CS3610 Project 5

Dr. Jundong Liu

Here in the great State of Ohio, each city is connected to at least one other city by at least one bi-directional road. In other words, Ohio is laid out as an undirected graph with some vertices sharing multiple (parallel) edges. Now the State is working to install recharging stations for electric cars in every city. Your job is to help find the minimum battery range needed to travel between any two cities in Ohio. Specifically, what you need to do is first calculate a **minimum spanning tree (MST)** among the Ohio cities and then identify within that minimum spanning tree the road with the longest distance (i.e., **the longest edge in the MST**). Given this information, car companies can fine tune their car batteries to ensure that no driver is left stranded with a dead car battery outside a city.

Implementation: array version, 6 final points

In order to find a minimum spanning tree, you must implement the Prim's algorithm described on page 728 in the textbook. This is the algorithm explained in April 8's lecture (I will talk about it again on April 13), where the details are shown in the attached lecture PPT.

If you recall, this algorithm runs in $O(V + V^2 + E)$ time, where V is the number of vertices in the graph and E is the number of edges. The $O(V^2)$ component refers to the time taken to find every unvisited, minimum weight vertex at the start of the while loop. The $O(E)$ component results from comparing and possibly updating the weights of all nodes adjacent to the minimum weight vertices selected at the start of the while loop. In other words, the for loop within the while loop runs $O(E)$ operations in total.

Heap version, 3 bonus points

If your graph is rather sparse (meaning the graph does not contain an overwhelming number of edges), you may want to consider storing the set of unvisited vertices in a min heap. This will help you find all the minimum weight vertices at the start of the while loop in $O(V\log(V))$ time as opposed to $O(V^2)$ time. Of course, when you now update the weight of a neighboring vertex in the for loop below, you must also update that vertex's position in the min heap. As you already know, bubbling up an element in a min heap of n elements takes just $O(\log(n))$ time, but the initial searching takes $O(n)$ time. In order to avoid the $O(n)$ search, you must implement a lookup table that returns a vertex's index in the min heap in $O(1)$ time. If implemented correctly, the overall time complexity of updating the weight values of

vertices throughout the entire run of the min heap Prim's algorithm is $O(E\log(V))$. In other words, using a lookup table and a min heap, the for loop within the while loop runs $O(E\log(V))$ operations in total as opposed the $O(E)$ time in the Prim's algorithm described in the previous paragraph. Thus, the total time complexity of this modified Prim's algorithm is $O(V\log(V) + E\log(V))$. As said early, it is more advantageous to use the min heap version if your graph is not overwhelmingly dense.

Lookup table and min-heap: There could be different ways to implement this lookup table. One setup could be an array of pointers, where each element stores the address of the corresponding vertex in the min-heap. In order to communicate back to the lookup table, the elements in the min-heap should be designed as the combinations of (distance, vertex-index) or (distance, vertex-index, predecessor), which is similar to the ($170B, Washington) setup in Project 4. In project 4, I suggested you use STL's *priority_queue* to implement your max-heap. In this project, however, you may have to implement a heap class by yourself, as the needed operations are not available in STL *priority_queue* or *heap*.

In this project, the 6 final grade points will be awarded if you successfully implement the Prim's algorithm described on page 728 in your textbook, which is an array version. If you successfully implement the min heap version, you will be awarded the 6 final grade points plus 3 bonus points.

Please note that the `Prim2(G, W, n, s)` function on page 728 has several typos. The corrected version can be found in the lecture notes attached in this assignment.

Input Format:

The input may seem a little strange, so let me explain carefully. The first line of each input file is the number of test cases T. Each test case is a new graph and thus a new problem. The first line of each test case is the number of cities N and number of roads M. The cities are conveniently labeled with integers ranging from 0 to $N-1$. Following the city and road counts are M lines describing the M roads. Each road is defined as a pair of cities along with the distance between those cities. Remember, two cities can be connected by more than one road and each road is bi-directional (undirected). Here is an example:

```
2
3 2
0 1 5
1 2 7
3 4
0 1 100
1 2 110
1 2 104
2 1 120
```

The first line indicates the number of test cases T, which in this example is 2. In the first test case $N = 3$ and $M = 2$. So there exists 3 cities and 2 undirected roads in this graph. Following are the 2 lines describing the roads. The first road connects city 0 and city 1 and has a length of 5. The second road connects city 1 and city 2 and has a length of 7. After printing out the minimum spanning tree and battery range for this graph, you will move on to the second test case which has $N = 3$ cities and $M = 4$ roads. Following are the 4 lines describing the 4 roads. Notice that three roads connect cities 1 and 2. Two of these roads are listed as $(1, 2)$, the other is listed as $(2, 1)$. Do not be confused by this. You are working with an undirected graph, so $(1, 2)$ is the same as listing $(2, 1)$. The routine to read an input graph has been provided in the start code.

I will always test with graphs that have a minimum spanning tree. So you should always make sure every city in your own test files have at least one road connecting it to another city.

Output Format:

For each test case, print the longest road of the minimum spanning tree on one line followed by the minimum spanning tree itself. The output for the example given above is as follows:

```
7
(1,0,5) (2,1,7)
104
(0,1,100) (1,2,104)
```

In the first test case, the longest road in the minimum spanning tree is 7, which lies between cities 1 and 2. So the minimum battery range will be calculated based on distance 7. Following is the minimum spanning tree which contains only two roads. These roads lie between cities 1 and 0 with weight 5 and cities 2 and 1 with weight 7. 104 is the longest road in the second test case. The 2 roads are listed last. Remember, you should only have $N - 1$ roads in your minimum spanning trees. DO NOT worry if the cities in your edges are ordered differently. `'(0,1,100)'` is the same thing as `'(1,0,100)'`. Also, DO NOT worry if you end up with a different minimum spanning tree. Just make sure that it is in fact a minimum spanning tree though. They all share the same edge weight total.

Turn In:

Submissions will be made through blackboard. If you have multiple files, package them into a compressed tar file (.tar.gz) or zip file.

Grading:

Total: 100 pts.

- **10**/100 - Code style, commenting, general readability.
- **05**/100 - Compiles successfully.
- **05**/100 - Follows provided input and output format.
- **80**/100 - Successfully found the max road length and minimum spanning tree.