ldelamotte17@georgefox.edu
Assignment 2
2018-02-14

1. Complete Dijkstra's algorithm.

2. Analyze the complexity of your pathLength() and shortestPathBetween() methods. For shortestPathBetween() you need to identify and account for all non-trivial complexities in your implementation as well as stating the overall complexity.
pathLength(): $O(n^2)$.
shortestPathBetween(): $O(n^3)$

3. Weiss exercise 2.25
Program A: $150N\log_2 N$, Program B: N^2.

a. Program A takes 19,933,778.14 units of time and Program B takes 100,020,001 units of time for N=10,001. **Program A has the better guarantee.**
b. Program A takes 42,328 units of time and Program B takes 2,500 units of time for N=50. **Program B has the better guarantee.**
c. Program A takes 1,494,867.643 units of time and Program B takes 1,000,000 units of time for N=1,000. **Program B has the better guarantee.**
d. **No, it isn't possible.**

4. Suppose you have a computer that requires 1 minute to solve problem instances of size *n*=1000. Suppose your new computer runs 1000 times faster. What instance sizes can be run in 1 minute, assuming the following time complexities *T(n)* for the problem? (hint: represent 1000 in the same way as the right hand side of the equation)

1. $T(n) = n \rightarrow$ **n = 1,000,000**
2. $T(n) = n^3 \rightarrow$ **n = 100**
3. $T(n) = 10^n \rightarrow$ **n = 6**

5. (15 points) Suppose we have a directed, weighted graph given by the following list of edges: [(A, B, 1), (A, D, 9), (B, C, 1), (D, B, -20)]. Give a step-by-step description of how Dijkstra's algorithm proceeds (and fails!) on this graph to find shortest paths from A.

1. Start with A. The distance from A to A is 0, so add that to the table. The distance to all other vertices are unknown, so make those distances ∞.
2. We want to visit the unvisited vertex with the smallest known distance from the start vertex. To do this, examine all adjacent vertices and their distances. If their distance from the start vertex is smaller than the current vertex in the table, update the table with the new distance and the most recent vertex. In this case, B and D are adjacent to A and both have smaller values than ∞. We update the table.
3. Add A to the list of visited vertices.

4. Choose the vertex with the shortest distance from the unvisited vertex list based off the table. That's B.
5. We look at B. We want to visit the unvisited vertex with the smallest known distance from the start vertex. To do this, examine all adjacent vertices and their distances. In this case, C is adjacent to B and this distance has a smaller value than ∞. We update the table.
6. Add B to the list of visited vertices.
7. Choose the vertex with the shortest distance from the unvisited vertex list based off the table. That's C.

8. C has no adjacent vertices, so we add C to the list of visited vertices. We then choose the vertex with the shortest distance from the unvisited vertex list based off the table. That's D.
9. We look at D. We examine all adjacent vertices and their distances. In this case, D is adjacent to B and this distance of -20 has a smaller value than 1. Herein lies the problem. Technically, -20 is *less than* the current value (1), but is still a greater *length* (based on absolute value. Dijkstra's algorithm requires a graph to be connected and to have **nonnegative weights**. This graph does not fit that qualification and therefore, **cannot find the shortest paths from A**.

Vertex	Shortest distance from A	Previous vertex
A	0	null
B	1	A
C	1	B
D	9	A