**Assignment No 3**

**Code**

Selection Sort :

#include <iostream>

#include <vector>

using namespace std;

// Function to perform Selection Sort

void selectionSort(vector<int> &arr) {

    int n = arr.size();

    for (int i = 0; i < n - 1; i++) {

        int minIndex = i;

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

            if (arr[j] < arr[minIndex]) {

                minIndex = j;

            }

        }

        swap(arr[i], arr[minIndex]);

    }

}

// Function to print an array

void printArray(const vector<int> &arr) {

    for (int num : arr) {

        cout << num << " ";

    }

    cout << endl;

}

int main() {

    vector<int> arr = {64, 25, 12, 22, 11};

    cout << "Original array: ";

    printArray(arr);

    selectionSort(arr);

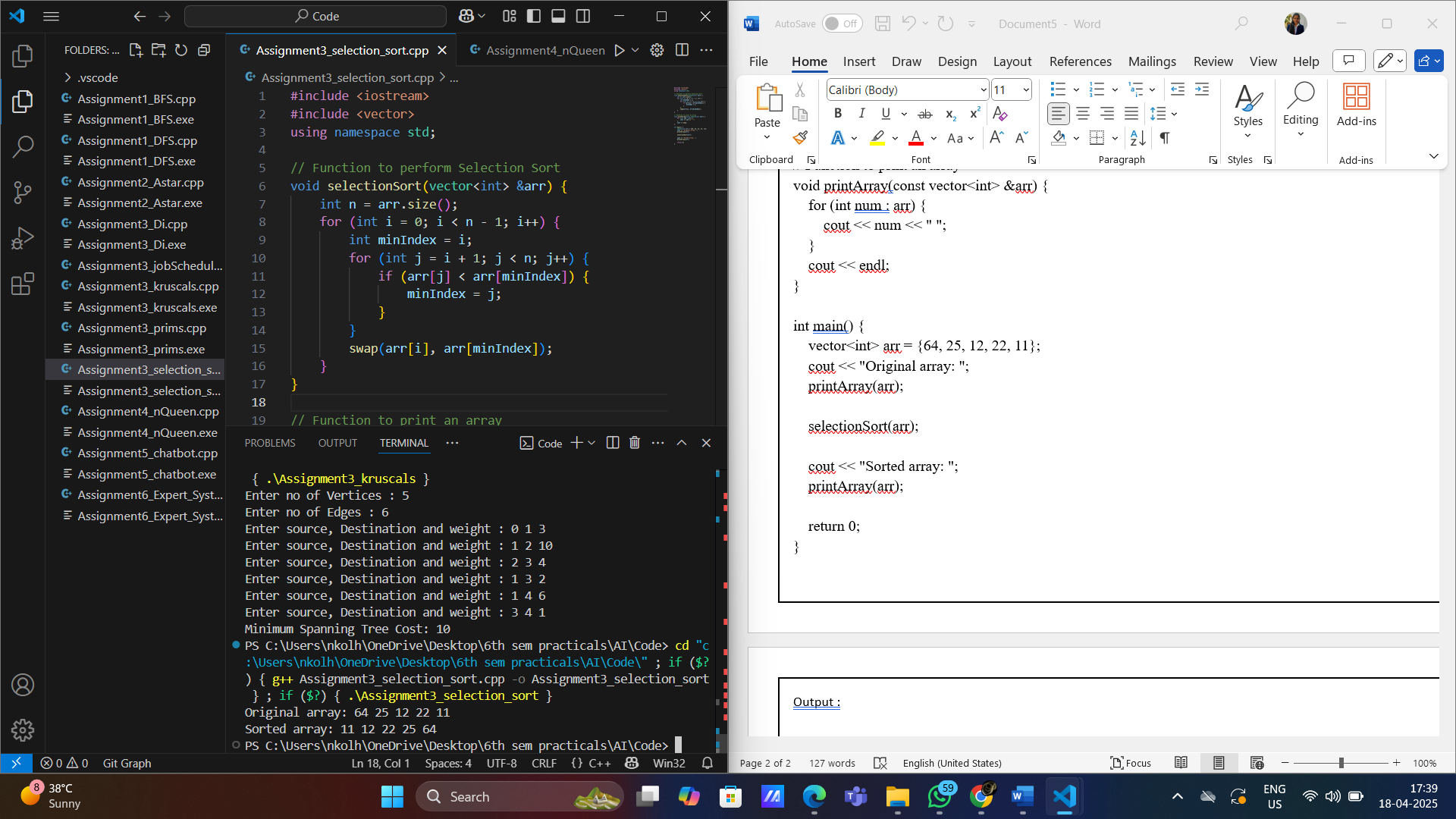
    cout << "Sorted array: ";

    printArray(arr);

    return 0;

