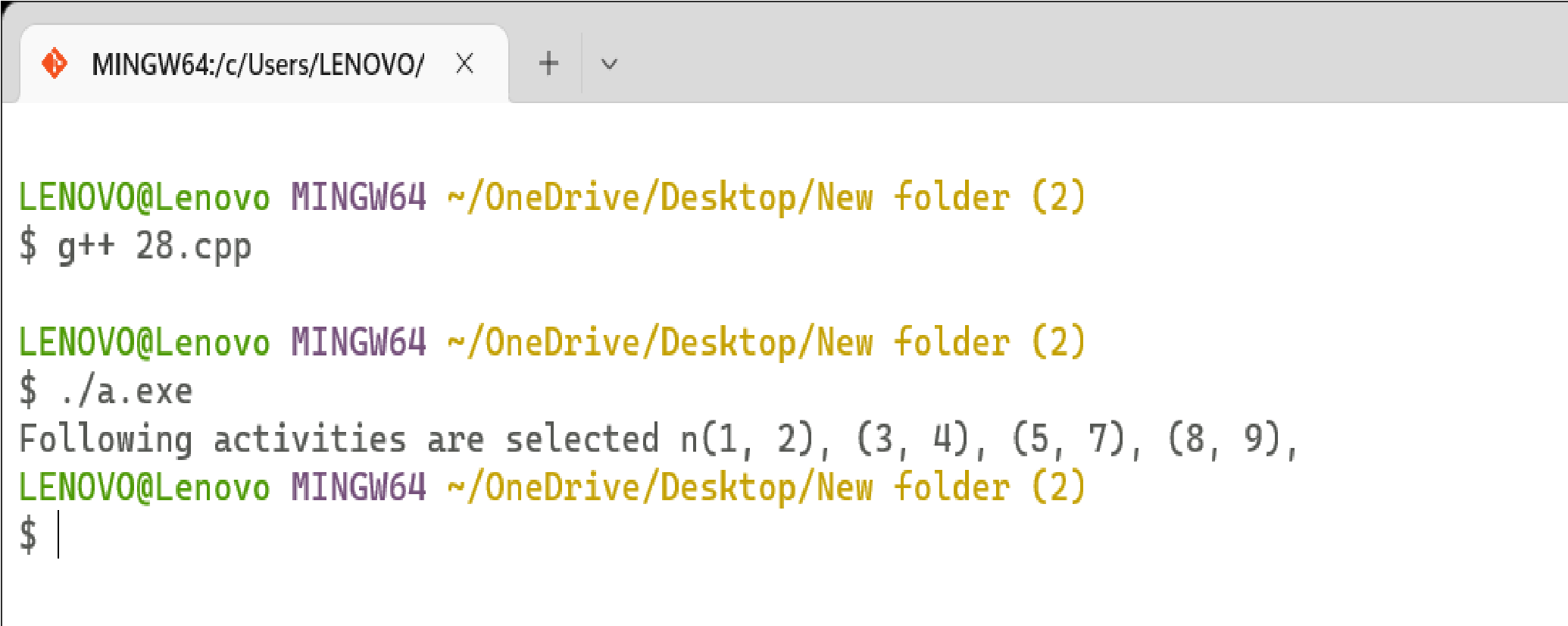
}

} int main()

{

Activitiy arr[] = {{5, 9}, {1, 2}, {3, 4}, {0, 6}, {5, 7}, {8, 9}}; int n = sizeof(arr)/sizeof(arr[0]); printMaxActivities(arr, n); return 0;}

# Output



# Program 29

**Problem Statement:** Given a long list of tasks. Each task takes specific time to accomplish it and each task has a deadline associated with it. You have to design an algorithm and implement it using a program to find maximum number of tasks that can be completed without crossing their deadlines and also find list of selected tasks.

**Algorithm:**

Job-Sequencing\_withdeadlines (D, J, n, k) D(0) := J(0) := 0 k := 1

J(1) := 1 // means first job is selected for i = 2 … n do r := k while D(J(r)) > D(i) and D(J(r)) ≠ r do r := r – 1 if D(J(r)) ≤ D(i) and D(i) > r then for l = k … r + 1 by -1 do

J(l + 1) := J(l) J(r + 1) := i k := k + 1

**Complexity Analysis:**

**Time Complexity** O(N log N) + O(N\*M).

**Space Complexity:**O(M)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

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

int id; // Job Id int dead; // Deadline of job int profit; // Profit if job is over before or on deadline

};

class Solution {

public:

bool static comparison(Job a, Job b) { return (a.profit > b.profit);

} pair < int, int > JobScheduling(Job arr[], int n) { sort(arr, arr + n, comparison); int maxi = arr[0].dead; for (int i = 1; i < n; i++) maxi = max(maxi, arr[i].dead); int slot[maxi + 1]; for (int i = 0; i <= maxi; i++) slot[i] = -1; int countJobs = 0, jobProfit = 0; for (int i = 0; i < n; i++) { for (int j = arr[i].dead; j > 0; j--) { if (slot[j] == -1) { slot[j] = i;

countJobs++;

jobProfit += arr[i].profit;

break;

}

}

}

return make\_pair(countJobs, jobProfit);

}

};

int main() {

int n = 4;

Job arr[n] = {{1,4,20},{2,1,10},{3,2,40},{4,2,30}};

Solution ob; //function call pair < int, int > ans = ob.JobScheduling(arr, n); cout << ans.first << " " << ans.second << endl;

return 0;

}

111

**Output**

# Program 30

**Problem Statement:** Given an unsorted array of elements, design an algorithm and implement it using a program to find whether majority element exists or not. Also find median of the array. A majority element is an element that appears more than n/2 times, where n is the size of array.

**Algorithm:**

Initialize an element m and a counter i = 0 for each element x of the input sequence:

if i = 0, then assign m = x and i = 1 else if m = x, then assign i = i + 1 else assign i = i – 1 return m

**Complexity Analysis:**

**Time Complexity :**

**Space Complexity:**

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

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

void findMajority(int arr[], int n)

{

int maxCount = 0; int index = -1; // sentinels for (int i = 0; i < n; i++) { int count = 0; for (int j = 0; j < n; j++) { if (arr[i] == arr[j]) count++;

}

if (count > maxCount) { maxCount = count; index = i;}

}

if (maxCount > n / 2) cout << arr[index] << endl;

else cout << "No Majority Element" << endl;

} int main(){ int n;

