Stromy

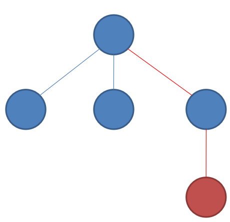
ADT strom
binárny strom
binárny vyhľadávací strom

Strom – definícia

1. Jediný vrchol je strom – tento vrchol je zároveň koreň tohto stromu
2. Nech **V** je vrchol a **S1**,**S2**..**Sn** sú stromy s koreňmi **V1**,**V2**..**Vn**. Nový strom môžeme zostrojiť tak, že vrchol V urobíme **PREDCHODCOM** vrcholov **V1**,**V2**..**Vn**. V tomto novom strome je V koreň a **S1**,**S2**..**Sn** sú jeho podstromy. Vrcholy **V1**,**V2**..**Vn** sú **NASLEDOVNÍCI** vrcholu V

Strom – základné definície

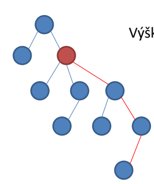
- Hĺbka vrcholu – počet hrán od koreňa stromu k danému vrcholu



hĺbka = 2

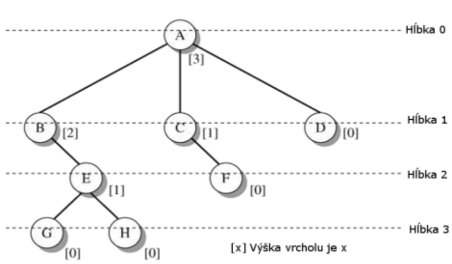
Strom - základné definície

- Výška vrcholu – najdlhšia cesta z vrcholu k ľubovoľnému koncovému vrcholu
- Výška stromu – výška jeho koreňa



Výška vrcholu = 3

Strom – základné definície



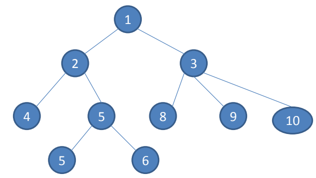
Strom – formálna špecifikácia

- Operácie:
  - EMPTY : vytvorenie prázdneho stromu
  - EMPTYn : nekonečná rodina operácií.
    - EMPTYi(a, S1,S2..Si) vytvorí nový vrchol V s hodnotou a, ktorý má i nasledovníkov – sú to korene stromov S1..Si
  - KOREŇ : Nájdenie koreňa stromu
  - PREDCHODCA : Nájdenie predchodcu daného vrcholu

Strom – formálna špecifikácia

– LNASLEDOVNIK : Nájdenie najľavejšieho nasledovníka
– PSUSED : Nájdenie vrcholu, ktorý má rovnakého predchodcu ale v usporiadaní stromu je vpravo za daným vrcholom.
– HOD : Získanie Ohodnotenia vrcholu

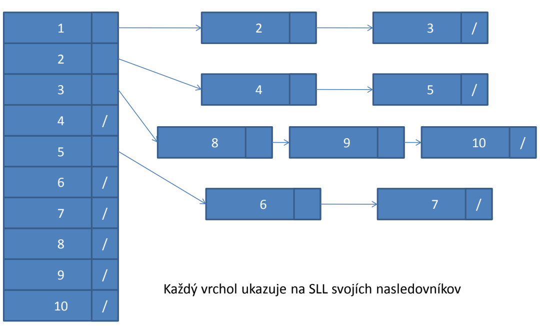
Strom – reprezentácia poľom



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

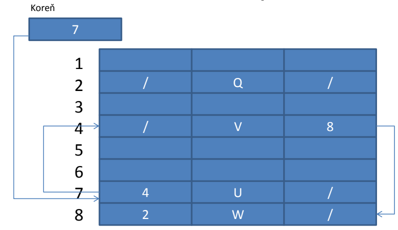
Index pola = hodnota vrcholu
Hodnota prvku poľa = ukazovateľ na rodiča

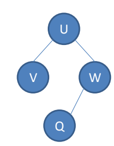
Strom – reprezentácia pomocou ZVP



Každý vrchol ukazuje na SLL svojích nasledovníkov

Strom – reprezentácia





Každý prvok obsahuje ukazovateľ na ľavého nasledovníka a pravého suseda

Binárny strom

- Strom, ktorý má najviac dvoch priamych nasledovníkov – potomkov.
- Potomkovia sa označujú ako ĽAVÝ a PRAVÝ nasledovník
- Jedno s bežných využití binárneho stromu je binárny vyhľadávací strom

Binárny strom - definície

- Jediný 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**.
- **List** - vrchol bez potomkov.
- **Úplný binárny strom** - binárny strom, v ktorom každý nelistový vrchol má práve dvoch potomkov.

