Skip-List

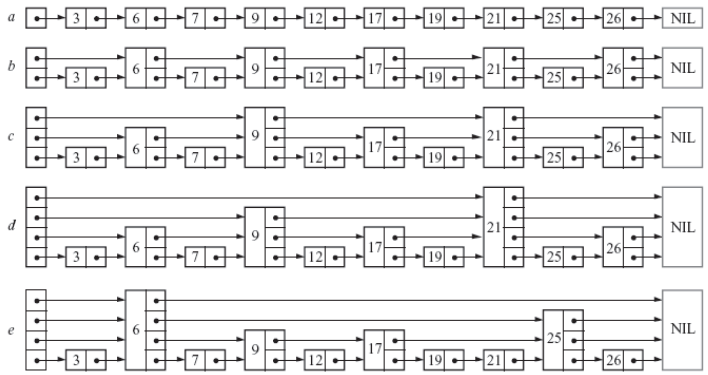
- je datová struktura, která může být použita jako náhrada za vyvážené stromy.
- představují pravděpodobnostní alternativu k vyváženým stromům (struktura jednotlivých uzlů se volí náhodně)
- Na rozdíl od stromů má **skip list** následující výhody:
  - jednoduchá implementace
  - jednoduché algoritmy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

Skip-List

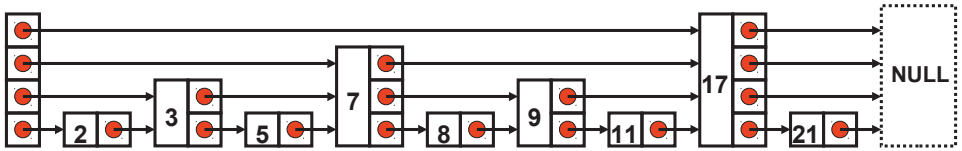
- prky v seznamu jsou uspořádány
- seznam obsahuje prvky, které mají k ukazatelů

  $1 \le k \le max\_level$
- 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

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)	???



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.

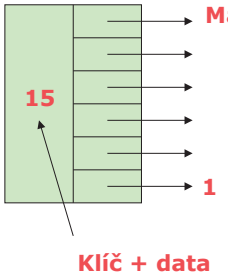
Výhoda: složitost vyhledávání O(log n)

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

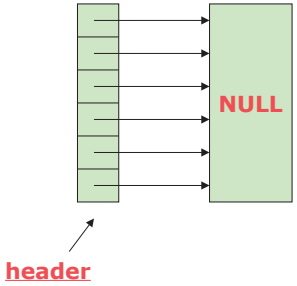
Řešení: ponechat rozložení uzlů ale vyhnout se restrukturalizaci – tj.
uzly úrovně k jsou vkládány náhodně s uvedeným pravděpodobnostním rozložením

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=$\log_2(N)$

Algoritmus vyhledávání

```
Search(list, searchKey)
    x := list→header
    -- loop invariant: x→key < searchKey
    for i := list→level downto 1 do
        while x→forward[i]→key < searchKey do
            x := x→forward[i]
    -- x→key < searchKey ≤ x→forward[1]→key
    x := x→forward[1]
    if x→key = searchKey then return x→value
        else return failure
```

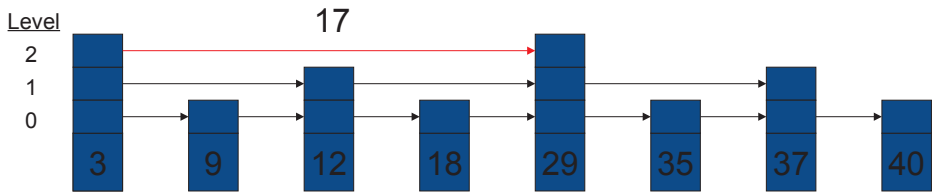
Vyhledávání

- Začínáme v nejvyšší úrovni
- Dokud je hledaný prvek větší než prvek na který ukazuje ukazatel,
  - posouváme se vpřed v dané úrovni .
- Pokud je hledaný prvek menší než následující klíč,
  - přesuneme se o jednu úroveň níž.
- Opakujeme postup pokud není prvek nalezen, nebo pokud není jisté (v úrovni 1), že prvek neexistuje.

- Časová složitost
  - nejlepší/průměrný případ : logaritmický
  - nejhorší případ : lineární (skip list přechází v normální spojový seznam)

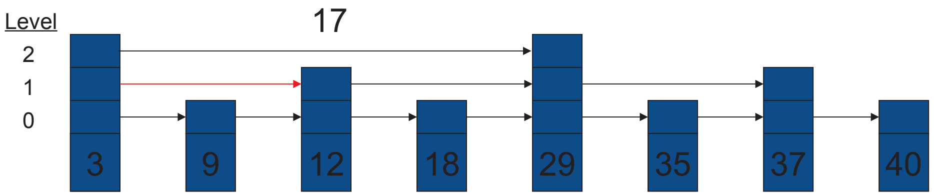
Vložení prvku

- Vyhledávacím algoritmem nalezněte pozici pro vložení prvku
  - zapamatujte pozici předchůdce
- Zvolte úroveň nově vkládaného uzlu
- Vložte nový uzel a pokud je to nutné zvětšete hodnotu MaxLevel



