CLRS 19 Fibonacci Heaps

Introduction to Algorithms, Third Edition

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
Mergeable Heaps

A mergeable heap is any data structure that supports the following five operations, (each element has a key):

- **MAKE-HEAP()** creates and returns a new heap containing no elements.
- **INSERT**(H, x) inserts element x, whose key has already been filled in, into heap H.
- **MINIMUM**(H) returns a pointer to the element in heap H whose key is minimum.
- **EXTRACT-MIN**(H) deletes from H the element whose key is minimum, returning a pointer to the element.
- **UNION**(H₁, H₂) creates and returns a new heap that contains all the elements of heaps H₁ and H₂. Heaps H₁ and H₂ are "destroyed" by this operation.

In addition to the mergeable-heap operations above, Fibonacci heaps support the following two operations:

- **DECREASE-KEY**(H, x, k) assigns to element x within heap H the new key value k, which we assume to be no greater than its current key value.
- **DELETE**(H, x) deletes element x from heap H.
<table>
<thead>
<tr>
<th>Procedure</th>
<th>Binary heap (worst-case)</th>
<th>Fibonacci heap (amortized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAKE-HEAP</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>INSERT</td>
<td>$\Theta(\lg n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>EXTRACT-MIN</td>
<td>$\Theta(\lg n)$</td>
<td>$O(\lg n)$</td>
</tr>
<tr>
<td>UNION</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>DECREASE-KEY</td>
<td>$\Theta(\lg n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>DELETE</td>
<td>$\Theta(\lg n)$</td>
<td>$O(\lg n)$</td>
</tr>
</tbody>
</table>
Fibonacci heaps
Fibonacci heaps — Insertion

FIB-HEAP-INSERT($H, x$)

1 $x$.$degree = 0$
2 $x$.$p = NIL$
3 $x$.$child = NIL$
4 $x$.$mark = FALSE$
5 if $H$.$min == NIL$
6 create a root list for $H$ containing just $x$
7 $H$.$min = x$
8 else insert $x$ into $H$'s root list
9 if $x$.$key < $H$.$min$.$key$
10 $H$.$min = x$
11 $H$.$n = H$.$n + 1$
Fibonacci heaps — Insertion
FIB-HEAP-UNION($H_1$, $H_2$)

1. $H = \text{MAKE-FIB-HEAP}()$
2. $H.min = H_1.min$
3. concatenate the root list of $H_2$ with the root list of $H$
4. if ($H_1.min == \text{NIL}$) or
   - ($H_2.min \neq \text{NIL}$ and $H_2.min.key < H_1.min.key$)
5. $H.min = H_2.min$
6. $H.n = H_1.n + H_2.n$
7. return $H$
Fibonacci heaps — Extract Minimum

FIB-HEAP-EXTRACT-MIN($H$)

1. $z = H.min$
2. if $z \neq$ NIL
3. for each child $x$ of $z$
4. add $x$ to the root list of $H$
5. $x.p = NIL$
6. remove $z$ from the root list of $H$
7. if $z == z.right$
8. $H.min = NIL$
9. else $H.min = z.right$
10. CONSOLIDATE($H$)
11. $H.n = H.n - 1$
12. return $z$
Fibonacci heaps — Extract Minimum

Figure 19.4: The action of FIB-HEAP-EXTRACT-MIN.

(a) A Fibonacci heap $H$.

(b) The situation after removing the minimum node $z$ from the root list and adding its children to the root list.
Fibonacci heaps — Consolidate

Lines 1–3 allocate and initialize the array $A$ by making each entry NIL.

The for loop of lines 4–14 processes each root $w$ in the root list. As we link roots together, $w$ may be linked to some other node and no longer be a root. Nevertheless, $w$ is always in a tree rooted at some node $x$, which may or may not be $w$ itself. Because we want at most one root with each degree, we look in the array $A$ to see whether it contains a root $y$ with the same degree as $x$. If it does, then we link the roots $x$ and $y$ but guaranteeing that $x$ remains a root after linking. That is, we link $y$ to $x$ after first exchanging the pointers to the two roots if $y$'s key is smaller than $x$'s key. After we link $y$ to $x$, the degree of $x$ has increased by 1, and so we continue this process, linking $x$ and another root whose degree equals $x$'s new degree, until no other root that we have processed has the same degree as $x$.

