## Lecture 14 – An efficient data structure for Dijkstra's algorithm

Recall that we saw an implementation where each Decrease-key operation costs $O(1)$ and the amortized cost of each Extract-min operation is $O(D)$.

- here $D$ is the maximum degree possible for any root node.

What we have to do now is to bound $D$.

- we would like to bound $D$ by $O(\log n)$ so that the total time taken by Dijkstra's algorithm is $O(m + n\log n)$.

Bounding $D$: We would like to show that if a root node has degree $D$, then the size of its subtree is $2^D$. This immediately implies that $D \le \log n$. Let us try to show this.

Let $x$ be a root node. Suppose $\text{degree}(x) = k$. How did $x$ come to have degree $k$?

- at some point in time, degree of $x$ was $k-1$ and $x$ then acquired another degree $k-1$ node as its child.

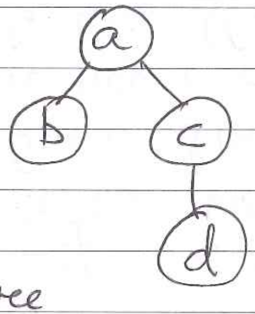
(Recall from the Clean-Up step that this is how the degree of a root node increases.)

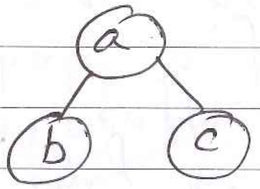
When $k=0$, it is certainly the case that a root node of degree $k$ has a subtree of size $2^k = 2^0 = 1$ (this tree consists of the node itself) rooted at itself.

- Assuming this claim (every node of deg. $k-1$ has a tree of size $2^{k-1}$ rooted at itself) for $k-1$, the claim holds for $k$ as well since $2^{k-1} + 2^{k-1} = 2^k$.

Except that we have forgotten the fact that we cut-off subtrees during decrease-key operation.

For example, say we had at some point and then we performed Decrease-Key on d, so this tree has become





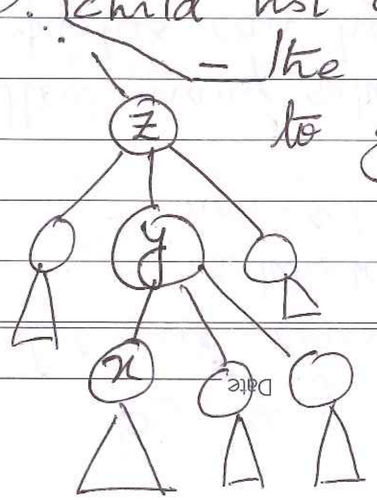
→ a's degree is 2 but $\underbrace{size(a)}_{\text{no. of nodes in a's tree}} = 3 < 4$

So how do we guarantee a link between the degree of a node and the size of the subtree rooted at this node?

The pseudo-code of Decrease-Key looks like:

Decrease-key $(x, k)$
- set $d[x] = k$
- $y = parent[x]$
- if $y \neq nil$ and $d[x] < d[y]$
  then $cut(x, y)$

Let us work on the $cut(x, y)$ subroutine now.
(r). Right now we simply remove $x$ from the child list of $y$ and add $x$ to the root list
- the modified subroutine wants a message to go upwards in the tree
  - it is OK for $y$ to lose 1 child but if $y$ loses another child, then $y$ will cut itself off from its parent and so on.
- So ultimately the root's degree will decrease if too many nodes in this tree

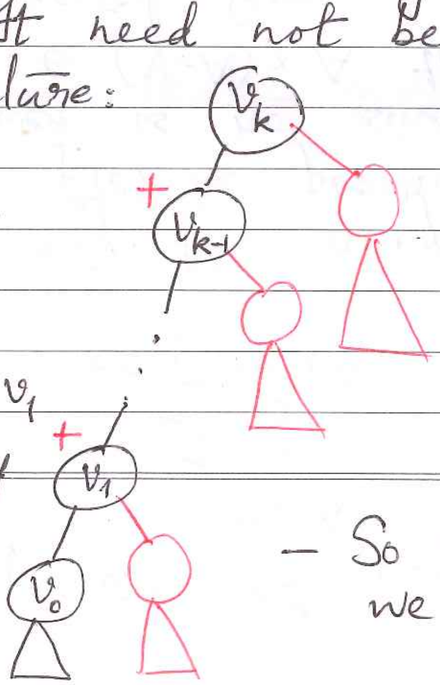


get cut-off. This is how we will maintain a link between the degree of a node and the size of the subtree rooted at this node.

Cut$(x, y)$
- Remove $x$ from the child list of $y$, decreasing degree$(y)$.
- Add $x$ to the root list.
- parent$(x)$ = nil and mark$(x)$ = false
- if $d[x] <$ MIN-value then
  update the MIN pointer
- if parent$(y) \neq$ nil then
  - if mark$(y)$ = false then
    - mark$(y)$ = true
  - else cut$(y$, parent$(y))$.

