STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

Binárne (vyhľadávacie) stromy

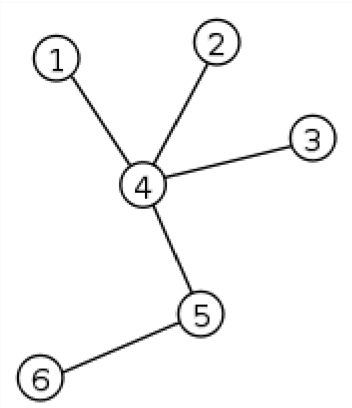
10. 3. 2021

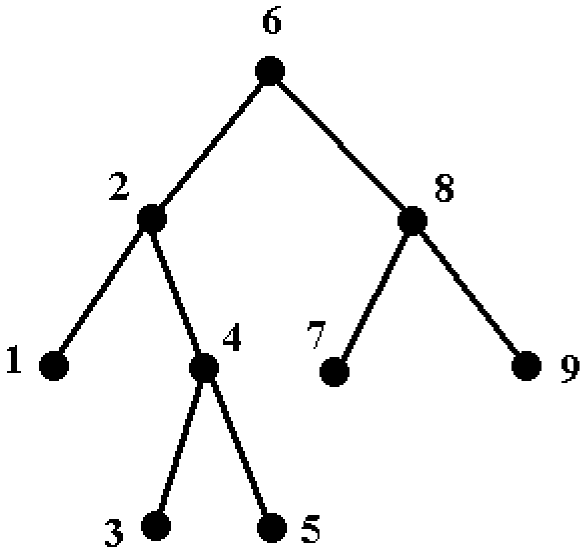
letný semester 2020/2021

prednášajúci: Lukáš Kohútka

Strom – Definícia (teória grafov)

Strom – Súvislý neorientovaný graf bez cyklov



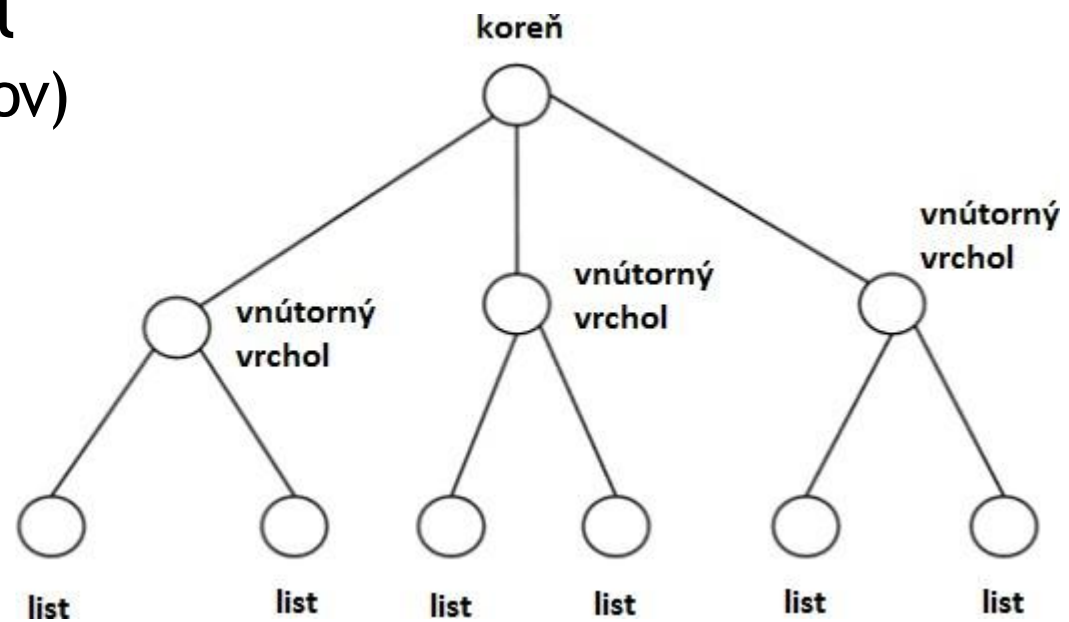


Graf G = (V, E)
V – množina vrcholov
E – množina hrán (dvojíc vrcholov)

Neorientovaný graf – hrany nemajú orientáciu (smer)
Súvislý graf – po hranách je možné prejsť z ľubovoľného vrcholu do ľubovoľného iného vrcholu v grafe
Cyklus – taký prechod po hranách, že začneme v nejakom vrchole, prejdeme po aspoň jednej hrane a skončíme v počiatočnom vrchole

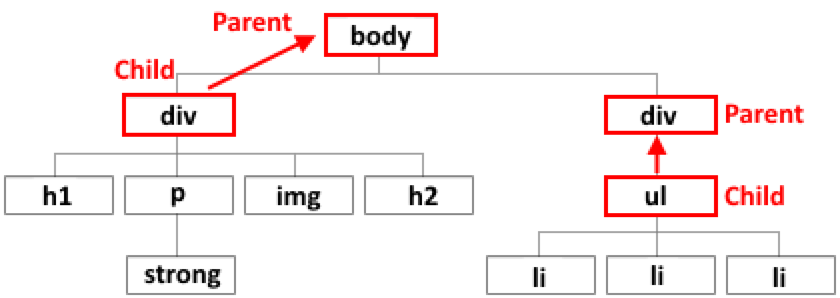
Zakoreňený strom

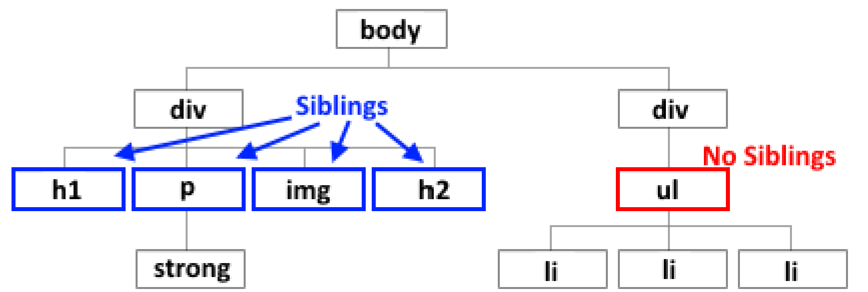
- Strom, v ktorom je význačný vrchol – **koreň** (root)
- Uvažujme vrchol **u**, ktorý leží na ceste z **koreňa** do **v**
  u nazývame **predchodca (ancestor) / rodič v**, resp. **v** je **nasledovník (descendant) / dieťa u**
- **List** – koncový vrchol (ktorý nemá nasledovníkov)
  - Ostatné vrcholy sú **vnútorné**
- Zvyčajne sa uvažuje orientácia hrán zhora dole (od koreňa k listom)



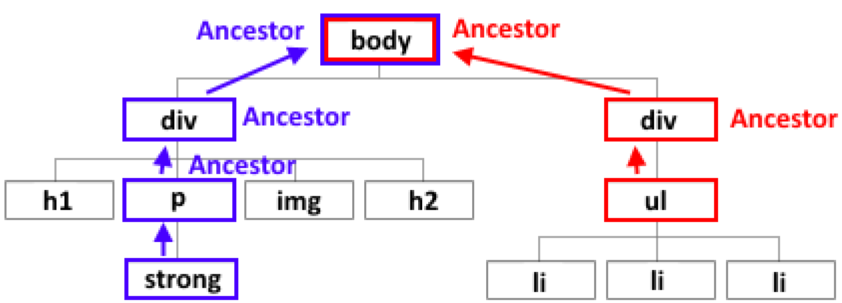
Zakoreňený strom (2) - príklad HTML

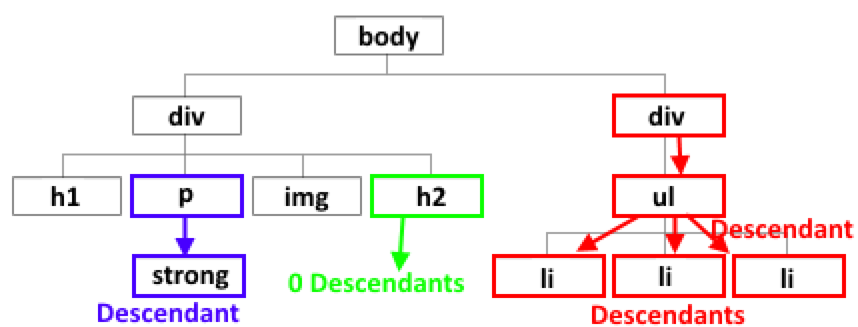
- **Rodič (parent)** – najbližší-priamy predchodca vrcholu
- **Dieťa /potomok (child)** – priamy nasledovník vrcholu
- **Súrodenci (siblings)** – vrcholy s rovnakým rodičom