We then set the appropriate entry of $A$ to point to $x$, so that as we process roots later on, we have recorded that $x$ is the unique root of its degree that we have already processed. When this for loop terminates, at most one root of each degree will remain, and the array $A$ will point to each remaining root.

The while loop of lines 7–13 repeatedly links the root $x$ of the tree containing node $w$ to another tree whose root has the same degree as $x$, until no other root has the same degree. This while loop maintains the following invariant:

At the start of each iteration of the while loop, $d = x.\text{degree}$.

We use this loop invariant as follows:

Initialization: Line 6 ensures that the loop invariant holds the first time we enter the loop.

Maintenance: In each iteration of the while loop, $A[d]$ points to some root $y$. Because $d = x.\text{degree} = y.\text{degree}$, we want to link $x$ and $y$. Whichever of $x$ and $y$ has the smaller key becomes the parent of the other as a result of the link operation, and so lines 9–10 exchange the pointers to $x$ and $y$ if necessary. Next, we link $y$ to $x$ by the call FIB-HEAP-LINK($H, y, x$) in line 11. This call increments $x.\text{degree}$ but leaves $y.\text{degree}$ as $d$. Node $y$ is no longer a root, and so line 12 removes the pointer to it in array $A$. Because the call of FIB-HEAP-LINK increments the value of $x.\text{degree}$, line 13 restores the invariant.

Termination: We repeat the while loop until $A[d] = \text{NIL}$, in which case there is no other root with the same degree as $x$.

After the while loop terminates, we set $A[d]$ to $x$ in line 14 and perform the next iteration of the for loop.

All that remains is to clean up. Once the for loop of lines 4–14 completes, line 15 empties the root list, and lines 16–23 reconstruct it from the array $A$. After consolidating the root list, FIB-HEAP-EXTRACT-MIN finishes up by decrementing $H.n$ in line 11 and returning a pointer to the deleted node $z$ in line 12.
CONSOLIDATE($H$)

1 let $A[0..D(H.n)]$ be a new array
2 for $i = 0$ to $D(H.n)$
3     $A[i] = \text{NIL}$
4 for each node $w$ in the root list of $H$
5     $x = w$
6     $d = x.\text{degree}$
7     while $A[d] \neq \text{NIL}$
8         $y = A[d]$  // other node of degree $x.\text{degree}$
9         if $x.\text{key} > y.\text{key}$
10            exchange $x$ with $y$
11            FIB-HEAP-LINK($H$, $y$, $x$)
12     $A[d] = \text{NIL}$
13     $d = d + 1$
14     $A[d] = x$
15 $H.\text{min} = \text{NIL}$
16 for $i = 0$ to $D(H.n)$
17     if $A[i] \neq \text{NIL}$
18         if $H.\text{min} == \text{NIL}$
19             create a root list for $H$ containing just $A[i]$
20             $H.\text{min} = A[i]$
21         else insert $A[i]$ into $H$'s root list
22         if $A[i].\text{key} < H.\text{min}.\text{key}$
23             $H.\text{min} = A[i]$
Fibonacci heaps — Consolidate

**CONSOLIDATE**(\(H\))

1. let \(A[0 \ldots D(H.n)]\) be a new array
2. for \(i = 0\) to \(D(H.n)\)
3. \(A[i] = \text{NIL}\)
4. for each node \(w\) in the root list of \(H\)
5. \(x = w\)
6. \(d = x.\text{degree}\)
7. while \(A[d] \neq \text{NIL}\)
8. \(y = A[d]\)  // other node of degree \(x.\text{degree}\)
9. if \(x.\text{key} > y.\text{key}\)
10. exchange \(x\) with \(y\)
11. \(\text{FIB-HEAP-LINK}(H, y, x)\)
12. \(A[d] = \text{NIL}\)
13. \(d = d + 1\)
14. \(A[d] = x\)
15. \(H.\text{min} = \text{NIL}\)
16. for \(i = 0\) to \(D(H.n)\)
17. if \(A[i] \neq \text{NIL}\)
18. if \(H.\text{min} == \text{NIL}\)
19. create a root list for \(H\) containing just \(A[i]\)
20. \(H.\text{min} = A[i]\)
21. else insert \(A[i]\) into \(H\)'s root list
22. if \(A[i].\text{key} < H.\text{min}.\text{key}\)
23. \(H.\text{min} = A[i]\)
Figure 19.4: The action of FIB-HEAP-EXTRACT-MIN.