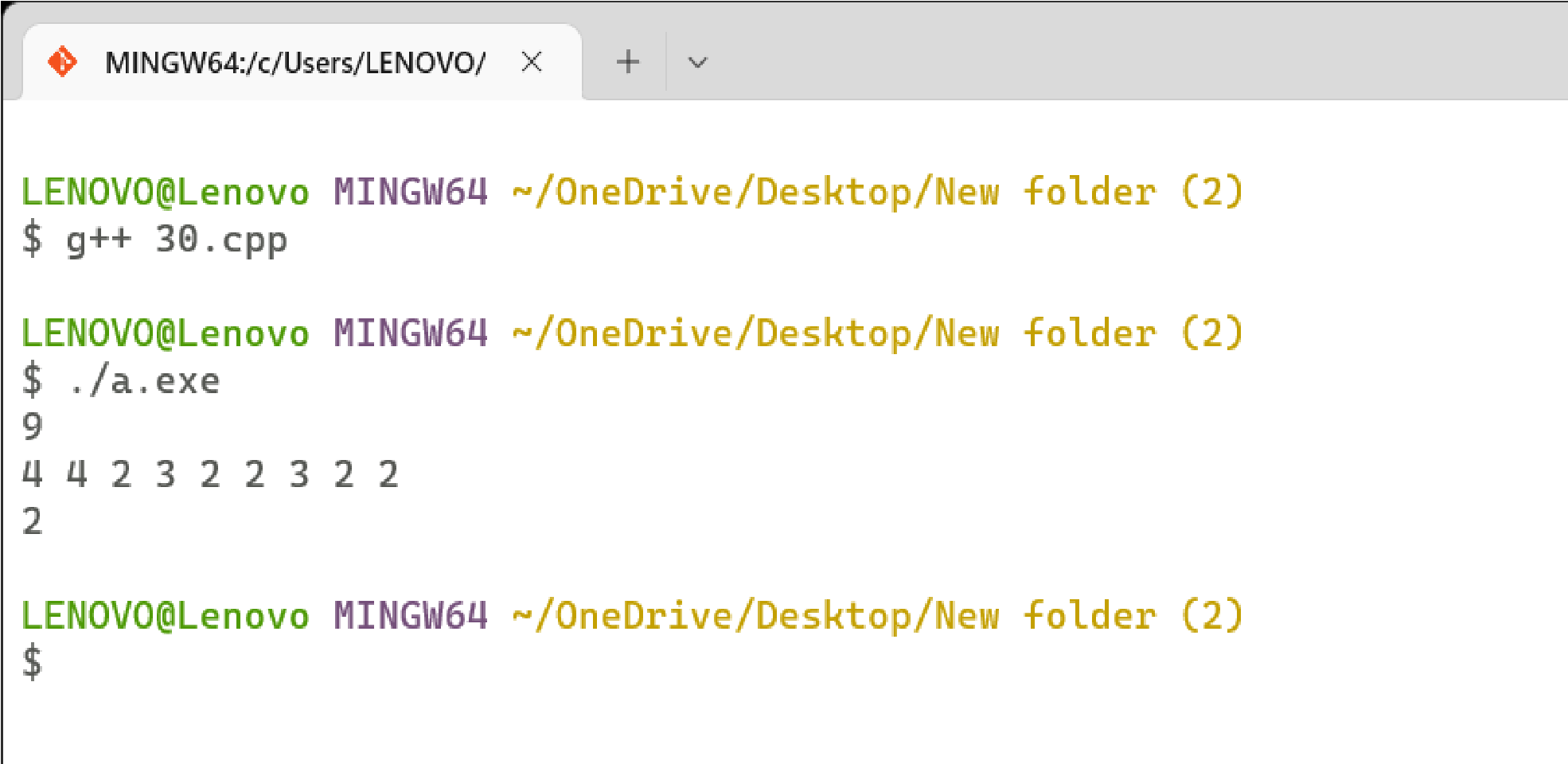
cin>>n;

int arr[n]; for(int i=0;i<n;i++){ cin>>arr[i];} findMajority(arr, n);

return 0

}

# Output



# Program 31

**Problem Statement: :** Given a sequence of matrices, write an algorithm to find most efficient way to multiply these matrices together. To find the optimal solution, you need to find the order in which these matrices should be multiplied.

**Algorithm:**

int B(int p[], int i, int j)

{

if(i == j) return 0 int min = INT\_MAX int count for (int k = i to k < j) { count = B(p, i, k) + B(p, k + 1, j) + p[i - 1] \* p[k] \* p[j]

if (count < min) min = count

}

return min

} int matrixChainMultiplication(int[] p) { return B(p, 1, p.size()-1)

}

**Complexity Analysis:**

**Time Complexity :** O(2^n)

**Space Complexity:** O(n)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

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

int MatrixChainOrder(int p[], int i, int j)

{ if (i == j) return 0; int k;

int min = INT\_MAX; int count; for (k = i; k < j; k++)

{

count = MatrixChainOrder(p, i, k)

+ MatrixChainOrder(p, k + 1, j)

+ p[i - 1] \* p[k] \* p[j];

if (count < min) min = count;

}

return min;

}

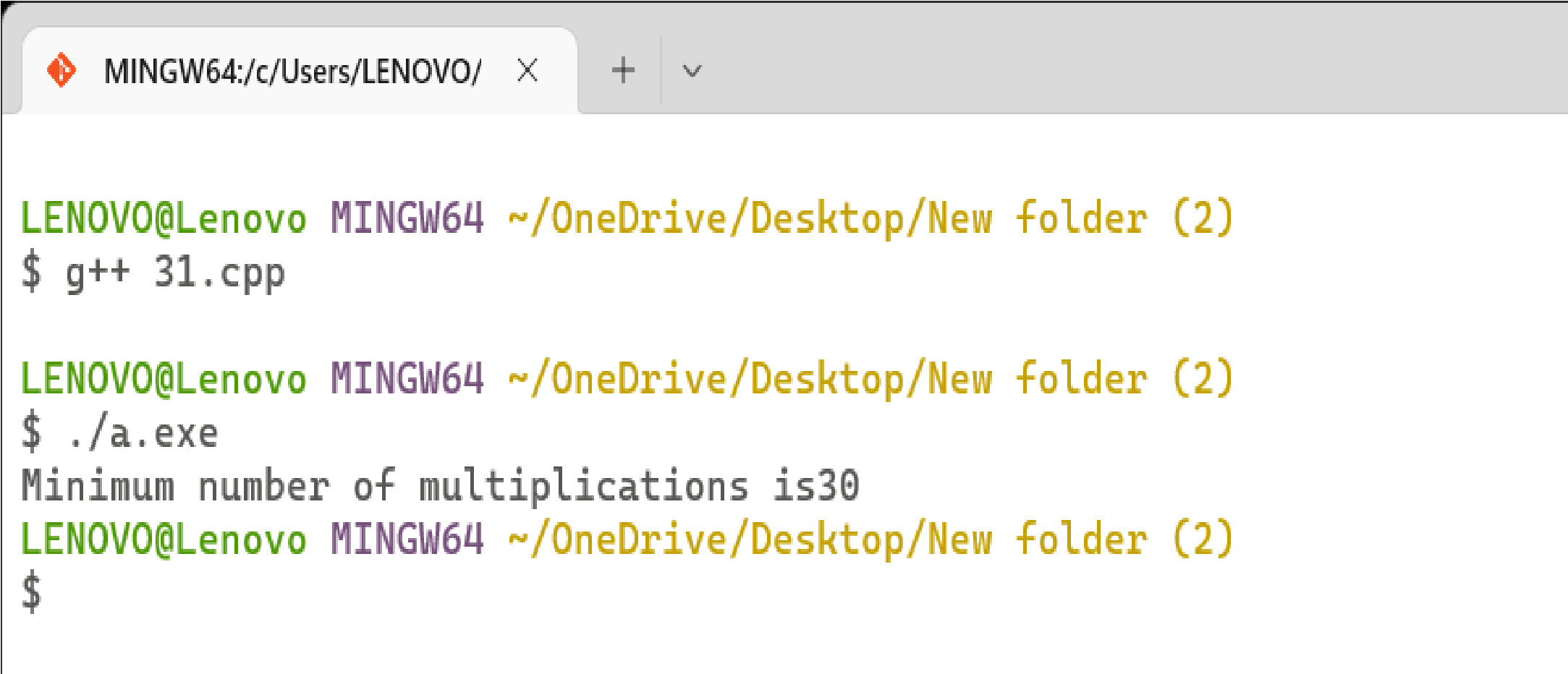
int main()