- **Nasledovník /predchodca je aj nepriamy:**

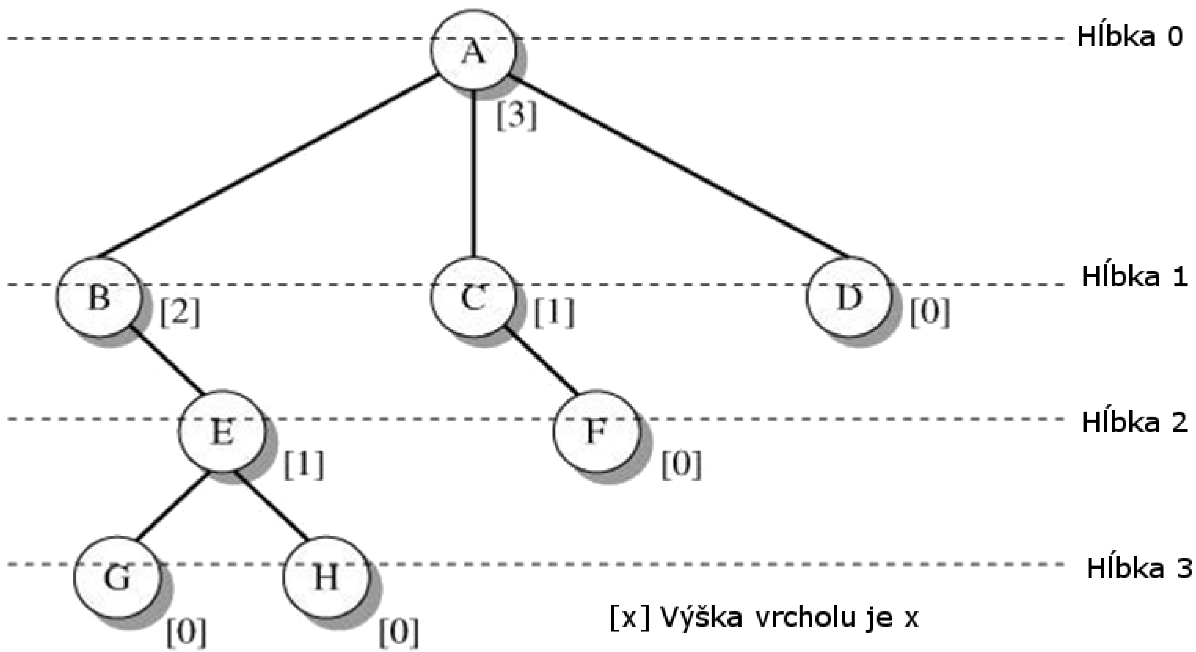




(Zakoreňený) strom – hĺbka, výška

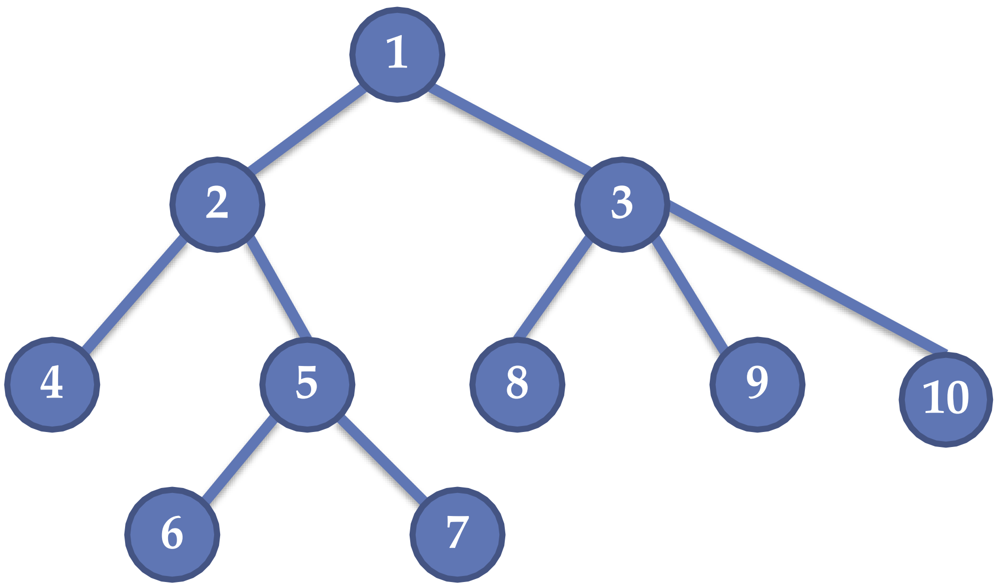
Hĺbka vrcholu – počet hrán od koreňa stromu k danému vrcholu
Výška vrcholu – dĺžka najdlhšej cesty z daného vrcholu k listu (koncovému vrcholu)

Výška stromu – výška jeho koreňa



Strom – Reprezentácia poľom

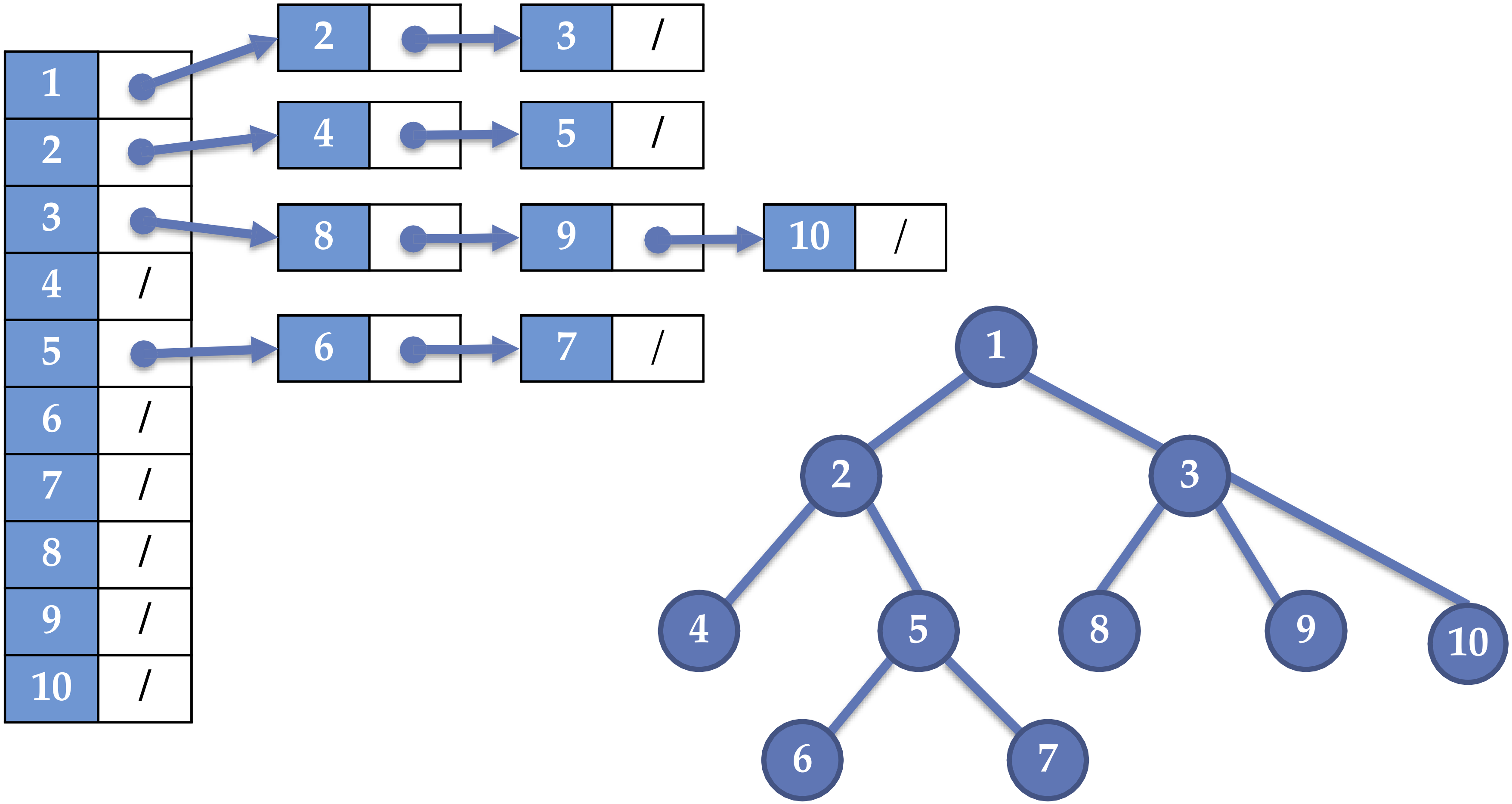
- Index do poľa = číslo vrcholu
- Hodnota prvku poľa = ukazovateľ na rodiča



pole[i]	0	1	1	2	2	5	5	3	3	3
index	1	2	3	4	5	6	7	8	9	10

Strom – Reprezentácia spájaným zoznamom

- Každý vrchol má spájaný zoznam priamych nasledovníkov



Rôzne typy stromov (terminologicky)

- Strom (**príroda**)

- Strom (**teória grafov**)
  - Súvislý neorientovaný graf bez cyklov
  - Zakoreňený strom
- Strom (**abstraktná dátová štruktúra**)
  - Reprezentácia hierarchických vzťahov
- Strom (**teória množín**)
  - Čiastočne usporiadaná množina
    (keď nie je nutné, aby sa dali porovnať všetky dvojice prvkov)

Binárny strom

- Strom, v ktorom každý vrchol má najviac **dvoch priamych nasledovníkov** (potomkovia)
- Potomkovia sa označujú ako ĽAVÝ a PRAVÝ
- Rekurzívna definícia:
  - Jeden vrchol je binárny strom a súčasne koreň.
  - Ak u je vrchol a T_1 a T_2 sú stromy s koreňmi v_1 a v_2, tak usporiadaná trojica (T_1, u, T_2) je binárny strom, ak v_1 je **ľavý potomok** koreňa u a v_2 je jeho **pravý potomok**.

