Algoritmy a datové struktury

Skiplisty a hashovací tabulky

Obsah přednášky

- SkipList
- Hašovací tabulka

Datové struktury

- Spojové seznamy
  - jednoduchá implementace
  - vysoká složitost vkládání a hledání

Datové struktury

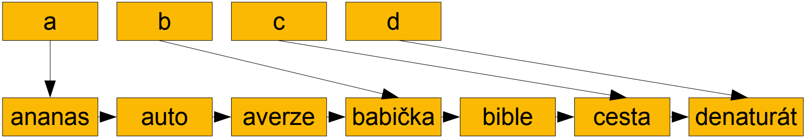
- Spojové seznamy
  - jednoduchá implementace
  - vysoká složitost vkládání a hledání O(n)
- Vyvažované stromy
  - nízká složitost vkládání a hledání

Datové struktury

- Spojové seznamy
  - jednoduchá implementace
  - vysoká složitost vkládání a hledání O(n)
- Vyvažované stromy
  - nízká složitost vkládání a hledání O(log n)
  - složitá implementace (operace vyvážení)
  - navíc – problém s iterátorem
    - projít všechny položky vyžaduje rekurzivní průchod celým stromem
- Zkusit získat výhody obojího…

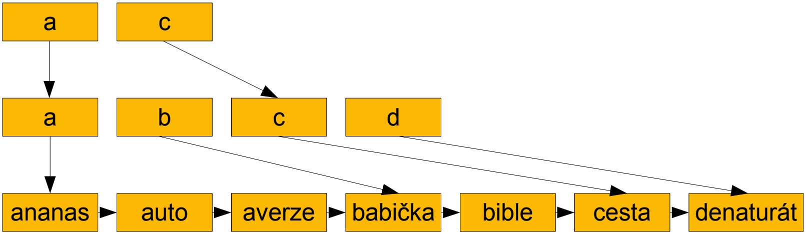
Rozšíření seznamů

- Rozšířit spojový seznam o seznam významných bodů
  - např. slovník



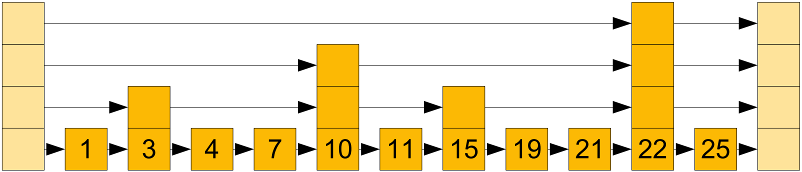
Rozšíření seznamů

- Rozšířit spojový seznam o seznam významných bodů
  - např. slovník
  - možnost rozšířit o další vrstvu...



SkipList

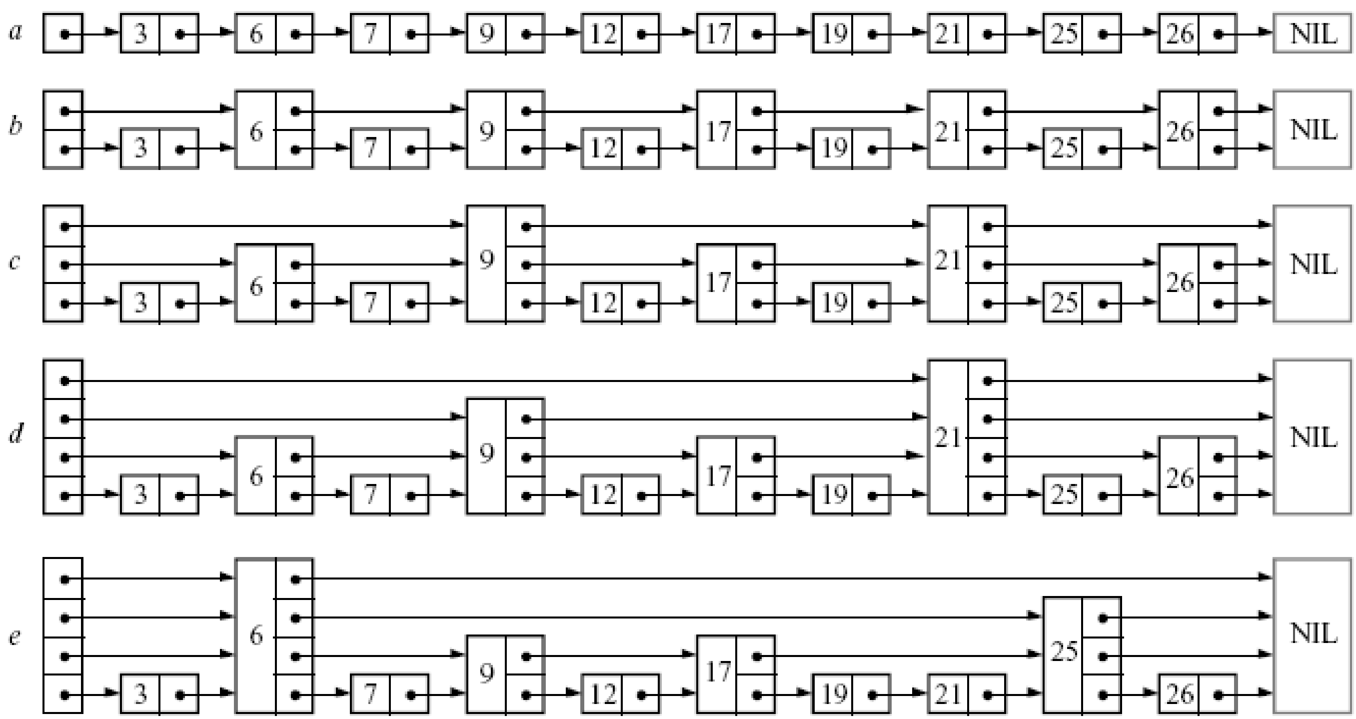
- Přeskakovací seznam
- Relativně mladá struktura (1990)
- N-vrstvá struktura
  - založená na pravděpodobnosti
- Podobně jako spojový seznam
  - hlavička (ocásek)
  - vícerozměrný



