Design Algorithm and Analysis Lab Practical

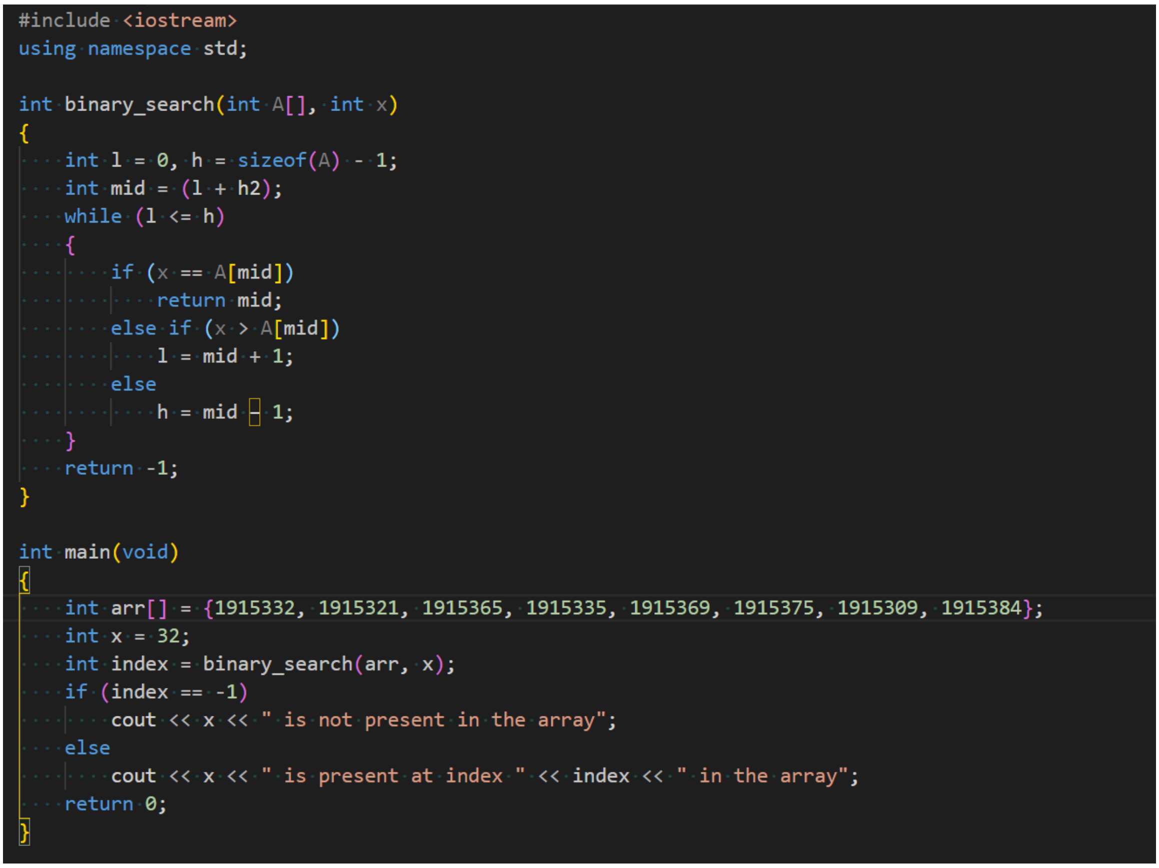
SUBMITTED TO: SUPREET MAM

Gautam Jain | BTech CSE C1 | 1915312/1905785

Practical – 1

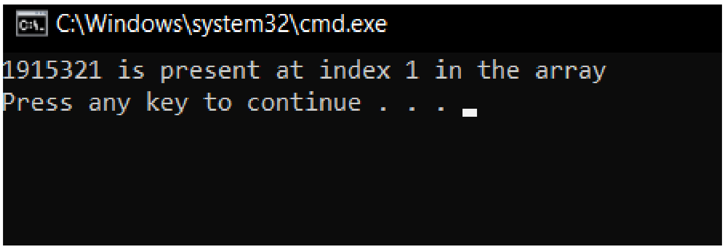
Aim: Write a program to find out a roll number from college database using binary search algorithm.

Code:



```cpp
#include <iostream>
using namespace std;

int binary_search(int A[], int x)
{
    int l = 0, h = sizeof(A) - 1;
    int mid = (l + h2);
    while (l <= h)
    {
        if (x == A[mid])
            return mid;
        else if (x > A[mid])
            l = mid + 1;
        else
            h = mid - 1;
    }
    return -1;
}

int main(void)
{
    int arr[] = {1915332, 1915321, 1915365, 1915335, 1915369, 1915375, 1915309, 1915384};
    int x = 32;
    int index = binary_search(arr, x);
    if (index == -1)
        cout << x << " is not present in the array";
    else
        cout << x << " is present at index " << index << " in the array";
    return 0;
}
```

Output:

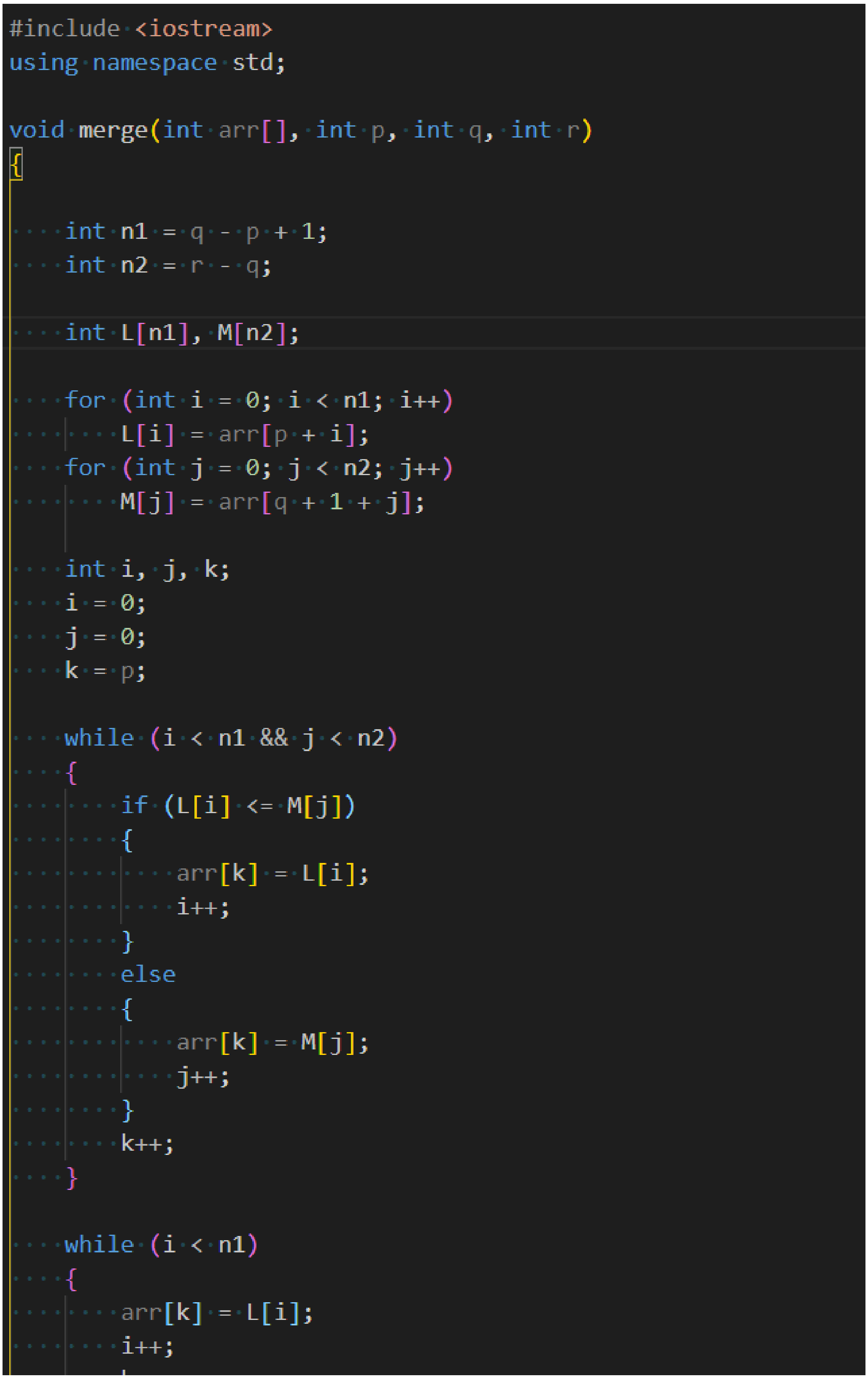


```
1915321 is present at index 1 in the array
Press any key to continue . . .
```

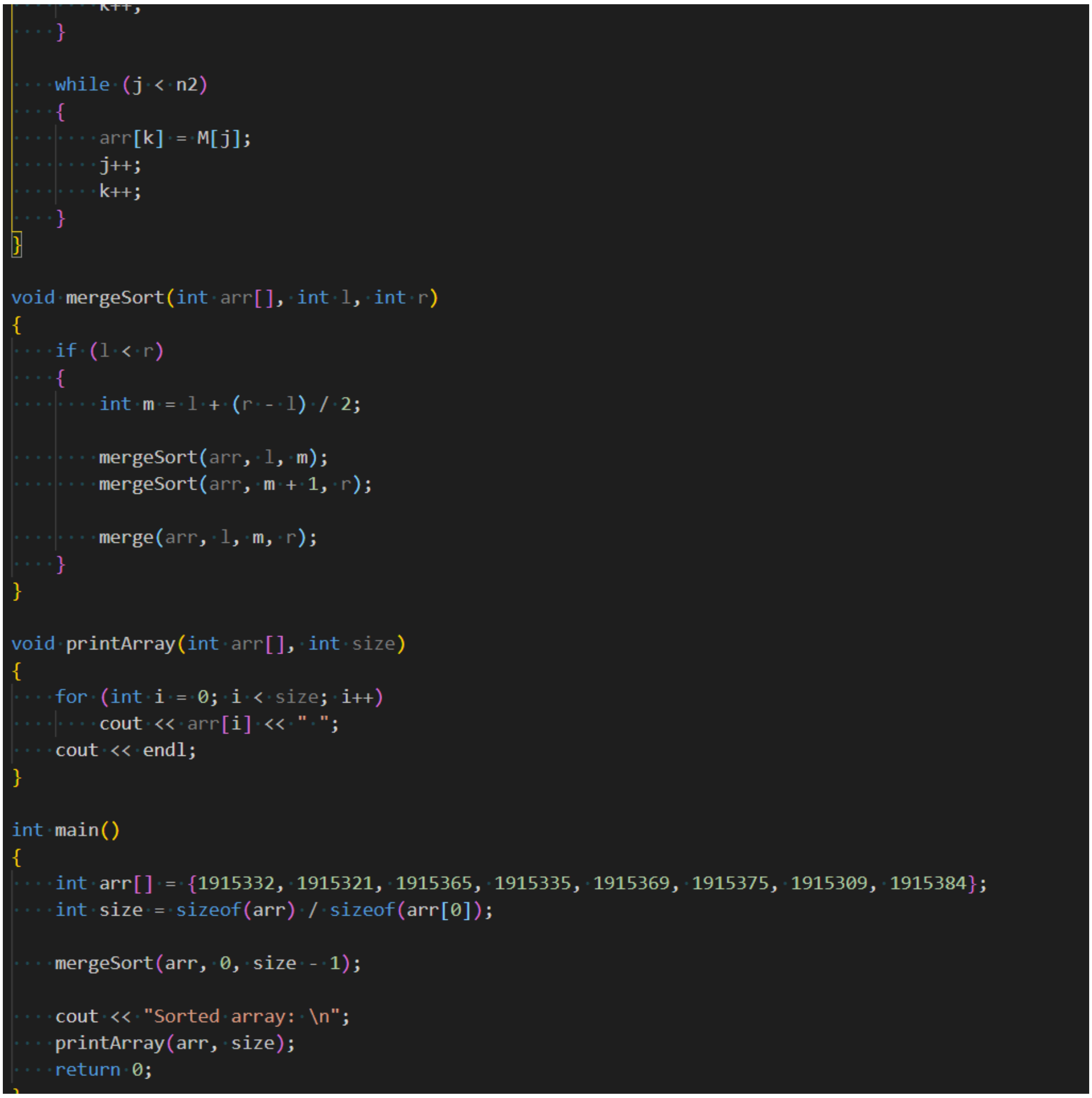
Practical – 2

Aim: Write a program to sort the class roll numbers of your class using merge sort algorithm and determine the time required to sort the elements.

Code:

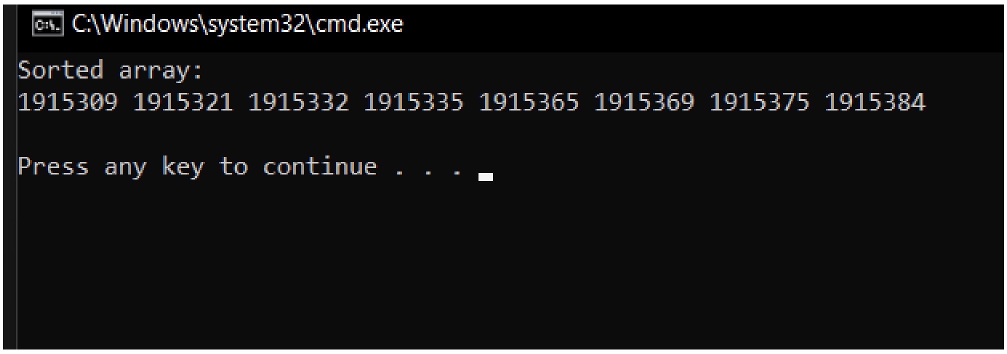


```cpp
#include <iostream>
using namespace std;

void merge(int arr[], int p, int q, int r)
{

    int n1 = q - p + 1;
    int n2 = r - q;

    int L[n1], M[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[p + i];
    for (int j = 0; j < n2; j++)
        M[j] = arr[q + 1 + j];

    int i, j, k;
    i = 0;
    j = 0;
    k = p;

    while (i < n1 && j < n2)
    {
        if (L[i] <= M[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = M[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
```



```
        k++;
    }

    while (j < n2)
    {
        arr[k] = M[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int arr[] = {1915332, 1915321, 1915365, 1915335, 1915369, 1915375, 1915309, 1915384};
    int size = sizeof(arr) / sizeof(arr[0]);

    mergeSort(arr, 0, size - 1);

    cout << "Sorted array: \n";
    printArray(arr, size);
    return 0;
}
```

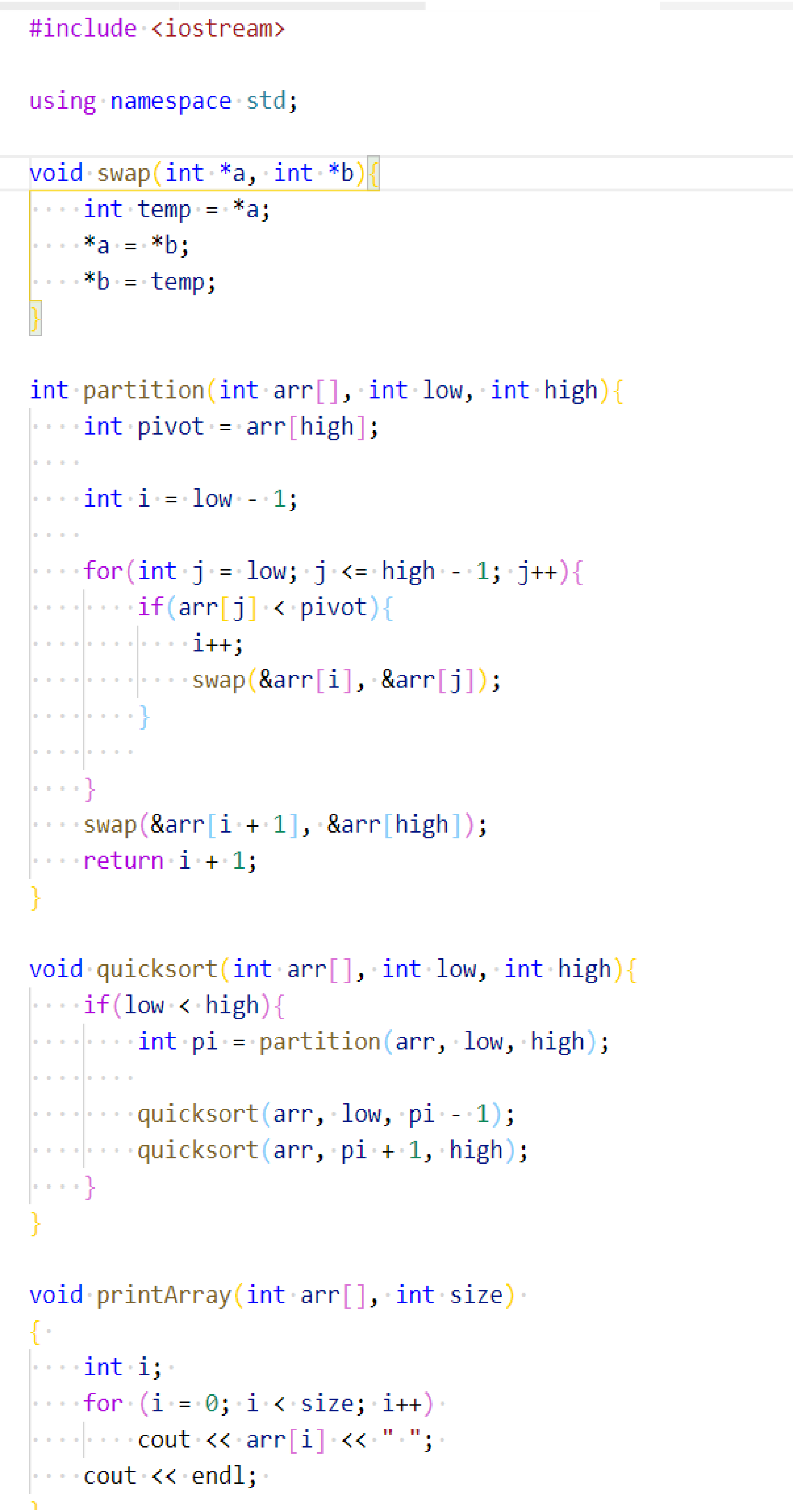
Output:



```
Sorted array:
1915309 1915321 1915332 1915335 1915365 1915369 1915375 1915384

Press any key to continue . . .
```

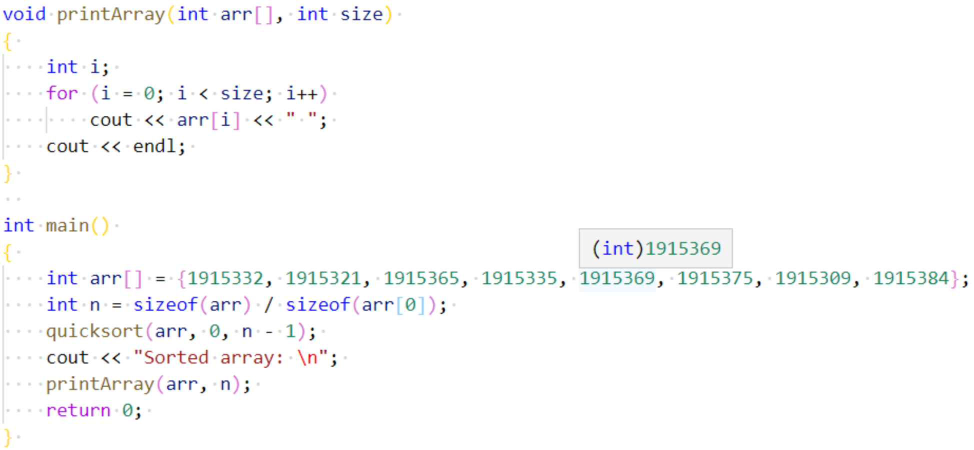
Practical – 3

Aim: Write a program to sort the university roll numbers of your class using Quick sort method and determine the time required to sort the elements.