Binárny strom – Operácie

- **CREATE**: vytvorenie prázdneho binárneho stromu
- **MAKE**: vytvorenie binárneho stromu z dvoch už existujúcich binárnych stromov a hodnoty
- **LCHILD**: vrátenie ľavého podstromu
- **DATA**: vrátenie hodnoty koreňa v danom binárnom strome
- **RCHILD**: vrátenie pravého podstromu
- **ISEMPTY**: test na prázdnosť

Binárny strom – Formálna špecifikácia

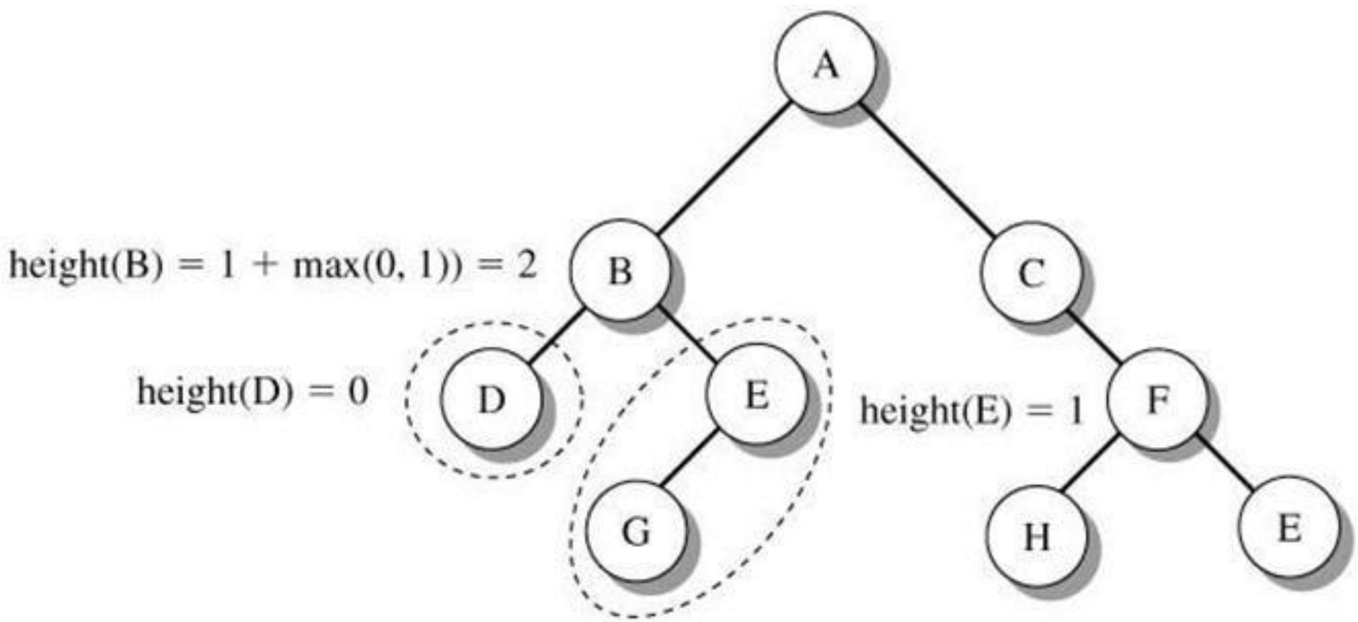
CREATE() → bintree
MAKE(bintree, item, bintree) → bintree
LCHILD(bintree) → bintree
DATA(bintree) → item
RCHILD(bintree) → bintree
ISEMPTY(bintree) → boolean

Pre všetky p,r ∈ bintree, i ∈ item platí:
ISEMPTY(CREATE) = true
ISEMPTY(MAKE(p,i,r)) = false
LCHILD(MAKE(p,i,r)) = p
LCHILD(CREATE) = error
DATA(MAKE(p,i,r) = i
DATA(CREATE) = error
RCHILD(MAKE(p,i,r)) = r
RCHILD(CREATE) = error

Binárny strom – Výpočet výšky stromu

- Výšku (height) stromu je možné vypočítať rekurzívne:

$$\text{výška}(T) = \begin{cases} -1 & \textit{ak podstrom T je prázdny} \\ 1 + \max(\text{výška}(T_L), \text{výška}(T_R)) & \textit{ak podstrom T nie je prázdny} \end{cases}$$



Binárny strom s výškou 3

Prehľadávanie binárnych stromov

Tri základné algoritmy:

- **pre-order** poradie prehľadávania:

  koreň → ľavý podstrom → pravý podstrom

- **in-order** poradie prehľadávania

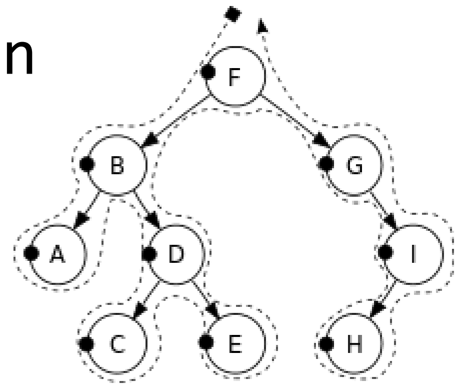
  ľavý podstrom → koreň → pravý podstrom

  post-order poradie prehľadávania:

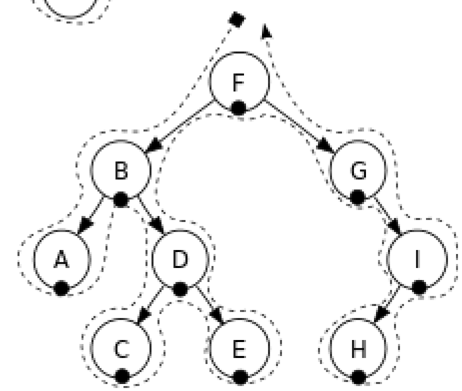
  ľavý podstrom → pravý podstrom → koreň

Prehľadávanie binárnych stromov (pseudokód)

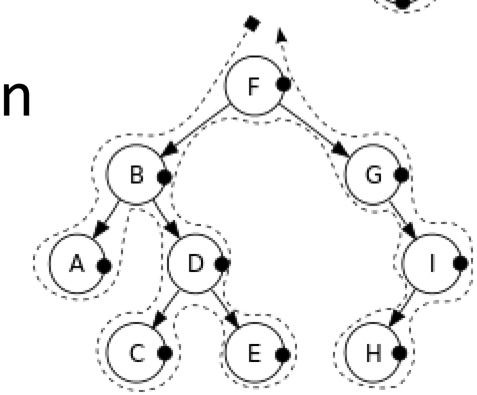
```
PREORDER(T): if T <> nil then
  OUTPUT(DATA(T))
  PREORDER(LCHILD(T))
  PREORDER(RCHILD(T))
```



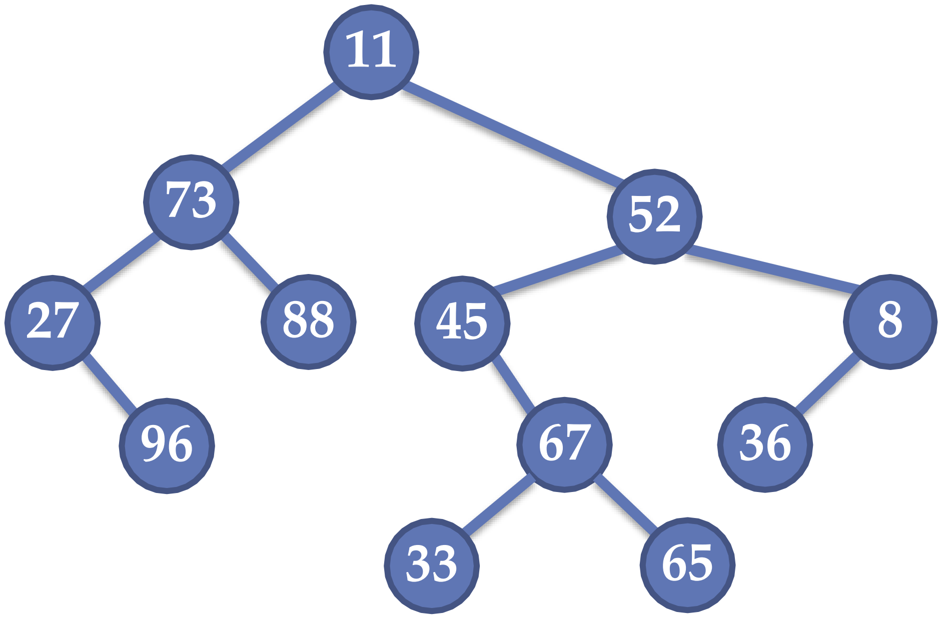

```
INORDER(T): if T <> nil then
  INORDER(LCHILD(T))
  OUTPUT(DATA(T))
  INORDER(RCHILD(T))
```




```
POSTORDER(T) if T <> nil then
  POSTORDER (LCHILD(T))
  POSTORDER (RCHILD(T))
  OUTPUT(DATA(T))
```



