COP 4530 Final Project

Dijkstra's Algorithm in C++

Co-authored by

Michelle McAveety | U46283961

Trace Comiskey | U49942059

Rylan Pietras | U14552649

Kadin Miller | U95099011

Introduction

For the COP 4530 Final Project, the team was tasked with programming an implementation of Dijkstra's algorithm in C++. Published in 1959, Dijkstra's algorithm enables the calculation of the shortest path between two vertices on a weighted graph through its structured traversal method [4]. The team was also challenged to create methods for adding/removing vertices and edges from the implemented graph within the program. Dijkstra's algorithm is widely implemented in real-world applications to this day. The team's implementation features a command-line based user interface, making it convenient to input custom graphs for analysis through the algorithm. As Dijkstra's algorithm can be utilized in a variety of contexts, the team hopes to improve upon their developed application to allow its integration with other services in the future.

Dijkstra's Algorithm

Edsger W. Dijkstra, a Dutch computer scientist, originally published his eponymous algorithm in 1959 [4]. Notably, he was not the first person to discover the algorithm, though he only uncovered previous publications after his individual findings. His conceptualization of the algorithm was used to demonstrate the capabilities of a new computer, titled ARMAC, in such a way that non-technical audiences could understand.

Dijkstra's algorithm, a solution to the shortest path problem, is rooted in graph theory. Performing this calculation requires the development of an undirected, traversable, weighted graph, where graph nodes (or 'vertices') represent locations, and edges connect vertices in addition to representing the paths between locations [3]. To conduct Dijkstra's algorithm, a repeating procedure must be applied. First, a starting vertex, or location, is selected to begin the path analysis from, and a target vertex is picked to find the shortest path and distance to. Neighboring vertices are then analyzed to determine the shortest path to the next node. As each path is evaluated, the shortest calculated path is then added to a minheap. When all adjacent nodes are exhausted, then the next unexplored location with the shortest path from the starting location is traveled to (selected). Once a node has been visited and all its neighboring paths evaluated, it is marked as 'explored' and is not returned to. Additionally, once a vertex is

traveled to, if a shorter path is evaluated from the starting location to that vertex, the location of origin is saved within the same vertex, which allows for the shortest path to be marked within the graph itself. Finally, the next node to explore is determined by popping the top entry off the minheap. If this node is already explored, it is ignored, and the next top is taken instead. This process is repeated until the destination node is discovered or all vertices are visited.

Dijkstra's algorithm is widely utilized in a variety of modern-day computer science applications [2]. For example, Dijkstra's algorithm can be used to determine the minimum distance between two geographical locations for GPS mapping services. Similarly, Dijkstra's algorithm is used in digital IP routing [1], enabling more efficient transmission of network communications. A lesser-known use case for the algorithm is its use in social networking applications, where a user may be suggested to interact with other users based on the similarities between their previous actions taken on the platform [1].

Method

To enable the team to build this program, a variety of digital resources were utilized. C++ was chosen as the programming language for use in this project as its object-oriented characteristics allowed for logical segmentation of the program. Git, integrated with GitHub, was chosen as the team's version control framework. Git facilitated the sharing of code across multiple devices so each team member had the most up-to-date files when collaborating on the project. Additionally, UNIX-based operating systems were used to compile and test the program at various stages of development.

To develop an implementation of Dijkstra's algorithm, specific class methods were created due to their importance in the algorithm's functionality. The addVertex and addEdge methods are used to build a traversable, weighted graph in an adjacency list structure. The removeVertex and removeEdge methods can be utilized by the user to remove specific vertices and edges of the existing graph. Additionally, the removeEdge method is used in the clear and destructor methods, both of which entirely remove an existing graph. The printAll and printAllVert methods display the existing graph to the user in its adjacency list structure. Finally, the shortestPath method comprises the core functionality of Dijkstra's algorithm, returning an

unsigned long integer with the resultant shortest path length as calculated by the algorithm. Additionally, the shortest node-by-node path taken is saved in a string vector variable, which is provided by the user as an argument for reference.

With the core functionality of Dijkstra's algorithm developed, the team then opted to create a command-line interface to facilitate the creation and evaluation of variable graph data structures. The addition of this procedure concluded the development process, and the team moved to testing and debugging the program. This process ensured that the final product performed consistently regardless of the graph under evaluation.

Results

The team was able to successfully develop a C++ program which implements Dijkstra's algorithm. The resulting program features a command-line user interface to allow for dynamic input of various graph data structures. Notably, the team faced difficulty managing the data types in use by their program. The requirement set to use unsigned long integers produced conflicts with other data types in use by the program. This was resolved by including Boolean types for comparison instead. See Appendix A for the complete project source code.

Conclusion

Dijkstra's algorithm, a solution to the shortest path problem, has been in use for many years in a variety of applications to efficiently calculate relationships between two specified subjects. This most commonly is observed in geographic and networking applications, though its uses extend to other similar fields. For the COP 4530 Final Project, the team was able to successfully develop a user-controllable implementation of Dijkstra's algorithm in C++.

To improve the C++ implementation of Dijkstra's algorithm, the team believes that future adjustments could be made to accept graph input through file parsing. File parsing would allow for more efficient analysis of larger weighted graphs, as they could be accepted from a persisted file as opposed to being manually entered for analysis, which is an error-prone and time-

consuming process. This improvement would allow the C++ implementation to be used in a variety of external applications, as this additional modularity makes it flexible for multiple environments.

References

[1] Soumalya Bhattacharyya. How is Dijkstra's algorithm used in the real world? | Analytics Steps. Retrieved from https://www.analyticssteps.com/blogs/how-dijkstras-algorithm-used-real-world

[2] GeeksforGeeks. 2022. Applications of Dijkstra's shortest path algorithm. *GeeksforGeeks*. Retrieved from https://www.geeksforgeeks.org/applications-of-dijkstras-shortest-path-algorithm/

[3] GeeksforGeeks. 2024. What is Dijkstra's Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm. *GeeksforGeeks*. Retrieved from https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/

[4] Wikipedia contributors. 2024. Dijkstra's algorithm. *Wikipedia*. Retrieved from https://en.wikipedia.org/wiki/Dijkstra's_algorithm

Appendix A

The source code for the team's C++ implementation of Dijkstra's algorithm can be found at the following URL.

https://github.com/mcaveety/cop4530/tree/main/finalproject