Skip-List

- je datová struktura, která může být použita jako náhrada za vyvážené stromy.
- představují pravděpodobnostní alternativu k vyváženým stromům (struktura jednotlivých uzlů se volí náhodně)
- Na rozdíl od stromů má **skip list** následující výhody:
  - jednoduchá implementace
  - jednoduché algoritmy vložení/zrušení
  - časová složitost vyhledávání je obdobná jako u stromů

Základní myšlenka zavedení skip-listů

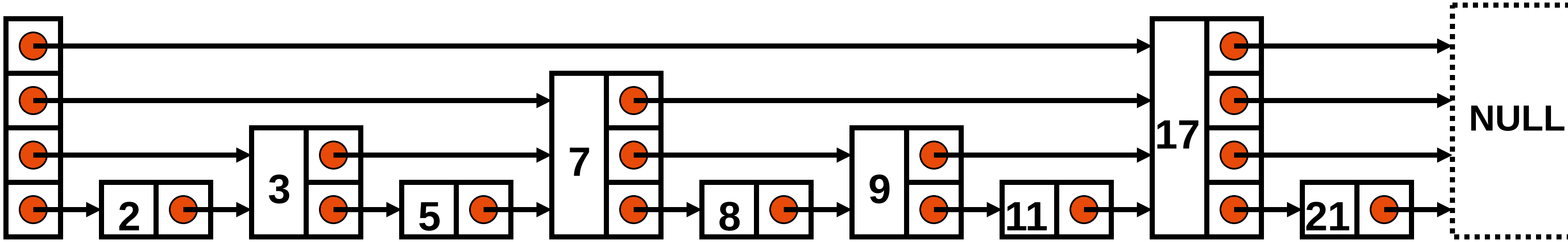


seznam	složitost vyhledávání – nejhorší případ
a) obyčejný spoj.seznam	n
b) extra ukazatele mezi každým 2. uzlem	$\lceil n/2 \rceil + 1$
c) extra ukazatele mezi každým 4. uzlem	$\lceil n/4 \rceil + 1$
d) extra ukazatele mezi každým 2^i. uzlem	$\lceil \log n \rceil$
e) náhodná volba extra uzlů s ukazateli (skip list)	???

Skip-List

- prky v seznamu jsou uspořádány
- seznam obsahuje prvky, které mají k ukazatelů

  $1 \leq k \leq$ max_level
- uzel s *k*-ukazateli se nazývá uzel úrovně k
- seznam úrovně k – obsahuje prvky s maximálně k ukazateli
- ideální skip-list – každý 2^i-tý prvek má ukazatel, který ukazuje o 2^i prvků dopředu



Pokud má každý 2^itý uzel 2^i ukazatelů na následující uzly, pak jsou uzly jednotlivých úrovní rozloženy následovně:

50% uzlů úrovně 1
25% uzlů úrovně 2
12.5% uzlů úrovně 3
atd.

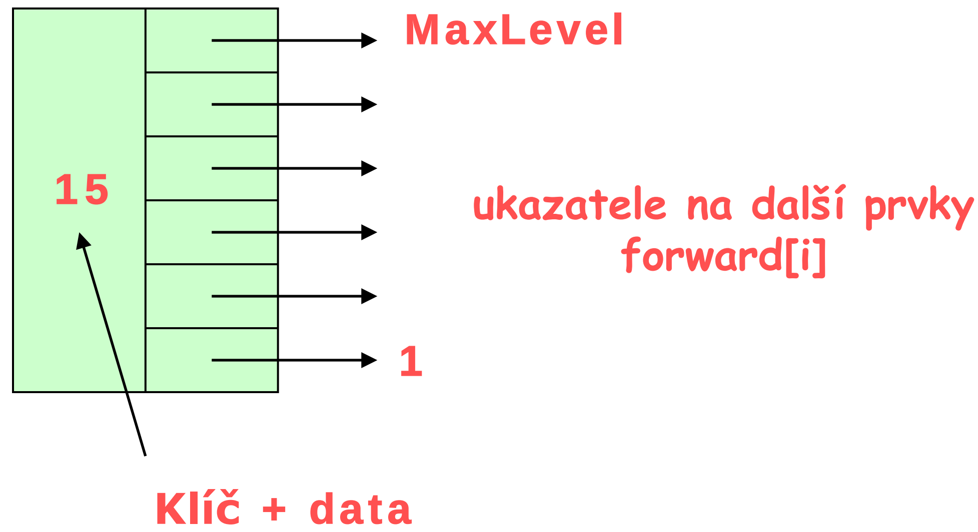
Výhoda: složitost vyhledávání O(log n)

Nevýhoda: po provedení operací insert/delete je nutné provádět restrukturalizaci seznamu

