ADA Homework Three

Donald Hsu

December 5, 2023

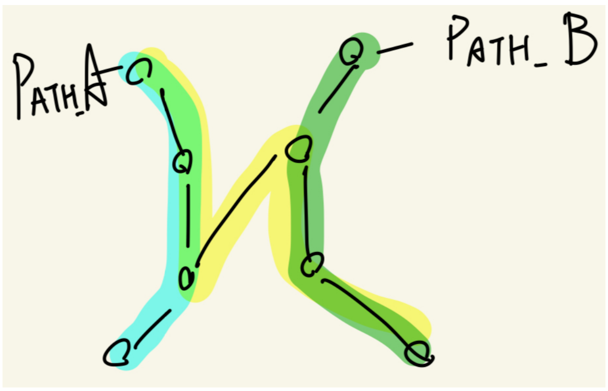
Problem 0

- **Problem 1:** b11902046 李明奕, b11902155 陳致和, b11902011 陳愷欣
- **Problem 2:** b11902084 張閔堯, b11902132 官毓韋, b11902158 黃昱凱
- **Problem 3:** b11902027 陳思瑋, b11902030 王褕立
- **Problem 4:** b11902084 張閔堯, b11902030 王褕立

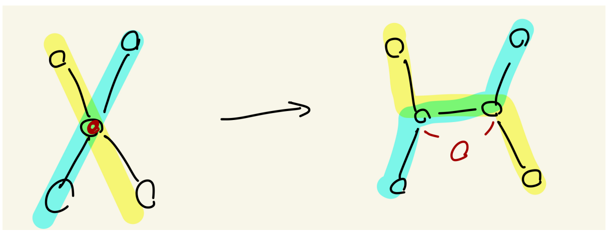
Problem 4

(a) Longest trip of Xian Chong Country

1. If there exist two longest PATH_A and PATH_B with length k in Xian Chong Country but they doesn't cross each other, but since that it is possible to go to any island from any other island, there must be a way to link PATH_A and PATH_B on island i and j without passing any island already on PATH_A and PATH_B. By doing so, I can find that there should be a path that is longer than k that cross PATH_A and PATH_B as the picture below.



Therefore I know PATH_A and PATH_B must cross each other or pass through the same island.
Assume that Arctan follows PATH_A and Alexandar follows PATH_B, and Arctan passed island i in the n_{th} day while Alexandar passed island i in the m_{th} day and $n > m$.
I can find out that after passing the island, Arctan still has $k - n$ island to pass while Alexandar still has $k - m$ island to pass. But since $n > m$, $(k - n) < (k - m)$, Alexandar can get a path that is longer by follow the Arctan path after island i, so I can tell that when $n < m$ the assumption is still wrong, and n will equals$m$ in this stituation.
and since that $n = m$ should be fulfilled if they want to go through the path in any direction, so we can say that $n = m = \frac{k}{2}$.
I can expand the island as PATH_O as the picture below.



by the picture above, we can knew that if they pass PATH_O in the same direction,

they will pass every island and bridge in the same day. If they want to pass PATH_O in the different direction, they will pass the bridge or island in the center of PATH_O at the same day.
Therefore, we knew that they will meet during the trip.

2. First we can do BFS from the root and find one of the longest path's end i and BFS from i to find another longest path's end j and the length of the longest path k (the bridges the path crossed).
   Since that in the previous problem, we can tell that at center of every longest path they must pass the same node (bridge) in the center of them.

   – If it is a node in the center, we can set every node to red, and start BFS from the center, when the node found out it's depth is $\frac{k}{2}$, the node know it is one of the longest path's end, and will change it's color to green. After a node changed it's color, it will tell it's parent node to change it's color to green.

   – If it is a bridge in the center, we can set every node to red, we temporarily remove the bridge, and start BFS from the two ends of the bridge, when the node found out it's depth is $\frac{k-1}{2}$, the node know it is one of the longest path's end, and will change it's color to green. After a node changed it's color, it will tell it's parent node to change it's color to green.

   After then, we can know if the node is on the longest path simply by checking if the color of the node is green.

   correctness:
   If there exist a node n that can't be found we can assume that the length from the one of the longest path end e (which got n between center and e) is shorter than $\frac{k}{2}$ or $\frac{k-1}{2}$ (if the center is bridge.), but it isn't possible since the middle point or the middle bridge had the same length to every end of the longest path.

   time complexity:
   In this algorithm, we did only did BFS ($O(n)$) two time. Therefore the time complexity is $O(n)$.

3. First we can do BFS from the root and find one of the longest path's end i and BFS from i to find another the longest path's end j and the length of the longest path k (the bridges the path crossed).
   Since that in the previous problem, we can tell that at center of every longest path they must pass the same node (bridge) in the center of them.

