Pravdepodobnostné náhodné štruktúry: skiplist a treap

Prednáša Broňa Brejová

Obrázky a kód Pat Morin http://opendatastructures.org/
kde nájdete aj ďalšie detaily implementácie a dôkazov

Dátová štruktúra asociatívne pole, slovník

Udržujeme **množinu kľúčov**, každý môže mať pripojené ďalšie dáta
Operácie find, add, remove

Slovník pomocou hašovacích tabuliek

- Jednoduché **deterministické** hašovanie
  (pevne zvolená hašovacia funkcia, vedierka):
  $O(n)$ v najhoršom prípade
  (všetky prvky sa hašujú na tú istú hodnotu)
- **Náhodná hašovacia funkcia** z vhodnej triedy
  $O(1)$ očakávaný čas
- **Perfektné hašovanie**
  find v $O(1)$ v najhoršom prípade,
  randomizácia ovplyvňuje iba čas add/remove

Viac o hašovaní na cvičeniach

Slovník pomocou binárnych vyhľadávacích stromov

- Bez vyvažovania $O(n)$ v najhoršom prípade
- S vyvažovaním (napr. AVL stromy) $O(\log n)$ v najhoršom prípade

Na rozdiel od hašovania, binárne stromy sa dajú **rozšíriť** na ďalšie operácie, napr.:

- Vypíš všetky prvky v utriedenom poradí
- Ak kľúč x nie je v množine, nájdi najbližší, ktorý je
- Nájdi k-ty najmenší prvok množiny

Pravdepodobnostné alternatívy binárnych vyhľadávacích stromov

Deterministické:

- Bez vyvažovania $O(n)$ v najhoršom prípade
- S vyvažovaním $O(\log n)$ v najhoršom prípade (napr. AVL stromy)

Dnes uvidíme:

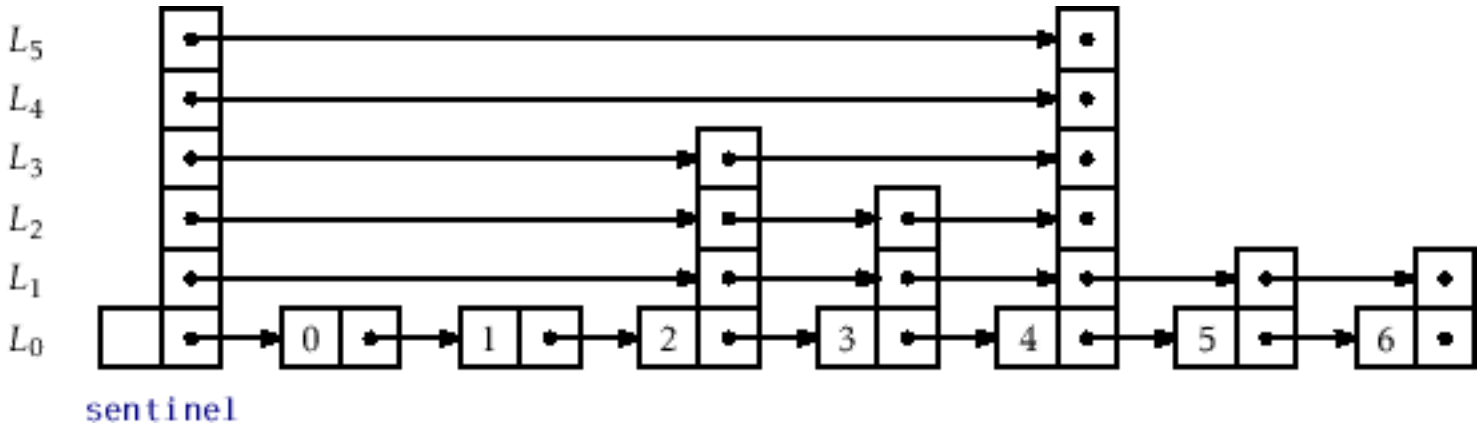
- **Skiplist** [Pugh. 1990]
- **Treap** [R. Seidel and C. Aragon. 1989; Vuillemin 1980]

Obidve sú pravdepodobnostné:

