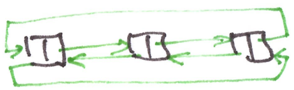
Fibonacci heaps

- kolekce stromů, z nichž každý splňuje min-heap podmínku (každý uzel má klíč větší než jeho rodič, pokud rodiče má).

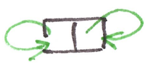
```
struct {
    key,      // klíč
    left,     // pointery sloužící pro obousměrný
    right,    // cyklický seznam
    child,    // pointer na libovolného ~~rodiče~~ potomka
    degree,   // počet potomků
    mark,     // už dítě (true/false)
      P,      // pointer na rodiče.
}
```

- Potomci uzlu jsou pomocí left a right zapojeni do obousměrného cyklického seznamu.



Rodič si tento seznam „pamatuje" pomocí pointeru child, který ukazuje na libovolného potomka.

Seznam potomků je bez zarážky, pokud má pouze jeden prvek, pak

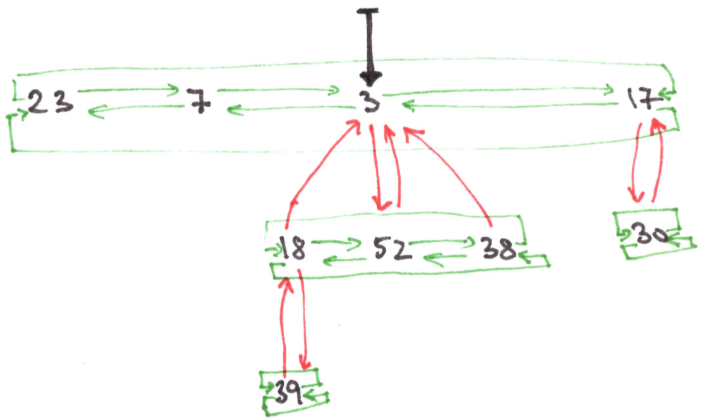


- Na pořadí potomků v seznamu nezáleží

- stromy ve frontě (jejich kořeny) jsou pomocí left a right zapojeny do obousměrného cyklického seznamu bez zarážky, (podobně jako seznamy potomků)

- uchováváme si pointer na kořen s nejmenším klíčem

Př.




```
struct {      // struktura pro heap
    min       // pointer na kořen s min. klíčem
    n         // počet uzlů
}
```

I) Operace, ve kterých je FH pouze „neuspořádaná" BH.

- neuspořádané binomické stromy
- halda může obsahovat více stromů stejného stupně (v kořeni)
- opravu předchozího bodu odkládáme na co nejpozdější dobu.

A) neuspořádané operace:

Insert (H, x)

1) inicializujeme položky x na

$x.degree \leftarrow 0$
$x.mark \leftarrow false$
$x.child \leftarrow nil$
$x.p \leftarrow nil$

2) spojíme seznamy s prvkem x a $H.min$
3) pokud $H.min.key > x.key$, pak $H.min \leftarrow x$.
4) $H.n \leftarrow H.n + 1$

UNION (H_1, H_2)

1) Spojíme seznamy $H_1.min$ a $H_2.min$
2) Výsledná Halda H má

$$H.min \leftarrow \begin{cases} H_1.min & H_1.min.key < H_2.min.key \\ H_2.min & \text{jinak} \end{cases}$$

$H.n \leftarrow H_1.n + H_2.n$.

Obě operace jenom v konstantním čase spojí dva seznamy a vyberou nový kořen s min prvkem.

Operace $MIN(H)$ je také v konstantním čase, jenom vrátíme $H.min$.

B) OPERACE, KTERÁ UKLÍZÍ

EXTRACT-MIN (H)

$z \leftarrow H.min$ // pokud je $z == nil$, konec!

1) spojíme seznamy $z.child$ a $H.min$
2) odstraníme z ze seznamu $H.min$, přičemž nová hodnota | + zmenšíme $H.n$ o jedna

$$H.min \leftarrow \begin{cases} z.right & z.right \neq z \\ nil & \end{cases}$$

to se stane, když byl seznam prázdný!

3) zavoláme CONSOLIDATE(H).

- Úlohou procedury consolidate je zařídit, aby všechny stromy v haldě měli různý stupeň. (= jejich kořeny)

- procedura předpokládá existenci ~~konstanty~~

  $D : \mathbb{N} \to \mathbb{N}$,

  která říká, že max. stupeň ~~stro~~ kořene stromu v haldě s n prvky je $D(n)$.

  Později ukážeme, že $D(n) \in O(\log n)$.

- nejdříve pomocná procedura

  HEAP-LINK(H, y, x)

  1) odstraň y ze seznamu H.min
  2) zapoj y jako potomka x (a zvyš x.degree o 1)
  3) nastav y.mark ← false // toto pochopíme později.

- procedura consolidate dělá v podstatě dva kroky, dokud to jde:

  1) nalezne dva stromy s kořeny x a y takové, že x.degree = y.degree a x.key < y.key
  2) zavolá HEAP-LINK(H, y, x)

Za účelem efektivního hledání v kroku 1 používá procedura pomocné pole A s $D(H.n)+1$ prvky. Účel tohoto pole je uchovávat v políčku A[i] pointer na strom s kořenem, jehož stupeň je i, nebo je tam NIL.