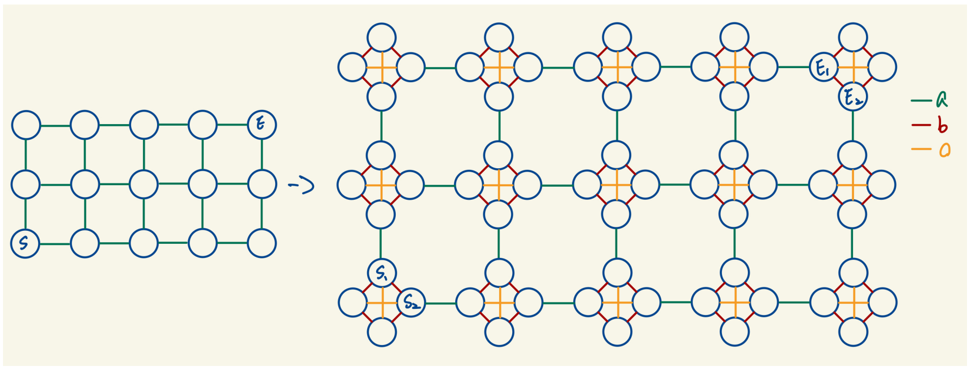
   - If it is a node in the center, we can set every node to red, and start BFS from the center, and count the number n of the nodes that has the depth $\frac{k}{2}$. After then, we can get the numbers of longest path by calculating $n \cdot (n-1)$
   - If it is a bridge in the center, we can set every node to red, we temporarily remove the bridge, and start BFS from the two ends of the bridge, and count the number n_a, n_b of the nodes that has the depth $\frac{k-1}{2}$ from side a and side b of the bridge. After then, we can get the numbers of longest path by calculating $(n_a \cdot n_b) \cdot 2$.

   correctness:
   Since that every longest path will pass the center node/ bridge, if the center we only need to calculate the number of permutation of the end of the longest paths is a node the number of paths will be ${}_nP_2$, if the center is a bridge we know the end point at the bridge's same side can't form a longest path, therefore we only need to match the end of the longest path at the bridge's both side, after multiply $(n_a \cdot n_b)$ by two $(A \rightarrow B \ \& \ B$
   $rightarrowA)$ we can get the number of the longest path.

   time complexity:
   In this algorithm, we did only did BFS $(O(n))$ two time. Therefore the time complexity is $O(n)$.

(b) Grid King and Albert

1. First of all, we can split every node (intersection) in the grid to four node,as the follow pitcture, and give the green lines weight a, since this represent the edges on the original graph, and give the red lines weight b, since this represent the turning in the nodes on the original graph, and give the orange line weight 0, since this represent going straight through the intersection on the original graph.



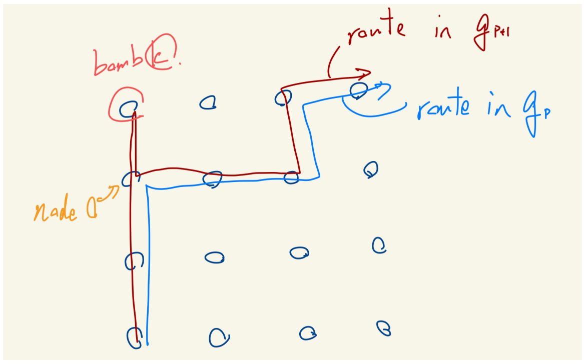
After then, we take the node that are placed with bomb off the graph, and do Dijkstra from node S_1 and S_2, after each Dijkstra, we get the weight sum recorded at node E_1 and E_2, and take min of these four number.

correctness:
In my algorithm, since "turning" takes time, so i split every node into four, representing go up, down, left and right, let turning act like waling through let it able to behavior as a Dijkstra's algorithm's variation. Therefore if the Dijkstra's algorithm is correct, my algorithm will be correct.

time complexity:
the time complexity of Dijkstra through the graph is $O(E + V \log V) = O(mn + (8mn - (m+n)) \log(8mn - (m+n)))$, convert the graph is $O(mn)$. Therefore, the overall time complexity is $O(mn \log(mn)) = O(mn(\log m + \log n))$.

2. I can follow the graph I've made in the question (b)-1, split every node to four and construct the Dijkstra's algorithm, but while we're constructing the tree, every nodes will remember how many landmine the path had passed (deactivated), and every time it met a new landmine n, we'll check how many land mine we've passed deactivated, if it is less than d, we can consider the route passes n.
   And since that the best route would be
   $$(1,1) \to (1,2) \to \cdots \to (1,n) \to (2,n) \to \cdots \to (m,n)$$
   which will pass at most $m+n-3$ landmine, therefore when $d > m+n-3$, we only need to consider the stituation of $d = m+n-3$.
   To practice this algorithm, we need to construct $\min(d,\, m+n-3)+1$ layers of graphs, as for the maps of deactivate 0 bomb (g_0) to deactivate $\min(d,\, m+n-3)$ bombs ($g_{\min(d,\, m+n-3)}$). When we do Dijkstra on g_0 and we met a bomb, we will renew the bomb nodes on g_1 and push the node to g_1's priority queue, after finish Dijkstra on g_0, I will move on to g_1 and so on to $g_{\min(d,\, m+n-3)}$. When I finish Dijkstra on the graphs, all I had to do is get the min among E_1 and E_2 on g_0 to $g_{\min(d,\, m+n-3)}$ and we can get the quickest path when we can deactivate d bombs.

   correctness:
   Since this algorithm are mostly same as the algorithm I've implemented in (b)-1, the only thing need to worry is passing through the same node on the upper layers after we've passed it on the lower layer, but if passing the same node creates a shorter path, this route should be done in the lower layers since it would be fine to no to pass the bomb b just pass the node O as the graph below.



   Therefore, the result in g_p should be better than the result I've got in g_{p+1}, so my algorithm should be correct.

   time complexity:
   Since doing Dijkstra on the modified graph will take $O(mn(\log m + \log n))$ by the proof I've done at (b)-1 and in this algorithm, we'll do it at most $m+n$ times, so the over all time complexity should be $O(mn(\log m + \log n)) \cdot O(m+n) = O(mn(m+n)(\log m + \log n))$.