(c)–(e) The array $A$ and the trees after each of the first three iterations of the for loop of lines 4–14 of the procedure CONSOLIDATE. The procedure processes the root list by starting at the node pointed to by $H.min$ and following right pointers. Each part shows the values of $w$ and $x$ at the end of an iteration.
Fibonacci heaps — Consolidate

CONSOFlDATE(H)

1 let A[0 .. D(H.n)] be a new array
2 for i = 0 to D(H.n)
3 A[i] = NIL

4 for each node w in the root list of H
5 x = w
6 d = x.degree
7 while A[d] ≠ NIL
8 y = A[d] // other node of degree x.degree
9 if x.key > y.key
10 exchange x with y
11 FIB-HEAP-LINK(H, y, x)
12 A[d] = NIL
13 d = d + 1
14 A[d] = x

H.min = NIL
16 for i = 0 to D(H.n)
17 if A[i] ≠ NIL
18 if H.min == NIL
19 create a root list for H containing just A[i]
20 H.min = A[i]
21 else insert A[i] into H's root list
22 if A[i].key < H.min.key
23 H.min = A[i]

FIB-HEAP-LINK(H, y, x)

1 remove y from the root list of H
2 make y a child of x, incrementing x.degree
3 y.mark = FALSE
Figure 19.4: The action of FIB-HEAP-EXTRACT-MIN.

(f)–(h) The next iteration of the for loop, with the values of $w$ and $x$ shown at the end of each iteration of the while loop of lines 7–13. Part (f) shows the situation after the first time through the while loop. Node 23 has been linked to node 7, which $x$ now points to. In part (g), node 17 has been linked to node 7, which $x$ still points to. In part (h), node 24 has been linked to node 7. Since no node was previously pointed to by $A[3]$, at the end of the for loop iteration, $A[3]$ is set to point to the root of the resulting tree.
Figure 19.4: The action of FIB-HEAP-EXTRACT-MIN.

(i)–(l) The situation after each of the next four iterations of the for loop.
Fibonacci heaps — Consolidate

CONSOLIDATE($H$)

1. let $A[0 \ldots D(H.n)]$ be a new array
2. for $i = 0$ to $D(H.n)$
3. \hspace{1em} $A[i] = \text{NIL}$
4. for each node $w$ in the root list of $H$
5. \hspace{1em} $x = w$
6. \hspace{2em} $d = x.\text{degree}$
7. \hspace{2em} while $A[d] \neq \text{NIL}$
8. \hspace{3em} $y = A[d]$ \hspace{1em} // other node of degree $x.\text{degree}$
9. \hspace{3em} if $x.\text{key} > y.\text{key}$
10. \hspace{4em} exchange $x$ with $y$
11. \hspace{4em} FIB-HEAP-LINK($H$, $y$, $x$)
12. \hspace{2em} $A[d] = \text{NIL}$
13. \hspace{2em} $d = d + 1$
14. \hspace{2em} $A[d] = x$
15. $H.\text{min} = \text{NIL}$
16. for $i = 0$ to $D(H.n)$
17. \hspace{1em} if $A[i] \neq \text{NIL}$
18. \hspace{2em} if $H.\text{min} == \text{NIL}$
19. \hspace{3em} create a root list for $H$ containing just $A[i]$
20. \hspace{3em} $H.\text{min} = A[i]$
21. \hspace{2em} else insert $A[i]$ into $H$'s root list
22. \hspace{3em} if $A[i].\text{key} < H.\text{min}.\text{key}$
23. \hspace{4em} $H.\text{min} = A[i]$
Figure 19.4: The action of FIB-HEAP-EXTRACT-MIN.

(m) Fibonacci heap $H$ after reconstructing the root list from the array $A$ and determining the new $H.min$ pointer.
For a given Fibonacci heap $H$, we indicate by $t(H)$ the number of trees in the root list of $H$ and by $m(H)$ the number of marked nodes in $H$.

