# Pokročilá algoritmizace

## amortizovaná složitost, Fibonacciho halda, počítačová aritmetika

Jiří Vyskočil, Marko Genyg-Berezovskyj

2009

# Amortizovaná složitost

- Asymptotická složitost často dostatečně nevypovídá o složitosti algoritmů, které se spouštějí v sekvencích, a u kterých je jejich složitost proměnlivá, závislá na vnitřním stavu jejich datových struktur.
- **Amortizovaná časová** složitost je průměrný čas potřebný pro vykonání určité operace v sekvenci operací v nejhorším případě. Na rozdíl od *časové složitosti v průměrném případě* nevyužívá pravděpodobnosti. U amortizované složitosti je průměrný čas na operaci skutečně zaručený.
- Základní myšlenka amortizované složitosti spočívá v tom, že operace s nejhorší složitostí změní stav datové struktury algoritmu tak, že tento nejhorší případ nemůže nastávat příliš často, tudíž *amortizuje* svou cenu.

# Amortizovaná složitost

- příklad
  - Složitost vkládání prvků do tzv. *dynamického pole*.
  - **dynamického pole** je pole, které zdvojnásobuje svou velikost pokaždé, když dojde k jeho naplnění.
  - Samotné vkládání prvků (bez nutnosti realokace) vyžaduje čas O(1), pro $N$ prvků tedy také $O(N)$.
  - V případě naplnění pole je nutná realokace. V nejhorším případě tato operace potřebuje čas až $O(N)$.
  - Pro vložení $N$ prvků (včetně realokace) je tedy potřeba v nejhorším případě $O(N) + O(N) = O(N)$.
  - Amortizovaný čas na jedno vložení prvku je pak $O(N)/N = O(1)$.

# Fibonacciho halda

- **Fibonacciho halda** je druh haldy, který Principiálně vychází z binomiální haldy.
- Hlavní výhodou Fibonacciho haldy je nízká asymptotická složitost prováděných algoritmů.
- Operace Insert, AccessMin a Merge probíhají v O(1).
- Operace DecreaseKey probíhá v amortizovaném konstantním čase.
- Operace Delete a DeleteMin pracují s amortizovanou časovou složitostí O(log($n$)).
- Užití Fibonacciho haldy není vhodné pro real-time systémy, protože některé operace mohou mít v nejhorším případě lineární složitost.

# Fibonacciho halda

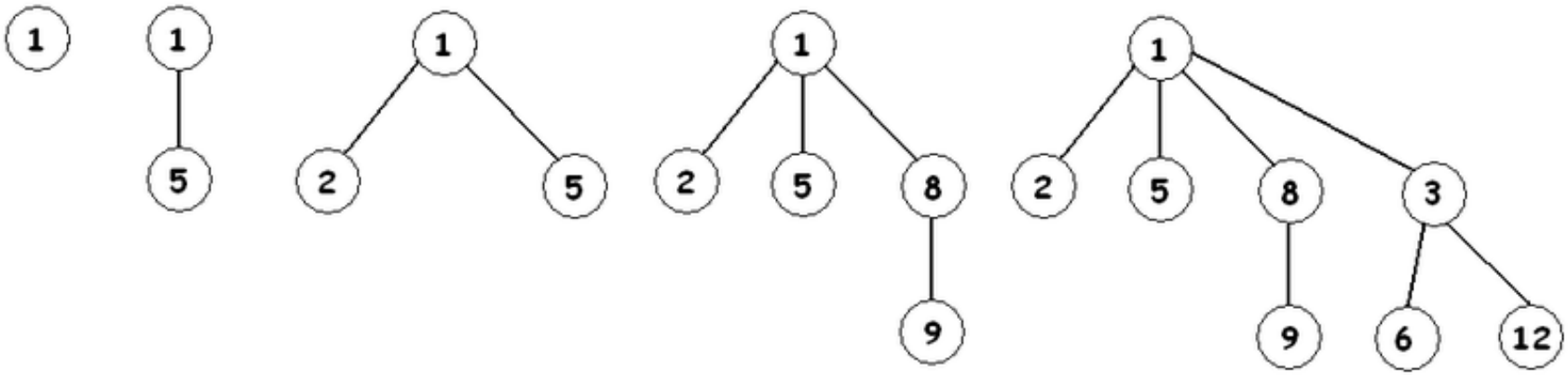
- Fibonacciho haldu tvoří skupina stromů vyhovující lokální podmínce na uspořádání haldy, která vyžaduje, aby pro každý uzel stromu platilo, že prvek, který reprezentuje, je menší než prvek reprezentovaný jeho potomky. Z této podmínky vyplývá, že minimálním prvkem je vždy kořen jednoho ze stromů.
- Vnitřní struktura Fibonacciho haldy je v porovnání s binomiální haldou daleko více flexibilní. Jednotlivé stromy nemají pevně daný tvar a v extrémním případě může každý prvek haldy tvořit izolovaný strom nebo naopak všechny prvky mohou být součástí jediného stromu hloubky $N$. Tato flexibilní struktura umožňuje velmi jednoduchou implementaci operací s haldou.
- Operace, které nejsou potřebné, odkládáme a vykonáváme je až v okamžiku, kdy je to nevyhnutelné, například spojení nebo vložení nového prvku se jednoduše provede spojením kořenových seznamů (s konstantní náročností) a jednotlivé stromy spojíme až při operaci snížení hodnoty klíče.

