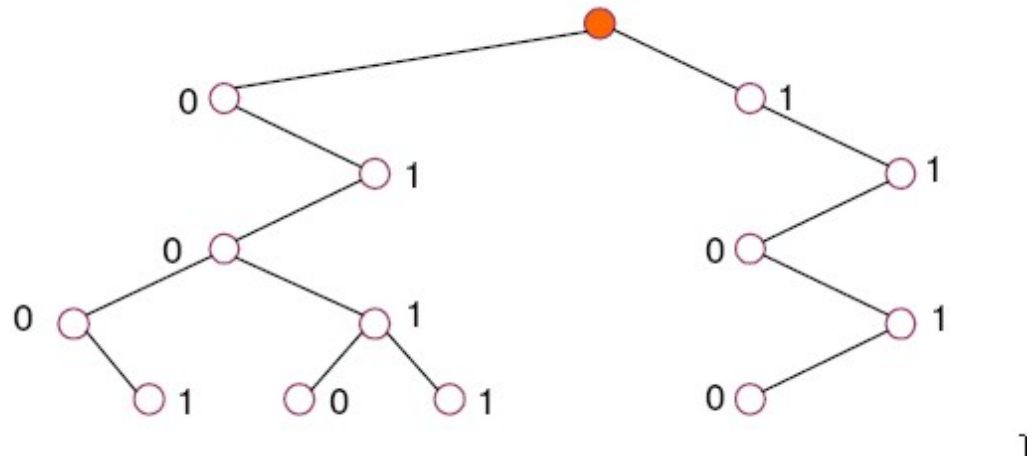
Vyhledávací stromy

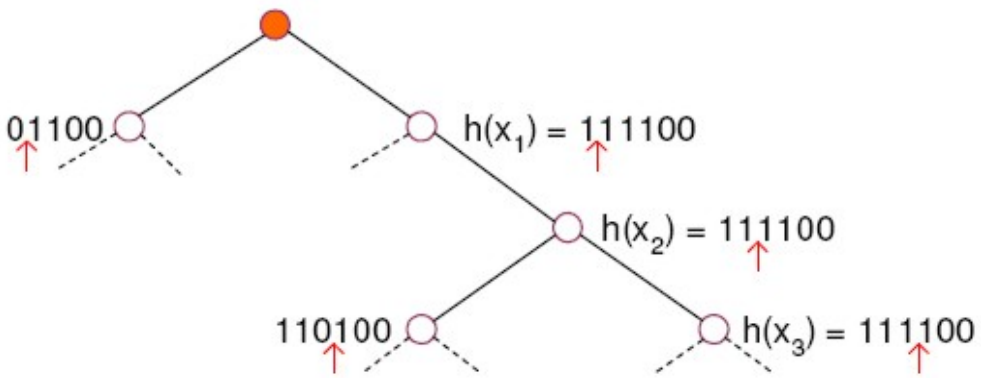
- Slouží jako pomůcka pro organizaci dat umožňující efektivní vyhledávání.
- Vytvářejí se vždy nad již existující datovou strukturou (zpravidla tabulkou).
- **Vyhledávací stromy** můžeme **rozdělit** na:
  - **Adresní** - umístění prvku ve stromu je dáno přímo jeho klíčem
  - **Vyhledávací** - umístění je dáno vztahy mezi prvky, vyhledává se porovnáváním klíčů
  - **Přímé** - uzly stromu obsahují přímo klíče, podle kterých je strom organizován
  - **Nepřímé** - v uzlech stromu jsou uchovávány pouze indexy např. do tabulky, kde jsou teprve uloženy vlastní prvky, resp. klíče
  - **Binární stromy** – strom, jehož každý uzel má (maximálně) dva následovníky (syny)
  - **1-2 stromy** - smíšené stromy (v uzlech jsou klíče, v listech odkazy na data)
  - **B-stromy** - *n*-ární stromy, které mají speciální strukturu, proto se vyčleňují zvlášť

Adresní stromy

- **Adresní stromy** lze rozdělit na:
  - **Hashovací** - vznikly kombinací hash-tabulek se stromovými grafy
    - Hodnoty hash-funkce se interpretují jako bitové řetězce, jimiž se kódují přístupové cesty v patřičném stromu (0 = levý syn, 1 = pravý syn).



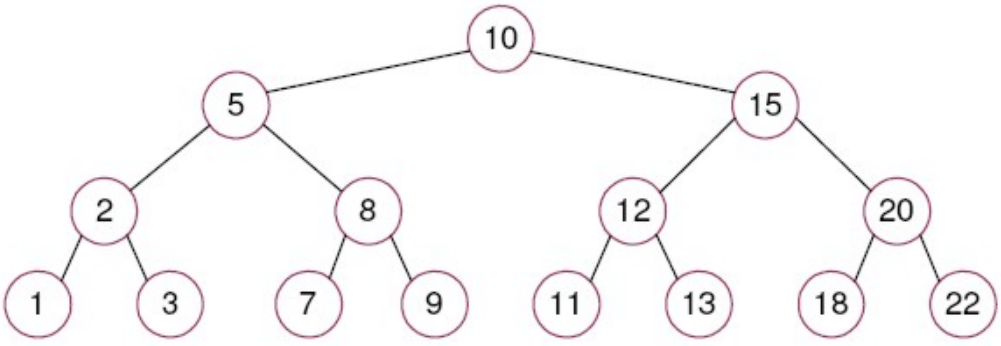
- **Bitové** - vzniknou z hashovacích, položíme-li $h(x) = x$, prvky umístíme do listů (na konec cesty) odpovídajícím bitovým řetězcům.
  - *Příklad*: Adresní strom reprezentující množinu **{01001, 01010, 01011, 11010}**



Binární vyhledávací stromy

- **Binární vyhledávací strom** (**BVS**) reprezentující uspořádanou množinu **S** je kořenový uzlově ohodnocený binární strom s množinou uzlů **V** a ohodnocením **κ** takovým, že:
  - $\kappa(v) = s \qquad v \in V; s \in S$
  - $\kappa(v_l) \leq \kappa(v) \leq \kappa(v_p)$ pro libovolný uzel v_l v levém a v_p v pravém podstromu s kořenem v uzlu v.