}

Output :



Prim's Minimal Spanning Tree Algorithm :

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

int main(){

    int v=0, e=0;

    cout<<"Enter no of Vertices : ";

    cin>>v;

    cout<<"Enter no of Edges : ";

    cin>>e;

    vector<vector<pair<int,int>>> adj(v);

    for (int i = 0; i < e; i++){

        int s,d,w;

        cout<<"Enter source, Destination and weight : ";

        cin>>s>>d>>w;

        adj[s].push\_back(make\_pair(d, w));

        adj[d].push\_back(make\_pair(s,w));

    }

    vector<int> key(v);

    vector<bool> mst(v);

    vector<int> parent(v);

    for (int  i = 0; i < v; i++){

        key[i]=INT\_MAX;

        mst[i]=false;

        parent[i]=-1;

    }

    key[0]=0;

    parent[0]=-1;

    for (int i = 0; i < v; i++){

        //find minimum from key

        int mini=INT\_MAX;

        int u;

        for (int j = 0; j < v; j++){

            if(mst[j]==false && key[j]<mini){

                mini=key[j];

                u=j;

            }

        }

        //set mst[u]=true

        mst[u]=true;

        //explore adjacent nodes

        for (auto it: adj[u]){

            int v=it.first;

            int w=it.second;

            if(mst[v]==false && w<key[v]){

                key[v]=w;

                parent[v]=u;

            }

        }

    }

    cout << "Edges in MST:\n";

    for (int i = 1; i < v; i++) {

        cout << parent[i] << " - " << i << " with weight " << key[i] << endl;

    }

    int minCost = 0;

    for (int i = 0; i < v; i++) {

        if (key[i] != INT\_MAX)

            minCost += key[i];