# Fibonacciho halda

- každý vrchol má nejvýše $\log(n)$ synů a velikost stromu řádu $k$ je nejméně $F_{k+2}$, kde $F_k$ je $k$-té Fibonacciho číslo. Kořen každého stromu řádu $k$ má právě $k$ potomků.

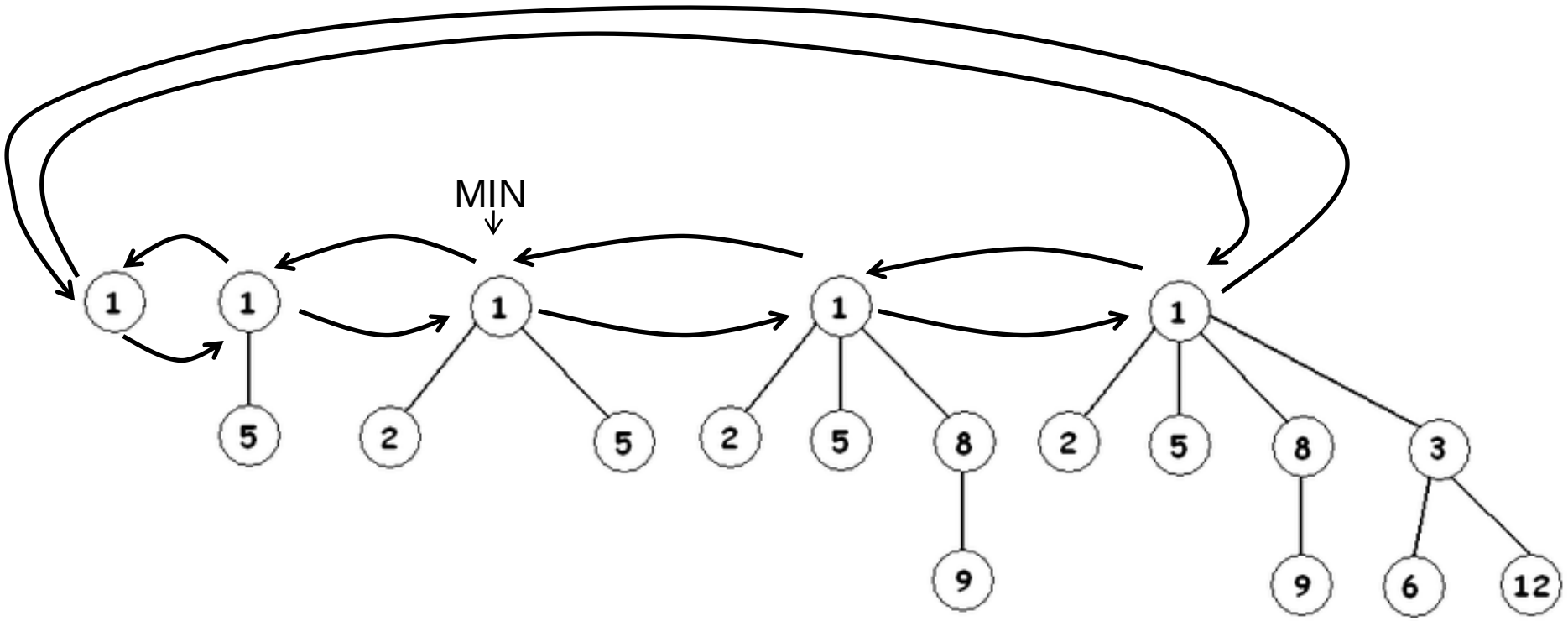
$$F_n = \begin{cases} 0, & \text{pro } n = 0; \\ 1, & \text{pro } n = 1; \\ F_{n-2} + F_{n-1} & \text{jinak.} \end{cases}$$