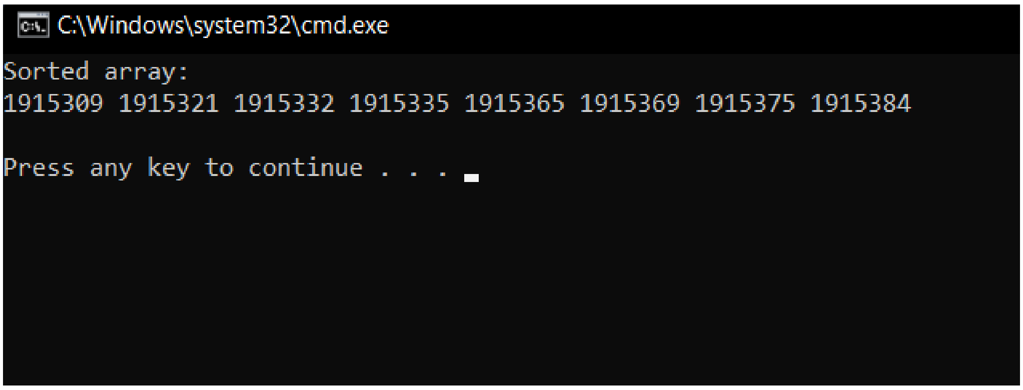
```cpp
#include <iostream>

using namespace std;

void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partition(int arr[], int low, int high){
    int pivot = arr[high];
    
    int i = low - 1;
    
    for(int j = low; j <= high - 1; j++){
        if(arr[j] < pivot){
            i++;
            swap(&arr[i], &arr[j]);
        }
        
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}

void quicksort(int arr[], int low, int high){
    if(low < high){
        int pi = partition(arr, low, high);
        
        quicksort(arr, low, pi - 1);
        quicksort(arr, pi + 1, high);
    }
}

void printArray(int arr[], int size) 
{ 
    int i; 
    for (i = 0; i < size; i++) 
        cout << arr[i] << " "; 
    cout << endl; 
}
```

Code:



```
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int arr[] = {1915332, 1915321, 1915365, 1915335, 1915369, 1915375, 1915309, 1915384};
    int n = sizeof(arr) / sizeof(arr[0]);
    quicksort(arr, 0, n - 1);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

Output:

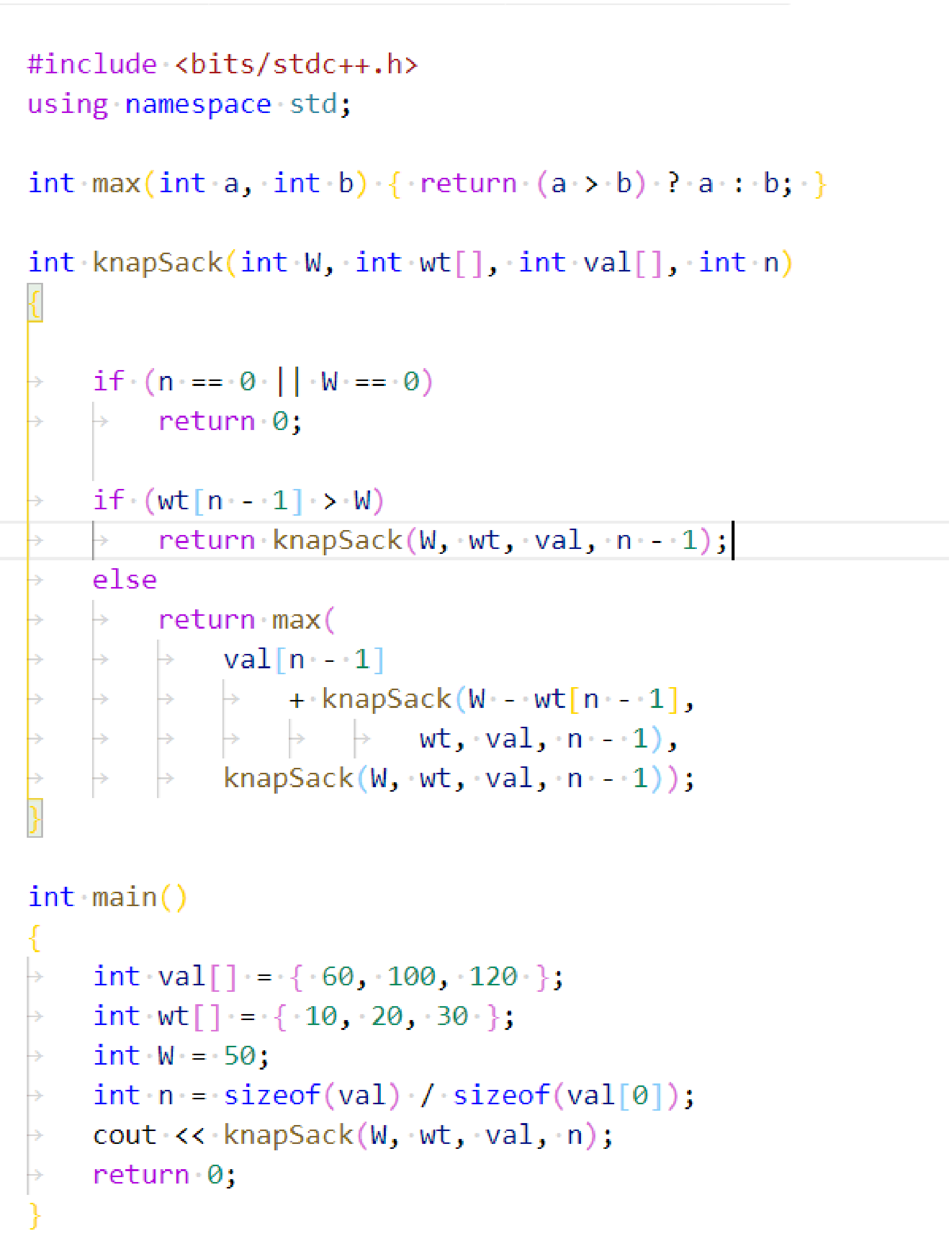


```
Sorted array:
1915309 1915321 1915332 1915335 1915365 1915369 1915375 1915384

Press any key to continue . . .
```

Practical – 4

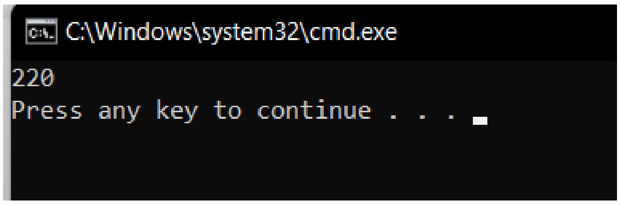
Aim: Write a program to solve 0/1 knapsack using Greedy algorithm.

Code:



```
#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)
{

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

    if (wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);
    else
        return max(
            val[n - 1]
                + knapSack(W - wt[n - 1],
                            wt, val, n - 1),
            knapSack(W, wt, val, n - 1));
}

int main()
{
    int val[] = { 60, 100, 120 };
    int wt[] = { 10, 20, 30 };
    int W = 50;
    int n = sizeof(val) / sizeof(val[0]);
    cout << knapSack(W, wt, val, n);
    return 0;
}
```

Output:

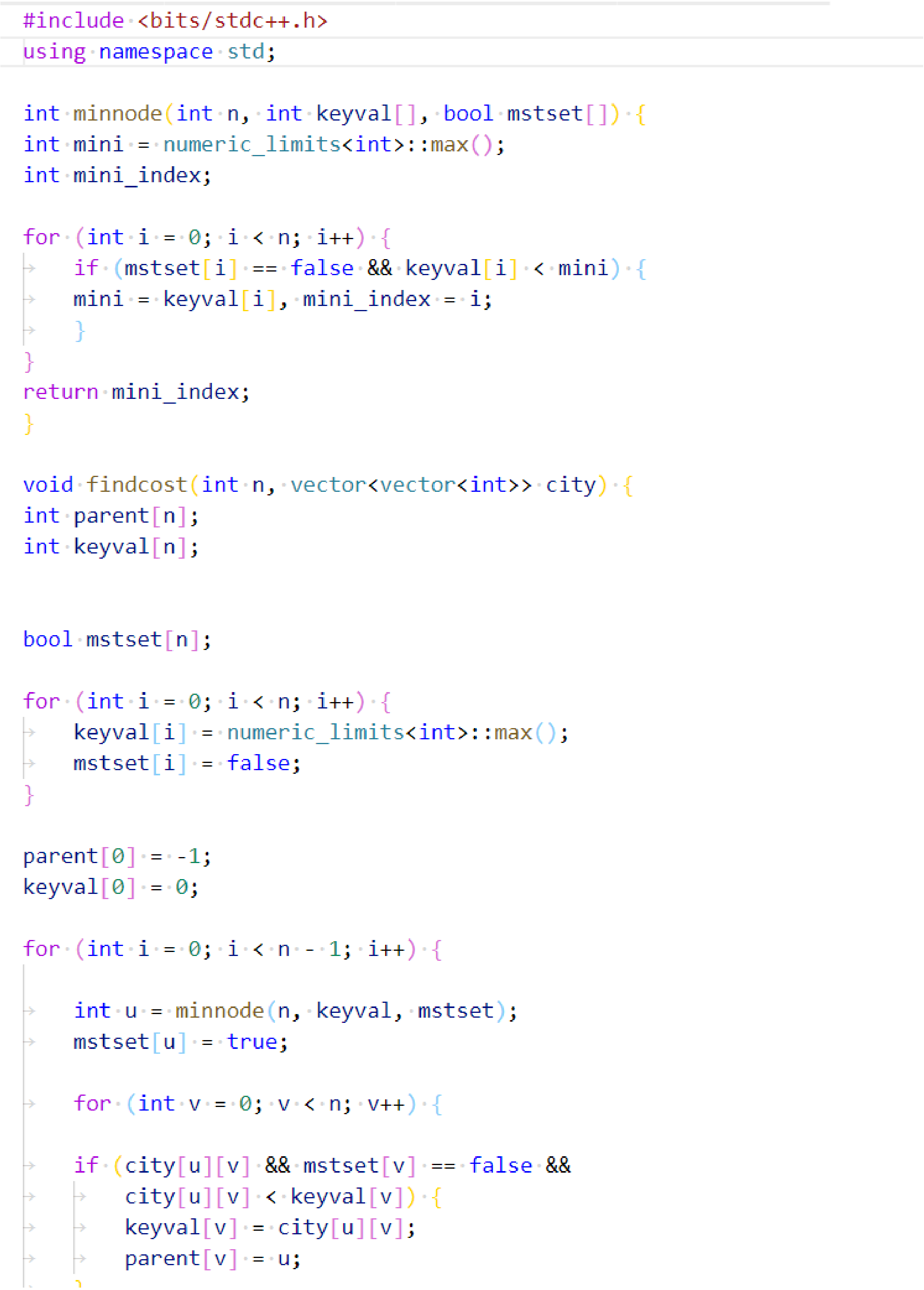


```
220
Press any key to continue . . .
```

Practical – 5

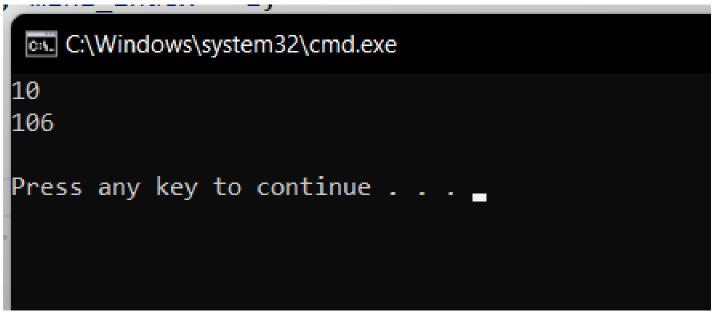
Aim: Write a program to find minimum cost to set the phone lines to connect all the cities of your state using Prim's algorithm

Code:



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

int minnode(int n, int keyval[], bool mstset[]) {
int mini = numeric_limits<int>::max();
int mini_index;

for (int i = 0; i < n; i++) {
    if (mstset[i] == false && keyval[i] < mini) {
    mini = keyval[i], mini_index = i;
    }
}
return mini_index;
}

void findcost(int n, vector<vector<int>> city) {
int parent[n];
int keyval[n];


bool mstset[n];

for (int i = 0; i < n; i++) {
    keyval[i] = numeric_limits<int>::max();
    mstset[i] = false;
}

parent[0] = -1;
keyval[0] = 0;

for (int i = 0; i < n - 1; i++) {

    int u = minnode(n, keyval, mstset);
    mstset[u] = true;

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

    if (city[u][v] && mstset[v] == false &&
        city[u][v] < keyval[v]) {
        keyval[v] = city[u][v];
        parent[v] = u;
```

Output:

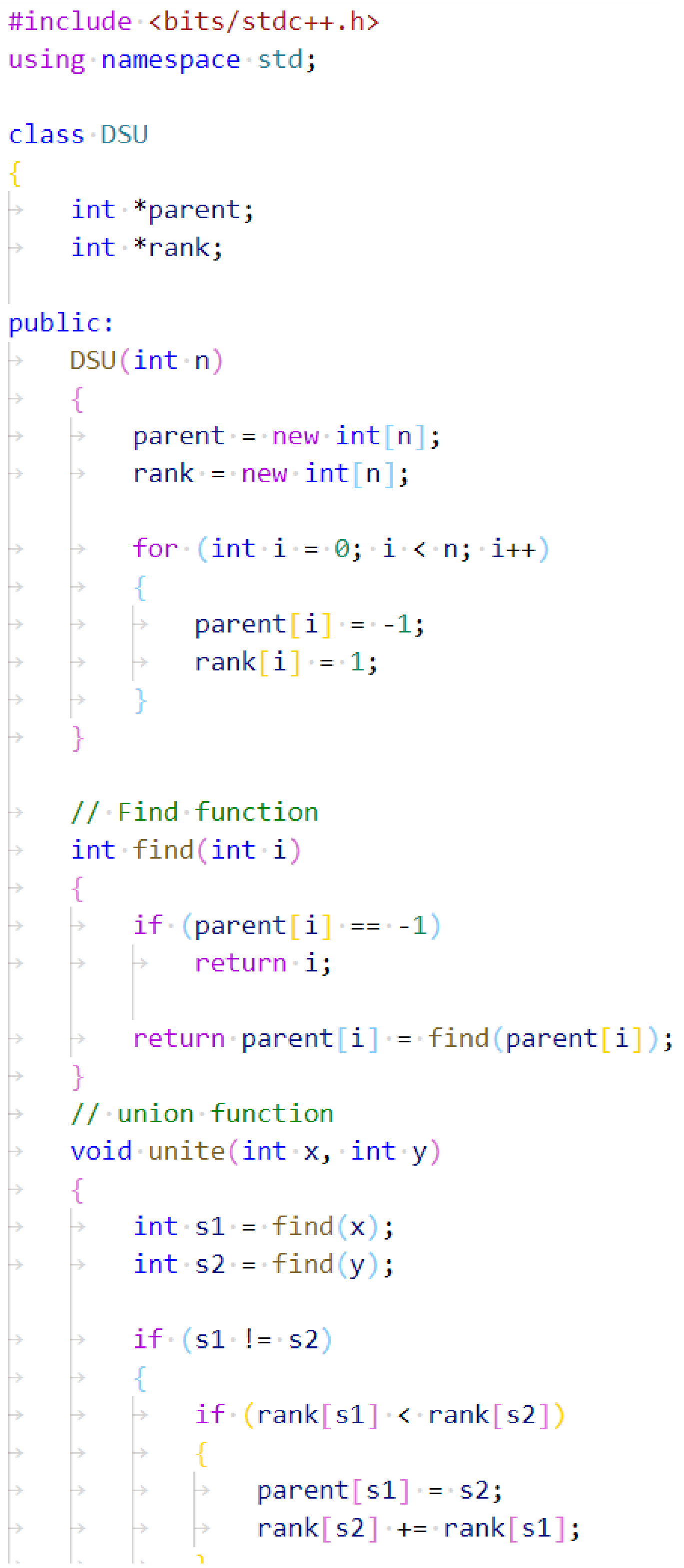


```
10
106

Press any key to continue . . .
```