CONSOLIDATE(H)

1. inicializujeme pole A na NIL (= všechny políčka jsou NIL)

2. procházíme uzly v seznamu H.min dokud neprojdeme celý seznam a ~~ne~~ nedojde k zavolání HEAP-LINK. Akt. uzel označíme w.

   ~~Pro~~ jedna iterace

   ~~3a~~ x ← w
   d ← w.degree

   TOTO JE HLAVNÍ TRIK: připojujeme k x stromy se stejným stupněm kořene, dokud to jde

   dokud A[d] ≠ NIL
     y ← A[d]
     pokud x.key > y.key, prohodíme hodnoty proměnných x a y
     ~~FIB HEAP~~
     HEAP-LINK(H, y, x)
     A[d] ← NIL
     d ← d + 1

   Tady se nedá spojit, nebo jsme nenašli potřebný strom.

   A[d] ← x
   w ← x.~~le~~right // tady se posuneme s w

3. ~~Všechny stromy~~ Na všechny stromy se odkazujeme v poli A, sestavíme je do seznamu a najdeme minimum.

Skutečná složitost consolidate:

– $D(n)$ ... max stupeň kořene stromu
$t(H)$ .... počet ~~kořů~~ stromů v haldě H

před zavoláním consolidate má seznam

$$D(n) + t(H) - 1$$

stromů.

– v bodu 2 algoritmu ~~projde~~ navštívíme každý ~~uzel~~ kořen konstantní počet krát: buď ho nastavíme do A nebo ho zapojíme jako potomka. Složitost je tedy $O(D(n) + t(H))$

Amortizovaná složitost všech operací

$t(H)$ .... počet stromů v H
$m(H)$ .... počet uzlů s položkou marked rovnou true
(zatím je vždy ~~ta~~ tato položka false, její použití uvidíme později)

Potenciál H definujeme jako

$$\phi(H) = t(H) + 2m(H).$$

(vidíme, že $\phi(H) \geq 0$).

INSERT

skutečná složitost .. $O(1)$

změna potenciálu (zvedneme počet ~~prvků~~ stromů o 1, jinak se nic nezmění) je 1.

Amortizovaná složitost je $O(1)$.

UNION

skutečná složitost -- $O(1)$

H... nový heap, H_1, H_2... argumenty

$$\phi(H) - (\phi(H_1) + \phi(H_2)) = 0,$$

protože neměníme počet markovaných prvků a $t(H) = t(H_1) + t(H_2)$.

Amortizovaná složitost je také $O(1)$.

EXTRACT-MIN

potenciál po provedení je nejvýše

$$(D(n) + 1) + 2m(H)$$

Amortizovaná složitost je nejvýše

$$\underbrace{O(D(n) + t(H))}_{\text{skutečná slož}} + \Big[\underbrace{(D(n) + 1 + 2m(H))}_{\text{potenciál po}} - \underbrace{(t(H) + 2m(H))}_{\text{potenciál před}}\Big] =$$

$$= O(D(n)) + O(t(H)) - t(H) = O(D(n)).$$

jednotku potenciálu nastavujeme tak, aby zaplatila konstantu v O-notaci

Věta: Pokud provádíme pouze INSERT, UNION, EXTRACT-MIN, je $D(n) = \lfloor \lg n \rfloor$.

Důkaz: Všechny operace zachovávají to, že stromy v haldě jsou neuspořádané binomiální stromy. ~~[illegible]~~ Neuspořádaný B_k má 2^k prvků, musíme mít $2^k \le n$, pokud je B_k v haldě. Odtud $k \le \lfloor \lg n \rfloor$ (k je celé číslo!). ∎

II Operace, které způsobí existenci stromů, které nejsou binomiální

```
DECREASE-KEY(H, x, k)
    assert(x.key > k)
    x.key ← k
    y ← x.p
    if y ≠ nil and x.key < y.key    // porušení heap podmínky
        CUT(H, x, y)
        CASCADING-CUT(H, y)
    if x.key < H.min.key
        H.min ← x
```

Úpravy v Haldě dělají následující procedury

CUT(H, x, y)

1) odstraníme x ze seznamu potomků uzlu y
2) přidáme x do seznamu H.min
3) x.p ← NIL, ~~[illegible]~~
4) x.mark ← FALSE
5) y.degree ← y.degree − 1

Procedura ~~[illegible]~~ odstraní x ze stromu a přesune jej do seznamu kořenů. Spolu s x se přesune i celý podstrom.