Prehľadávanie binárnych stromov (ukážka)



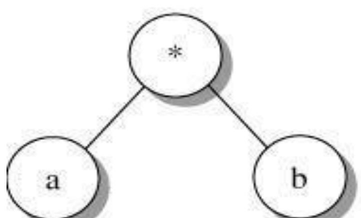
Preorder: 11,73,27,96,88,52,45,67,33,65,8,36

Inorder: 27,96,73,88,11,45,33,67,65,52,36,8

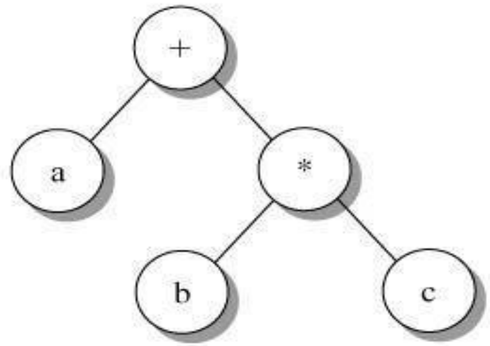
Postorder: 96,27,88,73,33,65,67,45,36,8,52,11

Reprezentácia aritmetických výrazov

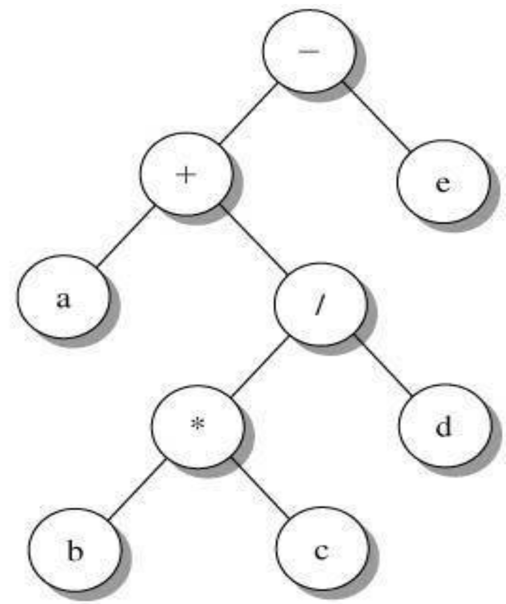
- Základné využitie binárnych stromov v informatike
- Operátor je vnútorný vrchol a jeho potomkom môže byť:
  - Podstrom predstavujúci ďalší výraz
  - Operand



(a) a*b



(b) a + b* c



(c) a + b*c/d − e

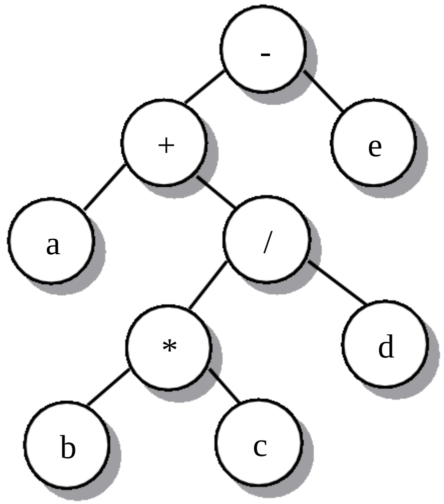
Prehľadávanie stromu aritmetického výrazu

- Pre-order prechádzanie stromu poskytne prefixový zápis výrazu
- Post-order prechádzanie stromu poskytne postfixový zápis výrazu
- In-order prechádzanie stromu poskytne infixový zápis výrazu (bez zátvoriek)

Preorder(Prefix): - + a / * b c d e

Inorder(Infix): a + b * c / d – e

Postorder(Postfix): a b c * d / + e -



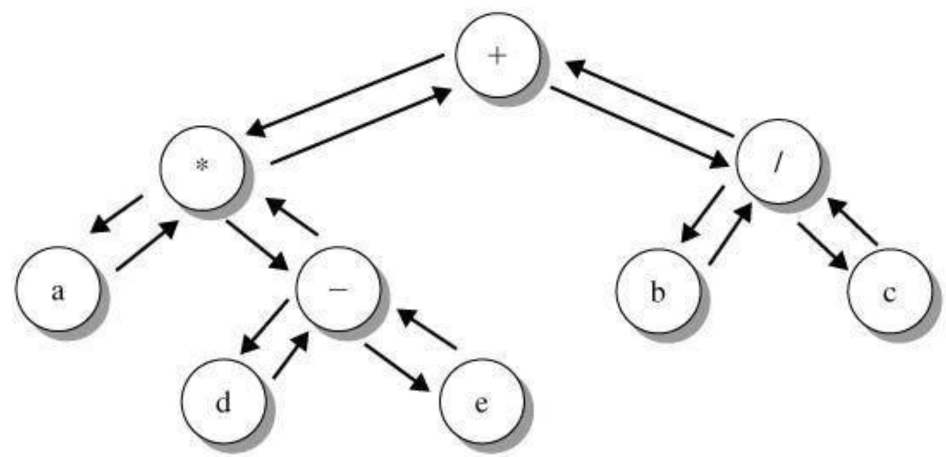
a + b*c/d -e

Eulerov ťah (Euler tour)

- Predchádzajúce spôsoby prechádzali binárny strom vždy tak, že každý vrchol navštívili iba raz.
- Uvažujme **prehľadávanie, ktoré prejde každú hranu raz** (v každom smere)
- Každý vrchol, ktorý má potomkov sa prechádza vždy tri krát:
  - pri prechode od rodiča
  - pri prechode od ľavého potomka
  - pri prechode od pravého potomka

Eulerov ťah v binárnom strome (pseudokód)

```
eulerTour(t):
if t ≠ null
  if t is a leaf node
      visit t
  else
      visit t;
      eulerTour(t.left);
      visit t;
      eulerTour(t.right);
      visit t;
```

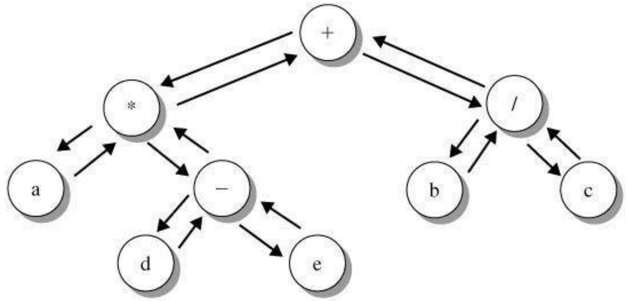


Eulerov ťah: + * a * - d - e - * + / b / c / +

Úplne uzátvorkovaný výraz

- Upravený algoritmus pre Eulerov ťah
- Vstup: matematický výraz reprezentovaný binárnym stromom
- Postup:
  - Pri prechode operandu sa vloží do výstupu operand
  - Pri prechode operátora sa vloží do výstupu:
    - Ľavá zátvorka (pri prechode od rodiča
    - Pravá zátvorka) pri prechode z pravého potomka
    - Operátor pri prechode z ľavého potomka
- Výstup takto upraveného algoritmu:

((a*(d-e))+(b/c))



Binárny vyhľadávací strom (BVS)

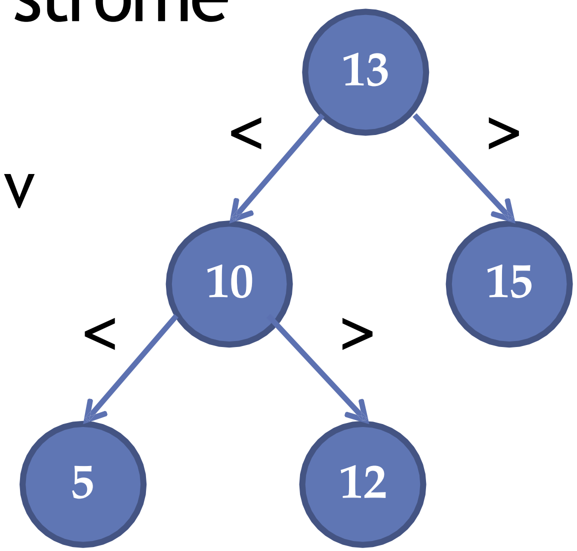
- BVS je binárny strom
- BVS môže byť prázdny
- Ak BVS nie je prázdny, tak spĺňa tieto podmienky:
  - každý prvok má kľúč a všetky kľúče sú rôzne,
  - všetky kľúče v ľavom podstrome sú menšie ako kľúč v koreni stromu
  - všetky kľúče v pravom podstrome sú väčšie ako kľúč v koreni stromu,
  - ľavý aj pravý podstrom sú tiež BVS.