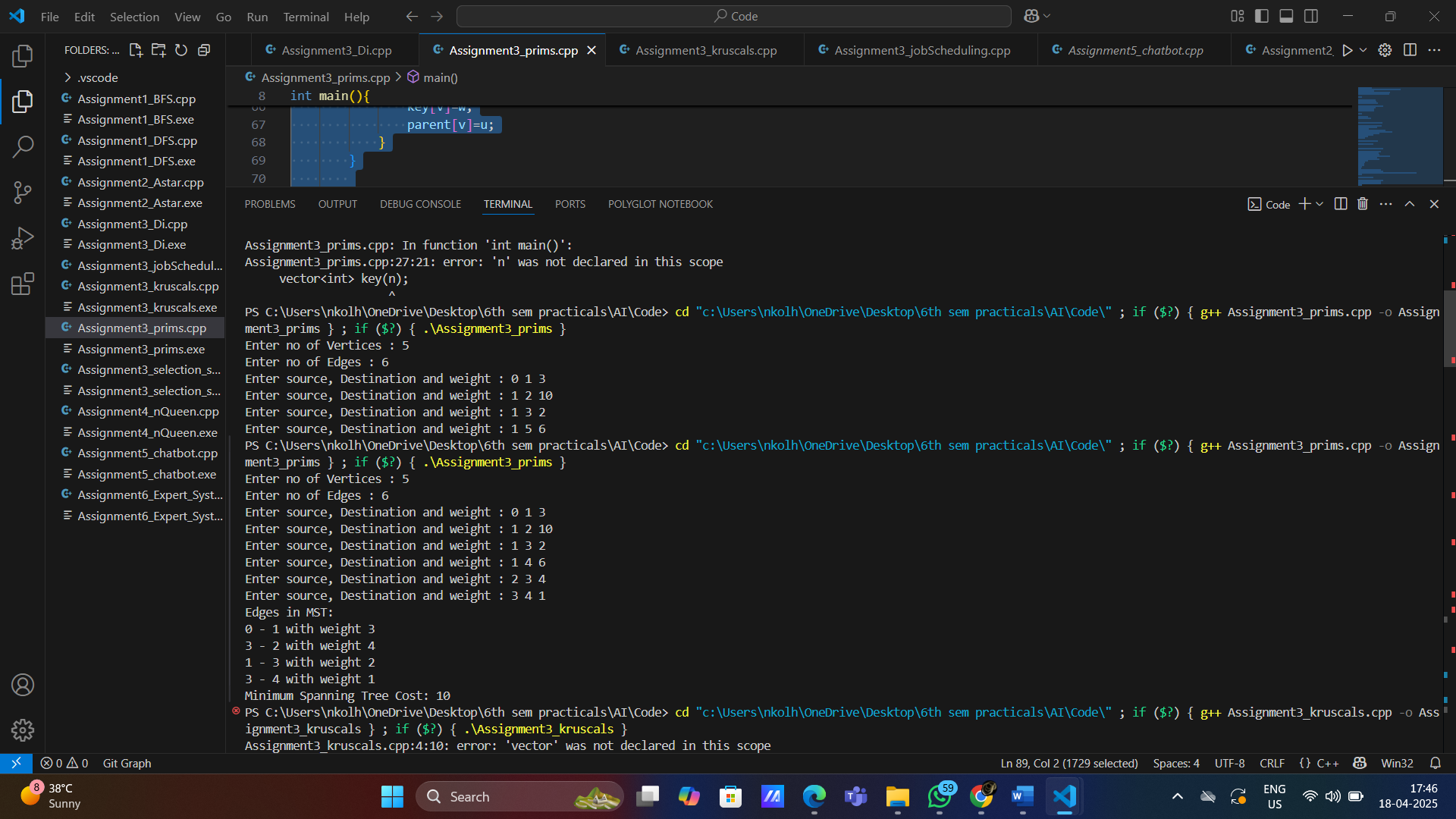
    }

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

    return 0;

}

Output :



Kruskal's Minimal Spanning Tree Algorithm

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

bool cmp(vector<int> &a,vector<int> &b){

    return a[2]<b[2];

}

void makeSet(vector<int> &parent,vector<int> &rank,int v){

    for (int i = 0; i <v ; i++)

    {

        parent[i]=i;

        rank[i]=0;

    }

}

int findParent(int node,vector<int> &parent){

    if(parent[node]==node){

        return node;

    }

    return parent[node]=findParent(parent[node],parent);

}

void unionSet(int u,int v,vector<int> &parent,vector<int> &rank){

    u=findParent(u,parent);

    v=findParent(v,parent);

    if(rank[u]>rank[v]){

        parent[v]=u;

    }else if(rank[v]>rank[u]){

        parent[u]=v;

    }else{

        parent[v]=u;

        rank[u]++;

    }

}

int main(){

    int v = 0, e = 0;

    cout << "Enter no of Vertices : ";

    cin >> v;

    cout << "Enter no of Edges : ";

    cin >> e;

    vector<vector<int>> adj;

    for (int i = 0; i < e; i++) {

        int s, d, w;

        cout << "Enter source, Destination and weight : ";

        cin >> s >> d >> w;

        adj.push\_back({s, d, w});

    }

    sort(adj.begin(), adj.end(), cmp);

    vector<int> parent(v);

    vector<int> rank(v);

    makeSet(parent, rank, v);

    int minCost = 0;

    for (int i = 0; i < e; i++) {

        int u = adj[i][0];

        int v = adj[i][1];

        int w = adj[i][2];

        if (findParent(u, parent) != findParent(v, parent)) {

            minCost += w;

            unionSet(u, v, parent, rank);

