Final Project Report: Dijkstra's Algorithm

Dayne Guy
Joao Luiz Dal Cortivo
Nhi Nguyen
Thy Tran

Introduction

Graphs are a fundamental data structure in computer science. It models relationships between objects where each connection has a significant meaning. They have vast applications across various fields, from social network analysis to the design of computer networks and beyond. In essence, a graph G is composed of a set of vertices V (nodes) and a set of edges E (links between nodes), often represented as G(V, E). In a weighted graph, edges carry weights, signifying the cost or distance between two vertices. This additional complexity leads to various applications, one of which is finding the shortest path between two nodes — a problem with real-world implications such as route planning and network traffic optimization.

The solution to the problem of how to find the shortest path from one vertex to another vertex of a weighted graph is called Dijkstra's Algorithm. In other words, Dijkstra's algorithm is when the greedy method is applied to the shortest path problem. (Goodrich et al., 2011) In order to implement Dijkstra's Algorithm, we needed to implement a weighted Graph ADT and also learn more about Dijkstra's Algorithm. To gain a better understanding of Dijkstra's Algorithm we researched it and found many different ways to represent it. For example, if we suppose that the weighted graph is actually a map, the vertices are cities, and the edges are roads we would

end up with something similar to what is represented in Figure 1.

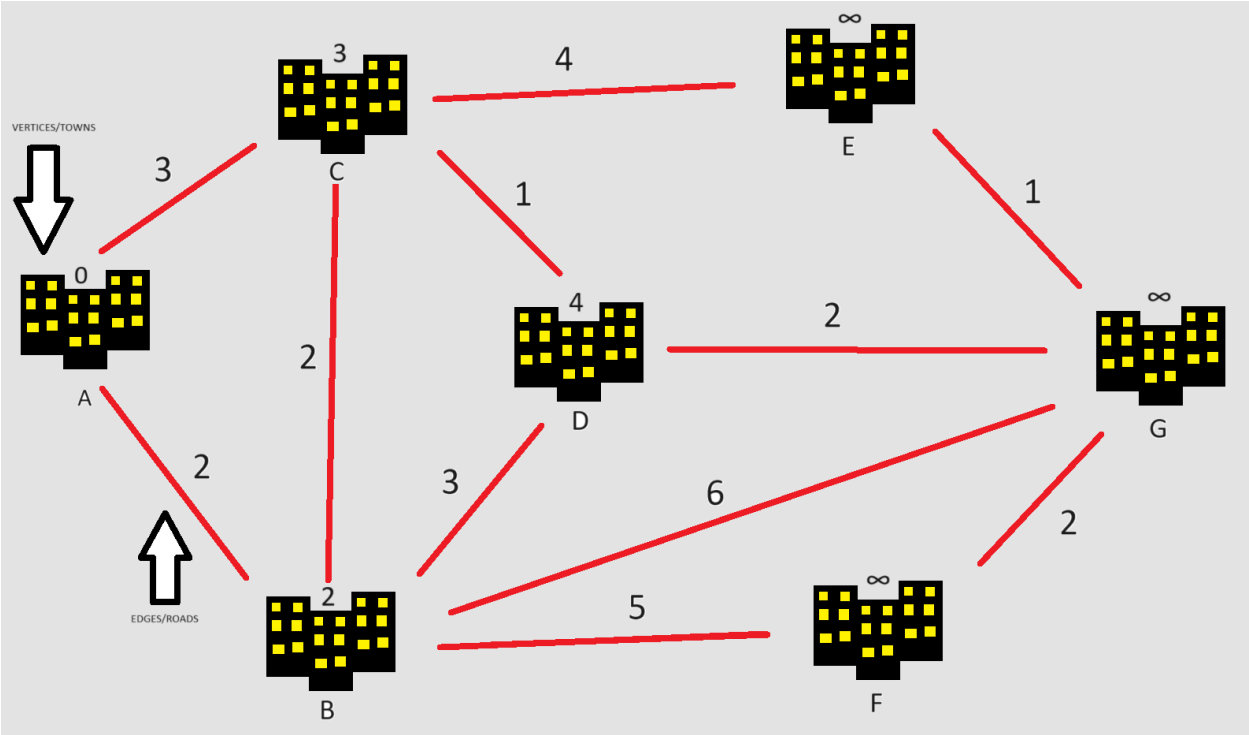

Figure 1: Weighted Graph represented as a map with towns and roads

With the weighted edges actually being the time to travel from one city to another, and the numbers on top of the towns that are estimates of how long it takes to get from town A to that town. Additionally, some of the cities have infinity signs on top of them because they haven't been explored yet so the estimate of how long it takes to get there is not yet computed. (Spanning Tree, 2020) Now, if we pose the question "what is the fastest way we could get from town A to town G?", we could answer that by using Dijkstra's Algorithm because the algorithm will always prioritize choosing the roads that take less time, so it will explore cities and update the estimates as it goes. By utilizing this concept it made it easier to understand Dijkstra's algorithm. Now what we had to do was apply what we learned in class and online to implement an undirected weighted Graph ADT and how to implement Dijkstra's Algorithm on our graph.

Body

Section 1 - Implementation