- Toto je dosaženo díky pravidlu, které dovoluje oříznout nejvýše jednoho syna od každého nekořenového prvku.
- Pokud je odříznut druhý syn, vrchol musí být odříznut od svého otce a stává se kořenem nového stromu.
- Počet stromů je snižován při operaci DeleteMin, kdy dochází ke spojování stromů.



# Fibonacciho halda - reprezentace

- Jednotlivé stromy haldy jsou propojeny dvojitým kruhovým spojovým seznamem.



# Fibonacciho halda - reprezentace

- ***N*** je aktuální počet prvků v haldě.
- ***MIN*** je ukazatel na strom jehož kořen obsahuje minimální prvek haldy
- **key(*x*)** je hodnota klíče vrcholu ***x***.
- **mark(*x*)** je boolovská hodnota vrcholu ***x***. Je to pomocná značka pro odebírání vrcholů. Pokud je nastavena na **true** nesmíme již žádného potomka ***x*** ze stromu odebírat.

- **descendants(*x*)** vrací všechny potomky ***x***.
- **parent(*x*)** vrací rodiče ***x***. Pro prvek ***x***, který nemá rodiče vrací *x*.

# Fibonacciho halda - Merge, Insert

- **Merge** ($H_1$, $H_2$)
  - Propojí oba dvojité kruhové seznamy do jednoho a updatuje ukazatel na *MIN*.
  - O(1)
- **AccessMin**
  - Vrátí prvek reprezentovaný kořenem stromu, na nějž ukazuje *MIN ukazatel*.
  - O(1)
- **Insert** ($x$)
  - Vytvoří se strom řádu 0, tedy jediný vrchol reprezentující přidávaný prvek $x$. Z tohoto stromu vytvoří Fibonacciho haldu.
  - Nastaví mark($x$) na false.
  - Poté se zavolá **Merge** na obě haldy.
  - O(1)

# Fibonacciho halda - DeleteMin

- **DeleteMin**

```
1.  z = MIN;
2.  if z ≠ null then {
3.      for each x ∈ descendants(z) do
4.          Vlož x do spojového seznamu stromů haldy;
5.      Vyjmi z ze spojového seznamu stromů haldy;
6.      if N = 1 then
7.          MIN = null
8.      else {
9.          MIN = ukazatel na libovolný kořen stromu v haldě;
10.         Konsolidace;
11.     }
12.     N--;
13. }
```

