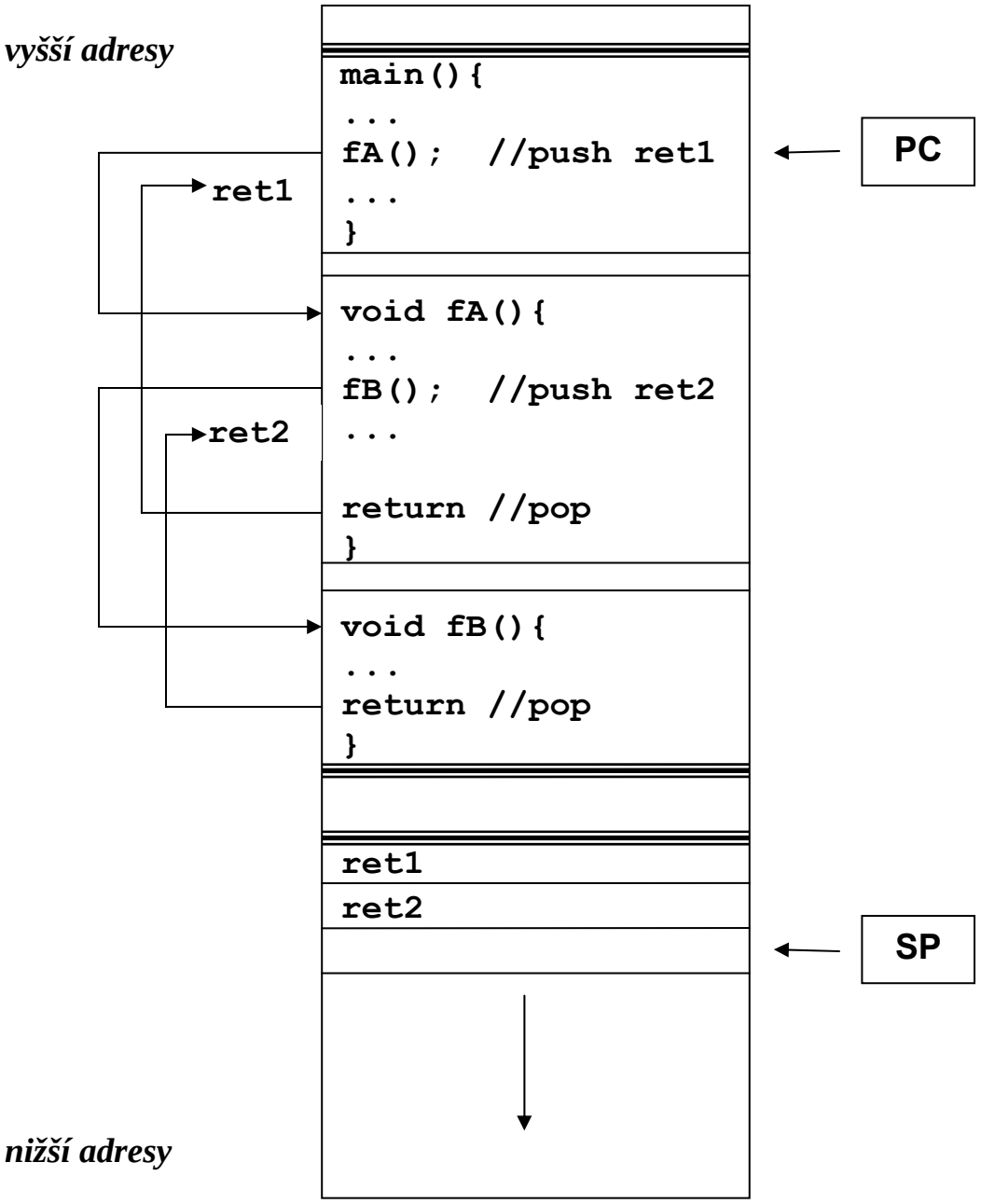
Rekurze a zásobník

Jak se „vypočítá“ rekurzivní program?