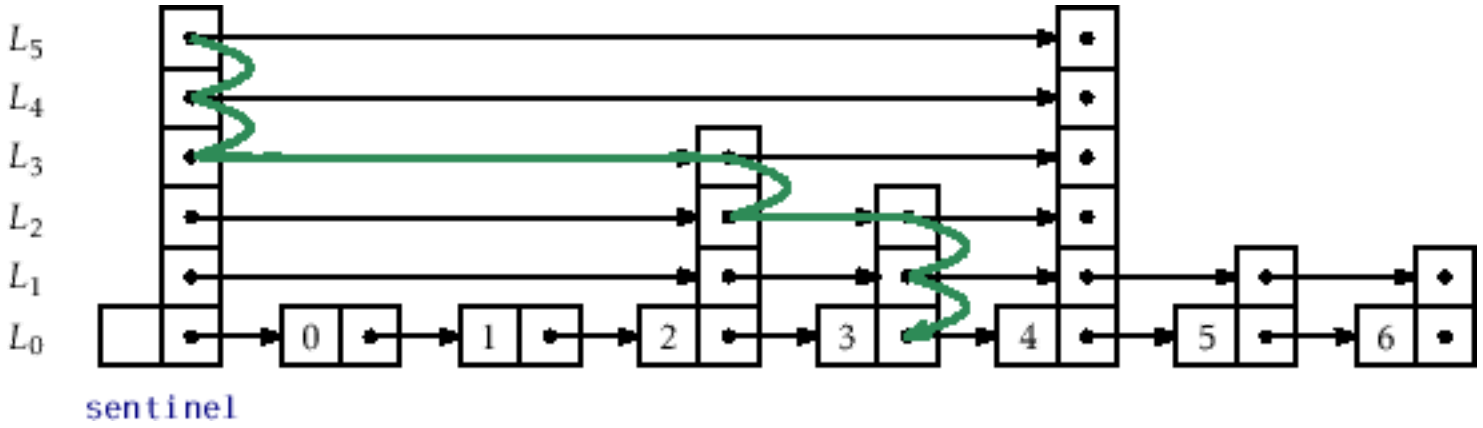
- Očakávaný čas $O(\log n)$
- Jednoduchšia implementácia ako vyvažovanie

Skiplist

- **Utriedené pole:**
  find $O(\log n)$, ale update znamená posúvanie, $O(n)$
- **Utriedený spájany zoznam:**
  find $O(n)$, ale ak vieme, kde chceme robit update, čas $O(1)$
- **Skiplist:**
  utriedený spájaný zoznam so skratkami
  rýchlejšie nájdeme správne miesto



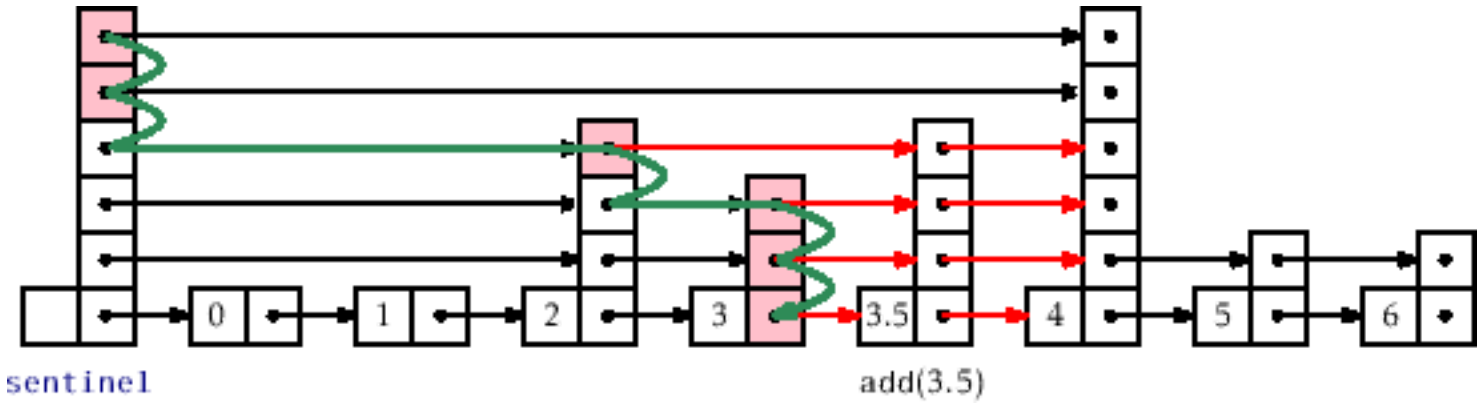
Skiplist: find, vyhľadanie predchodcu




```
Node findPredNode(T x) {
    Node u = sentinel; int r = h;
    while (r >= 0) {
        while (u.next[r] != null && u.next[r].x < x)
            u = u.next[r];    // go right in list r
        r--;                  // go down into list r-1
    }
    return u;
}
```

