CSE548, AMS542: Analysis of Algorithms, Fall 2023 **Date:** Nov 10

Homework #3

(Due: Nov 23)

Task 1. [200 Points] The Diamond Heap

In this task, we will construct and analyze the *Diamond Heap*. Like the binomial heap we discussed in the class, a diamond heap will support Make-Heap, Insert, Minimum, Extract-Min, and Union efficiently, but unlike the binomial heap, it will also offer efficient support for Maximum and Extract-Max.

A diamond heap is a set of diamond pendants along with a couple of loose diamonds. We define diamonds[1] and pendants below.

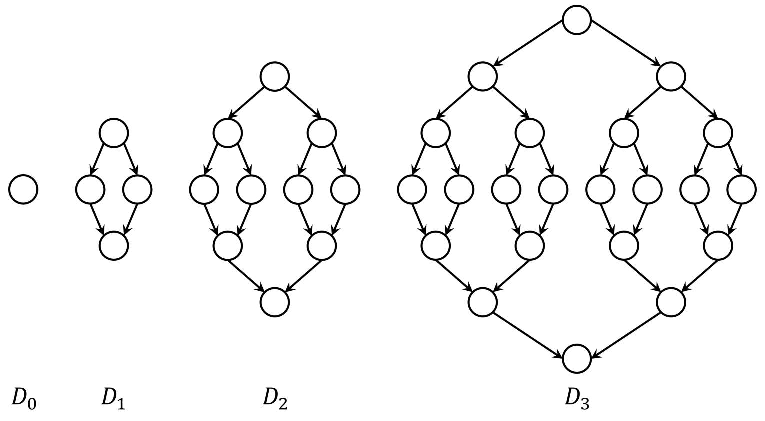

Figure 1: Diamonds D_k for $0 \leq k \leq 3$.

A *diamond* of *rank k* is denoted by D_k. Figure 1 shows D_k's for $0 \leq k \leq 3$. A D_0 is a single node. As shown in Figure 2, for $k \geq 0$, a D_{k+1} is constructed by connecting the top nodes of two D_k's using an additional node, and their bottom nodes using another new node.

A *pendant* P_k is constructed from a D_k by adding an extra node as the parent of its top node and another extra node as the child of its bottom node (see Figure 2). Figure 3 shows P_k's for $0 \leq k \leq 3$.

A diamond heap H is a collection of pendants and a few loose diamonds (i.e., D_0's) that satisfy the following properties.

- Each node has a key.
- Each pendant satisfies the min-heap property, i.e., the parent of a node x cannot have a key larger than that of x. In the figures, the parent-child relationship is shown using arrows. If there is an arrow pointing from node x to node y then x is y's parent.

[1] i.e., diamond-shaped structures

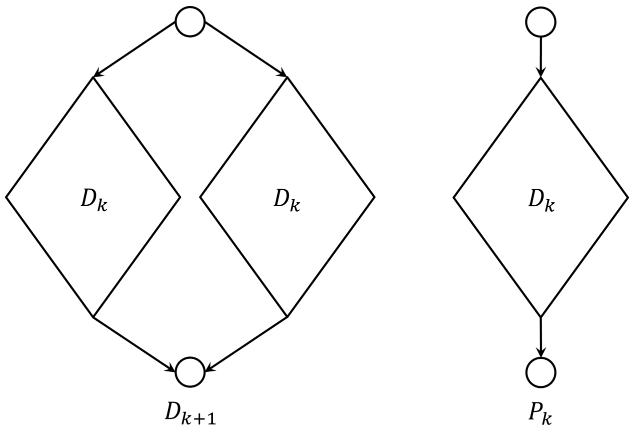

Figure 2: A D_{k+1} is constructed from two D_k's and two additional nodes. A P_k is constructed from one D_k and two additional nodes.

- For every integer $k \geq 0$, there is at most one P_k in H.
- There are at most two D_0's in H.

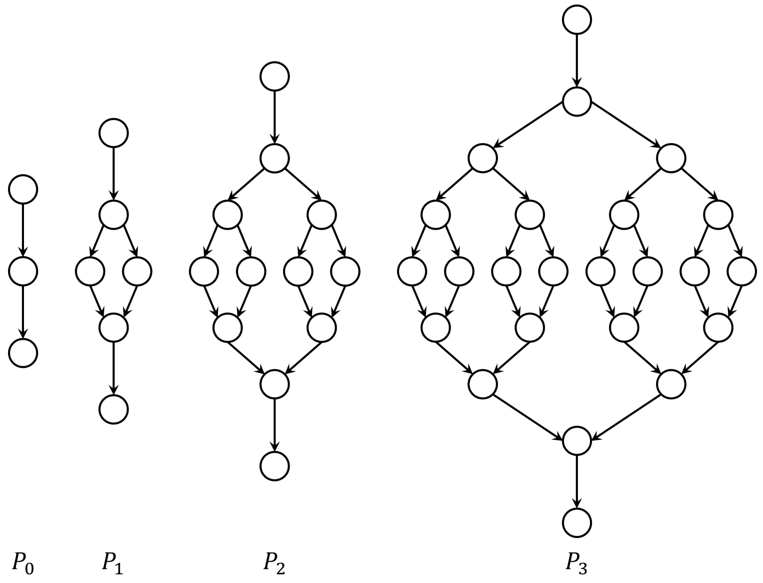

Figure 3: Pendants P_k for $0 \leq k \leq 3$.

(a) [**10 Points**] How do you efficiently merge two min-heap-ordered P_k's to produce a min-heap-ordered P_{k+1}? What is the complexity of this merge operation?

(b) [**30 Points**] Suppose that you want to support only MAKE-HEAP, MINIMUM, MAXIMUM, INSERT, and UNION under eager union (i.e., using the array of pointers version of the data structure). Show that the amortized cost of each operation will be $\mathcal{O}(1)$ except for UNION which will be $\mathcal{O}(\log n)$, where n is the number of items currently in the data structure.

(c) [**30 Points**] Extend part (b) to support EXTRACT-MIN and EXTRACT-MAX without changing the amortized complexities of the operations the data structure already supports. Show that the two new operations can be supported in $\mathcal{O}\left(\log^2 n\right)$ amortized cost each.

(d) [**50 Points**] Suppose that the data structure will never contain more than N items during its entire lifetime and that you are willing to support MAKE-HEAP, INSERT and UNION operations in $\mathcal{O}\left(\log N\right)$ amortized cost each. Now, re-analyze your data structure from part (c) to show that all other operations including EXTRACT-MIN and EXTRACT-MAX can be supported in $\mathcal{O}\left(1\right)$ amortized cost each.

Important: Do not prove part (d) *as a corollary of part* (e)*. This task asks you to prove part* (d) *first and then extend it to part* (e)*.*

(e) [**30 Points**] Extend your analysis from part (d) to show that the bounds proved in that part continue to hold even if N represents the number of items currently in the data structure (i.e., $N = n$) provided the size of at least one of the heaps given as input to the UNION operation is upper-bounded by a constant. What amortized bound do you get for UNION if there is no restriction on the sizes of the input heaps, but you want to keep the amortized bounds of all other operations unchanged?

(f) [**30 Points**] How do the bounds for the diamond heap operations you proved in part (e) change if you use lazy union (i.e., using the doubly linked list version of the data structure) instead of eager union?

(g) [**20 Points**] Analyze the amortized complexities of DECREASE-KEY, INCREASE-KEY, and DELETE operations for your data structure from part (f). Does any of those bounds hold in the worst case?