  - *Příklad*: BVS množiny **S = {10, 5, 15, 2, 8, 12, 20, 1, 3, 7, 9, 11, 13, 18, 22}**



- **Deklarace BVS:**

```
type    Typ_hodnoty = … ;
        Bod = ^Uzel;
        Uzel = record
             Hodnota: Typ_hodnoty;
             Levy, Pravy: Bod;
        end;

var BVS: Bod;
```

Prohledávání BVS:

```
function Prohledavani(X:Typ_hodnoty; Strom:Bod):boolean;
begin
                                {prvek X nenalezen}
   if S=nil then Prohledavani:=false
   else
   if X<Strom^.Hodnota then {prohlížení levého podstromu}
        Prohledavani:=Prohledavani(X,Strom^.Levy)
   else
   if X>Strom^.Hodnota then {prohlížení pravého podstromu}
        Prohledavani:=Prohledavani(X,Strom^.Pravy)
   else
        Prohledavani:=true  {prvek X nalezen}
end;
```

- Lehce modifikovatelný algoritmus lze použít i pro **vkládání uzlů** (prvků) do stromu.

Rušení (smazání) uzlu:

- **V listu** - není problém
- **Ve vnitřním uzlu** - prvek nahradíme **symetrickým předchůdcem** (nejpravějším uzlem z levého podstromu) nebo **symetrickým následovníkem** (nejlevějším uzlem z pravého podstromu) některou z následujících metod:
  - Zvolíme napevno předchůdce, nebo následovníka - musíme ale zajistit jeho existenci, což může být problém.
  - Stejný přístup, ale pokud předchůdce, případně následovník neexistuje, zvolíme druhou alternativu.
  - Zjistíme, který strom je větší, a z té strany vezmeme náhradní uzel (zaručuje existenci předchůdce, příp. následovníka uzlu, který chceme odebrat)

```
procedure Zruseni(X:Typ_hodnoty; var Strom:Bod);

var Q: Bod;

   procedure Nahrada(var W:Bod);
             {náhrada rušeného prvku symetrickým předchůdcem}
   begin
             {jdeme doprava tak dlouho, až to už nejde
              -> najdeme list - symetrického předchůdce}
        if W^.Pravy<>nil then Nahrada(W^.Pravy)
        else begin          {náhrada rušeného prvku}
             Q^.Hodnota:=W^.Hodnota;
             {hodnota sym. předch. se zapíše na místo
              rušeného prvku}
             Q:= W;
             {posléze, mimo tuto proceduru, se bude
              dealokovat uzel, na který ukazuje Q, a my
              chceme zrušit W}
             W := W^.Levy;  {vlastní vypuštění sym. předch.}
        end;
   end;       {konec Nahrada}
```

```
begin
   if S = nil then ERROR
             {rušený prvek vůbec nebyl ve stromu}
   else if X<Strom^.Hodnota then Zruseni(X,Strom^.Levy)
             {prvek bude v levém podstromu}
   else if X>Strom^.Hodnota then Zruseni(X,Strom^.Pravy)
             {prvek bude v pravém podstromu}
   else begin
             {nalezli jsme rušený prvek}
        Q:=Strom;
        if Q^.Pravy=nil then Strom:=Strom^.Levy
             {uzel má jen levého syna}
        else if Q^.Levy=nil then Strom:=Strom^.Pravy
             {uzel má jen pravého syna}
        else Nahrada(Q^.Levy);
             {uzel má oba syny, musíme jej nahradit
              symetrickým předchůdcem}
        Dispose(Q);
             {zmíněná (Nahrada) dealokace nadbytečného uzlu}
   end;
end;
```

- **Strategie prohledávání BVS**:

Strategie	dopředný směr	zpětný (reverzní) směr
PreOrder	kořen levý podstrom pravý podstrom	kořen pravý podstrom levý podstrom
InOrder	levý podstrom kořen pravý podstrom	pravý podstrom kořen levý podstrom
PostOrder	levý podstrom pravý podstrom kořen	pravý podstrom levý podstrom kořen

- Vzestupný výpis prvků stromu (podle hodnoty klíče) užitím strategie **InOrder**

```
procedure Tisk_Stromu_InOrder(S:Bod);
begin
     if Strom<>nil then
     begin
          Tisk_Stromu_InOrder(Strom^.Levy);
          Write(Strom^.Hodnota);
          Tisk_Stromu_InOrder(Strom^.Pravy)
     end
end;
```

- Výpis prvků stromu od kořene k listům strategií **PreOrder**

```
procedure Tisk_Stromu_PreOrder(S:Bod);
begin
     if Strom<>nil then
     begin
          Write(Strom^.Hodnota);
          Tisk_Stromu_PreOrder(Strom^.Levy);
          Tisk_Stromu_PreOrder(Strom^.Pravy)
     end
end;
```

- **Nevýhody BVS**:
  - Složitost vyhledávání závisí na pořadí vkládání prvků do stromu
  - Složité rušení prvku
  - Je nutno prohledávat vždy od kořene, tj. není možnost vracet se ve stromu k rodičům
    - Do ukazatelů, kde by měl být nil, vložíme ukazatel na prvek, kterému je tento symetrickým následovníkem (pro levý ukazatel), resp. symetrickým předchůdcem (pro pravý ukazatel), tyto ukazatele zpět musí být ovšem odlišeny od obyčejných ukazatelů na syny) – **BVS se zpětným zřetězením**.

Vyvážené stromy

- **Vyvážený strom** je binární vyhledávací strom, pro který je délka jeho nejdelší větve úměrná $\log_2 n$, kde $n = |S|$.

- Rozeznáváme dva **typy vyvážení**:
  - **výškově vyvážené stromy** - požadujeme, aby výšky podstromů byly v rovnováze
  - **váhově vyvážené stromy**

Výškově vyvážené stromy - AVL stromy

- BVS je **AVL-stromem** (Adel'son-Velského a Landisovým) právě tehdy, když se výšky levého a pravého podstromu každého uzlu liší maximálně o 1.