Skiplist: add(x)

- Nájdeme predchodcu prvku x
- Pamätáme si posledný navštívený uzol na každej úrovni
- Smerníky z týchto uzlov bude možno treba zmeniť
- Novému uzlu nastavíme náhodnú výšku
  koľko hodov mincou treba, kým padne hlava
  (geometrické rozdelenie)
- Operácia remove funguje podobne



Skiplist: analýza

Lema 1: Nech T je počet hodov mincou kým padne hlava. Potom $E[T] = 2$.

Dôkaz: Nech t_i je 1 ak $T \geq i$, 0 inak.
$T = \sum_{i=1}^{\infty} t_i$
$E[t_i] = 0 \Pr(t_i = 0) + 1 \Pr(t_i = 1) = \Pr(t_i = 1) = 1/2^{i-1}$

$$\begin{aligned} E[T] &= E\left[\sum_{i=1}^{\infty} t_i\right] = \sum_{i=1}^{\infty} E[t_i] \\ &= \sum_{i=1}^{\infty} 1/2^{i-1} = 1 + 1/2 + 1/4 + 1/8 + \ldots = 2. \end{aligned}$$

Dôsledok: Každý uzol skiplistu je teda v priemere v dvoch úrovniach.

Skiplist: analýza

Lema 2: Nech H je počet úrovní skiplistu s n prvkami.
$E[H] \leq \log_2 n + 3$

Dôkaz: Nech h_i je 1 ak je zoznam L_i neprázdny, 0 inak.

$E[h_i] \leq E[|L_i|] = n/2^i$

$$
\begin{aligned}
E[H] &= E\left[\sum_{i=0}^{\infty} h_i\right] = \sum_{i=1}^{\infty} E[h_i] \\
&= \sum_{i=0}^{\lfloor \log_2 n \rfloor} E[h_i] + \sum_{i=\lfloor \log_2 n \rfloor + 1}^{\infty} E[h_i] \\
&\leq \sum_{i=0}^{\lfloor \log_2 n \rfloor} 1 + \sum_{i=\lfloor \log_2 n \rfloor + 1}^{\infty} n/2^i \\
&\leq (\log_2 n + 1) + (1 + 1/2 + 1/4 + \ldots)
\end{aligned}
$$

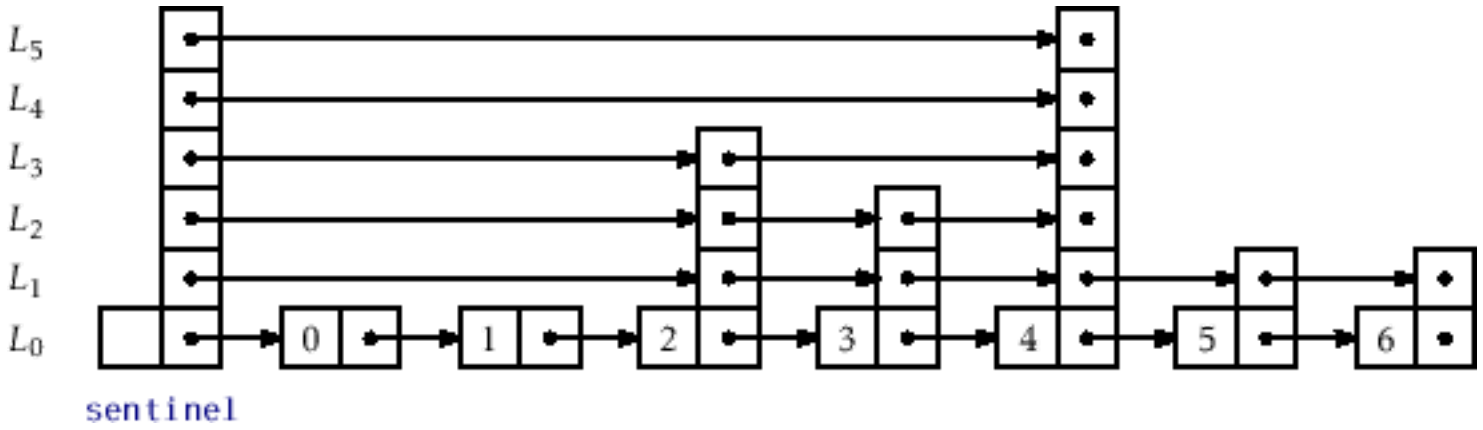
Skiplist: pamäťová zložitosť v očakávanom prípade