{ int arr[] = { 1, 2, 3, 4, 3 }; int n = sizeof(arr) / sizeof(arr[0]);

cout<<"Minimum number of multiplications is"<<MatrixChainOrder(arr,1,n-1);

}

# Output



# Program 32

**Problem Statement: :** Given a set of available types of coins. Let suppose you have infinite supply of each type of coin. For a given value N, you have to Design an algorithm and implement it using a program to find number of ways in which these coins can be added to make sum value equals to N. **Algorithm:**

If X[m-1] == Y[n-1] then

L(X[0..m-1], Y[0..n-1]) = 1 + L(X[0..m-2], Y[0..n-2])

Else

L(X[0..m-1], Y[0..n-1]) = MAX ( L(X[0..m-2], Y[0..n-1]), L(X[0..m-1], Y[0..n-2])

// Implementation int L[n+1][m+1];

// Start for(int i=1; i<=n; i++) { for(int j=1; j<=m; i++) { if(i == 0 || j== 0) {

L[i][j] = 0;

} else if (A[i-1] == B[j-1]) {

L[i][j] = L[i-1][j-1] + 1;

} else {

L[i][j] = MAX(L[i-1][j], L[i][j-1]);

}

}

}

**Complexity Analysis:**

**Time Complexity :** O(n\*c)

**Space Complexity:** O(n\*c)(n)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249 Section: C**

**\*/**

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

int count(int S[], int m, int n)

{ if (n == 0) return 1; if (n < 0) return 0; if (m <= 0 && n >= 1) return 0; return count(S, m - 1, n) + count(S, m, n - S[m - 1]);

} int main()

{ int n;

cin>>n;

int arr[n];

for(int i=0;i<n;i++) { cin>>arr[i]; } int k;

cin>>k;

cout << " " << count(arr, n, k); return 0;

}

# Output



# Program 33

**Problem Statement: :** Given a set of elements, you have to partition the set into two subsets such that the sum of elements in both subsets is same. Design an algorithm and implement it using a program to solve this problem

**Algorithm:**

bool canPartition(int[] nums, int n) { int sum = 0 for(int num in nums){ sum = sum + num

}

if( sum%2 != 0) return false sum = sum/2 bool [n+1][sum+1] dp dp[0][0] = true for(int i=1 to i <= n){ for(int j=0 to j <= sum){ if(j-nums[i-1] >= 0){ dp[i][j]=dp[i-1][j-nums[i-1]]

} dp[i][j] = dp[i][j] or dp[i-1][j]

}

}

return dp[n][sum]

}

**Complexity Analysis:**

**Time Complexity :**

**Space Complexity:**

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include <bits/stdc++.h> using namespace std; bool subsetSumUtil(int ind, int target, vector<int>& arr, vector<vector<int>> &dp){ if(target==0) return true; if(ind == 0) return arr[0] == target; if(dp[ind][target]!=-1) return dp[ind][target]; bool notTaken = subsetSumUtil(ind-1,target,arr,dp); bool taken = false; if(arr[ind]<=target) taken = subsetSumUtil(ind-1,target-arr[ind],arr,dp); return dp[ind][target]= notTaken||taken;

}

bool canPartition(int n, vector<int> &arr){ int totSum=0; for(int i=0; i<n;i++){ totSum+= arr[i];

}

if (totSum%2==1) return false; else{ int k = totSum/2; vector<vector<int>> dp(n,vector<int>(k+1,-1)); return subsetSumUtil(n-1,k,arr,dp);

}

}

int main() { int n;

cin>>n; vector<int> arr; for(int i=0;i<n;i++) { int c;

cin>>c; arr.push\_back(c);

} if(canPartition(n,arr))

cout<<"Yes";

else

cout<<" No";

return 0;

}

# Output



# Program 34

**Problem Statement: :** Given two sequences, Design an algorithm and implement it using a program to find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

**Algorithm:**

X and Y be two given sequences

Initialize a table LCS of dimension X.length \* Y.length

X.label = X Y.label = Y

LCS[0][] = 0

LCS[][0] = 0

Start from LCS[1][1]

Compare X[i] and Y[j]

If X[i] = Y[j]

LCS[i][j] = 1 + LCS[i-1, j-1]

Point an arrow to LCS[i][j]

Else

LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])

Point an arrow to max(LCS[i-1][j], LCS[i][j-1])

**Complexity Analysis:**

**Time Complexity :** worst case: O(n\*m)

Average case: O(n\*m)

Best case: O(n\*m)

**Space Complexity:** O(n\*m)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include <bits/stdc++.h> using namespace std; void lcsAlgo(char \*S1, char \*S2, int m, int n) {

int LCS\_table[m + 1][n + 1]; for (int i = 0; i <= m; i++) { for (int j = 0; j <= n; j++) { if (i == 0 || j == 0)

LCS\_table[i][j] = 0; else if (S1[i - 1] == S2[j - 1])

