Fibonacci Heaps

Ugur Dogrusoz
Computer Eng Dept, Bilkent Univ
Amortized analysis

- **Average cost of an operation is small** when averaged over a sequence of operations even though **a single operation might be expensive**

- **Methods**
  - Aggregate
  - Accounting (associated with each object)
  - Potential (associated with whole data structure)

- **Example**: ArrayList in Java
Potential method

Represents prepaid work as potential energy or just potential that can be released to pay for the future operations

\[ C_i : \text{actual cost of } i^{\text{th}} \text{ operation} \]
\[ D_i : \text{data structure after } i^{\text{th}} \text{ operation} \]
\[ \phi(D_i) : \text{potential associated with } D_i \]
\[ \hat{C}_i : \text{amortized cost of } i^{\text{th}} \text{ operation w.r.t. } \phi \]
Potential method

\[ C_i = C_i + \phi(D_i) - \phi(D_{i-1}) \]

\[ \sum_{i=1}^{n} C_i = \sum_{i=1}^{n} [C_i + \phi(D_i) - \phi(D_{i-1})] = \]

\[ \sum_{i=1}^{n} C_i + \phi(D_n) - \phi(D_0) \]

- If we ensure that \( \phi(D_i) \geq \phi(D_0), 0 \leq i \leq n \)
  then total amortized cost is an upper bound on actual cost
Fibonacci heaps

Mergeable heaps support:

- **MAKE-HEAP()**: create and return a new empty heap
- **INSERT(H,x)**: insert element x into heap H
- **MINIMUM (H)**: return a pointer to element with minimum key in H
- **EXTRACT-MIN(H)**: delete and return a pointer to element with minimum key in H
- **UNION(H₁,H₂)**: create and return a new heap containing all elements of H₁ and H₂
Fibonacci heaps

- Additionally Fibonacci heaps support:
  - Decrease-Key(H, x, k): assign key k (no greater than current key value) to element x in H
  - Delete(H, x): deletes element x from heap H
# Fibonacci heaps

<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>
Each node $x$ of Fibonacci heap $H$ contains

- $x.key$: its key
- $x.p$: its parent
- $x.child$: any one of its children (forms a circular list)
- $x.degree$: number of children
- $x.mark$: whether $x$ has lost a child since the last time $x$ was made the child of another node
- $y.left/right$: each child maintains left and right siblings (forms a circular list)
Fibonacci heaps

For Fibonacci heap $H$

- $H\.min$: the root of a tree containing the minimum key
- $H\.n$: number of nodes in $H$
Fibonacci heaps

- a collection of rooted trees that are min-heap ordered
Potential function

For Fibonacci heap $H$

- $t(H)$: number of trees in the root list of $H$
- $m(H)$: number of marked nodes in $H$
- $\Phi(H) = t(H) + 2m(H)$: potential of $H$
- $\Phi(H_0) \leq \Phi(H_i)$ holds
Amortized analysis assumes we know

- an upper bound on the maximum degree \( D(n) \) of any node in a Fibonacci heap with \( n \) nodes
- When only mergeable heap operations are supported \( D(n) \leq \lg n \)
- Need to show \( D(n) = \Theta(\lg n) \) with \texttt{Decrease-Key} and \texttt{Delete} as well
Mergeable heap operations

- Delay work as long as possible
  - Do not consolidate trees in root list on operations like \textsc{Insert} or \textsc{Union}
  - Delay until the next \textsc{Extract-Min} when we really need to find a new minimum
  - After consolidation, each node in the root list has a degree that is unique, ensuring a root list of size at most \( D(n) + 1 = O(\lg n) \)

- Creating a new empty Fibonacci heap is straightforward in \( O(1) \) time, returning \( H \) with
  - \( H.n = 0 \) and \( H.min = \text{NIL} \)
Inserting a node

- The node with new key becomes its own min-heap-ordered tree, and added to root list

```
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
```
Inserting a node

