Evan Zhang | z5383657

Question 2

2.1 and 2.2

For the first day, check the population of every suburb and its adjacent suburbs and store the combined value in an array corresponding to each suburb.

1. Say for a certain suburb on day 1, its population is ‘x’, and the total population of its adjacent suburbs is ‘y’. If on day 2, the population of that suburb is x + y + 3, we will know that there are at least 3 fast people in that suburb alone.
2. Say on day 1, a suburb has ‘x’ population and total population of adjacent suburbs is ‘y’. If on day 2, the sum of the population of the suburb and its adjacent suburbs is x - 4, we will know that there are at least 4 fast people who have left that suburb on that day.

(If we repeat this check on every single suburb for day 2, we will get an amount of confirmed fast people for that city for that day (i.e., the minimum amount of fast people for that day). If we repeat this check every single day and only store the maximum over all days, we will get the minimum possible number of fast people living in that city based on given information.)

Say for instance that all people in a city are slow. If one suburb ‘i’ has population ‘x’ on day 1 and its adjacent suburbs have total population ‘y’. To maximise the amount of people in suburb i, x people would need to stay and y people would need to move to i for a total of x + y on day 2. However, if the population exceeds this, we can no longer conclude that all people in the city are slow as there is no way for additional slow people to make it to suburb i. Therefore, we can conclude that the people exceeding the maximum are fast people who have arrived.

Similar logic applies for 2. Say for instance that all people in a city are slow. If one suburb ‘i’ has population ‘x’ on day 1 and its adjacent suburbs have total population ‘y’. To minimise the amount of people in suburb i + adjacent suburbs, x people would remain/leave for adjacent suburbs and y people would need to move out of all adjacent suburbs into a suburb that is not i or an adjacent suburb, giving a grand total of x on day 2 for that suburb + adjacent suburbs. However, if the population is below this, we can no longer conclude that all people in the city are slow as there is no way for additional slow people to leave suburb i + adjacent suburbs. Therefore, we can conclude that the people below the minimum are fast people who have left.

Once we have carried these two logical steps out on every single suburb, we take the highest of (fast people arrived) and (fast people left). We do not add them since a fast person who left could be a fast person who arrived.

Now, we run the above for every day and take the highest minimum number of fast people over all days. We do not add them since we are determining a threshold for the minimum amount of fast people, which is a constant value.

Once we determine the minimum number of fast people, we subtract it from the ‘k’ people living in the city. The result is the possible amount of slow people and thus we will determine whether there could possibly be at least one slow person (i.e., if the result is at least 1).

1. To find the population of all suburbs + its adjacent suburbs, we will use a modified BFS. The only modification is that at each suburb, we will add its population to all connected suburbs. As such, we will only visit each suburb once, taking O(n) time and when we carry out the additional step, it will take O(2m) time, since each road/edge is traversed twice (if a pair of

nodes are connected by an edge, each node will add its population to the other, which is two steps, hence O(2m)).

2. To check whether a suburb is greater than the sum of itself and adjacent suburbs on the previous day will take O(n) time, since we are comparing each value of an array to each suburb.
3. To check whether a suburb + adjacent suburbs is less than the suburb on the previous day will take O(n) time, since we are comparing a value of an array for the current day to each suburb's previous population.

These three steps are carried out almost every day (only step 1 on day 1), hence taking O(d(3n + 2m)) time, or O(d(n+m)) time, as required.