LCS\_table[i][j] = LCS\_table[i - 1][j - 1] + 1; else

LCS\_table[i][j] = max(LCS\_table[i - 1][j], LCS\_table[i][j - 1]); }

}

int index = LCS\_table[m][n]; char lcsAlgo[index + 1]; lcsAlgo[index] = '\0'; int i = m, j = n; while (i > 0 && j > 0) { if (S1[i - 1] == S2[j - 1]) { lcsAlgo[index - 1] = S1[i - 1];

i--; j--;

index--;

} else if (LCS\_table[i - 1][j] > LCS\_table[i][j - 1]) i--;

Else

j--;

}

cout << "S1 : " << S1 << "\nS2 : " << S2 << "\nLCS: " << lcsAlgo<< "\nlength: "<<strlen(lcsAlgo) <"\n";

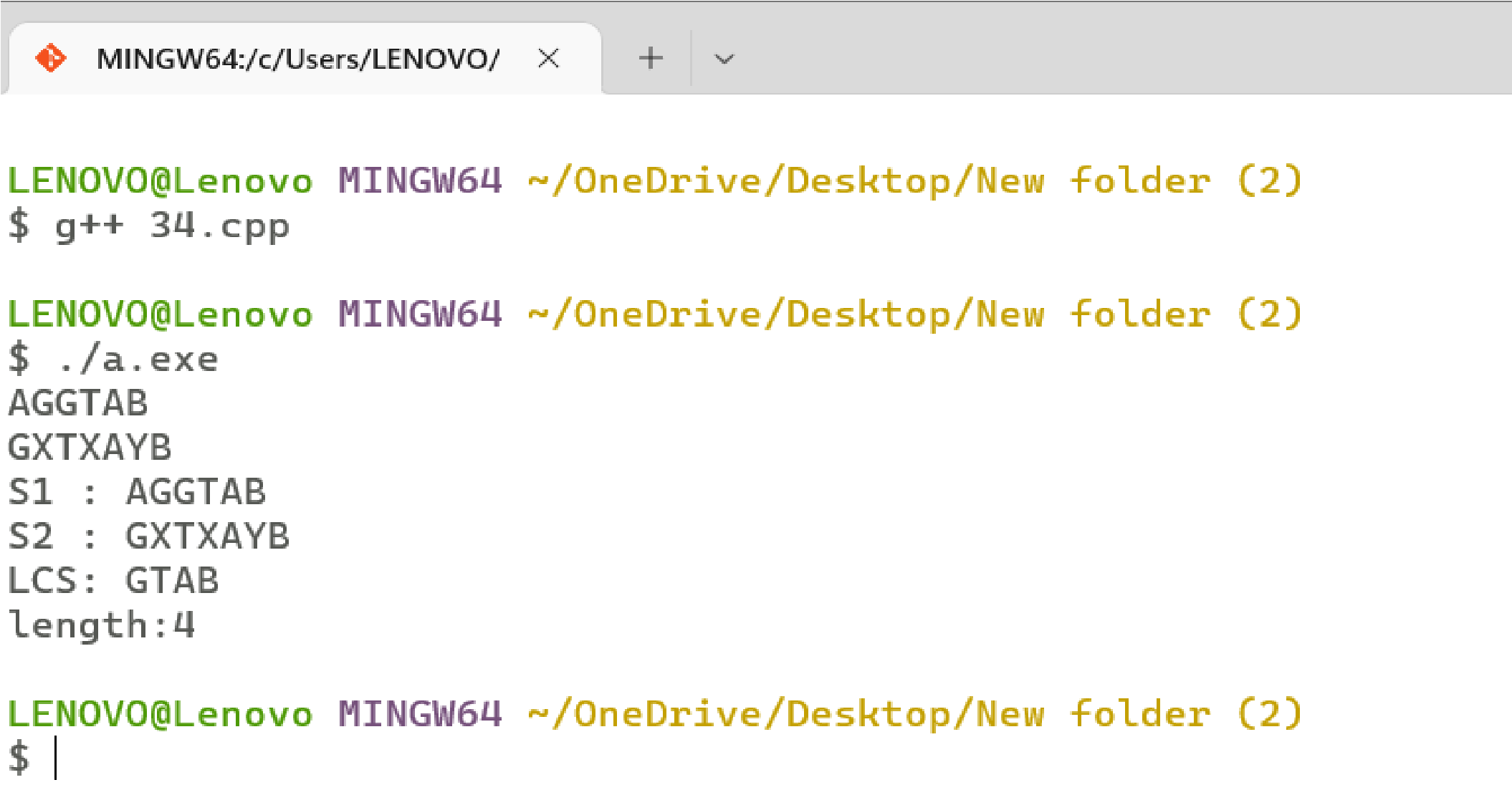
} int main() {

char S1[100] ; char S2[100] ; cin>>S1>>S2;

int m = strlen(S1); int n = strlen(S2); lcsAlgo(S1, S2, m, n);

}

# Output



# Program 35

**Problem Statement: :** Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight <= w and has maximum value. Here, you cannot break an item i.e. either pick the complete item or don't pick it. (0-1 property).

**Algorithm:**

Dynamic-0-1-knapsack (v, w, n, W) for w = 0 to W do c[0, w] = 0 for i = 1 to n do c[i, 0] = 0 for w = 1 to W do if wi ≤ w then if vi + c[i-1, w-wi] then c[i, w] = vi + c[i-1, w-wi] else c[i, w] = c[i-1, w] else

c[i, w] = c[i-1, w]

**Complexity Analysis:**

**Time Complexity :** O(N\*W)

**Space Complexity:** O(N\*W)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

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

{ return (a > b) ? a : b;

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

{ int i, w;

vector<vector<int>> K(n + 1, vector<int>(W + 1)); for(i = 0; i <= n; i++)

{

for(w = 0; w <= W; 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][W];

} int main()

{ int n,w; cin>>n>>w;

int wt[]={2,3,3,4,6}; int v[]={1,2,5,9,4};

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

{

cout<<wt[i]<<" "<<v[i]<<endl;