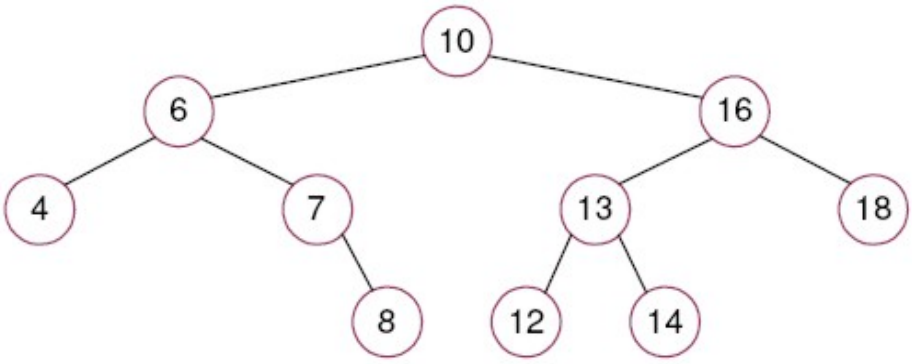
  - Pro výšku $h(n)$ AVL-stromu s n uzly platí:

$$\log_2(n+1) \le h(n) \le 1{,}4404 \cdot \log_2(n+2) - 0{,}328$$

  - Operace `Prohledavani` se nemění, `Vlozeni` a `Ruseni` musí obsahovat operaci **vyvážení stromu.**

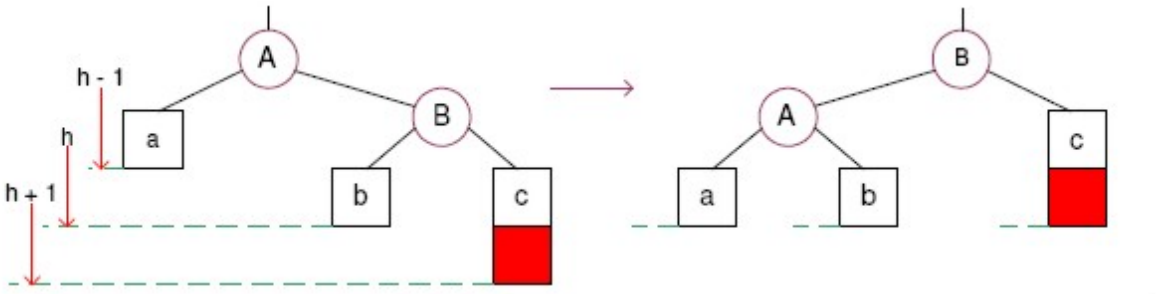
- Pro každý uzel definujeme **stupeň vyváženosti**:
  - **0** - oba podstromy jsou stejně vysoké
  - **-1** - levý podstrom je o 1 vyšší
  - **-2** - levý podstrom je o 2 vyšší
  - **1** - pravý podstrom je o 1 vyšší
  - **2** - pravý podstrom je o 2 vyšší

- *Příklad*: výškově vyvážený AVL-strom

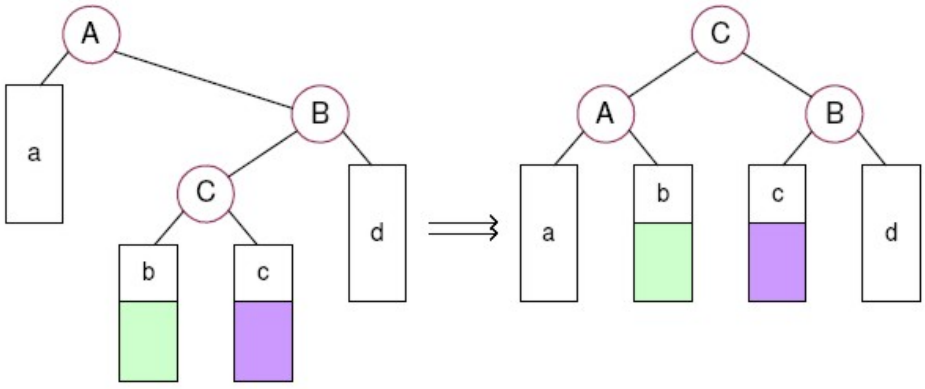


- Pro vyvažování stromu zavádíme **operaci rotace**. Rotace užijeme, má-li nějaký uzel **stupeň vyváženosti ±2**, tedy strom jako celek nesplňuje denici AVL-stromu – je nutné provést vyvažování

  - **Rotace jednoduchá**
    - Jednoduchou rotaci použijeme, jestliže vyvažujeme přímou větev, tj. jsou-li **znaménka** stupňů vyváženosti **stejná**
    - *Příklad*: jednoduchá rotace (pravá) **a(bc) → (ab)c**



  - **Rotace dvojitá**
    - Dvojitá rotace je na místě, jestliže nelze použít jednoduchou, tedy jedná-li se o zalomenou větev, tj. **znaménka** stupňů vyváženosti jsou **různá**.
    - Dvojitou rotaci lze převést na dvě jednoduché (viz symbolický zápis), z toho název dvojitá rotace
    - *Příklad*: dvojitá rotace (pravá) **a((bc)d) → a(b(cd)) → ((ab)(cd))**



- **Rotace levá** - pokud je levý podstrom větší
- **Rotace pravá** - pokud je pravý podstrom větší
  - Levá a pravá rotace je principiálně stejná, jen se vše odehrává zrcadlově, uvažujeme proto vždy jen jeden případ.

- **Zařazení prvku X** do vyváženého stromu **S** představuje kroky:

  - **Vyhledávání X v S**, dokud není zřejmé, že se **X** ve stromu nenachází.
  - **Zařazení X do S** v místě, kde se ukončilo jeho vyhledávání
    - První dva kroky jsou totožné s algoritmem zařazení prvku do obyčejného BVS.

  - Zpětný **přepočet stupňů vyvážení S** od přidaného uzlu vzhůru, tedy od listu ke kořeni. V případě, že došlo k rozvážení **S** (tedy někde se poprvé vyskytne stupeň vyvážení ±2), vykoná se ještě následující bod
  - **Vyvážení S** jednou ze dvou výše uvedených rotací. Nutno podotknout, že po této operaci je vše výše od místa, kde se vyvažovalo, v pořádku.

