13. Metody vyhledávání. Adresní vyhledávání (přímý přístup, zřetězené a otevřené rozptylování, rozptylovací funkce). Asociativní vyhledávání (sekvenční, binárním půlením, interpolační, binární vyhledávací stromy). Operační a paměťová složitost algoritmů vyhledávání. (A7B36ALG, A7B36DSA)

Adresní vyhledávání (přímý přístup, zřetězené a otevřené rozptylování, rozptylovací funkce)

porovnáváním klíčů (nepatří mezi adresní vyhledávání, jen pro porovnání)

- Nalezeno, když klíč_prvku = hledaný klíč (složitost omega(log n))
- např. sekvenční vyhledávání, BVS,...

Adresní vyhledávaní - z klíče se vypočte adresa umístění prvku. Využívá se jednoznačného vztahu mezi hodnotou klíče prvku a umístěním prvku ve struktuře představující vyhledávací prostor

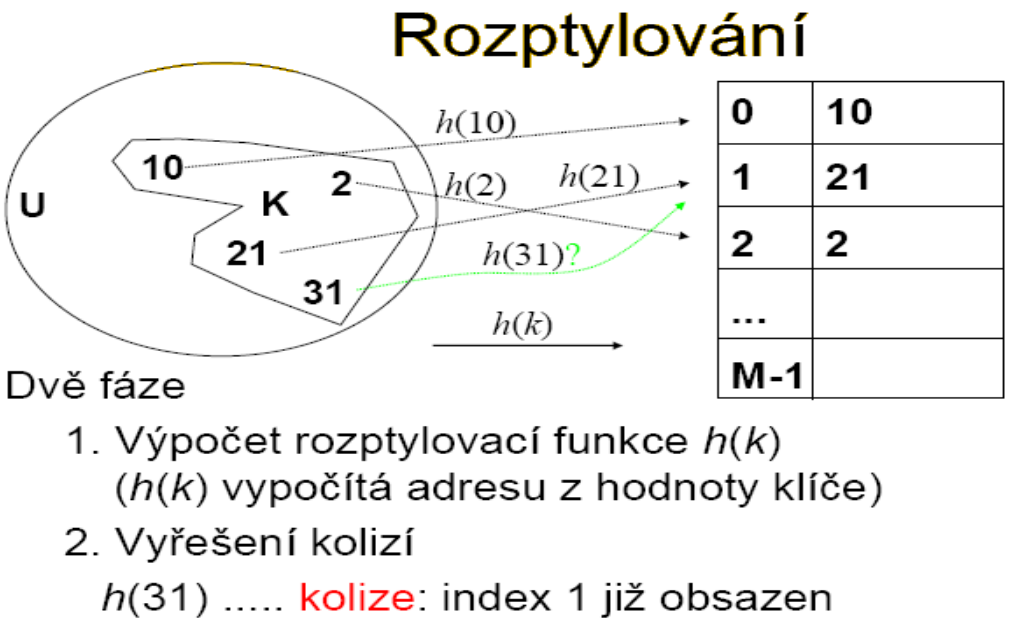
- **Přímý přístup** – klíč je přímo adresou(indexem) hledaného prvku, rozsah klíčů ~ rozahu indexů (složitost omega(1)).
- **Rozptylování** – výpočtem adresy z hodnoty klíče (složitost průměrně omega(1))

Rozptylování – Hešování = kompromis mezi rychlostí a spotřebou paměti

Něco za něco:

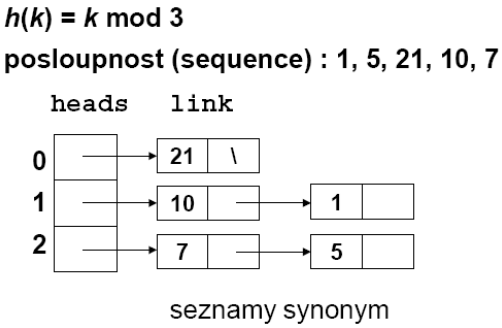
– čas provádění ~ délce klíče

– není vhodné pro operace výběru podmnožiny a řazení (select a sort)



Pro výpočet adresy se používá rozptylovací funkce ve tvaru h(k)=a, kde $k \in K$ je množina identifikačních klíčů a $a \in A=[a_{min},\ldots,a_{max}]$ je souvislý interval adres. Vzhledem k tomu, že $|K|>>|A|$ může nastat, že pro dva různé prvky jejichž klíče se nerovnají $k_1 \neq k_2$ (říká se jim synonyma) platí $h(k_1)=h(k_2)$. Takovému stavu se říká kolize. Kolize se řeší zřetězeným rozptylováním nebo otevřeným rozptylováním.

