Interactive Maps

Samin Mahi, Sadman Md Sakib and Taseen Hakim
CS 4310
Western Michigan University
Kalamazoo, MI

I. Abstract

This project aims to develop a custom bike routing app for the city of Kalamazoo and Portage, Michigan, that recommends safe and efficient bike routes based on street typologies and stress levels. The purpose of the app is to promote active transportation and improve bike infrastructure by providing a user-friendly solution that prioritizes safety and convenience for bicyclists.

II. Research & Design

Our research included going through section 4.4 in Algorithms (4th ed), Sedgewick, R, & Wayne, K(2011). The pseudocode for the algorithm helped us understand the structure for building our own which would achieve our goal. We explored the internet to learn more about the algorithm and compare it to other path finding algorithms. We learned about how google maps works as much as we could as google does not explicitly disclose the details of their routing algorithm. Other than this most of our research was done online through basic google search, youtube etc.

Dijkstra's algorithm is a graph search algorithm that finds the shortest path from a starting node to all other nodes in a graph with non-negative edge weights. It works by iteratively selecting the unvisited node wit\h the smallest distance, updating the distances of its neighbors, and marking the node as visited. The algorithm continues until all nodes are visited or the target node is reached. The shortest path can be obtained by backtracking from the target node to the starting node, following the nodes with the smallest distances.

As our program finds the shortest path from one point to another, the inputs are the starting point and the ending point. The interface design is as follows:

1. A graph is produced using a random set of data.
2. Before the inputs are compiled all the vertices are colored black.
3. All vertices are labelled (e.g A,B,C,etc.).
4. All the edges connecting the vertices have their weight/distance displayed.
5. The inputs are taken in input boxes located at the bottom of the interface.
6. Once the inputs are compiled the starting node turns green and the ending node turns red.
7. The path is shown by the edges that light up as the edges turn white.
8. The middle/visiting nodes turn orange.
9. At the bottom the shortest distance is calculated and shown in terms of the total distance of the edges.
10. Below that the path along with the vertices involved are shown using arrows in text format.

III. Implementation

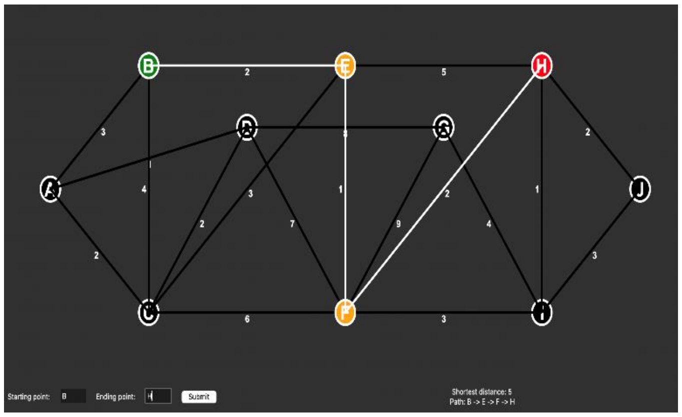
We implemented this project using Python and imported heapq and tkinter as tk which helps create the python interface and represent the gui in the interface.

We defined the Dijkstra function first that takes the graph, start node and end node as the parameters. Here, the distance from the starting vertex to itself is set to zero. The function initializes a dictionary to store the minimum distance of each vertex from the starting vertex, a dictionary to store the previous vertex in the shortest path of each vertex and a priority queue with the starting vertex and its distance. Using a loop this queue is emptied starting with the vertex with the shortest distance from the starting vertex. Within this loop whenever the current vertex becomes the end vertex, the current vertex and the path from the starting vertex is returned.

The graph building required three other functions first one being the draw_graph function, that draws the initial graph on the canvas. The reset_graph function resets the graph to its initial state. The color_path function labels the edges and shows the path for the shortest distance.

Next, the Dijkstra_GUI class is defined. This includes the run_algorithm function that runs the Dijkstra algorithm using the graph. The draw_edge, draw_node and draw_edge_label functions colors the nodes accordingly and boldens and whitens the edges showing the path deduced by the algorithm clearly. There's a function called _init_ that does the other jobs like shaping the nodes, sizing of the nodes and edges, creating the input boxes with labels, creating the submit button and displaying the path and total shortest distance in text format.

The data sample that we used in the demonstration was generated randomly and were included in the class in the graph database as vertices, edges and node positions.



Starting point: B Ending point: H

Shortest distance: 5
Path: B -> E -> F -> H

Figure 1

Finally, the program is run with a specified number of vertices and edges. The initial graph is drawn according to the database and upon given input the program runs dijkstra's algorithm. It calculates the shortest path and shows and labels the path and distance along with the vertices involved.

IV. TESTING & SAMPLES

In the testing phase, we have conducted a series of experiments to evaluate the performance of our graph algorithm. We ran the 'testing.py' script on graphs with varying number of nodes, ranging from 10 to 2000 nodes. This allowed us to investigate the impact of graph size on the execution time of the algorithm. For each graph size, we generated a random graph with a random number of edges and neighbors for each node. Then, we measured the time taken by our algorithm to process the graph and recorded the results.