Váhově vyvážené stromy

- **Váhové vyvážení** spočívá ve vyvážení počtu uzlů levého a pravého podstromu každého uzlu BVS.

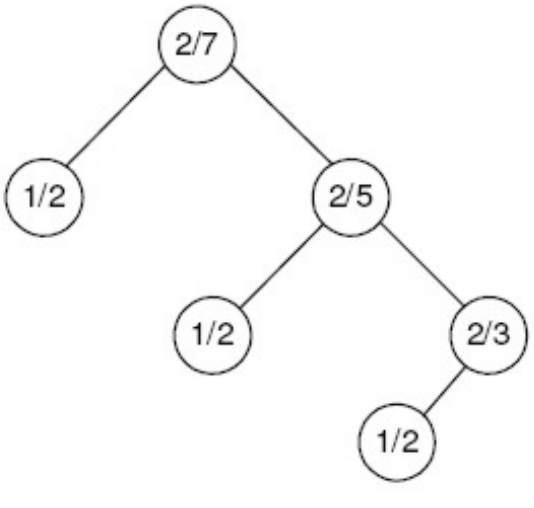
- Funkci $\rho(U)$:

$$\rho(U) = \frac{|T_L| + 1}{|T| + 1}$$

,kde T_L je levý podstrom uzlu U a T je celý strom s kořenem U, nazveme **stupněm vyvážení** uzlu U.

  - **Stupeň vyvážení** každého **listu je ½**.

  - *Příklad*:



- Musíme stanovit, pro jaké hodnoty stupně vyvážení budeme strom považovat za vyvážený, například pro **BB[α] stromy** platí:

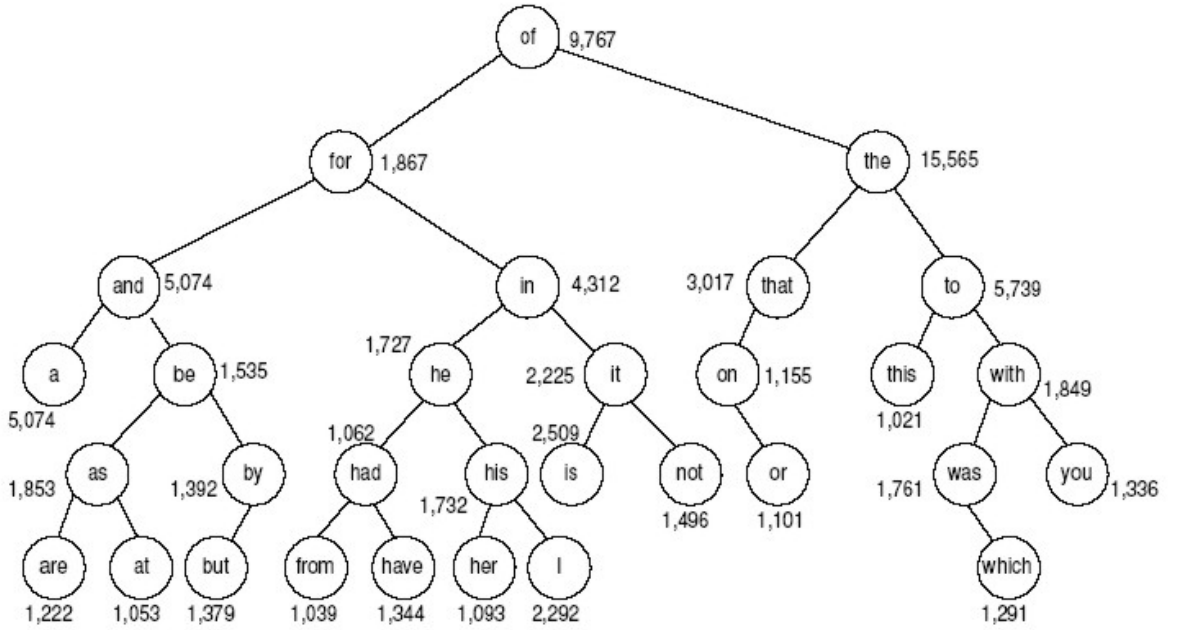
$$\alpha \leq \rho(U) \leq 1 - \alpha$$

- Pro váhově vyvážené stromy platí stejné algoritmy rotací jako pro AVL-stromy.

Optimální vyhledávací stromy

- BVS, které minimalizují složitost vyhledávání v průměrném případě na základě znalosti frekvence přístupů k jednotlivým prvkům reprezentované množiny.

- Řeší se konkrétně pro danou aplikaci
  - *Příklad*: Optimální BVS pro množinu 31 nejfrekventovanějších anglických slov (čísla jsou četnosti výskytu slov)



Samoorganizující se BVS

- Jedná se o variantu optimálních vyhledávacích stromů.

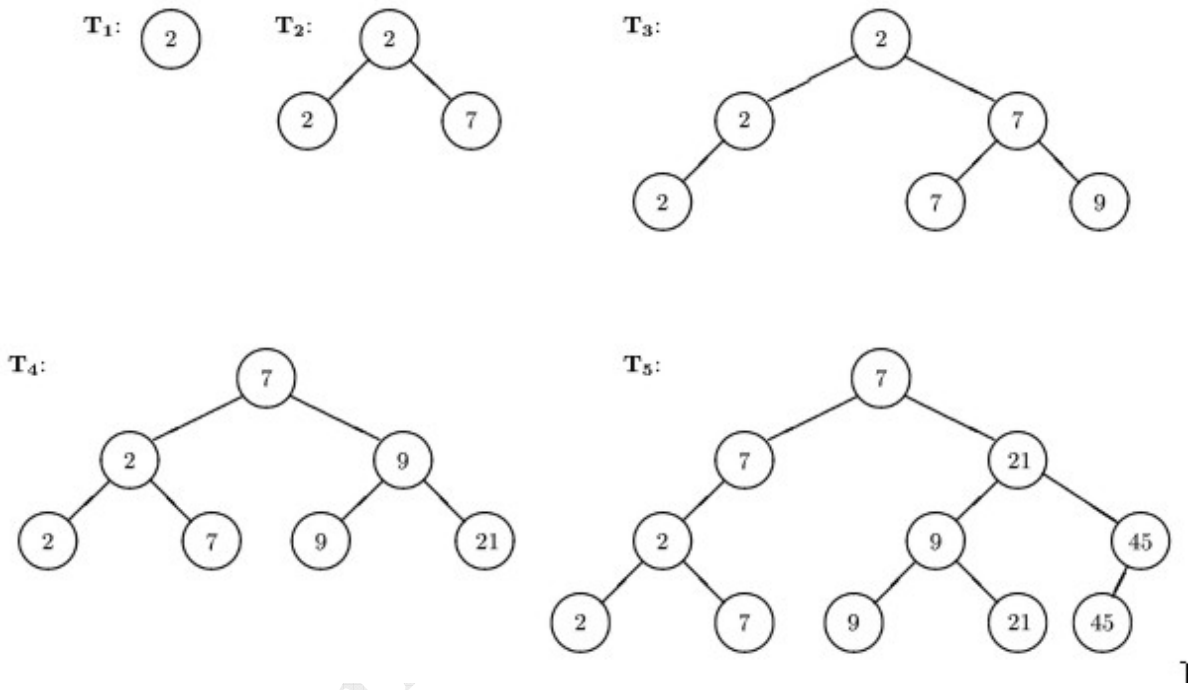
- Automaticky se přizpůsobují měnícím se frekvencím přístupu, není proto nutné frekvence přístupů znát na začátku, strom se doladí za běhu.

