Mikita Bandarevich - Part 1

We perform a breadth first search on bus numbers. When we start at source, originally we might be able to board many buses, and if we end at T we may have many targets for our goal state.

One difficulty is to efficiently decide whether two buses are connected by an edge. They are connected if they share at least one bus stop. Whether two lists share a common value can be done by set intersection (HashSet), or by sorting each list and using a two pointer approach.

Time Complexity is **O(ElogV)**. E is the length of flights and V is the number of cities. It happens to be the same idea of Dijkstra's algorithm, however, we need to keep the path while exploring. We don't need a visited map here, since if a city is visited multiple times (e.g. there is a circle), the distance will increase, which will decrease it's priority in PriorityQueue. The key difference with the classic Dijkstra algorithm is that we don't maintain the global optimal distance to each node such that we can ignore the following optimization:

alt ← dist[u] + length(u, v)
if alt < dist[v]:

Moreover, there could be routes whose length is shorter but at the same time they pass a higher number of stops -> those routes don't necessarily constitute the best route in the end. In order to solve this, rather than maintaining the optimal routes with 0..K stops for each node, the solution is to simply put all possible routes into the priority queue, so that all of them have a chance to be processed.