Binárny vyhľadávací strom – Operácie

- Implementácia ADT Dynamická množina
  - **create** – vytvor prázdny strom
  - **insert** – vložiť/pridať prvok do stromu
  - **search** – vyhľadať prvok v strome
  - **delete** – odstrániť prvok zo stromu

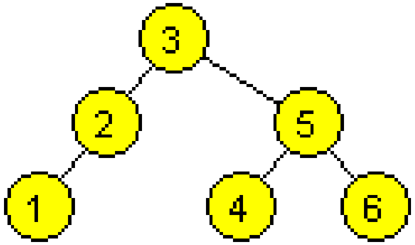
Binárny vyhľadávací strom - intuitívna implemenetácia

- **insert(x)** – vložiť prvok X do stromu
  - Najskôr sa pokúsim X vyhľadať, a potom vložím na miesto kde by mal byť (ale nebol)
- **search(x)** – vyhľadaj prvok X v strome
  1. Začnem v koreni ...vrchol v
  2. Porovnám kľúč X s hodnotou vo v
  3. Ak je rovný, našiel som.
  4. Inak, presuniem sa do príslušného potomka, nastavím ho ako nový v, choď na 2.



Binárny vyhľadávací strom – search (pseudokód)

Rekurzívna verzia

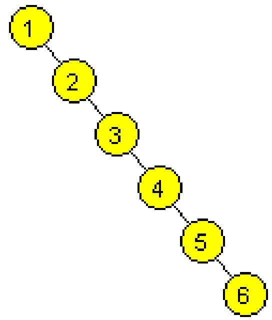


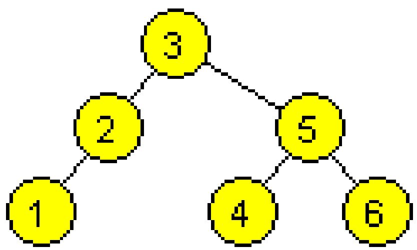

```
bintree TREE-SEARCH(T,k):
if T=nil or k=DATA(T)
      then return T
if k<DATA(T)  then return TREE-SEARCH(LCHILD(T),k)
      else return TREE-SEARCH(RCHILD(T),k)
```

Iteratívna verzia

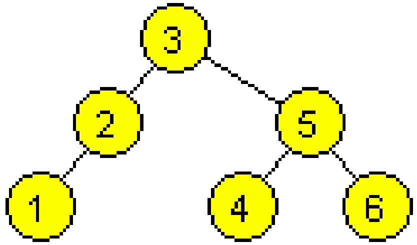
```
bintree ITERATIVE-TREE-SEARCH(T,k):
while T <>  nil and k<>DATA(T) do
      if k<DATA(T)  then T ← LCHILD(T)
                 else T ← RCHILD(T)
return T
```

Analýza zložitosti – search

- Závisí od hĺbky h – O(h)
  - Najhorší prípad O(n)
    nájdenie uzla 6
    

  - Priemerný a zároveň najlepší prípad O(log n)
    nájdenie uzla 6
    

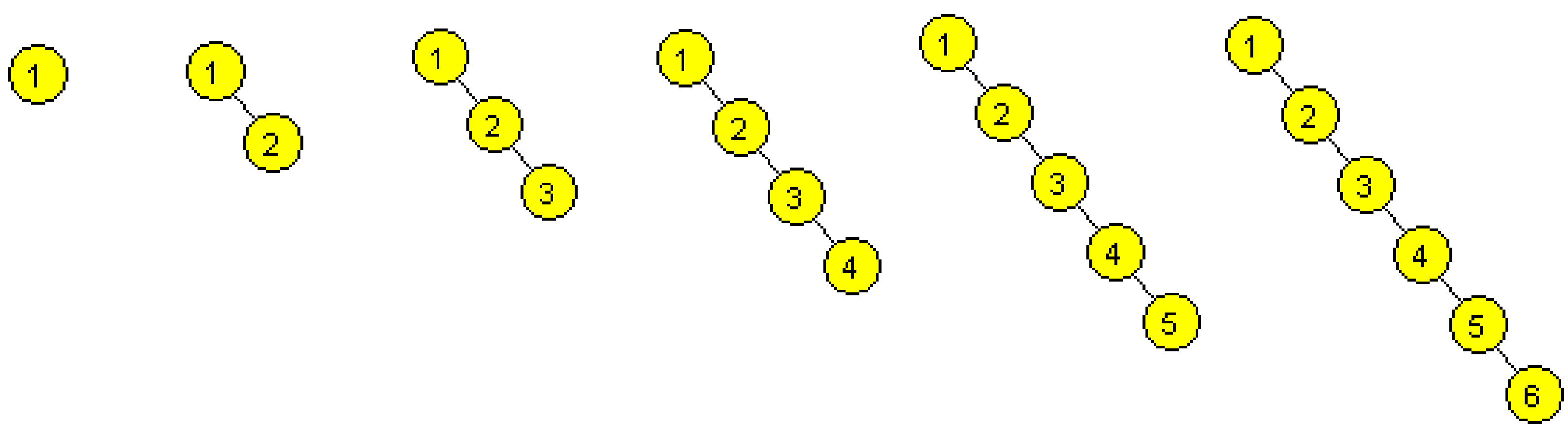
Binárny vyhľadávací strom - insert (pseudokód)




```
TREE-INSERT(T,n):
   Y ← nil; X ← ROOT(T)
   while X  <> nil do
      Y←X
      if DATA(n) <  DATA(X)  then X ← LCHILD(X)
           else X  ←RCHILD(X)
   PARENT(n) ←Y
   If Y  =  nil then ROOT(T)← n
   else if DATA(n) <  DATA(Y)  then LCHILD(Y) ← n
           else RCHILD(Y) ← n
```

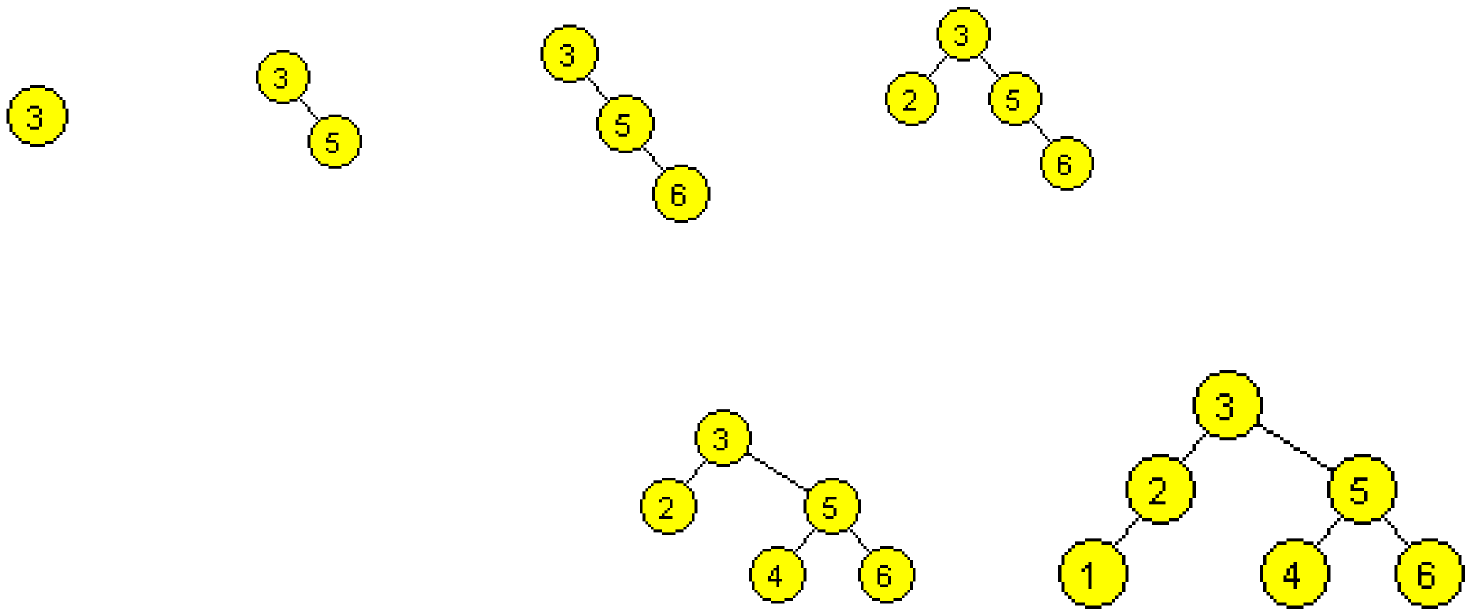
Analýza zložitosti - insert

- Musíme nájsť miesto, kde môžeme prvok vložiť – časová zložitosť závisí od hĺbky stromu – O(h)
  - Najhorší prípad O(n): zoradená postupnosť – vytvárame nevyvážený strom – rýchle zväčšovanie hĺbky stromu
    Napr. 1, 2, 3, 4, 5, 6



