CS160: Algorithms, Fall 2025, **Homework 10**

- Due by 11:59pm on Tuesday, November 18.
- To obtain full credit, you must justify your answers. When describing an algorithm, do not forget to **state any assumptions that you make**, explain why the algorithm is correct, and analyze its time and space complexity.
- Although not specifically stated, you can assume that we look for algorithms that are as fast as possible, and bounds as tight as possible, within the set of tools we've learned in class.
- You may discuss these problems with others, but remember to write the answers on your own. Always cite all sources you used to solve the assignment.
- Remember to submit each question on a separate page.
- Your answers must be typed (LaTeX highly recommended!)

CLRS Reading

See website.

Notation for Graphs and Trees

Recall from lecture: Let $G = (V, E)$ be an undirected graph. We say there is a *path* from $u \in V$ to $v \in V$ if there is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $\{v_t, v_{t+1}\} \in E$ for $t = 1, 2, \ldots, k-1$ and $v_1 = u, v_k = v$. A path $v_1, v_2, \ldots, v_k$ is a *simple path* if no vertex is repeated twice in it. We say that G is *connected* if for any two vertices $u, v \in V$ there is a path from u to v. A *cycle* is a list of vertices $v_1, \ldots, v_k$ for $k \geq 3$, where $\{v_t, v_{t+1}\}$ is an edge for $t = 1, \ldots, k-1$, vertices $v_1, \ldots, v_{k-1}$ are distinct and $v_k = v_1$.

There are four equivalent definitions for a tree. An undirected graph T on n vertices is a *tree* if one of the following holds:

1. T is connected and the number of edges in T is $n - 1$;
2. T is connected, but for any edge e in T, the graph becomes disconnected if we remove e;
3. T is a connected graph with no cycles;
4. for any pair of nodes in T there is exactly one simple path between them.

Problems

0. On the first page of your homework, list all resources you end up using to help you solve the homework problems below (e.g., people, TAs, websites), and specify which problem you used each resource for.

1. MerwioKart is a computer game about racing cars in 2D. Your car is a pixel and the course is encoded as a set of *valid* pixels on an $n \times n$ screen: you're given a 2D array where you can look up any pixel to see if it's valid. Your objective is to get from a given start position to a given end position as fast as possible. Here are the rules:

   - Time is measured in unit steps.
   - As mentioned, you begin at some start position, naturally with zero velocity. At every time step you can modify your horizontal velocity by 1, or keep it the same. The same holds independently for vertical velocity. So if at a particular time you are at pixel x, y and already have velocity v_x, v_y, then at the next time step you will be at position $x + v_x, y + v_y$, after which each component of your velocity may change by ±1 if you wish.
   - You're not allowed to shoot through the end position with arbitrary velocity. You must stop there to pick up your trophy.
   - At every time step your car must be at a valid pixel, but also between steps you must not drive over invalid pixels. To help with this last part, you're given a table, T, of pixel pairs, where for each pair there is a bit letting you know if it's legal to travel from one pixel to the other in a straight line. So in constant time you can you can look up any pair to see if moving directly from one to the other is ok.

   Formulate this game as a graph problem and describe how to find the optimal route for any given race course. What is the time complexity of your algorithm?

   Hint: Your vertices should represent both position and *speed. In other words, vertices represent the* state *that you're in. Thus you should have $O(n^4)$ vertices, each with a "coordinate" $\{x, y, v_x, v_y\}$.*

   Challenge (no additional credit): While a $O(n^4)$ runtime is perfectly sufficient for full credit, by squinting and thinking for a long time we can actually guarantee $O(n^3)$. Can you see how?

2. Suppose we are given a tree T along with its root r. We would like to run some preprocessing on T such that after that we will be able to answer any query of the form "is u an ancestor of v?" in constant time. The preprocessing itself should take linear time

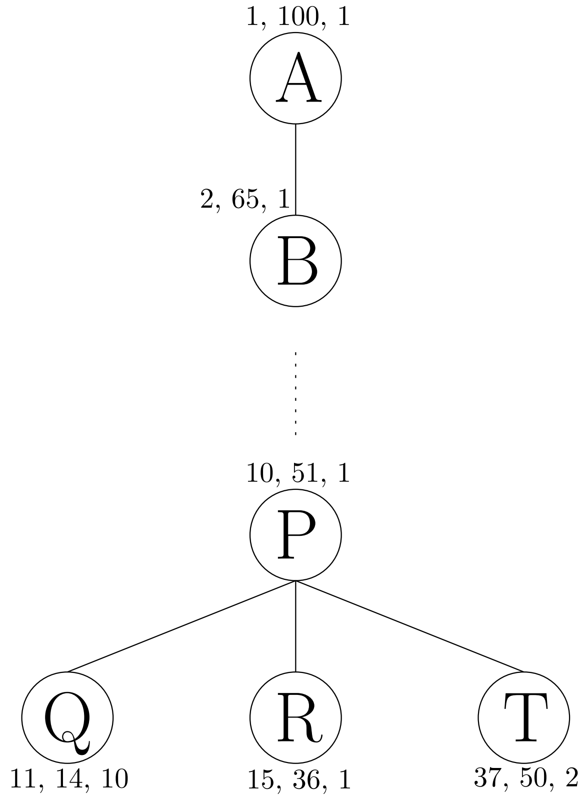
   (a) Describe such a preprocessing and explain how to answer the query.

   (b) Justify the correctness.

