Algorithm Theory

Chapter 5
Data Structures

Part V:
Fibonacci Heaps, Amortized Analysis

Fabian Kuhn

Cost of Delete-Min & Decrease-Key

Delete-Min:

1. Delete min. root r and add $r.child$ to $H.rootlist$

   time: $O(1)$

2. Consolidate $H.rootlist$

   time: $O(\text{length of } H.rootlist + D(n))$

- Step 2 can potentially be linear in n (size of H)

Decrease-Key (at node v):

1. If new key $<$ parent key, cut sub-tree of node v

   time: $O(1)$

2. Cascading cuts up the tree as long as nodes are marked

   time: $O(\text{number of consecutive marked nodes})$

- Step 2 can potentially be linear in n

Remark: Both operations can take $\Theta(n)$ time in the worst case!

Cost of Delete-Min & Decrease-Key

- Cost of delete-min and decrease-key can be $\Theta(n)$…
  - Seems a large price to pay to get insert in $O(1)$ time

- Maybe, the operations are efficient most of the time?
  - It seems to require a lot of operations to get a long rootlist and thus, an expensive consolidate operation
  - In each decrease-key operation, at most one node gets marked:
    We need a lot of decrease-key operations to get an expensive decrease-key operation

- Can we show that the average cost per operation is small?

$\Longrightarrow$ requires **amortized analysis**

Amortized Cost of Fibonacci Heaps

- **Initialize-heap**, **is-empty**, **get-min**, **insert**, and **merge** have **worst-case** and **amortized cost** $\boldsymbol{O(1)}$

- **Delete-min** has **amortized cost** $\boldsymbol{O(\log n)}$
- **Decrease-key** has **amortized cost** $\boldsymbol{O(1)}$

- Starting with an empty heap, any sequence of n operations with at most n_d delete-min operations has total cost (time)

$$\boldsymbol{T = O(n + n_d \log n).}$$

- We will now need the marks…

- Cost for Dijkstra & Prim/Jarník: $O(m + n \log n)$

Fibonacci Heaps: Marks

Cycle of a node:

1. Node v is removed from root list and linked to a node

   $$\boldsymbol{v.mark = \text{false}}$$

2. Child node u *of* v is cut and added to root list

   $$\boldsymbol{v.mark := \text{true}}$$

3. Second child of v is cut

   node v is cut as well and moved to root list

   $\boldsymbol{v.mark := \text{false}}$

The boolean value $v.mark$ indicates whether node v has lost a child since the last time v was made the child of another node.

Potential Function

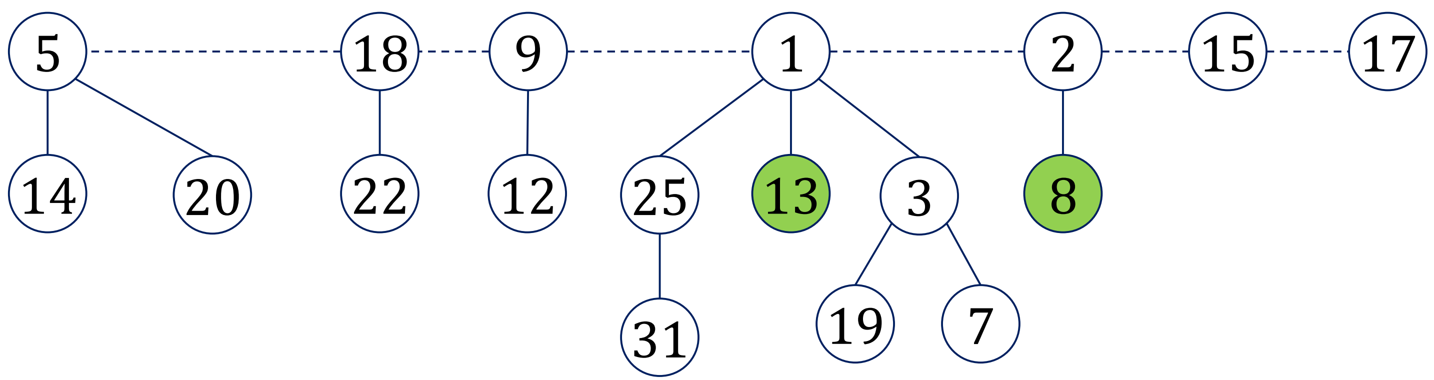
System state characterized by two parameters:

