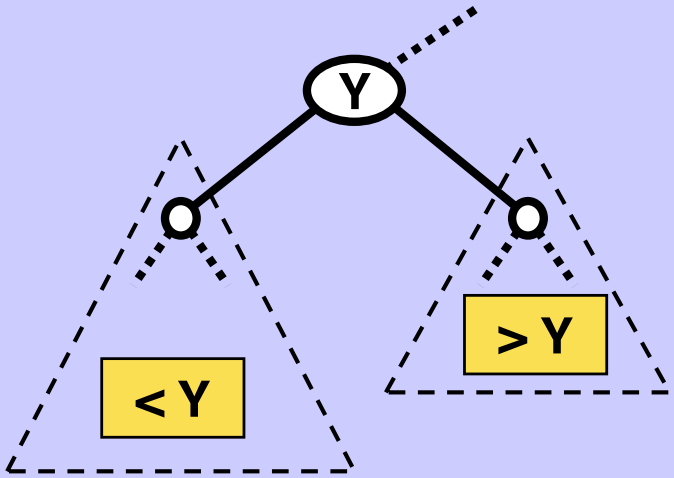
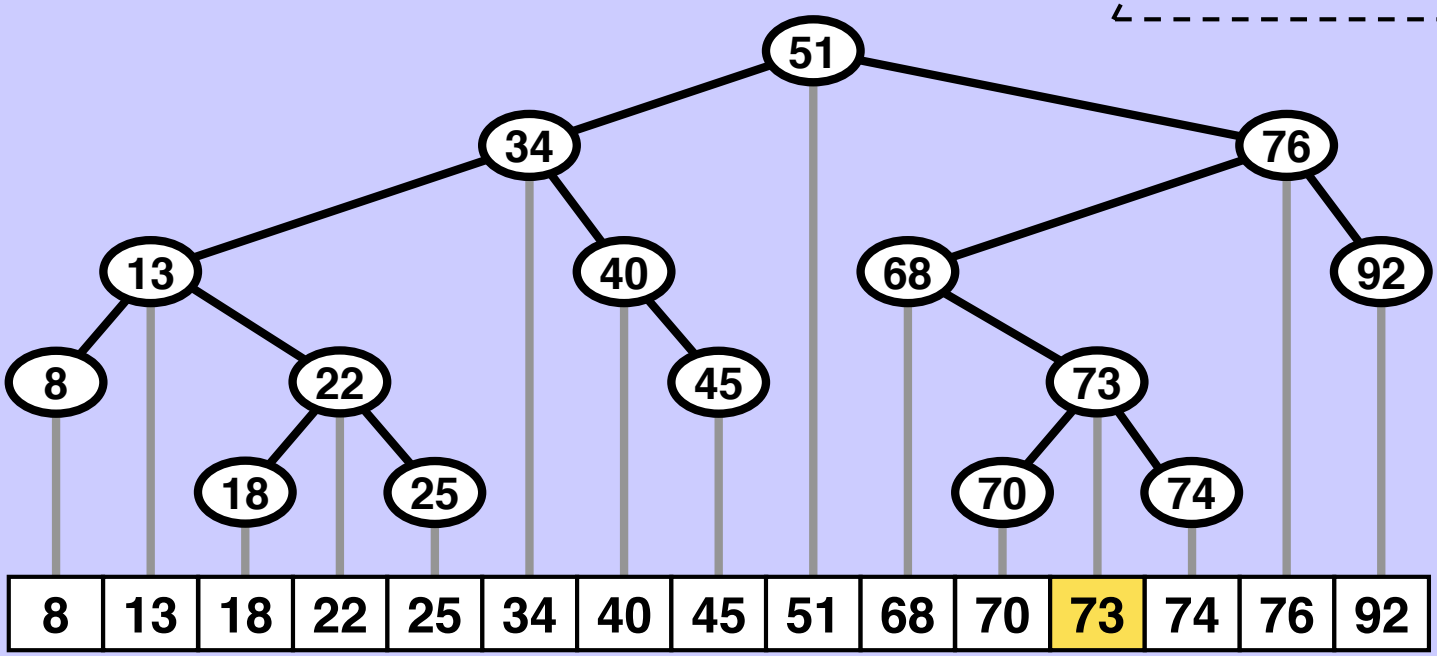
Binární vyhledávací strom

V levém podstromu každého uzlu jsou všechny klíče menší.