volání metody



Rekurzivní volání metody

- volání rekurzivní metody z metody **`main()`**
- vstup do metody
- nula nebo více rekurzivních volání
- dosažení základního případu a následné výstupy z rekurzivních volání až po návrat do metody **`main()`**

```
class Faktorial {
  public static void main (String[] args) {
    int n = Integer.parseInt(args[0]);
    System.out.println(n+"! = "+f(n)); // volání
    // rekurzivní metody f, uložení do zásobníku
    // parametru n a adresy pro návrat
}

  static int f(int n) {
    if (n > 1) // vstup, rozhodnutí nastal-li
               // základní případ
      return n * f(--n); // rekurzivní volání,
               // do zásobníku vložíme parametr
               // n-1 a adresu násobení
    else
      return 1;// jsme na základním případu a
               // vykonáme výstup, přičemž podle
               // adresy v záznamu na vrcholu
               // zásobníku pokračujeme násobením
               // nebo návratem do metody main()
  }
}
```

Eliminace rekurze uživatelským zásobníkem

Aktivační záznam

`par -` parametr pro faktoriál
`segmentKodu -` indikace pokračování výpočtu s hodnotami
návrat - skončení rekurze
násobení - výpočet faktoriálu při výstupu z rekurzivního volání

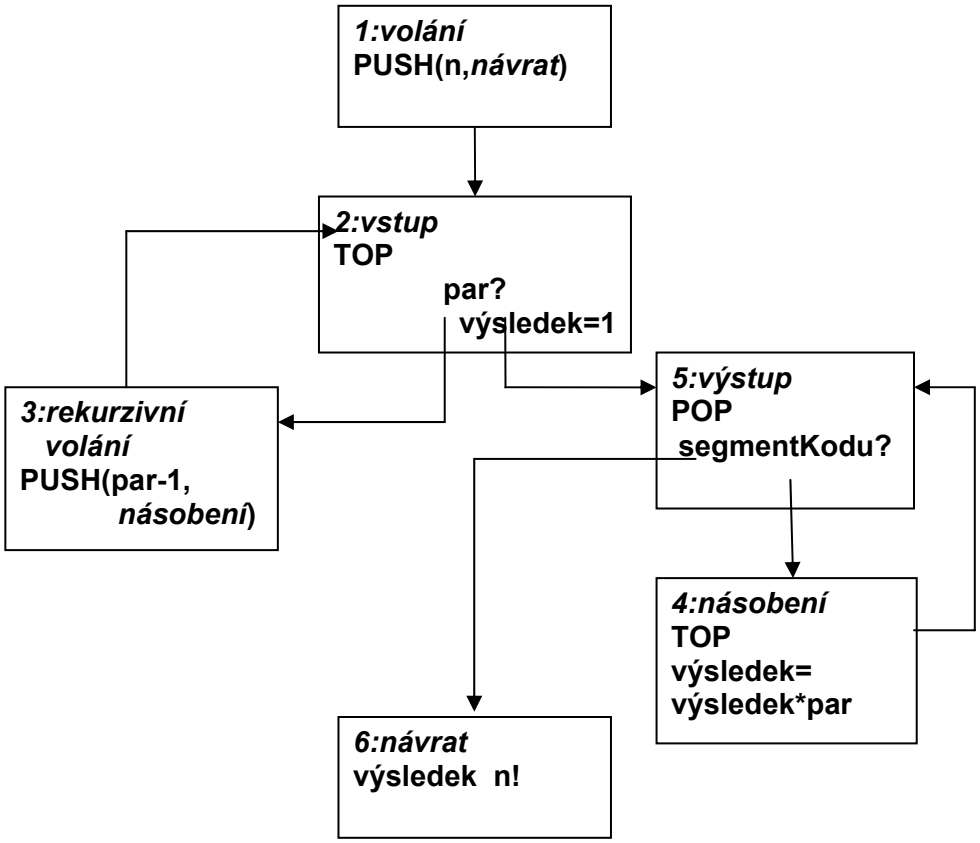
```
class AZaznam {

  int par;
  int segmentKodu;

  AZaznam(int parametr, int segment) {
    par = parametr;
    segmentKodu = segment;
  }
}
```

vstup do rekurzivní metody zjišťuje hodnotu parametru
- **`if (n > 1)…`**

čtení prvku na vrcholu zásobníku (ne **`pop`**) - **`top`**

```
class AZasobnik {

  private AZaznam[] z;
  private int vrchol;
  final int maxN=10;


  AZasobnik() {
    z = new AZaznam[maxN];
    vrchol = -1;
  }

    void push(AZaznam ref) {
    z[++vrchol] = ref;
  }

  AZaznam pop() {
    return z[vrchol--];
  }

  AZaznam top() {
    return z[vrchol];
  }
}
```