- $\boldsymbol{R}$: number of trees (length of $H.rootlist$)
- $\boldsymbol{M}$: number of marked nodes (not in the root list)

Potential function:

$$\boldsymbol{\Phi := R + 2M}$$

Example:



- $R = 7, M = 2 \rightarrow \Phi = 11$

Actual Time of Operations

- Operations: ***initialize-heap***, ***is-empty***, ***insert***, ***get-min***, ***merge***

  actual time: $O(1)$

  – Normalize unit time such that

  $$t_{init}, t_{is-empty}, t_{insert}, t_{get-min}, t_{merge} \leq 1$$

- Operation ***delete-min***:
  – Actual time: $O(\text{length of } H.rootlist + D(n))$
  – Normalize unit time such that

  $$t_{del-min} \leq D(n) + \text{ length of } H.rootlist$$

- Operation **descrease-key**:
  – Actual time: $O(\text{length of path to next unmarked ancestor})$
  – Normalize unit time such that

  $$t_{decr-key} \leq \text{length of path to next unmarked ancestor}$$

Amortized Times

Assume operation i is of type:

- **initialize-heap:**
  - actual time: $t_i \leq 1$, potential: $\Phi_{i-1} = \Phi_i = 0$
  - amortized time: $a_i = t_i + \Phi_i - \Phi_{i-1} \leq 1$

- **is-empty, get-min:**
  - actual time: $t_i \leq 1$, potential: $\Phi_i = \Phi_{i-1}$ (heap doesn't change)
  - amortized time: $a_i = t_i + \Phi_i - \Phi_{i-1} \leq 1$

- **merge:**
  - Actual time: $t_i \leq 1$
  - combined potential of both heaps: $\Phi_i = \Phi_{i-1}$
  - amortized time: $a_i = t_i + \Phi_i - \Phi_{i-1} \leq 1$

Amortized Time of Insert

Assume that operation i is an *insert* operation:

- **Actual time:** $t_i \leq 1$
- **Potential function:**
  - M remains unchanged (no nodes are marked or unmarked, no marked nodes are moved to the root list)
  - R grows by 1 (one element is added to the root list)

$$M_i = M_{i-1}, \qquad R_i = R_{i-1} + 1$$
$$\Phi_i = \Phi_{i-1} + 1$$

- **Amortized time:**

$$\boldsymbol{a_i = t_i + \Phi_i - \Phi_{i-1} \leq 2}$$

Amortized Time of Delete-Min

Assume that operation i is a *delete-min* operation:

Actual time: $t_i \leq D(n) + |H.rootlist|$

Potential function $\boldsymbol{\Phi = R + 2M}$:

- R: changes from $|H.rootlist|$ to at most $D(n) + 1$
- M: (# of marked nodes that are not in the root list)
  - Number of marks does not increase

$$M_i = M_{i-1}, \qquad R_i \leq R_{i-1} + D(n) + 1 - |H.rootlist|$$
$$\Phi_i \leq \Phi_{i-1} + D(n) + 1 - |H.rootlist|$$

Amortized Time:

$$\boldsymbol{a_i = t_i + \Phi_i - \Phi_{i-1} \leq 2D(n) + 1}$$

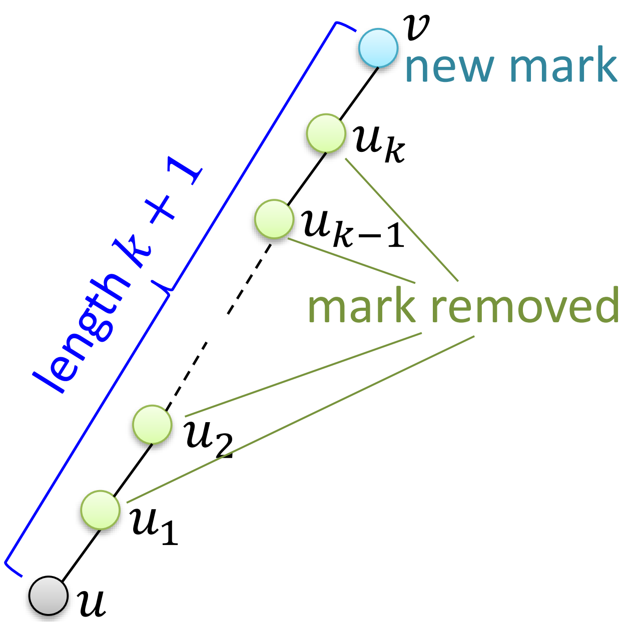
Amortized Time of Decrease-Key

Assume that operation i is a *decrease-key* operation at node u:

