Solutions for Chapter 20

Zhixiang Zhu
zzxiang21cn@hotmail.com

January 25, 2011

Solution to Exercise 20.2-3

A Fibonacci heap remains a collection of unordered binomial trees after any mergeable-heap operation is performed. Therefore, $D(n)$ is at most $\lfloor \lg n \rfloor$.

Solution to Exercise 20.2-4

The inserion and union operations without consolidation both take $O(1)$ time. The consolidation operation takes $O(D(n[H]) + t(H))$ time, H is the input heap of consolidation. Therefore, the new insertion and union operations both take $O(D(n[H]) + t(H))$ actual time, where H is the heap resulted from the insertion or union operation. Since the amortized costs of the original insertion and union operations are already $O(1)$, the new operations do no good to it. Actually they make the actual running time even worse.

Solution to Exercise 20.3-1

If x is a marked child of some root, then x will become a marked root when the parent of x is deleted and x does not become a child of any other node.

Solution to Exercise 20.3-2

Suppose that we consecutively invoke Fib-Heap-Decrease-Key n times on a Fibonacci heap of n elements. As we have known, the only non-constant time operation in Fib-Heap-Decrease-Key is Cascading-Cut. Each recursive Cascading-Cut (those which are neither the top nor the bottom in the invocation stack) invokes Cut on a node, with the node being the x input of Cut. Therefore, the only non-constant time operation in Fib-Heap-Decrease-Key is actually the multiple Cut procedures. Since the x input node of Cut will become a root and Cut won't be invoked on a root with the root being x, CUT is invoked at most n times in total. Therefore, the total time of the n invocations of FIB-HEAP-DECREASE-KEY is $O(n)$, and the amortized time of FIB-HEAP-DECREASE-KEY is $O(n)/n = O(1)$.

Solution to Exercise 20.4-1

After we create an empty Fibonacci heap, we insert a node into the heap. After the insertion, we perform a sequence of the following operations:

I2, E, I2, E, I2, E, ...

where I2 stands for inserting two nodes whose keys are smaller than the currnet minimum key in the heap, that is, suppose the current minimum key in the heap is A, then we insert two nodes whose keys are B and C, where $B < A$ and $C < A$; E stands for extracting the node with the minimum key.

After $n-1$ I2 and $n-1$ E operations, we will get a Fibonacci heap in which there's only one tree, which is just a linear chain of n nodes.

Solution to Execise 20.4-2

As long as k is a positive integer, we will have $D(n) = O(\lg n)$. The proof is given as follows.

If a node x is cut as soon as it loses its k'th child, then the last sentence of lemma 20.1 in the text will be changed to: "Then, $\mathit{degree}[y_i] \geq 0$ for $i = 1, 2, \ldots, k'-1$ and $\mathit{degree}[y_i] \geq i - k'$ for $i = k', k'+1, \ldots, k$." Therefore, the the first half of the proof of lemma 20.3 will also be changed to

$$
\begin{aligned}
\text{size}(x) &\geq s_k \\
&\geq k' + \sum_{i=k'}^{k} s_{\mathit{degree}[y_i]} \\
&\geq k' + \sum_{i=k'}^{k} s_{i-k'} .
\end{aligned}
$$

(to be continued...)

Solution to Problem 20-1

a. The p fields of x's children should be set to NIL in line 7, which takes $\Theta(\mathit{degree}[x])$ actual time.

b. $O(\mathit{degree}[x] + c)$.

c. The potential of H' is at most $t(H) + c + \mathit{degree}[x] + 2(m(H) - c + 2)$.

d. The change in potential is at most

$$\begin{aligned}&(t(H) + c + \mathit{degree}[x] + 2(m(H) - c + 2)) - (t(H) + 2m(H)) \\ &\qquad = \mathit{degree}[x] + 4 - c.\end{aligned}$$

Solution to Problem 20-2

a. Here's the FIB-HEAP-CHANGE-KEY procedure:

```
FIB-HEAP-CHANGE-KEY(H, x, k)
 1  if k > key[x]
 2     then key[x] ← k
 3          for each child y in the child list of x
 4              do if key[y] < key[x]
 5                    then CUT(H, y, x)
 6                         CASCADING-CUT(H, x)
 7          if x = min[H]
 8            then CONSOLIDATE(H)
 9                 for each node z in the root list of H
10                     do if key[z] < key[min[H]]
11                           then min[H] ← z
12     else FIB-HEAP-DECREASE-KEY(H, x, k)
```

Let H denote the Fibonacci heap just prior to the FIB-HEAP-CHANGE-KEY operation.

