Fibonacci Heaps

1. Definition:

- A forest of heap-ordered trees. The parent's priority is always less than or equal to its children's priority.

- The roots are stored in a circular doubly-linked list. Furthermore, the circular doubly-linked list is called The Root List.

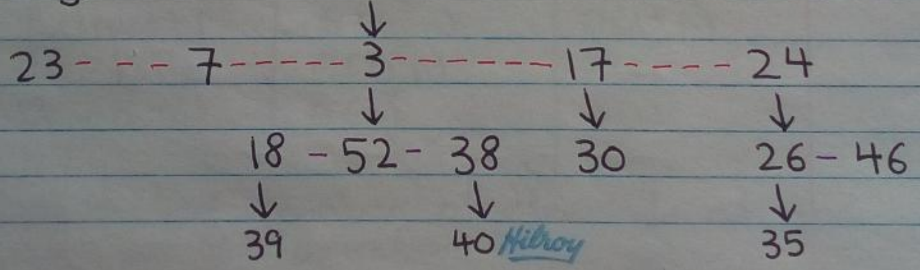
- There is a pointer to the min root.

- The siblings are also stored in a circular doubly-linked list. However, the parent only knows one arbitrary child.

- We define H.n to be the number of nodes in H.

- We define deg(x) to be the number of children in x's child list.
  E.g. deg(3) = 3

- E.g.



Note: Dashed lines represent circular doubly-linked lists and red represents the root list.

- Each node has the following:
  - key: The node's priority.
  - Left, Right: Pointer to the left and right sibling.
  - Parent: Pointer to the parent.
  - Child: Pointer to one child.
  - degree: The number of children.
  - mark: A boolean. Set to false, but if a node loses a child, it is set to true. This is important during decrease-priority.
- The whole heap has:
  min: A pointer to the min root.

2. Operations:

- Make_Heap(): Creates a new empty heap.
- Insert (H, x): Insert x to H.
- Min(H): Return a pointer to the min key in H.
- Extract_Min(H): Deletes the element from H whose key is min and returns a pointer to the key.
- Union(H_1, H_2): Creates and returns a new heap that contains the elements from H_1 and H_2.
- Decrease_key(H, x, key): Assigns to element x in heap H the new key value key.
- Delete(H, x): Delete key x from H.

3. Binary Heaps Vs Fibonacci Heaps:

-

	Binary Heap Worst Case	Fib Heap Amortized
Insert	$\Theta(\lg n)$	$\Theta(1)$
Extract-Min	$\Theta(\lg n)$	$O(\lg n)$
Dec-Pri	$\Theta(\lg n)$	$\Theta(1)$
Union	$\Theta(n)$	$\Theta(1)$

n is the number of elements at the time of the operation.

- If Prim's alg used Fib heap:
  - $|V|$ extract-mins: $O(|V|\lg|V|)$ time
  - Up to $|E|$ dec-pris: $O(|E|)$ time
  - Total: $O(|V|\lg|V| + |E|)$ time

4. Insert:

- insert(k):
  - Insert the new node in the root list.
  - key = k
  - mark = false
  - Change min if k < min.key
- Takes $\Theta(1)$ time.
- E.g.

Suppose we have this Fib heap and want to insert 21.

```
                    ↓
17 — 15 — 20 — 3          }
↓                   ↓           } Og Heap
30                 19 — 18   }

17 — 15 — 20 — 21 — 3     }
↓                         ↓      } New
30                       19 — 18 } Heap
```

5. Union(H_1, H_2):

- Concatenate the 2 root lists.
- H_1.min and H_2.min compete the new min.
- Takes $\Theta(1)$ time.
- E.g. Consider the 2 fib heaps.

```
             ↓                  ↓
23 - 24 - 17            7 - 3 - 21
↓                           ↓
30                          18 -   33
```

If we union them, we get

```
                      ↓
23 - 24 - 17 - 7 - 3 - 21
↓                  ↓
30                 18 - 33
```

6. Extract-Min:

- The min key is already pointed to by H. min.
- We remove the min root and promote its children to the root list.
- Now, H.min may point to any node on the root list. (May not be the correct node.)