Zřetězené rozptylování (chaining) – jedná se o zřetězení synonym v dynamické paměti



Řetěz synonym má ideálně délku $\alpha = n/m$, $\alpha > 1$ (plnění tabulky)
(n = počet prvků, m = velikost tabulky, m<n)

velmi nepravděpodobný extrém

Insert	$I(n) = t_{hash} + t_{link} = O(1)$	
Search	$Q(n) = t_{hash} + t_{search}$	průměrně
	$= t_{hash} + t_c^* \, n/(2m) = O(n)$	$O(1 + \alpha)$
Delete	$D(n) = t_{hash} + t_{search} + t_{link} = O(n)$	$O(1 + \alpha)$

pro malá α (velká m) se hodně blíží O(1) !!!
pro velká α (malá m) m-násobné zrychlení x seq. s.

Shrnutí:

\+ nemusíme znát n předem

– potřebuje dynamické přidělování paměti

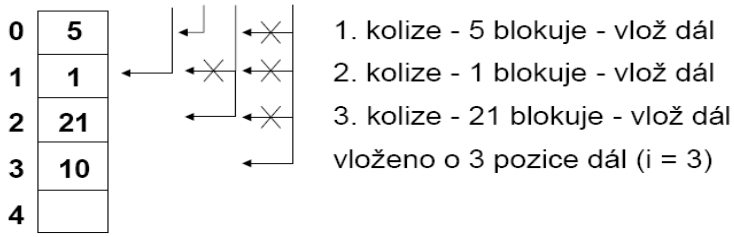
– potřebuje paměť na ukazatele a na tabulku[m]

Otevřené rozptylování

Znám předem počet prvků (odhad) nechci ukazatele (v prvcích ani tabulku) => posloupnost do pole

- Linear probing - sekvenční ukládání do pole

***h*(*k*) = (k + i) mod 5**
posloupnost: 1, 5, 21, 10, 7

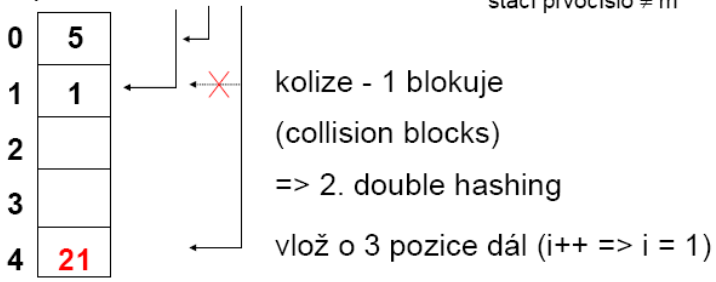


Double Hashing: h(k) = (k + i.3) mod x

$h(k) = [(k \bmod 5) + i.h_2(k)] \bmod 5$, $h(k) = (k + i.3) \bmod 5$

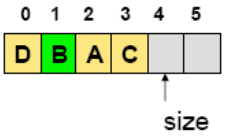
posloupnost: 1, 5, 21, 10, 7

stačí prvočíslo ≠ m



Asociativní vyhledávání (sekvenční, půlením, binární vyhledávací stromy)

Sekvenční – polem se postupně prochází, dokud nenajdeme požadovaný prvek



Unsorted array
Sequential search
insert
delete
min, max

P(n) = O(n)
Q(n) = O(n) ☹
I(n) = O(1) ☺
D(n) = O(n) ☹
in O(n) ☹

```
nodeT seqSearch( key k, nodeT a[] ) {
      int i = 0;
      while( (i < a.size) && (a[i].key != k) )
            i++;
      if( i < a.size ) return a[i];
      else return NODE_NOT_FOUND;
}
```