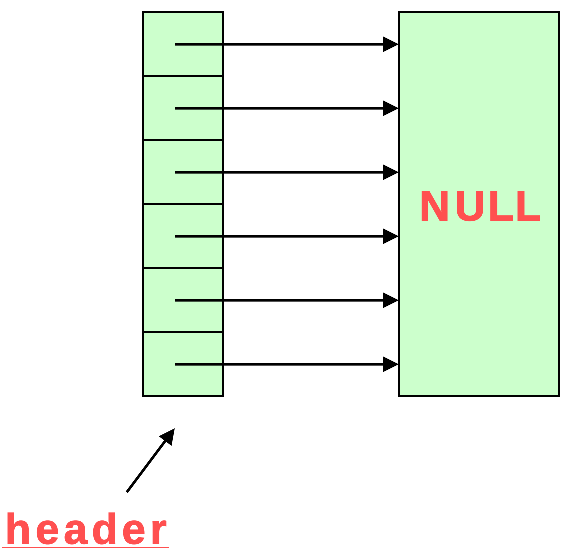
Řešení: ponechat rozložení uzlů ale vyhnout se restrukturalizaci – tj.
uzly úrovně k jsou vkládány náhodně s uvedeným pravděpodobnostním rozložením

Prvek Skip-listu

• každý prvek seznamu úrovně k má k ukazatelů (k se volí náhodně při vytvoření prvku)



Prázdný seznam



Inicializace seznamu

• je vytvořena hlavička seznamu (obsahuje MaxLevel ukazatelů)

• všechny ukazatele se inicializují na NIL

• celkový počet úrovní MaxLevel se volí na základě maximálního počtu prvků N

MaxLevel=$\log_2(N)$

Proč pravděpodobnost?

- Proč použít náhodná čísla?
  - proč nedát každý druhý prvek do 2. vrstvy, čtvrtý do 3. vrstvy. . . ?

Proč pravděpodobnost?

- Proč použít náhodná čísla?
  - proč nedát každý druhý prvek do 2. vrstvy, čtvrtý do 3. vrstvy. . . ?
  - dynamická struktura, náhodná čísla zajistí „rozumné“ rozložení
- Prvek z vrstvy I se objeví s pravděpodobností p v I+1 vrstvě
- Prvek se objeví v průměru v $1/(1-p)$ vrstvách
  - $1 + p + p^2 + \dots$
- Volba výšky hlavičky
  - $\log_{(1/p)}(n)$
- Možnost neomezit výšku
  - příliš složité

Porovnání s ostatními datovými strukturami

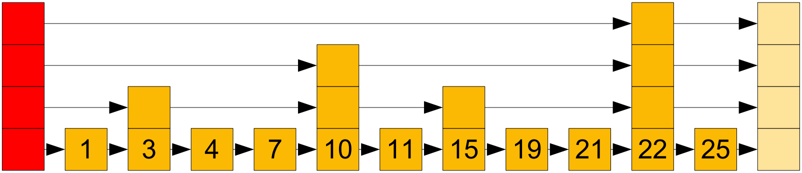
Implementation	Search Time		Insertion Time		Deletion Time	
Skip lists	0.051 msec	(1.0)	0.065 msec	(1.0)	0.059 msec	(1.0)
non-recursive AVL trees	0.046 msec	(0.91)	0.10 msec	(1.55)	0.085 msec	(1.46)
recursive 2–3 trees	0.054 msec	(1.05)	0.21 msec	(3.2)	0.21 msec	(3.65)
Self–adjusting trees:						
top-down splaying	0.15 msec	(3.0)	0.16 msec	(2.5)	0.18 msec	(3.1)
bottom-up splaying	0.49 msec	(9.6)	0.51 msec	(7.8)	0.53 msec	(9.0)

Table 2 - Timings of implementations of different algorithms

p	Normalized search times (*i.e.*, normalized $L(n)/p$)	Avg. # of pointers per node (*i.e.*, $1/(1-p)$)
$1/2$	1	2
$1/e$	0.94...	1.58...
$1/4$	1	1.33...
$1/8$	1.33...	1.14...
$1/16$	2	1.07...

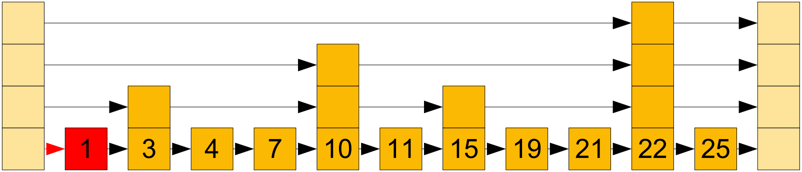
Operace hledání

- Hledání ve spojovém seznamu
  - lineární procházení spodní vrstvy
  - jednoduché, ale pomalé
  - složitost $O(n)$
- Např. číslo 19



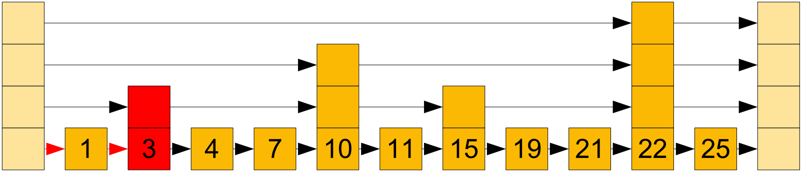
Operace hledání

- Hledání ve spojovém seznamu
  - lineární procházení spodní vrstvy
  - jednoduché, ale pomalé
  - složitost $O(n)$
- Např. číslo 19



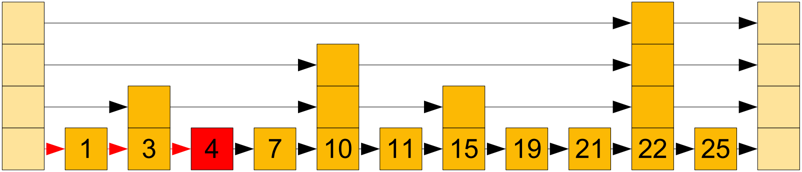
Operace hledání

- Hledání ve spojovém seznamu
  - lineární procházení spodní vrstvy
  - jednoduché, ale pomalé
  - složitost $O(n)$
- Např. číslo 19



