Name: Mark Gerald Guerrero
Course: DcpET
Section: 2-3
Description: Dijkstra’s Algorithm

This is a simple program that applies Dijkstra's algorithm in C++.

In my code, I have four functions. The first one is add_edge(), which takes four arguments: the first is the vector array of type pair, which will store the edges, their weights, and their neighboring vertices. This stores it both ways, as the graph I used for this sample is undirected.

The second function is print_adj_list(). It takes two arguments: the vector array of our vertices, edges, and nodes, and the number of vertices denoted by V. This function prints the adjacency list or the data structure representation of our graph. I opted for an adjacency list instead of an adjacency matrix because our example is a sparse graph. An adjacency matrix would work better on denser graphs. I thought using a matrix would be overkill, as I wouldn’t have time to study it alongside the min-heap data structure and the algorithm itself.

The third function is print_min_heap(), which takes one argument, the min-heap. This just prints the min-heap or the priority queue for storing the current shortest neighboring vertices. I multiply the weight by -1 to make it a min-heap, as by default, a priority queue in C++ is a max-heap. (There are other ways to make it a min-heap using constructors and advanced C++ techniques, but I’m more familiar with this approach as it is straightforward.)

The fourth function is the algorithm itself, named dijkstra(). It takes three arguments: the adjacency list, the origin node denoted by its index in the array, and V, the number of vertices. This is a simple Dijkstra implementation that updates the distance vector if a cheaper path is found.

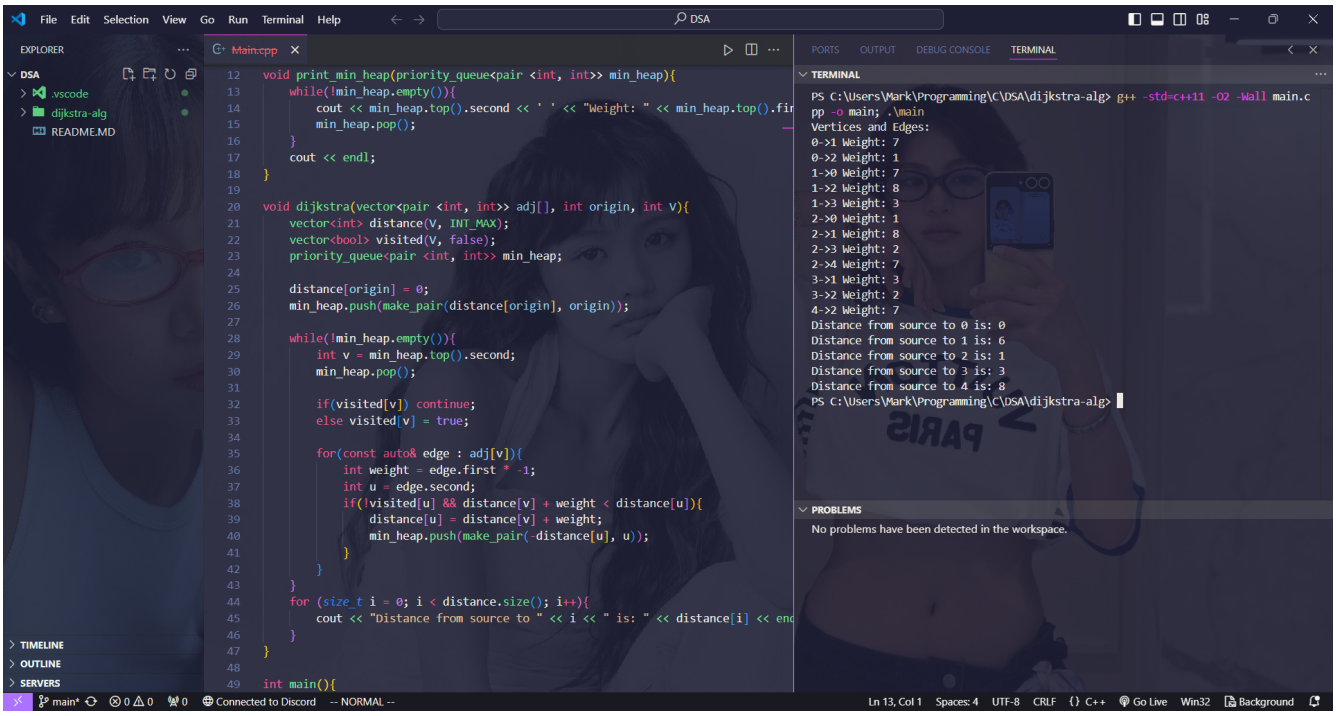
Lastly, the main method contains the initialization of our adjacency list using the add_edge() function and the call to the algorithm itself.