V pravém podstromu každého uzlu jsou všechny klíče větší.

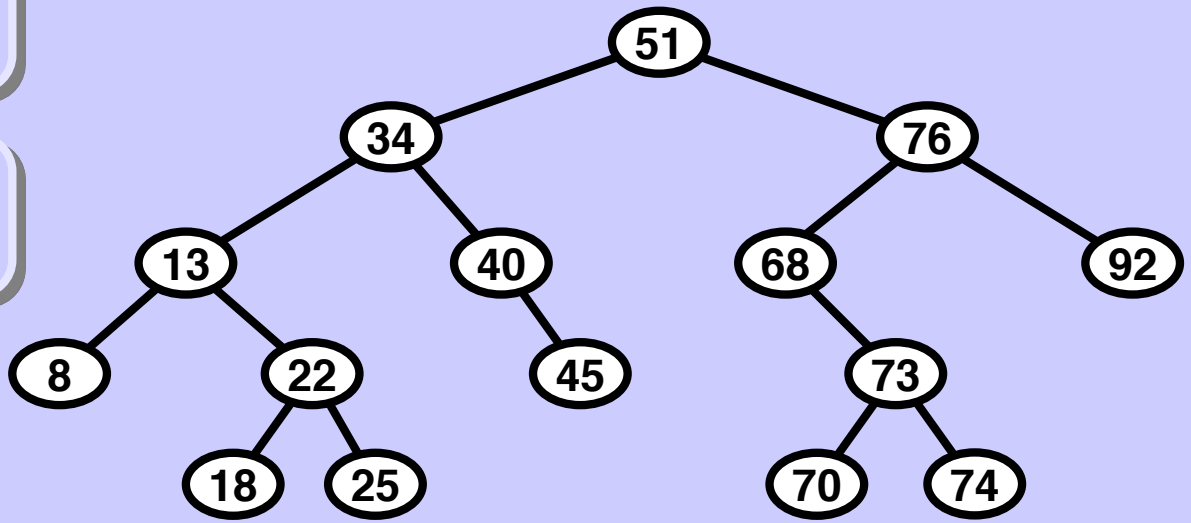




Binární vyhledávací strom

BVS nemusí být
a nebývá vyvážený.

BVS nemusí být
a nebývá pravidelný.



Výpisem prvků BVS
v pořadí INORDER
získáme uspořádané
hodnoty klíčů.

BVS je flexibilní díky operacím:

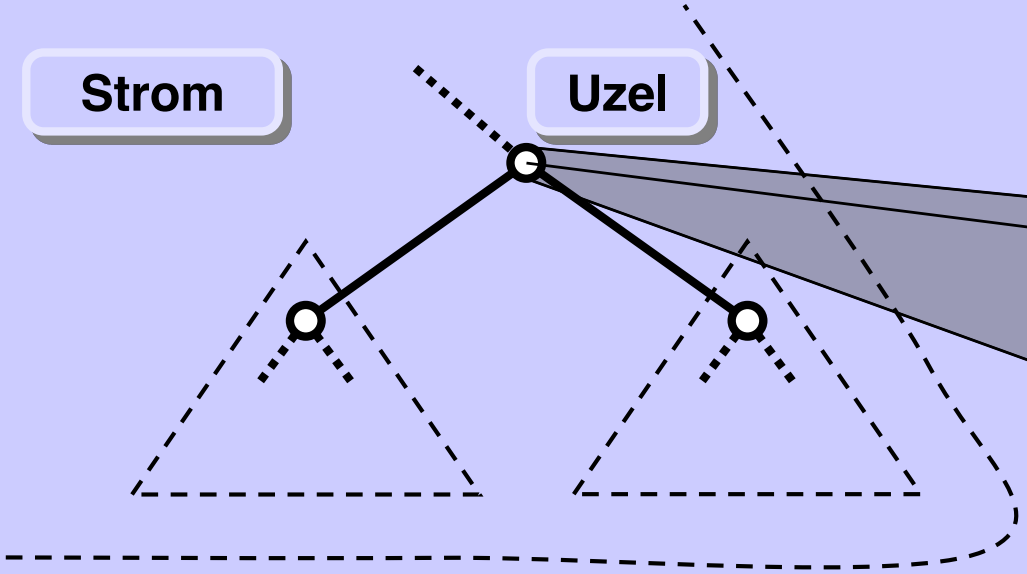
Find – najdi prvek s daným klíčem
Insert – vlož prvek s daným klíčem
Delete – (najdi a) odstraň prvek s daným klíčem

