### In lieu of recitations – 320 Office Hours

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thurs</th>
<th>Fri</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Dr. Georg</td>
<td>Dr. Georg</td>
<td>Cole</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Dr. Georg/ Jim</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Ali</td>
<td>Jim</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Cole</td>
<td>Ali</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Cole/ Shannon</td>
<td></td>
<td>Ali</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Shannon</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Dr. Georg</td>
<td>Dr. Georg</td>
<td>Jim</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Upcoming -- Check Progress page, Piazza postings for updates**
- Topological sort program due Oct 28
- MST Canvas Quiz due 11:59pm Oct 29
Prim(G,r)
    foreach (u ∈ V)
        a[u] ← ∞; u.π = ∅
    a[r]= 0
    min priority queue Q = {}
    foreach (u ∈ V) insert u onto Q (key: ‘a’ value)
    set S ← {}
    while (Q is not empty) {
        u ← extract min element from Q
        S ← S ∪ { u }
        foreach (edge e = (u, v) incident to u)
            if ((v ∉ S) and (c_e < a[v]))
                decrease priority a[v] to c_e
            v.π = u
Complexity and Data Structures

Prim’s: $|V| = n$ and $|E| = m$
- $m$ reduce-key operations
- $n$ extract-min operations

With a binary heap:
- $O(m \log n + n \log n) = O(m \log n)$

As the graph gets dense, $m \to n^2$:
- $O(n^2 \log n)$

Flat array:
- $O(1)$ for reduce-key, $O(n)$ for extract min
- $O(m + n^2) = O(n^2)$

Trade-off point? $m \log n = n^2$, $m= n^2/\log n$,
- $m > n^2/\log n$ flat array is better than bin heap
Binary Heaps

A complete binary tree has the heap property:
all nodes have a lower value than the nodes in their subtrees.

We insert new nodes to maintain a complete binary tree, then swap nodes as necessary to restore the heap property.
Example
Example
Fibonacci Heap

Mergeable Heap – supports 5 operations:
• Make-Heap, Insert, Minimum, Extract-Min, Union

Fibonacci heaps also support:
• Decrease-Key, Delete

Operations run in constant *amortized* time:
• Lazy – delay work as long as possible so that time over multiple operations is minimized
# Binary, Fibonacci Heap Complexities

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Binary heap (worst-case)</th>
<th>Fibonacci heap (amortized)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAKE-HEAP</strong></td>
<td>( \Theta(1) )</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td><strong>INSERT</strong></td>
<td>( \Theta(lg\ n) )</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td><strong>MINIMUM</strong></td>
<td>( \Theta(1) )</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td><strong>EXTRACT-MIN</strong></td>
<td>( \Theta(lg\ n) )</td>
<td>( O(lg\ n) )</td>
</tr>
<tr>
<td><strong>UNION</strong></td>
<td>( \Theta(n) )</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td><strong>DECREASE-KEY</strong></td>
<td>( \Theta(lg\ n) )</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>( \Theta(lg\ n) )</td>
<td>( O(lg\ n) )</td>
</tr>
</tbody>
</table>
Trade-offs

Good when:

• number of Extract-Min and Delete small compared to other ops
• dense graphs: lots of Decrease-Key ops

Downsides:

• large constants and complexity!

Conclusions:

• binary heaps usually better
• theoretically interesting
Terminology 1

• Fibonacci heap $H$: forest of min-heap ordered trees
  • $t(H)$: number of trees in $H$
  • $m(H)$: number of marked nodes in $H$
• Minimum node $H.min$: pointer to the root of a tree with the minimum key
• Node degree (or rank): number of children; max degree $D(n)$ in $n$-node Fibonacci heap is $O(\lg n)$
• Node size: num nodes (including $x$) in subtree rooted at $x$
• Node mark: Boolean – whether node has lost a child since the last time it was made the child of another node
Side Track

The Golden Ratio

- $\varphi = (1 + \sqrt{5})/2$: positive root of $x^2 = x + 1$
- divide a line into 2 parts: longer / smaller = whole / longer
- Fibonacci numbers: ratio close to Golden ratio, gets closer with larger nums
- Golden rectangle
- Golden spiral: adjacent squares of Fibonacci dimensions
Where does the Fibonacci come from?