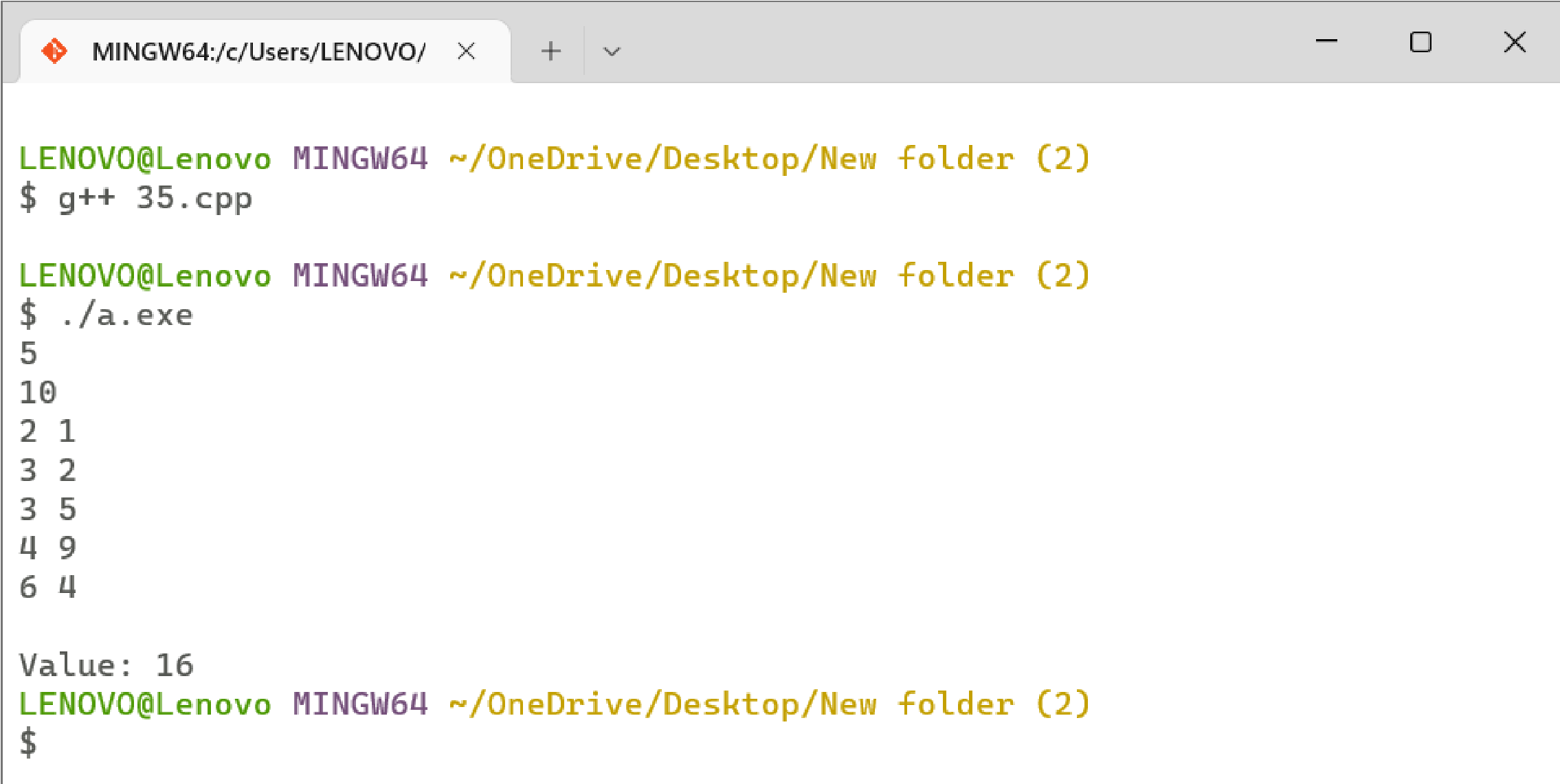
}

cout <<endl<< "Value: "<<knapSack(w, wt, v, n);

return 0;

}

# Output



# Program 36

**Problem Statement:** Given a string of characters, design an algorithm and implement it using a program to print all possible permutations of the string in lexicographic order.

**Algorithm:**

We’ll define a function generatePermutaionsHelper(Str, l, r). This function will generate the permutations of the substring starting from index “l” and ending at index “r”.

* Calling the above function, generatePermutaionsHelper(Str, l, r).
* If “l” is equal to “r”, a new permutation is found. Insert this string in the “ans” list.
* Else, continue to iterate on the string from “l” to“r”.
* Let “i” denote the current index.
* Swap Str[ l ] and Str[ i ] to fix the “ith” character on the index ”l”.
* Call generatePermutaionsHelper(Str, l + 1, r) to get the permutation of the rest of thecharacters.
* Now, backtrack and swap Str[ l ] and Str[ i ] again.

In the end, we’ll have the list “ans” having all the permutations of the given string. If we want the permutations in lexicographically increasing order, we have to sort the list

**Complexity Analysis:**

**Time Complexity :** O((N! \* log(N!))

**Space Complexity:** O(N \* N!)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include <bits/stdc++.h> using namespace std; void generatePermutationsHelper(string &str, int l, int r, vector<string> &ans)

{

// base case

if (l == r){

ans.push\_back(str); return;

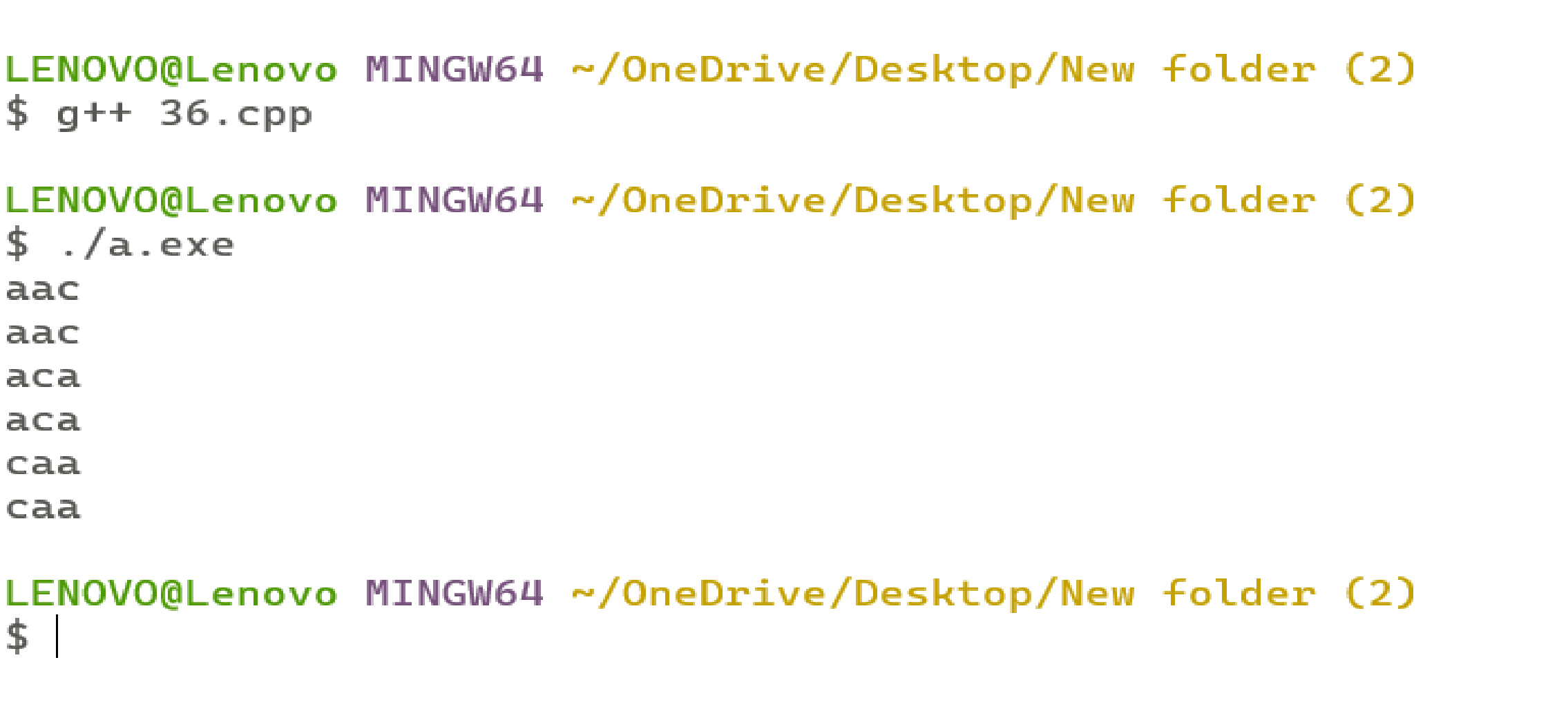
} for (int i = l; i <= r; i++){ swap(str[l], str[i]); generatePermutationsHelper(str, l + 1, r, ans); swap(str[l], str[i]);

