Dijkstra with Fibonacci heaps

Wednesday, 15 September 2021 2:06 PM

FIBONACCI HEAPS :

n times m times

	extract_min	decr_key	running time
array	$O(n)$	$\underline{O(1)}$	$O(n^2 + m)$
min heap	$\underline{O(\log n)}$	$O(\log n)$	$O(m \log n)$
Fibonacci heap	$O(\log n)^*$	$O(1)^*$	$O(n \log n + m)$

* amortized analysis

AMORTIZED ANALYSIS :

- start with a n-bit string consisting of all zeros
- an "operation" adds 1 to bit string
- $X(i)$ is bit string after i additions

  0000 0001 0010 0011 0100 0101 ...
- cost of an operation is # of bits flipped.
- in the worst case, cost of an operation could be n.
- want to show: starting from 0^n, the total cost for t operations is $2t$.

  i.e, the amortized cost is $O(1)$

Claim : Let $X = 0^n$. Total cost of adding t 1's to X is at most $2t$. I.e, the total number of bits flipped in the sequence

$$X = X(0) = 0^n \to 0\ldots01 \to 0\ldots010 \to \cdots \to X(t)$$

is at most $2t$.

Proof 0: In t operations, bit n flips everytime. t

bit $(n-1)$ flips everytime bit n is 1

$t/2$

bit $(n-2)$ flips everytime bits n & $n-1$ are 1

$t/4$

...

Thus total # of bit flips is $t\left(1+\frac{1}{2}+\frac{1}{4}+\cdots\right) \leq 2t$

Proof 1: $X(t-1) \quad 0011101\overline{\underset{\uparrow}{0}111} \to 0011101\overline{1000}$

Charging argument.

every operation cost of 2

every '1' in the bit string has a stored 'charge' of 1.

every operation :

- Changing '1' to '0' : uses stored charge
- changing '0' to '1' : uses 1 from it's own charge & stores 1 in the flipped bit

$$0^{0}0^{0}0^{0}0^{0} \to 0001^{1} \to 001^{1}0^{0} \to 001^{1}1^{1} \to 01^{1}0^{0}0^{0}$$

1 Re to charge

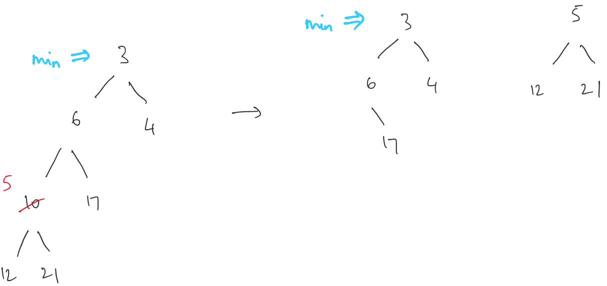
bit 4

$\downarrow$

Proof 2: Let $\phi(t)$ be # bits sent in X(t) (decimal representation of t). Then # bit flips between X(t), X(t+1) is exactly $(\phi(t) - \phi(t+1) + 2)$ (check this!)
Hence # bit flips in t operations $= \phi(0) - \phi(t) + 2t$
$\leq 2t$ ▨

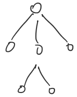
Fibonacci : $F_0 = 0$, $F_1 = 1$, $F_2 = 1$, $F_3 = 2$, ...
Numbers $F_i = F_{i-1} + F_{i-2}$

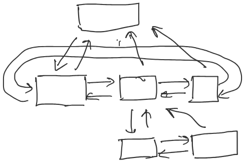
Fibonacci Heaps combine ideas from arrays & min-heaps.
We will now have an array of min-heaps.
For decr-key (v,k), if new value for v is smaller than v's parent, we will simply create a new heap w/ v as root.



- now we will have a collection of trees, not necessarily almost complete or even binary
- however each tree by itself is a min-heap
- will maintain pointer to smallest value in the Fibonacci

heap





Will use circula doubly linked lists to store trees in a Fibonacci heap.

Decr_key (will call Cascading Cut) : increases # of trees, decreases size of trees

Extract_Min (will call Consolidate): (roughly) decrease # of trees, increase size of trees.

