Problem Set 2 - Data Structures & Paradigms

Problem E

Ancient Berland Roads was a pretty satisfying one to get. I - like many others I'm sure - was quick to realise that we were dealing with a union find problem. I think the two hard parts about this problem were first, getting the trick to the actual algorithm (figuring out how to handle union find deletions), and then second, the actual implementation… that took a lot of debugging of edge cases etc…

Initially when doing this problem, the key was to figure out how to delete items out of a union find, and after a few failed google searches of how to implement deletion on a union find, I realised that this question could just be restructured as a series of additions - essentially constructing the final graph first, calculating what it would look like after all of the deletions, and then slowly adding in edges in reverse order as they are 'deleted'. Then, calculating the sum at each step would get us to the answer.

So the initial setup wasn't too bad - reading in all the populations, reading in all of the roads and connections - and then reading in the deletions. After this, I would connect the regions with roads that haven't been deleted, leading us to the final data structure.

I would then loop over the deletions in reverse, adding them into the union find, and then keeping track of the maximum sum at each step. The maximum sum keeping was quite interesting in itself. I knew that I could keep track of the max population with a priority queue, adding in a population everytime two cities are joined or one is updated. However, I also needed a way to delete out of that priority queue to invalidate the old populations. In trying to search for a fast way to do this, I came across invalidation mapping, which solved all of my issues! Basically iirc, keeping a map of all of how many times each number has been invalidated - and then before getting a value out of the pq, checking if it's invalid, if so, skipping it and decrementing the invalid mapping. If it isn't invalid (invalid[pop] == 0), we can take it.

Pretty nifty, saves having to search through pq and delete an element!

In all, the question was quite a hard one, the algorithm didn't take too long, but there were so many things that I needed to fix with the implementation that it became quiteeee difficult.