Assignment 4: Binomial and Fibonacci Heaps

This problem set explores binomial heaps, Fibonacci heaps, and their variants. We hope that it solidifies your understanding of the mechanics and theory behind them.

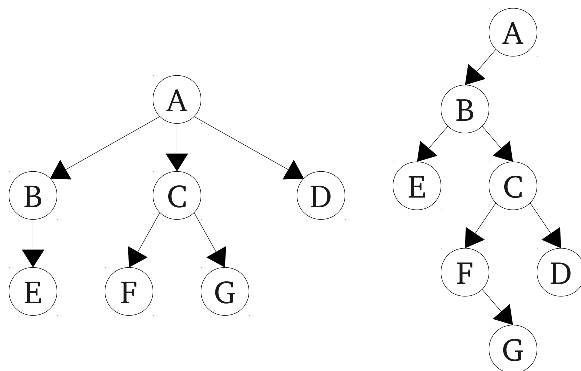
Working in Pairs

We suggest working on this problem set in pairs. If you work in a pair, you should jointly submit a single assignment, which will be graded out of 10 points. If you work individually, the problem set will be graded out of 9 points, but we will not award extra credit if you earn more than 9 points. Note that the point totals have been scaled back relative to the first few problem sets, so each point here is worth more than on earlier problem sets.

Due Wednesday, April 30 at 2:15PM at the start of lecture.

Problem One: Implementing Lazy Binomial Heaps (3 Points)

In this problem, you'll implement a lazy binomial heap to familiarize yourself with its operation and to explore a few implementation details we glossed over in lecture.



When implementing the lazy binomial heap, we'd like you to represent binomial trees using the *left-child/right-sibling* (*LCRS*) representation, which encodes a multiway tree as a binary tree. In LCRS, each node has two pointers: a *left* pointer storing a pointer to its first child, and a *right* pointer storing a pointer to its next sibling. If the node has no children, the *left* pointer is null, and if the node is the last child of its parent, then its *right* pointer is null. A sample multiway tree and its LCRS representation is shown to the left.

We've provided Java starter files for this programming question at `/usr/class/cs166/assignments/ps4/`. Your job is to implement the `LazyBinomialHeap` type defined in `LazyBinomialHeap.java` such that each binomial tree is encoded as a binary tree using the LCRS representation. The root nodes of the trees in the heap should be stored in a doubly-linked list, ***meld***s should be done by concatenating the root lists together, and trees should only be coalesced during an ***extract-min***. All operations should run in amortized time O(1), except for ***extract-min***, which should run in amortized time O(log n).

Lazy binomial heaps require the roots of the trees to be stored in a doubly-linked list. Unfortunately, the Java `LinkedList` type is insufficient here, as you cannot concatenate `LinkedList`s in time O(1). We *strongly* recommend writing your own custom linked list type to use while implementing `LazyBinomialHeap`. From our own experience, factoring this code out of `LazyBinomialHeap` will save you a lot of time debugging!

Problem Two: Palos Altos (1 Point)

Prove that for any positive integers k and m, there is a series of operations that, starting with an empty Fibonacci heap, causes the heap to have a tree of order k containing at least m nodes. This shows that there is no upper bound on the number of nodes in a tree in a Fibonacci heap.

Problem Three: Meldable Heaps with Addition (5 Points)

Meldable priority queues support the following operations:

- ***new-pq***(), which constructs a new, empty priority queue;
- *pq*.***insert***(v, k), which inserts element v with key k;
- *pq*.***find-min***(), which returns an element with the least key;
- *pq*.***extract-min***(), which removes and returns an element with the least key;
- ***meld***(pq_1, pq_2), which destructively modifies priority queues pq_1 and pq_2 and produces a single priority queue containing all the elements and keys from pq_1 and pq_2.

Some graph algorithms, such as the Chu-Liu-Edmonds algorithm for finding minimum spanning trees in *directed* graphs, also require the following operation:

- *pq*.***add-to-all***(Δk), which adds Δk to the keys of each element in the priority queue.

Using lazy binomial heaps as a starting point, design a data structure that supports all ***new-pq***, ***insert***, ***find-min***, ***meld***, and ***add-to-all*** in amortized time O(1) and ***extract-min*** in amortized time O(log n). Some hints:

1. You may find it useful, as a warmup, to get all these operations to run in time O(log n) by starting with an *eager* binomial heap and making appropriate modifications. You may end up using some of the techniques you develop in your overall structure.
2. Try to make all operations have worst-case runtime O(1) except for ***extract-min***. Your implementation of ***extract-min*** will probably do a lot of work, but if you've set it up correctly the amortized cost will only be O(log n). This means, in particular, that you will only propagate the Δk's through the data structure in ***extract-min***.
3. If you only propagate Δk's during an ***extract-min*** as we suggest, you'll run into some challenges trying to ***meld*** two lazy binomial heaps with different Δk's. To address this, we recommend that you change how ***meld*** is done to be even lazier than the lazy approach we discussed in class. You might find it useful to construct a separate data structure tracking the ***meld***s that have been done and then only actually combining together the heaps during an ***extract-min***.
4. To get the proper amortized time bound for ***extract-min***, you will probably need to define a potential function both in terms of the structure of the lazy binomial heaps and in terms of the auxiliary data structure hinted at by the previous point.

Problem Four: Course Feedback (1 Point)

We want this course to be as good as it can be and would really appreciate your feedback on how we're doing. For a free point, please take a few minutes to answer the course feedback questions available at https://docs.google.com/forms/d/1pn_tXQYwrcGe6DfF1dozp1x0YMoR0K6P6cj5dPRcN1I/viewform. If you are submitting in a group, **please have each group member fill this out individually**.