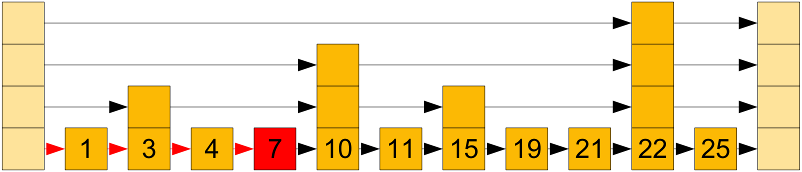
Operace hledání

- Hledání ve spojovém seznamu
  - lineární procházení spodní vrstvy
  - jednoduché, ale pomalé
  - složitost $O(n)$
- Např. číslo 19



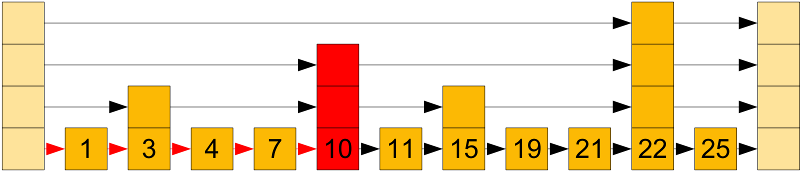
Operace hledání

- Hledání ve spojovém seznamu
  - lineární procházení spodní vrstvy
  - jednoduché, ale pomalé
  - složitost $O(n)$
- Např. číslo 19



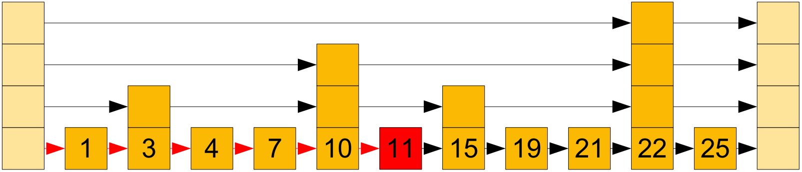
Operace hledání

- Hledání ve spojovém seznamu
  - lineární procházení spodní vrstvy
  - jednoduché, ale pomalé
  - složitost $O(n)$
- Např. číslo 19



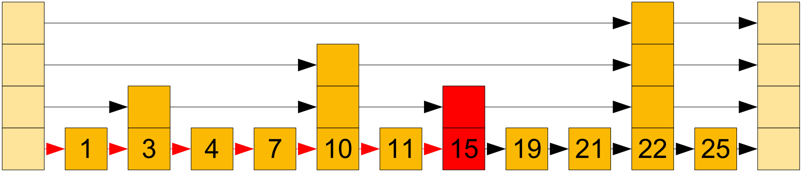
Operace hledání

- Hledání ve spojovém seznamu
  - lineární procházení spodní vrstvy
  - jednoduché, ale pomalé
  - složitost $O(n)$
- Např. číslo 19



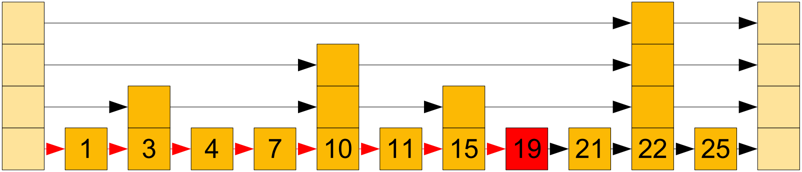
Operace hledání

- Hledání ve spojovém seznamu
  - lineární procházení spodní vrstvy
  - jednoduché, ale pomalé
  - složitost $O(n)$
- Např. číslo 19



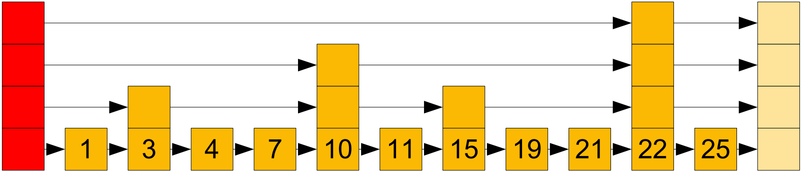
Operace hledání

- Hledání ve spojovém seznamu
  - lineární procházení spodní vrstvy
  - jednoduché, ale pomalé
  - složitost $O(n)$
- Např. číslo 19



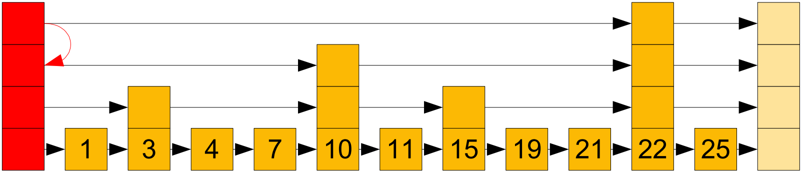
Operace hledání

- Víceúrovňové hledání
  - začneme v nejvyšším patře
  - lineárně prohledáváme
  - když narazíme na větší prvek, přesuneme se o patro níž
  - podobá se hledání půlením intervalu
    - v každé vrstvě omezený počet uzlů $O(1)$
    - celkem $\log n$ vrstev



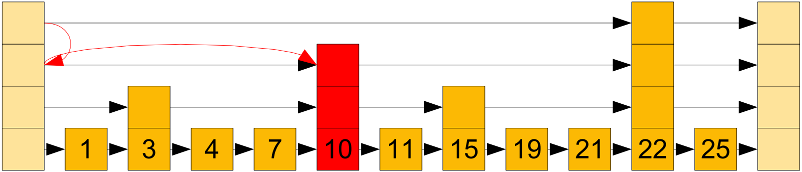
Operace hledání

- Víceúrovňové hledání
  - začneme v nejvyšším patře
  - lineárně prohledáváme
  - když narazíme na větší prvek, přesuneme se o patro níž
  - podobá se hledání půlením intervalu
    - v každé vrstvě omezený počet uzlů $O(1)$
    - celkem $\log n$ vrstev



