Due: Tuesday, March 14, 2023, 11:59pm

Submit online at (clickable link): `https://forms.gle/dvH2dCSY3QVgcBrM8`

1 Frequent Consolidation

In Fibonacci Heaps, the consolidate procedure is called only during ExtractMin(). That makes the data structure rather "messy". In this problem, we explore some ways that might make Fibonacci Heaps tidier. Throughout, we will keep the same invariant as before: \$1 credit on each root in the root list and \$2 on each marked node.

1. Recall that in the standard implementation of Fibonacci Heaps, when we insert an item x, we just make x its own heap (with a single item) and add that heap to the root list.

   Suppose that we modify the Insert operation and have it call the consolidate procedure after each new item is added to the root list. Explain why we cannot maintain the invariant and still have Insert run in $\Theta(1)$ amortized time.

2. The astute observer would correctly claim that the previous question is silly because if we consolidate after every insertion, then the roots in the root list would always have unique degree. If we keep the roots sorted by degree in the root list (starting with degree 0), then we can easily combine the newly inserted x with the heaps already in the root list. The process will be very similar to the ripple-carry adder in the binary counter example for amortized analysis.

   For example, if the root list has roots with degree 0, 1, 2, 4 and 7, then we fist merge the new heap (which has degree 0) with the degree 0 root. This makes a root with degree 1, which we merge with the existing root with degree 1. That makes a root with degree 2 which gets merged and forms a root with degree 3. After that we stop merging and the resulting root list has roots with degree 3, 4 and 7. We can just use the \$1 stored at each root to pay for the merges, add \$1 to the new root and Insert would still run in $\Theta(1)$ amortized time.

   Explain what happens to the amortized running times of Extract-Min, Decrease-Key and Merge operations under such a scheme.

3. Another variation of the previous scheme is to keep an array `Root[]` indexed by degree so that `Root[i]` points to the unique heap where the root has degree i (if one exists). That is, we keep an array of roots instead of a list of roots. Does the binary counter scheme still work to keep Insertion at $\Theta(1)$ amortized running time? Explain.

   Explain what happens to the amortized running times of Extract-Min, Decrease-Key and Merge operations under our new scheme.

2 All Links before Finds

Suppose that we use the disjoint-set union data structure for a sequence of m Make-Set, Find-Set and Link operations, but we are told that all of the Link operations are performed before any Find-Set operation. (As usual, assume that there are n Make-Set operations.) Furthermore, suppose that we implement the data structure using both Union-by-Rank and Path Compression. Show that the total actual running time for all m operations is $O(m)$.

Note: Recall that the Link operation is given the two roots of the trees to be merged. Thus, you do not need to perform any Find-Set operations to execute a Link. (This is just an exercise. It would be unclear how an application using this data structure would know which items are roots.)

Hint: Think about paying for the path compression during a Find-Set.

Note: There is no limit on the number of Find-Set operations. While we do know that m is the total number of operations (including Find-Sets), you cannot assume a relationship between m and the number of items n. I.e., the number of Find-Set operations could greatly exceed the number of items n.

3 Offline Minimum

Do Problem 19-1 in CLRS (Problem 21-1 in the third edition), except in part a use the following sequence of operations:

4 6 E 3 1 2 E 7 E 5 E 9 E 8 E E

Note: The premise of this problem is that you can turn the extract-min question around when the problem is offline. Instead of asking

Which item will this call to Extract-Min return?

you ask

Which Extract-Min call will return this item?

For example, if the item in question is 1 (the smallest item), then it will be returned by some call to Extract-Min (unless 1 is inserted after the last call to Extract-Min). So, the question is which call will return 1.