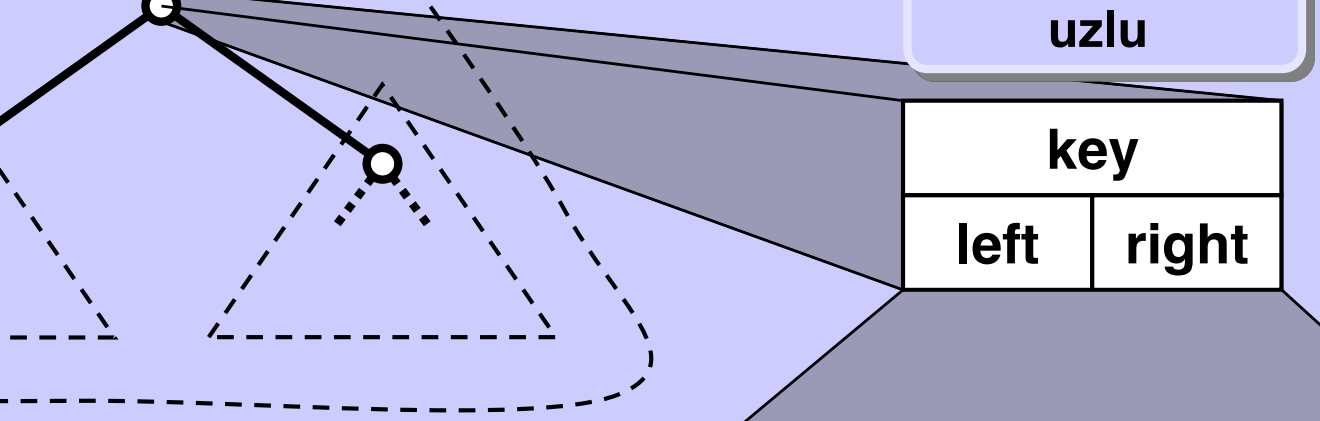
Implementace binárního stromu -- C

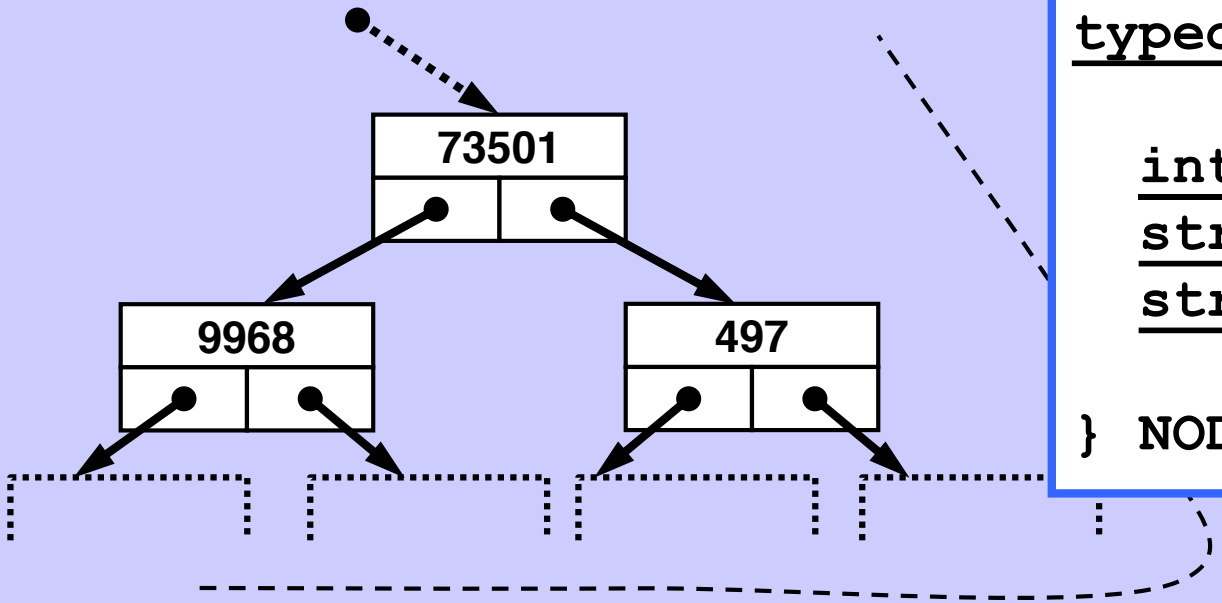
Strom

Uzel

Reprezentace uzlu