\( k = x.\text{degree}, \ s_k \) is minimum possible size of any node of degree \( k \)

\( s_k \geq F_{k+2} \) where \( F_{k+2} = (1 + \sum_{i=0}^{k} F_i) = \phi^k \)

And \( F_i = \begin{cases} 
0 & k = 0 \\
1 & k = 1 \\
F_{i-1} + F_{i-2} & k \geq 2 
\end{cases} \)
Terminology 3

Potential: method to analyze op complexity

• Potential function: $\Phi(H) = t(H) + 2m(H)$

• A unit of potential can pay for a constant amount of work, where the constant is large enough to cover the cost of any specific constant-time pieces of work we need to do
Structure

Circular, doubly linked lists for root, child lists – insert, remove, concatenate: $O(1)$. Only 1 child pointer, to any of the children. $t(H) = \text{num root list nodes}$

Maximum degree (num children) for any node in n-node heap $D(n): O(\log n)$
Where is the laziness?

1. Inserting nodes! Nodes just get inserted into the root list next to $H_{\min}$. So this list can get pretty big.

2. Decreasing a key; if the key becomes less than the parent key it gets added to the root list. If the parent isn’t in the root list, it either gets marked (because it lost a child) or if it was already marked it gets added to the root list and it’s parent is considered. Again, the root list can get pretty big.
Where is restructuring work done?

When we extract the minimum!

Nodes in the root list are consolidated so that we end up with $D(H.n) = O(lg n)$ tree roots in the root list, one for each degree up to $lg n$: 
Example

**MAKE-HEAP:**
\[ H.n = 0; \ H.\text{min} = \text{NIL}; \ t(H) = 0; \ m(H) = 0; \]
\[ \Phi(H) = 0 \text{ so amortized cost} = \text{actual cost} = O(1) \]
**INSERT(H, x₁):**

\[ x₁.\text{degree} = 0; \ x₁.p = \text{NIL}; \ x₁.\text{child} = \text{NIL}; \]
\[ x₁.\text{mark} = \text{FALSE}; \ H.n = H.n + 1; \]

create a root list with \( x₁ \) (set right and left pointers to \( x₁ \));

if \( H.\text{min} == \emptyset \) then \( H.\text{min} = x₁ \)

Creates \( H' \), so increase in \( \Phi \) is:

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

add to actual cost to get amortized cost \( O(1) + 1 = O(1) \)
**INSERT(H, x₂):**

\[ x₂.\text{degree} = 0; \ x₂.\text{p} = \text{NIL}; \ x₂.\text{child} = \text{NIL}; \]
\[ x₂.\text{mark} = \text{FALSE}; \ H.n = H.n + 1; \]
insert \( x₂ \) into root list; if \( x₂.\text{key} < H.\text{min}.\text{key} \) then \( H.\text{min} = x₂ \)
**INSERT(H, x₃):**

\[ x₃.\text{degree} = 0; \ x₃.p = \text{NIL}; \ x₃.\text{child} = \text{NIL}; \]
\[ x₃.\text{mark} = \text{FALSE}; \ H.n = H.n +1; \]

insert \( x₃ \) into root list; ; if \( x₃.\text{key} < H.\text{min.key} \) then \( H.\text{min} = x₃ \)
**INSERT**(*H*, *x₄*):

\[ x₄.\text{degree} = 0; \ x₄.p = \text{NIL}; \ x₄.\text{child} = \text{NIL}; \]
\[ x₄.\text{mark} = \text{FALSE}; \ H.n = H.n +1; \]

insert *x₄* into root list; if *x₄*.key < *H*.min.key then *H*.min = *x₄*
**Extract-Min(H):**

\[ z = H.min; \ H.min = z.right; \]

**Consolidate(H);** \[ H.n = H.n - 1; \text{ return } z \]
**EXTRACT-MIN(H):**

\[ z = H\text{.min}; \quad H\text{.min} = z\text{.right}; \]

