CSC373 – Problem Set 4

Remember to write your **full name(s)** and **student number(s)** prominently on your submission. To avoid suspicions of plagiarism: at the beginning of your submission, **clearly state any resources (people, print, electronic) outside of your group, the course notes, and the course staff, that you consulted**.

Remember that you are required to submit your problem sets as both LaTeX `.tex` source files and `.pdf` files. There is a 10% penalty on the assignment for failing to submit both the `.tex` and `.pdf`.

Due Oct 24, 2020, 22:00; required files: ps4.pdf, ps4.tex

Answer each question completely, always justifying your claims and reasoning. Your solution will be graded not only on correctness, but also on clarity. Answers that are technically correct that are hard to understand will not receive full marks. Mark values for each question are contained in the [square brackets].

You may work in groups of up to THREE to complete these questions.

Please see the course information sheet for the late submission policy.

[15 points]

You are studying the layout of the city of Mediviala. The city is filled with several historical buildings. Being medivial, the dominant material used for constructing these buildings is wood. This makes protecting these buildings from fire a very important aspect of city planning. We will determine that among all the historical buildings in the city, what was the longest time it would have taken in order to get help in case of a fire in one of these buildings.

You have access to a map of Mediviala. All the locations in the city have been numbered from $[n] = \{1, \ldots, n\}$. You are given a list S of streets in the city, with $|S| = m$. Each street is connects two end points that are among the city locations numbered $1, \ldots, n$, and is specified as a pair of such locations. For each street, you are also provided the time in minutes required to travel along the street in either direction.

Finally, you are given two lists. The first list $H \subseteq [n]$ contains the locations of all the historical buildings in the city. The second list $F \subseteq [n]$ represents the list of fire stations in the city.

Your goal is to determine, for each historical building in the city, its shortest distance to the *closest* fire station. Follow the steps below to design an algorithm for this problem that has a worst-case time complexity of $O(m \log n)$.

1. **(1 point)** Describe how will you model the problem, and the data-structure(s) that you will use to store the input information.

   To solve this problem, we will be using an adjacency matrix as our graph implementation, a python dictionary to store the distance between our starting point and the destination point, a min-heap for quick-Dijkstra algorithm, and a python list to store each vertex's parent once they have been picked and removed from the min-heap.

2. **(2 points)** Briefly describe in up to 3 lines how you can solve the problem in $O(|H| m \log n)$ worst-case time, where $|H|$ denotes the number of historical buildings in the city.

To find the shortest path for all historical buildings to a fire station in O($|H|$mlogn), we can loop over each historical buildings "h", and call Dijkstra's algorithm on h with binary min-heap to get the shortest distance between this building and all locations in the city.

Then we can loop over the historical buildings to find the shortest distance to a fire station by going through the dictionary.

3. **(2 points)** Briefly describe in up to 3 lines how you can solve the problem in $O(|F|m \log n)$ worst-case time, where $|F|$ denotes the number of fire-stations in the city.

We will use a similar approach to solve it to q2, where we call Dijkstra algorithm on every fire station, and then loop through all historical buildings and look up the shortest distance to a fire station in our python dictionary.

4. **(5 points)** Propose an algorithm that solves the problem in $O(m \log n)$ worst-case time, and justify its correctness.

Algorithm:

first run Dijkstra's algorithm (with binary min-heap) from a randomly selected fire station. During the process, if we find a fire station at current index(vertex) y, we will set d(y) = 0 (by doing so, we have manually created a path from our "starting" fire station to y that costs 0, so any historical building we run into that's closer to y will have dist(y,h) = dist(s,h)), and we keep going until the algorithm finishes.

There is the possibility of a special case, since we go through the algorithm in one direction, there might be historical buildings that are between two fire stations in the path. We need to find these historical building and take the shortest distance from these two fire station and update the dictionary.