Zarážka (sentinel): $O(\log n)$ smerníkov

Ostatné smerníky v uzloch: $2n$

Ostatné dáta: $O(n)$

Spolu: $O(n)$

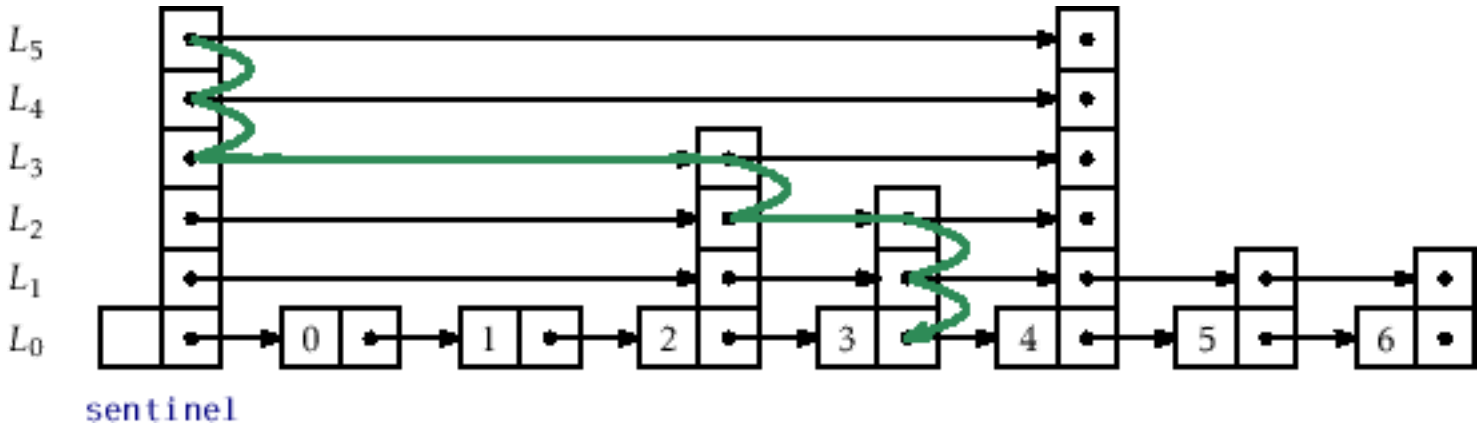


Skiplist: časová zložitosť v očakávanom prípade

Lema 3: Nech F je počet smerníkov prejdených vo findPredNode(x).

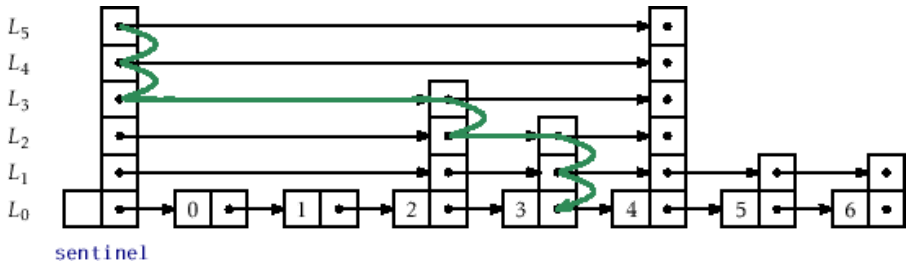
$$E[F] \leq 2 \log_2 n + O(1).$$

Dôsledok: V skipliste s n prvkami pracujú operácie find, add a remove v očakávanom čase $O(\log n)$.