}

} int main(){ vector<string> ans; string str = "aac"; int l = 0; int r = str.size() - 1; if(str.length()==0) cout<<"No Permutations Possible!!"; else generatePermutationsHelper(str, l, r, ans); // lexicographically increasing order sort(ans.begin(), ans.end()); for(int i = 0;i<ans.size();i++) cout<<ans[i]<<endl;

return 0;}

# Output



# Program 37

**Problem Statement:** Given an array of characters, you have to find distinct characters from this array. Design an algorithm and implement it using a program to solve this problem using hashing. (Time Complexity = O(n))

**Algorithm:**

prCharWithFreq( s)

{

Unordered Map d; for(char i to s) d[i]++;

for(char i to s)

{ if(d[i] != 0)

{

PRINT i, d[i] d[i] = 0;

} }

}

**Complexity Analysis:**

**Time Complexity :** O(N)

**Space Complexity:** O(N)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249 Section: C**

**\*/**

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

void prCharWithFreq(string s)

{

unordered\_map<char, int> d;

for(char i : s)

{ d[i]++; }

for(char i : s)

{ if(d[i] != 0) { cout << i << d[i] << " "; d[i] = 0;

}

}

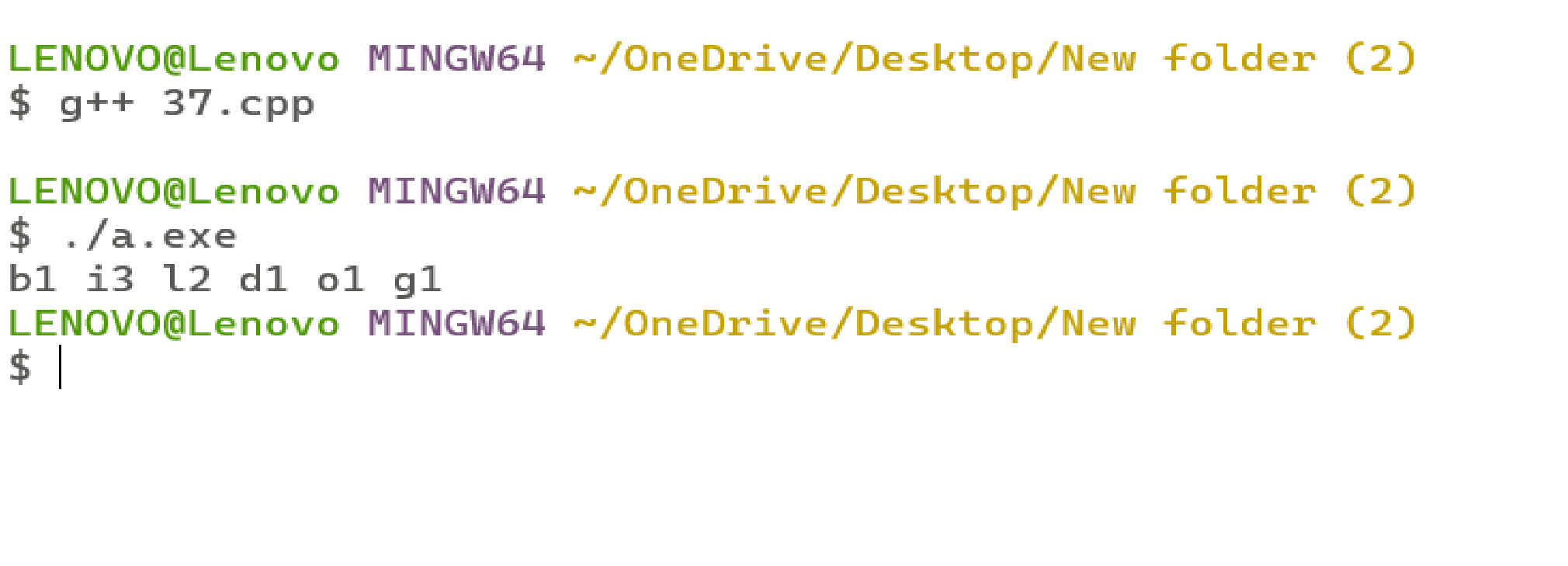
}

int main()

{ string s="billidogi"; prCharWithFreq(s);

}

# Output



# Program 38

**Problem Statement:**Given an array of integers of size n, design an algorithm and write a program to check whether this array contains duplicates within a small window of size k < n.

**Algorithm:**

1. First, create the empty hash set in which we will store the elements of the array.
2. Traverse all elements of the array from left to right.
3. Check if the element is present in hash or not.
4. If it’s in there then return “true.”
5. Else add that element to the hash.
6. After that, remove the arr[i-k] element from the hash if ‘I’ is greater or equal to ‘k’.

**Complexity Analysis:**

**Time Complexity :** O(N)

**Space Complexity:** O(K)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

#include <iostream>

#include <vector> #include <unordered\_set> using namespace std; bool contains(unordered\_set<int> const &set, int x) { return set.find(x) != set.end();

}

bool hasDuplicate(vector<int> &input, int k)

{

unordered\_set<int> window;

for (int i = 0; i < input.size(); i++)

{ if (contains(window, input[i])) return true; window.insert(input[i]); if (i >= k)

window.erase(input[i - k]);

} return false;

}

int main()

{ int t;

cin>>t; while(t--){ vector<int> in; int n;

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

cin>>a; in.push\_back(a);

} int k ;

cin>>k;

if (hasDuplicate(in, k)) cout << "\nDuplicates found"; else cout << "\nNo duplicates were found";

} return 0;

}

# Output



# Program 39

**Problem Statement:**Given an array of nonnegative integers, Design an algorithm and implement it using a program to find two pairs (a,b) and (c,d) such that a\*b = c\*d, where a, b, c and d are distinct elements of the array.