Schéma výpočtu programu **`Faktorial`**



```
class ZFaktorial {

  static int n;
  static int vysledek;
  static AZasobnik azasob;
  static int segment;
  static AZaznam azaz;

  public static void main(String[] args) {
    n=7;
    faktorial();
    System.out.println(n + "! = " + vysledek);
  }

  static void faktorial() {
    azasob = new AZasobnik();
    segment = 1;
    while (pokracuj())
      ;
  }
```

```
  static boolean pokracuj() {
    switch(segment) {
      case 1:                  // volání
        azaz = new AZaznam(n,6);
        azasob.push(azaz);
        segment = 2;
        break;
      case 2:                  // vstup
        azaz = azasob.top();
        if(azaz.par > 1)
          segment = 3;
        else {
          vysledek = 1;
          segment = 5;
        }
        break;
      case 3:                  // rekurzivní volání
        AZaznam novyZaznam = new
                        AZaznam(azaz.par - 1, 4);
        azasob.push(novyZaznam);
        segment = 2;
        break;
      case 4:                  // násobení
        azaz = azasob.top();
        vysledek = vysledek * azaz.par;
        segment = 5;
        break;
      case 5:                  // výstup
        azaz = azasob.pop();
        segment = azaz.segmentKodu;
        break;
      case 6:                  // návrat
        return false;
    }
  return true;
  }
}
```

program je možné dále upravit

Výpočet 0! // hodnoty na začátku případů, větve

volani:	vysledek=	Z: ( )
vstup:	vysledek=	Z: ([0,navrat])
vystup:	vysledek=1	Z: ([0,navrat])
navrat:	vysledek=1	Z: ( )

Výpočet 2!

volani:	vysledek=	Z:( )
vstup:	vysledek=	Z:([2,navrat])
rekurzivni volani:	vysledek=	Z:([2,navrat])
vstup:	vysledek=	Z:([2,navrat] [1,nasobeni])
vystup:	vysledek=1	Z:([2,navrat] [1,nasobeni])
nasobeni:	vysledek=1	Z:([2,navrat])
vystup:	vysledek=2	Z:([2,navrat])
navrat:	vysledek=2	Z:( )

ADT Strom (tree)