- The node with new key becomes its own min-heap-ordered tree, and added to root list
Inserting a node

For a heap \( H \) before insertion and heap \( H' \) after insertion, we have

- \( t(H') = t(H) + 1 \)
- \( m(H') = m(H) \)

Increase in potential

- \([t(H) + 1 + 2m(H)] - [t(H) + 2m(H)] = 1\)

Since actual cost is \( O(1) \), the amortized cost is

- \( O(1) + 1 = O(1) \)
Finding minimum node

- Trivial by following $H\cdot \text{min}$ in $O(1)$ time
- Potential of $H$ does not change, thus amortized cost is equal to its $O(1)$ actual cost
Uniting two Fibonacci heaps

- Simply concatenates root lists of given heaps and determines the new minimum in $O(1)$ time.

\[
\text{FIB-HEAP-UNION}(H_1, H_2)
\]

1. $H = \text{MAKE-FIB-HEAP}()$
2. $H.\text{min} = H_1.\text{min}$
3. Concatenate the root list of $H_2$ with the root list of $H$
4. \textbf{if} ($H_1.\text{min} = \text{NIL}$) or ($H_2.\text{min} \neq \text{NIL}$ and $H_2.\text{min}$.key $< H_1.\text{min}$.key)
5. \hspace{.5cm} $H.\text{min} = H_2.\text{min}$
6. $H.n = H_1.n + H_2.n$
7. \textbf{return} $H$

Ugur Dogrusoz
CS 502, Algorithms II, Bilkent Univ
Uniting two Fibonacci heaps

- No change in potential since

\[
\phi(H) - [\phi(H_1) + \phi(H_2)] \\
= (t(H) + 2m(H)) - [(t(H_1) + 2m(H_1)) + (t(H_2) + 2m(H_2))] \\
= 0
\]

since \(t(H) = t(H_1) + t(H_2)\) and \(m(H) = m(H_1) + m(H_2)\)

- Amortized cost then is equal to its actual \(O(1)\) cost
Extracting the minimum

Most complicated operation so far with delayed consolidation of trees in the root taking place

Works as follows:

- Make a root out of each of the minimum node’s children
- Remove the minimum node from root list
- Consolidate the root list by linking roots of equal degree until at most one root remains of each degree
Extracting the minimum

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

1. $z = H\.min$
2. if $z \neq \text{NIL}$
3. for each child $x$ of $z$
4. add $x$ to the root list of $H$
5. $x\.p = \text{NIL}$
6. remove $z$ from the root list of $H$
7. if $z == z\.right$
8. $H\.min = \text{NIL}$
9. else $H\.min = z\.right$
10. CONSOLIDATE($H$)
11. $H\.n = H\.n - 1$
12. return $z$
Consolidation during extracting

Repeatedly execute until every root has a distinct degree:

- Find two roots \(x\) and \(y\) with same degree \(x.key \leq y.key\)
- Link \(y\) to \(x\) by removing \(y\) from root list and making \(y\) a child of \(x\) with \textsc{Fib-Heap-Link}
  - Increments \(x\).degree and clears \(y\).mark

\[
\textsc{Fib-Heap-Link}(H, y, x) \\
1 \quad \text{remove } y \text{ from the root list of } H \\
2 \quad \text{make } y \text{ a child of } x, \text{ incrementing } x\text{.degree} \\
3 \quad y\text{.mark} = \text{FALSE}
\]
Consolidation during extracting

(a) 23 7 21
     \    /
    3 - 17
     /    /
 18 - 52 - 38
    \    /
     39 - 41

(b) 23 7 21
     \    /
    3 - 17
     /    /
 18 - 52 - 38
    \    /
     39 - 41

(c) 23 7 21
     \    /
    3 - 17
     /    /
 18 - 52 - 38
    \    /
     39 - 41

(d) 23 7 21
     \    /
    3 - 17
     /    /
 18 - 52 - 38
    \    /
     39 - 41
Consolidation during extracting
Consolidation during extracting