Analýza zložitosti - insert (2)

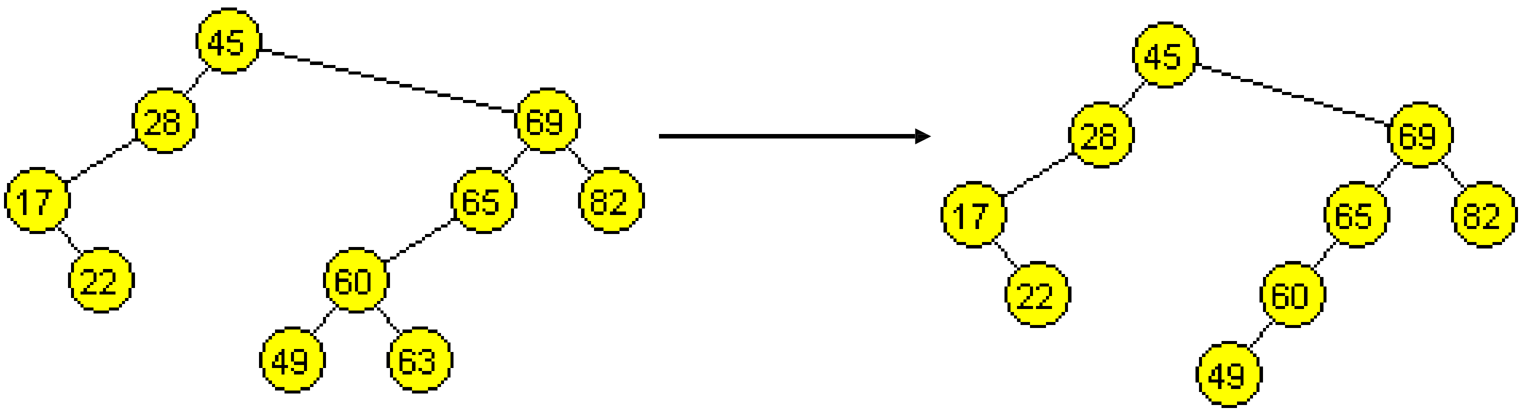
- Priemerný prípad O(log n): náhodná postupnosť – vytvárame väčšinou „dobre“ vyvážený strom – pomalé zväčšovanie hĺbky stromu
  Napr. 3, 5, 6, 2, 4, 1



Binárny vyhľadávací strom – delete

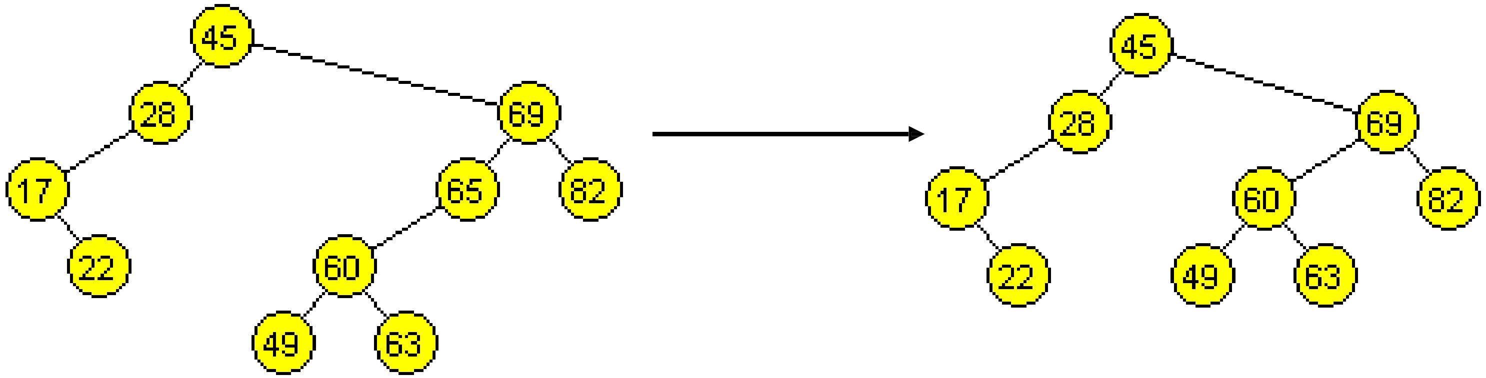
- Môžu nastať tri prípady

1. **vrchol na odstránenie nemá žiadny podstrom:** jednoduché odstránenie vrcholu, napr. odstránenie 63



Binárny vyhľadávací strom – delete (2)

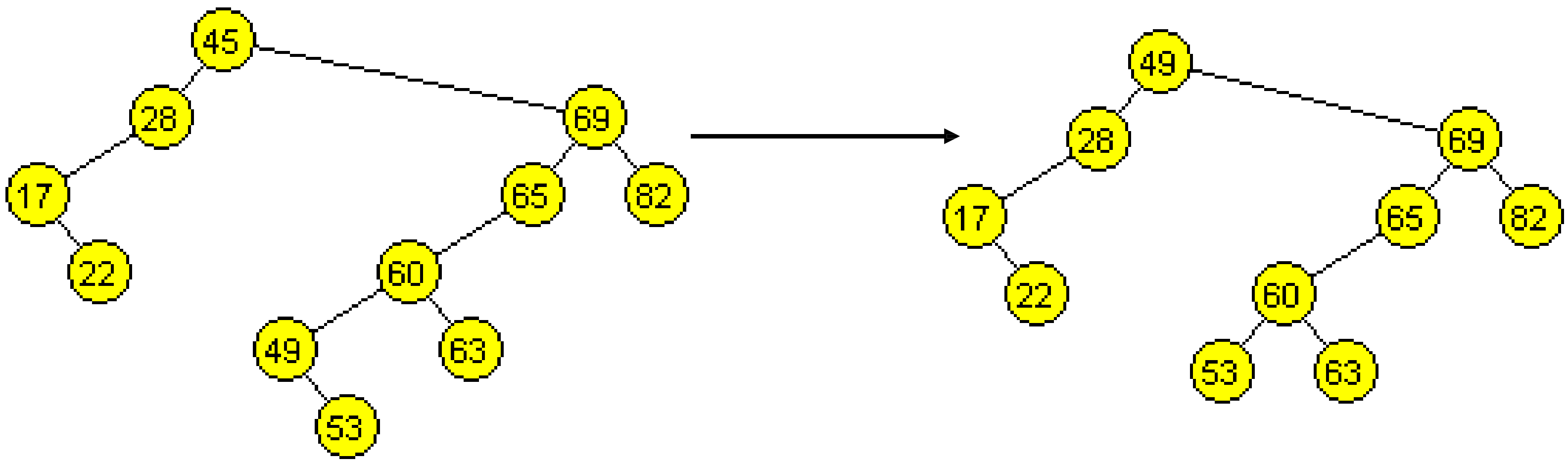
2. **vrchol na odstránenie má jeden podstrom:** odstránenie vrcholu, prepojenie koreňa jeho podstromu s jeho rodičom, napr. odstránenie 65



Binárny vyhľadávací strom – delete (3)

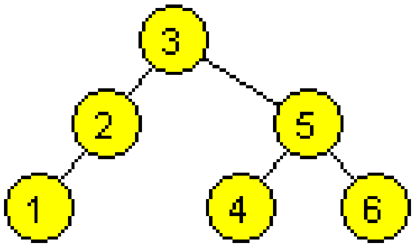
3. **vrchol V na odstránenie má dva podstromy:**
   - nájsť za neho náhradu Y (najmenší väčší prvok – successor: najľavejší z jeho pravého podstromu),
- skopírovať kľúč z Y do V, odstrániť zo stromu vrchol Y a (ak existuje) prepojiť jediné dieťa Y s rodičom Y
   - (náhrada môže byť aj najväčší menší prvok – predecessor)

Napr. odstránenie 45 (náhrada bude 49)