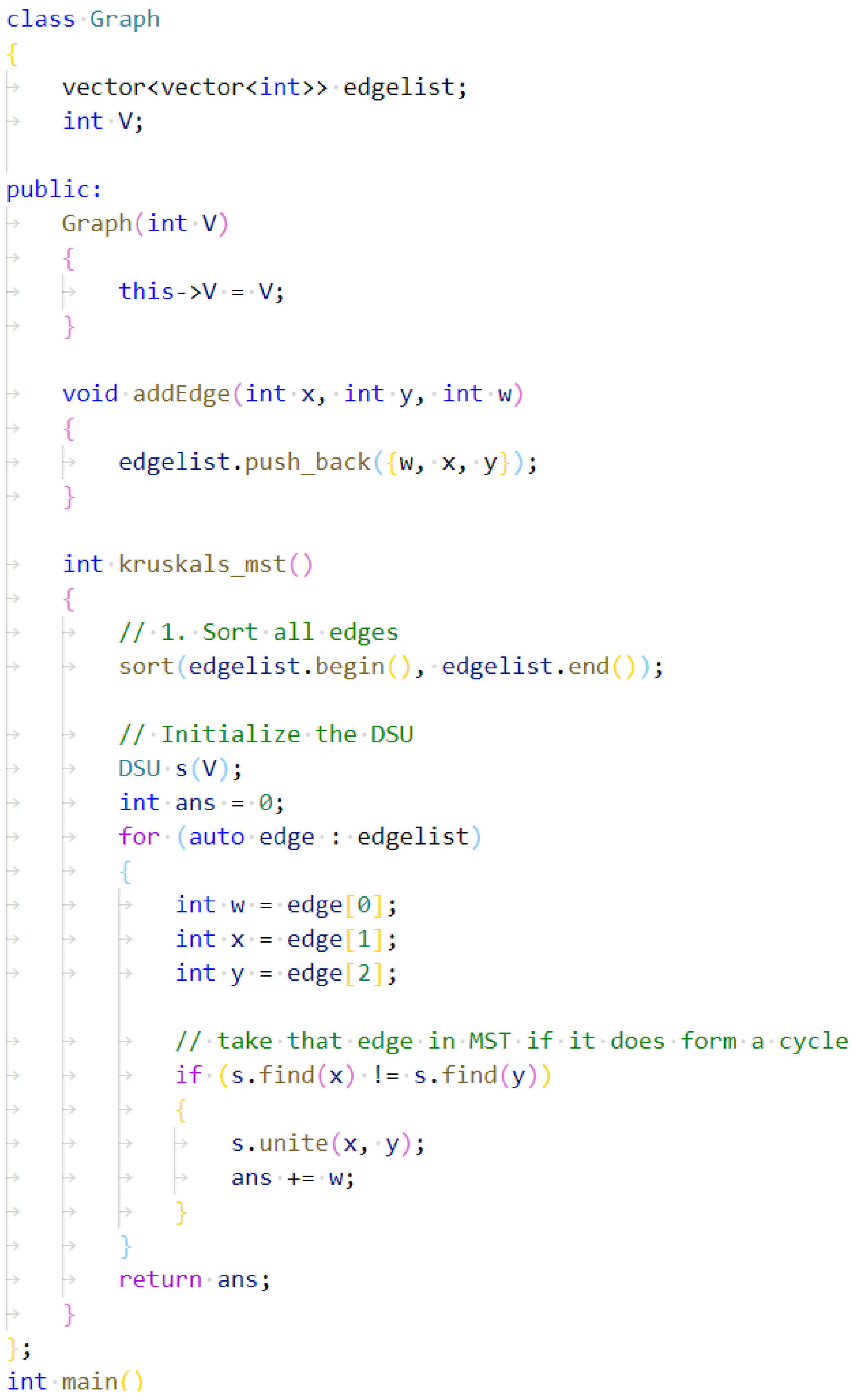
Practical – 6

Aim: Write a program to find the minimum cost of connecting all the engineering colleges in your state using Kruskal's algorithm.

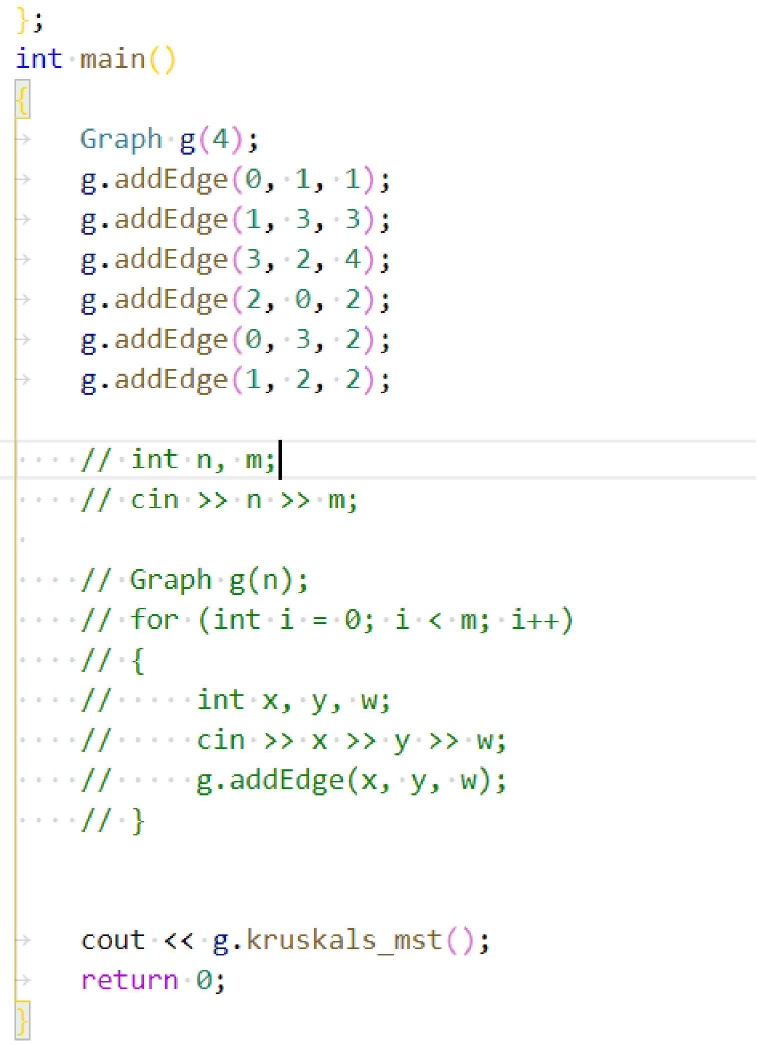
Code:



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

class DSU
{
    int *parent;
    int *rank;

public:
    DSU(int n)
    {
        parent = new int[n];
        rank = new int[n];

        for (int i = 0; i < n; i++)
        {
            parent[i] = -1;
            rank[i] = 1;
        }
    }

    // Find function
    int find(int i)
    {
        if (parent[i] == -1)
            return i;

        return parent[i] = find(parent[i]);
    }
    // union function
    void unite(int x, int y)
    {
        int s1 = find(x);
        int s2 = find(y);

        if (s1 != s2)
        {
            if (rank[s1] < rank[s2])
            {
                parent[s1] = s2;
                rank[s2] += rank[s1];
            }
```

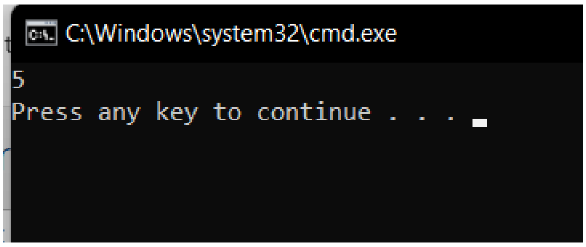


```
class Graph
{
    vector<vector<int>> edgelist;
    int V;

public:
    Graph(int V)
    {
        this->V = V;
    }

    void addEdge(int x, int y, int w)
    {
        edgelist.push_back({w, x, y});
    }

    int kruskals_mst()
    {
        // 1. Sort all edges
        sort(edgelist.begin(), edgelist.end());

        // Initialize the DSU
        DSU s(V);
        int ans = 0;
        for (auto edge : edgelist)
        {
            int w = edge[0];
            int x = edge[1];
            int y = edge[2];

            // take that edge in MST if it does form a cycle
            if (s.find(x) != s.find(y))
            {
                s.unite(x, y);
                ans += w;
            }
        }
        return ans;
    }
};
int main()
```



```
};
int main()
{
    Graph g(4);
    g.addEdge(0, 1, 1);
    g.addEdge(1, 3, 3);
    g.addEdge(3, 2, 4);
    g.addEdge(2, 0, 2);
    g.addEdge(0, 3, 2);
    g.addEdge(1, 2, 2);

    // int n, m;
    // cin >> n >> m;

    // Graph g(n);
    // for (int i = 0; i < m; i++)
    // {
    //     int x, y, w;
    //     cin >> x >> y >> w;
    //     g.addEdge(x, y, w);
    // }


    cout << g.kruskals_mst();
    return 0;
}
```

Output:

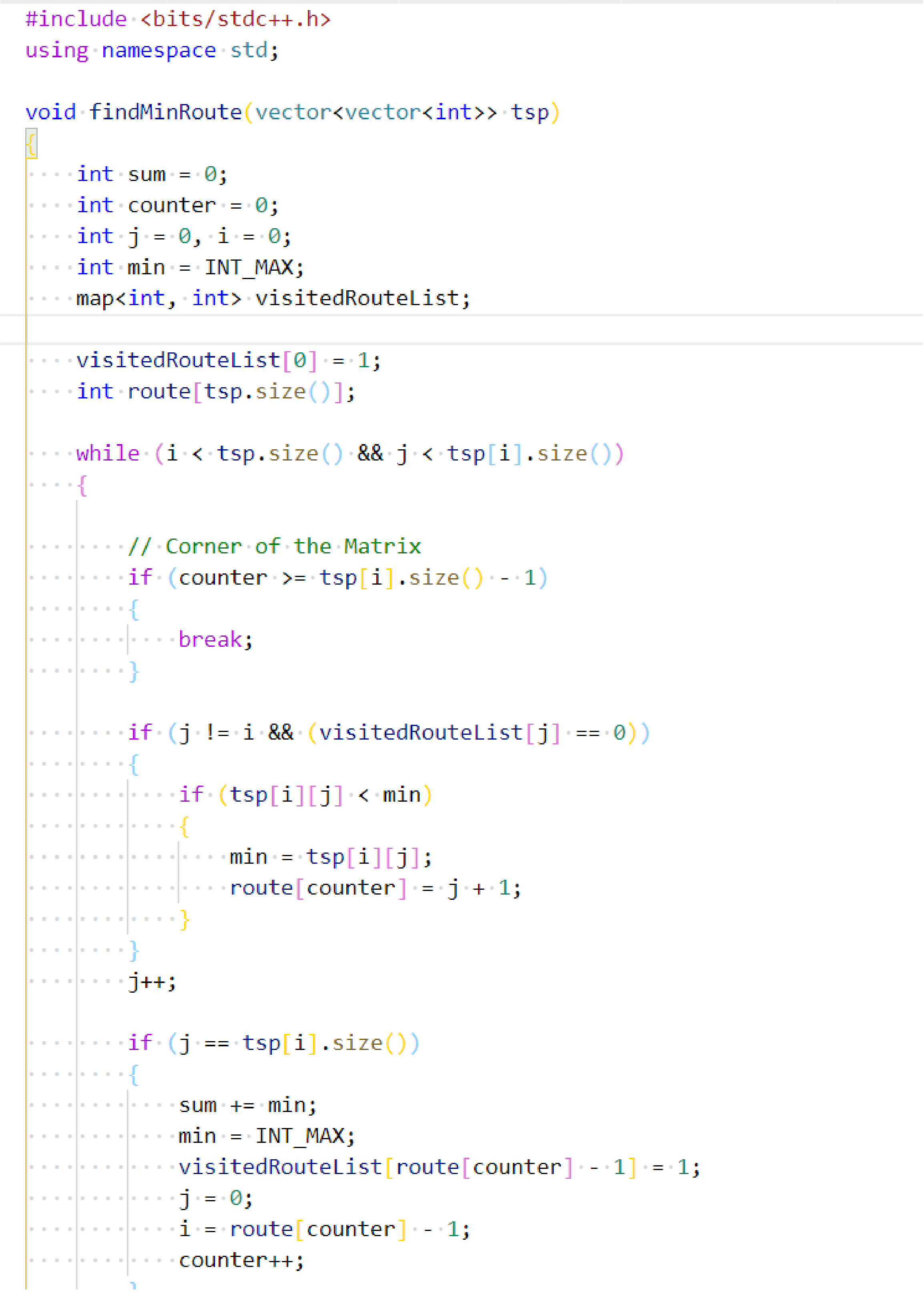


```
5
Press any key to continue . . .
```

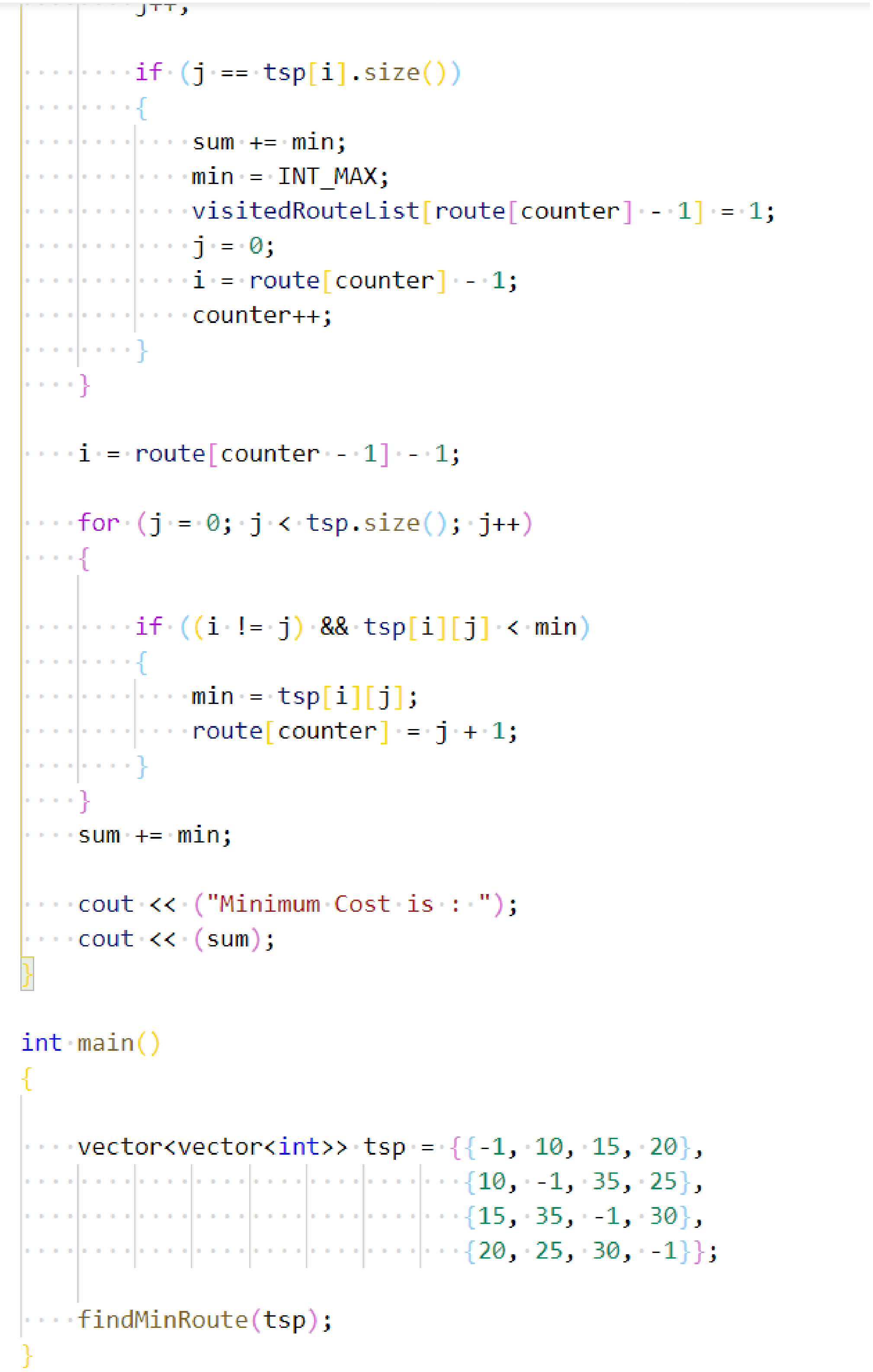
Practical – 7

Aim: Write a program to find minimum route for a newspaper distributer of your locality using Greedy algorithm.

Code:

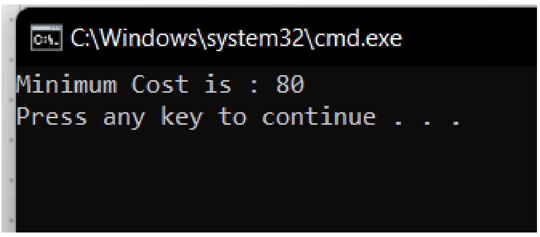


```
#include <bits/stdc++.h>
using namespace std;

void findMinRoute(vector<vector<int>> tsp)
{
    int sum = 0;
    int counter = 0;
    int j = 0, i = 0;
    int min = INT_MAX;
    map<int, int> visitedRouteList;

    visitedRouteList[0] = 1;
    int route[tsp.size()];

    while (i < tsp.size() && j < tsp[i].size())
    {

        // Corner of the Matrix
        if (counter >= tsp[i].size() - 1)
        {
            break;
        }

        if (j != i && (visitedRouteList[j] == 0))
        {
            if (tsp[i][j] < min)
            {
                min = tsp[i][j];
                route[counter] = j + 1;
            }
        }
        j++;

        if (j == tsp[i].size())
        {
            sum += min;
            min = INT_MAX;
            visitedRouteList[route[counter] - 1] = 1;
            j = 0;
            i = route[counter] - 1;
            counter++;
```