Also, we need to check the streets around each building since the shortest path will only pass that building once with two streets connected to it, the other streets (paths) we did not choose might have shorter distance with other fire stations since we start from a random fire station and this fire station may not be the nearest fire station to this building. And, when we finally update the dictionary, we will get the closest distance from fire stations for every historical buildings.

We will justify the correctness of algorithm by showing every parts of the algorithm will return the output we want.

Part 1 (Line 2-9):

This is the initial part. Similar to the set up in the lecture, we initialize the shortest path graph and a min heap. "key" will be a dictionary to save the shortest distance from source point to vertex "v". We assign every value to infinity, and the source point to 1.

Notice, we are resetting the distance to 0 every time we encounter a fire station. So here the key dictionary basically stores the closest distance from the current index (vertex) to the previous fire station(or starting point if we haven't encountered any fire station yet) as the algorithm progresses. Then we insert every v into a min-heap with value key[v].

Part 2 (Line 10-29):

This is the part we do the modified Dijkstra's algorithm. It's the same as the Dijkstra's algorithm expect we reset the distance to 0 every time we encounter a fire station, also, we will be keeping track of which fire station/historical building we come across (stored using "passed_building" and "passed_station"), and when both of them are not "None", we will select the shortest path between the (passed_station,passed_building) and (s,passed_building) to ensure we take the minimum path between the 2 (mentioned as the special casein the explanation above).

So part 2 will promised to give the graph that contains the shortest distances from each building (vertex) to the previous fire station the algorithm passed.

Part 4 (Line 30-38):

What we have above is correct if and only if we follow the path create by the algorithm. But there can be some hidden paths which are not selected by the algorithm but it will provide a shorter distance between the current vertex to the a fire station.

So we need to check, for all buildings (b), all the edges excluded by the algorithm that connect to b, and see if it connected with a path inside X can give us a shorter distance to a fire station(or starting point).

This will go over every possible path around each building and updating our dictionary according to it will promised that the distance we have will be the closest distance between the current building and a fire station.

5. **(3 points)** Write the pseudo-code for the final algorithm described in part 4

```
1   def shortest():
2     X <- empty graph, H <- empty binary heap
3     key = dict()
4     key[s] = 0, key[v] = inf (for all v not s)
5     ans = dict()
6     # to keep track of the actual shortest distance from fire station y to
7     alt = dict()
8     # to keep track of which historical building is between fire stations
9     passed_building = None
10    passed_station = None
11    for each v in V do
12        inset v into H with key[v]
13    while H is not empty:
14        m = ExtractMin(H)
15        add m to X
16        for each edge(m,y) do
17            delete y from heap
18            if y is a fire station:
19                key[y] = 0
20                alt[y] = key[m]+edge(m,y)
21                passed_station = y
22            if y is a historical building:
```

```
23                  passed_building = y
24              if passed_building and passed_station:
25                  key[passed_building] = min(key[passed_building],
26                  alt[passed_station]-key[passed_building])
27                  passed_building = None
28                  passed_station = None
29              key[y] = min(key[y],key[m]+edge(m,y))
30              insert y to H with key[y]
31      for each b (buildings) in the graph X that has
32      edges not included in shortest path graph H:
33          curr = key(b)
34          for each b-connecting edge(b,i):
35              curr = min(d(h), key(i) +
36              edge(b,i))
37          key[b] = curr
38          if b in historical buildings:
39              ans[b] = curr
40      return ans
```

6. **(2 points)** Describe a short modification to the above algorithm so that it also finds the closest fire station for each historical building in addition to the shortest distance in $O(m \log n)$ worst-case time.

   The run time for part 1 and 2 will be O(mlogn) since it's basically Dijkstra's algorithm with changes that takes constant time (adding values and look up dictionary).

   Part 3 go over X once, with take at most O(m) (go through every streets), and the if statements and the update dictionary take constant time.

   Part 4 will take at most O(2m) time if we go over each streets twice (An edge(street) is connected by 2 vertices(locations))

   So the final running time will be O(mlogn)