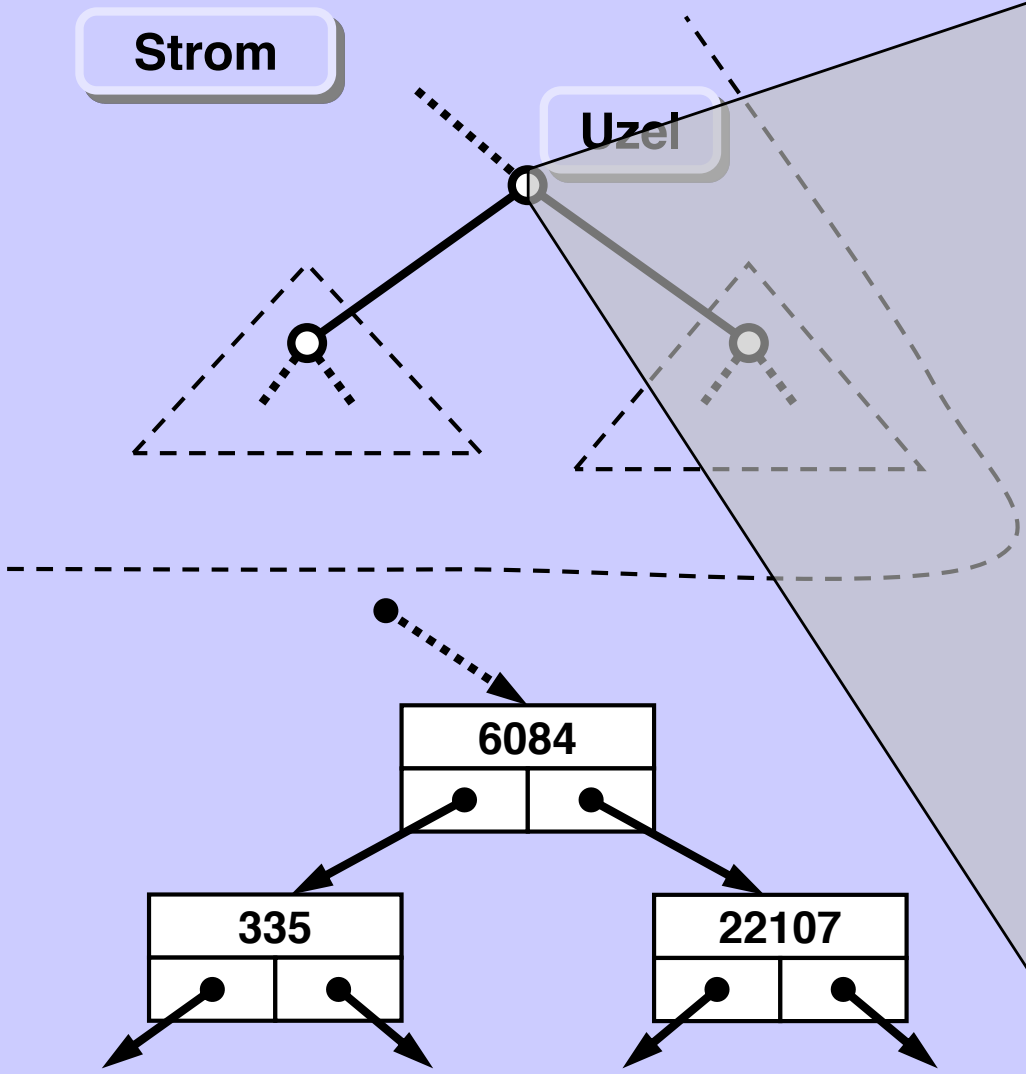



```
typedef struct Node {

   int key;
   struct Node *left;
   struct Node *right;

} NODE;
```