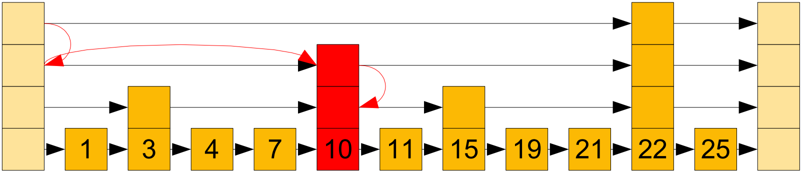
Operace hledání

- Víceúrovňové hledání
  - začneme v nejvyšším patře
  - lineárně prohledáváme
  - když narazíme na větší prvek, přesuneme se o patro níž
  - podobá se hledání půlením intervalu
    - v každé vrstvě omezený počet uzlů $O(1)$
    - celkem $\log n$ vrstev



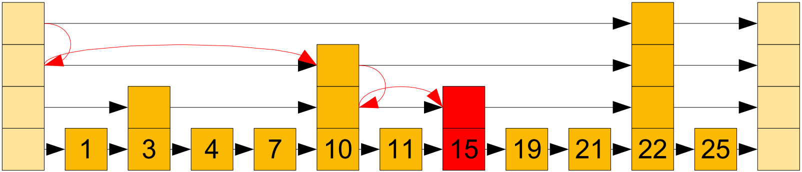
Operace hledání

- Víceúrovňové hledání
  - začneme v nejvyšším patře
  - lineárně prohledáváme
  - když narazíme na větší prvek, přesuneme se o patro níž
  - podobá se hledání půlením intervalu
    - v každé vrstvě omezený počet uzlů $O(1)$
    - celkem $\log n$ vrstev



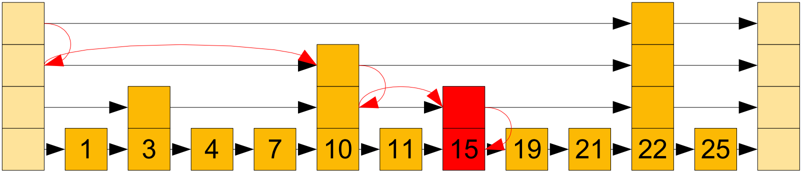
Operace hledání

- Víceúrovňové hledání
  - začneme v nejvyšším patře
  - lineárně prohledáváme
  - když narazíme na větší prvek, přesuneme se o patro níž
  - podobá se hledání půlením intervalu
    - v každé vrstvě omezený počet uzlů $O(1)$
    - celkem $\log n$ vrstev



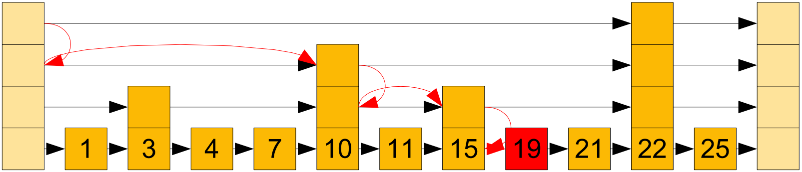
Operace hledání

- Víceúrovňové hledání
  - začneme v nejvyšším patře
  - lineárně prohledáváme
  - když narazíme na větší prvek, přesuneme se o patro níž
  - podobá se hledání půlením intervalu
    - v každé vrstvě omezený počet uzlů $O(1)$
    - celkem $\log n$ vrstev



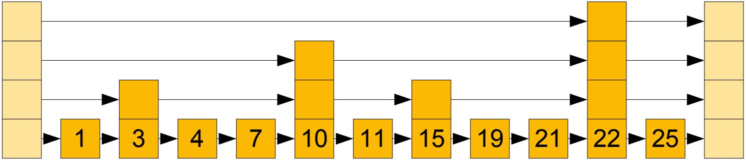
Operace hledání

- Víceúrovňové hledání
  - začneme v nejvyšším patře
  - lineárně prohledáváme
  - když narazíme na větší prvek, přesuneme se o patro níž
  - podobá se hledání půlením intervalu
    - v každé vrstvě omezený počet uzlů $O(1)$
    - celkem $\log n$ vrstev



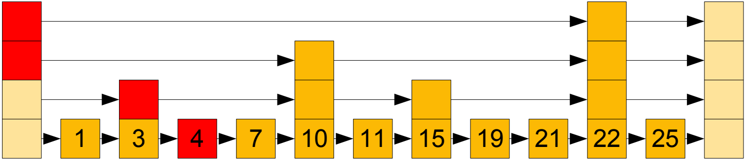
Operace vkládání

- Vygeneruje se výška vkládaného prvku
- Při hledání pozice prvku se zapamatují předchozí prvky v jednotlivých vrstvách
- Prvek se vloží do všech vrstev listu
  - podobně jako u spojového seznamu
- Např. číslo 5



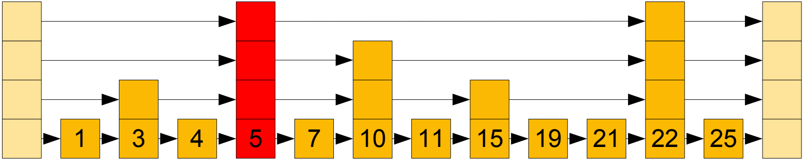
Operace vkládání

- Vygeneruje se výška vkládaného prvku
- Při hledání pozice prvku se zapamatují předchozí prvky v jednotlivých vrstvách
- Prvek se vloží do všech vrstev listu
  - podobně jako u spojového seznamu
- Např. číslo 5