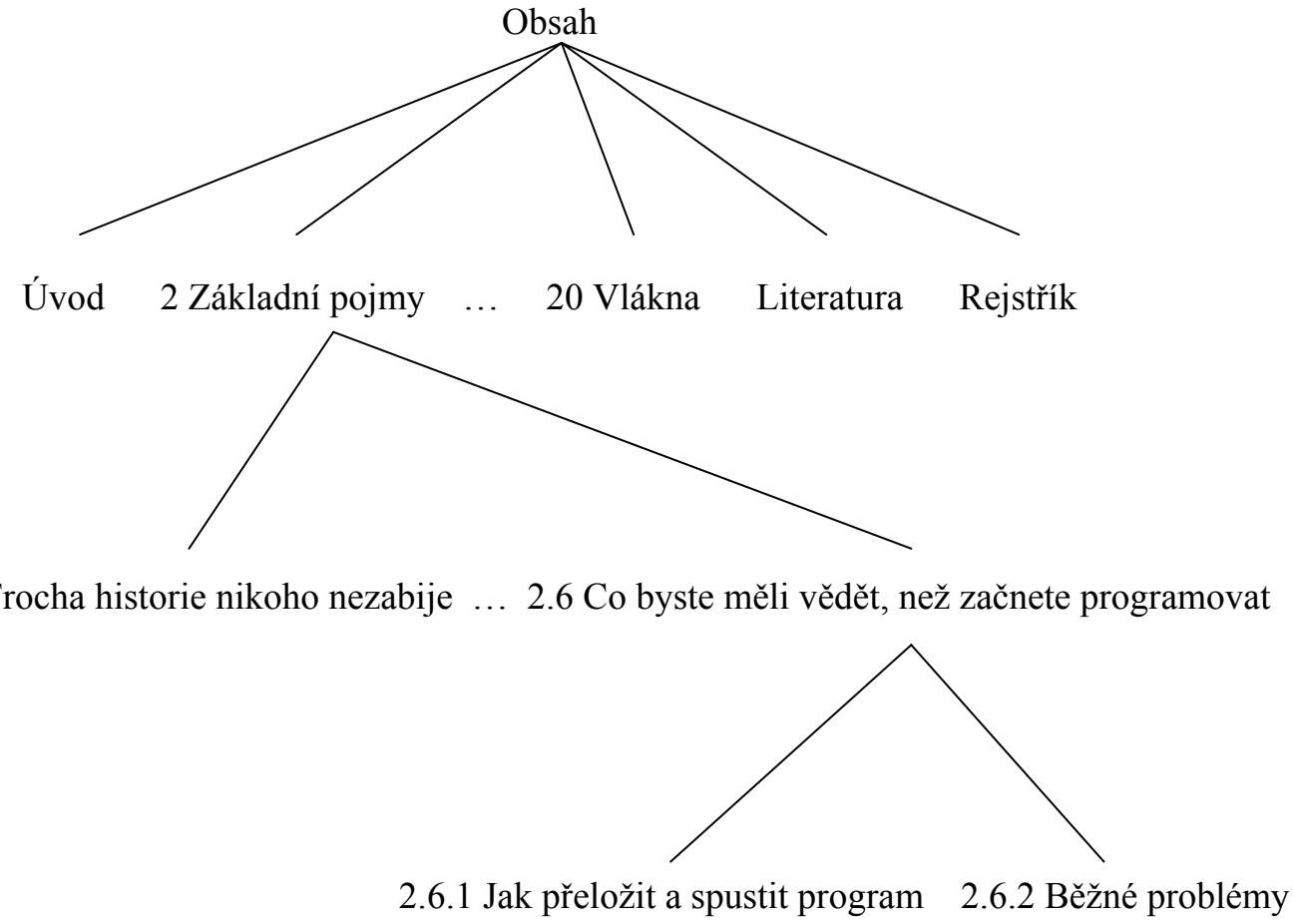
Příklady

- rodokmen (family tree)

- vylučovací systém soutěží (turnaj)

- aritmetický výraz

Organizace knížky – Herout P.: Učebnice jazyka Java



- systém souborů

Základní pojmy

vrchol - prvek ADT strom

hrana - spojení dvou vrcholů

Strom

a) Jeden vrchol je **strom**. Tento vrchol se nazývá **kořen** stromu.

b) Nechť x je vrchol a T1, T2, ..., Tn jsou stromy. **Strom** je vrchol x spojený s kořeny stromů T1, T2, ..., Tn a x je kořenem.

T1, T2, ..., Tn - **podstromy**

kořeny T1, T2, ..., Tn – **následníci (synové)** vrcholu x

vrchol x - **předchůdce (otec)** kořenů T1, T2, ..., Tn

prázdná množina prvků - strom

list – vrchol bez následníků

vnitřní vrchol – vrchol, který není listem

cesta - posloupnost vrcholů, ve které po sobě jdoucí vrcholy jsou spojeny hranou

délka cesty - počet hran cesty

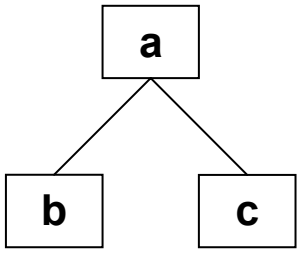
Ke každému vrcholu je z kořene právě jedna cesta.

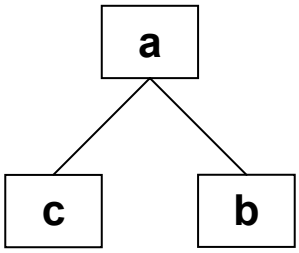
hloubka vrcholu ve stromě (úroveň, na které se nachází) - délka cesty z kořene k vrcholu

Úroveň (hloubka) kořene stromu je nulová.

výška stromu - maximální hloubka vrcholu stromu

Následníci můžou být uspořádani.





Binární strom je prázdný strom anebo vrchol, který má levý a pravý podstrom, které jsou binární stromy.