- We consolidate the fib heap.
  I.e. We want to end up with a root list with nodes of unique degrees.

Pseudo-Code:

- Repeat until all nodes in the root list have unique degrees.
  - Walk through the root list.
  - Remember the degree of each node passed. We can do this by using an array A of pointers.
  - A[1] has a pointer to the root node with degree 1, A[2] has a pointer to the root node with degree 2, etc.
  - If we find a node x with the same degree as an already seen node y pointed to by A, we do remove-max(x, y) and make one the child of the other.
- Find the new min root and set H.min to it.

- E.g. Consider the fib heap below.

```
            ↓
8 — 9 — 1 — 3
|       |    |
12      10  5-4
```

Here are the steps if we do extract_min on the fib heap.

1. Remove 1 and promote 10 to the root list.

```
8 — 9 — 10 — 3
|            |
12          5-4
```

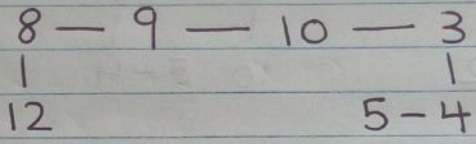
2. Go through the root list and remember the degree of each node. We will start at 8.

```
8 — 9 — 10 — 3
|            |
12          5-4
```

A:

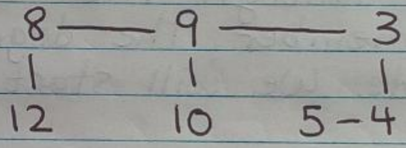
0	1	2	...
	8		

3. Go to 9 and do the same thing.

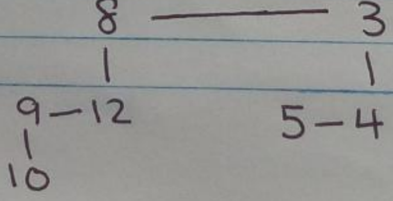


$A: \left[\begin{array}{c|c|c} 0 & 1 & \cdots \\ 9 & 8 & \end{array}\right]$

4. Go to 10 and do the same thing. However, we notice that 9 is already in the spot pointed to by A[0]. We remove the bigger node and attach it to the smaller node as its child. Therefore, 10 becomes the child of 9.



5. Since 9 got a new child, its degree changed from 0 to 1. If we update 9's place in the array, we see that 8 is in A[1]. Since 9 is bigger than 8, 9 becomes a child of 8.

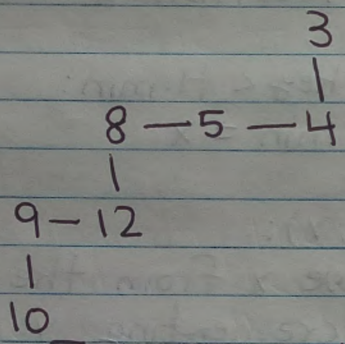


6. 8 has a degree of 2. We update 8's place in the array.

A:

0	1	2	...
		8	

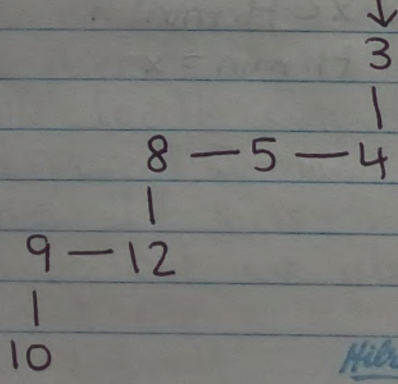
7. We go to 3. 3 has a degree of 2, and when we go to A[2], we see that 8 is already there. Since 8 is bigger than 3, 8 becomes a child of 3. Furthermore, 3 has a degree of 3, and we point A[3] to it.