Skiplist: dôkaz lemy 3

- Cestu k prvku x budujme odzadu: začneme v x na úrovni 0, keď sa dá ísť vyššie, ideme vyššie, inak doľava.
- Nech f_i je počet uzlov, ktoré cesta navštívi na úrovni i.
- V každom uzle sa dá ísť dohora s pravdepodobnosťou 1/2. Obdoba hádzania kockou. $E[f_i] \leq 2$.
- Taktiež $E[f_i] \leq 2E[|L_i|] = 2n/2^i$
- $F = \sum_{i=0}^{\infty} f_i = \sum_{i=0}^{\lfloor \log_2 n \rfloor} f_i + \sum_{i=\lfloor \log_2 n \rfloor + 1}^{\infty} f_i$
- $E[F] \leq (2 \log_2 n + 2) + 2(1 + 1/2 + 1/4 + \ldots) = 2 \log_2 n + O(1)$

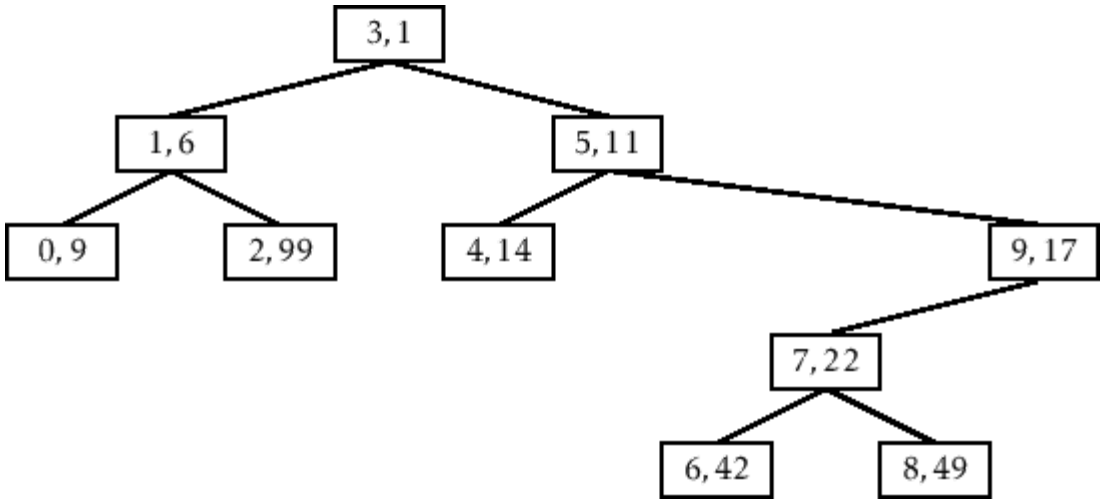


Treap (tree+heap)

Binárny vyhľadávací strom:
Ak má vrchol kľúč x, všetky vrcholy v ľavom podstrome majú kľúče $< x$ a všetky vrcholy v pravom podstrome majú kľúče $> x$.

Halda: Ak má vrchol kľúč x, všetky jeho deti majú kľúče $> x$.

Treap: Každý vrchol má kľúč k a prioritu p.
Kľúče tvoria binárny vyhľadávací strom, priority tvoria binárnu haldu.



Treap (tree+heap)

Každý vrchol má kľúč k a prioritu p.
Kľúče tvoria binárny vyhľadávací strom, priority tvoria binárnu haldu.