Actual time: $t_i \leq$ length of path to next unmarked ancestor v

Potential function $\Phi = R + 2M$:

- Assume, node u and nodes $u_1, \dots, u_k$ are moved to root list
  - $u_1, \dots, u_k$ are marked and moved to root list, $v.\text{mark}$ is set to true



marks	root list
Removed marks: $u_1, \dots, u_k$ (and u, if u is marked) Added mark: v $\boldsymbol{M_i - M_{i-1} \leq -(k-1)}$	Added to root list: $u, u_1, \dots, u_k$ $\boldsymbol{R_i - R_{i-1} = k+1}$

Amortized Time of Decrease-Key

Assume that operation i is a *decrease-key* operation at node u:

Actual time: $t_i \leq \text{length of path to next unmarked ancestor } v$

Potential function $\boldsymbol{\Phi = R + 2M}$:

- Assume, node u and nodes $u_1, \dots, u_k$ are moved to root list
  - $u_1, \dots, u_k$ are marked and moved to root list, $v.\text{mark}$ is set to true
- $\geq k$ marked nodes go to root list, ≤ 1 node gets newly marked
- R grows by $\leq k + 1$, M grows by 1 and is decreased by $\geq k$

$$R_i \leq R_{i-1} + k + 1, \qquad M_i \leq M_{i-1} + 1 - k$$
$$\Phi_i \leq \Phi_{i-1} + (k + 1) - 2(k - 1) = \Phi_{i-1} + 3 - k$$

Amortized time:

$$\boldsymbol{a_i = t_i + \Phi_i - \Phi_{i-1} \leq k + 1 + 3 - k = 4}$$

Complexities Fibonacci Heap

- **Initialize-Heap**: $\boldsymbol{O(1)}$
- **Is-Empty**: $\boldsymbol{O(1)}$
- **Insert**: $\boldsymbol{O(1)}$
- **Get-Min**: $\boldsymbol{O(1)}$
- **Delete-Min**: $\boldsymbol{O(D(n))}$
- **Decrease-Key**: $\boldsymbol{O(1)}$

amortized

- **Merge** (heaps of size m and n, $m \leq n$): $\boldsymbol{O(1)}$

- **How large can $D(n)$ get?**

Rank of Children

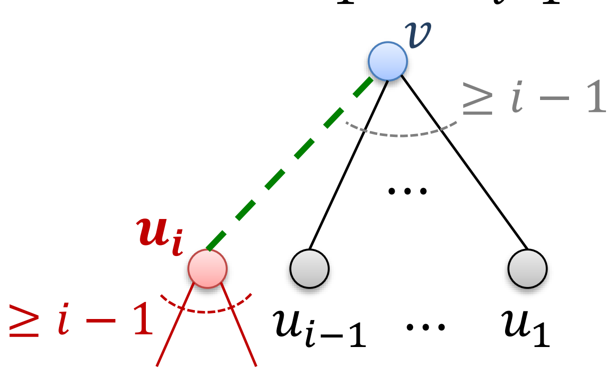
Lemma:

Consider a node v of rank k and let $u_1, \dots, u_k$ be the children of v in the order in which they were linked to v. Then,

$$\boldsymbol{rank(u_i) \geq i - 2}.$$

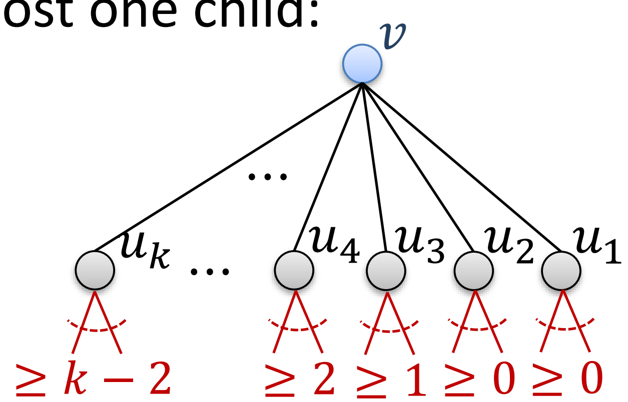
Proof:

When u_i is added, v already has children $u_1, \dots, u_{i-1}$:



$\Longrightarrow rank(u_i) \geq i - 1$ when u_i is linked to v.

