# Homework 4 Writeup

Jackson Hart

May 15, 2022

## Problem 1

### A)

I would recommend Dijkstra's algorithm. To find the best route from the distribution center, you would be able to just input the distribution center as the source, and run it with each of the other towns as the target. As an example, to find the best route to city B, the algorithm would look at E because it has the smallest edge from G, but all of E's edges + the edge it took to E are greater than G's other edge to H. From there, it would consider the smallest edge of H which would allow it to find B. Here are the resulting best paths:

| City | Path |
|---|---|
| A | G $\rightarrow$ F $\rightarrow$ A |
| B | G $\rightarrow$ H $\rightarrow$ B |
| C | G $\rightarrow$ C |
| D | G $\rightarrow$ E $\rightarrow$ D |
| E | G $\rightarrow$ E |
| F | G $\rightarrow$ F |
| H | G $\rightarrow$ H |

## B)

This algorithm assumes there is already a shortest distance function created called Dijkstra which takes the graph, the source, and target, defined as G, s, t respectively.

**Algorithm 1** Optimal Distribution Location

```
Require: $G \leftarrow$ graph
  Paths $\leftarrow$ Dictionary which takes a pair of nodes as its key (which is commutative) and a path for the value
  MaxPath $\leftarrow$ List containing the maximum path of the corresponding vertex
  for Vertex source in $G$ do
      Max $\leftarrow -\infty$
      for Vertex target in $G-$source do
          if Paths contains (source, vertex) then
              Max $\leftarrow$ max(Max, Paths[source, vertex])
          else
              Paths $\leftarrow$ Dijkstra(G, source, target)
              Max $\leftarrow$ max(Max, Paths[source, vertex])
      MaxPath[source] $\leftarrow$ Max
  return min(MaxPath)
```

Dijkstra's algorithm can be as low as $\Theta(V + ElogV)$, and it runs once for every min path, which is equal to $\binom{t}{2} = \frac{t!}{2(t-2)!} = \frac{t(t-1)}{2}$. So, this gives us a final run time complexity of $\Theta(t^3 + t^2 rlogt)$.

## C)

Town E

## D)

I would give the town I got through the algorithm described above, and the town that is furthest away from it. To do this, my algorithm would work exactly the same except I would return the min(MaxPath) and that path's target.

### E)

A

## Problem 2

### Verbal Description of my Algorithm

I use a set called tree, which contains the vertices that I have visited, a list of the weights of each corresponding vertices in the tree called weights, and the list of vertices with tuple values containing their location. The algorithm starts with the 0th vertex being in tree. While the tree does not contain all the vertices, it searches for the lowest edge in the tree that goes to a vertex outside the tree, and then adds it to the set and adds the weight to weights. Once the set contains all vertices, the algorithm returns the sum of all the weights.

### Pseudocode

**Algorithm 2** MST Algorithm

```
weights ← empty list
tree ← set containing vertices[0]
while tree does not contain all vertices do
    find minimum edge that goes outside the tree
    add to tree and add weight of edge to weights
return sum(weights)
```

### Theoretical Running Time

$\Theta(|V|^2)$