Problem Set 7

Data Structures and Algorithms, Fall 2020

Due: November 5, in class.

Problem 1

(a) Give a *tight* asymptotic bound on the running time of the following sequence of operations, assuming the linked-list representation and the weighted-union heuristic. You need to justify your answer.

$$\textsc{MakeSet}(x_1), \cdots, \textsc{MakeSet}(x_n), \textsc{Union}(x_2, x_1), \textsc{Union}(x_3, x_2), \cdots, \textsc{Union}(x_n, x_{n-1})$$

(b) Suppose you are given a collection of trees representing a partition of the set $\{1, 2, \cdots, n\}$ into disjoint subsets. You have no idea how these trees were constructed. You are also given an array $node[1, \cdots, n]$, where $node[i]$ is a pointer to the tree node containing element i. You can create a new array $label[1, \cdots, n]$ using the following algorithm. Prove that if we implement FIND using path compression, LABELEVERYTHING runs in $O(n)$ time in the worst case.

```
LABELEVERYTHING
1: for (i ← 1 to n) do
2:     label[i] ← FIND(node[i])
```

Problem 2

Suppose we want to maintain an array $X[1, \cdots, n]$ of bits, which are all initially zero, subject to the following operations.

- LOOKUP(i): Given an index i, return $X[i]$.
- BLACKEN(i): Given an index $i < n$, set $X[i] \leftarrow 1$.
- NEXTWHITE(i): Given an index i, return the smallest index $j \geq i$ such that $X[j] = 0$. (Because we never change $X[n]$, such an index always exists.)

If we use the array $X[1, \cdots, n]$ itself as the only data structure, it is trivial to implement LOOKUP and BLACKEN in $O(1)$ time and NEXTWHITE in $O(n)$ time. But you can do better! Describe data structures that support LOOKUP in $O(1)$ worst-case time and the other two in the following time bounds.

(a) The amortized time for BLACKEN is $O(\log n)$, and the worst-case time for NEXTWHITE is $O(1)$.

(b) The worst-case time for BLACKEN is $O(\log n)$, and the amortized time for NEXTWHITE is $O(\log^* n)$.

(c) [Bonus Question] BLACKEN in $O(1)$ worst-case, and NEXTWHITE in $O(\log^* n)$ amortized.

Problem 3

Given an adjacency matrix for a directed graph $G = (V, E)$, design an algorithm that determines whether G contains a vertex satisfying the following property: the vertex has in-degree $|V| - 1$ and out-degree 0. To get full credit, your algorithm should only take $O(|V|)$ time.

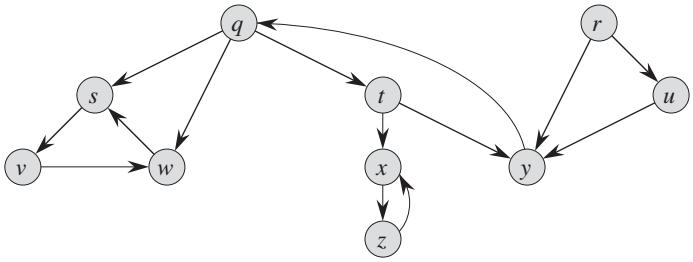
Problem 4

(a) Argue that in a breadth-first search, the value $u.d$ assigned to a vertex u is independent of the order in which the vertices appear in each adjacency list. Nonetheless, show (via example) that the breadth-first tree computed by BFS can depend on the ordering within adjacency lists.

(b) Give an example of a directed graph $G = (V, E)$, a source vertex $s \in V$, and a set of tree edges $E_\pi \subseteq E$ such that for each vertex $v \in V$, the unique simple path in the graph (V, E_π) from s to v is a shortest path in G, yet the set of edges E_π cannot be produced by running BFS on G, no matter how the vertices are ordered in each adjacency list.

Problem 5

(a) Show how DFS works on the following graph. Specifically, assume that the DFS procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Show the discovery and finishing times for each vertex, and show the classification of each edge.



(b) Give a counterexample to the conjecture that if a directed graph G contains a path from u to v, and if $u.d < v.d$ in a depth-first search of G, then v is a descendant of u in the depth-first forest produced.

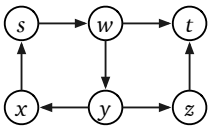
Problem 6

Modify the pseudocode for depth-first search so that it prints out every edge in the directed graph G, together with its type.

Problem 7

[Choose either one of the following two problems, but you are welcome to submit solutions for both.]

(a) Suppose you are given a directed graph $G = (V, E)$ and two vertices s and t. Describe and analyze an algorithm to determine if there is a walk in G from s to t (possibly repeating vertices and/or edges) whose length is divisible by 3. For example, given the graph shown below, your algorithm should return `True`, because the walk $s \to w \to y \to x \to s \to w \to t$ has length 6. To get full credit, your algorithm should run in $O(|V| + |E|)$ time.



(b) Suppose you are given a directed graph G where some edges are red and the remaining edges are blue. Describe and analyze an algorithm to find the shortest walk in G from one vertex s to another vertex t in which no three consecutive edges have the same color. That is, if the walk contains two red edges in a row, the next edge must be blue, and if the walk contains two blue edges in a row, the next

edge must be red. For example, given the following graph as input, your algorithm should return the integer 7, because $s \to a \to b \implies d \to c \implies a \to b \to t$ is the shortest legal walk from s to t. To get full credit, your algorithm should run in $O(|V| + |E|)$ time.