**CONSOLIDATE(H);**

\[ H\text{.n} = H\text{.n} - 1; \quad \text{return } z \]
**CONSOLIDATE(H):**
create $A[0..1] = [\emptyset, \emptyset]$ (size is $D(H.n) = \log_\phi 3$)
Start at $H.min$ and set $A[H.min.degree]$ to point to it. Go right and either set another $A[]$ or use FIB-HEAP-LINK to join the trees (same degree). Move $H.min$ if needed.
**FIB-HEAP-LINK (H, y, x):**
Remove y from root list of H; make y a child of x; increment x.degree; y.mark = FALSE
**DECREASE-KEY (H, x, k):**
If \( k < x.\text{key} \) then set \( x.\text{key} \) to \( k \); \( y = x.\text{p} \); if \( y.\text{key} \) larger, \( \text{CUT}(H, x, y) \);
\( \text{CASCADING-CUT}(H, y) \);
fix \( H.\text{min} \) if needed
**DECREASE-KEY** *(H, x,k):*
If \( k < x\text{.key} \) then set \( x\text{.key} \) to \( k \); \( y = x.p \);
if \( y\text{.key} \) larger:

\[ \text{CUT}(H,x,y); \text{CASCADING-CUT}(H,y) \]

fix \( H\text{.min} \) if needed

**CUT:** remove \( x \) from child list of \( y \); decrement \( y\text{.degree} \); add \( x \) to root list; \( x.p = \emptyset \); \( x\text{.mark} = \text{FALSE} \)

**H.min**
**DECREASE-KEY** \((H, x, k)\):
If \(k < x\text{.key}\) then set \(x\text{.key}\) to \(k\); \(y = x\text{.p}\); if \(y\text{.key}\) larger, \(\text{CUT}(H, x, y)\); **CASCADING-CUT** \((H, y)\); fix \(H\text{.min}\) if needed

**CASCADING-CUT**: \(z = y\text{.p}\);
if \(z \neq \emptyset\):
    set mark if not set & otherwise: \(\text{CUT}(H, y, z)\);
**CASCADING-CUT** \((H, z)\)
Prim(G,r)

foreach (u ∈ V)
    a[u] ← ∞; u.π = ∅

a[r]= 0

min priority queue Q = {}

foreach (u ∈ V) insert u onto Q (key: ‘a’ value)

set S ← {}

while (Q is not empty) {
    u ← extract min element from Q
    S ← S ∪ { u }

    foreach (edge e = (u, v) incident to u)
        if ((v ∉ S) and (c_e < a[v]))
            decrease priority a[v] to c_e
            v.π = u
Muddy City 2 – Prim’s
min priority queue Q = {}

foreach (u ∈ V)
    insert u in Q set
    S ← {}

while (Q is not empty) {
    u ← Q min element(delete)
    S ← S ∪ {u}

    foreach edge e = (u, v)
        if ((v∉S) and (c_e < a[v]))
            a[v] = c_e
            v.π = u

Using Fibonacci Heaps

```
a/ 0
k/ ∞
r/ ∞k/ ∞k/ ∞k/ ∞
k/ ∞
k/ ∞
k/ ∞
k/ ∞
k/ ∞
k/ ∞
```
Extract Min/Consolidate - 1
Extract Min/Consolidate - 2

(i)  
```
    7
   /\    \n  24 17  23 18
   \  /  \    
     w x 52 38
```

(j)  
```
    7
   /\    \n  24 17  23 18
   \  /  \    
     w x 52 38
```

(k)  
```
    7
   /\    \n  24 17  23 18
   \  /  \    
     w x 52 38
```

(l)  
```
    7
   /\    \n  24 17  23 18
   \  /  \    
     w x 52 38
```

(m)  
```
    7
   /\    \n  24 17  23 18
   \  /  \    
     w x 52 38
```

H. min
Decrease Key

(a)  

(b)  

(c)  

(d)  

(e)  

H.min
Image Credits

bean: http://clipart-library.com/bean-people.html

goldenRatio: https://www.livescience.com/37704-phi-golden-ratio.html

Fibonacci Heap diagrams from Cormen, et. al., Introduction to Algorithms, 3rd Edition, 2009, MIT