Homework 5

Hu Jiarun He Zhanhao Lu Keyu Zhou Yang

March 20, 2021

Contents

1 Exercise 1

1.1 exercise (i)

For $Change\_Key(H, x, k)$, we consider two operations:

$$\begin{cases} Decrease(H, x, k) & \text{if } k > x.key \\ Increase(H, x, k) & \text{if } k < x.key \\ Undefined & \text{if } k = x.key \end{cases}$$

For $Decrease(H, x, k)$, we need a cut operation: $Cut(x, parent)$, which cut the node whose key we decrease from its parent and add it to the root lists. And if the node is not an single node, but a root of a subtree. Then we cut the entire subtree and add it to the root lists.
We use operation $CascadedCut(parent)$ for its parent node $parent$.

i. if $parent.marked = False$ then $parent.marked = True$.

ii. else $Cut(parent, grandparent)$ and $CascadedCut(grandparent)$

And for $Increase(H, x, k)$

i. assume x is the child that we want to change its key. And add left child and its siblings to the root lists

ii. $CascadedCut(x)$ and add x node to the root lists

Then we start our amortised complexity analysis. We have known that the potential of an Fibonacci Heap is:

$$\Phi(H) = t(H) + 2m(H)$$

Then we can get the amortised time complexity for $Decrease(H, x, r)$ is:

$$\phi(H') = \phi(H) + actual\_coast = t(H') - t(H) + 2(m(H') - m(H)) + actual\_coast$$

Then we analyse $CascadedCut(H, x, r)$. Assuming there is c cascading cuts, the actual cost is decrease the number, increase c times, so it is $O(c)$
Thus,we can determine the number of new trees is c

$$t(H') - t(H) = c$$

and c-1 marked nodes become unmark.

$$(m(H') - m(H)) + actual\_cost = c - 1$$

Then the amortised time complexity is $O(c + c - 2(c-1)) = O(1)$
For the $Increase(H, x, k)$, the first case is the same as the *delete* operation, as lecture metions the amortised time complexity is $O(log(n))$
And after *delete*, we add the x node to root lists. Then the amortised complexity time $O(log(n) + 1) = O(log(n))$

1.2 exercise (ii)

Suppose that the additional cost to the potential function that was proportional to the size of the heap. Because the size only increase when we do an insertion, and then only by a constant amount. Thus we don't need to consider this increased potential function raising the amortised cost of any operations. Then we modify the heap itself by having a doubly linked list along all of the leaf nodes in the heap.
Then for $Prune(H, r)$ we can pick any leaf node, remove it from it parents child list. This causes the potential decrease by the amount which is proportional to r. The actual cost of ehat just happened since the deletions from the linked list take only constant time. So it is $O(c)$

2 Exercise 2

2.1 exercise (i)

See the code for implementation.

2.2 exercise (ii)

See the code for implementation.

3 Exercise 3

For a single item in represented set, we can throw them to the box randomly. Thus the possiblity is $\frac{1}{m}$. Then the possiblity that the item doesn't in the box is

$$1 - \frac{1}{m}$$

So for all n items, the possibility that box is empty is

$$(1 - \frac{1}{m})^n$$

Thus, if $X_i = 1$ if A_i is empty, the expect number is

$$m(1 - \frac{1}{m})^n$$

4 Exercise 4

See the code for implementation