Implementace polem

Pozice vrcholů úplného binárního stromu lze očíslovat:

kořenu přiřadíme 1
levému následníku 2
pravému následníku 3
pokračujeme na každé úrovni zleva až do konce

pozice = index prvku pole

je-li na ní vrchol – klíč
není-li – např. -1

Má-li vrchol hodnotu indexu ***i***, potom

levý následník má index ***2i***

pravý následník má index ***2i + 1***

předchůdce, pokud existuje, má index ***i/2*** (celočíselně).

- musíme vytvořit pole pro předpokládanou maximální velikost stromu
- navíc musí obsahovat i prvky pro pozice neobsazené vrcholy

Poznámka: Uvedené vztahy platí, má-li kořen stromu index 1. Pokud bychom kořen umístnili do prvku pole s indexem 0, tyto vztahy nutno upravit.

Implementace spojovou strukturou

```
class Vrchol {
  int klic;
  Vrchol levy;
  Vrchol pravy;

  ...

  void tiskVrcholu() {
    System.out.print(klic+" ");
  }
}
```

Přímý průchod (preorder) navštívíme vrchol, potom levý a pravý podstrom

Vnitřní průchod (inorder) navštívíme levý podstrom, vrchol a pravý podstrom

Zpětný průchod (postorder) navštívíme levý a pravý podstrom a potom vrchol

Rekurzivní průchod stromem

preorder:

```
void pruchodR(Vrchol v) {
  if (v == null)
    return;
  v.tiskVrcholu();
  pruchodR(v.levy);
  pruchodR(v.pravy);
}

Vrchol koren;
pruchodR (koren);
```

inorder: posunutím řádku s tiskem mezi rekurzivní volání
postorder: posunutím za obě rekurzivní

Čas průchodu stromem:

$T(0) = c_{por}$

$T(n) = T(m) + T(n-m-1) + c_{por} + c_{tisk}$ pro $n > 0$

$T(n) = (2c_{por} + c_{tisk})n + c_{por}$

Čas průchodu stromem je **$\Theta(n)$**.

Nerekurzivní implementace průchodů

1. Do zásobníku vložíme procházený strom
2. Dokud zásobník není prázdný, v cyklu vybereme prvek ze zásobníku a je-li ním klíč vytiskneme ho, jinak do zásobníku vložíme

pro *preorder:* pravý podstrom, levý podstrom, klíč vrcholu

pro *inorder:* pravý podstrom, klíč vrcholu, levý podstrom

pro *postorder:* klíč vrcholu, pravý podstrom, levý podstrom

Nerekurzivní *preorder*

Pro *preorder* průchod je při vkládání jako poslední vložen klíč a ten je tedy na začátku uvedeného cyklu vybrán a vytisknut.

```
void pruchod(Vrchol v) {
  VZasobnik z = new VZasobnik();
  z.push(v);
  while (!z.jePrazdny()) {
    v = z.pop();
    v.tiskVrcholu();
    if (v.pravy != null) z.push(v.pravy);
    if (v.levy != null) z.push(v.levy);
  }
}
```

Průchod po úrovních (level order)

```
void pruchod(Vrchol v) {
  VFronta f = new VFronta();
  f.vloz(v);
  while (!f.jePrazdna()) {
    v = f.vyber ();
    v.tiskVrcholu();
    if (v.levy != null) f.vloz(v.levy);
    if (v.pravy != null) f.vloz(v.pravy);
  }
}
```

Binární vyhledávací stromy (BVS)

uspořádaná množina

	vyhledání	vložení
implementace polem	O(log n)	O(n)
implementace seznamem	O(n)	O(n)

Je lepší implementace ?

Prvky ve vrcholech BVS splňují **BVS vlastnost**:

Nechť *x* je vrchol stromu.
Je-li *y* vrchol v levém podstromu, potom *y.klíč* < *x.klíč*.
Je-li *y* vrchol v pravém podstromu, potom *y.klíč* > *x.klíč*.