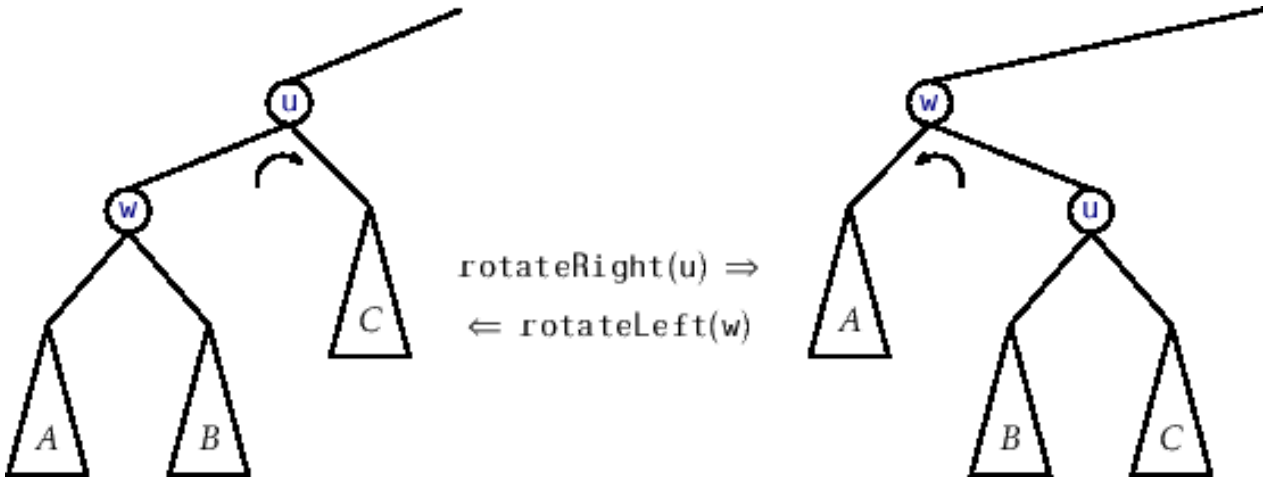
Pre každú množinu dvojíc kľúč a priorita (bez opakujúcich sa hodnôt) existuje **práve jeden** treap.

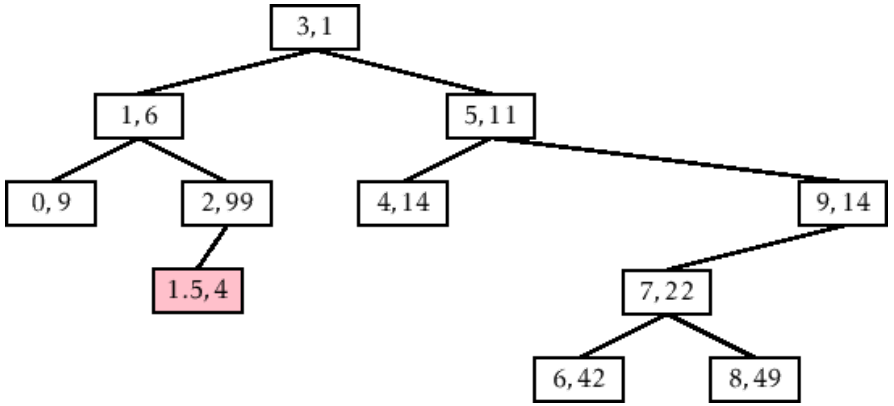
Dôkaz:
Koreň musí byť prvok s minimálnou prioritou
Ostatné prvky rozdelíme vľavo a vpravo podľa kľúča
Ľavý a pravý strom vybudujeme rovnakým spôsobom

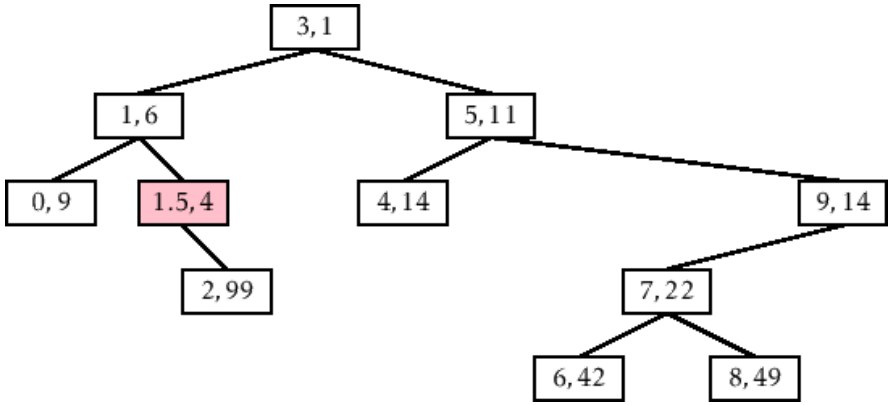
V treape sa priority kľúčom generujú **náhodne** pri vkladaní prvku.

