Kick Start 2021 - Round F

Analysis: Graph Travel

Test Set 1

We use a brute force strategy by DFS:

1. Pick a room to start. Add it to the path and save the path as visited. If we have $\mathbf{K}$ points, increment our paths counter.
2. Pick a room outside the path that keeps our total points under $\mathbf{K}$. Check if any coordidor connects the room to another room in the path. If yes, add the new connected room to the path and remove the corridor from the set. If no, try the next room outside the path. Evaluating each room choice requires checking every corridor and takes $O(\mathbf{M})$ time.
3. If we are out of corridors or our total points $\geq \mathbf{K}$, we backtrack. Remove the latest room from the path, and try the next available corridor.

In terms of run time in the worst case, we try each possible ordering of rooms. There are $\mathbf{N}!$ orderings. Each ordering has $\mathbf{N}$ room "addition" steps and each step takes us $\mathbf{M}$ time to evaluate (see step 2 above). That gives us $O(\mathbf{N}! \times \mathbf{M})$

Test Set 2

Brute force is not good enough. We approach this problem with dynamic programming. Our strategy will be the following:

1. Find the total magic points of every combination of rooms regardless of whether they can be visited.
2. For each combination of rooms, determine if a room can be visited as a new room.
3. Use dynamic programming to determine which combinations can actually be visited given any single room as a starting point.
4. Filter down to room combinations with point total $\mathbf{K}$ that can be visited.

There are $2^{\mathbf{N}}$ different ways to pick combinations of rooms. Let us first assume that it is possible to visit that combination of rooms and calculate the total magic points. Since there are $2^{\mathbf{N}}$ possible combinations, we can express each combination as a number whose binary representation tells us whether we visit that room. For example:

- $7 = 111$ contains rooms 1, 2, and 3
- $4 = 100$ contains only room 3

We can express the total number of magic points for each room combination as a list of length $2^{\mathbf{N}}$ and takes $O(2^{\mathbf{N}} \times \mathbf{N})$ time to compute. We can also do it in $O(2^{\mathbf{N}})$ using Sum Over Subset Dp technique.

Next, assume we have just visited a combination of rooms, we want to find out if we are able to visit a particular room and add it as a new unique room.

1. If at least one of the rooms in starting combination is adjacent to the new room. To check this, we can maintain a bitmask of rooms adjacent to each room (adjacency bitmask) and then we can check if the starting combination and adjacency bitmask have a common element (this can be done checking that the `AND` of starting combination and adjacency bitmask is non zero).

2. We have the correct number of points to break that room's shield (say our destination is room i and we have P points, this means $\mathbf{L_i} \leq P \leq \mathbf{R_i}$).

We can express this information as a $2^{\mathbf{N}} \times \mathbf{N}$ list of lists called canVisit. Let us say we find that we can visit room i from combination of rooms c. $canVisit[c][i] = true$. We also allow ourselves to start in any room so $canVisit[0][i] = true$. This list of lists takes $O(2^{\mathbf{N}} \times \mathbf{N} + \mathbf{M})$ time to compute since checking each of above two steps takes $O(1)$ time and the computing of adjacency bitmasks takes $O(\mathbf{M})$ time because we need to iterate over all the corridors.

Now we use dynamic programming. We iterate through all the integers between 0 and $2^{\mathbf{N}}$, each integer representing a possible combination of rooms. For a combination x, we iterate through every room already in x. We remove a room and check if it is possible to visit that room (i) from the remaining combination (x') using the $canVisit$ array ($canVisit[x'][i]$). The number of ways to visit combination x is equal to the sum of all the numbers of ways to visit all possible x'. Since we are iterating from smaller combinations up, we know we only need to iterate once. This will tell us which room combinations can be visited. This takes $O(2^{\mathbf{N}} \times \mathbf{N} + \mathbf{M})$

Finally, we pick out all the room combinations that have total of $\mathbf{K}$ points. We filter that down to the combinations that we can actually visit and sum all the unique ways to visit that combination. This just takes $O(2^{\mathbf{N}})$ time. Our final run time is $O(2^{\mathbf{N}} \times \mathbf{N} + \mathbf{M})$