V. RESULT ANALYSIS

We conducted experiments on our graph algorithm with varying numbers of nodes, from 10 to 2000 nodes. Results showed that execution time increases with the number of nodes, but not uniformly. Processing a graph with 300 nodes takes 0.002252817153930664 seconds, while processing a graph with 320 nodes takes 0.0014200210571289062 seconds.

In general, the performance of the algorithm appears to degrade as the number of nodes increases, but there are some instances where the execution time decreases for larger graphs. This could be due to the random nature of the graph generation and the fact that the algorithm's performance is affected by the specific structure of the graph.

Overall, the results of our testing phase provide valuable insights into the behavior of our graph algorithm under different conditions. While the algorithm's performance tends to degrade with increasing graph size, the actual impact of the graph size varies depending on the specific graph structure. Further research and optimization may be needed to improve the performance of the algorithm for larger and more complex graphs.

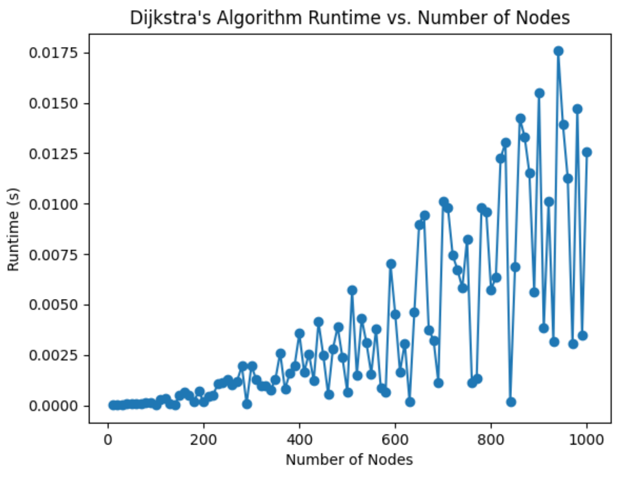


Figure 2

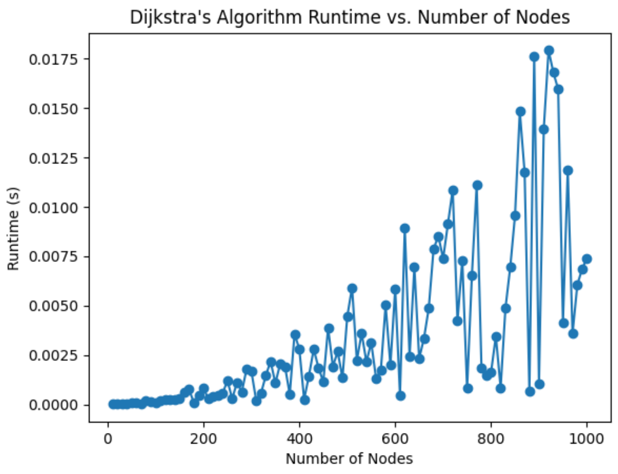


Figure 3

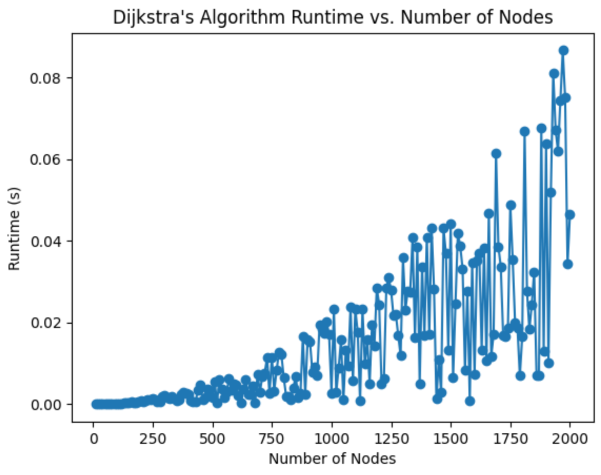


Figure 4

VI. USER GUIDE

This program provides the shortest path in a graph for specific start point and end point. It calculates the shortest distance and displays it on an interface.

The program takes the starting point and the ending point as inputs from the user. The submit button allows the user to confirm the inputs.

Once the inputs are submitted, the program compiles the data and runs the algorithm to give a detailed visual path of the shortest path from the starting point to the end point and also highlights the visited nodes.

REFERENCES

- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), 269-271.
- This is the original paper where Edsger W. Dijkstra introduced the algorithm. It provides a foundational understanding of the algorithm and its conception.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press.

- Chapter 24 of this widely-used textbook covers Dijkstra's algorithm in detail, including its time complexity and correctness proofs.
- Wikipedia - Dijkstra's Shortest Path Algorithm This Wikipedia article provides a comprehensive explanation of Dijkstra's algorithm, its history, pseudocode, and practical applications.
  https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

- GeeksforGeeks - Dijkstra's Shortest Path AlgorithmThis article on GeeksforGeeks explains the algorithm with examples and provides code implementations in multiple programming languages.