University of South Florida

COP4530 Data Structures

Final Project Report
Dijkstra's Algorithm

Semester: Fall 2025

Group Members:

Andy Ho U25812672

Sara Cathey U85228757

Kevin Mikuta U87519920

Colby Mullins U69093975

Date: November 23rd, 2025

Introduction:

Dijkstra's Algorithm is an algorithm which determines the most optimal path between points on an undirected graph. Each edge of the graph has a positive *weight* which determines the cost of traversing that edge. The goal of the algorithm is to find the lowest cost possible to get from a start point to an end point for all possible points on the graph. In this report, we will go over our specific implementation of the algorithm and exactly how it works.

Process:

The algorithm requires the use of a priority queue; we also implement a distance array to find the cost of a particular route. The distance array stores the minimum distance to a vertex that has been found so far. Each vertex's distance is initialized to infinity (not explored yet), in our code this is classified as ULONG_MAX, this is not infinity but still a very large number. All vertices except for the starting position are set to this value; the starting value is initialized to zero. The priority queue stores a queue of vertices scheduled to be explored along with their total cost up to that point. The vertex at the front of the queue is the one with the lowest cost. If weights are equal, then they are sorted in alphabetical order. At the start of the algorithm, the starting vertex is added to the queue with a cost of 0. The algorithm then repeats the following steps until the queue is empty:

1. Pop the first vertex *x* out of the queue.
2. For each vertex *y* adjacent to vertex *x* via edge *e*, if distanceArray[y] > distanceArray[x] + weight(*e*), add vertex *y* to the queue and update distanceArray[y] to the lower cost.

When the priority queue is empty, we have processed every single vertex in the graph, and the minimum cost of every vertex in the graph from the starting point has been found. The if statement in the second step ensures we do not re-explore vertices which have already been found to have a lower cost than what was just discovered.

Discussion:

Based on the process described above, Dijkstra’s Algorithm finds the minimum cost of *all* vertices in a graph from a starting vertex at once. It returns the shortest distance to the specified vertex e and modifies the given path vector to contain the shortest path from the start vertex to the end vertex. Additionally, if a graph is frequently changing and we need the minimum cost of a vertex often, then we would have to run the algorithm every time the graph changes. For sufficiently large graphs, this may not be optimal as the algorithm has a time complexity of O((V+E) log V) time. The algorithm is best suited when all we care about is distance in a static graph which does not change frequently. Once we have run the algorithm once, we can simply query the returned distance array to get the distance of any vertex later. Caching the returned array is best, as querying it has a time complexity of O(1).

For our graph data structure, we used a map with keys as strings and values as lists of Edge structs. In a standard graph where the vertices are numbered, it would make sense to use a vector of vectors, but since our vertices are indexed by a label string only, using a map with strings as keys was chosen as the best option. Because of this, many functions must perform extra checks to see if something already exists on the map since the standard “[]” operator would simply create a new entry in the map if it did not exist, unlike a vector.

The priority queue is implemented using a min heap. Both the priority queue and min heap used do not need a comparator function passed to them because with how the adjacency list and Edge struct are created, the default “<” operator functions as intended to compare values. A standard binary heap was used as it often outperforms Fibonacci heaps for smaller data sizes and was much simpler to implement.

Conclusion:

Our design correctly implements Dijkstra's algorithm—it successfully computes the shortest distance from a starting vertex to all nodes in a graph. We were able to achieve this with a run time of $O((V + E) \log V)$. While it may have been possible to achieve a potentially faster run time by utilizing a Fibonacci Heap, which has a time complexity of $O(E + V \log V)$, for a smaller project like this, it is better to implement with a binary heap as it is much simpler to design.