```
        j++;

        if (j == tsp[i].size())
        {
            sum += min;
            min = INT_MAX;
            visitedRouteList[route[counter] - 1] = 1;
            j = 0;
            i = route[counter] - 1;
            counter++;
        }
    }

    i = route[counter - 1] - 1;

    for (j = 0; j < tsp.size(); j++)
    {

        if ((i != j) && tsp[i][j] < min)
        {
            min = tsp[i][j];
            route[counter] = j + 1;
        }
    }
    sum += min;

    cout << ("Minimum Cost is : ");
    cout << (sum);
}

int main()
{

    vector<vector<int>> tsp = {{-1, 10, 15, 20},
                               {10, -1, 35, 25},
                               {15, 35, -1, 30},
                               {20, 25, 30, -1}};

    findMinRoute(tsp);
}
```

Output:

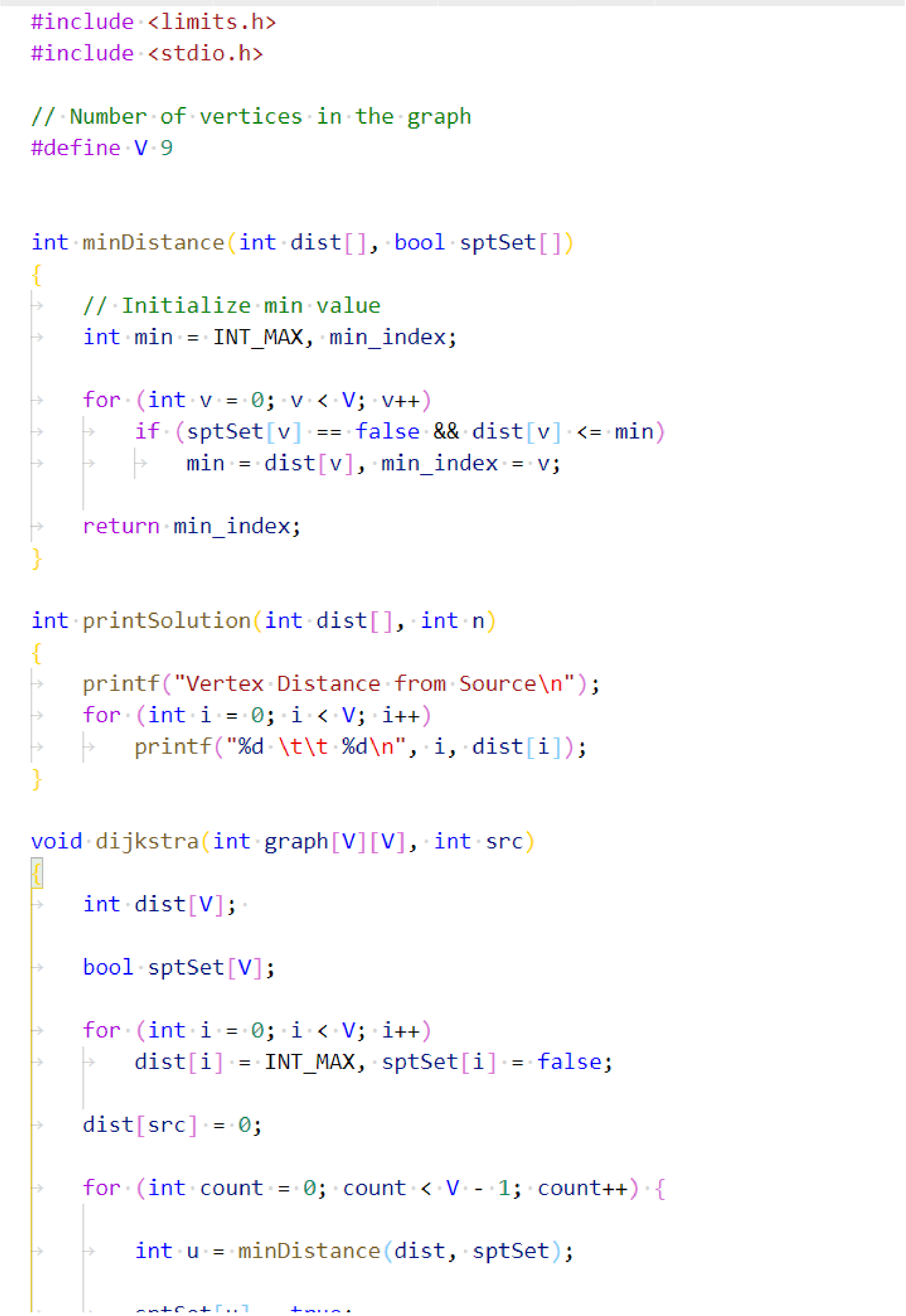


```
Minimum Cost is : 80
Press any key to continue . . .
```

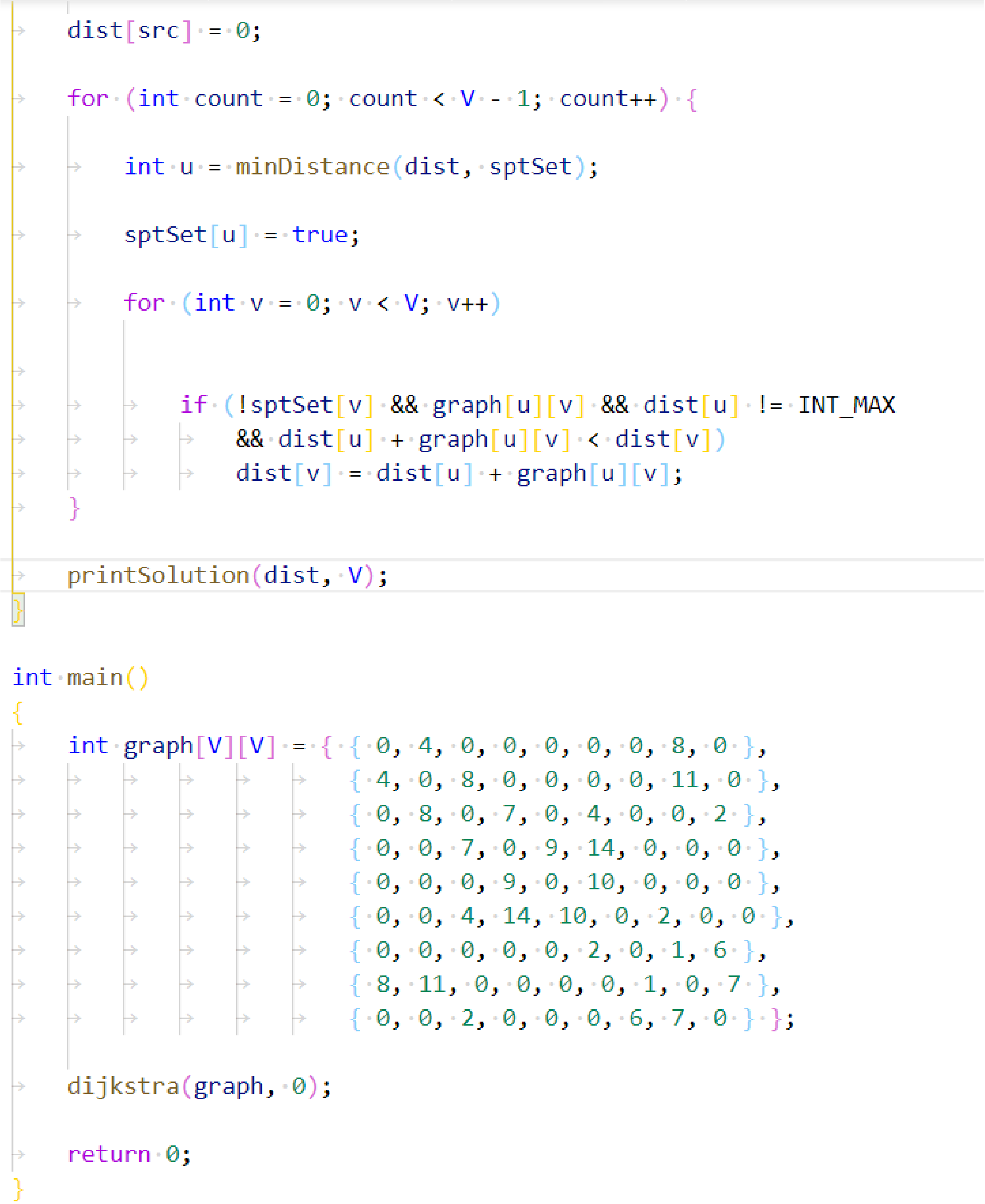
Practical – 8

Aim: Write a program to find shortest path from your home to college using Dijkstra's algorithm

Code:

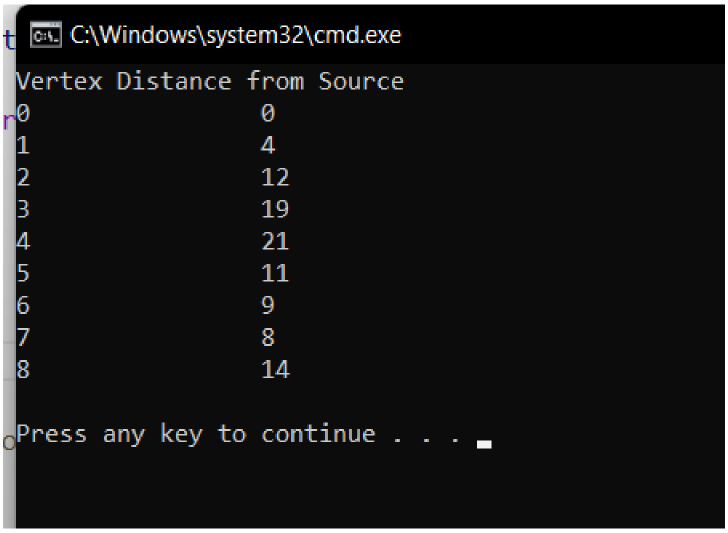


```
#include <limits.h>
#include <stdio.h>

// Number of vertices in the graph
#define V 9


int minDistance(int dist[], bool sptSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

int printSolution(int dist[], int n)
{
    printf("Vertex Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int src)
{
    int dist[V];

    bool sptSet[V];

    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    dist[src] = 0;

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

        int u = minDistance(dist, sptSet);

        sptSet[u] = true;
```



```
    dist[src] = 0;

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

        int u = minDistance(dist, sptSet);

        sptSet[u] = true;

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


            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    printSolution(dist, V);
}

int main()
{
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    dijkstra(graph, 0);

    return 0;
}
```

Output:

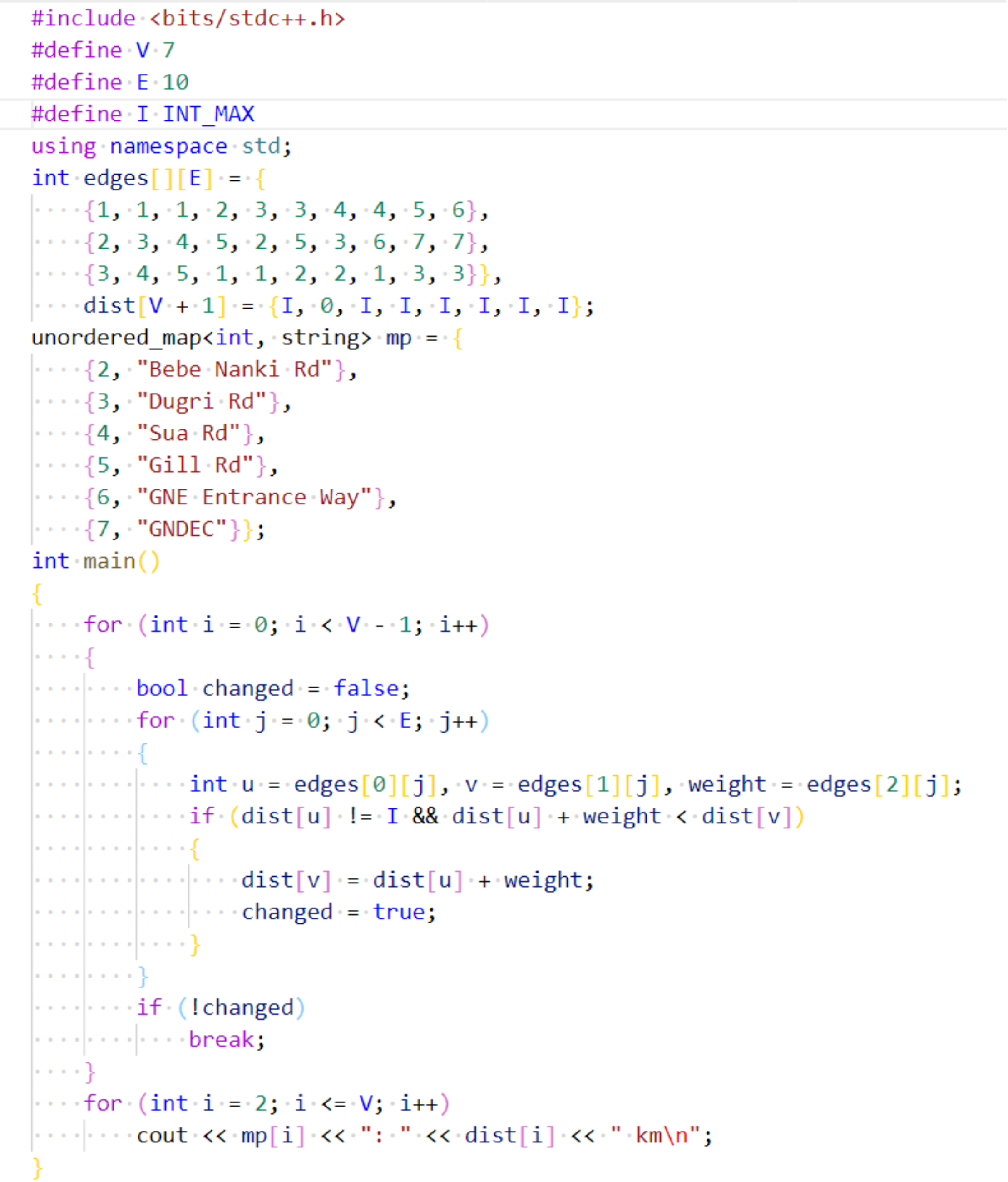


```
Vertex Distance from Source
0                0
1                4
2                12
3                19
4                21
5                11
6                9
7                8
8                14

Press any key to continue . . .
```

Practical – 9

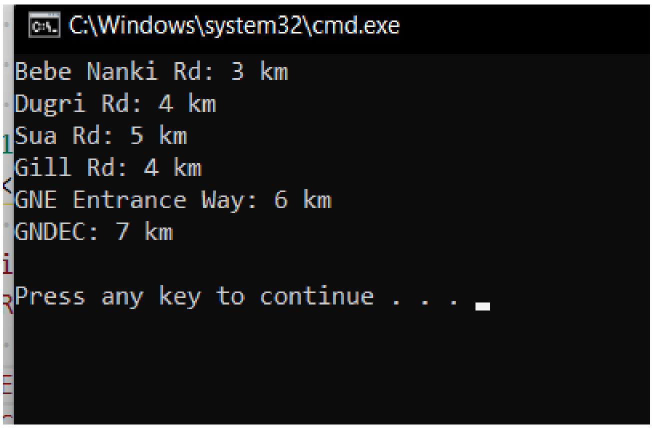
Aim: Write a program to find shortest path from your home to college using Bellman-Ford algorithm

Code:



```cpp
#include <bits/stdc++.h>
#define V 7
#define E 10
#define I INT_MAX
using namespace std;
int edges[][E] = {
    {1, 1, 1, 2, 3, 3, 4, 4, 5, 6},
    {2, 3, 4, 5, 2, 5, 3, 6, 7, 7},
    {3, 4, 5, 1, 1, 2, 2, 1, 3, 3}},
    dist[V + 1] = {I, 0, I, I, I, I, I, I};
unordered_map<int, string> mp = {
    {2, "Bebe Nanki Rd"},
    {3, "Dugri Rd"},
    {4, "Sua Rd"},
    {5, "Gill Rd"},
    {6, "GNE Entrance Way"},
    {7, "GNDEC"}};
int main()
{
    for (int i = 0; i < V - 1; i++)
    {
        bool changed = false;
        for (int j = 0; j < E; j++)
        {
            int u = edges[0][j], v = edges[1][j], weight = edges[2][j];
            if (dist[u] != I && dist[u] + weight < dist[v])
            {
                dist[v] = dist[u] + weight;
                changed = true;
            }
        }
        if (!changed)
            break;
    }
    for (int i = 2; i <= V; i++)
        cout << mp[i] << ": " << dist[i] << " km\n";
}
```

Output:

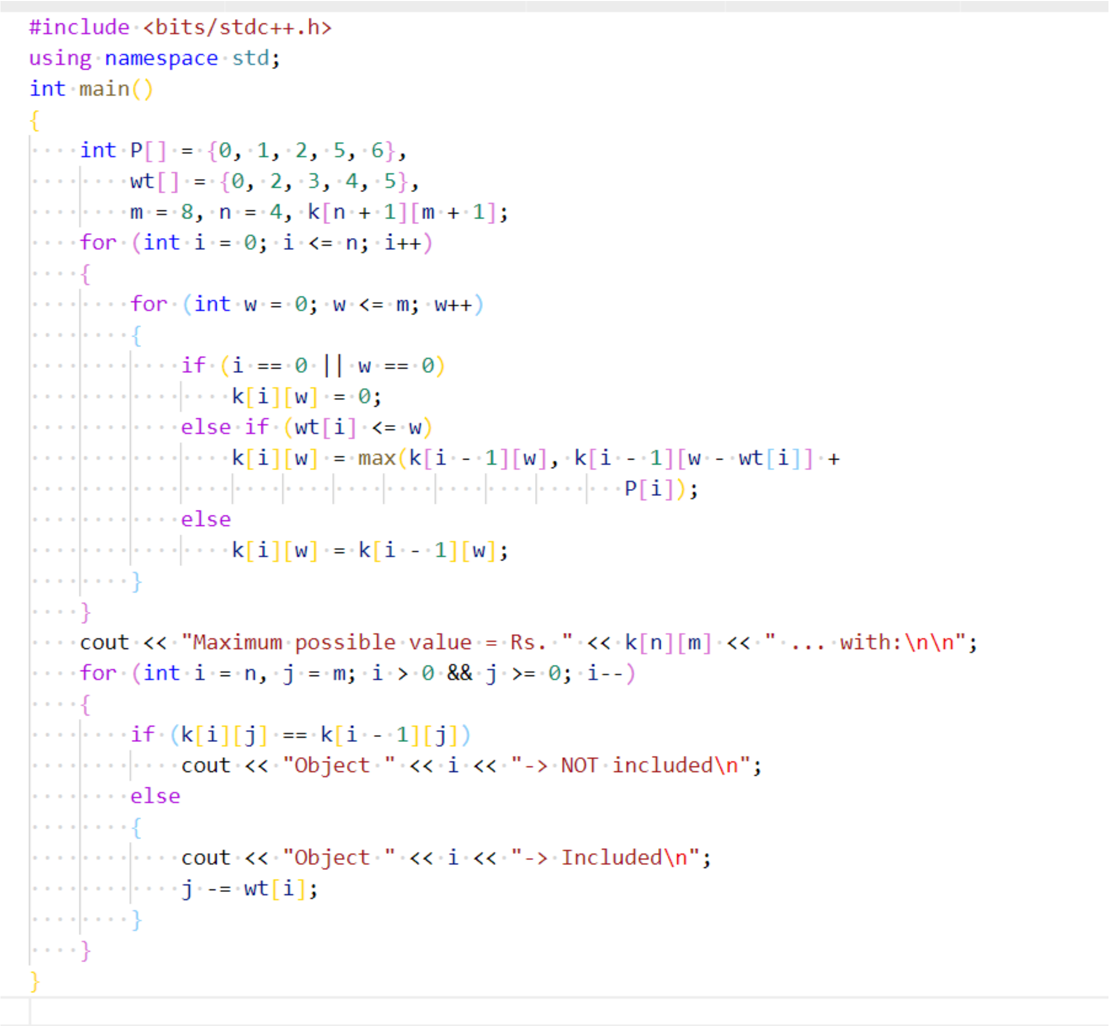


```
Bebe Nanki Rd: 3 km
Dugri Rd: 4 km
Sua Rd: 5 km
Gill Rd: 4 km
GNE Entrance Way: 6 km
GNDEC: 7 km

Press any key to continue . . .
```

Practical – 10

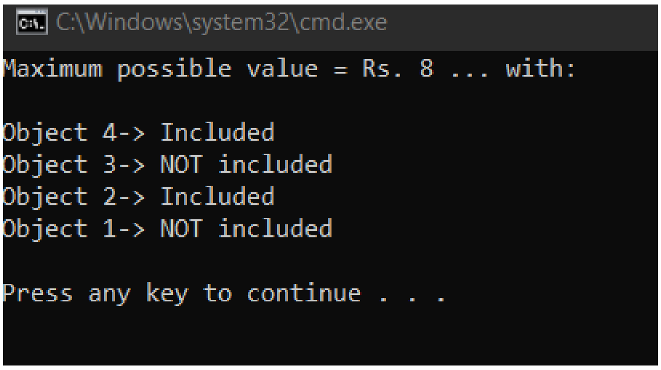
Aim: Write a program to solve 0/1 knapsack using dynamic programming.

Code:



```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int P[] = {0, 1, 2, 5, 6},
        wt[] = {0, 2, 3, 4, 5},
        m = 8, n = 4, k[n + 1][m + 1];
    for (int i = 0; i <= n; i++)
    {
        for (int w = 0; w <= m; w++)
        {
            if (i == 0 || w == 0)
                k[i][w] = 0;
            else if (wt[i] <= w)
                k[i][w] = max(k[i - 1][w], k[i - 1][w - wt[i]] +
                                                P[i]);
            else
                k[i][w] = k[i - 1][w];
        }
    }
    cout << "Maximum possible value = Rs. " << k[n][m] << " ... with:\n\n";
    for (int i = n, j = m; i > 0 && j >= 0; i--)
    {
        if (k[i][j] == k[i - 1][j])
            cout << "Object " << i << "-> NOT included\n";
        else
        {
            cout << "Object " << i << "-> Included\n";
            j -= wt[i];
        }
    }
}
```

Output:

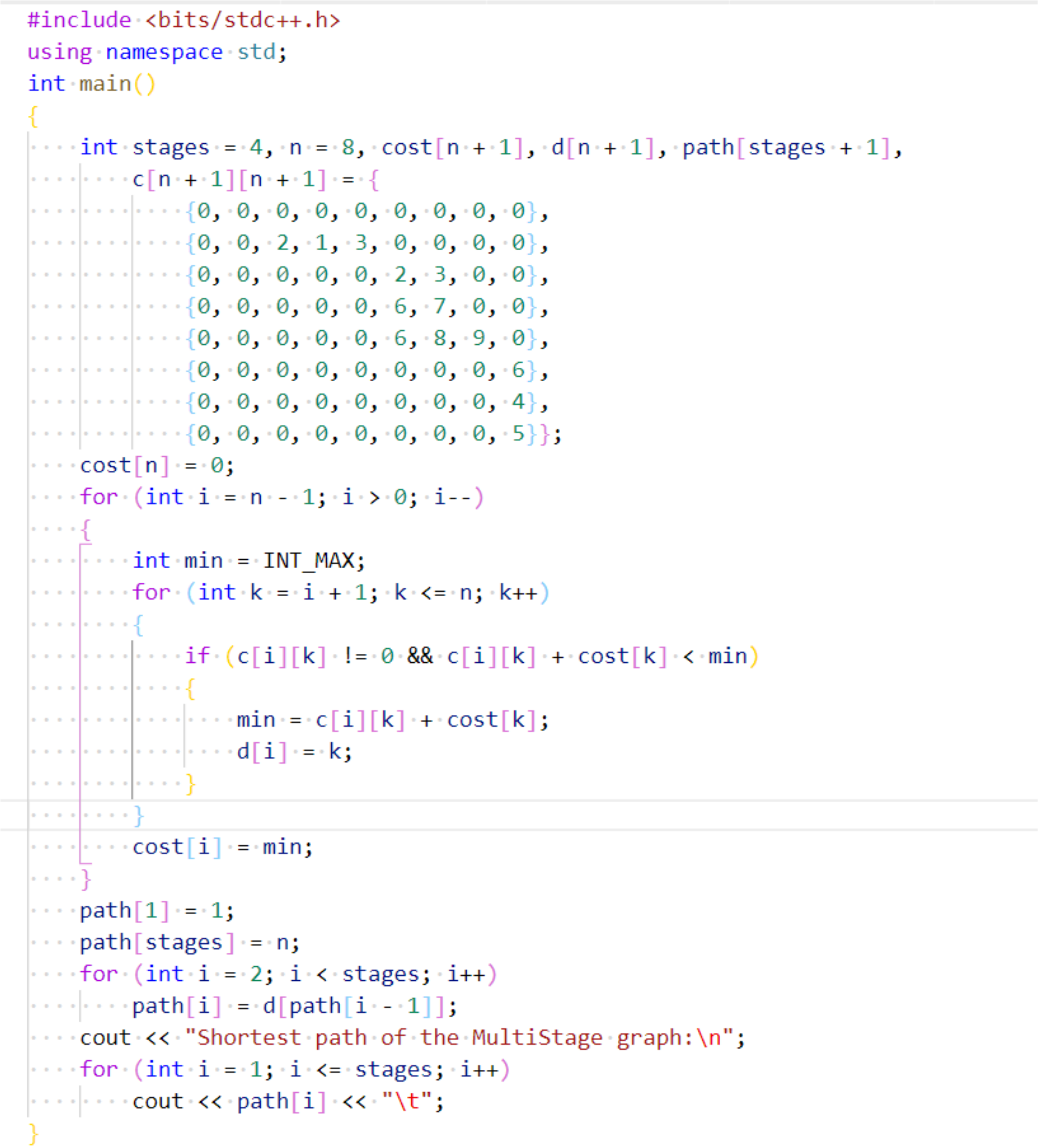


```
Maximum possible value = Rs. 8 ... with:

Object 4-> Included
Object 3-> NOT included
Object 2-> Included
Object 1-> NOT included

Press any key to continue . . .
```

Practical – 11

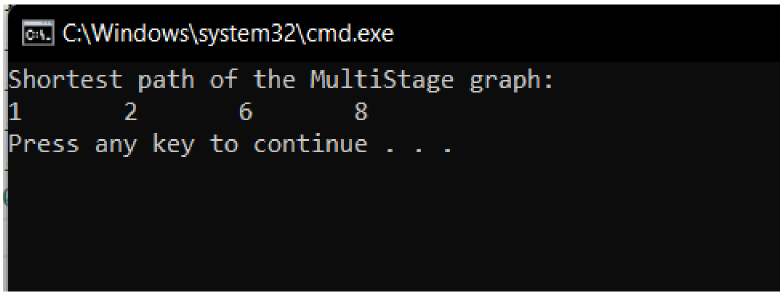
Aim: Write a program to find the shortest path of the multistage graph using dynamic programming.

Code:



```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int stages = 4, n = 8, cost[n + 1], d[n + 1], path[stages + 1],
        c[n + 1][n + 1] = {
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 2, 1, 3, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 2, 3, 0, 0},
            {0, 0, 0, 0, 0, 6, 7, 0, 0},
            {0, 0, 0, 0, 0, 6, 8, 9, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 6},
            {0, 0, 0, 0, 0, 0, 0, 0, 4},
            {0, 0, 0, 0, 0, 0, 0, 0, 5}};
    cost[n] = 0;
    for (int i = n - 1; i > 0; i--)
    {
        int min = INT_MAX;
        for (int k = i + 1; k <= n; k++)
        {
            if (c[i][k] != 0 && c[i][k] + cost[k] < min)
            {
                min = c[i][k] + cost[k];
                d[i] = k;
            }
        }
        cost[i] = min;
    }
    path[1] = 1;
    path[stages] = n;
    for (int i = 2; i < stages; i++)
        path[i] = d[path[i - 1]];
    cout << "Shortest path of the MultiStage graph:\n";
    for (int i = 1; i <= stages; i++)
        cout << path[i] << "\t";
}
```

Output:

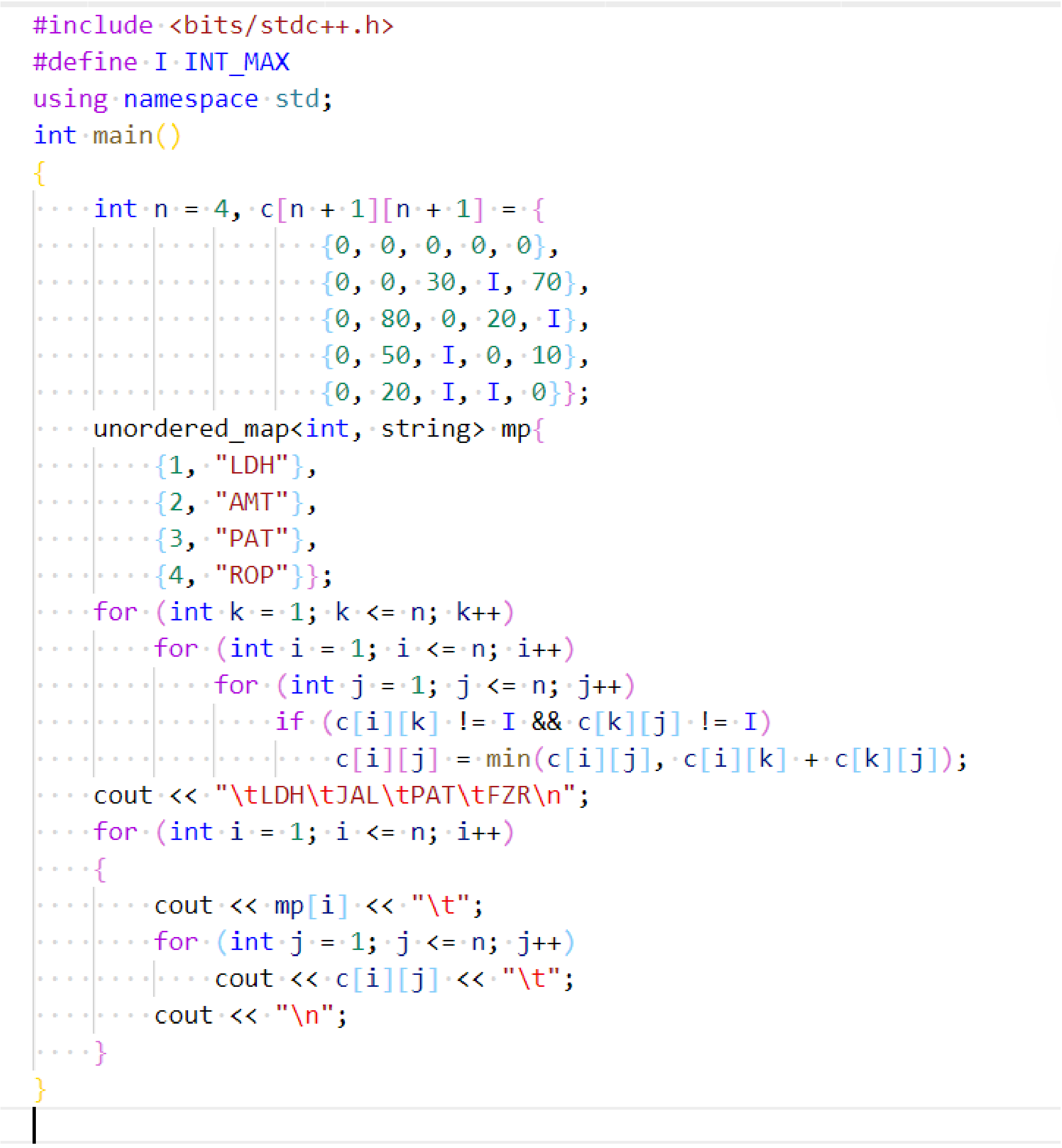


```
Shortest path of the MultiStage graph:
1       2       6       8
Press any key to continue . . .
```

Practical – 12

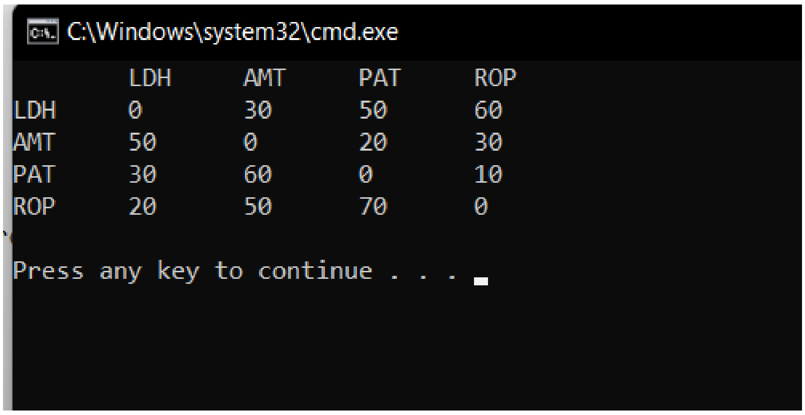
Aim: Write a program to find minimum distance between different cities of your state using FloydWarshall algorithm

Code:



```
#include <bits/stdc++.h>
#define I INT_MAX
using namespace std;
int main()
{
    int n = 4, c[n + 1][n + 1] = {
                   {0, 0, 0, 0, 0},
                   {0, 0, 30, I, 70},
                   {0, 80, 0, 20, I},
                   {0, 50, I, 0, 10},
                   {0, 20, I, I, 0}};
    unordered_map<int, string> mp{
        {1, "LDH"},
        {2, "AMT"},
        {3, "PAT"},
        {4, "ROP"}};
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                if (c[i][k] != I && c[k][j] != I)
                    c[i][j] = min(c[i][j], c[i][k] + c[k][j]);
    cout << "\tLDH\tJAL\tPAT\tFZR\n";
    for (int i = 1; i <= n; i++)
    {
        cout << mp[i] << "\t";
        for (int j = 1; j <= n; j++)
            cout << c[i][j] << "\t";
        cout << "\n";
    }
}
```

Output:

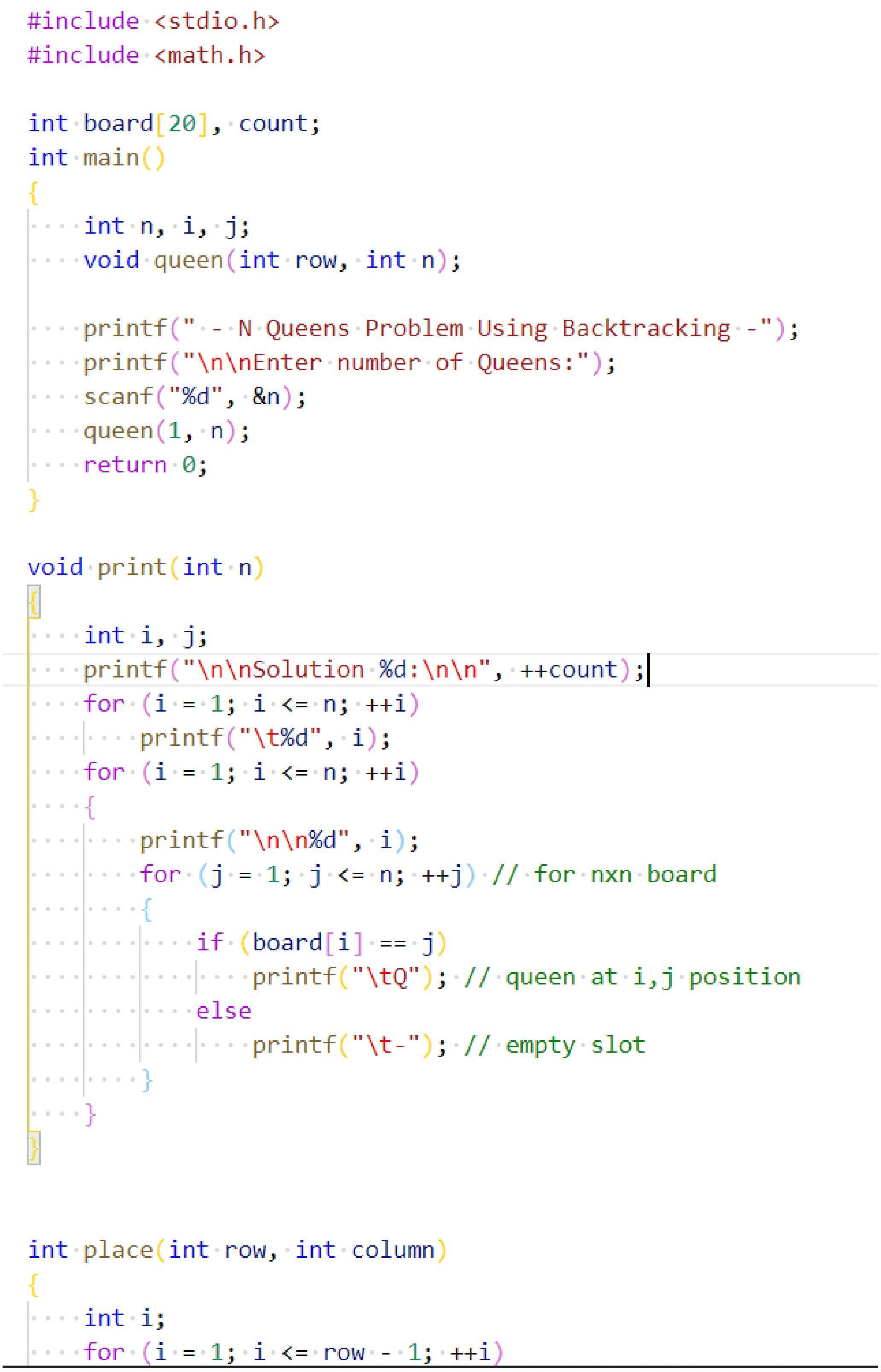