Ugur Dogrusoz

CS 502, Algorithms II, Bilkent Univ
Consolidation during extracting

```plaintext
CONSOLIDATE(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]  // another node with the same degree as x
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
15  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]
```
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. $y = A[d] \quad \text{// another node with the same degree as } x$
         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. $d = d + 1$

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. $H.\text{min} = A[i]$
Consolidation during extracting

- Amortized cost is at most

\[ \hat{C}_i = C_i + \phi(D_i) - \phi(D_{i-1}) \]
\[ = O(D(n) + t(H)) + ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H)) \]
\[ = O(D(n)) + O(t(H)) - t(H) \]
\[ = O(D(n)) \]

- since at most \( D(n) + 1 \) roots remain and no nodes become marked during the operation

- Intuitively cost of performing each link is paid for by the reduction in potential due to the link’s reducing the number of roots by one

- Since \( D(n) = O(\lg n) \), amortized cost is \( O(\lg n) \).
Decreasing a key

(a)

(b)

(c)

(d)

(e)

H.\text{min}
Decreasing a key

**FIB-HEAP-DECREASE-KEY** \((H, x, k)\)

1. if \(k > x.key\)
2. error “new key is greater than current key”
3. \(x.key = k\)
4. \(y = x.p\)
5. if \(y \neq \text{NIL}\) and \(x.key < y.key\)
6. \(\text{CUT}(H, x, y)\)
7. \(\text{CASCADING-CUT}(H, y)\)
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** \((H, y)\)

1. \(z = y.p\)
2. if \(z \neq \text{NIL}\)
3. if \(y.mark = \text{FALSE}\)
4. \(y.mark = \text{TRUE}\)
5. else \(\text{CUT}(H, y, z)\)
6. \(\text{CASCADING-CUT}(H, z)\)
Decreasing a key

- Amortized cost is

\[ \hat{C}_i = C_i + \phi(D_i) - \phi(D_{i-1}) \]
\[ = O(c) + (t(H) + c) + 2(m(H) - c + 2) - (t(H) + 2m(H)) \]
\[ = O(c) + 4 - c = O(1) \]

- where each \texttt{FIB-HEAP-DECREASE-KEY} results in \(c\) calls to \texttt{CASCADING-CUT}
Deleting a node

- First make $x$ the minimum node by decreasing its key to $-\infty$, and then extract it

\[
\text{Fib-Heap-Delete}(H, x)
\]

1. Fib-Heap-Decrease-Key\((H, x, -\infty)\)
2. Fib-Heap-Extract-Min\((H)\)

- Same amortized cost $O(D(n)) = O(\log n)$ as extraction
Lemma Let $x$ be any node in a Fibonacci heap, and suppose that $x.\text{degree}=k$. 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 $i=2, 3, \ldots, k$. 
Lemma For all integers $k \geq 0$,

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

Proof Use induction on $k$
Bounding maximum degree

**Lemma** For all integers $k \geq 0$, 
\[ F_{k+2} \geq \phi^k \]

**Proof** Use induction on $k$

\[
F_{k+2} = F_{k+1} + F_k \geq \phi^{k-1} + \phi^{k-2}
\]
\[
= \phi^{k-2} (\phi + 1) = \phi^{k-2} \phi^2 = \phi^k
\]
Bounding maximum degree

**Lemma** Let $x$ be any node in a Fibonacci heap, and let $k = x$.degree. Then,

$$\text{size}(x) \geq F_{k+2} \geq \phi^k,$$

where $\phi = (1 + \sqrt{5}) / 2$.

**Proof**

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

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2} \geq 2 + \sum_{i=2}^{k} F_i = 1 + \sum_{i=0}^{k} F_i = F_{k+2} \geq \phi_k$$
Corollary The maximum degree $D(n)$ of any node in an $n$-node Fibonacci heap is $O(lg \ n)$.

Proof

\[ n \geq size(x) \geq \phi^k \implies \log_\phi n \geq k \]