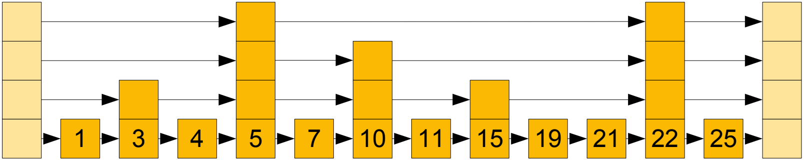
Operace vkládání

- Vygeneruje se výška vkládaného prvku
- Při hledání pozice prvku se zapamatují předchozí prvky v jednotlivých vrstvách
- Prvek se vloží do všech vrstev listu
  - podobně jako u spojového seznamu
- Např. číslo 5



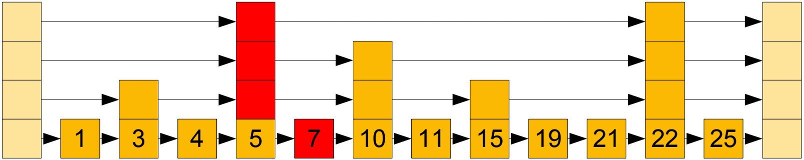
Operace mazání

- Přímočará operace
  - nalezení prvku a zapamatování si předchůdců
  - upravení ukazatelů na následníky



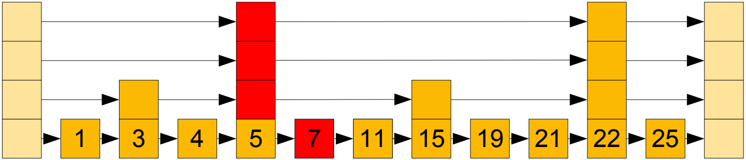
Operace mazání

- Přímočará operace
  - nalezení prvku a zapamatování si předchůdců
  - upravení ukazatelů na následníky



Operace mazání

- Přímočará operace
  - nalezení prvku a zapamatování si předchůdců
  - upravení ukazatelů na následníky



Vlastnosti

- Logaritmické složitosti pro všechny operace
- Rychlost srovnatelná s vyváženými stromy
  - paměťově náročnější
  - rychlejší pro některé operace
- Omezená hloubka stromu nemusí být na překážku
  - logatitmická závislost (pro hloubku 15 přes 65 tisíc položek)
  - lze upravit na neomezené
    - komplikované
- Možnost doladit rychlost/paměť
  - změna pravděpodobnosti
    - ideálně 1/e
    - programově snadné 1/2
    - menší čísla sníží paměťové nároky
    - větší než 1/2 nemá cenu

Opakování

- Pole
  - hledání

Opakování

- Pole
  - hledání $O(n), O(logn)$
  - vkládání

Opakování

- Pole
  - hledání $O(n), O(logn)$
  - vkládání $O(n)$
    - navíc problém s omezenou velikostí
  - mazání

Opakování

- Pole
  - hledání $O(n), O(logn)$
  - vkládání $O(n)$
    - navíc problém s omezenou velikostí
  - mazání $O(n)$

Opakování

- Pole
  - hledání $O(n), O(logn)$
  - vkládání $O(n)$
    - navíc problém s omezenou velikostí
  - mazání $O(n)$
- Spojové seznamy
  - hledání, vkládání, mazání

Opakování

- Pole
  - hledání $O(n), O(logn)$
  - vkládání $O(n)$
    - navíc problém s omezenou velikostí
  - mazání $O(n)$
- Spojové seznamy
  - hledání, vkládání, mazání $O(n)$
    - není problém s velikostí

Opakování

- Pole
  - hledání $O(n), O(log n)$
  - vkládání $O(n)$
    - navíc problém s omezenou velikostí
  - mazání $O(n)$
- Spojové seznamy
  - hledání, vkládání, mazání $O(n)$
    - není problém s velikostí
- Stromy
  - hledání, vkládání, mazání

Opakování

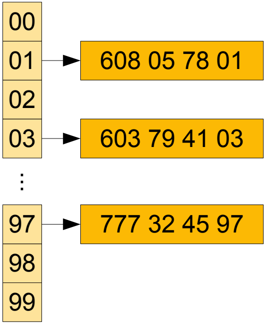
- Pole
  - hledání $O(n)$, $O(logn)$
  - vkládání $O(n)$
    - navíc problém s omezenou velikostí
  - mazání $O(n)$
- Spojové seznamy
  - hledání, vkládání, mazání $O(n)$
    - není problém s velikostí
- Stromy
  - hledání, vkládání, mazání $O(logn)$
- Nešlo by to rychleji?
  - přístup do paměti pomocí adresy – O(1)
  - najít způsob, jak přepočítat klíč na adresu

Hašování

- Hašovací funkce
  - mapuje klíče na celá čísla z intervalu $[0, N-1]$
    - např. pro celá čísla $h(x) = xmodN$
  - $h(x)$ je hašovací hodnota klíče x
- Hašovací tabulka
  - pole velikosti N
- Položka (klíč, data) se ukládá do tabulky na pozici $h(k)$

Příklad

- Telefonní čísla podle posledního dvojčíslí
  - počet možných čísel: 1 000 000 000
  - velikost tabulky: 100



Hašovací funkce

- Obvykle dvě funkce
  - generování haš kódu
    - klíč $\longrightarrow$ celé číslo
  - kompresní funkce
    - celé číslo $\longrightarrow [0, N-1]$
- Cílem hašovací funkce je rovnoměrné rozprostření klíčů na celý interval $[0, N-1]$