```
        LDH     AMT     PAT     ROP
LDH     0       30      50      60
AMT     50      0       20      30
PAT     30      60      0       10
ROP     20      50      70      0

Press any key to continue . . .
```

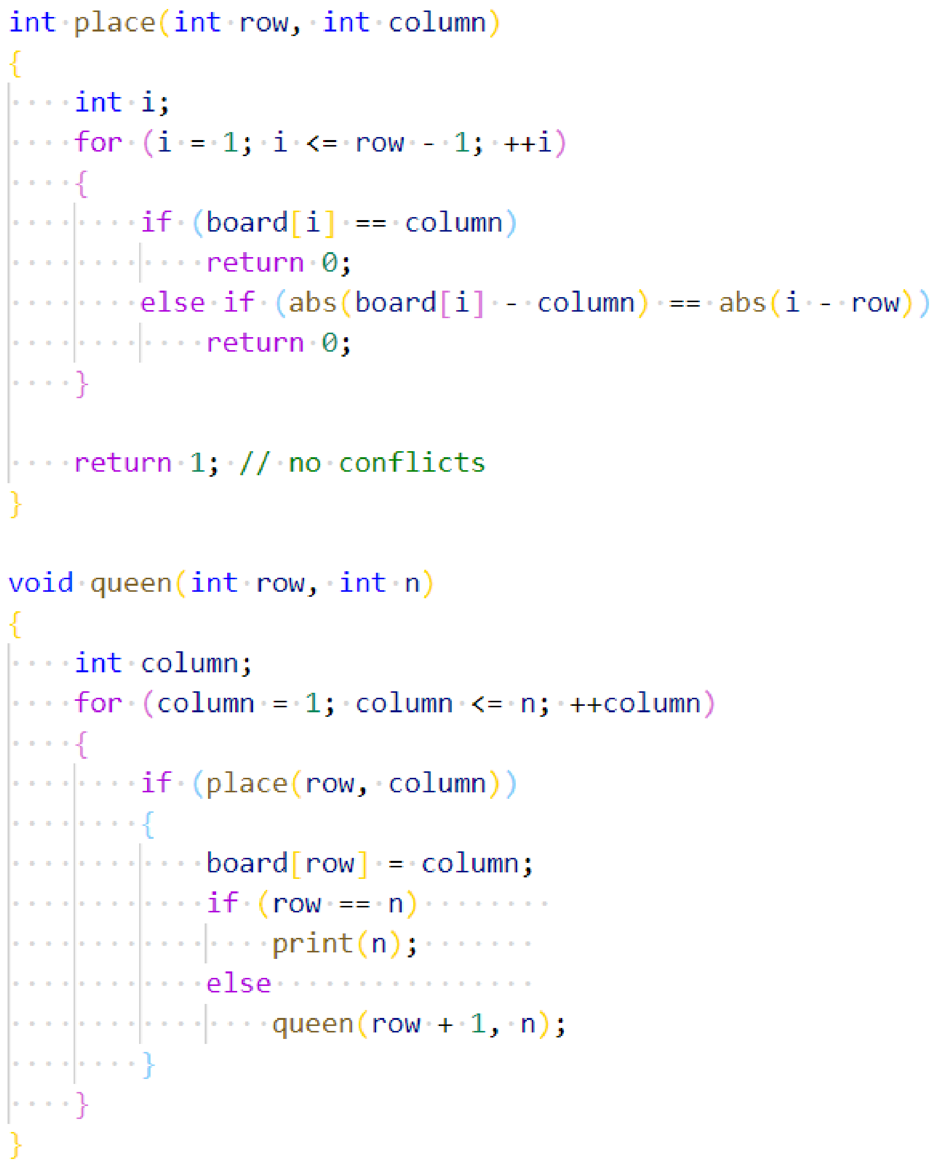
Practical – 13

Aim: Write a program to find the solution to the 8 queen’s problem using the backtracking.

Code:

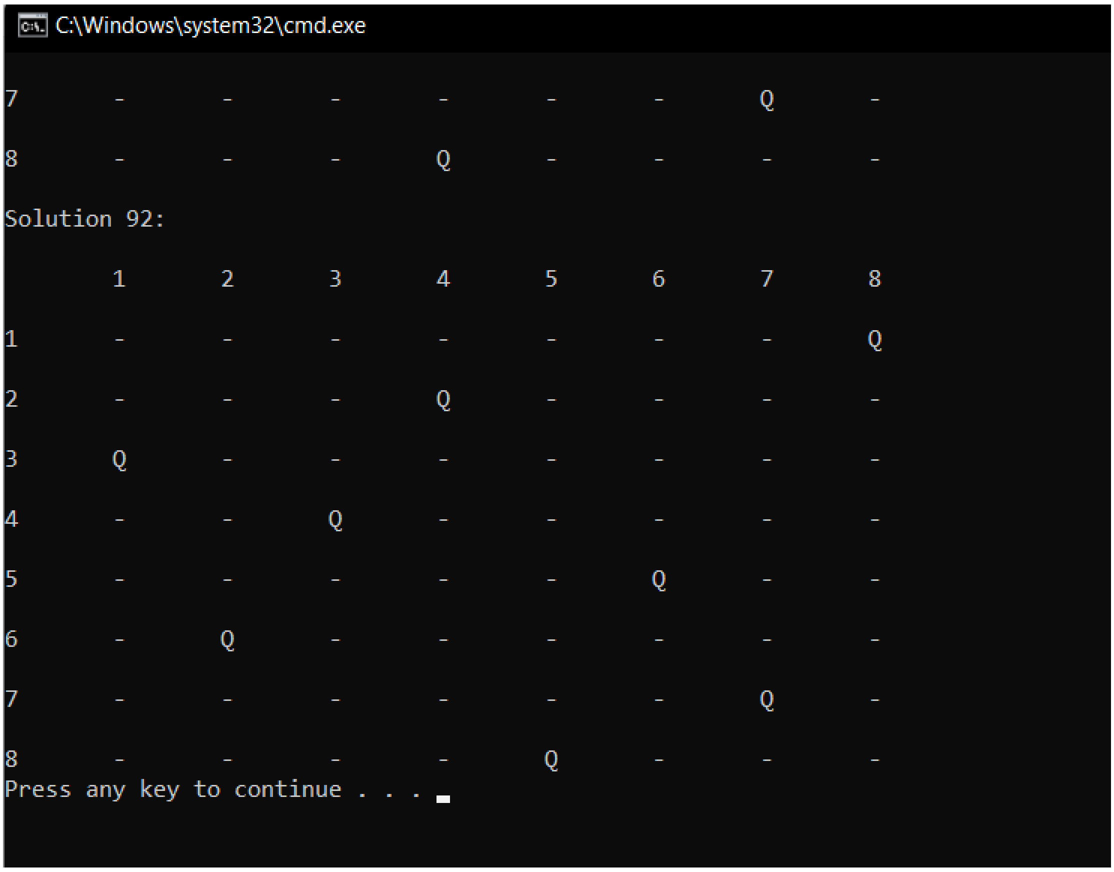


```c
#include <stdio.h>
#include <math.h>

int board[20], count;
int main()
{
    int n, i, j;
    void queen(int row, int n);

    printf(" - N Queens Problem Using Backtracking -");
    printf("\n\nEnter number of Queens:");
    scanf("%d", &n);
    queen(1, n);
    return 0;
}

void print(int n)
{
    int i, j;
    printf("\n\nSolution %d:\n\n", ++count);
    for (i = 1; i <= n; ++i)
        printf("\t%d", i);
    for (i = 1; i <= n; ++i)
    {
        printf("\n\n%d", i);
        for (j = 1; j <= n; ++j) // for nxn board
        {
            if (board[i] == j)
                printf("\tQ"); // queen at i,j position
            else
                printf("\t-"); // empty slot
        }
    }
}


int place(int row, int column)
{
    int i;
    for (i = 1; i <= row - 1; ++i)
```



```
int place(int row, int column)
{
    int i;
    for (i = 1; i <= row - 1; ++i)
    {
        if (board[i] == column)
            return 0;
        else if (abs(board[i] - column) == abs(i - row))
            return 0;
    }

    return 1; // no conflicts
}

void queen(int row, int n)
{
    int column;
    for (column = 1; column <= n; ++column)
    {
        if (place(row, column))
        {
            board[row] = column;
            if (row == n)
                print(n);
            else
                queen(row + 1, n);
        }
    }
}
```

Output:

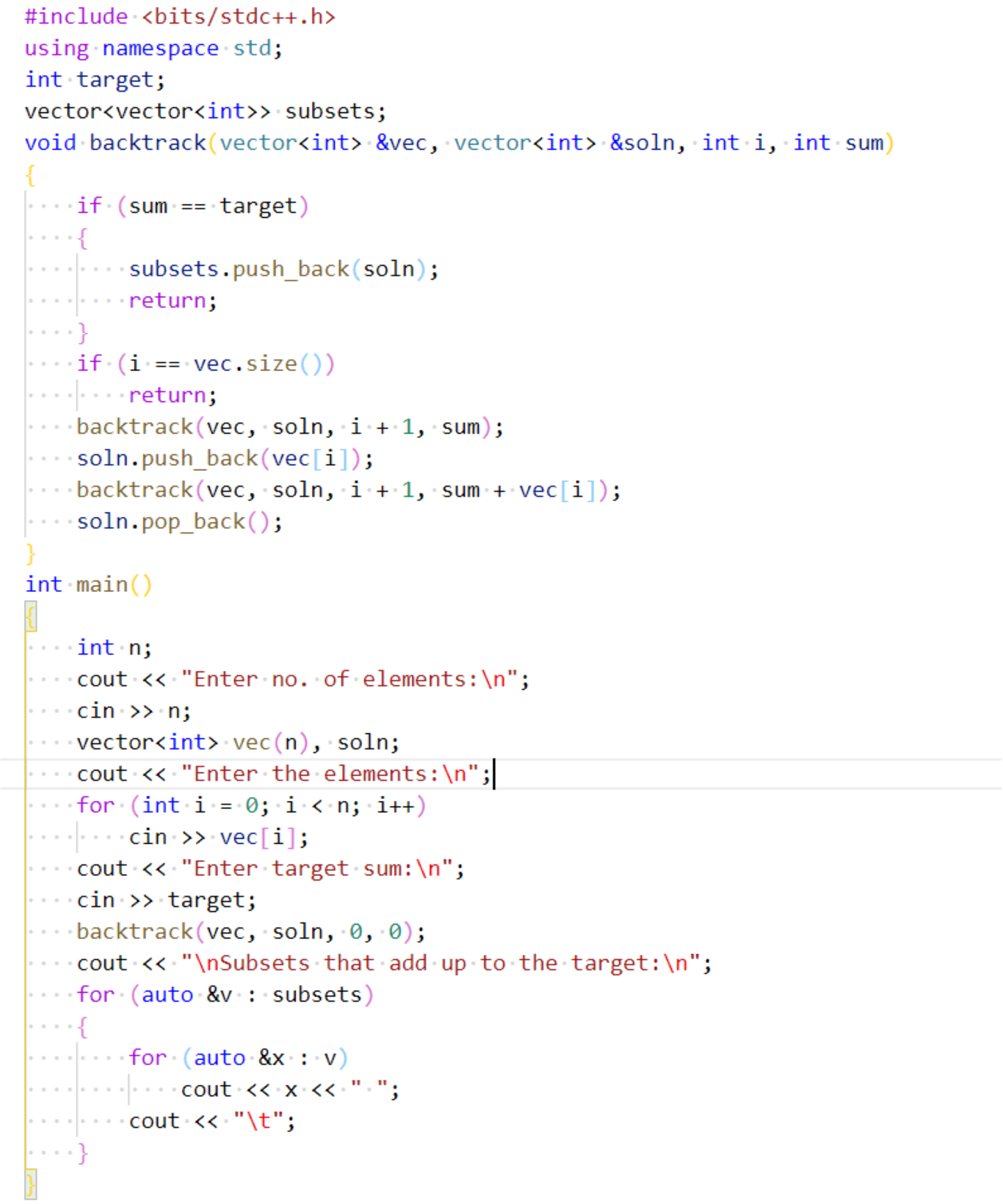


```
7       -       -       -       -       -       -       Q       -

8       -       -       -       Q       -       -       -       -

Solution 92:

        1       2       3       4       5       6       7       8

1       -       -       -       -       -       -       -       Q

2       -       -       -       Q       -       -       -       -

3       Q       -       -       -       -       -       -       -

4       -       -       Q       -       -       -       -       -

5       -       -       -       -       -       Q       -       -

6       -       Q       -       -       -       -       -       -

7       -       -       -       -       -       -       Q       -

8       -       -       -       -       Q       -       -       -
Press any key to continue . . .
```

Practical – 14

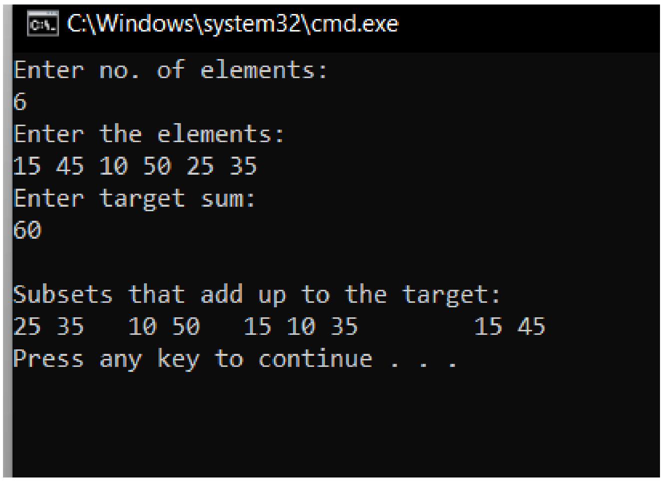
Aim: Write a program to solve subset sum problem using Backtracking.

Code:



```cpp
#include <bits/stdc++.h>
using namespace std;
int target;
vector<vector<int>> subsets;
void backtrack(vector<int> &vec, vector<int> &soln, int i, int sum)
{
    if (sum == target)
    {
        subsets.push_back(soln);
        return;
    }
    if (i == vec.size())
        return;
    backtrack(vec, soln, i + 1, sum);
    soln.push_back(vec[i]);
    backtrack(vec, soln, i + 1, sum + vec[i]);
    soln.pop_back();
}
int main()
{
    int n;
    cout << "Enter no. of elements:\n";
    cin >> n;
    vector<int> vec(n), soln;
    cout << "Enter the elements:\n";
    for (int i = 0; i < n; i++)
        cin >> vec[i];
    cout << "Enter target sum:\n";
    cin >> target;
    backtrack(vec, soln, 0, 0);
    cout << "\nSubsets that add up to the target:\n";
    for (auto &v : subsets)
    {
        for (auto &x : v)
            cout << x << " ";
        cout << "\t";
    }
}
```

Output:



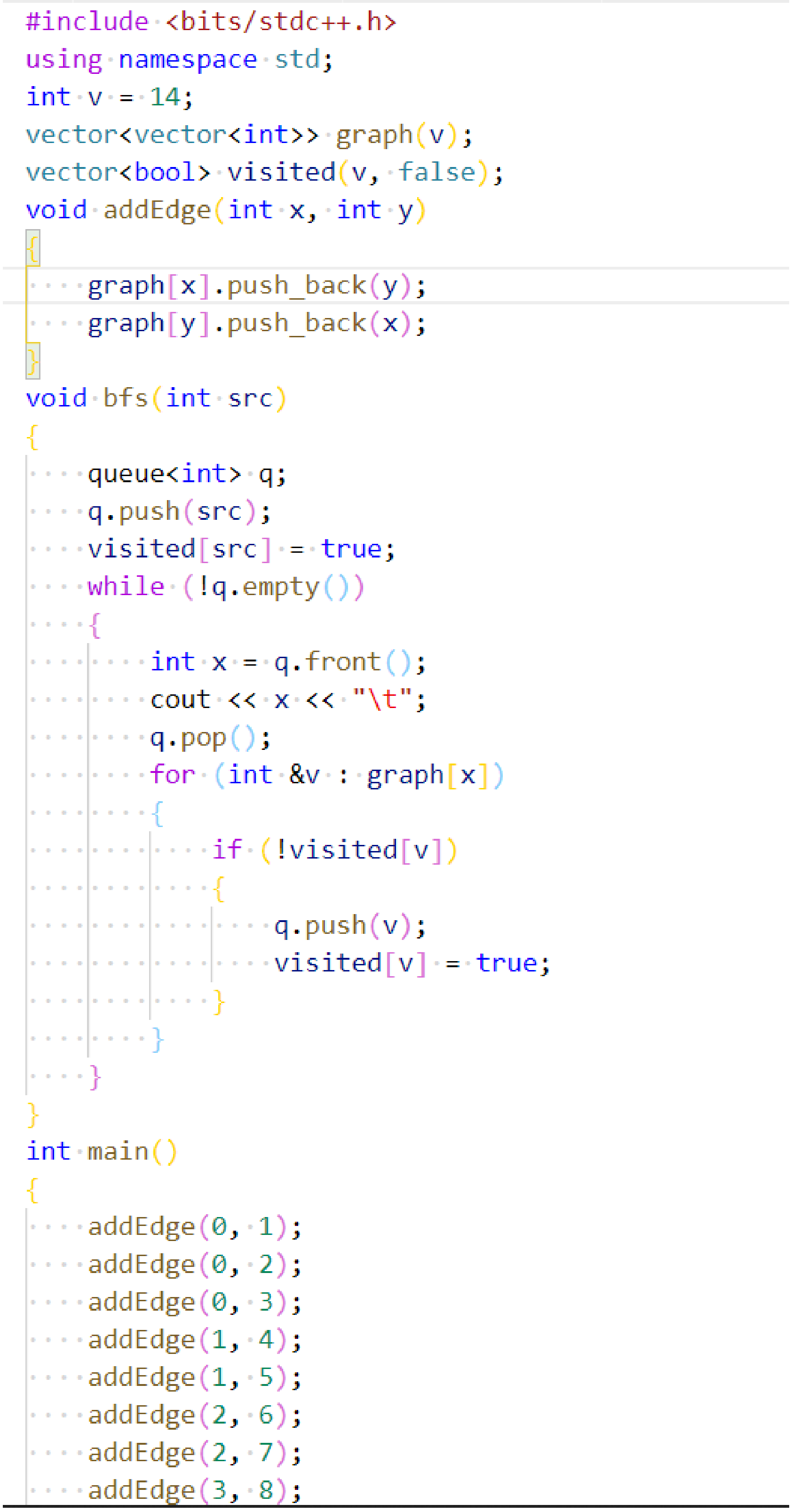
C:\Windows\system32\cmd.exe

```
Enter no. of elements:
6
Enter the elements:
15 45 10 50 25 35
Enter target sum:
60

Subsets that add up to the target:
25 35   10 50   15 10 35        15 45
Press any key to continue . . .
```

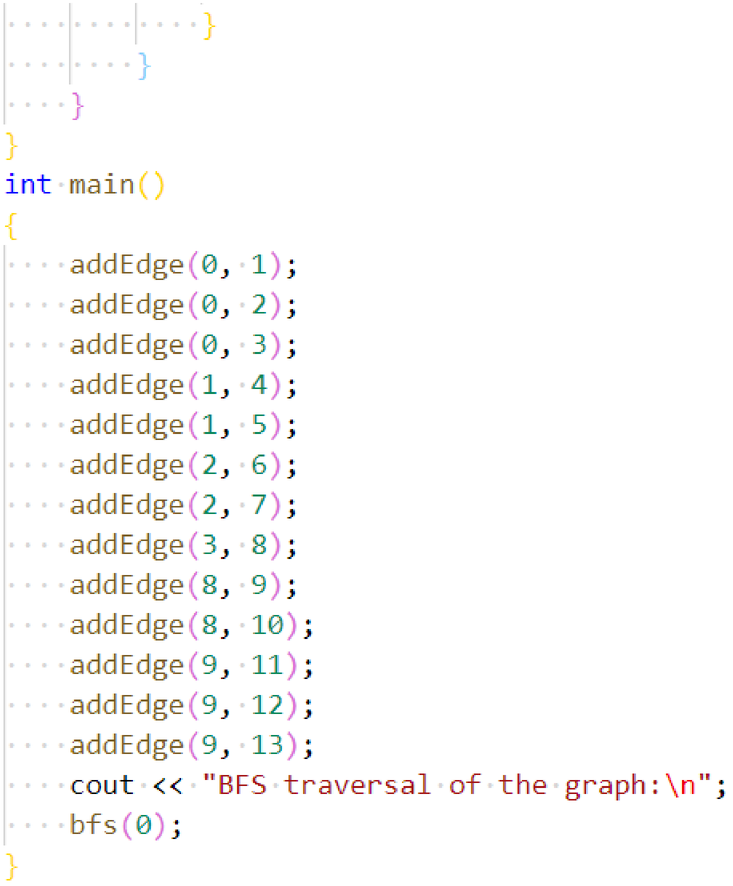
Practical – 15

Aim: Write a program to use a queue to store the node and mark it as 'visited' until all its neighbors (vertices that are directly connected to it) are marked. Implement by using bfs algorithm for a graph.

Code:

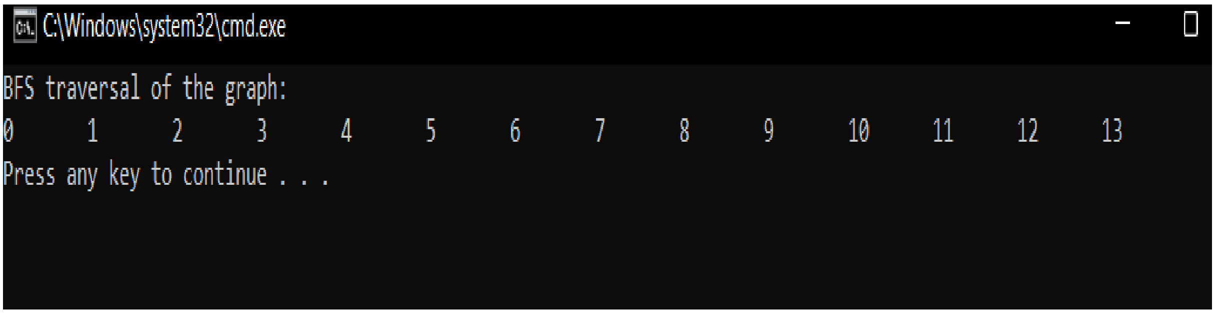


```cpp
#include <bits/stdc++.h>
using namespace std;
int v = 14;
vector<vector<int>> graph(v);
vector<bool> visited(v, false);
void addEdge(int x, int y)
{
    graph[x].push_back(y);
    graph[y].push_back(x);
}
void bfs(int src)
{
    queue<int> q;
    q.push(src);
    visited[src] = true;
    while (!q.empty())
    {
        int x = q.front();
        cout << x << "\t";
        q.pop();
        for (int &v : graph[x])
        {
            if (!visited[v])
            {
                q.push(v);
                visited[v] = true;
            }
        }
    }
}
int main()
{
    addEdge(0, 1);
    addEdge(0, 2);
    addEdge(0, 3);
    addEdge(1, 4);
    addEdge(1, 5);
    addEdge(2, 6);
    addEdge(2, 7);
    addEdge(3, 8);
```



```
            }
        }
    }
}
int main()
{
    addEdge(0, 1);
    addEdge(0, 2);
    addEdge(0, 3);
    addEdge(1, 4);
    addEdge(1, 5);
    addEdge(2, 6);
    addEdge(2, 7);
    addEdge(3, 8);
    addEdge(8, 9);
    addEdge(8, 10);
    addEdge(9, 11);
    addEdge(9, 12);
    addEdge(9, 13);
    cout << "BFS traversal of the graph:\n";
    bfs(0);
}
```

Output:

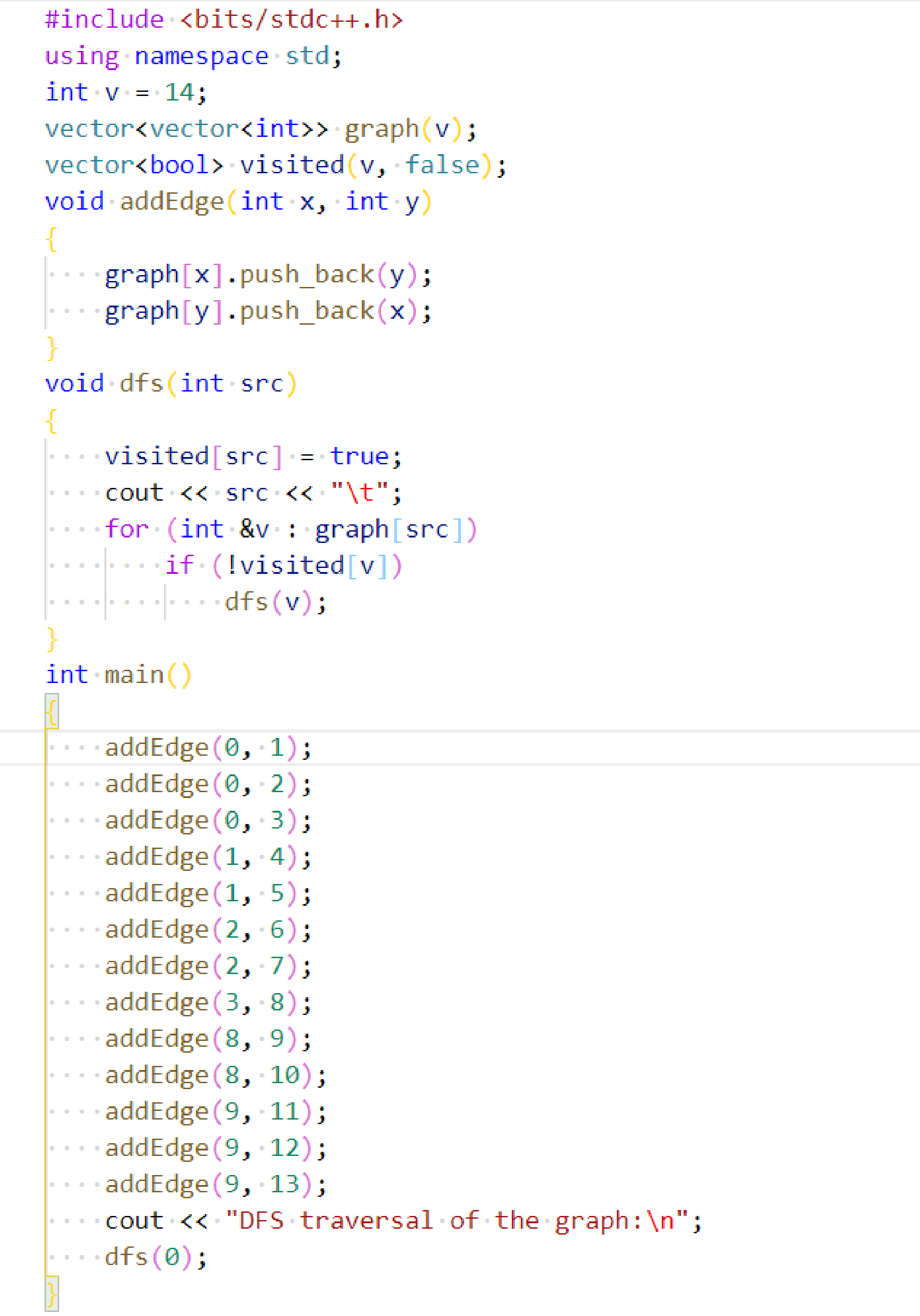


```
BFS traversal of the graph:
0       1       2       3       4       5       6       7       8       9       10      11      12      13
Press any key to continue . . .
```

Practical – 16

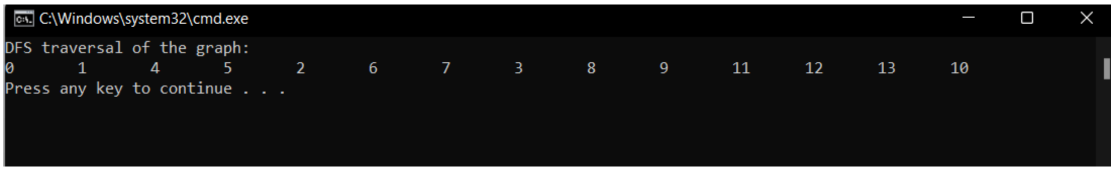
Aim: Write a program to implement the dfs algorithm for a graph.

Code:



```cpp
#include <bits/stdc++.h>
using namespace std;
int v = 14;
vector<vector<int>> graph(v);
vector<bool> visited(v, false);
void addEdge(int x, int y)
{
    graph[x].push_back(y);
    graph[y].push_back(x);
}
void dfs(int src)
{
    visited[src] = true;
    cout << src << "\t";
    for (int &v : graph[src])
        if (!visited[v])
            dfs(v);
}
int main()
{
    addEdge(0, 1);
    addEdge(0, 2);
    addEdge(0, 3);
    addEdge(1, 4);
    addEdge(1, 5);
    addEdge(2, 6);
    addEdge(2, 7);
    addEdge(3, 8);
    addEdge(8, 9);
    addEdge(8, 10);
    addEdge(9, 11);
    addEdge(9, 12);
    addEdge(9, 13);
    cout << "DFS traversal of the graph:\n";
    dfs(0);
}
```

Output:

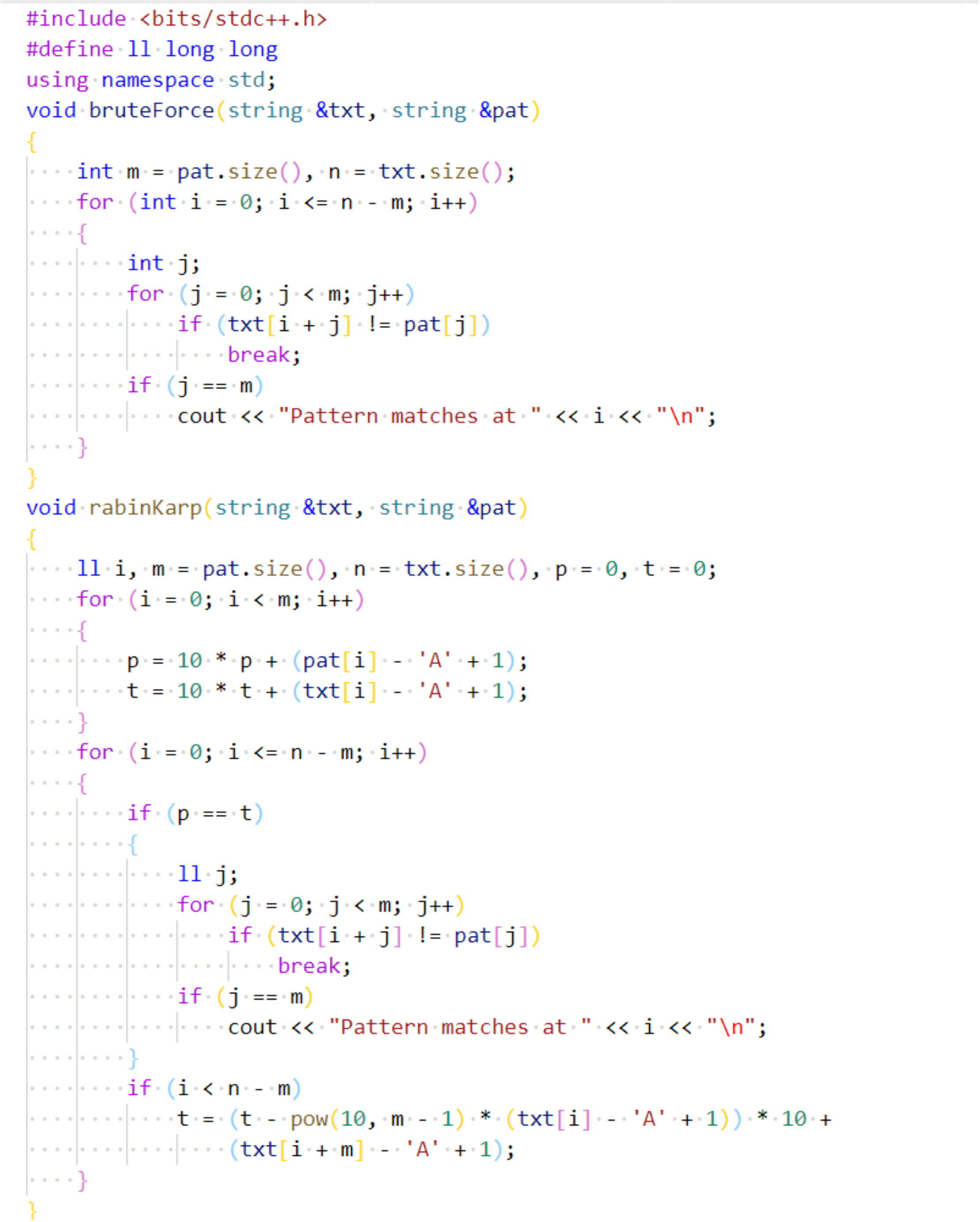