        }

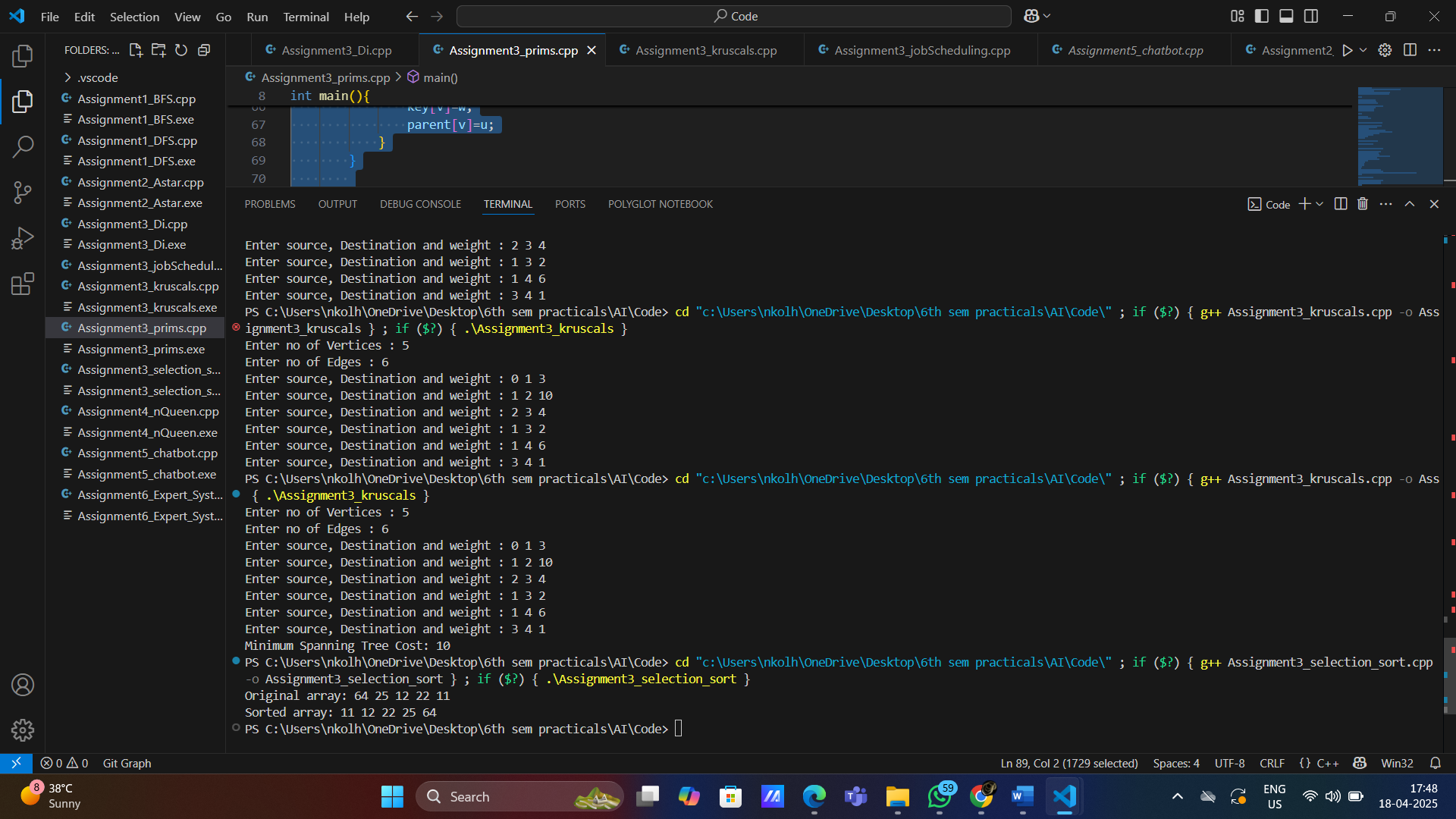
    }

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

    return 0;

}

Output :



Dijkstra's Algorithm :

#include <iostream>

#include <vector>

#include <set>

#include <climits>

using namespace std;

int main(){

    int v=0, e=0;

    cout<<"Enter no of Vertices : ";

    cin>>v;

    cout<<"Enter no of Edges : ";

    cin>>e;

    vector<vector<pair<int,int>>> adj(v);

    for (int i = 0; i < e; i++)

    {

        int s,d,w;

        cout<<"Enter source, Destination and weight : ";

        cin>>s>>d>>w;

        adj[s].push\_back(make\_pair(d, w));

        adj[d].push\_back(make\_pair(s,w));

    }

    vector<int> dist(v);

    for (int i = 0; i < v; i++)

    {

        dist[i]=INT\_MAX;

    }

    set<pair<int,int>> st;

    dist[0]=0;

    st.insert(make\_pair(0,0));

    while (!st.empty())

    {

        pair<int,int> topNode = \*st.begin();

        int distance=topNode.first;

        int node=topNode.second;

        st.erase(st.begin());

        for (auto neighbor : adj[topNode.second])

        {

            if(distance+neighbor.second<dist[neighbor.first]){

                auto record=st.find(make\_pair(dist[neighbor.first],neighbor.first));

                if(record!=st.end()){

                    st.erase(record);

                }

                dist[neighbor.first]=distance+neighbor.second;

                st.insert(make\_pair(dist[neighbor.first],neighbor.first));

            }

        }

    }

    for (int i = 0; i < v; i++)

    {

        cout<<dist[i]<<"  ";

    }

    cout<<endl;

    return 0;

}

Output :