Haš kód

- Integer cast
  - klíč se rozdělí na části, které se interpretují jako cifry
  - vhodné pro krátké klíče
    - vejdou se do typu int
  - např. float klíče
- Součet komponent
  - klíč se rozdělí na části, které se poscítají
    - bez ošetření přetečení
  - vhodné pro velká čísla
  - např. long, double

Haš kód

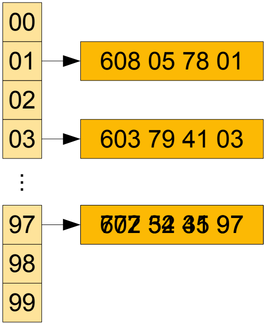
- Výpočet polynomu
  - klíč se rozdělí na části $a_0..a_{n-1}$
  - vyhodnotí se polynom
    $p(z) = a_0 \cdot z^0 + a_1 \cdot z^1 + \cdots + a_{n-1} \cdot z^{n-1}$
  - z je fixní, neošetřuje se přetečení
  - rychlé vyčíslení pomocí Hornerova schématu
    - složitost $O(n)$
      $p_0(z) = a_{n-1}, p_i(z) = a_{n-i-1} + z \cdot p_{i-1}(z)$
  - vhodné pro řetězce

Kompresní funkce

- Celočíselné dělení
  - $h(y) = y \mod N$
  - N je velikost tabulky
    - $N \approx 2\cdot$ očekávaný počet položek
- Multiply, Add, Devide (MAD)
  - vynásobení, přičtení a vydělení
  - $h(y) = (a \cdot y + b) \mod N$
  - a, b libovolná celá čísla
    - $a \mod N \neq 0$

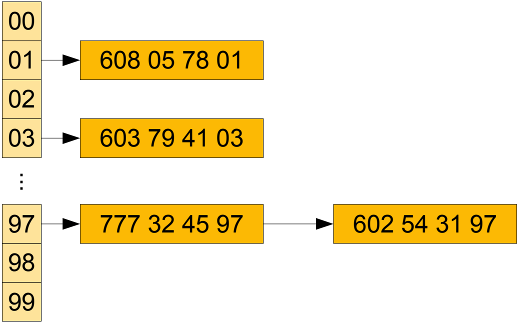
Kolize

- Po kompresi nemusí platit, že jednomu kód odpovídá právě jeden klíč
  - v tabulce dochází ke kolizi



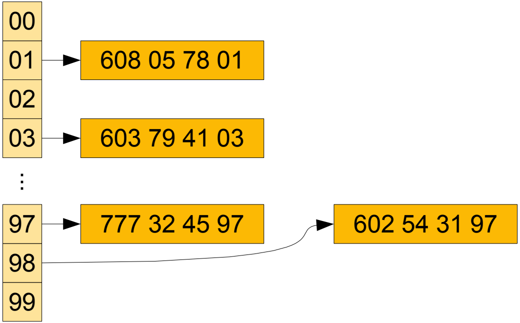
Oddělené řetězení

- Rozšíření položky v poli na spojový seznam
  - libovolný počet prvků
  - roste spotřeba paměti
  - potřebuje složitější zacházení



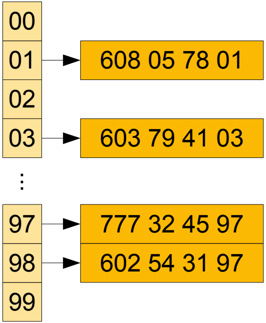
Otevřené adresování

- Kolidující položky se umístí do jiné buňky v tabulce
- Lineární zkoušení
  - pokouší se umístit kolidující položku do další buňky (cyklicky)
  - mohou vznikat kolize, které by předtím nevznikly
- Možnost upravit na kvadratické zkoušení
  - na pozicích $x + 1$, $x + 2$, $x + 4$ ...



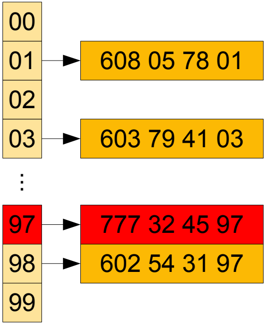
Otevřené adresování

- Kolidující položky se umístí do jiné buňky v tabulce
- Lineární zkoušení
  - pokouší se umístit kolidující položku do další buňky (cyklicky)
  - mohou vznikat kolize, které by předtím nevznikly
- Možnost upravit na kvadratické zkoušení
  - na pozicích $x + 1$, $x + 2$, $x + 4$ ...



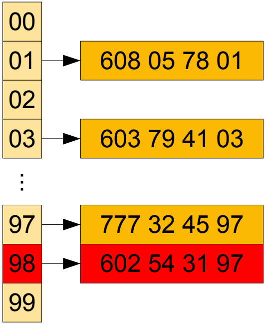
Operace hledání

- Začátek hledání na pozici $h(k)$
- Pokud je buňka prázdná, prvek neexistuje
- Pokud obsahuje prvek s klíčem k, je nalzeno
- Jinak se posuneme na další buňku a opakujeme postup
  - v nejhorším teoretickém případě končíme po zkontrolování celého pole
- Např. číslo 605 89 63 97



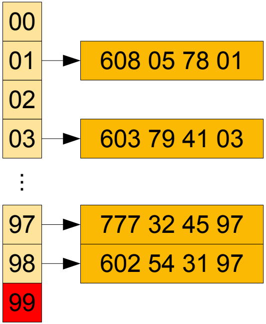
Operace hledání

- Začátek hledání na pozici $h(k)$
- Pokud je buňka prázdná, prvek neexistuje
- Pokud obsahuje prvek s klíčem k, je nalzeno
- Jinak se posuneme na další buňku a opakujeme postup
  - v nejhorším teoretickém případě končíme po zkontrolování celého pole
- Např. číslo 605 89 63 97