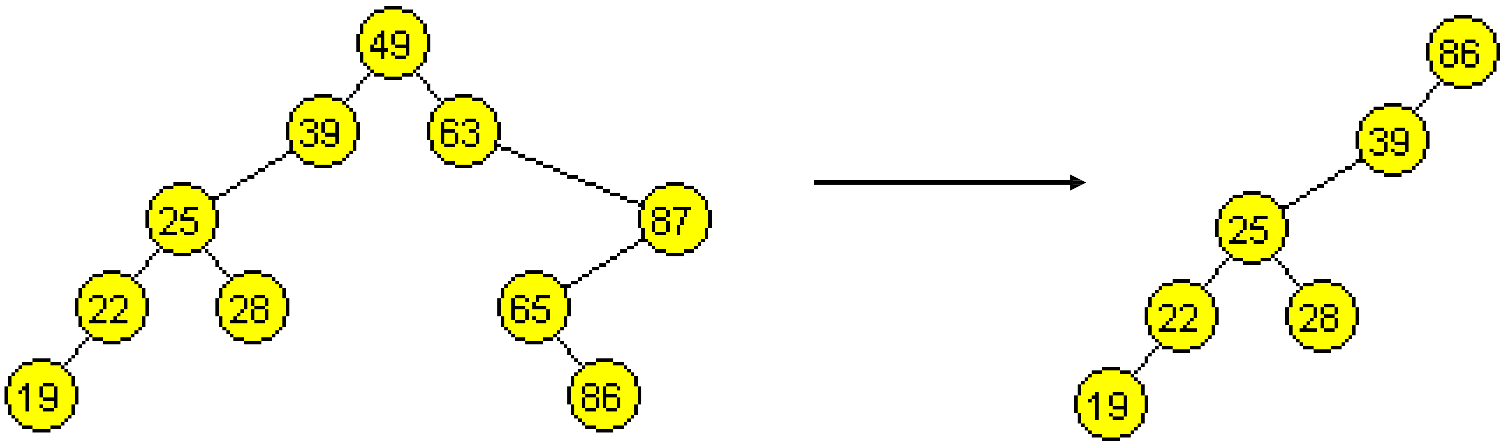
Binárny vyhľadávací strom - delete (pseudokód)

```
bintree TREE-DELETE(T,v):
    if LCHILD(v) = nil or RCHILD(v) = nil then Y ← v
        else Y ← TREE-SUCCESSOR(v)
    if LCHILD(Y) <> nil then X ← LCHILD(Y)
        else X ← RCHILD(Y)
    if X <> nil
        then PARENT(X) ← PARENT(Y)
    if PARENT(Y) = nil then ROOT(T) ← X
        else if Y = LCHILD(PARENT(Y))
                then LCHILD(PARENT(Y)) ← X
                else RCHILD(PARENT(Y)) ← X
    if Y <> v
        then DATA(v) ← DATA(Y)
return Y
```



Analýza zložitosti – delete

- Musíme nájsť vrchol, ktorý chceme odstrániť a vrchol, ktorý sa stane náhradou – časová zložitosť závisí od hĺbky stromu – O(h)
- Odstraňovanie vrcholov spôsobuje nevyváženosť stromu, pretože vždy vyberáme ako náhradu nasledovníka – počet v pravom podstrome sa znižuje, počet v ľavom podstrome zostáva rovnaký
- preto najhorší prípad má zložitosť O(n), ináč v priemere je to O(log n)
- Napr. po odstránení vrcholov 49, 63, 65, 87, 65



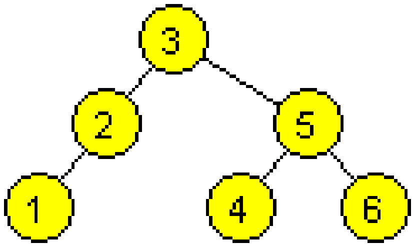
Binárny vyhľadávací strom – Ďalšie operácie

- BVS je NIELEN implementácia ADT Dynamická množina
- Navyše máme binárnu reláciu usporiadania < kľúčov
- Ďalšie operácie BVS:
  - **min / max** – vyhľadať najmenší / najväčší prvok
  - **successor** – vyhľadať najbližší väčší prvok
  - **predecessor** – vyhľadať najbližší menší prvok

Binárny vyhľadávací strom - min / max (pseudokód)

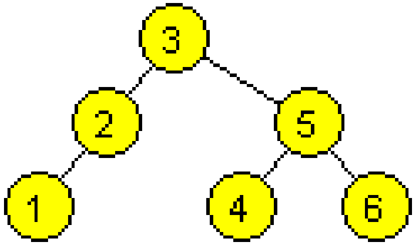
```
bintree TREE-MINIMUM(T):
   while LCHILD(T) <> nil do
            T ← LCHILD(T)
   return T
```

```
bintree TREE-MAXIMUM(T):
   while RCHILD(T) <> nil do
            T ← RCHILD(T)
   return T
```



Binárny vyhľadávací strom - successor (pseudokód)

```
bintree TREE-SUCCESSOR(T):
   if RCHILD(T) <> nil
        then return TREE-MINIMUM(RCHILD(T))
   S ← PARENT(T)
   while S <> nil and T = RCHILD(S) do
        T←S
        S ← PARENT(T)
   return S
```



Implementácia v jazyku C (ukážka)

- Dve štruktúry: strom, prvok stromu (vrchol)

```c
struct Strom
{
    int pocet;
    struct Vrchol *koren;
};

struct Vrchol
{
    int hodnota;
    struct Vrchol *lavy, *pravy;
};


struct Strom *strom_vytvor()
{
    struct Strom *s = (struct Strom *)malloc(sizeof(struct Strom));
    s->pocet = 0;
    s->koren = NULL;
    return s;
}
```

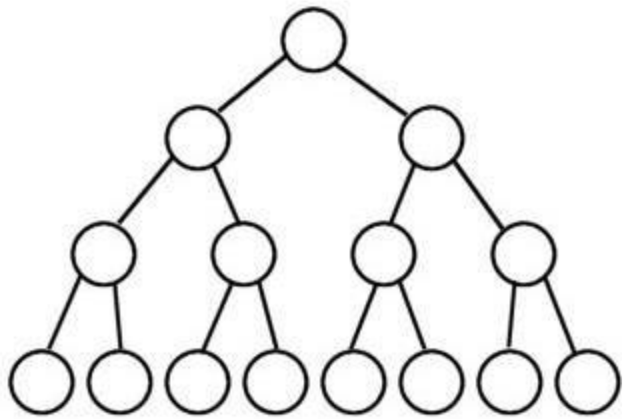
Implementácia v jazyku C (testovanie)

- Ako si rýchlo otestujem moju implementáciu?
  - Vložím tam náhodné čísla a vypíšem ich (napr. INORDER prehľadávanie)

```c
int main(void)
{
    struct Strom *s = strom_vytvor();

    int i;
    for (i = 0; i < 50; i++)
        strom_pridaj(s, rand()%1000);

    strom_vypis(s);

    return 0;
}
```

Efektivita vykonania operácií BVS

- Zložitosť operácií nad BVS lineárne závisí od hĺbky stromu – O(h)
- Operácie pracujú lepšie keď je strom „vyvážený“
- Najlepšie, aby bol takýto (tzv. úplný binárny strom)



Vyvážené vyhľadávacie stromy

- Prioritný rad/front (halda) nie je implementácia všeobecnej dynamickej množiny
- Ako vylepšiť všeobecné vyhľadávacie stromy?
  - Obmedziť ich štruktúru, aby sme mohli o nej prehlásiť nejaké vlastnosti – napr. že bude vždy nízka výška stromu
  - Z týchto garancií (na veľkosť výšky) vyplynú efektívne zložitosti operácií nad takýmito stromami
- Na získanie najlepšej zložitosti O(log n) musíme zabezpečiť, aby strom po vykonaní operácií (insert, delete) zostal vyvážený – použitie samovyvažovacích stromov ako sú AVL stromy alebo červeno-čierne stromy, ktoré automaticky menia svoju štruktúru tak, aby po týchto operáciách bol rozdiel hĺbok ľavého a pravého podstromu „malý“

