References

OI Wiki: https://oi-wiki.org/graph/tree-diameter/

Problem 4 - Graphics

(a) Longest trip of Xian Chong Country

1. Prove that they will meet during the trip.
   Solution.

Claim.
The center (can be a vertex or an edge) of all the longest paths are the same, which can also named as the center of the tree.

Proof.
Notice that the center of the tree will be on the edge if and only if the longest path goes through even vertices, and it will be on a vertex if and only if the path is odd.

We can do some transformation to simplify the problem. That is, put some extra virtual nodes on each edge (so there will be $2n - 1$ vertices in total). Then, the center of this new tree will always be a vertex (no matter the longest path on original tree is odd or even, it will be odd in the new tree). Also, if we find the center of the new tree, we can simply map it to the original tree.

After that, consider the proof by contradiction as follows:
Suppose that there exists 2 different longest path $\delta(s,t)$ and $\delta(s',t')$ such that $\{s,t\} \neq \{s',t'\}$, where their center are m, m' respectively such that $m \neq m'$.
We have $\delta(s,m) = \delta(m,t) = \delta(s',m') = \delta(m',t') = \frac{d}{2}$.
Since not both $\delta(m',s')$ and $\delta(m',t')$ will cross $\delta(s,m')$ at some points except m', which is trivial (or else $\delta(s',t')$ doesn't pass through m').
W.L.O.G., assume that $\delta(m',s')$ doesn't cross $\delta(s,m')$ at some points except m'. Hence, we have:

$$\delta(s,s') = \delta(s,m) + \delta(m,m') + \delta(m',s') = d + \delta(m,m') > d$$

It leads to a contradiction, so we have $m = m'$.

□

Since all the longest trips have the same center, it's trivial that they will meet at the center of the tree.

□

2. Check which island can be visited during one of the **longest trips**.
   Solution.

We can apply the transformation mentioned in (a-1) to build a new tree with $2n-1$ vertices, so that we only have the case that center is on the vertex.
Then, we have the algorithm to find the center as follows:

- Find the diameter of the tree d and the two endpoints (s,t) by bfs twice. (taught in the lecture on week10)
- To find the center of the path, which is also the center of the tree. First, we bfs the tree from t to compute the distance from t to each vertex. Since we've already computed the distance from s to each vertex, we can easily find the center by enumerating each vertex while finding the center. That is, find a point c such that $dis[t][c] = dis[s][c] = \frac{d}{2}$.

After finding the center c and two endpoints (s,t), we dfs the tree with c as the root, and mark the vertices that is on one of the longest trips as follows:
By observation, we notice that for each vertex u is on one of the longest trips, the conditions as follows will always hold on a tree with c as the root:

- $dis[c][u] = \frac{d}{2}$, if u is a leaf.
- One of the child of u is on one of the longest trips, otherwise.

It's easy to show that this observation is true:
For every two distinct subtrees whose parent are c, if there exists two point p_1, p_2 in different subtrees such that $dis[c][p_1] = dis[c][p_2] = \frac{d}{2}$, then (p_1, p_2) will be one of the longest trips. Hence, the ancestors of the p_1 and p_2 will be on one of the longest trips. Also, it's trivial that s, t are in different subtrees. Thus, for any leaf p such that $dis[c][p] = \frac{d}{2}$ (won't satisfy if it's not a leaf), p will be either in different subtree from either s or t. So p will be one of the endpoints of some longest trip.
For the time complexity because $2n-1 \in O(n)$ and we have done dfs once and bfs three times, we have:

$$4 \times O(n) \in O(n)$$

□

3. Calculate the number of the **longest trips**.
Solution.
Similarly, we apply the transformation in (a-1) to build a new tree with $2n-1$ vertices and computes the diameter d and center c like (a-2).
According to the result in (a-2), there will be one longest trip whenever the two leaves in different subtrees are far enough from c. By dfs from c, we only need to compute the number of the far enough leaves of each subtree whose parent is c, which can be computed as follows with dynamic programming:

$$cnt[u] = \begin{cases} [dis[c][u] = \dfrac{d}{2}], & \text{if u is a leaf} \\ \displaystyle\sum_{v \text{ is a child of } u} cnt[v], & \text{otherwise} \end{cases}$$