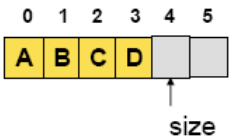
Java-like pseudo code

Hledání s označením (Sentinel):

search(“E“, a);

Sekvenční vyhledávání mírně rychlejší, ale stále Q(n) = O(n) ☹

Půlením – před vlastním hledáním se pole vzestupně setřídí. Poté se najde prostřední prvek a porovná se s hledaným. Pokud je větší, tak se hledaný prvek nachází ve spodní polovině. Pokud je menší, tak se hledaný nachází v horní polovině. V polovičním půlení se pokračuje až do pole délky dva. Něco jako hádání čísel od 1 do 100 s nápovědou větší menší. Složitosti se počítají bez setřídění pole:



Sorted array
Binary search
insert
delete
min, max

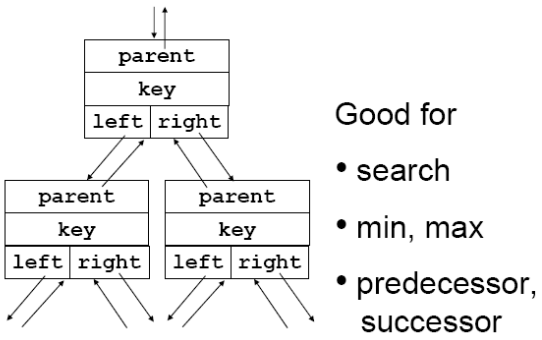
P(n) = O(n)
Q(n) = O(log(n)) ☺
I (n) = O(n) ☹
D(n) = O(n) ☹
in O(1) ☺

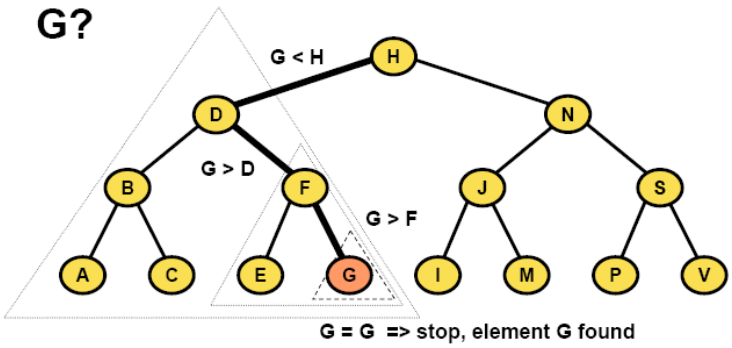
```
nodeT binarySearch( key k, nodeT sortedArray[] ) {
  int pos = bs( k, sortedArray, 0, sortedArray.size - 1 );

  if( pos >= 0 ) return sortedArray[pos];
  else           return NODE_NOT_FOUND;
                 // bs can return  -(pos+1), i.e.
                 // position to insert the node with key k
}
```

Java-like pseudo code

Binární vyhledávací stromy – používá se v úlohách, kde máme soubor stavů s definovanými přechody mezi nimi. Binární strom je takový strom, kde každý uzel má 0 nebo 2 následníky. Strom je uspořádán tak, že levý následník je menší než předchůdce a pravý následník je větší než předchůdce.

- pro všechny uzly u_L z levého podstromu platí: klíč(u_L) < klíč(u)
- pro všechny uzly u_R z pravého podstromu platí: klíč(u_R) > klíč(u)
- u je kořen