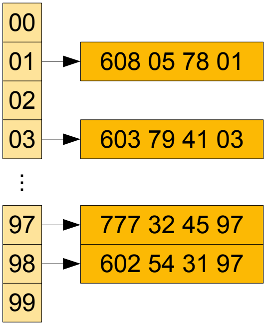
Operace hledání

- Začátek hledání na pozici $h(k)$
- Pokud je buňka prázdná, prvek neexistuje
- Pokud obsahuje prvek s klíčem k, je nalzeno
- Jinak se posuneme na další buňku a opakujeme postup
  - v nejhorším teoretickém případě končíme po zkontrolování celého pole
- Např. číslo 605 89 63 97



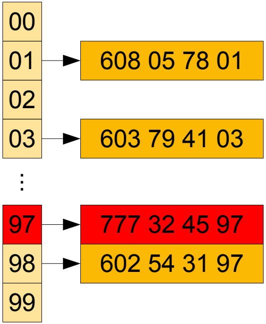
Operace vkládání

- Pokud je tabulka plná, nelze vložit další prvek
  - pokud k tomu došlo, udělali jsme chybu v návrhu
- Začneme na pozici $h(k)$
- Zkoušíme postupně pozice dokud nenarazíme na „vhodnou“ buňku
- Na tuto pozici vložíme prvek
- Např. číslo 605 89 63 97



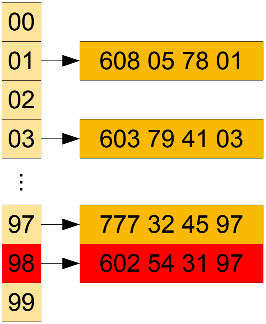
Operace vkládání

- Pokud je tabulka plná, nelze vložit další prvek
  - pokud k tomu došlo, udělali jsme chybu v návrhu
- Začneme na pozici $h(k)$
- Zkoušíme postupně pozice dokud nenarazíme na „vhodnou" buňku
- Na tuto pozici vložíme prvek
- Např. číslo 605 89 63 97



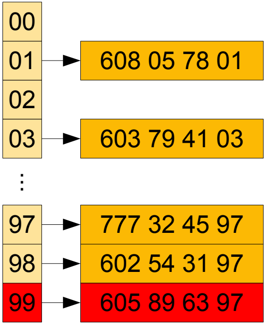
Operace vkládání

- Pokud je tabulka plná, nelze vložit další prvek
  - pokud k tomu došlo, udělali jsme chybu v návrhu
- Začneme na pozici $h(k)$
- Zkoušíme postupně pozice dokud nenarazíme na „vhodnou“ buňku
- Na tuto pozici vložíme prvek
- Např. číslo 605 89 63 97



Operace vkládání

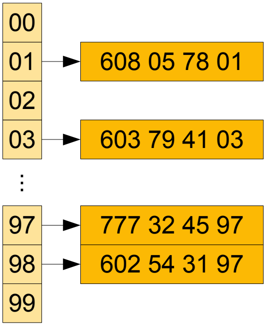
- Pokud je tabulka plná, nelze vložit další prvek
  - pokud k tomu došlo, udělali jsme chybu v návrhu
- Začneme na pozici $h(k)$
- Zkoušíme postupně pozice dokud nenarazíme na „vhodnou“ buňku
- Na tuto pozici vložíme prvek
- Např. číslo 605 89 63 97



Operace mazání

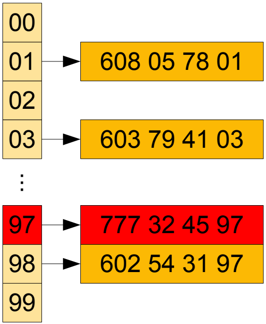
- Nalzneme prvek s klíčem k a...

- Např. číslo 777 32 45 97



Operace mazání

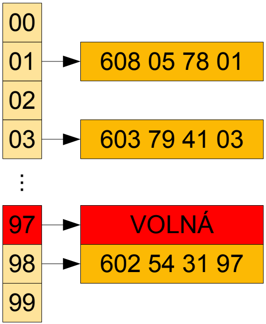
- Nalzneme prvek s klíčem k a...
- Např. číslo 777 32 45 97



Operace mazání

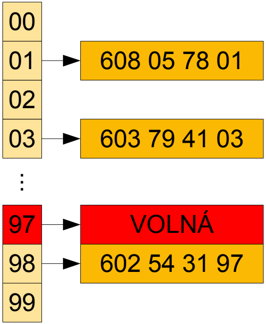
- Nalzneme prvek s klíčem *k* a...
  uložíme specielní položku označující volnou buňku

- Např. číslo 777 32 45 97



Operace mazání

- Nalzneme prvek s klíčem *k* a...
  uložíme specielní položku označující volnou buňku
- Proč tak složitě?
  - při hledání musíme vědět, že tady něco bylo
- Např. číslo 777 32 45 97



Dvojité hašování

- Pro zlepšení řešení kolizí možno použít druhou hašovací funkci
- Při kolizi se posouváme o krok daný druhou funkcí
  - např. $d(k) = a - (k \mod a)$, kde a je konstanta
- $d(k)$ nesmí vracet 0
- $d(k)$ se nesmí chovat stejně jako primární hašovací funkce

Vlastnosti

- Nejhorší možný případ
  - všechny klíče kolidují
  - operace O(n)
  - chyba je na naší straně
- Faktor naplnění
  - $A = n/N$
  - poměr počtu položek k velikosti tabulky
  - čím víc se blíží 1, tím větší je pravděpodobnost kolize
- Očekávaná složitost
  - O(1)
    - pro faktor naplnění kolem 0.7 (70%)

Konec