Data Structures and Algorithms

Lab 5 on Priority Queues (Heaps)

ENSIA 2023-2024

Objectives

- Efficient implementation of the priority queue ADT.
- Uses of priority queues.
- Advanced implementations of priority queues.

Prerequisites

C++ Classes (1.4), C++ Details (1.5), and Template (1.6) from the course textbook[1].

Exercise 1

Write a program to take N elements and do the following:

- Insert the elements into a heap one by one.
- Build a heap in linear time.

Compare the running time of both algorithms for sorted, reverse-ordered, and random input.

Exercise 2

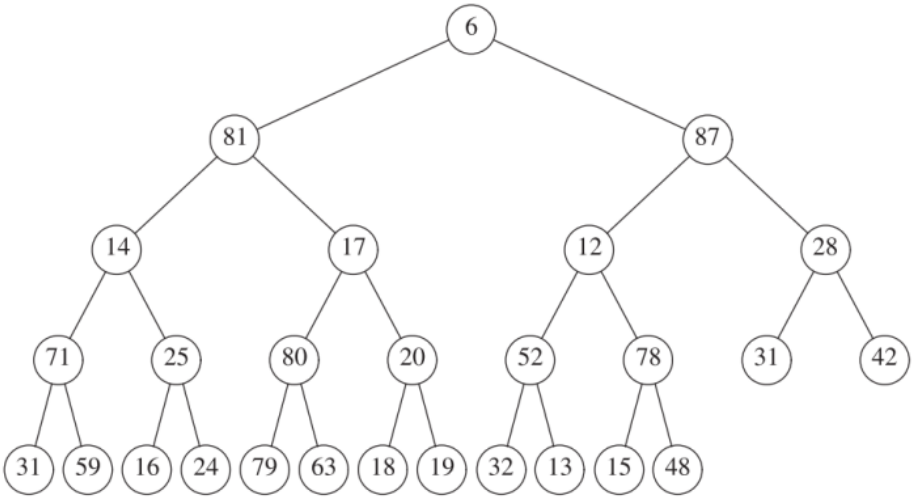
Rewrite the `BinaryHeap` insert function by placing a copy of the inserted item in position 0.

Exercise 3

- Write a program to find all nodes less than some value X in a binary heap. Your algorithm should run in $O(K)$, where K is the number of nodes output.
- Does your algorithm extend to any of the other heap structures discussed in this chapter?

[1] Data Structures and Algorithm Analysis in C++, Fourth Edition, Mark Allen Weiss

Figure 1: Min-max heap



Exercise 4

A min-max heap is a data structure that supports both `deleteMin` and `deleteMax` in $O(\log N)$. The structure is identical to a binary heap, but the heap order property ensure the following:

"First, for any node X at even depth, the element stored at X is smaller than the parent but larger than the grandparent (where this makes sense). Second, for any node X at odd depth, the element stored at X is larger than the parent but smaller than the grandparent (see Figure 1)".

- How do we find the minimum and maximum elements?
- Give an algorithm to insert a new node into the min-max heap.
- Give an algorithm to perform `deleteMin` and `deleteMax`.
- Can you build a min-max heap in linear time?
- Let's suppose we would like to support the operations `deleteMin`, `deleteMax`, and `merge`. Propose a data structure to support all operations in $O(\log N)$ time.

Exercise 5

Write a program to build a binomial queue of N elements using at most $N-1$ comparisons between elements.

Exercise 6

Write an efficient function to perform `insert` using binomial queues. Do not call the `merge` function.

Exercise 7

Modify the `merge` function of binomial queue:

- To terminate merging if there are no trees left in H_2 and the `carry` tree is `nullptr`.
- To ensure that the smaller tree is always merged into the larger.

Exercise 8

Let's suppose you have a number of boxes, each of which can hold a total weight C and items $i_1, i_2, i_3, \ldots, i_N$ whose weights are $w_1, w_2, w_3, \ldots, w_N$ respectively. The aim is to pack all the items without placing more weight in any box than its capacity, while using as few boxes as possible. For instance, if $C = 5$ and the items have weights $2, 2, 3, 3$, then we can solve the problem with two boxes.

In general, this problem is very hard, and no efficient solution is known. Now, write programs to implement efficiently the following approximation strategies:

- Place the weight in the first box for which it fits (create a new box if there is no box with enough room). This strategy and all that follow would give three boxes, which is suboptimal.
- Place the weight in the box with the most room for it.
- Place the weight in the most filled box that can accept it without overflowing.
- Are any of these strategies enhanced by presorting the items by weight?