# Fibonacciho halda - Konsolidace

## Konsolidace

1. **for** i = 0 **to** max. možný řád stromu ve Fibonacciho haldě velikosti $N$ **do** A[i] = **null**;
2. **for each** $s \in$ všechny stromy v haldě **do {**
3. $x$ = umístění kořene s; $d$ = řád stromu $s$;
4. **while** A[$d$] $\neq$ **null do {**
5. $y$ = A[$d$];
6. **if** key($x$) > key($y$) **then** prohoď $x$ a $y$;
7. Vyjmi $y$ z haldy a připoj toto $y$ k vrcholu $x$ jako potomka;
8. mark($y$) = **false**; A[$d$] = **null**; $d$++;
9. **}**
10. A[$d$] = $x$;
11. **}**
12. $MIN$ = **null;**
13. **for** i = 0 **to** max. řád stromu v poli A **do**
14. **if** A[i] $\neq$ **null then {**
15. vlož A[i] do spojového seznamu stromů haldy;
16. **If** ($MIN$ = **null**) **or** (key(A[i]) < key($MIN$)) **then** $MIN$ = A[i];
17. **}**

# Fibonacciho halda - DecreaseKey, Delete

- **DecreaseKey** ($x$,$d$)
  1. key($x$) = key($x$) – $d$;
  2. $y$ = parent($x$);
  3. **if** ($x \neq y$) **and** (key($x$) < key($y$)) **then {**
  4. **Řez**($x$,$y$);
  5. **KaskádovýŘez**($y$);
  6. **}**
  7. **If** key($x$) < key(*MIN*) **then** *MIN* = $x$;

- **Delete**($x$)
  1. Pomocí DecreaseKey($x$,∞) snížíme hodnotu $x$ na -∞;
  2. DeleteMin;

- **Řez**($x$,$y$)
  8. Odřízni podstrom $x$ z potomků $y$;
  9. Vlož $x$ do spojového seznamu haldy;
  10. mark($x$) = **false**;

- **KaskádovýŘez**($y$)
  1. $z$ = parent($y$);
  2. **if** ($y \neq z$) **then**
  3. **if** mark(y) = **false then** mark(y) = **true**
  4. **else {**
  5. **Řez**(y,z);
  6. **KaskádovýŘez**(z);
  7. **}**

# Haldy - shrnutí

| | binární halda | d-regulární halda | binomiální halda | Fibonacciho halda |
|---|---|---|---|---|
| AccessMin | O(1) | O(1) | O(1) | O(1) |
| DeleteMin | O(log(n)) | O(log(n)) | O(log(n)) | O(n)<br>amortizovaně:<br>O(log(n)) |
| Insert | O(log(n)) | O(log(n)) | O(log(n))<br>amortizovaně:<br>O(1) | O(1) |
| Delete | O(log(n)) | O(log(n)) | O(log(n)) | O(n)<br>amortizovaně:<br>O(log(n)) |
| Merge | O(n) | O(n) | O(log(n)) | O(1) |
| DecreaseKey | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n))<br>amortizovaně:<br>O(1) |

# Reprezentace přirozených čísel

- Nejčastější reprezentací přirozených čísel v počítači je číslo v binární soustavě:

$$\text{hodnota čísla} = \sum_{i=0}^{n} b_i \times 2^i$$

kde $n$ je počet bitů čísla a $b_i$ je hodnota $i$-tého bitu.

- Často se ještě používá tzv. BCD (Binary Coded Decimal) notace, která kóduje číslo v desítkové soustavě po jednotlivých číslicích jako čtveřicích bitů (nibblů).
  - není tak efektivní (všechny možné kombinace čtveřice bitů nejsou využity)
  - často se používá v bankovních aplikacích, protože přesně koresponduje s desítkovou soustavou.

# Reprezentace celých čísel

- Doplňková reprezentace (nejčastější):

$$\text{hodnota čísla } N = \begin{cases} \sum_{i=0}^{n-1} b_i \times 2^i & \text{pro } b_n = 0, \text{ tedy } N \in \langle 0; 2^{n-1} - 1 \rangle \\ -1 - \sum_{i=0}^{n-1} (1 - b_i) \times 2^i & \text{pro } b_n = 1, \text{ tedy } N \in \langle -2^{n-1}; -1 \rangle \end{cases}$$

| most-significant bit | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | = 127 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | = 126 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | = 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | = 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | = 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | = −1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | = −2 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | = −127 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | = −128 |

8-bit two's-complement integers

- Na operace sčítání a odčítání lze použít stejné algoritmy jako u předchozí binární reprezentace přirozených čísel.
- Znaménko +/- lze zjistit z nejvyššího bitu.
- V reprezentaci je pouze jedna nula.