Binárny strom

Operácie nad binárnym stromom:

- 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

13

Binárny strom – formálna špecifikácia

CREATE() → btree

MAKE(item,btree,item) → btree

LCHILD(btree) → btree

DATA(btree) → item

RCHILD(btree) → btree

ISEMPTY(btree) → boolean

14

Binárny strom – formálna špecifikácia

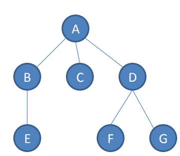
Pre všetky p,r ∈ btree, i ∈ item platí:

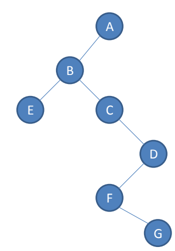
- ISEMPTY(CREATE) = true
- ISEMPTY(MAKE(p,i,r)) = false
- LCHILD(MAKE(p,i,r)) = p
- LCHILD(CREATE) = error
- DATA(MAKE(p,i,r) = i
- DATA(CREATE) = error
- RCHILD(MAKE(p,i,r)) = r
- RCHILD(CREATE) = error

15

Reprezentácia stromu pomocou binárneho stromu

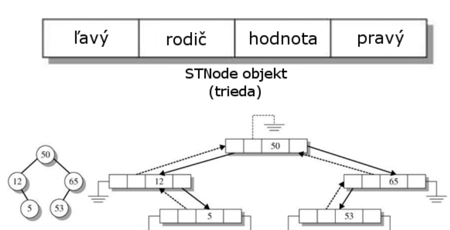
- LCHILD = ľavý nasledovník daného vrcholu
- RCHILD = pravý sused daného vrcholu





16

Binárny strom – implementácia



Binárny vyhľadávací strom

STNode reprezentácia binárneho vyhľadávacieho stromu s použitím ukazovateľov na rodiča.

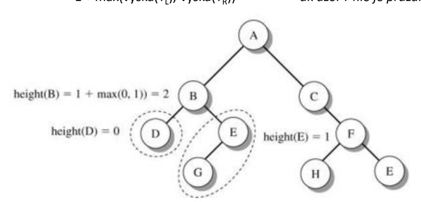
Ford, W., Topp, W.: *Data Structures with Java.* Pearson Prentice Hall, 2004. ISBN 0130477249, 9780130477248.

17

Výpočet výšky stromu

- Výšku (height) stromu je možné vypočítať rekurzívne:

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



Binárny strom s výškou 3

18

Prehľadávanie binárnych stromov

Tri základné algoritmy:

- ***preorder*** - poradie prehľadávania:
  koreň - ľavý podstrom - pravý podstrom
- ***inorder*** - poradie prahľadávania:
  ľavý podstrom - koreň - pravý podstrom
- ***postorder*** - poradie prahľadávania:
  ľavý podstrom - pravý podstrom - koreň.

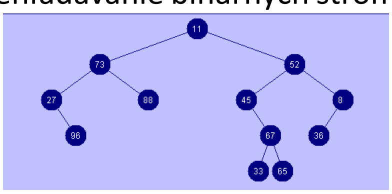
19

Prehľadávanie binárnych stromov

```
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))
```

20

Prehľadávanie binárnych stromov



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

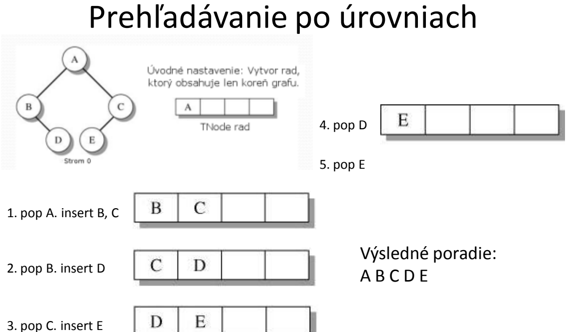
21

Prehľadávanie po úrovniach

- Začína sa koreňom, postupne sa prechádzajú vrcholy po úrovni (najskôr všetky 1. úrovne, potom 2. úrovne, atď.)
- Pri prehľadávaní je možné využiť front ako pomocnú štruktúru:
  - 1. krok: do frontu sa vloží koreň
  - n. krok: vyberie sa prvok z frontu, zistia sa jeho potomkovia a pokiaľ existujú tak sa vložia do frontu. Pokračuje sa, kým nie je front prázdny.

22

Prehľadávanie po úrovniach



Výsledné poradie:
A B C D E

pop je kombinácia front a delete

23

Binárne vyhľadávacie stromy (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.