Implementace binárního stromu -- java

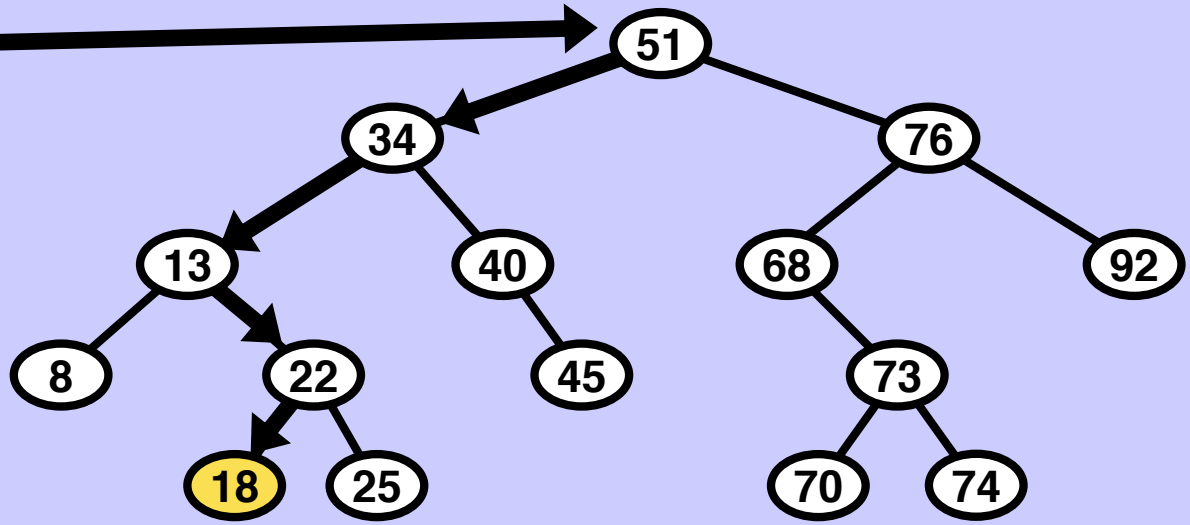



```
public class Node {
   public Node left;
   public Node right;
   public int key;
   public Node(int k) {
      key = k;
      left = null;
      right = null;
   }
}

public class Tree {
   public Node root;
   public Tree() {
      root = null;
   }
}
```

Operace Find v BVS

Najdi 18



Každá operace se stromem začíná v kořeni.

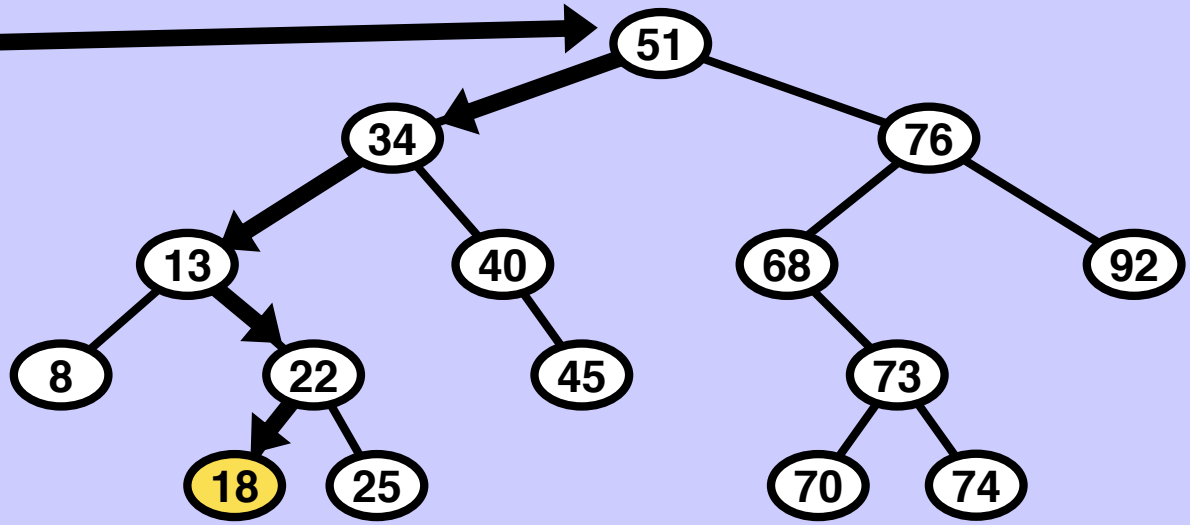
Iterativně

```
Node find(int k, Node node){
  while (true) {
     if (node == null) return null;
     if (node->key == k) return node;
     if (k < node->key) node = node->left;
        else node = node->right;
} }
```