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

A $O(D(n))$ contribution comes from FIB-HEAP-EXTRACT-MIN processing at most $D(n)$ children of the minimum node and from the work in lines 2–3 and 16–23 of CONSOLIDATE.

The size of the root list upon calling CONSOLIDATE is at most

$$D(n) + t(H) - 1$$

since it consists of the original $t(H)$ root-list nodes, plus the children of the extracted node, which number is at most $D(n)$, minus the extracted node.

Within a given iteration of the for loop of lines 4–14, the number of iterations of the while loop of lines 7–13 depends on the root list. But we know that every time through the while loop, one of the roots is linked to another, and thus the total number of iterations of the while loop over all iterations of the for loop is at most the number of roots in the root list. Hence, the total amount of work performed in the for loop is at most proportional to $D(n) + t(H)$. Thus, the total actual work in extracting the minimum node is $O(D(n) + t(H))$. 
The size of the root list upon calling CONSOLIDATE is at most

\[ D(n) + t(H) - 1 \]

since it consists of the original \( t(H) \) root-list nodes, plus the children of the extracted node, which number is at most \( D(n) \), minus the extracted node.

Within a given iteration of the for loop of lines 4–14, the number of iterations of the while loop of lines 7–13 depends on the root list. But we know that every time through the while loop, one of the roots is linked to another, and thus the total number of iterations of the while loop over all iterations of the for loop is at most the number of roots in the root list. Hence, the total amount of work performed in the for loop is at most proportional to \( D(n) + t(H) \). Thus, the total actual work in extracting the minimum node is \( O(D(n) + t(H)) \).

The potential before extracting the minimum node 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 amortized cost is thus at most

\[
O(D(n) + t(H)) + ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H))
= O(D(n)) + O(t(H)) - t(H)
= O(D(n))
\]
FIB-HEAP-DECREASE-KEY($H$, $x$, $k$)

1 if $k > x.key$
2   error "new key > current key"
3 $x.key = k$
4 $y = x.p$
5 if $y \neq \text{NIL}$ and $x.key < y.key$
6   CUT($H$, $x$, $y$)
7   CASCADING-CUT($H$, $y$)
8 if $x.key < H\.min\.key$
9   $H\.min = x$

CUT($H$, $x$, $y$)

1 remove $x$ from the child list of $y$, decrementing $y\.degree$
2 add $x$ to the root list of $H$
3 $x.p = \text{NIL}$
4 $x\.mark = \text{FALSE}$

CASCADING-CUT($H$, $z$)

1 $z = y.p$
2 if $z \neq \text{NIL}$
3   if $\neg y\.mark$
4     $y\.mark = \text{TRUE}$
5 else CUT($H$, $y$, $z$)
6   CASCADING-CUT($H$, $z$)
Figure 19.5: Two calls of FIB-HEAP-DECREASE-KEY.

(a) The initial Fibonacci heap.

(b) The node 46 has its key decreased to 15. The node becomes a root, and its parent 24, which had previously been unmarked, becomes marked.
Figure 19.5: Two calls of FIB-HEAP-DECREASE-KEY.

(c)–(e) The node 35 has its key decreased to 5. In part (c), the node 5, becomes a root. Its parent, 26, is marked, so a cascading cut occurs. The node 26 is cut from its parent and made an unmarked root in (d). Another cascading cut occurs, since the node 24 is marked as well. This node is cut from its parent and made an unmarked root in part (e). The cascading cuts stop at this point, since the node 7 is a root. (Even if this node were not a root, the cascading cuts would stop, since it is unmarked.) Part (e) shows the result of the FIB-HEAP-DECREASE-KEY operation, with $H.min$ pointing to the new minimum node.
Fibonacci heaps — Delete

FIB-HEAP-DELETE($H$, $x$)

1  FIB-HEAP-DECREASE-KEY($H$, $x$, $H.min.key - 1$)
2  FIB-HEAP-EXTRACT-MIN($H$)
Lemma 19.1

Let $x$ be any node in a Fibonacci heap, with $x\text{.degree} = k$.

Let $y_1, y_2, \ldots, y_k$ denote the children of $x$ in the order in which they were linked to $x$, from the earliest to the latest. Then, $y_1\text{.degree} \geq 0$ and $y_i\text{.degree} \geq i - 2$ for $i = 2, 3, \ldots, k$.