A:

0	1	2	3	...
			3	

8. We get H.min to point to 3.



7. Decrease-key:

- Pseudo Code:

- Decrease_key(H, x, k):

```
if k > x.key:
    Do Nothing

x.key = k
Y = x.Parent

if Y ≠ NULL and x.key < Y.key:
    CUT(H, x, Y)
    CASCADING-CUT(H, Y)

if x.key < H.min:
    H.min = x
```

- Cut(H, x, Y):

Remove x from the child list of Y, decrementing Y.degree, and adding x to the root list of H.

```
X.parent = NULL
X.mark = False

if x.key < H.min
    H.min = x
```

- Cascading-Cut(H, y):

```
z = y.parent

if z != NULL:
    if y.mark == False:
        y.mark = True
    else:
        cut(H, y, z)
        Cascading-Cut(H, z)
```

- Explanation of Pseudo Code:

  - If k is greater than x's priority, we don't do anything cause we want to decrease x's priority.

  - Otherwise, we get x's parent. If x's parent is not NULL, I.e. If x isn't on the root list, and x's priority is less than the priority of its parent, we remove x from its parent and insert it into the root list.

  - After we cut x from its parent, we do a cascading-cut on x's parent. If x is the first child to be cut, then we set x's parent's mark value to be True. If x is the second child to be cut, we also cut x's parent. We do a cascading-cut on x's grandparent.

- Example: Consider the fib heap below.

```
↓
7
|
24 — 17 — 23
|
26 — 46
|
35
```

Note: Red means that node's mark value is True.

1. Decrease the priority of 24 to 20. Since $20 > 7$, nothing else happens.

```
↓
7
|
20 — 17 — 23
|
26 — 46
|
35
```

2. Decrease the priority of 46 to 15. Since $15 < 20$, we cut 15 from 20 and put it on the root list.

```
↓                          ↓
7                          7 ———————— 15
|                          |
20 — 17 — 23       ⇒       20 — 17 — 23
|                          |
26 — 15                    26
|                          |
35                         35
```

3. We decrease the priority of 35 to 5. Since $5 < 26$, we cut 5. However, because 26 has already lost a child, we cut 26, too. This also means that we cut 20 as it now has lost 2 children. Lastly, we update the min pointer to point to 5.

```
1. ↓
   7 ——————— 15
   |
   20 — 17 — 23
   |
   26
   |
   35

2. ↓
   7 ——————— 15
   |
   20 — 17 — 23
   |
   26
   |
   5
```

Hilroy

```
                    ↓
3. 7 ————————— 15 — 5
   |
   20 - 17 - 23
   |
   26

                ↓
4. 7 ———————— 15 — 5 — 26
   |
   20 - 17 - 23

            ↓
5. 7 ——— 15 — 5 — 26 — 20
   |
   17 - 23
```

8. Complexity:

- Let's look at the actual worst case costs.

- insert: O(1)

- extract_min:
  - Removing the min costs O(1).

  - Moving the removed node's children up to the root list takes O(d(v)) where v is the min.

  - When we consolidate, each root can be the child of another root at most once. We do at most O(trees(H)) merges.

- Finding the new min takes $O(d(n))$ where $d(n)$ is the max degree for all root nodes n.

- Total: $O(trees(H) + d(n))$

- dec-pri:

  - Update the priority of the key: $O(1)$
    Note: If the heap order is not violated, then we're done. However, if the heap order is violated, then we cut the node and insert it in the root list. This takes $O(1)$. So, in total, updating the priority of the key takes $O(1)$ time.

  - Cascading cut takes $O(\#$ of cuts$)$.

  - In total, it costs $O(\#$ cuts $+ 1)$.
    Note: $O(\#$ cuts $+1) \leq O(marks(H)+1)$

- Recall that amortized time is actual cost + change in potential.

- When we dec-pri, we mark and/or move nodes to the root list and unmark.

- When we extract-min, we turn root nodes into child nodes.

- We can think of this as each time we mark a node, it will take 2 steps to get it back into an unmarked child position.

- Leads to the potential function:
  $\phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$

- $\phi(H_0) = \text{tree}(H_0) + 2 \cdot \text{marks}(H_0)$
  $= 0$

- When an insert is performed, the potential changes by 1.

  $\Delta(\phi) = \phi(H_{i+1}) - \phi(H_i)$
  $= \text{tree}(H_{i+1}) + 2 \cdot \text{mark}(H_{i+1}) - \text{tree}(H_i) - 2 \cdot \text{mark}(H_i)$
  $= 1$

- Potential Function For Dec_Pri:

  - Suppose we make x cuts. For each cut made, we gain a root node (I.e. A tree).