Initially mark$(u)$ = false for all vertices $u$. Whenever a node loses a child, its mark is set to true. Thereafter if it loses another child, it cuts itself off from its parent.
- for a root node $r$, mark$(r)$ = false.

What is the time taken by a single Decrease-Key operation? It need not be $O(1)$. Consider the following picture:




The red nodes denote children that $v_1, \ldots, v_{k-1}, v_k$ had in the past & they no longer have them as children.

Suppose we call cut$(v_0, v_1)$
→ so $v_0$ cuts itself off from $v_1$
→ $v_1$ is already marked. So cut$(v_1, v_2)$ is called, ...

- So in 1 Decrease-Key operation we may spend a lot of time.

Work done during a Decrease-Key(·,·) operation
= $O(1)$ + number of new roots created

Total work done during all Decrease-Key(·,·) operations = $O(m)$ + Total number of roots created during the entire algorithm

$\leq O(m)$ + Total number of marks set to true due to cut(·,·) subroutine

$\leq O(m) + \underbrace{\text{Total number of Decrease-Key opns.}}$ $= O(m)$

Every Decrease-key opn. sets at most 1 mark to true — note that this is totally analogous to the binary addition example

Thus the amortized cost of Extract-Min is $O(D)$ and the amortized cost of Decrease-Key is $O(1)$.

Let us again attempt to bound $D$
For each node $x$ ($x$ need not be in the root list), define $size(x)$ = number of nodes (incl. $x$) in the subtree rooted at $x$

We will show that $size(x) \geq F_{k+2}$ where $k = degree(x)$ and $F_{k+2} = (k+2)$-th Fibonacci number.

It is easy to show that $F_{k+2} \geq \phi^k \; \forall k \geq 2$, where $\phi$ is the golden ratio. So $size(x) \geq \phi^k$, this means

$k \leq \log_\phi(size(x))$. The value $\phi \geq 1.6$,

so $degree(x) = k \leq \log_{1.6} n = O(\log n)$.

Due to its connection with Fibonacci numbers, this data structure is called Fibonacci heap or F-heap. To summarize, an F-heap is a
* collection of min-heap ordered trees
* there is a MIN pointer pointing to the root with min d-value.
* all the roots are linked through a doubly linked list.

We have shown that the amortized cost of Extract-min is $O(D)$ and the amortized cost of Decrease-key is $O(1)$.

Bounding D

Claim 1. Let $x$ be any node in any F-heap. Suppose $degree(x) = k$. Let $y_1, \dots, y_k$ denote the children of $x$ in the order that they were linked to $x$, from the earliest to the latest. Then $degree(y_1) \geq 0$ and $degree(y_i) \geq i-2$ for $i \geq 2$.

Proof. It is obvious that $degree(y_1) \geq 0$. For $i \geq 2$, when $y_i$ got linked to $x$, $degree(x)$ at that time was at least $i-1$ since $y_1, \dots, y_{i-1}$ were already $x$'s children by then. Hence $degree(y_i)$ at that point in time was $\geq i-1$. Thereafter $y_i$ lost at most 1 child as it would have cut itself off from $x$ if it lost a second child. Hence $degree(y_i) \geq i-2$ now. □

Claim 2. Let $x$ be any node in an F-heap and let $degree(x) = k$. Then $size(x) \geq F_{k+2}$, where $F_t$ = $t$-th Fibonacci number.

$[F_0 = 0, F_1 = 1, F_t = F_{t-1} + F_{t-2}$ for $t \geq 2]$

*Proof.* Let $s_k$ denote the minimum possible value of size(z) over all nodes z with degree(z) = k.

Trivially $s_0 = 1$, $s_1 = 2$, $s_2 = 3$.

Let $y_1, \ldots, y_k$ denote the children of $x$ in the order that they were linked to $x$. To compute a lower bound on size($x$), we count 1 for $x$ itself and 1 for the first child $y_1$, giving us:

$$\text{size}(x) = 2 + \sum_{i=2}^{k} \text{size}_{\text{degree}(y_i)}$$

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

(by Claim 1)

We now show by induction on $k$ that $s_k \geq F_{k+2}$. Base cases: $k = 0, 1$ are trivially true.

For the induction step, we assume that $k \geq 2$ and $s_i \geq F_{i+2}$ for $i = 0, 1, \ldots, k-1$.

Recall that

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2}$$

$$\geq 2 + \sum_{i=2}^{k} F_i \quad \text{(by induction hypothesis)}$$

$$= \underbrace{1 + \sum_{i=0}^{k} F_i = F_{k+2}}_{\text{Please prove this by induction.}} \quad \square$$

Exercise. Show that

$F_{t+2} \geq \phi^t$ for all $t \geq 0$.

Use the fact that $\phi^2 = \phi + 1$.