1-2 stromy

- **1-2 stromy** jsou smíšené stromy, tj. v uzlech jsou klíče, v listech odkazy na data.

- Binární vyhledávací strom se nazývá 1-2 stromem, jestliže splňuje následující **podmínky**:
  - Všechny listy mají stejnou vzdálenost od kořene
  - Uzel, který má jen jednoho syna, má bratra se dvěma syny
  - Kořen 1-2 stromu musí mít 2 syny, jinak je list
  - Má-li uzel jen jediného syna P, musí P mít dva syny; jinak je P list

- *Příklad*: 1-2 stromy o *n* listech



- **Vytvoření 1-2 stromu** nad uspořádanou množinou **S** o ***n* prvcích**:
  - Množinu o *n* prvcích reprezentujeme 1-2 stromem o *n* listech tak, že jednotlivé prvky množiny přiřadíme ve vzrůstajícím pořadí postupně zleva doprava listům stromu a vnitřním uzlům stromu přiřadíme vždy největší prvek z levého podstromu, jehož je daný uzel kořenem.
  - Tím získáme novou úroveň prvků (prvků z vnitřních uzlů), kterou zpracujeme stejným postupem.
  - Toto opakujeme tak dlouho, až zbude jediný prvek, a ten označíme za kořen.
  - Při postupu je ovšem nutno dávat pozor, aby **nebyly porušeny podmínky 1-2 stromu**, tj. nestačí každou úroveň zpracovávat zleva doprava. Správného řešení je možno docílit například tím, že směr zpracování jednotlivých úrovní střídáme, nebo dáváme pozor na připojování nejpravějšího prvku - jestliže byl při zpracování minulé úrovně ponechán sólo, tentokrát se musí spárovat.

- **Složitost prohledávání** 1-2 stromu je úměrná výšce stromu, čili logaritmická, neboť výška stromu je dána vztahem:

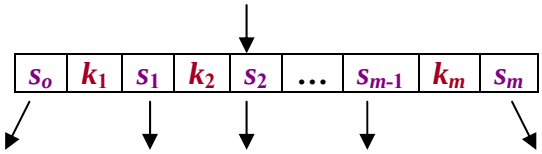
$$h = k \cdot \log_2(N_U + 1)$$

kde $k \in \langle 1{,}0;\ 1{;}44 \rangle$ a N_U je celkový počet uzlů stromu (včetně vnitřních)

B-stromy

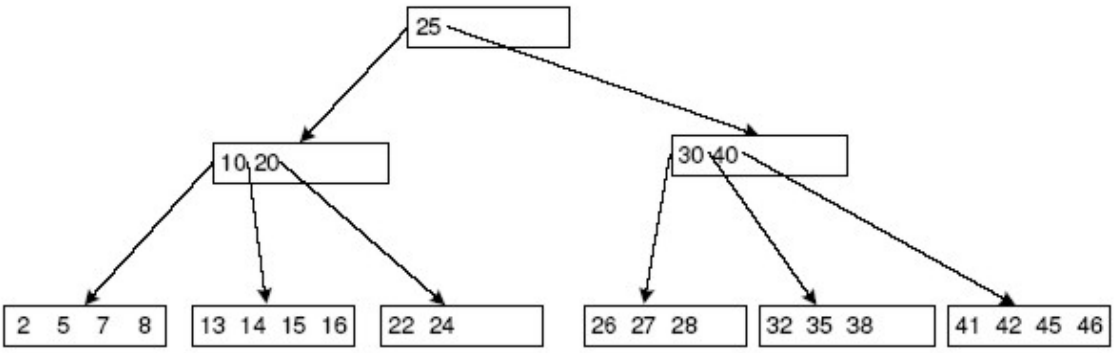
- **B-stromy** jsou zvláštním typem *n*-árních stromů, kde uzly se nazývají **stránky** a obsahují více klíčů ($k_1 \ldots k_m$) a obvykle více odkazů na syny ($s_0 \ldots s_m$).

stránka:

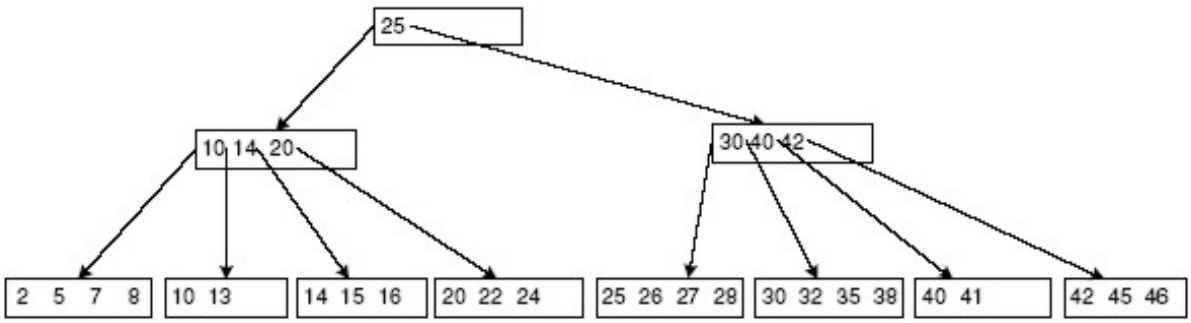


- **Obecné pravidlo pro vytvoření B-stromu** *n*-tého řádu o ***N*** položkách:
  - Každá **stránka** stromu (kromě jedné - kořene) musí obsahovat ***n* až 2*n* prvků** pro daný řád stromu ***n*** (konstanta).
  - Zpravidla určíme ***n* = počet úrovní stromu - 1**
  - Potom pro strom s ***N*** položkami a max. velikostí stránky **2*n*** prvků bude nejhorší případ vyhledávání vyžadovat **$\log_n N$** přístupů ke stránkám
  - Faktor **plnění ≥ 0,5**

- **Souhrnná charakteristika vlastností B-stromu:**
  - Každá stránka obsahuje **nejvýše 2*n* položek** (klíčů)
  - Každá stránka (kromě kořenové) obsahuje **alespoň *n* položek**
  - Každá stránka je buď listovou stránkou, tj. nemá **žádného syna**, nebo má ***m* + 1 synů**, kde *m* je počet jejích prvků (**$n \le m \le 2n$**)
  - Všechny listy jsou na stejné úrovni



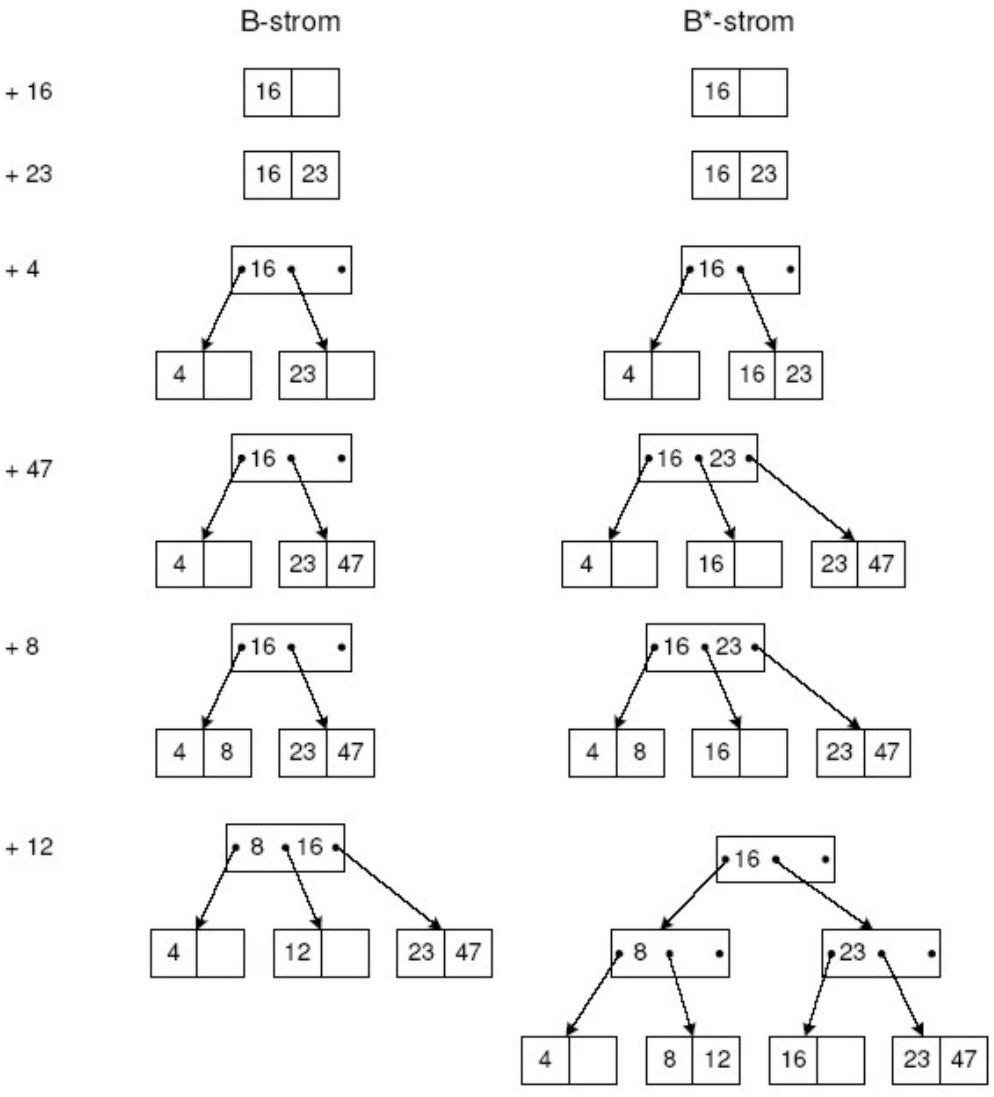
  - **B*stromy** jsou stromy podobné B-stromům, avšak hodnoty klíčů se zde opakují, všechny klíče se vyskytují v listových stránkách.



- **Vyhledání prvku** (s klíčem **X**):
  - pro $k_i < X < k_{i+1}$ pro $1 \leq i < m$, vyhledáváme v datech příslušných stránce s_i
  - pro $X > k_m$, vyhledáváme v s_m
  - pro $X < k_1$, vyhledáváme v s_0

  - Postup vyhledávání **opakujeme** tak dlouho, dokud požadovaný záznam s klíčem **X** nenajdeme.
  - Má-li však s_i hodnotu **nil**, další stránka neexistuje a hledaný **prvek** (záznam) se v B-stromu **nevyskytuje**.