Now let's determine the amortised cost of FIB-HEAP-CHANGE-KEY in the case of $k > \mathit{key}[x]$.

If $x = \mathit{min}[H]$, there will be no recursive calls of CASCADING-CUT since $p[x] = \text{NIL}$. Therefore, the **for** loop of lines 3–6 contributes an actual cost of $O(D(n))$, since there're at most $D(n)$ children of x. According to the analysis in section 20.2 of the textbook, the CONSOLIDATE procedure contributes an actual cost $O(D(n)+t(H))$, so as the **for** loop of lines 9–11. Therefore, the actual cost of FIB-HEAP-CHANGE-KEY when $x = \mathit{min}[H]$ is $O(D(n) + t(H))$.

The potential before changing the key is $t(H) + 2m(H)$, and the potential afterward is at most $(D(n) + 1) + 2m(H)$, since at most $D(n) + 1$ roots remain and no nodes become marked during the operation. The amortised cost when $x = \mathit{min}[H]$ is thus at most

$$\begin{aligned}&O(D(n) + t(H)) + ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H)) \\ &\qquad = O(D(n)) + O(t(H)) - t(H) \\ &\qquad = O(D(n)),\end{aligned}$$

since we can scale up the units of potential to dominate the constant hidden in $O(t(H))$.

If $x \neq min[H]$, the only cost occurs in the **for** loop of lines 3–6, in which case the actual cost is $O(D(n) + c)$, where c is the number of times CASCADING-CUT is recursively called.

We next compute the change in potential. Each recursive call of CASCADING-CUT, except for the last one, cuts a marked node and clears the mark bit. The children of x may also be cut. Afterward, there are at most $t(H) + D(n) + c - 1$ trees (the original $t(H)$ trees, $D(n)$ trees produced by cutting x's children, and $c - 1$ trees produced by cascading cuts), and at most $m(H) - c + 2$ marked nodes ($c - 1$ were unmarked by cascading cuts and the last call of CASCADING-CUT may have marked a node). The change in potential is therefore at most

$$((t(H) + D(n) + c - 1) + 2(m(H) - c + 2)) - (t(H) + 2m(H))$$
$$= D(n) + 4 - c$$

Thus, the amortised cost of FIB-HEAP-CHANGE-KEY when $x \neq min[H]$ is at most

$$O(D(n) + c) + D(n) + 4 - c = O(D(n)),$$

since we can scale up the units of potential to dominate the constant hidden in $O(c)$.

Therefore, the amortised cost of FIB-HEAP-CHANGE-KEY for the case in which k is greater than $key[x]$ is $O(D(n))$. Because a node is cut as soon as it loses two children, lemma 20.1 in the textbook still holds, so do lemma 20.3 and corollary 20.4. Thus, the amortised cost of FIB-HEAP-CHANGE-KEY for the case in which k is greater than $key[x]$ is $O(\lg n)$.

According to section 20.3 of the textbook, the amortised cost is $O(1)$ for the case in which k is equal to or less than $key[x]$.

b. We can modify the data structure to maintain a list of leaves. As long as the number of deleted nodes is smaller than $\min(r, n[H])$, we will delete a leaf from the list. Each time a leaf is deleted, the CASCADING-CUT procedure must be called on its parent to keep lemma 20.1 in the textbook true, so that the amortised running time of any other Fibonacci-heap operations won't be changed. Also, each time a leaf is added or deleted, the leaf list must be updated. The potential function doesn't need to be modified. The amortised cost of FIB-HEAP-PRUNE(H, r) is $O(\min(r, n[H]))$.