Fibonacci heap H:

(i) Collection of min heaps, each of which is a tree

(ii) Pointer to minimum root $H.min$

(iii) For each node v in Fibonacci heap, store flag $v.lost \in \{0,1\}$ initially 0

(iv) For a node v in Fibonacci heap H:

$v.degree$: # of children

$v.size$: size of subtree rooted at v

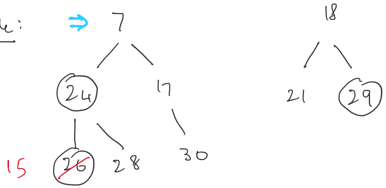
Decr_key (v,k) {

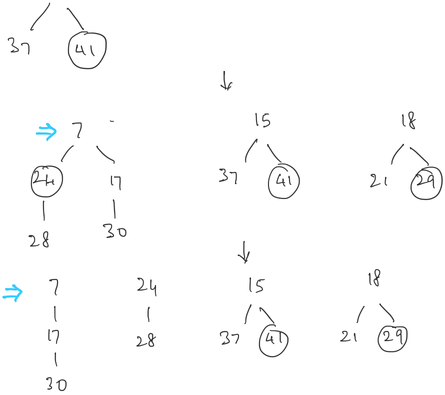
```
    v.key ← k
    if v.key ≥ v.parent.key, or if v is parent,
            do nothing
    else  p ← v.parent
          Add v to root list in H, remove v from p's children
          v.lost ← 0
          Cascading_cut (p)
}

Cascading_Cut (v) {
    if (v.lost == 1)
        p ← v.parent
        add v to root list in H, remove v from p's children
        v.lost ← 0
        Cascading_cut (p)
    else  v.lost = 1
}
```

(if v.lost is set, then v has lost one child. Hence next time Cascading_Cut (v) is called, it will be made into a separate tree)

Example:





Note: ① extract_min is not going to change v.lost for any node v

(ii) Each call to Decr_key "sets" to 1 at most one vertex's lost bit

Time taken for a single Decr_key operation:

$O(1 + \#\text{ lost bits cleared})$

Total time taken for Decr_key operations:

$O(\#\text{ of Decr\_key operations} + \text{total } \#\text{ lost bits set to } 1)$

$= O(\#\text{ of Decr\_key operations})$

& hence amortized cost of Decr_key is $O(1)$

Stopped here for Lecture 6

How many trees are we creating?

In a call to decr-key (v, k), v could become root of a tree. However subsequent nodes become roots only if their lost bits are set.

Hence, total # trees created in decr-key operations

$$\leq \# \text{ decr-key operations} + \text{total } \# \text{ lost bits set}$$

$$\leq 2 \# \text{ decr. key operations.}$$

In the extract_min operations, the following claims will hold:

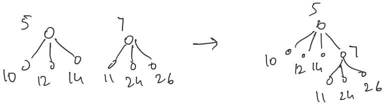
Claim 1: For any node v in F. heap w/ degree k, $v.size \geq F_{k+2} \geq \phi^k$ (where $\phi = (1+\sqrt{5})/2$).

(proof at end)

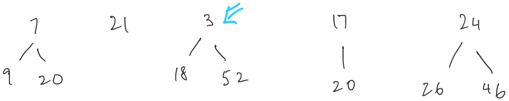
Corollary: For any node v in F. heap, $v.degree \leq \lfloor \log_\phi n \rfloor$

```
Extract_Min(H) {
    $x \leftarrow H.min.key$
    for each child $v$ of $H.min$
        Add $v$ to root list of $H$
    Remove $H.min$ from root list of $H$
    Consolidate($H$)
    return $x$
}
```

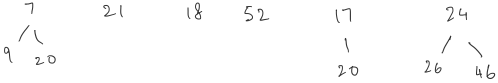
```
Consolidate (H) {
    $D \leftarrow \lfloor \log_\phi n \rfloor$
    Initialize array $A(0 \ldots D)$ with each element = NULL
    For each root $x \in H$ {
        $d \leftarrow x.degree$
        if $A(d)$ = NULL, $A(d) \leftarrow x$
        else while ($y = A(d) \neq$ NULL) {
            if $y.key < x.key$
                $x.parent \leftarrow y$, $x \leftarrow y$
            else $y.parent \leftarrow x$, $y \leftarrow x$
            $A(d) \leftarrow$ NULL, $d \leftarrow d+1$
        }
        $A(d) \leftarrow x$
    }
}
```



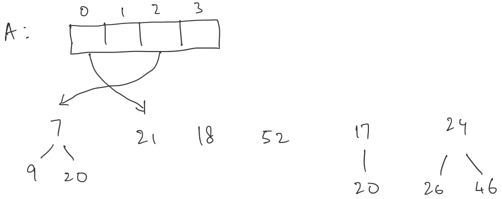
Example for Extract-Min(H):



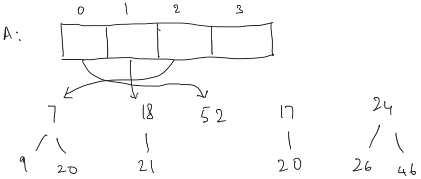
3 gets removed, 18 & 52 added to root list.