- **Vkládání prvku:**
  - Vkládáme prvky na místa, kam patří
  - Pokud by měla stránka více, než **2*n*** vrcholů (tzv. **přeplněná stránka**), rozdělí se na dvě stránky (levá, pravá) a prostřední klíč se přesune do nadřazené stránky.
  - Tím se do nadřazené stránky přidává prvek a to řešíme stejným způsobem (tj. opět může dojít k přeplnění a budeme spojovat stránky na vyšší úrovni, něco se přesune výše, …)

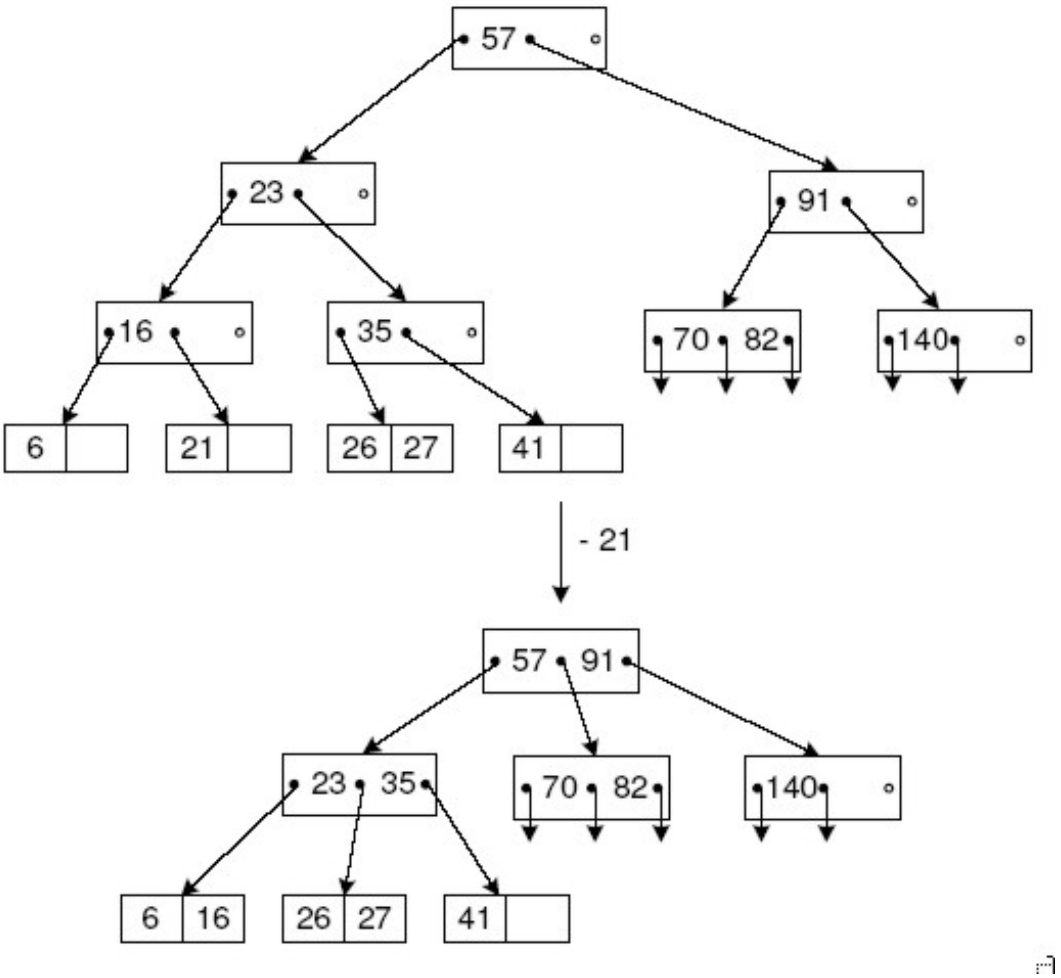


  - U **B*stromu** používáme jako klíč v nadřazené stránce vždy **hodnotu klíče symetrického následovníka**

Rušení prvku:

- Pokud je rušený prvek v listové stránce, jednoduše jej umažeme.
- Pokud je ve vnitřní stránce, nahradíme jej **symetrickým následovníkem** (který musí být u B-stromu a B*stromu vždy v listové stránce, neboť tyto stromy jsou vždy doleva rozvětvené) a ten z listové stránky umažeme. Tedy opět mažeme jen v listové stránce.

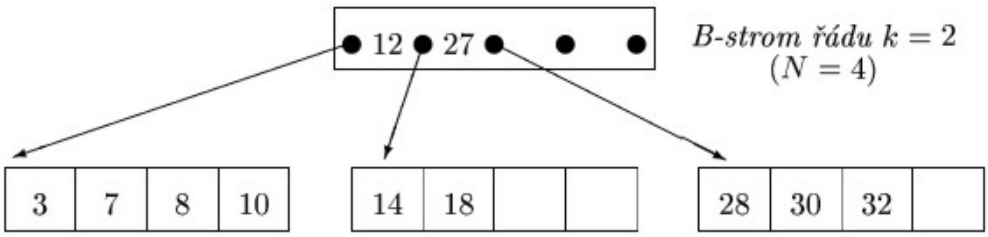
- Problém nastává, pokud tato stránka po odebrání prvku obsahuje méně, než *n* vrcholů. Potom musíme provést **rekonfiguraci** – máme **dvě možnosti:**
  - **Vypůjčení prvku ze sousední stránky** - má-li jedna z vedlejších stránek dost prvků (tedy více, než je řád stromu *n*), krajní prvek z této sousední stránky zaměníme s rodičem dělící naši stránku a tuto sousední. Rodiče pak vložíme do naší stránky.
  - **Spojování stránek** - pokud nelze užít postup z předešlého bodu, je zřejmě v obou sousedních stránkách právě *n* prvků. Proto můžeme naši stránku (ta má *n* -1 prvků) s jednou z nich spojit (tedy obsah jedné ze stránek vkopírovat do druhé, uprázdněnou stránku smazat a konečně vložit do výsledné stránky rodiče, který v minulosti spojované stránky dělil). Nová stránka tudíž obsahuje právě 2*n* prvků. Tímto postupem jsme ovšem odebrali jeden prvek (rodiče) z nadřazené stránky, může tedy opět dojít k nedostatku prvků. Problém řešíme opětnou aplikací celého postupu (tj. oba body - vypůjčení i spojování) na stránky na této úrovni.

