Problem Set 8

Data Structures and Algorithms, Fall 2021

Due: November 25, in class.

Problem 1

Show the data structure that results and the answers returned by the `Find` operations in the following program, using linked-list representation with weighted-union heuristic. Assume that if the sets containing x_i and x_j have same size, then operation `Union`(x_i, x_j) appends x_j's list onto x_i's list.

```
1: for (i ← 1 to 16) do
2:     MakeSet(x_i)
3: for (i ← 1 to 15 step by 2) do                ▷ i = 1, 3, 5, ···
4:     Union(x_i, x_{i+1})
5: for (i ← 1 to 13 step by 4) do                ▷ i = 1, 5, 9, ···
6:     Union(x_i, x_{i+2})
7: Union(x_1, x_5)
8: Union(x_11, x_13)
9: Union(x_1, x_10)
10: Find(x_2)
11: Find(x_9)
```

Problem 2

Consider a union-find data structure that uses union by height without path compression. For all integers m and n such that $m \geq 2n$, prove that there is a sequence of n `MakeSet` operations, followed by m `Union` and `Find` operations, that require $\Omega(m \log n)$ time to execute.

Problem 3

Suppose you are given a collection of trees representing a partition of the set $\{1, 2, \cdots, n\}$ into disjoint subsets. These trees are *not* necessarily built a sequence of `MakeSet`, `Union`, `Find` operations. You are also given an array $node[1 \cdots n]$, where $node[i]$ is a pointer to the tree node containing element i. Your task is to create a new array $label[1 \cdots n]$ using the following simple procedure:

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

(a) What is the worst-case running time of the above procedure if we implement `Find` without path compression?
(b) Prove that if we implement `Find` using path compression, the above procedure runs in $O(n)$ time in the worst case.

Problem 4

Suppose we want to maintain a collection of strings (sequences of characters) with following operations:

- `NewString`(c) creates a new string of length 1 containing only the character c and returns a pointer to that string.
- `Concat`(S, T) removes the strings S and T (given by pointers) from the data structure, adds the concatenated string ST to the data structure, and returns a pointer to the new string.
- `Reverse`(S) removes the string S (given by a pointer) from the data structure, adds the reversal of S to the data structure, and returns a pointer to the new string.
- `Lookup`(S, k) returns the k-th character in string S (given by a pointer), or `Null` if the length of S is less than k.

Describe and analyze a simple data structure that supports `Concat` in $O(\log n)$ amortized time, supports every other operation in $O(1)$ worst-case time, and uses $O(n)$ space, where n is the sum of the current string lengths. You need to give the pseudocode for each of the operation, you also need to argue your implementation indeed has the desired time complexity.

Problem 5

Prove that any connected acyclic graph with $n \geq 2$ vertices has at least two vertices with degree 1. Do not use the words "tree" or "leaf", or any well-known properties of trees; your proof should follow entirely from the definitions of "connected" and "acyclic".

Problem 6

The incidence matrix of a directed graph $G = (V, E)$ with no self-loops is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that

$$b_{ij} = \begin{cases} -1 & \text{if edge } j \text{ leaves vertex } i \\ 1 & \text{if edge } j \text{ enters vertex } i \\ 0 & \text{otherwise} \end{cases}$$

Describe what the entries of the matrix product BB^T represent, where B^T is the transpose of B.