And the answer would be:

$$\sum_{v \text{ is a child of } c} (cnt[c] - cnt[v]) \times cnt[v]$$

Since we only apply dfs for one more time, the time complexity will be:

$$O(n) + O(n) \in O(n)$$

□

(b) Grid King and Albert

1. *Solution.*
We can build a graph as follows:

   - Vertices represent not only the intersections from the original grid but also the directions, which are left, right, up, and down (so there are $4 \times n \times m$ vertices). Personally, I will represent the vertices by a tuple (x, y, dir).
   - For each vertex where there isn't a landmine (x, y, dir), we create an edge from it forward the direction it specifies (x', y', dir) with the weight a. Also, if the edge isn't available (i.e. the destination doesn't exist or there is a landmine), we can simply skip that one.
   e.g. if the vertex is $(2, 2, left)$, we build an edge from it to $(2, 1, left)$ with the weight a.
   - For each vertex where there isn't a landmine (x, y, dir), we can choose to make turn. That is, to build 3 edges to the other 3 directions with the weight b. e.g. if the vertex is $(2, 2, left)$, we build 3 edges to $(2, 2, right)$, $(2, 2, up)$, and $(2, 2, down)$ with the weight b respectively.

   Then, we can simply run Dijkstra's algorithm for the shortest distance from $(1, 1)$ (initialize the distances of 4 directions to 0 respectively) to (m, n) (calculate the minimum distance among the 4 directions).
   For the time complexity, we have:

$$|V| = 4mn \in O(mn)$$

$$|E| \leq 4mn \times 4 = 16mn \in O(mn)$$

   Hence, the time complexity for the Dijkstra's algorithm will be:

$$O(|E| \log |E|) = O(mn \log mn) \in O(mn(\log m + \log n))$$

□

2. *Solution.*

First, we make an observation that we only have to remove $m + n - 2$ landmines at most. Here's a proof, since if we can move $m + n - 2$ landmines, then we can go from $(1, 1)$ to $(1, n)$ and make a single turn to go to (m, n) while removing all the landmines along the path. And this path takes the shortest time because it only makes 1 turn and $m + n - 2$ move, and both of them are minimized, which is trivial.

After that, we can build a graph like (b-1) as follows:

- Vertices will have one more dimension represents the number of landmines we have removed (by the observation we made, the value will be at most $m + n - 2$). Personally, I write it like (x, y, dir, rm), which represent the 4 dimensions respectively.
- For each vertex (no matter whether there is a landmine or not) (x, y, dir, rm), we create an edge from it forward the direction it specifies with the weight a. Specifically, if there is a landmine at the destination (x', y'), then build the edge to $(x', y', dir, rm + 1)$; otherwise, build the edge to (x', y', dir, rm). Also, if the edge isn't available (i.e. the destination doesn't exist or $rm + 1 > m + n - 2$), we can simply skip that one.
  e.g. if the vertex is $(2, 2, left, 1)$, and there is a landmine at $(2, 1)$, then we build an edge from it to $(2, 1, left, 2)$ with the weight a.
- For each vertex (x, y, dir, rm), we can choose to make turn. That is, to build 3 edges to the other 3 directions with the weight b. e.g. if the vertex is $(2, 2, left, 1)$, we build 3 edges to $(2, 2, right, 1)$, $(2, 2, up, 1)$, and $(2, 2, down, 1)$ with the weight b respectively.

Then, we can simply run Dijkstra's algorithm for the shortest distance from $(1, 1)$ (initialize the distances of 4 directions with $rm = 0$ to 0 respectively) to (m, n) (calculate the minimum distance among the 4 directions and all the rm). For the time complexity, we have:

$$|V| = 4mn \times (m + n - 2) \in O(mn(m + n))$$

$$|E| \leq |V| \times 4 \in O(mn(m + n))$$

Hence, the time complexity for the Dijkstra's algorithm will be:

$$\begin{aligned} O(|E| \log |E|) &= O(mn(m + n) \times \log (mn(m + n))) \\ &= O(mn(m + n)(\log m + \log n + \log (m + n))) \end{aligned}$$

$$\because \log (m + n) \leq \log (2 \max\{m, n\}) \leq \log m + \log n + \log 2 \in O(\log m + \log n)$$
$$\therefore O(|E| \log |E|) = O(mn(m + n)(\log m + \log n))$$

□