Abychom udrželi strom „blízko" binomiálnímu, pokud nějakému uzlu odstraníme 2 potomky, vyřízneme jej taky. (to platí pro uzly mimo kořenů) (kořen nemáme kam přesunout).

```
CASCADING-CUT(H, y)
    z ← y.p
    if z ≠ nil
        ~~then~~ if y.mark = ~~true~~ False
            y.mark ← true
        else
            CUT(H, y, z)
            CASCADING-CUT(H, z)
```

Amortizovaná složitost DECREASE-KEY

- skutečná složitost je $O(c)$, kde c je počet rekurzivních zavolání procedury CASCADE-CUT

- ~~[illegible]~~ změna potenciálu.
  jedno zavolání CASCADING-CUT přidá ~~1~~ 1 strom
  (poslední zavolání strom nepřidá)
  a nastaví mark na ~~true~~ false ~~[illegible]~~,
  ~~[illegible]~~ nastaví na true jeden mark.
  poslední zavolání může

změna celkem je: - přidáme c stromů
(c-1 CASCADE-CUT,
1 PRVNÍ-CUT)

ubude $c-2$ marked uzlů

změna potenciálu je

$$\Phi(H_1)\ [t(H) + c + 2(m(H) - c + 2)] - (t(H) + 2m(H)) =$$

$$= 4 - c$$

Amortizovaná složitost je tak

$$O(c) + 4 - c = O(1)$$

jednotku potenciálu můžeme naškálovat na konstantu v O notaci

Poslední věc je nutnost omezit $D(n)$, i když povolíme operaci DECREASE-KEY

def: pro uzel x ve stromě definujeme size(x) jako počet uzlů ve stromě s kořenem x (bráno jako podstrom generovaný x a všemi jeho následníky)

Lemma: x... uzel ve Fibonacciho haldě. Předp. $x.degree = k$ a označme

$y_1, y_2, y_3, \dots y_k$

potomky x v pořadí, ve kterém byli zapojeni jako potomci x (v proceduře consolidate). Potom

$(y_1).degree \geq 0,$ (1)

$(y_i).degree \geq i-2,$ (2)

pro $i = 2, 3, 4, \dots k$

Důkaz: (1) je triviální.

(2) V momentě připojení y_i měl uzel x ~~později~~ potomky $y_1, \dots y_{i-1}$, takže $x.degree = i-1$. Proc. consolidate připojila y_i k uzlu x v momentě, kdy $x.degree = (y_i).degree$. Po připojení tedy bylo $(y_i).degree = i-1$. Potom mohl uzel y_i ztratit nejvýše jednoho potomka (jinak by ho cascading-cut odstranil). Tedy $(y_i).degree \geq i-2$.

∎

Připomeneme Fibonacciho čísla

$$F_k = \begin{cases} 0 & k = 0 \\ 1 & k = 1 \\ F_{k-1} + F_{k-2} & k \geq 2 \end{cases}$$

Lemma: $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$

Důkaz:

indukcí při k.

1) $k = 0$. $1 + \sum_{i=1}^{0} F_i = 1 + F_0 = 1 = F_2$

2) Předp. $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$, potom

$$F_{k+2} = F_k + F_{k+1} =$$

$$= F_k + \left(1 + \sum_{i=0}^{k-1} F_i\right)$$

$$= 1 + \sum_{i=0}^{k} F_i$$

∎

Lemma:

$x \dots$ uzel ve Fib. haldě, $x.\text{degree} = k$. Potom

$\text{size}(x) \geq F_{k+2}$

Důkaz:

ozn. s_k min. možnou hodnotu $\text{size}(z)$ pro uzly z s $z.\text{degree} = k$. (Napr. $s_0 = 1$, $s_1 = 2$, $s_2 = 3 \dots$)

triviálně $s_k \leq \text{size}(x)$. Hodnota s_k je rostoucí v závislosti na k.

Předp. že x má potomky $y_1, y_2 \dots y_k$ očíslované v pořadí podle toho, jak je procedura consolidate přidala k x.

$$\text{size}(x) \geq s_k$$

$$= \underbrace{1}_{s^x} + \underbrace{1}_{s_1} + \sum_{i=2}^{k} s_{(y_i).\text{degree}}$$

$\text{size}(y_1) \geq 1$

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

použila se
1) ~~y_i size~~ $(y_i).\text{degree} \geq i-2$ z lemmatu
2) ~~s_{degree}~~ $s_{(y_i).\text{degree}} \geq s_{i-2}$ díky monotonii s_k !

Indukcí dokážeme $s_k \geq F_{k+2}$.

Triviální pro $k=0$, $k=1$.

Obecně předp. že $s_i \geq F_{i+2}$ pro $i = 0, 1, \dots k-1$. Dokážeme pro k.

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

$$\geq 2 + \sum_{i=2}^{k} F_i = 1 + \sum_{i=0}^{k} F_i = F_{k+2}.$$

$(F_0 + F_1 = 1 \,!)$

∎

Důsledek:

Maximální stupeň $D(n)$ uzlu ve Fib. haldě s n uzly je $O(\lg n)$.

Důkaz:

1) Platí, že $F_{k+2} \geq \phi^k$, kde $\phi = (1+\sqrt{5})/2 = 1.61803\ldots$ a podle předchozího lemmatu $size(x) \geq \phi^k$, kde $x.degree = k$, platí pro každý uzel x. Tedy max stupeň uzlu je $\lfloor \log_\phi n \rfloor$.

∎