  - We are done cutting when we reach an unmarked node, possibly a root.

  - x or x-1 nodes may have been marked. Furthermore, we may or may not mark the last node.

- H_{i+1} will lose at least $x-1$ marks but may gain 1.

$$\begin{aligned}\text{marks}(H_{i+1}) &\le \text{marks}(H_i) - (x-1) + 1 \\ &= \text{marks}(H_i) - x + 2\end{aligned}$$

- The amortized cost for dec-pri is $C_i + \Phi(H_{i+1}) - \Phi(H_i)$

- $$\begin{aligned}\Phi(H_{i+1}) - \Phi(H_i) &= \text{tree}(H_{i+1}) + 2\cdot\text{mark}(H_{i+1}) - (\text{tree}(H_i) + 2\cdot\text{mark}(H_i)) \\ &= \text{tree}(H_{i+1}) + 2\cdot\text{mark}(H_{i+1}) - \text{tree}(H_i) - 2\cdot\text{mark}(H_i) \\ &= \text{tree}(H_{i+1}) - \text{tree}(H_i) + 2(\text{mark}(H_{i+1}) - \text{mark}(H_i)) \\ &\le x + 2(-x+2) \\ &= 4 - x\end{aligned}$$

- $\therefore$ The amortized cost is

$$\begin{aligned}&(\#\text{cuts} + 1) + 4 - x \\ &= (x+1) + 4 - x \\ &= 5 \\ &= O(1)\end{aligned}$$

- Potential Function For Extract_Min:

  - Recall that the actual cost is $O(\text{tree}(H) + d(n))$.

- Each time we do extract_min, all of the marked children becomes unmarked.

  $\therefore \text{mark}(H_{i+1}) \leq \text{mark}(H_i)$

- After an extract_min and a consolidate, we have $d(n)+1$ roots. This is true if each root has an unique degree.

  E.g. Consider the fib heap below.

```
1 — 2 — 4
    |   |
    3   5 — 6
```

  4 has a degree of 2, but there are 3 roots.

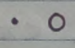
  Since there are $d(n)+1$ roots after an extract_min and a consolidate, $\text{trees}(H_{i+1}) \leq d(n)+1$.

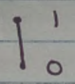
- Since $\Phi(H) = \text{tree}(H) + 2 \cdot \text{mark}(H)$,

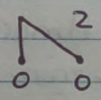
  $$\Delta(\Phi) = \text{tree}(H_{i+1}) - \text{tree}(H_i) + \underbrace{2(\text{mark}(H_{i+1}) - \text{mark}(H_i))}_{\text{This is less than } 0.}$$

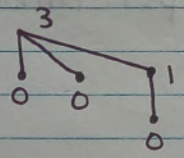
  $\therefore \Delta(\Phi) \leq d(n)+1 - \text{tree}(H_i)$ and the amortized cost is $O(d(n))$.

- We need to find a bound on $d(n)$. To do this, we need to determine the min number of nodes that is possible in a tree with a root of degree k. We will denote this number as $N(k)$.

$N(0) =$  $= 1$ node

$N(1) =$  $= 2$ nodes

$N(2) =$  $= 3$ nodes

$N(3) =$  $= 5$ nodes

$$N(k) = N(k-1) + N(k-2)$$
$$= Fib(k+2)$$

- Recall the golden ratio, ϕ, which equals to $\frac{1+\sqrt{5}}{2}$.

I.e. $\phi = \frac{1+\sqrt{5}}{2}$

$\approx 1.61803\ldots$

- For all integers $k \geq 0$, $F(k+2) \geq \phi^k$.
Since $N(k) = F(k+2)$, $N(k) \geq \phi^k$.
Let n be the number of nodes in a tree with degree k.
$n \geq N(k) \geq \phi^k$
$\log_\phi n \geq k$, where k is $d(n)$.

Therefore, extract_min has an amortized cost of $O(\log n)$.

- Summary:

  - insert: Amortized cost $O(1)$
  - extract_min: Amortized cost $O(\log n)$
  - dec_pri: Amortized cost $O(1)$