The major aspects of our project can be summarized into three classes related to graph theory. These are the Vertex class, Edge class and the Graph class. The Vertex class represents a node within the graph. Each holds a unique identifiable label, and and includes methods like addEdge, removeEdge, and an edges adjacency list that is implemented with an unordered map to keep track of all neighbors. The Edge class represents the relationship between two vertices. Each Edge is represented by a source and target vertex, as well as, a weight to represent the distance between two vertices. The Graph class is the all encompassing structure that hosts the entire network of vertices and edges. Internally, the code uses an unordered map to associate each label to its appropriate vertex. The graph class further provides methods like addVertex, addEdge, printAdjacencyList, removeVertex, and removeEdges which calls on the underlying Edge and Vertex classes to add, remove or show relationships within the graph.

Section 2 - Issues faced

At first, we tried implementing the graph and Dijkstra's Algorithm all in the same file. However, further down the line, debugging became too hard because there were too many lines of code. Therefore, we had to start the project again, but this time we made separate classes for the edge and vertex. For the edge implementation, we made a header file with the class definition, method declarations and a C++ file with the method definitions. Each edge has a weight and connects two vertices together. Now for the vertex implementation, we made a header file with the class definition and method declarations and definitions. On top of that, we

had to do a forward declaration of the Vertex class since the Edge class uses elements from the other class.

Section 3 - Dijkstra's Algorithm

Developing the methods that remove or add a vertex or edge was straightforward, but the shortestPath part was the challenge because we had to implement Dijkstra's algorithm to the Graph we implemented. For that we initialized two maps, a priority queue and a comparator. The algorithm itself iterates while the priority queue is not empty. Then it extracts the vertex with the smallest distance from the priority queue. Whenever the extracted vertex is equal to the destination vertex, the algorithm breaks out of the loop because the destination has been reached. Furthermore, for each neighbor of the current vertex, it calculates the distance to each neighbor through the current vertex. If a shorter path from start to neighbor is found, then the distance and the "previous" relations are updated. The updated neighbor is then pushed back into the priority queue. Finally, it then reconstructs the shortest path by starting from the destination vertex and iterating through the "previous" vertices until it reaches the start vertex, and then it returns the shortest path.

Conclusion

Project 4 was challenging, but nonetheless it was a great opportunity to display and hone the skills and knowledge we acquired during the semester. Before starting the project, Dijkstra's Algorithm sounded like something really hard to understand and even harder to implement.

However, by working in groups, helping each other understand concepts and doing this project together made this project less of a challenge.

References

Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++*. John Wiley & Sons.

Spanning Tree. (2020, August 15). *How Dijkstra's algorithm works* [Video]. YouTube. https://www.youtube.com/watch?v=EFg3u_E6eHU