(klíče všech prvků jsou různé)

vyhledání dat uložených ve vrcholu

```
private class DVrchol {
  int klic;
  String data;
  DVrchol levy;
  DVrchol pravy;

  DVrchol (int klic, String data) {
    this.klic = klic;
    this.data = data;
  }

  void tiskVrcholu() {
    System.out.print(data+" ");
  }
}
```

BVS strom

```
private DVrchol koren;
```

prázdný strom

```
koren == null
```

signatura metody `hledej`

```
String hledej(int)
```

rekurzivní implementace

```
private String hledejR(DVrchol v, int klic) {
  if (v == null)
    return null;
  if (klic == v.klic)
    return v.data;
  if (klic < v.klic)
    return hledejR(v.levy, klic);
  else
    return hledejR(v.pravy, klic);
}

String hledej(int klic) {
  return hledejR(koren, klic);
}
```

odstranění koncové rekurze

```
  String hledej(int klic) {
    DVrchol x = koren;
    while (x != null && klic != x.klic)
      if (klic < x.klic )
        x = x.levy;
      else
        x = x.pravy;
    return x == null ? null : x.data;
  }
```

nalezení minimálního a maximálního prvku (neprázdného BVS)

```
int minKlic() {
  DVrchol x = koren;
    while (x.levy != null)
      x = x.levy;
  return x.klic;
}


int maxKlic() {
  DVrchol x = koren;
  while (x.pravy != null)
    x = x.pravy;
  return x.klic;
}
```

vložení a výběr prvku

predch - ukazatel na předchůdce

```
  private class DVrchol {
    int klic;
    String data;
    DVrchol levy;
    DVrchol pravy;
    DVrchol predch;

    DVrchol (int klic, String data) {
      this.klic = klic;
      this.data = data;
    }

    void tiskVrcholu() {
      System.out.print(data+" ");
    }
  }
```

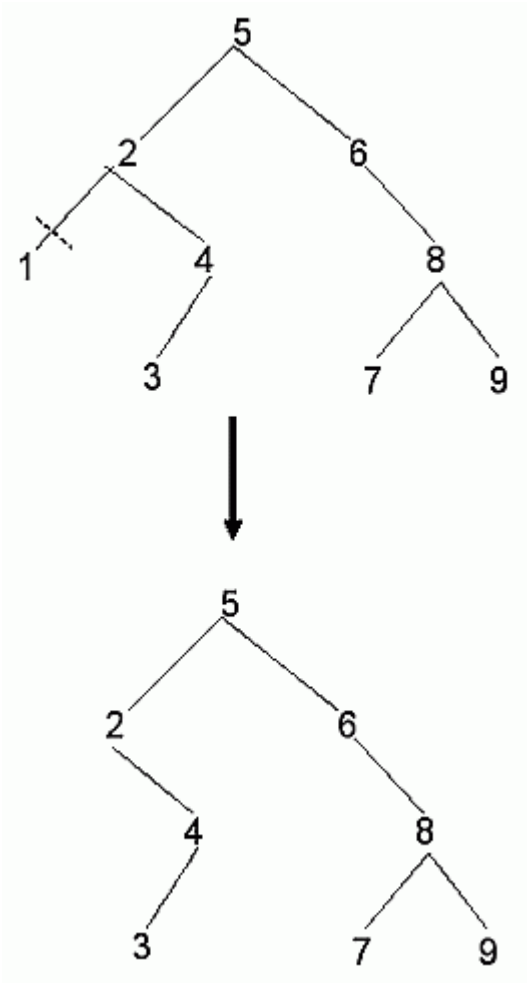
signatura metody `vloz`

```
void vloz(int, String)
```

