Amortized Complexity

Jin Long Cao

November 2022

$L = [x_1, x_2, \ldots, x_n]$ is a list of items, each with a distinct min-heap key, i.e. $x_i.p$ has key p. We want a bound on the worst-case sequence run-time complexity of inserting the items from L into binomial heap H using the Binomial-Heap-Insert algorithm below. Answer parts (a) and (b) using the indicated method.

```
Binomial-Heap-Insert(H, x)
1   p[x] ← NIL                          ▷ make a new B_0
2   child[x] ← NIL
3   degree[x] ← 0
4   sibling[x] ← head[H]                ▷ insert new B_0 at head of list of B_i s
5   head[H] ← x
6   cur_node ← head[H]                  ▷ prepare to walk the list
7   next_node ← sibling[head[H]]
8   while next_node ≠ NIL and degree[cur_node] = degree[next_node]
9       do
            if key[cur_node] ≤ key[next_node]          ▷ min-heap order
10              sibling[cur_node] ← sibling[next_node]
11              Binomial-Link(next_node, cur_node)     ▷ cur_node now parent of next_node
12          else
13              head[H] ← next_node
14              Binomial-Link(cur_node, next_node)     ▷ next_node now parent of cur_node
15              cur_node ← next_node
16      next_node ← sibling[cur_node]       ▷ check if next two trees must link
17  return H                ▷ all done linking binomial trees
```

(a) The aggregate method from CLRS. Show your work.

(b) The accounting method from CLRS. Show your work.

Solutions

Let $L = [x_1, x_2, \ldots, x_n]$ be a list of items, each with a distinct priorities ($x_i.p$ has priority p). Let $k \in \mathbb{N}$. When inserting into a binomial heap, we need to consider two things.

- MAKE-HEAP(x): return a new heap containing only element x.
- UNION(H_1, H_2): return a new heap containing all elements of heap H_1 and H_2.
  - ① First merge H_1 and H_2 into H' with binomial trees in ascending order (based on priority).
  - ② If H' contains 2 B_k adjacent to each other, call them B_{k_1} and B_{k_2}, then merge them into a B_{k+1}.
  - ③ If there are now 3 B_{k+1} adjacent then we merge the second and third B_{k+1}.

Note: ② may run multiple times until there aren't anymore pairs of B_k.
For example, consider a binomial heap H_1 with binomial trees B_0, B_1, B_2. if we insert an element, MAKE-HEAP(x) would creates a new heap with single node B_0, then UNION(H_1, B_0) would merge the two B_0 into a B_1 (② ran once), but now we have two B_1 which will run ② again resulting in another binomial heap with two B_2 which will trigger ② to run again. In this example, ② ran 3 times from inserting one element.

(a) Aggregate Method

Notice that ② occurs every other time you insert something. Every time you insert something, you create a single node (B_0) and then you add it with the rest of your binomial heap. That means every other time you insert something, you'll have two B_0 and ② will run (at least once). Also,

- Every fourth insert, we'll have two B_1 and ② will have to run again.
- Every eighth insert, we'll have two B_2 and ② will have to run again.
  ⋮
- Every 2^k-th insert, we'll have two B_{k-1} and ② will have to run again.

In a sequence of n operations: MAKE-HEAP(x) and ① always run **($2n$)**, ② will run at least once every other time **($\frac{n}{2}$)** to combine the two B_0. But in total, ② will run in the sequence of $\sum_{i=1}^{k} \lfloor \frac{n}{2^i} \rfloor$. Since we're starting with an empty binomial heap and inserting one element at a time, there won't be any possibility that 3 B_{k+1} are adjacent such that ③ will run zero times **(0)**. Putting it all together, the total number of times each operation runs for n insertion is

$$\begin{aligned} 2n + \frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{2^k} &= 2n + n(1 - (\frac{1}{2})^k) \\ &= 2n + n - n(\frac{1}{2})^k \\ &< 3n \end{aligned}$$

Using aggregate method, we can conclude that the total cost of the sequence is $O(n)$ and the amortized cost per operation is $\frac{O(n)}{n} = O(1)$.

(b) Accounting Method:

Notice that if a Binomial heap have $2^k - 1$ nodes (with k binomial trees) and we insert once. Then the binomial heap will have 2^k nodes which means it should have only one binomial tree (B_k). To get one binomial tree, UNION will merge k times.

Total nodes	number of merges*	B_i
1	0	B_0
2	1	B_1
3	0	B_0, B_1
4	2	B_2
5	0	B_0, B_2
6	1	B_1, B_2
7	0	B_0, B_1, B_2
8	3	B_3
$\vdots$	$\vdots$	$\vdots$
n-1	$\leq \lg(n-1)$	
n	$\leq \lg n$	

*not including the first merge ①

Note: x represents rounding to the closest integer to x

For MAKE-HEAP(x), it cost 1 to create a singleton heap and it cost another 1 to store an item from L into a binomial heap. Then the amortized cost $= 1 + 1 = 2 = \theta(1)$.

For UNION(H_1, H_2), it cost 1 to just merge H_1 and H_2 and we would like to save $\lg n$ credit in the bank so that we can merge if there are multiple B_k adjacent. From the table above, we know that it will cost less than $\lg n$ so we'll always have enough credit. Then the amortized cost $= 1 + \lg n = \theta(\lg n)$

Combining it together, we now know that the amortized cost of insert $= 2 + 1 + \lg n = \theta(\lg n)$ and can conclude that the bound on the worst-case run-time complexity of inserting the items from L into a binomial heap is $\theta(\lg n)$