Proof Obviously, $y_1\text{.degree} \geq 0$.

For $i \geq 2$, we note that when $y_i$ was linked to $x$, all of $y_1, y_2, \ldots, y_{i-1}$ were children of $x$, and so we must have had $x\text{.degree} \geq i - 1$. Because node $y_i$ is linked to $x$ only if $x\text{.degree} = y_i\text{.degree}$ (by CONSOLIDATE), we must have also had $y_i\text{.degree} \geq i - 1$ at that time. Since then, node $y_i$ has lost at most one child, since it would have been cut from $x$ (by CASCADING-CUT) if it had lost two children. We conclude that $y_i\text{.degree} \geq i - 2$. 
Fibonacci heaps — Time

Lemma 19.2

\[ F_{k+2} = 1 + \sum_{i=0}^{k} F_i \]

Lemma 19.3

\[ \geq \Phi^k \]
Lemma 19.4

Let \( x \) be any node in a Fibonacci heap, and let \( k = \text{degree}(x) \).
Then \( \text{size}(x) \geq F_{k+2} \geq \Phi^k \), where \( \Phi = \frac{1 + \sqrt{5}}{2} \approx 1.618034 \), a solution of \( \Phi^2 = \Phi + 1 \).

**Proof** Let \( s_k \) denote the minimum possible size of any node of degree \( k \) in any Fibonacci heap.
Trivially, \( s_0 = 1 \) and \( s_1 = 2 \). The number \( s_k \) is at most \( \text{size}(x) \) and, because adding children to a node cannot decrease the node's size, the value of \( s_k \) increases monotonically with \( k \). Consider some node \( z \), in any Fibonacci heap, such that \( \text{z.degree} = k \) and \( \text{size}(z) = s_k \). Because \( s_k \) is a lower bound on \( \text{size}(x) \), we compute a lower bound on \( \text{size}(x) \) by computing a lower bound on \( s_k \).

...
Lemma 19.4

Let $x$ be any node in a Fibonacci heap, and let $k = x.\text{degree}$. Then $\text{size}(x) \geq F_{k+2} \geq \Phi^k$, where $\Phi = (1 + \sqrt{5})/2 \approx 1.618034$, a solution of $\Phi^2 = \Phi + 1$.

Proof Let $s_k$ denote the minimum possible size of any node of degree $k$ in any Fibonacci heap.

As in Lemma 19.1, let $y_1, y_2, \ldots, y_k$ denote the children of $z$ in the order in which they were linked to $z$. To bound $s_k$, we count one for $z$ itself and one for the first child $y_1$ (for which $\text{size}(y_1) \geq 1$), giving

$$
\text{size}(x) \geq s_k \\
\geq 2 + \sum_{i=2}^{k} s_{y_i.\text{degree}} \\
\geq 2 + \sum_{i=2}^{k} s_{i-2},
$$

where the last line follows from Lemma 19.1 (so that $y_i.\text{degree} \geq i - 2$) and the monotonicity of $s_k$ (so that $s_{y_i.\text{degree}} \geq s_{i-2}$).
Lemma 19.4

Let $x$ be any node in a Fibonacci heap, and let $k = x.\text{degree}$.
Then $\text{size}(x) \geq F_{k+2} \geq \Phi^k$, where $\Phi = (1+\sqrt{5})/2 \approx 1.618034$, a solution of $\Phi^2 = \Phi + 1$.

Proof Let $s_k$ denote the minimum possible size of any node of degree $k$ in any Fibonacci heap.

We now show by induction on $k$ that $s_k \geq F_k + 2$ for all nonnegative integers $k$. The bases, for $k \leq 1$, are trivial. For the inductive step, we assume that $k \geq 2$ and that $s_i \geq F_{i+2}$ for $i = 0, 1, \ldots, k - 1$. We have

\[
s_k \geq 2 + \sum_{i=2}^{k} s_{i-2} \geq 2 + \sum_{i=2}^{k} F_i = 1 + \sum_{i=0}^{k} F_i = F_{k+2} \geq \Phi^k.
\]

Thus, we have shown that $\text{size}(x) \geq s_k \geq F_k + 2 \geq \Phi^k$. 

---

**Fibonacci heaps — Time**
CLRS 19 Fibonacci Heaps