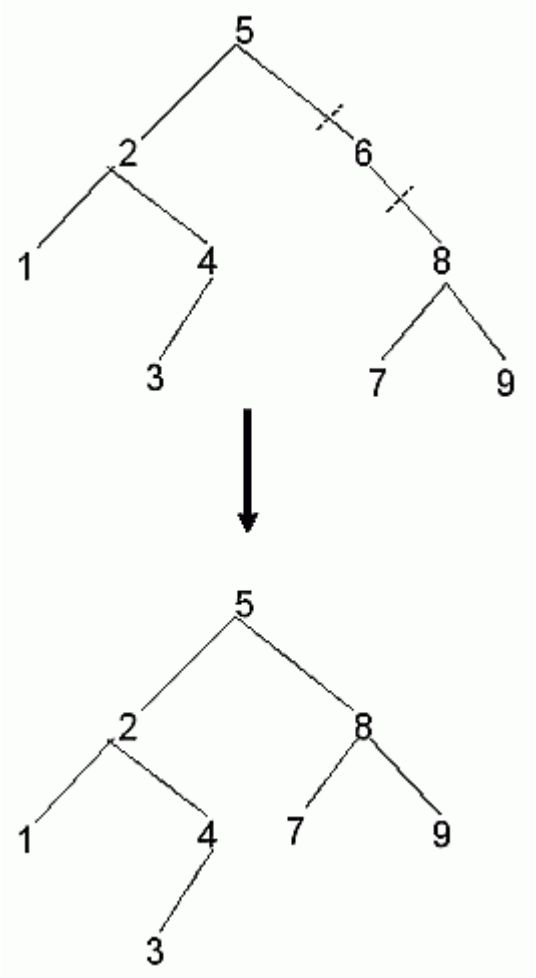
```
  void vloz(int klic, String data) {
    DVrchol x = koren, predch = null;
    while (x != null ) {
      predch = x;
      if (klic < x.klic)
        x = x.levy;
      else
        x = x.pravy;
    }
    DVrchol z = new DVrchol(klic, data);
    z.predch = predch;
    if (predch == null)
      koren = z;
    else if (klic < predch.klic)
           predch.levy = z;
         else
           predch.pravy = z;
  }
```

Průchod *inorder* BVS vytiskne tyto prvky uspořádané vzestupně podle klíče.

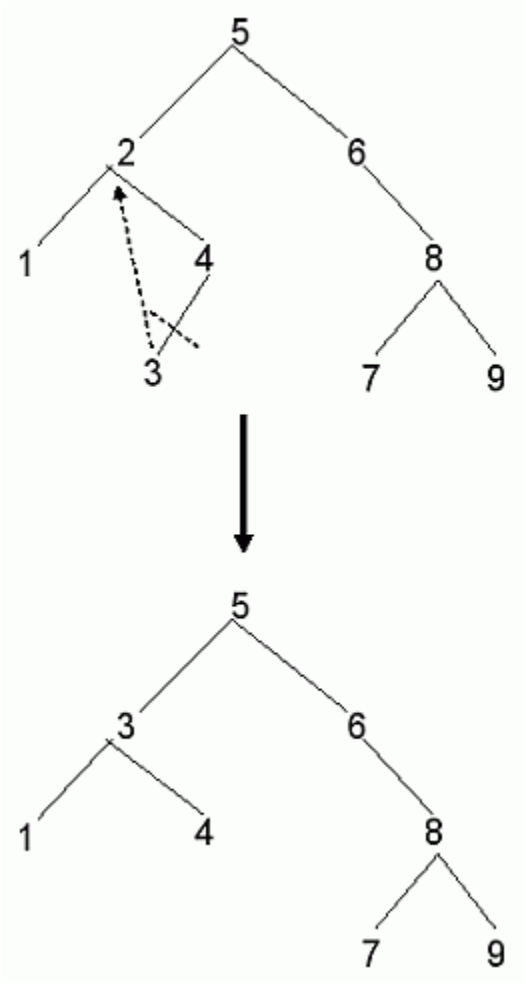
výběr prvku – list



výběr prvku – vnitřní prvek s jedním následníkem



výběr prvku – vnitřní prvek se dvěma másledníky



signatura metody `vyber`

```
void vyber(int)
```

```
  void vyber(int klic) {
    // najdeme vrchol z na vylouceni
    DVrchol z = koren;
    while (z != null && klic != z.klic)
      if (klic < z.klic )
        z = z.levy;
      else
        z = z.pravy;
    // urcime vrchol y na odstraneni
    DVrchol y = z;
    if (z.levy != null && z.pravy != null) {
      y = z.pravy;
      while (y.levy != null)
        y = y.levy;
    }
    // x ukazuje na naslednika y anebo je null,
    // kdyz nema zadneho naslednika
    DVrchol x;
    if (y.levy != null)
      x = y.levy;
    else
      x = y.pravy;
    // modifikaci y.predch a x odpojime y
    if (x != null)
      x.predch = y.predch;
    if (y.predch == null)
      koren = x;
    else
      if (y == y.predch.levy)
        y.predch.levy = x;
      else
        y.predch.pravy = x;
```

```
  // nebyl-li odpojen z, skopirujeme klic a
  // data
  if (y != z) {
    z.klic = y.klic;
    z.data = y.data;
  }
  // uvolnime y
  y = null;
}
```