   (c) Justify the running time of preprocessing and of a query.

3. The figure below depicts a portion of a graph (many edges and vertices are missing) after running the cut vertex algorithm. Node P is a descendant of node B in the DFS tree. Next to each vertex we note the discovery time ($v.d$), finishing time ($v.f$), and the min of {the earliest discovery time of a node reachable by a back edge from the descendants of the vertex, the discovery time of the vertex itself} ($v.low$).

   For example, for node B, the discovery time is 2, the finishing time is 65, and the earliest discovery time of a node reachable by descendants is 1.



(a) State how many children each of the nodes P, Q, R and T have in the DFS tree, or state that you can't tell. Justify your answers.

(b) State how many descendants each of the nodes P, Q, R and T have in the DFS tree, or state that you can't tell. Note that a node is a descendant of itself. Justify your answers.

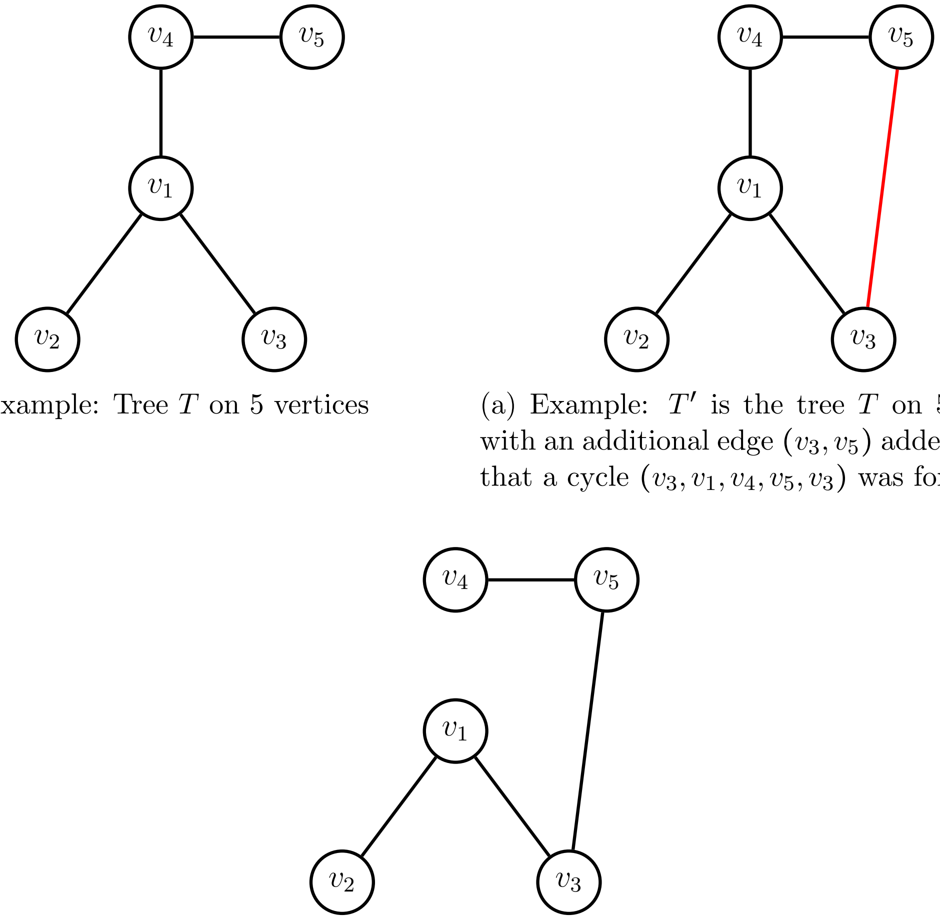
(c) Which of the nodes P, Q, R, and T are cut vertices, and why? For each of the 4 nodes, either explain why it is a cut vertex, why it is not a cut vertex, or why you cannot tell.

4. Let T be a tree on n vertices.

(a) Suppose you add one more edge, call it e, to T (the number of vertices is kept the same). Call the resulting graph T'. Prove that T' contains a cycle. (See Figure 1 for an example).

(b) Let C denote the cycle in T'. You now remove one edge, call it e', from C. Prove that the resulting graph, $T'' = T' - \{e'\}$, is a tree. Note that T'' is not necessarily equal to T since *any* edge can be deleted from the cycle C (i.e., e' need not equal e).

Hints for (b): We know that T'' has $n - 1$ edges, so by one of the definitions of a tree, it is sufficient to show that T'' is still connected. For that, you'll need to show that for any two vertices u and v, there is a path in T'' from u to v. We know that there is a path between them in T' (why?); is this path also a path in T''? If not, how can we change it to become a path in T''?



Example: Tree T on 5 vertices

(a) Example: T' is the tree T on 5 vertices with an additional edge (v_3, v_5) added. Notice that a cycle $(v_3, v_1, v_4, v_5, v_3)$ was formed.

(b) Example: We removed the edge (v_1, v_4) from T'. The resulting graph has no cycles but is still connected (thus it is a tree).

Figure 1: Illustrations for Question 4

Note: This lemma is essential for proving correctness of our Minimum Spanning Tree algorithms next week.

Optional Programming Practice:

- Given a sequence of vertex degrees (e.g., (1, 1, 1, 3)), decide whether or not there exists a tree with that set of degrees.
- https://leetcode.com/problems/number-of-islands/ (Medium)
- https://leetcode.com/problems/max-area-of-island/ (Medium)