# Reprezentace čísel s pohyblivou řádovou čárkou

- reprezentace:
  - $$s \times \frac{c}{b^{p-1}} \times b^e$$

kde $s$ je signum (znaménko +/-)

$c$ je mantisa (fraction)

$b$ je základ (base) číselné soustavy (nejčastěji 2 nebo 10)

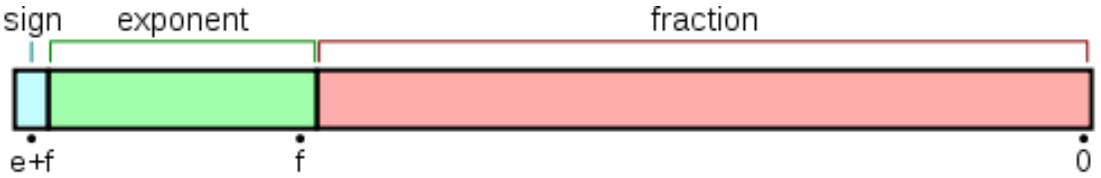
$p$ je přesnost mantisy (počet číslic mantisy)

$e$ je celočíselný exponent

  - Do tohoto formátu bychom ještě chtěli zakódovat +∞ a -∞.
- Pokud je $b$=2 (nejčastější případ) dochází při zpracování vstupů a výstupů k některým problémům způsobeným konverzí z/do desítkové soustavy.

# Reprezentace čísel s pohyblivou řádovou čárkou

- Reprezentace podle IEEE 754:




| význam | exponent | fraction |
|---|---|---|
| +/- nula | 0 | 0 |
| denormalizovaná čísla | 0 | nenulový |
| normalizovaná čísla | 1 až $2^e - 2$ | cokoliv |
| +/- ∞ | $2^e - 1$ | 0 |
| NaN (Not a Number) | $2^e - 1$ | nenulový |

- Normalizovaná hodnota:
  - Hodnota = $(-1)^{sign} \times 2^{exponent-exponent\ bias} \times (1.fraction)$
- Denormalizovaná hodnota:
  - Hodnota = $(-1)^{sign} \times 2^{exponent-exponent\ bias+1} \times (0.fraction)$

# Reprezentace čísel s pohyblivou řádovou čárkou

- Reprezentace podle IEEE 754:
  - **NaN** (Not a Number) se používá k reprezentaci čísel, která byla výsledkem aritmetických operací s nestandardními vstupy:
    - všechny aritmetické operace s NaN jako s alespoň jedním operandem
    - výsledek dělení: $0/0$, $\infty/\infty$, $\infty/-\infty$, $-\infty/\infty$ a $-\infty/-\infty$
    - výsledek násobení: $0\times\infty$ a $0\times-\infty$
    - výsledek sčítání: $\infty + (-\infty)$, $(-\infty) + \infty$ a k němu odpovídající odčítání.
    - aplikace argumentu funkce mimo svou doménu:
      - druhá odmocnina záporného čísla
      - logaritmus záporného čísla
      - trigonometrické funkce …
  - NaN se ještě rozlišuje na **Quiet** (většinou nevyvolávají výjimky) a **Signalling** (většinou vyvolávají výjimky (přetečení, podtečení)).

# Rozdíly počítačové a standardní aritmetiky

- Platí:
  - $1 \cdot x = x$
  - $x \cdot y = y \cdot x$
  - $x + x = 2 \cdot x$
- Obecně nemusí platit
  - $x \cdot (1/x) = 1$
  - $(1 + x) - 1 = x$
  - $(x + y) + z = x + (y + z)$
- Častou chybou bývá přičítání např. jedničky ve floatu v cyklu s podmínkou ukončení na rovnost libovolného čísla nebo nerovnost u velkých čísel. Cyklus se pak typicky nezastaví.