Your name
CS 261, Spring 2021
Homework 10
Due Monday, June 7

1. An exercise from week 6 asked you to use binary trees to implement command line completion: finding the unique string, in a data structure representing a set of strings, that starts with a given query string, or detecting the situations that there is no completion or that there is more than one completion. Describe how to perform the same completion query in a compressed trie.

■

2. For a string with n symbols (not counting the dollar sign at the end), the suffix tree has $n + 1$ leaves and anywhere from 1 to n non-leaf nodes. For instance, the example of "Mississippi" in the lecture notes has $n = 11$, 12 leaves, and 7 internal nodes.

(a) Find a string with $n = 4$ whose suffix tree has only one internal node. Show its suffix tree.

(b) Find a string with $n = 4$ whose suffix tree has exactly four internal nodes. Show its suffix tree.

■

3. Suppose that we want to make the union-find data structure partially persistent. We cannot use the path compression analysis, because that uses amortization, which does not work well with persistence. And we cannot directly use path copying persistence, because there are too many starting points for query paths (one for each different set element).

(a) If we use union by size (without path copying) and the fat node technique for partial persistence, how much storage space is required per union? What is the query time in the resulting persistent structure?

(b) Instead of using fat nodes, it is possible to obtain a partially persistent structure by using union by size (without path copying), keeping only one parent pointer per node (non-persistently), and recording for each node a timestamp of the update that first changed its parent pointer from None to another parent. Describe how to perform a find query in this structure. Your query should take as arguments the element on which to perform the find, and the timestamp of a version of the structure to be queried.

■

4. As the lecture notes describe, the union-find structure can be used for maintaining the connected components in an incremental dynamic graph problem, and to answer queries that ask which component contains a given vertex, as used in Kruskal's algorithm for minimum spanning trees. A variation of Kruskal's algorithm, which can be used for finding minimum spanning pseudoforests (see https://en.wikipedia.org/wiki/Pseudoforest), needs to know an additional piece of information about each component: is it a tree, or does it contain a cycle?

Suppose that we augment the union-find structure with an additional Boolean variable at each root of a union-find tree: True if that root corresponds to a component that is a tree, False otherwise. Describe how to update these variables when inserting an edge into the dynamic graph. (Your answer should handle both the case that the new edge connects two previously-disconnected components, and the case that both of its endpoints already belong to the same component. You may assume that it does not have the same two endpoints as any previously-inserted edge.)

■