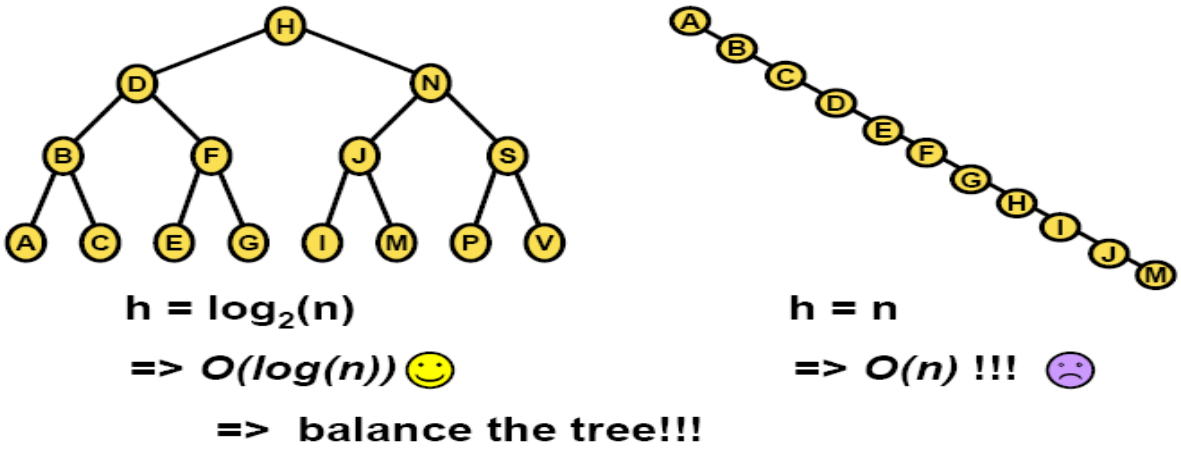




Minimum v BST je úplně levý prvek a maximum je pravý prvek

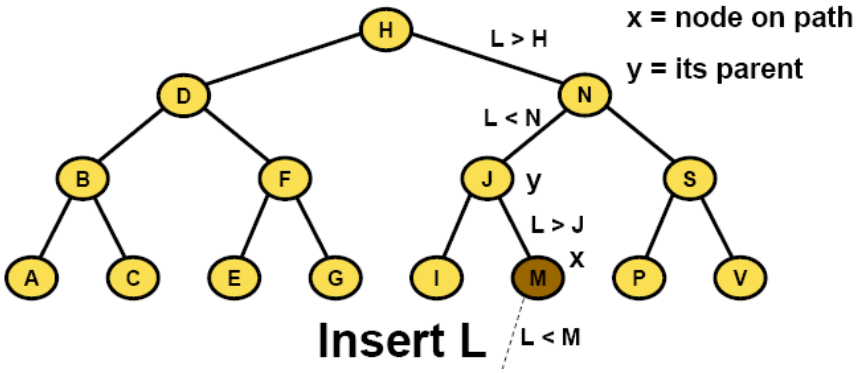
Všechny operace:

Search, Maximum, Minimum, Successor, Predecessor můžou běžet v čase *O(h)*, kde h je hloubka stromu



Insert (vložení prvku)

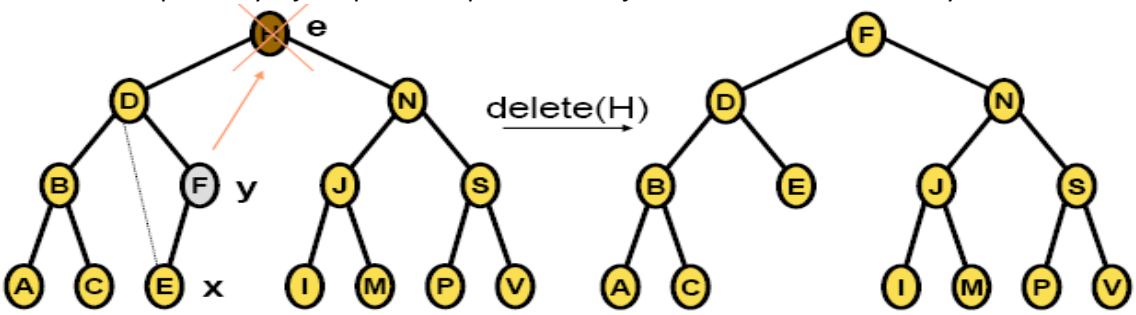
1. Najdi rodičovský uzel ve vyváženém stromu O(log(n)), nebo O(h) v normalním
2. Připoj nový element jako jeho uzel O(1)



Delete (smazaní prvku)

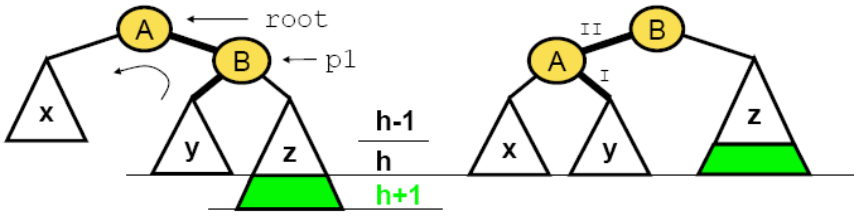
3 typy mazání:

- Uzel nemá potomky – prostě ho smažeme
- Uzel má jednoho potomka – přemostíme vymazaný uzel tak, že místo něj se jenom spojí jeho rodič a následník
- Uzel má 2 potomky – jeho potomka potomek na jeho xové úrovni se s ním vymění ..

