CS 170 Homework 3

Due **Saturday 9/20/2025, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, explicitly write "none".

2 Werewolves (13 points)

You are playing a party game with n other friends, who play either as werewolves or villagers. You do not know who is a villager and who is a werewolf, but all your friends do. There are always more villagers than there are werewolves.

Your goal is to identify one player who is certain to be a villager.

Your allowed 'query' operation is as follows: you pick two people as partners. You ask each person if their partner is a villager or a werewolf. When you do this, a villager must tell the truth about the identity of their partner, but a werewolf doesn't have to (they may lie or tell the truth about their partner).

Your algorithm should work regardless of the behavior of the werewolves.

(a) (3 points) For a given person x, devise an algorithm that returns whether or not x is a villager using $O(n)$ queries. Just an informal description of your test and a brief explanation of why it works is needed.

(b) (10 points) Show how to find a villager in $O(n \log n)$ queries (where one query takes two people x and y and asks x to identify y and y to identify x).

Hint: Split the group into two groups, and use part (a). What invariant must hold for at least one of the two groups?

Give a 3-part solution.

(c) (**Extra credit**, 3 points) Can you give a $O(n)$ query algorithm?

Hint: Don't be afraid to sometimes 'throw away' a pair of people once you've asked them to identify their partners.

Give a 3-part solution.

3 Depth first search (10 points)

Depth first search is a useful and often efficient way to organize computations on a graph.

Let G be an undirected connected tree, and let $wt : E \to \mathbb{R}^+$ be positive weights on its edges. We show a template for DFS-based computations below.

```
1: Input: Undirected connected tree G = (V, E) and positive weights wt(u, v) for each
   edge (u, v) ∈ E.
2:
3: Initialization:
4: visited[v] ← False for all vertices v.
5: A[v] ← 0 and B[v] ← 0 for all vertices v.
6: t ← 1.
7:
8: function Explore(Vertex u)
9:     visited[u] ← True
10:    for each edge (u, v) in E do
11:        if NOT visited[v] then
12:            PreVisit(u, v)
13:            Explore(v)
14:            PostVisit(u, v)
```

DFS can be used for different purposes by defining the procedures PreVisit and PostVisit appropriately.

(a) (4 points) In each of the following cases, PreVisit and PostVisit have been defined for you. After execution, the array $A[v]$ will hold a value for each vertex v. Describe in words what $A[v]$ represents.

(i)
```
1: procedure PreVisit(u, v)
2:     return
3:
4: procedure PostVisit(u, v)
5:     A[u] ← max(A[u], A[v] + 1)
```

(ii)
```
1: procedure PreVisit(u, v)
2:     B[u] ← B[u] + 1
3:
4: procedure PostVisit(u, v)
5:     A[u] ← max(A[u], B[v] + 1)
```

(b) (6 points) In each of the following cases, write down pseudocode for PreVisit and PostVisit routines to perform the computation needed.

(i) For each vertex v, compute the maximum weight of an edge along the path from root r to vertex v and store it in array $A[v]$.

(ii) For each vertex v, compute the maximum weight of any edge in the subtree rooted

at vertex v and store it in array $A[v]$.

(iii) For each vertex v, compute the maximum pre-order number of any of its children and store it in array $A[v]$. If v has no children, then $A[v]$ should be 0.

4 Biconnected components (11 points)

Consider any undirected connected graph $G = (V, E)$. We say that an edge $(u, v) \in E$ is *critical* if removing it disconnects the graph. In other words, the graph $(V, E \setminus (u, v))$ is no longer connected.
Similarly, we call a vertex $v \in V$ critical if removing v (and all its incident edges) leaves the graph disconnected.

(a) (2 points) Suppose that $|V| \geq 2$. Can you always find a vertex $v \in V$ that is **not** critical? What about an edge that is not critical?

(b) (6 points) Give a linear time algorithm to find all the critical edges of G.

(c) (3 points) Modify your algorithm above to find all the critical vertices of G.

5 Topological sort proofs (14 points)

(a) (6 points) A directed acyclic graph G is *semiconnected* if for any two vertices A and B, there is a path from A to B or a path from B to A. Show that G is semiconnected if and only if there is a directed path that visits all of the vertices of G. Make sure to prove both sides of the "if and only if" condition.

Hint: Is there a specific arrangement of the vertices that can help us solve this problem?

(b) (4 points) Show that a DAG has a unique topological ordering if and only if it has a directed path that visits all of its vertices.

Remark: This means that a semiconnected DAG always has a unique topological ordering.

(c) (4 points) This subpart is unrelated to the notion of semiconnectedness. Consider what would happen if we ran the topological sorting algorithm from class on a directed graph that had cycles.

Prove or disprove the following: The algorithm would output an ordering with the least number of edges pointing backwards.

6 [Coding] DFS & edge classification (6 points)

For this week's homework, you'll implement DFS and use it to classify edges in a graph as forward/tree, backward, or cross edges. There are two ways that you can access the notebook and complete the problems:

1. **On Local Machine**: `git clone` (or if you already cloned it, `git pull`) from the coding homework repo,

   https://github.com/Berkeley-CS170/cs170-fa25-coding

   and navigate to the `hw03` folder. Refer to the `README.md` for local setup instructions.

2. **On Datahub**: click here and navigate to the `hw03` folder.