Michael Mroczka

Bruce Elenbogen - CIS 3501

Program 3: Highway – Get My Entourage to the Premier on Time

**Justifying the Data Structure**

In this program I chose to use an *Adjacency Matrix* as opposed to using a wide variety of other data structures — the primary antagonist being an *Adjacency List*. While both data structures have their place I determined that an Adjacency Matrix (implemented with classes and structs) would be a better choice for this assignment.

From what I researched online and learned in class, an Adjacency Matrix uses more memory and is slow to iterate over all the edges, but the upsides include very fast lookup times when checking for the presence or absence of a specific edge. The Adjacency List uses less memory but at a cost of slower lookup times. Adjacency matrices are O(n2) space and have the capability of adding or identifying an adjacent edge in constant time O(1), while an Adjacency List takes O(n+m) space and O(n) for lookup time.

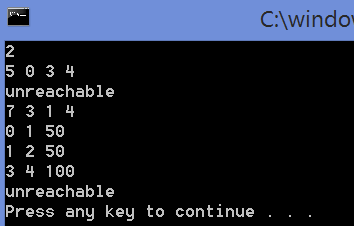
The only real downsides to an Adjacency Matrix is its larger memory consumption and the redundant information when doing undirected graphs. This particular assignment’s entire purpose was speed and getting to the correct answer as fast as possible. Our graph was a directed graph and since memory wasn’t an issue, it made sense to use the faster data structure.

|  |  |  |
| --- | --- | --- |
| * + n vertices, m edges   + no parallel edges   + no self-loops | **Adjacency List** | **Adjacency Matrix** |
| Space | **n** + **m** | **n**2 |
| Finding all adjacent  vertices to **v** | deg(**v**) | **n** |
| Determining if **v** is adjacent to **w** | deg(**v**) | 1 |
| Adding a vertex | 1 | **n**2 |
| Adding an edge to **v** | 1 | 1 |
| Removing vertex **v** | **n?** | **n**2 |
| Removing an edge from **v** | deg(**v**) | 1 |

**White Box Test**

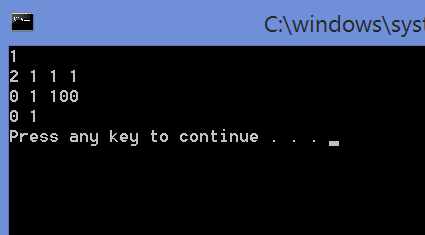
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Being Tested** | **Test** | **Reasoning** | **Input** | **Actual Output** |
| Int main() | Multiple test cases | My graph wasn’t resetting properly before since I was declaring the matrix outside of the main program so output would be right for first test, but no subsequent tests | 3  3 2 1 2  0 1 100  2 1 50  4 3 2 1  0 1 200  0 2 300  2 1 5000  5 4 2 4  0 1 100  1 2 300  2 1 500  1 4 800 | unreachable  5000 2 1  1300 2 1 4 |
| void dijkstras() | Testing unreachable vertices.   1. unreachable since there are no streets 2. unreachable since you just can’t get to it from starting node | Nodes can’t be reached if nodes are unique and there are no streets | 2  5 0 3 4  7 3 1 4  0 1 50  1 2 50  3 4 100 | unreachable  unreachable |
| void shortestPath() | Testing ouput when your starting/ending point are identical.  I wasn’t sure if you’d consider this illegal, but since it is reachable I considered it a valid test and therefore shouldn’t display ‘unreachable’ | If start/end node are the same, then distance should always be 0 and the path should always just be that node | 1  2 1 1 1  0 1 100 | 0 1 |
| void dijkstras() | Testing that it will pick the SHORTEST path even if there is more than one way to get to the destination | We want the algorithm to always pick the shortest path that is *also* the most cost effective | 1  3 3 0 2  0 2 500  0 1 10  1 2 10 | 20 0 1 2 |

Test 1

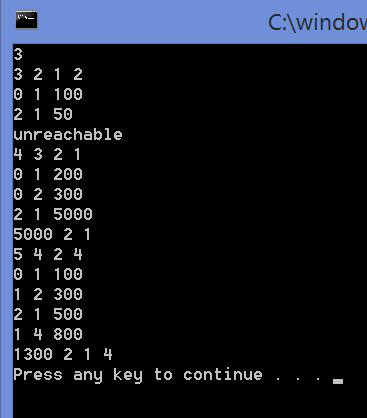


Test 2





Test 3



Test 4