24

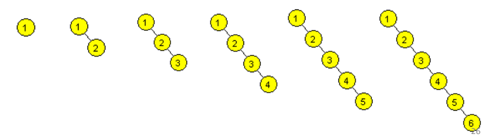
BVS – insert (implementácia)

```
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
```


BVS – insert (zložitosť)

- Musíme nájsť miesto, kde môžeme prvok vložiť – časová zložitosť závisí od hĺbky stromu
  - Najhorší prípad O(n)
    - Na vstupe je zoradená postupnosť – vytvárame nevyvážený strom –> rýchle zväčšovanie hĺbky stromu

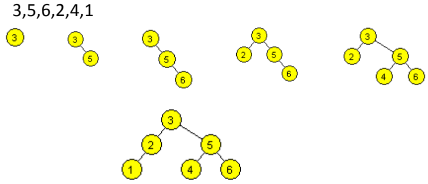
      1,3,4,5,6



BVS – insert (zložitosť)

- Priemerný prípad O(log n)
  - Na vstupe náhodná postupnosť – vytvárame väčšinou „dobre“ vyvážený strom –> pomalé zväčšovanie hĺbky stromu

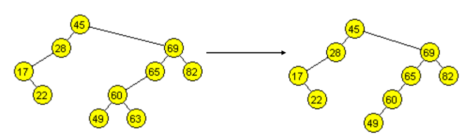
    3,5,6,2,4,1



BVS - DELETE

- Rozloženie algoritmu na tri prípady
  - uzol na odstránenie nemá žiadny podstrom
    - jednoduché odstránenie uzla

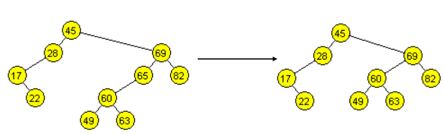
Odstránenie uzla 63



BVS - DELETE

- uzol na odstránenie má jeden podstrom
  - odstránenie uzla, prepojenie koreňa jeho podstromu s jeho rodičom

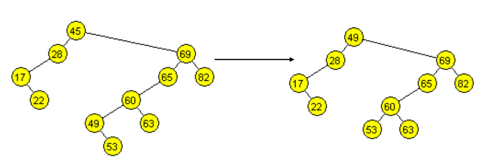
Odstránenie uzla 65



BVS - DELETE

- uzol na odstránenie má dva podstromy
  - nájsť za neho náhradu, skopírovať kľúč z náhr. uzla, odstrániť náhr. uzol, prepojiť koreň podstromu náhr. s rodičom náhrady
  - náhradou je jeho nasledovník, t.j. najmenší (najľavejší) prvok z jeho pravého podstromu –> nasledovník má pravý alebo žiadny podstrom (náhradou môže byť aj jeho predchodca, t.j. najväčší prvok z jeho ľavého podstromu)

Odstránenie uzla 45 – nahradenie 49



BVS – delete (implementácia)

```
btree TREE-DELETE(T,n)
    if LCHILD(n) = nil or RCHILD(n) = nil
        then Y ← n
        else Y ← TREE-SUCCESSOR(n)
    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 <> n
        then DATA(n) ← DATA(Y)
return Y
```

31

BVS - Nájdenie nasledovníka

```
btree 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
```

32

BVS - Nájdenie minima, resp. maxima

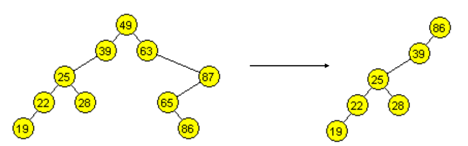
```
btree TREE-MINIMUM(T)
    while LCHILD(T) <> nil
        do
                T ← LCHILD(T)
    return T

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

33

BVS – delete (zložitosť)

- Musíme nájsť uzol, ktorý chceme odstrániť a uzol, ktorý sa stane náhradou – časová zložitosť závisí od hĺbky stromu
- Odstraňovanie uzlov spôsobuje nevyváženosť stromu, pretože vždy vyberáme ako náhradu nasledovníka –> pravý podstrom sa redukuje, ľavý ostáva
  - preto najhorší prípad má zložitosť O(n), ináč v priemere je to O(log n)

Po odstránení uzlov 49,63,65,87,65



34

BVS – search (implementácia)

Rekurzívna verzia:

```
    btree 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)
```

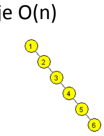
Iteratívna verzia:

```
    btree 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
```

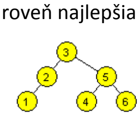
35

BVS – search (zložitosť)

- Závisí od hĺbky, resp. úplnosti stromu
  - najhoršia zložitosť je O(n)
    - nájdenie uzla 6



  - priemerná a zároveň najlepšia zložitosť je O(log n)
    - nájdenie uzla



36

BVS – výpis obsahu

- Inorder – usporiadaný výpis obsahu BVS
- Časová zložitosť pre preorder, inorder, postorder je O(n)

37

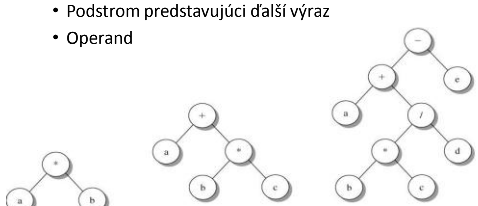
BVS – zložitosť

- Operácie search, delete, insert majú najhoršiu časovú zložitosť O(n)
- Na získanie najlepšej zložitosti O(log n) musíme zabezpečiť, že strom po týchto operáciach zostane úplný (vyvážený) –> použitie samo vyvažovacích stromov ako sú AVL stromy alebo červeno-čierne stromy, ktoré automaticky menia svoje rozloženie tak, aby po týchto operáciách bol rozdiel hĺbok ľavého a pravého podstromu nanajvýš 1

38

Využitie binárnych stromov

- Reprezentácia aritmetických výrazov
  - Operátor je vnútorný uzol a jeho potomkom môže byť
    - Podstrom predstavujúci ďalší výraz
    - Operand



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

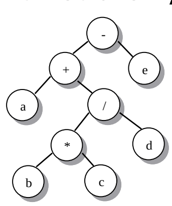
39

Aritmetické výrazy

- Preorder prechádzanie stromu poskytne prefixový zápis výrazu
- Postorder prechádzanie stromu poskytne postfixový zápis výrazu
- Inorder prechádzanie stromu poskytne infixový zápis výrazu (bez zátvoriek)

40

Aritmetické výrazy



a + b*c/d - e

```
Preorder(Prefix):    - + a / * b c d e
Inorder(Infix):      a + b * c / d – e
Postorder(Postfix):  a b c * d / + e -
```

41

Eulerovo prechádzanie stromu

- Predchádzajúce spôsoby prechádzali strom vždy tak, že každý vrchol navštívili iba raz.
- V niektorých prípadoch potrebujeme univerzálny spôsob prechádzania, ktorý by bol schopný prechádzať jednotlivé uzly aj viackrat. Riešením je Eulerovo prechádzanie stromu

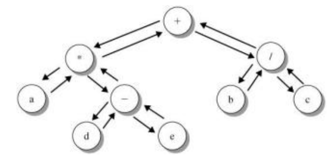
42

Eulerovo prechádzanie stromu

- Každý uzol, ktorý má potomkov sa prechádza vždy 3 krát
  - Pri prechode od rodiča
  - Pri prechode od ľavého potomka
  - Pri prechode od pravého potomka

Eulerovo prechádzanie stromu

Eulerovo prechádzanie: + * a * - d - e - * + / b / c / +



Eulerovo prechádzanie stromu

```
eulerTour(TNode t)
    if t ≠ null
     if t is a leaf node
           visit t
     else
           visit t              // on the left
           eulerTour(t.left);
           visit t;             // from below
           eulerTour(t.right);
           visit t;             // on the right
```


Eulerovo prechádzanie stromu

- Pomocou upraveného algoritmu je možné vygenerovať úplne ozátvorkovaný výraz zo stromu, ktorý reprezentuje matematický výraz
  - Pri prechode operandu sa vloží do výstupu operand
  - Pri prechode operátora sa vloží do výstupu
    - (pri prechode od rodiča
    -) pri prechode z pravého potomka
    - Operátor pri prechode z ľavého potomka

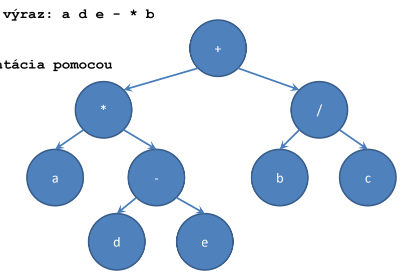
Eulerovo prechádzanie stromu

```
public static <T> String fullParen(TNode<Character> t) {
   String s = "";

   if (t != null) {
      if (t.left == null && t.right == null)
         s += t.nodeValue;          // visit a leaf node
      else {
         s += "(";                  // visit on left
         s += fullParen(t.left);
         s += t.nodeValue;          // visit from below
         s += fullParen(t.right);
         s += ")";                  // visit on right
      }
   }
   return s;
}
```


Eulerovo prechádzanie stromu

postfix výraz: a d e - * b c / +

Reprezentácia pomocou stromu:



Výstup eulerového prechodu: ((a*(d-e))+(b/c))