Operace Find v BVS

Najdi 18

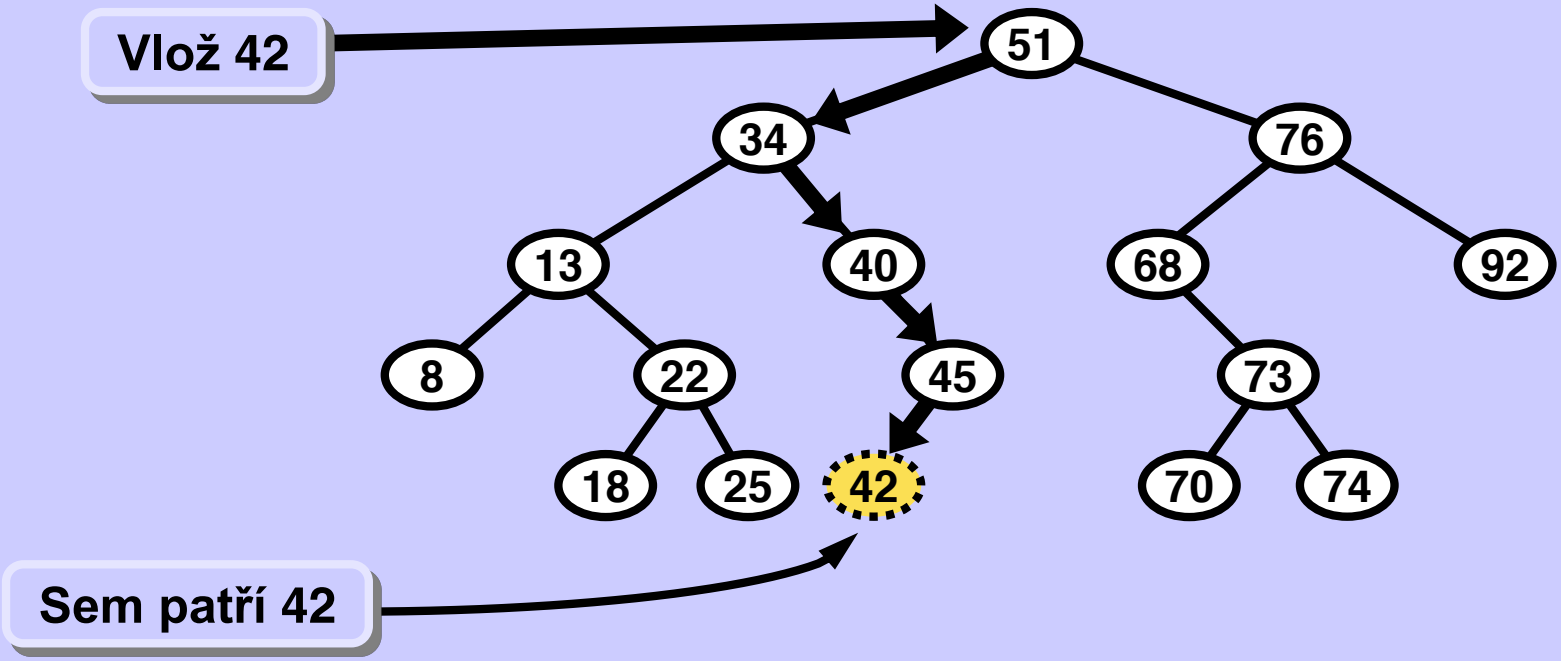
Každá operace se stromem začíná v kořeni.