Each node can lose at most one child:



$\Longrightarrow rank(u_i) \geq i - 2$ as long as u_i is linked to v.

Size of Trees

Fibonacci Numbers:

$$F_0 = 0, \qquad F_1 = 1, \qquad \forall k \geq 2: F_k = F_{k-2} + F_{k-1}$$

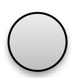
Lemma:

In a Fibonacci heap, the size of the sub-tree of a node v with rank k is at least F_{k+2}.

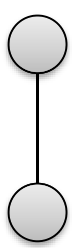
Proof:

- S_k: minimum size of the sub-tree of a node of rank k

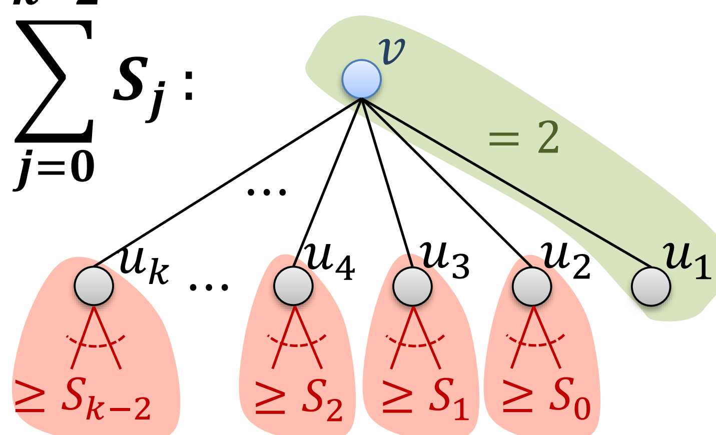
$\boldsymbol{S_0 = 1}$:



$\boldsymbol{S_1 = 2}$:



$$\boldsymbol{S_k \geq 2 + \sum_{j=0}^{k-2} S_j}:$$



Size of Trees

$$S_0 = 1, \qquad S_1 = 2, \qquad \forall k \geq 2: S_k \geq 2 + \sum_{i=0}^{k-2} S_i$$

Claim about Fibonacci numbers:

$$\forall k \geq 0: F_{k+2} = 1 + \sum_{i=0}^{k} F_i \qquad (F_0 = 0, F_1 = 1)$$

Proof of claim (by induction on k):

- **Base case ($k = 0$):** $F_2 = 1 + \sum_{i=0}^{0} F_i = 1 + F_0 = 1$
- **Induction step ($k > 0$):**

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

I.H.: $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$

Size of Trees

$$S_0 = 1, S_1 = 2, \forall k \geq 2: S_k \geq 2 + \sum_{i=0}^{k-2} S_i, \qquad F_{k+2} = 1 + \sum_{i=0}^{k} F_i$$

Claim of lemma: $S_k \geq F_{k+2}$

Proof by induction on k:

- Base case ($k = 0, k = 1$): $\quad S_0 \geq F_2 = 1 \qquad S_1 \geq F_3 = 2$
- Induction step ($k > 1$):

$$S_k \geq 2 + \sum_{i=0}^{k-2} S_i \geq 2 + \sum_{i=0}^{k-2} F_{i+2} = 2 + \sum_{j=2}^{k} F_j = 1 + \sum_{j=0}^{k} F_j = F_{k+2}$$

I.H.

$j = i + 2$

$F_0 = 0, F_1 = 1$

previous claim on Fibonacci numbers

Size of Trees

Lemma:

In a Fibonacci heap, the size of the sub-tree of a node v with rank k is at least F_{k+2}.

Theorem:

The maximum rank of a node in a Fibonacci heap of size n is at most

$$\boldsymbol{D(n) = O(\log n)}\,.$$

Proof:

- The Fibonacci numbers grow exponentially:

$$F_k = \frac{1}{\sqrt{5}} \cdot \left(\left(\frac{1+\sqrt{5}}{2} \right)^k - \left(\frac{1-\sqrt{5}}{2} \right)^k \right)$$

- For $D(n) \geq k$, we need $n \geq F_{k+2}$ nodes.

Binary Heaps & Fibonacci Heaps

	Binary Heap	Fibonacci Heap
initialize	$O(1)$	$O(1)$
insert	$O(\log n)$	$O(1)$
get-min	$O(1)$	$O(1)$
delete-min	$O(\log n)$	$O(\log n)$ *
decrease-key	$O(\log n)$	$O(1)$ *
is-empty	$O(1)$	$O(1)$

* **amortized time**