Yash Nain

July 31st, 2018

00270-62448

Project 3 Report

Overview:

This Dijkstra implementation originally relied on an adjacency matrix for its graph implementation, but for larger inputs the system began to run out of memory to allocate on the heap, since an adjacency matrix takes O(|V|^2) space complexity, which for larger, sparsely connected graphs is inefficient. Instead, an adjacency list implementation is used, which only takes O(|V|) space.

Compile flags used:

```
gcc -std=c99 -Wall -Wshadow -Wvla -Werror -O3 -pedantic
```

Program layout:

Firstly, the program reads the first input file in order to build the graph. Each vertex and edge is saved to a list within a data structure that stores each of the vertices, the edges, and the counts for both. Next, the adjacency list is constructed, where the absolute distance from a node to each adjacent node is calculated and saved to a list, in descending order. Given these pieces of information, a graph traversal can take place. Using the queries provided by the second input file, the minimum distance between two nodes can be found through Dijkstra's algorithm. The first step to occur is to create an array to store the distance of each other vertex in the graph from the source vertex. Every value is set to INF (infinite), with the exception of the source vertex, which is set to a distance of 0. Next, a min-heap is built with the current distances implemented, and the path to traverse each node in the graph is initialized to the node itself. This step takes an overall time complexity of O(|V|).

Then, the smallest node in the heap is removed, and the heap is heapified until valid. The algorithm will visit each node, check the distance to each adjacent node, and update the distance array as necessary with the minimum distance required to traverse to each node. The shortest path is saved to a buffer. This process continues until the heap is empty and takes O(log|V|) time over each iteration, but |E| iterations occur (since each edge is visited once). After every task, or in the event any task fails, all associated memory with the program is freed, to ensure that no memory leakage can occur.

Performance:

Since this Dijkstra implementation uses a heap to rank the distances for each node, the overall time complexity for the traversal itself is O(|E|log|V|), and a space complexity of O(|V|). Each query of the program took < 0.00e-3 seconds to process, but as the number of edges increased, making the graph more strongly connected, a marked effect occurred for each node, bringing the time up to a full second. For a complete graph, the overall time complexity was O(|E| * |V|), since the adjacency list begins to behave more like an adjacency matrix.