Daniel Chai

March 3rd, 2017

HW5 Write-Up

1. **Output:** GAFBDEC
   1. GA > AF > FB > B > D > E > C
2. A) **Output:** (B, E) > (C, G) > (D, G) > (A, B) > (E, F) > (E, G)

1: (B, E) (C, G) (D, G)

2: (A, B) (C, D) (E, F)

3: (A, E) (B, F) (E, G)

4: (A, D) (F, G)

B) **Output:** (A, B) > (B, E) > (E, F) > (E, G) > (G, D) > (G, C)

|  |  |  |  |
| --- | --- | --- | --- |
| Vertex | Known? | Cost | Prev |
| A | Y | 0 |  |
| B |  | 2 | A |
| C |  | ∞ |  |
| D |  | 4 | A |
| E |  | 3 | A |
| F |  | ∞ |  |
| G |  | ∞ |  |
| Vertex | Known? | Cost | Prev |
| A | Y | 0 |  |
| B | Y | 2 | A |
| C |  | ∞ |  |
| D |  | 4 | A |
| E |  | 1 | B |
| F |  | 3 | B |
| G |  | ∞ |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Vertex | Known? | Cost | Prev |
| A | Y | 0 |  |
| B | Y | 2 | A |
| C |  | ∞ |  |
| D |  | 4 | A |
| E | Y | 1 | B |
| F |  | 2 | E |
| G |  | 3 | E |
| Vertex | Known? | Cost | Prev |
| A | Y | 0 |  |
| B | Y | 2 | A |
| C |  | ∞ |  |
| D |  | 4 | A |
| E | Y | 1 | B |
| F | Y | 2 | E |
| G |  | 3 | E |

|  |  |  |  |
| --- | --- | --- | --- |
| Vertex | Known? | Cost | Prev |
| A | Y | 0 |  |
| B | Y | 2 | A |
| C |  | 1 | G |
| D |  | 1 | G |
| E | Y | 1 | B |
| F | Y | 2 | E |
| G | Y | 3 | E |
| Vertex | Known? | Cost | Prev |
| A | Y | 0 |  |
| B | Y | 2 | A |
| C | Y | 1 | G |
| D | Y | 1 | G |
| E | Y | 1 | B |
| F | Y | 2 | E |
| G | Y | 3 | E |

1. a) Map temp = new map<Vertex, Set<Vertex>>

for each vertex1 in graph.keySet:

for each vertex2 in Set<Vertex>:

if(temp.get(vertex2) is null)

create new tempSet

temp.add(vertex2, tempSet)

add vertex1 into the set that belongs to the vertex 2 in temp

return temp

b) Set temp = new Set

for each vertex1 in graph.get(source):

for each vertex2 in graph.get(vertex1):

temp.add(vertex2.value());

return temp

c) a) The worst case would be N(N-1) or N^2 because if all the vertices were connected to each other, there would be a lot of duplicates that this code would run through

b) Similar to part a, I would have to go through every single vertices of the vertices that I am pointing to which makes it N^2

1. a) Dijkstra’s algorithm will not always find the path with the fewest edge. I think this is because of the way ties are resolved. So, if the ties between costs are resolved by the number edges to them, it might fix the problem.

b) (A, B, -2) (A, C, 1) (C, B, -6)

When asking for A to B, Dijkstra’s algorithm will return -2 as the lowest cost. However, going through C will make this cost -5, which makes the algorithm faulty.

c) (A, B, 1) (A, C, 2) (C, B, -5)

If you add 6 to every cost, the algorithm will return the 7 as the lowest cost from A to B. Subtracting 6, it becomes 1. However, the lowest cost is -3 by going through C.

This does not work, because by adding to every cost the end cost is skewed by the number of edges leading to it. From my example, if I went to B through C, I would have to add 6 twice, since I am going through two edges.

1. A) I created my own edge.txt and vertex.txt to see if the results that I wanted came out.

b) I implemented the algorithm using a PriorityQueue that was part of the java collection.