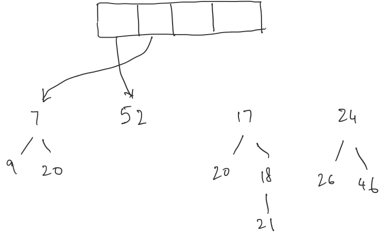
Consolidate (note that Consolidate also creates pointer to new minimum element)



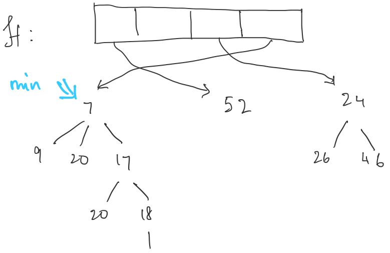
- 18 & 21 combined into a single tree:



- now 18 - 21 & 17 - 20 get combined



- trees w/ degree 2 roots get combined



Since there are at most $\log_d n$ trees, each of which is a min-heap, can find min elt. in $O(\log n)$ time

ANALYSIS OF EXTRACT_MIN

NOTE: FOR DECREASE-KEY / CASCADING-CUT

(i) makes some non-root nodes into root nodes but does not affect trees o.w.

(ii) say v is a non-root node currently. Say v had degree d the last time it was a root (or at the beginning of the algo if v was never a root). Then v has at least d-1 children

(iii) has amortized runtime $O(1)$

(iv) creates total of $O(\#\text{ decrease-key operation})$ trees

(v) called once for every edge in the graph (hence $O(m)$ times).

(vi) H.min pointer could change to the vertex passed to decrease_key (·)

FOR EXTRACT-MIN / CONSOLIDATE

(i) does not change v.lost for any vertex v

(ii) creates new trees

(iii) joins trees that have the same degree at the root

(iv) H.min pointer does change

(v) called once for every vertex (thus $O(n)$ times)

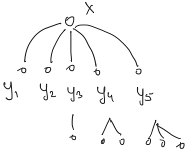
To get bound on runtime for extract_min:

Claim 1: Let x be a node in F. heap w/ degree k,

$$x.size \geq F_{k+2} \geq \phi^k$$

Corollary 1: For any vertex x, $x.degree \leq \lfloor \log_\phi n \rfloor$

Claim 2: Let x be a node in H, w/ children $y_1, y_2, \ldots, y_k$ in order in which they were linked to x. Then $y_1.degree \geq 0$ and $y_i.degree \geq i-2$ for $i = 2 \ldots k$



Proof: The only operation that adds a child to x is Consolidate. There we only link two nodes that have the same degree. When y_i became of x, x had at least $i-1$ children. But to get linked to x, y_i must have had $i-1$ children. Since then, since y_i is not a root node, it has lost at most one child. And hence $deg(y_i) \geq i-2$ ∎

To prove Claim 1: $v.degree \geq k \Rightarrow v.size \geq F_{k+2} \geq \phi^k$

verify yourself: $F_{k+2} = F_k + F_{k+1}$

$$= 1 + \sum_{i=0}^{k} F_i \geq \phi^k \text{ — } ⊛$$

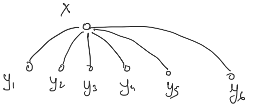
$F_0 = 0,\ F_1 = 1,\ F_2 = 1,\ F_3 = 2,\ F_4 = 3,\ F_5 = 5,\ F_6 = 8, \ldots$

Proof is by induction. Base case: v.degree ≥ 0 then

v.size $\geq 1 \geq F_2$

$= \phi^0$

Now say x.degree $\geq k$



deg $\geq$ 0 0 1 2 3 4

size $\geq$ $\underset{F_1}{\overset{\shortparallel}{F_2}}$ F_2 F_3 F_4 F_5 F_6

Then size of sub tree rooted at x

$\geq 1 + \underset{0}{\overset{\shortparallel}{F_0}} + F_1 + F_2 + F_3 + \cdots + F_k$ (By induction on k)

$= F_{k+2} \geq \phi^k$ (from ⊛) ∎

which gives bound of $D := \lfloor \log_\phi n \rfloor$ on degree of any vtx.

Finally, we will get a bound on amortized run time of extract-min operation.

For an extract-min operation, time taken is

$$O(\underbrace{\#\text{children of min element}}_{\leq D} + \underline{\#\text{trees present in heap}})$$

prove yourself

Total time taken for extract-min operations

$= O(\#\text{extract-min opns.} \times D + \#\text{trees created})$

↙ ↘

extract-min:
$\leq D$. # extract-min opns.

decrease-key:
$\leq$ # decrease-key opns.

$$= O(D \text{ \# extract min opns} + \text{\# decrease-key opns.})$$

(note that runtime of extract-min depends on # decrease-key opns. as well !)

$$= O(\log n \times n + m)$$

Total time on decrease-key opns. is $O(m)$

Hence runtime of algo $O(m + n \log n)$