**Algorithm:**

For i=0 to n-1

For j=i+1 to n-1

1. Find prod = arr[i]\*arr[j]
2. If prod is not available in hash then makeH[prod] = make\_pair(i, j) // H is hash table c) If product is also available in hash then print previous and current elements of array

**Complexity Analysis:**

**Time Complexity :** O(n2) **Space Complexity:**

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

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

void findPairs(int arr[], int n)

{

bool found = false; unordered\_map<int, pair < int, int > > H; for (int i=0; i<n; i++)

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

{ int prod = arr[i]\*arr[j]; if (H.find(prod) == H.end())

H[prod] = make\_pair(i,j); else

{ pair<int,int> pp = H[prod]; cout << arr[pp.first] << " " << arr[pp.second]<< " and " << arr[i]<<" "<<arr[j]<<endl; found = true;

}

}

} if (found == false) cout << "No pairs Found" << endl;

}

int main()

{

int n;

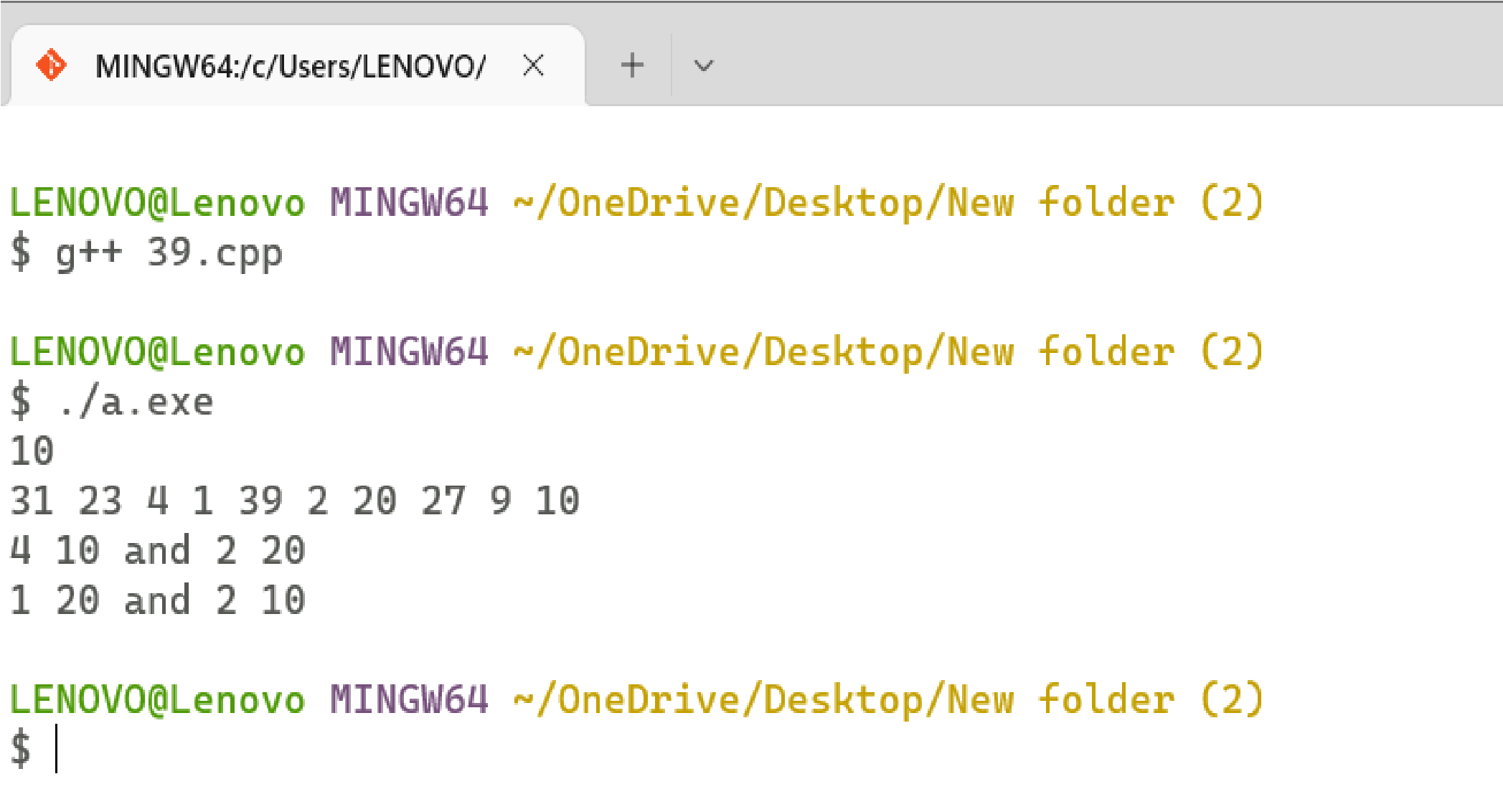
cin>>n;

int arr[n]; for(int i=0;i<n;i++) { cin>>arr[i]; }

findPairs(arr, n); return 0;

}

# Output



# Program 40

**Problem Statement:** Given a number n, write an algorithm and a program to find nth ugly number. Ugly numbers are those numbers whose only prime factors are 2, 3 or 5. The sequence 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24,. is sequence of ugly numbers.