Vyvážené vyhľadávacie stromy - Prehľad

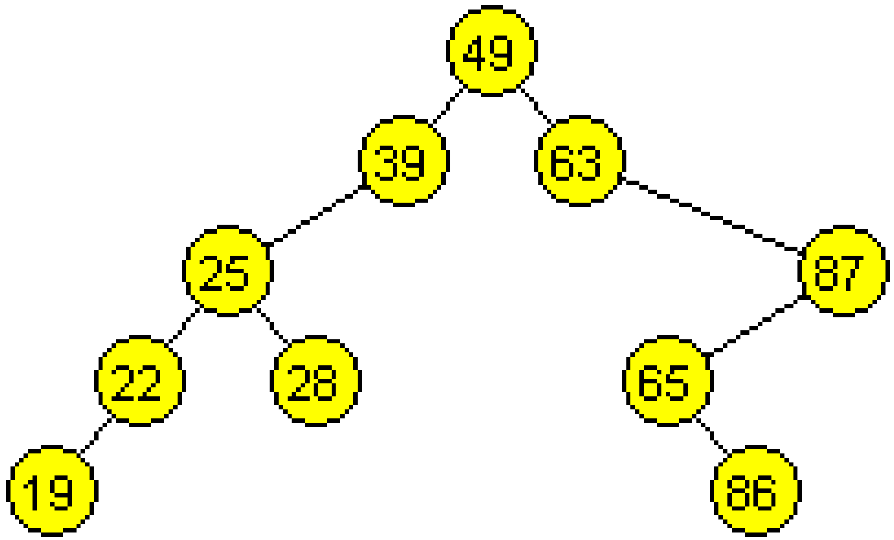
- Základné vyvažovanie – podrobne
  - AVL strom
  - Splay strom
- Ostatné: definícia, insert, vlastnosti
  - B stromy
  - (a,b) stromy: 2,3 a 2,3,4 stromy
  - Červeno-Čierne (Red-Black) stormy
  - Váhovo vyvážené
- Optimálne binárne vyhľadávacie stromy
- A ďalšie:
  - Trie – dynamická množina reťazcov
  - Radixový strom, lano, …

Opakovanie – Základné algoritmy

- Čím viac informácií o vstupnej postupnosti mám k dispozícii, tým rýchlejší algoritmus dokážem vytvoriť
  - Lineárne vyhľadávanie: O(n)
  - Binárne vyhľadávanie: O(log n)
  - Interpolačné vyhľadávanie: O(log log n)
- Binárne vyhľadávacie stromy
  - Priemerný prípad: O(log n)
  - **Najhorší prípad: O(n)**
- Niektoré špecializované typy vyhľadávania
  - Prioritný front (vyhľadávam len najprioritnejší prvok): insert /removeMax :O(log n)

Nová operácia: nájdi k-ty prvok v strome

- Prvé riešenie:
  Využiť in-order usporiadanie, zobrať k-ty prvok
  - Zložitosť O(k)
- Napr. k=5



- In-order: 19, 22, 25, 28, **39**, 49, 63, 65, 86, 87

Nová operácia: nájdi k-ty prvok v strome

- Lepšie riešenie: využiť princíp QuickSelect algoritmu **pri porovnaní vo vrchole pokračovať len v podstrome, v ktorom sa k-ty prvok nachádza**
- Potrebujeme pre každý vrchol x poznať: **počet prvkov v strome s koreňom x**
- Implementácia ako **rozšírenie štandardnej dátovej štruktúry BVS**, rozšírime údaje pre vrchol:
  - ľavý, pravý, rodič, **počet** (prvkov v podstrome) tzv. **váha**
  - rekurzívna definícia váhy
    váha(v) = váha(ľavýPodstrom(v)) + váha(pravýPodstrom(v)) + 1
- Hodnoty **váhy** vo vrcholoch upravujeme pri každej operácií ktorá mení štruktúru stromu: zložitosť O(h), kde h je výška stromu

Order statistic tree

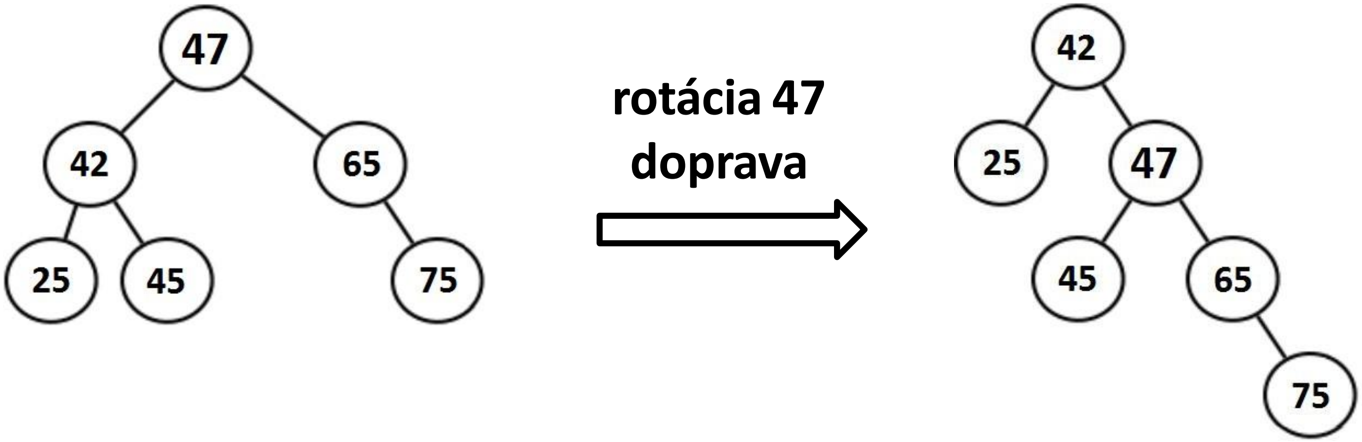
- Rozšírenie BVS stromu
- Pre každý vrchol BVS si navyše pamätáme **počet prvkov v podstrome vrcholu** tzv. váhu
- Hodnoty váhy vo vrcholoch upravujeme pri každej operácií ktorá mení štruktúru stromu (insert, delete)
- Rozšírený strom podporuje navyše operácie:
  - **select(k)** – nájdi k-ty najmenší prvok v množine
  - **rank(x)** – nájdi poradie prvku x v usporiadanej postupnosti prvkov stromu
- Zložitosť operácií O(h), kde h je výška stromu

Ako vylepšiť všeobecné vyhľadávacie stromy?

- Obmedziť ich štruktúru, aby sme mohli o nej prehlásiť nejaké vlastnosti – napr. že bude vždy nízka výška stromu
- Z týchto garancií (na veľkosť výšky) vyplynú efektívne zložitosti operácií nad takýmito stromami
- Na získanie optimálnej zložitosti O(log n) musíme zabezpečiť, aby strom po vykonaní každej operácie zostal vyvážený
- Ako zabezpečiť vyváženie stromu?
    - Hodnoty v strome meniť nemôžeme :)
    - Musíme nejako **upravovať štruktúru** stromu

Rotácia stromu – doprava

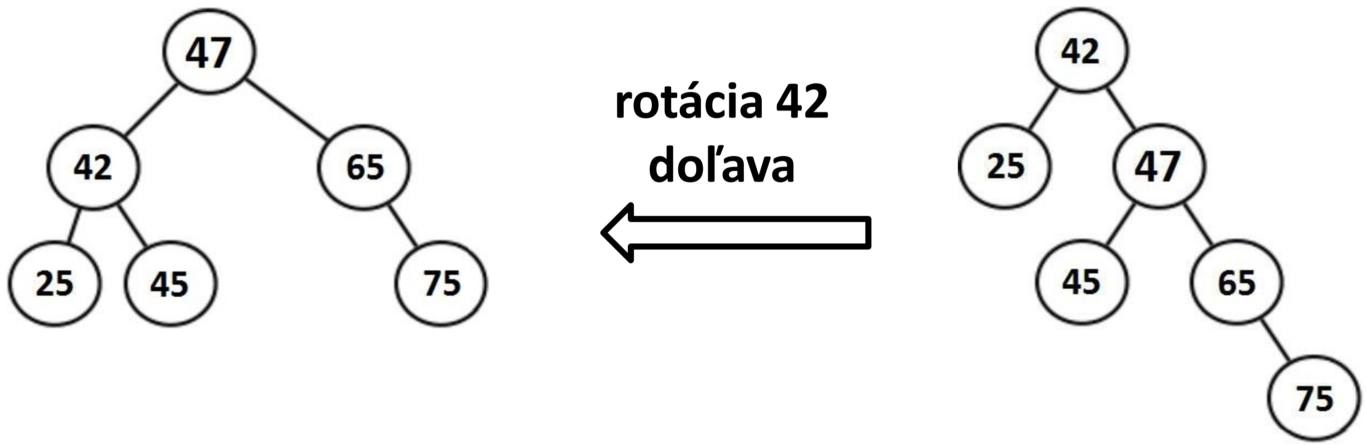
- Operácia, ktorá zmení štruktúru ale zachová usporiadanie
- Zmena tvaru stromu – zmena výšky stromu
- **Rotácia doprava:**
  ľavé dieťa sa presunie (doprava hore) na miesto rodiča



- Zmena hĺbky 25(-1), 42(-1), 47(+1), 65(+1), 75(+1)
- In-order poradie (oba stromy): 25, 42, 45, 47, 65, 75

Rotácia stromu – doľava

- Operácia, ktorá zmení štruktúru ale zachová usporiadanie
- Zmena tvaru stromu – zmena výšky stromu
- **Rotácia doľava:**
  pravé dieťa sa presunie (doľava hore) na miesto rodiča



- Zmena hĺbky 25(+1), 42(+1), 47(-1), 65(-1), 75(-1)
- In-order poradie (oba stromy): 25, 42, 45, 47, 65, 75