```
DFS traversal of the graph:
0       1       4       5       2       6       7       3       8       9       11      12      13      10
Press any key to continue . . .
```

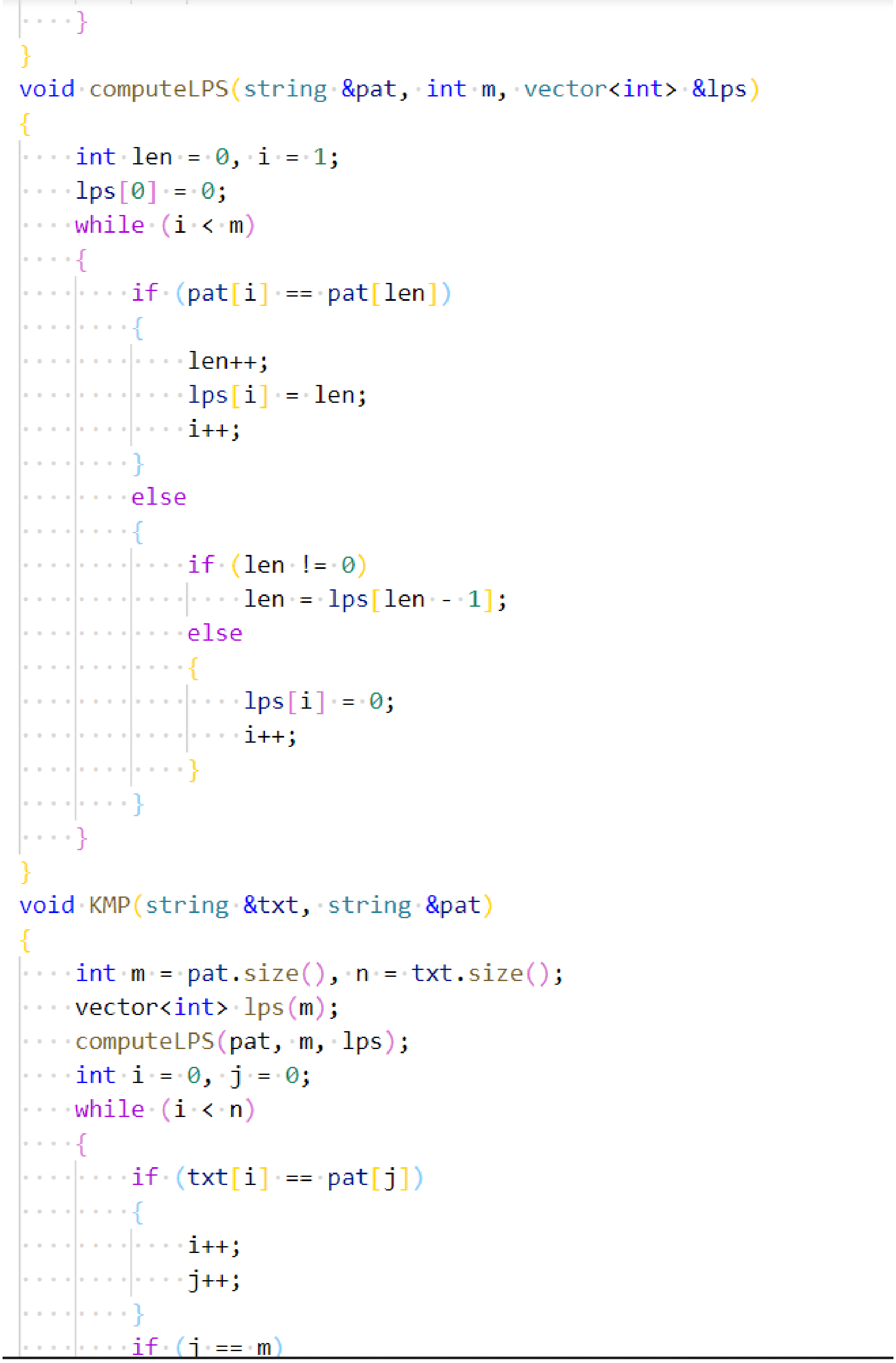
Practical – 17

Aim: Write a program to match the pattern by using Brute Force algorithm, Rabin-Karp algorithm, KMP algorithm and Boyer-Moore algorithm.

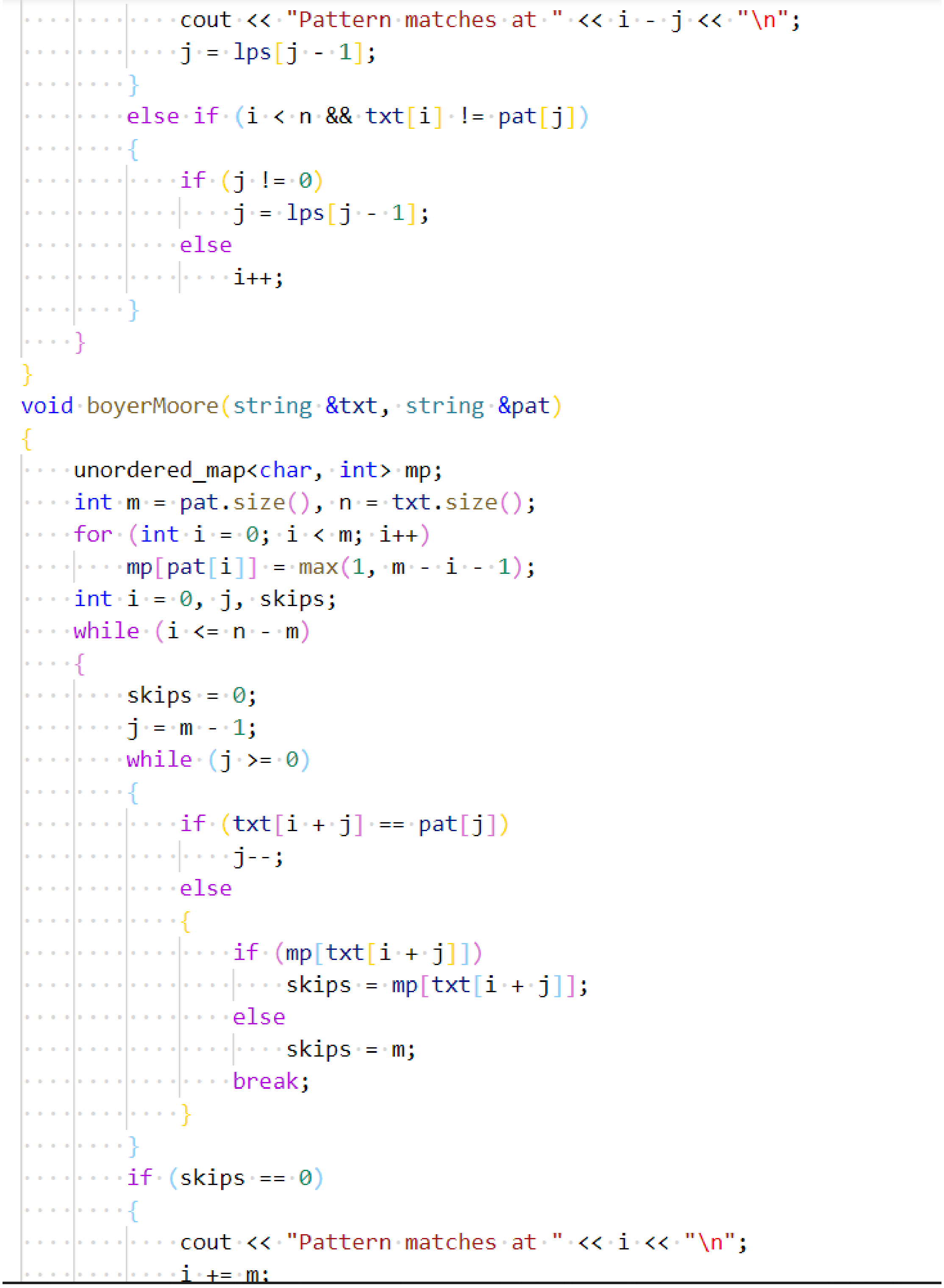
Code:



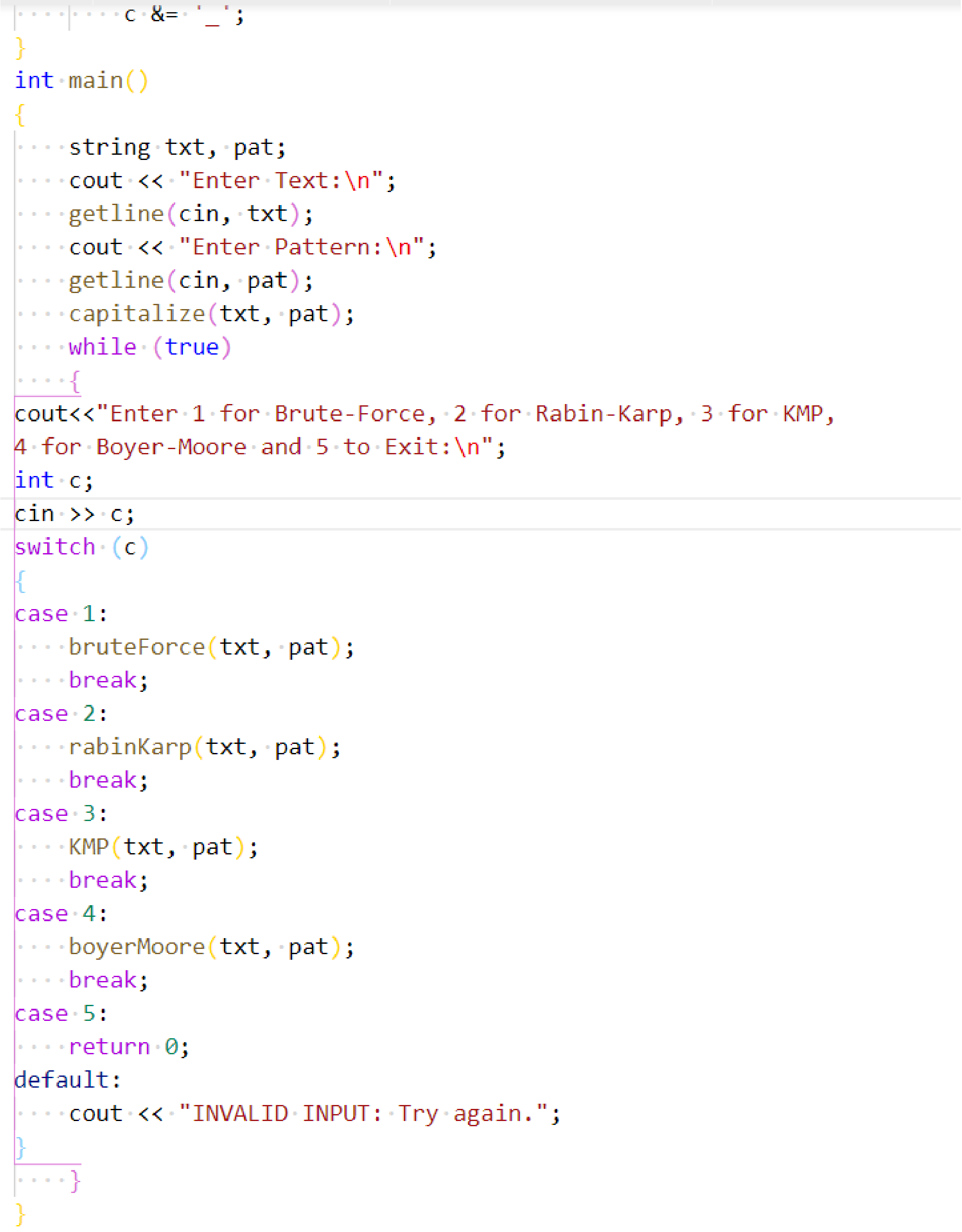
```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
void bruteForce(string &txt, string &pat)
{
    int m = pat.size(), n = txt.size();
    for (int i = 0; i <= n - m; i++)
    {
        int j;
        for (j = 0; j < m; j++)
            if (txt[i + j] != pat[j])
                break;
        if (j == m)
            cout << "Pattern matches at " << i << "\n";
    }
}
void rabinKarp(string &txt, string &pat)
{
    ll i, m = pat.size(), n = txt.size(), p = 0, t = 0;
    for (i = 0; i < m; i++)
    {
        p = 10 * p + (pat[i] - 'A' + 1);
        t = 10 * t + (txt[i] - 'A' + 1);
    }
    for (i = 0; i <= n - m; i++)
    {
        if (p == t)
        {
            ll j;
            for (j = 0; j < m; j++)
                if (txt[i + j] != pat[j])
                    break;
            if (j == m)
                cout << "Pattern matches at " << i << "\n";
        }
        if (i < n - m)
            t = (t - pow(10, m - 1) * (txt[i] - 'A' + 1)) * 10 +
                (txt[i + m] - 'A' + 1);
    }
}
```



```
    }
}
void computeLPS(string &pat, int m, vector<int> &lps)
{
    int len = 0, i = 1;
    lps[0] = 0;
    while (i < m)
    {
        if (pat[i] == pat[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {
            if (len != 0)
                len = lps[len - 1];
            else
            {
                lps[i] = 0;
                i++;
            }
        }
    }
}
void KMP(string &txt, string &pat)
{
    int m = pat.size(), n = txt.size();
    vector<int> lps(m);
    computeLPS(pat, m, lps);
    int i = 0, j = 0;
    while (i < n)
    {
        if (txt[i] == pat[j])
        {
            i++;
            j++;
        }
        if (j == m)
```

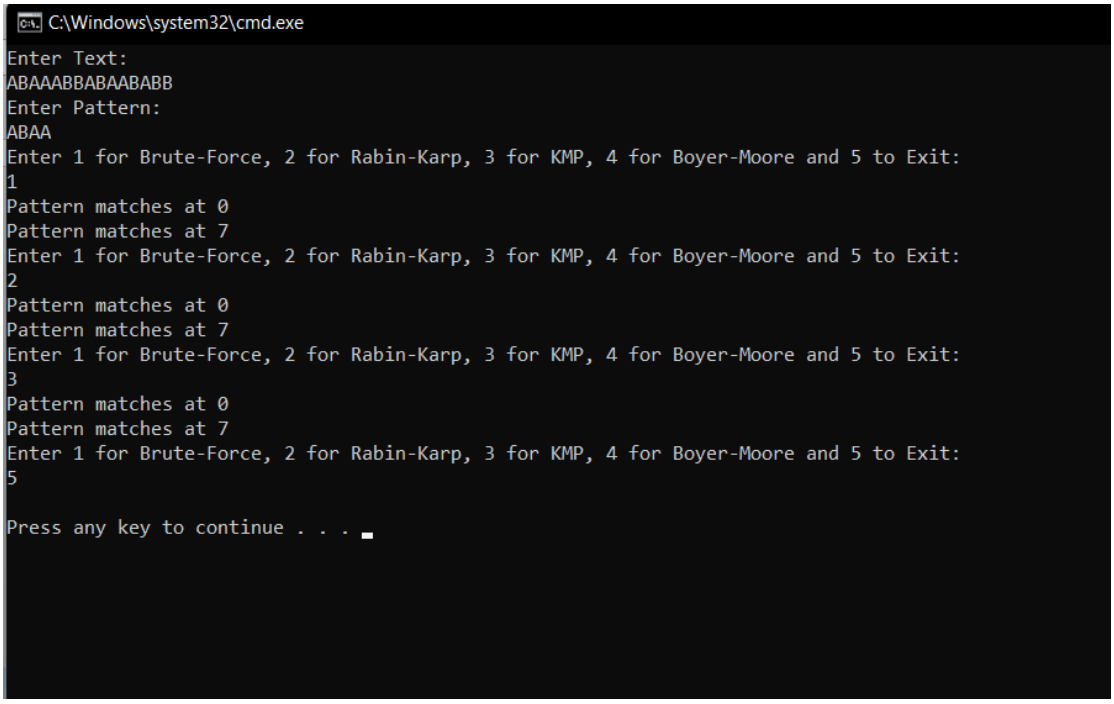


```
            cout << "Pattern matches at " << i - j << "\n";
            j = lps[j - 1];
        }
        else if (i < n && txt[i] != pat[j])
        {
            if (j != 0)
                j = lps[j - 1];
            else
                i++;
        }
    }
}
void boyerMoore(string &txt, string &pat)
{
    unordered_map<char, int> mp;
    int m = pat.size(), n = txt.size();
    for (int i = 0; i < m; i++)
        mp[pat[i]] = max(1, m - i - 1);
    int i = 0, j, skips;
    while (i <= n - m)
    {
        skips = 0;
        j = m - 1;
        while (j >= 0)
        {
            if (txt[i + j] == pat[j])
                j--;
            else
            {
                if (mp[txt[i + j]])
                    skips = mp[txt[i + j]];
                else
                    skips = m;
                break;
            }
        }
        if (skips == 0)
        {
            cout << "Pattern matches at " << i << "\n";
            i += m;
```



```
        c &= '_';
}
int main()
{
    string txt, pat;
    cout << "Enter Text:\n";
    getline(cin, txt);
    cout << "Enter Pattern:\n";
    getline(cin, pat);
    capitalize(txt, pat);
    while (true)
    {
cout<<"Enter 1 for Brute-Force, 2 for Rabin-Karp, 3 for KMP,
4 for Boyer-Moore and 5 to Exit:\n";
int c;
cin >> c;
switch (c)
{
case 1:
    bruteForce(txt, pat);
    break;
case 2:
    rabinKarp(txt, pat);
    break;
case 3:
    KMP(txt, pat);
    break;
case 4:
    boyerMoore(txt, pat);
    break;
case 5:
    return 0;
default:
    cout << "INVALID INPUT: Try again.";
}
    }
}
```

Output:



C:\Windows\system32\cmd.exe

```
Enter Text:
ABAAABBABAABABB
Enter Pattern:
ABAA
Enter 1 for Brute-Force, 2 for Rabin-Karp, 3 for KMP, 4 for Boyer-Moore and 5 to Exit:
1
Pattern matches at 0
Pattern matches at 7
Enter 1 for Brute-Force, 2 for Rabin-Karp, 3 for KMP, 4 for Boyer-Moore and 5 to Exit:
2
Pattern matches at 0
Pattern matches at 7
Enter 1 for Brute-Force, 2 for Rabin-Karp, 3 for KMP, 4 for Boyer-Moore and 5 to Exit:
3
Pattern matches at 0
Pattern matches at 7
Enter 1 for Brute-Force, 2 for Rabin-Karp, 3 for KMP, 4 for Boyer-Moore and 5 to Exit:
5

Press any key to continue . . .
```