Source Code: https://github.com/oshit0/dsa/blob/main/dijkstra-alg/Main.cpp

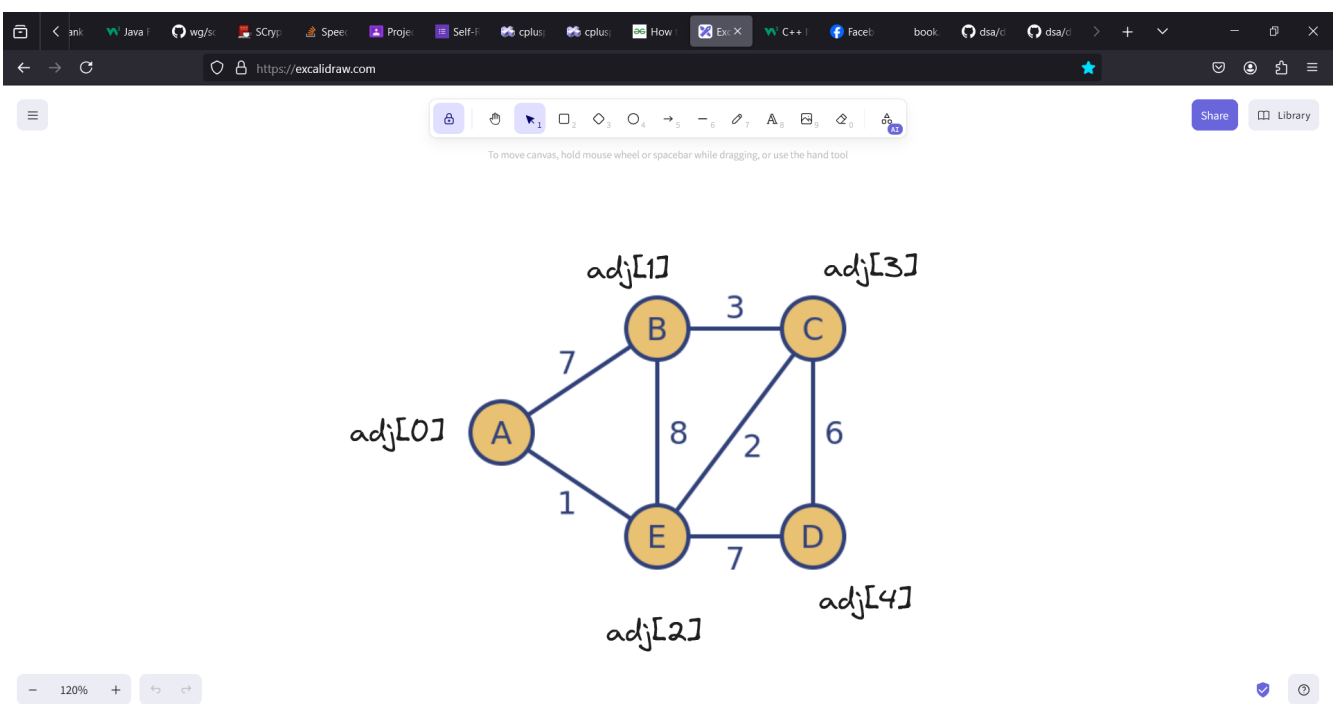
```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <climits>

#define endl '\n'

using namespace std;

void add_edge(vector<pair <int, int>> adj[], int v, int u, int wt){
    adj[v].push_back(make_pair(wt * -1, u));
    adj[u].push_back(make_pair(wt * -1, v));
}

void print_adj_list(const vector<pair <int, int>> adj[], int V){
    cout << "Vertices and Edges:" << endl;
    for(int v = 0; v < V; ++v){
        for(auto it = adj[v].begin(); it != adj[v].end(); ++it){
            cout << v << "->";
            cout << it->second << ' ' << "Weight: " << it->first * -1 << endl;
        }
    }
}

void print_min_heap(priority_queue<pair <int, int>> min_heap){
    while(!min_heap.empty()){
        cout << min_heap.top().second << ' ' << "Weight: " << min_heap.top().first * -1 << endl;
        min_heap.pop();
    }
    cout << endl;
}

void dijkstra(vector<pair <int, int>> adj[], int origin, int V){
    vector<int> distance(V, INT_MAX);
    vector<bool> visited(V, false);
    priority_queue<pair <int, int>> min_heap;

    distance[origin] = 0;
    min_heap.push(make_pair(distance[origin], origin));

    while(!min_heap.empty()){
        int v = min_heap.top().second;
        min_heap.pop();

        if(visited[v]) continue;
        else visited[v] = true;

        for(const auto& edge : adj[v]){
            int weight = edge.first * -1;
            int u = edge.second;
            if(!visited[u] && distance[v] + weight < distance[u]){
                distance[u] = distance[v] + weight;
                min_heap.push(make_pair(-distance[u], u));
            }
        }
    }
    for (size_t i = 0; i < distance.size(); i++){
        cout << "Distance from source to " << i << " is: " << distance[i] << endl;
    }
}

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    //Vertices
    const int V = 5;
    //Adjacency List
    vector<pair <int, int>> adj[V];

    //Init
    add_edge(adj, 0, 1, 7);
    add_edge(adj, 0, 2, 1);
    add_edge(adj, 1, 2, 8);
    add_edge(adj, 1, 3, 3);
    add_edge(adj, 2, 3, 2);
    add_edge(adj, 2, 4, 7);
    print_adj_list(adj, V);
    dijkstra(adj, 0, V);


    return 0;
}
```

Screenshots:

Text Editor:



Graph I Used:



Output:

```
TERMINAL
PS C:\Users\Mark\Programming\C\DSA\dijkstra-alg> g++ -std=c++11 -O2 -Wall main.c
pp -o main; .\main
Vertices and Edges:
0->1 Weight: 7
0->2 Weight: 1
1->0 Weight: 7
1->2 Weight: 8
1->3 Weight: 3
2->0 Weight: 1
2->1 Weight: 8
2->3 Weight: 2
2->4 Weight: 7
3->1 Weight: 3
3->2 Weight: 2
4->2 Weight: 7
Distance from source to 0 is: 0
Distance from source to 1 is: 6
Distance from source to 2 is: 1
Distance from source to 3 is: 3
Distance from source to 4 is: 8
PS C:\Users\Mark\Programming\C\DSA\dijkstra-alg>
```