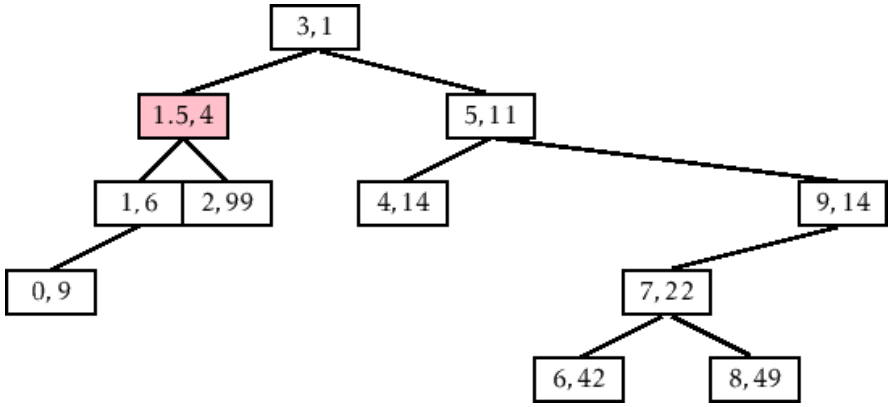
Algoritmy pre treap

- Find funguje ako v binárnom vyhľadávacom strome
- Pri add najskôr prvok vložíme do listu podľa kľúča, potom robíme rotácie, kým je porušená vlastnoť haldy
- Pri remove prvok rotujeme, kým sa nedostane do listu, potom ho zmažeme.




3, 1
1, 6
5, 11
0, 9
2, 99
4, 14
9, 14
1.5, 4
7, 22
6, 42
8, 49


3, 1
1, 6
5, 11
0, 9
1.5, 4
4, 14
9, 14
2, 99
7, 22
6, 42
8, 49


3, 1
1.5, 4
5, 11
1, 6
2, 99
4, 14
9, 14
0, 9
7, 22
6, 42
8, 49

Analýza treap-u

- Zložitosť operácií je úmerná **hĺbke stromu**
- Hĺbka stromu je v najhoršom prípade $O(n)$ a v najlepšom $O(\log n)$.
- Tvar treapu je ekvivalentný binárnemu vyhľadávaciemu stromu, do ktorého boli kľúče pridávané v poradí od najmenšej priority, teda **v náhodnom poradí**
- Očakávaná výška takého stromu je $O(\log n)$

Očakávaná výška binárneho vyhľadávacieho stromu s náhodným poradím vkladania

- Majme v strome prvky $x_1 < x_2 < \ldots < x_n$, hľadáme x_k.
- Ak find prejde cez uzol x_i, kde $i < k$,
  doteraz sme na ceste videli buď prvky väčšie ako x_k
  alebo menšie ako x_i
- Prvky $x_{i+1}, \ldots, x_k$ by sa až po x_i hľadali tak isto,
  boli teda vložené neskôr ako x_i
- Nech $h_i = 1$ ak prvok x_i je na ceste ku x_k, 0 inak.
  $E[h_i] = \Pr(h_i = 1) = 1/(k - i + 1)$
- Podobne pre väčšie prvky
- $E[H] = \sum_{i=1}^{n} E[h_i] \leq 2(1 + 1/2 + 1/3 + \ldots 1/n) = O(\log n)$
- Harmonické číslo $H_n = 1 + 1/2 + 1/3 + \ldots 1/n$
  $\ln n < H_n \leq \ln n + 1$

Pravdepodobnostné alternatívy binárnych vyhľadávacích stromov

Deterministické:

- Bez vyvažovania $O(n)$ v najhoršom prípade
- S vyvažovaním $O(\log n)$ v najhoršom prípade (napr. AVL stromy)

Dnes sme videli:

- **Skiplist** [Pugh. 1990]
- **Treap** [R. Seidel and C. Aragon. 1989; Vuillemin 1980]

Obidve sú pravdepodobnostné:

- Očakávaný čas $O(\log n)$
- Jednoduchšia implementácia ako vyvažovanie