Jack Duncan
Data Structures
5.2.23

Project 4 Cover Letter

For Project 4, I chose the Dijkstra's algorithm option. In order to complete the objective, I created three different data structures: a graph to represent the vertices and edges from the input file, a priority queue to find the minimum distance between 2 vertices, and a hash table to store the shortest paths from the start vertex to all the other vertices. My graph class used an adjacency list to represent the graph, utilizing a hash table with quadratic probing to store all the vertices with the adjacent vertices connected as a linked list. The graph has an array of adjacencyLists (modified linked lists), with adjNodes which contain the vertex, weight, and a pointer to the next adjNode. In a sense, my graph was a combination of the quadratic probing and chaining implementations, ensuring O(1 + n/m) time complexity to search for a vertex and O(n) to find a specific vertex it links to; however, each vertex in the test files had a maximum of 3 vertices linked, so it did not have much of an impact on time. The graph also has O(n) space complexity. For my methods, the graph class looks very similar to a quadratic probing hash table, but the addEdge method creates a new item in the array if a new starting vertex is added, but if an old starting vertex is added, the second vertex just moves to the back of the linked list.

In order to implement Dijkstra's algorithm in my main, I had to create a priority queue and then later a hash table. The priority queue has O(n) space complexity, O(1) to insert with tossIn, O(logn) for build heap, and also O(logn) for pop. The priority queue holds the data type Node, which contains the vertex, the preceding vertex in the path, and total cost. In implementing Dijkstra's algorithm, I decided to create a hash table of those same Nodes to essentially "check-off" when the algorithm finds the shortest path from the start vertex to a given vertex. The hash table uses quadratic probing and thus has the same time and space complexities of O(1 + n/m) and O(n) respectively.

Overall, this project was very engaging and I encountered minimal difficulties. I found myself constantly adding new accessor methods and modifying my mutator methods to enable dijkstra's algorithm to work. I dealt with a memory issue because in my adjList destructor I deleted both the head and tail pointers, but I only needed to delete the head pointer. Other than that one bug, I avoided most other major problems that my peers and I have encountered this semester. Below is the output information when run on the server.

	Path	Cost	Time (ns)
p4test0	a c d f h	7	97,021
p4test1	ORD IND MKC DEN SFO LAX	629	112,146

I decided to choose the Dijkstra option for Project 4 because it seemed challenging and had apparent real world uses. Although the huffman code option was also enticing, I did not want to deal with the large files like a *Tale of Two Cities* again. Project 4 was my favorite project because it felt the most rewarding when completed. I did not enjoy the BST project because it was very challenging, but I did not get the same feeling of accomplishment when I got the program to work. The trees seemed very abstract and rooted in theory rather than real world use. Although I know the binary search trees are an important structure for storing, accessing, and storing data, it was not as tangible as Dijkstra's algorithm. For this very reason, my second favorite project was project 2, where we created hash tables for a spell checking tool. Working on assignments with tangible, real-world applications helps motivate me to spend more time on it. Over the course of the semester, I have definitely grown as a C++ programer. In 051 and 052, most of the projects were very clear cut, with required methods and specific ways of doing things. However, we received little instructions for data structures projects, given a required structure, output, and time constraints. This allowed for a lot more creativity, and made coding significantly more enjoyable. Moving forward, I would keep all the projects the same because each of them taught me a different skill that seems important to developing software. Even though I did not enjoy project 3, I still found it valuable.