Question 0: Resources

Slides on BFS and DFS for all problems
This website for understanding properties of tree in prob 2

Question 1: MerwioKart

For each x, y, v_x, v_y create a node. Then from each node create four potential edges from $\{x, y, v_x, v_y\} \rightarrow \{x + v_x, y + v_y, v_x \pm 1, v_y \pm 1\}$. Only create these edges if they follow the following rules.

1. $(x, y) \rightarrow (x + v_x, y + v_y)$ is a legal path.
2. $(x + v_x, y + v_y)$ is a coordinate within the $n \times n$ grid.
3. $(x + v_x, y + v_x) + (v_x \pm 1, v_y \pm 1)$ is within the coordinate grid.

Algorithm: Using Breadth First Search traverse the list searching for $\{x_{goal}, y_{goal}, 0, 0\}$.
Justification: Breadth First Search guarantees the lowest amount of steps to a goal. As steps represent time steps in a 1:1 fashion this guarantees the fastest path.
Time and Space Complexity: Generating the graph and creating nodes is $O(n^4)$ and creating the 4 edges is a constant time operation. BFS is $O(V+E)$ with $V = n^4$ and $E \leq 4V$ as each vertex has at most 4 outgoing connections. Thus the total time complexity is $O(n^4)$. In the same vein we must store all V nodes for a total space complexity of $O(n^4)$.

Question 2: Ancestor Problem

1. **Pre-Processing Algorithm:** To preprocess the tree simply traverse it in its entirety with DFS. As you traverse the tree save the discovery time and the highest return time.

   To save this data in a way that has constant time access use a Hash Map on the keys of the Graph and store the discovery and return time pair as the value. Unfortunately this problem does not specify an ascending range such as $A...B$ or $1...n$ for vertices (which would allow us to use direct indexing in a dynamic array), so hashing will guarantee a near constant time.

   To answer a query like "Is A an ancestor of B" hash both keys and get their values. With the values (A_d, A_r) and (B_d, B_r)[1] we can simply check the inequality of $A_d \leq B_d$ and $A_r \geq B_r$.[2]

2. **Justification:** This discovery/return augmentation has been covered in class. By simply adding a time tracker to DFS we can find when we discover a node and when we return to it for the last time. This interval contains all items that are children of that node. So to answer the question "is A an ancestor of B" we can check if B is within the child interval of A.

3. **Time/Space Complexity:** For pre-processing we use Depth First Search on the tree which has a time complexity of $O(V + E)$. Since $V = n$ and a tree has exactly $n-1$ edges the total time of this operation is $O(n)$. The DFS must hold all V so it has a space complexity of $O(n)$. To store the key-value pairs in a hash-table the table will store some constant factor of n for a total space complexity of $O(n)$.

   To solve a query, the algorithm first hashes both values and retrieves the associated intervals. With these intervals it then performs a comparison and returns the result. Time is average $O(1)$ as hash-table access time is on average $O(1)$ and the comparison is a constant time operation. All auxiliary space needed for these operations is $O(1)$.

[1] with subscript d and r denoting discovery time and largest return time respectively.
[2] To get the proper ancestor you can make both into strict inequalities $\{<>\} \rightarrow \{\leq\geq\}$

Question 3: Graph Questions

1. From this graph we can see that P has 3 children. We can see that Q was discovered immediately after and that R and T follow before returning to P with zero gaps in discovery and finishing time.

   In addition, Q has only one descendant and thus has 1 child.

2. The number of descendants can be calculated with the following formula:

$$\frac{finish - discovery + 1}{2}$$

   P has:

$$\frac{51 - 10 + 1}{2} = 21$$

   Q has:

$$\frac{14 - 11 + 1}{2} = 2$$

   R has:

$$\frac{36 - 15 + 1}{2} = 11$$

   T has:

$$\frac{50 - 37 + 1}{2} = 7$$

3. P is a cut vertex because its removal will segment off Q. We know this because Q only has a connection to P and its sub-tree's earliest discovery time back edge connects to P as well.

   Q is not a cut vertex because its children have a back edge that reaches P.

   R and T both do not have enough information to tell. Since both can have more than a single child it is possible that one branch depends on R or T directly. It is also possible that R or T each have a single child that connects with the back edge.

1. **Proof:** A tree is a connected acyclic graph.

   Given any tree T with n vertices we know that all n vertices are connected.

   To be connected means that there is at least one path p_1 between any pair of nodes.

   To be acyclic limits the quantity of paths between each node to 1.

   When adding an edge between any two nodes in a tree T to make T' we create a new direct path p_2 between these two nodes.

   Since these nodes were previously connected, they already had a path p_1. The addition of this new path p_2 with p_1 creates a cycle.

2. **Proof:** A cycle is a path that starts and ends with a vertex without repeating nodes.

   Each node in this cycle is connected in two different ways to each other, as there is a path in either direction about the cycle.

   Removing any edge within the cycle will sever the cycle, removing it from the graph.

   The pair of nodes still remain connected, as no cut can sever both paths.

   Since the rest of the graph is still connected, the cycle is removed, and the nodes in the cycle are connected this is a tree.