Vlastnosti BVS

h – výška BVS
složitost hledání a vkládání je ***O(h)***

N – počet prvků (vrcholů)

nejhorší případ: $h = N - 1$
složitost je ***O(N)***

nejlepší případ:
kromě poslední úrovně, jsou zaplněny všechny vyšší úrovně

$$2^h \leq N < 2^{h+1}$$

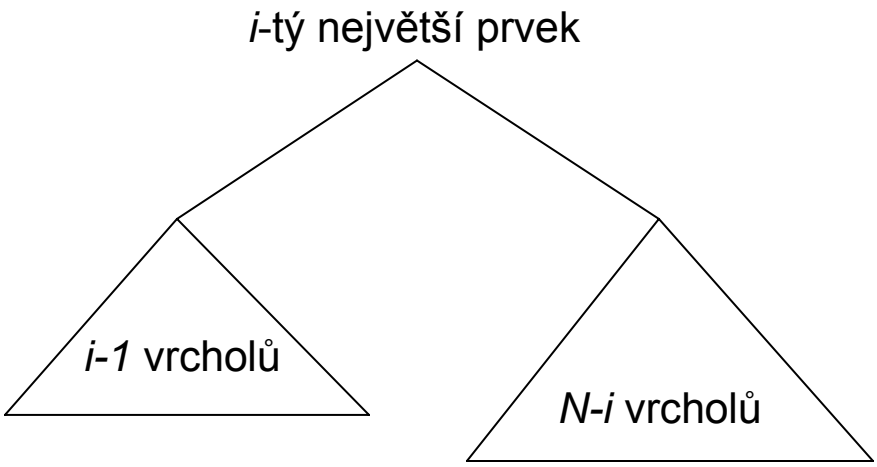
$h = log_2 N$
složitost je ***O(log N)***

obecně:

průměrná hloubka vrcholu – hledaného/vkládaného

součet délek cest k vrcholům BVS / N

kořen BVS - ***i***-tý největší prvek



,

P_N - průměrný součet délek cest BVS s N vrcholy

$$P_N = \frac{1}{N} \sum_{1 \le i \le N} (P_{i-1} + i - 1 + P_{N-i} + N - i)$$

$$P_N \approx 2N \ln N = 1{,}39N \log_2 N$$

průměrný hloubka vrcholu pro průměrný případ: P_N / N
složitost je ***O(log N)***

Poznámky:

Výběr prvku lze učinit několika způsoby, vedou však k tomu, že strom po odstranění nezůstává náhodným, někdy se průměrná hloubka změní na $\sqrt{N}$.

Možným řešením je označit vybraný prvek jako neplatný, například použitím logické členské proměnné. Často v aplikaci nemáme mnoho vybraných prvků a dokonce můžou být užitečné pokud by jich bylo později opět potřeba.

Další technikou je vytvoření nového stromu pro platné prvky, dosáhl-li počet neplatných vrcholů nějakou mez, co je také častý způsob práce s datovými strukturami.

Vyvážené stromy – nejhorší případ operací je ***O(****log* ***N).***
(*cesta z kořene ke každému listu není o moc delší než k ostatním*)

http://java.sun.com/j2se/1.5.0/docs/api/java/util/TreeMap.html

Odpovídá každé permutaci N prvků různý BVS?

Uložme prvky posloupnosti do mřížky tak, že řádek odpovídá hodnotě a sloupec poloze

Příklad

3,5,2,4,1

x	x	x	x	1
x	x	2	x	x
3	x	x	x	x
x	x	x	4	x
x	5	x	x	x

- mřížková reprezentace BVS

Výměnou sloupců odpovídajících vrcholům nad a pod libovolným vrcholem se vyhledávací strom nezmění, posloupnost ano.

x	x	x	1	x
x	x	2	x	x
3	x	x	x	x
x	x	x	x	4
x	5	x	x	x

3,5,2,1,4

Počet všech posloupností z *N* prvků je *N!*

Počet různých binárních stromů s *N* vrcholy je ~ $4^N/\sqrt{(\pi N)}$