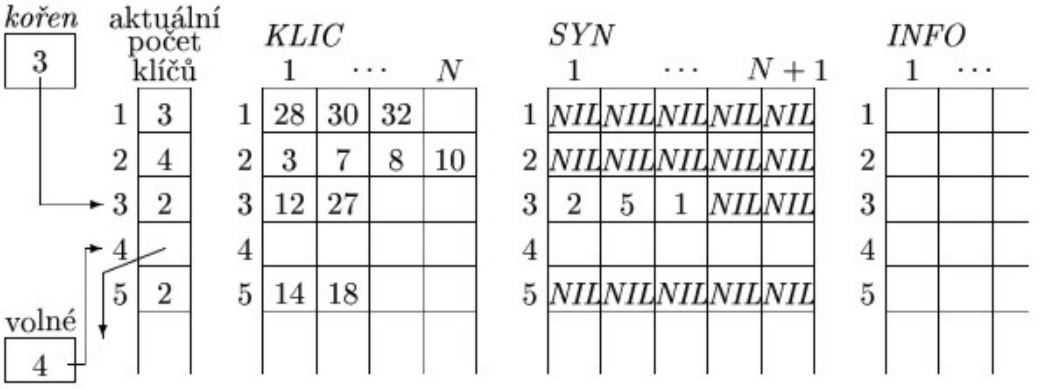


Implementace B-stromů:

Statická implementace

- Zásadně implementujeme strom jako tabulky
- Musíme zvolit maximální strom, který můžeme implementovat (podle počtu stránek volíme velikost implementujících tabulek)





- **Výhoda**: nejrychleji se implementuje, nejrychleji se s ním pracuje.

Dynamická implementace:

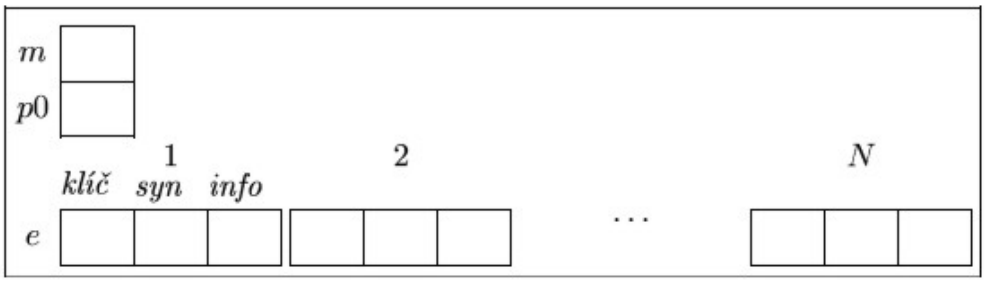
- Používáme dynamické datové objekty
- Možná definice struktury stránky:

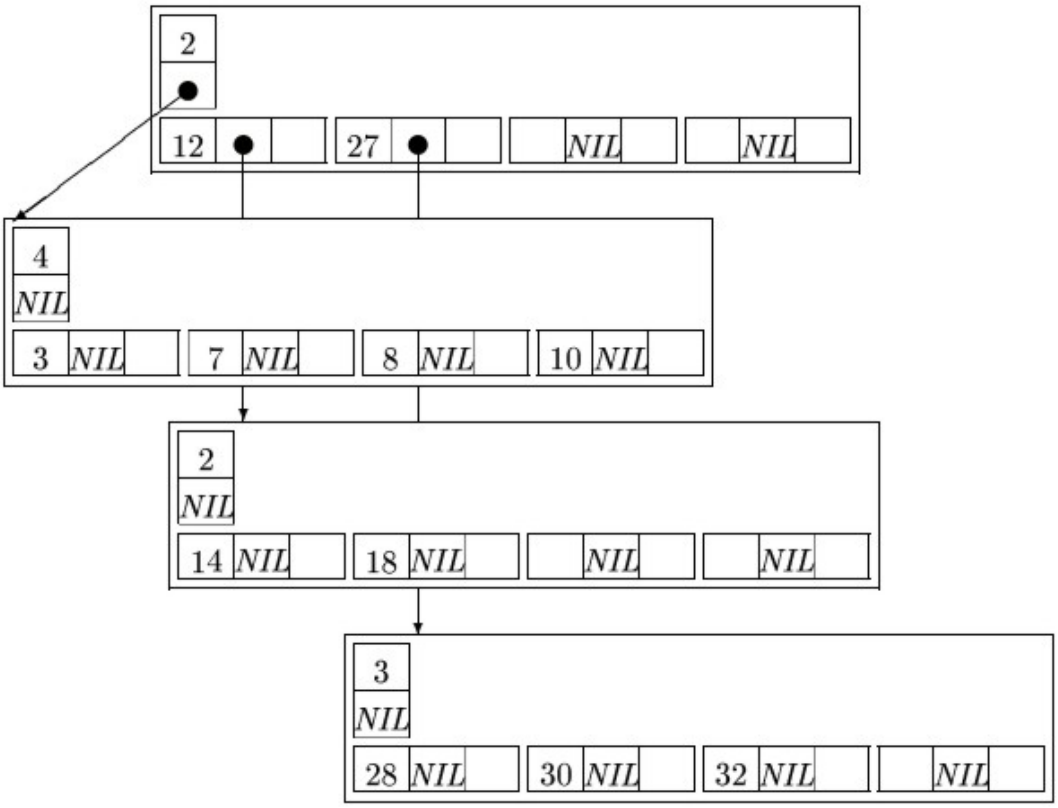
```
const    K=…;          {řád stromu}
         N=2*K;

type     index = 0..N;
         Odkaz = ^Stranka;

         Prvek = record
           KLIC:integer;
           SYN: Odkaz;
           INFO: Typ_info;
         end;

         Stranka = record
           M: index;
           P0: Odkaz;                    {prvek P0}
           P: array[1..N] of Prvek;      {ostatní prvky}
         end;
```


m
p0
1
2
N
klíč
syn
info
e
...


2
12
27
NIL
NIL
4
NIL
3
NIL
7
NIL
8
NIL
10
NIL
2
NIL
14
NIL
18
NIL
NIL
NIL
3
NIL
28
NIL
30
NIL
32
NIL
NIL