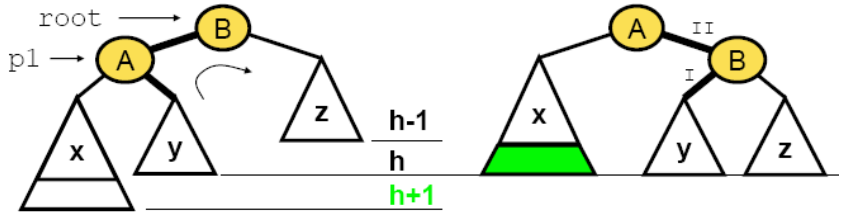


Vyvažování stromu:

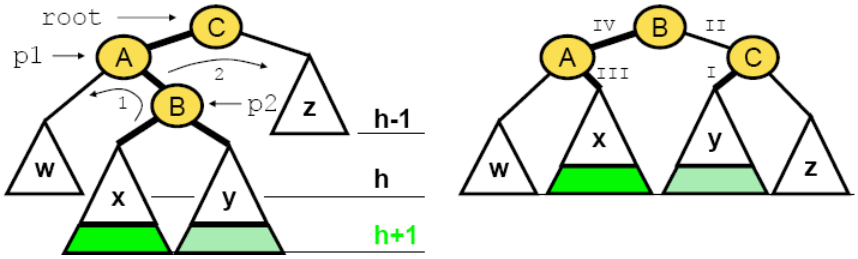
L rotace (Left Rotation)



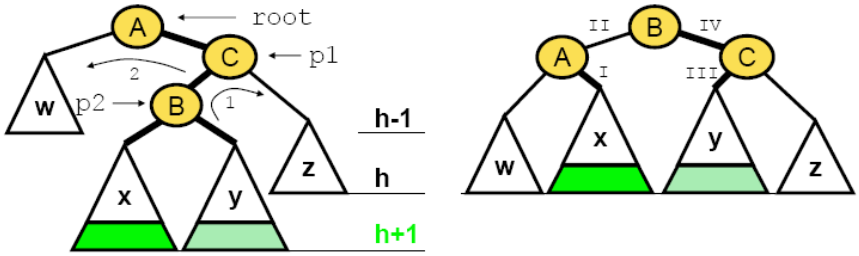
R rotace (Right Rotation)



LR station



RL rotatin



Operační a paměťová složitost algoritmů vyhledávání v průměrném, nejhorším a nejlepším případě

Vyhledávání	Typ	Paměťová složitost P(n)	Vyhledávání Q(n)	Vládání I(n)	Mazání D(n)	Nalezení minima, maxima	Nalezení předchůdce	Nalezení následníka
Sekvenční	Asoc	O(n)	O(n)	O(1)	O(n)	O(n)	N/A	N/A
Půlením (binární)	Asoc	O(n)	O(log n)	O(n)	O(n)	O(1)	N/A	N/A
Binární vyhledávací stromy	Asoc	O(n)	O(h)	O(h)	O(h)	O(h)	O(h)	O(h)
Přímý přístup	Adr.	Θ(1)	Θ(1)	Θ(1)	Θ(1)	Θ(1)	N/A	N/A
Zřetězené rozptylování	Adr.	O(m + n)	Extrém O(n) Průměrně O(1 + α)	O(1)	Extrém O(n) Průměrně O(1 + α)	Extrém O(n) Průměrně O(1 + α)	N/A	N/A
Otevřené rozptylování	Adr.	O(n)	úměrné α	úměrné α	úměrné α	úměrné α	N/A	N/A

ZDROJE :

fel.jahho.cz/statnice/SW/soft%20-%2004.doc