**Algorithm:**

getUglyNumbers(n)

Begin define array named uglyNum of size n i2 := 0, i3 := 0, i5 := 0

next2mul := 2, next3mul := 3, next5Mul := 5 next := 1 ugluNum[0] := 1

for i := 1 to n, do next := minimum of next2Mul, next3Mul and next5Mul uglyNum[i] := next if next = next2Mul, then

i2 := i2 + 1

next2mul := uglyNum[i2] \* 2 if next = next3Mul, then

i3 := i3 + 1

next3mul := uglyNum[i3] \* 3 if next = next5Mul, then

i5 := i5 + 1

next5mul := uglyNum[i5] \* 5

done

return next

End

**Complexity Analysis:**

**Time Complexity :** O(n)

**Space Complexity:** O(n)

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

# include<iostream> using namespace std;

int min(int x, int y, int z) {

if(x < y) { if(x < z) return x; else return z;

}else { if(y < z) return y; else return z;

}

}

int getUglyNum(int n) { int uglyNum[n]; int i2 = 0, i3 = 0, i5 = 0; int next2mul = 2; int next3mul = 3; int next5mul = 5; int next = 1; uglyNum[0] = 1;

for (int i=1; i<n; i++) { next = min(next2mul, next3mul, next5mul); uglyNum[i] = next;

if (next == next2mul) { i2++; next2mul = uglyNum[i2]\*2;

}

if (next == next3mul) { i3++; next3mul = uglyNum[i3]\*3;

}

if (next == next5mul) { i5++; next5mul = uglyNum[i5]\*5;

}

} return next;

} int main() {

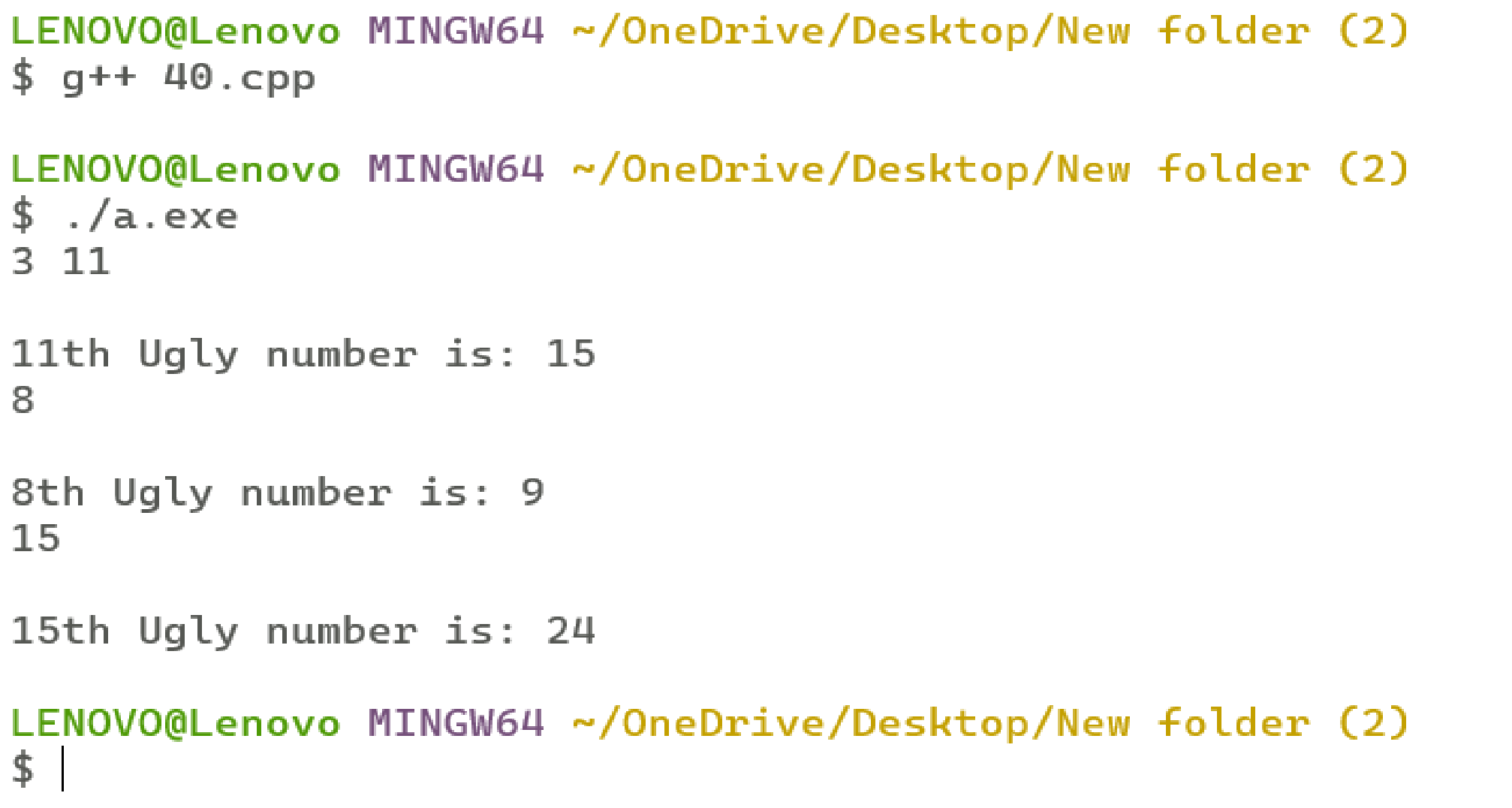
int t;

cin>>t; while(t--) { int n; cin >> n; cout <<"\n"<< n << "th Ugly number is: " << getUglyNum(n) << endl;

} return 0;

}

# Output



# Program 41

**Problem Statement:**Given a directed graph, write an algorithm and a program to find mother vertex in a graph. A mother vertex is a vertex v such that there exists a path from v to all other vertices of the graph**.**

**Algorithm:**

Do DFS traversal of the given graph. While doing traversal keep track of last finished vertex ‘ v’. This step takes O(V+E) time

If there exist mother vertex (or vertices), then v must be one (or one of them). Check if v is a mother vertex by doing DFS/BFS from v. This step also takes O(V+E) time.

**Complexity Analysis:**

**Time Complexity :** O(V+E) **Space Complexity:**

**Source Code:**

**/\***

**Name: Ashutosh Uniyal**

**Univ. Roll No: 2018249**

**Section: C**

**\*/**

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

{ int V; // No. of vertices list<int> \*adj; // adjacency lists void DFSUtil(int v, vector<bool> &visited);

public: Graph(int V); void addEdge(int v, int w); int findMother();

};

Graph::Graph(int V)

{

this->V = V; adj = new list<int>[V];

}

void Graph::DFSUtil(int v, vector<bool> &visited)

{ visited[v] = true; list<int>::iterator i; for (i = adj[v].begin(); i != adj[v].end(); ++i) if (!visited[\*i])

DFSUtil(\*i, visited);

}

void Graph::addEdge(int v, int w)

{ adj[v].push\_back(w); // Add w to v’s list.

}

int Graph::findMother()

{ vector <bool> visited(V, false); int v = 0; for (int i = 0; i < V; i++)

{ if (visited[i] == false)

{

DFSUtil(i, visited); v = i;

} }

fill(visited.begin(), visited.end(), false); DFSUtil(v, visited); for (int i=0; i<V; i++) if (visited[i] == false) return -1;

return v;

} int main()

{

Graph g(7);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(4, 1);

g.addEdge(6, 4);

g.addEdge(5, 6);

g.addEdge(5, 2);

g.addEdge(6, 0); cout << "A mother vertex is " << g.findMother(); return 0;

}

# Output

