Exercise  (Lower-bounding heap operations)

Theorem Suppose only comparisons are allowed on heap keys.

Then for all heap implementations,

\[ \text{Insert or (Extract and (Delete or Minimum))} \]

must take \( \Omega (\log n) \) amortized time on a heap of \( n \) elements.

Proof Suppose not, i.e. that there is a comparison-based heap for which

\[ \text{Insert and (Extract or (Delete and Minimum))} \]

take \( o(\log n) \) amortized time. Then given an array \( A[1:n] \), at least one of the following two algorithms,

\[
\begin{align*}
H := \text{Heap} \\
&\quad \text{for } i := 1 \text{ to } n \text{ do} \\
&\qquad \text{Insert} \ (A[i], A[i], H) \\
&\text{for } i := 1 \text{ to } n \\
&\qquad A[i] := \text{Extract} \ (H) \\
\end{align*}
\]

and

\[
\begin{align*}
H := \text{Heap} () \\
&\quad \text{for } i := 1 \text{ to } n \text{ do} \\
&\qquad P[i] := \text{Insert} \ (A[i], i, H) \\
&\text{for } i := 1 \text{ to } n \text{ do} \begin{align*}
&\qquad j := \text{Minimum} \ (H) \\
&\qquad A[i] := \text{Key} \ [P[j]] \\
&\text{Delete} \ (P[j], H) \\
&\end{align*} \\
\end{align*}
\]

Sorts \( A \) in time \( \Theta(n) \) or \( (\log n) = o(n \log n) \), which is impossible, since any comparison-based sorting algorithm must use \( \Omega(n \log n) \) time.
**Problem (Structure of Fibonacci heaps, II)** Prove the following.

**Definition** The Fibonacci tree $T_k$, for $k \geq 0$, is

$$T_k := \begin{cases} 
0, & k=0; \\
5, & k=1; \\
\text{rooted}, & k \geq 2.
\end{cases}$$

**Lemma** For any Fibonacci heap $H$ and any $k \geq 0$, one can construct $H$ so that the Fibonacci tree $T_k$ is rooted at any specified node of $H$.

**Proof** We first show how to construct $T_k$ by Fibonacci heap operations, using induction on $k$. For $k=0$, we do a single insert. For $k=1$, we do three inserts followed by one Extract. Thus the basis holds.

For $k \geq 2$, we:

(a) recursively (using the induction hypothesis)
construct three heaps: $H_1 = T_{k-1}$, $H_2 = T_{k-1}$, $H_3 = T_0$;

(b) Union $H_1$, $H_2$, $H_3$ together to form $H$;

(c) Delete the single node of $H_3$ in $H$,
yielding

```
          s
         / \  \
        T_{k-1} T_{k-1}
       /   \     /
      H_1   H_2  H_3
```

(d) Delete every node in the subtree rooted at
the first child of $s$. 
By definition, $H_1$ has the form

So Step (d) converts $H_1$ into $T_{k-2}$ and thus $H$ into $T_k$.

Finally, to have $T_k$ rooted at any specified node $v$ of an arbitrary heap $H$, let $S$ be a sequence of Fibonacci heap operations that constructs $H$ except for the subtree rooted at $v$. At the point in $S$ at which $v$ is created, construct a separate heap $H' = T_k$ whose root has the same key as $v$, and union $H'$ with the heap into which $v$ was inserted. The remainder of $S$ will construct $H$ with $T_k$ rooted at $v$. 

\[ \square \]
Problem (Fibonacci heaps on a pointer machine)

Show how to implement a Fibonacci heap on a pointer machine (i.e. on a random-access machine without arrays), so that all operations (in particular Extract) take the same amortized time.

Solution sketch

To efficiently perform Consolidate during Extract without using an array, maintain a linked list encoding the degree of a node:

- Every node in the heap has a pointer to the corresponding degree list. Since a node's degree changes by ≤1 during an operation, these pointers are easily maintained. Finding a root of the same degree is now easy during Consolidate.
- Also maintain $d$, the maximum degree of a node in the heap.
- On a Union of two heaps, merge the smaller of the degree lists into the larger.
- Add a term of $+d$ to $\Phi$, the potential function, to cancel out the merge time, so that a Union still takes $O(1)$ amortized time.
Problem (Path and cycle cover)

Given a directed, edge-weighted graph $G = (V, E, w)$, find a maximum-weight path and cycle cover:

* a subset $C \subseteq E$ s.t. in $(v, c)$ every vertex has
  in-degree $\leq 1$ and out-degree $\leq 1$ and $\sum w(c)$ is maximum.

Algorithm

To given graph $G$, we construct an undirected bipartite graph $\tilde{G} = (\tilde{V}, \tilde{E}, \tilde{w})$ as follows:

- Each vertex $v \in V$ is mapped to two vertices $v^{\text{in}}$, $v^{\text{out}}$.

Each edge $(v, w) \in E$ is mapped to an undirected edge $(v^{\text{out}}, w^{\text{in}}) \in \tilde{E}$.

Notice $\tilde{G}$ is bipartite:

For a maximum matching $M \subseteq \tilde{E}$ corresponds to a path and cycle cover in $G$.

Each in-vertex in $M$ is

* oriented by at most one edge in $M$, and

* each out-vertex in $M$.

This property implies the in- and out-degree constraints in a path and cycle cover.

So a maximum-weight bipartite matching in $\tilde{G}$ yields a maximum-weight path and cycle cover in $G$. 