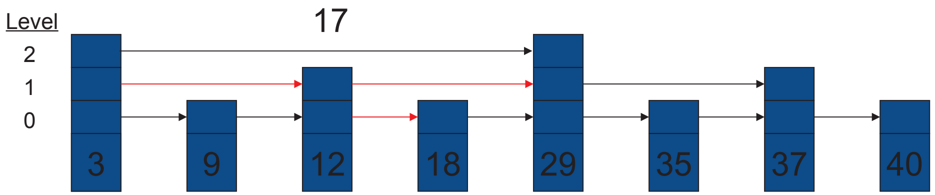
Vložení prvku

- Vyhledávacím algoritmem nalezněte pozici pro vložení prvku
  - zapamatujte pozici předchůdce
- Zvolte úroveň nově vkládaného uzlu
- Vložte nový uzel a pokud je to nutné zvětšete hodnotu MaxLevel



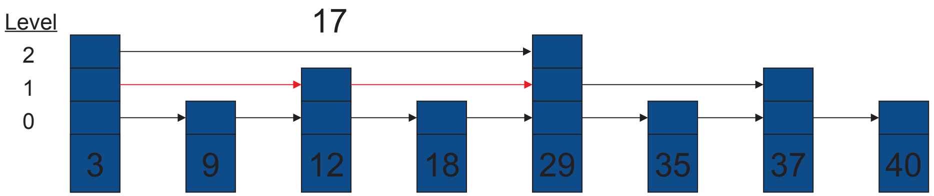
Vložení prvku

- Vyhledávacím algoritmem nalezněte pozici pro vložení prvku
  - zapamatujte pozici předchůdce
- Zvolte úroveň nově vkládaného uzlu
- Vložte nový uzel a pokud je to nutné zvětšete hodnotu MaxLevel



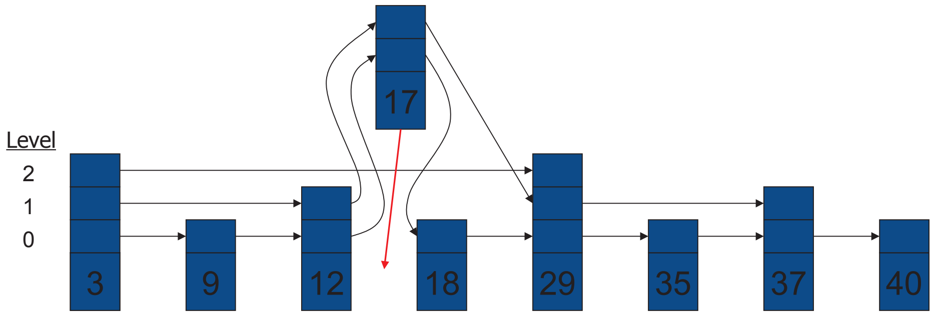
Vložení prvku

- Vyhledávacím algoritmem nalezněte pozici pro vložení prvku
  - zapamatujte pozici předchůdce
- Zvolte úroveň nově vkládaného uzlu
- Vložte nový uzel a pokud je to nutné zvětšete hodnotu MaxLevel



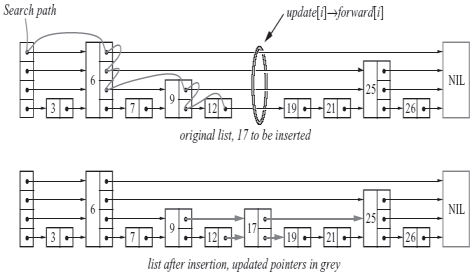
Vložení prvku

- Vyhledávacím algoritmem nalezněte pozici pro vložení prvku
  - zapamatujte pozici předchůdce
- Zvolte úroveň nově vkládaného uzlu
- Vložte nový uzel a pokud je to nutné zvětšete hodnotu MaxLevel



Volba náhodné úrovně

```
randomLevel()
    lvl := 1
    -- random() that returns a random value in [0...1)
    while random() < p and lvl < MaxLevel do
        lvl := lvl + 1
    return lvl
```



Algoritmus vložení prvku

```
Insert(list, searchKey, newValue)
    local update[1..MaxLevel]
    x := list→header
    for i := list→level downto 1 do
        while x→forward[i]→key < searchKey do
            x := x→forward[i]
        -- x→key < searchKey ≤ x→forward[i]→key
        update[i] := x
    x := x→forward[1]
    if x→key = searchKey then x→value := newValue
    else
        lvl := randomLevel()
        if lvl > list→level then
            for i := list→level + 1 to lvl do
                update[i] := list→header
            list→level := lvl
        x := makeNode(lvl, searchKey, value)
        for i := 1 to level do
            x→forward[i] := update[i]→forward[i]
            update[i]→forward[i] := x
```

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...

Zrušení prvku

- Vyhledávacím algoritmem nalezněte pozici pro zrušení prvku
  - zapamatujte pozici předchůdce
- Zrušte uzel, je-li to nutné zmenšete MaxLevel.

```
Delete(list, searchKey)
    local update[1..MaxLevel]
    x := list→header
    for i := list→level downto 1 do
        while x→forward[i]→key < searchKey do
            x := x→forward[i]
        update[i] := x
    x := x→forward[1]
    if x→key = searchKey then
        for i := 1 to list→level do
            if update[i]→forward[i] ≠ x then break
            update[i]→forward[i] := x→forward[i]
        free(x)
        while list→level > 1 and
                list→header→forward[list→level] = NIL do
            list→level := list→level – 1
```