Rekurzivně

```
Node findRec(int k, Node node){
  if (node == null) return null;
  if (node.key == k) return node;
  if (k < node->key) return findRec(k, node->left);
                else return findRec(k, node->right) ;
} }
```

Operace Insert v BVS



Insert
1. Najdi místo (jako ve Find) pro list, kam patří uzel s daným klíčem.
2. Vytvoř tento uzel a vlož jej do stromu.

Operace Insert v BVS iterativně

```
Node insert (int k, Node node) {
  if (node == null){                          // empty tree
    Node newNode = ...;                       // create node with key k
    return newNode;
  }
  while (true)
    if (node->key == k) return null; //can't insert a duplicate
    if (node->key > k)
       if (node->left == null) {
          Node newNode = ...;                 // create node with key k
          node->left = newNode;
          return newNode; }
       else node = node->left;
    else                                      // similarly to the right
       if(node->right == null){
          Node newNode = ...;
          node->right = newNode;
          return newNode; }
       else node = node->right; }
} }
```

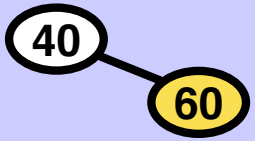
Operace Insert v BVS rekurzivně

```
Node insertRec (int k, Node node, Node parentNode) {
  if (node == null){                        // empty tree
    Node newNode = ...;                     // create node with key k
    if(parentNode != null){
       if(parentNode->key > k)
          parentNode->left = newNode;
       else
          parentNode->right = newNode;
    return newNode;
  }
  if (node->key == k) return null;    //can't insert a duplicate
  if (node->key > k)                        // chose direction
     return insertRec(k, node->left, node);
  else
     return insertRec(k, node->right, node);
}
```

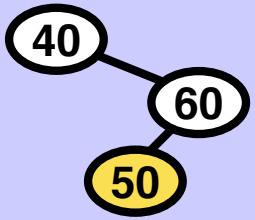
Stavba BVS operací Insert

insert 40

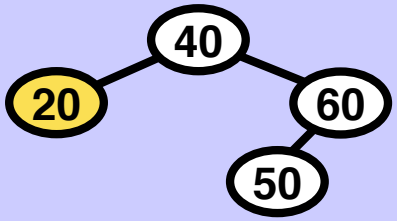
insert 60



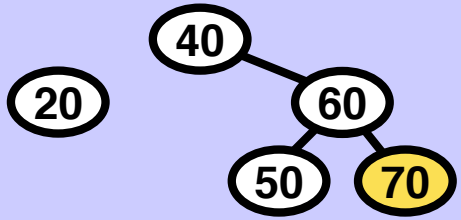
insert 50



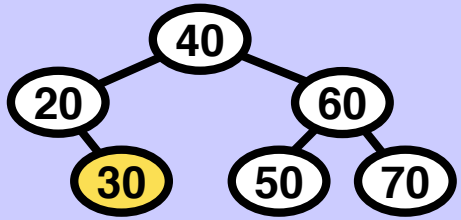
insert 20



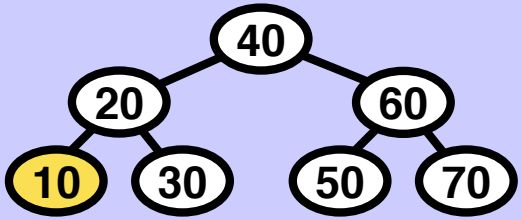
insert 70



insert 30



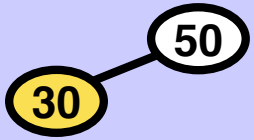
insert 10



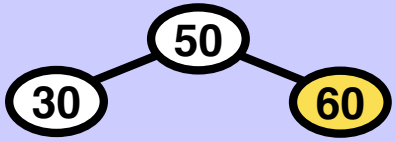
Tvar budovaného BVS závisí na pořadí vkládání dat.

insert 50

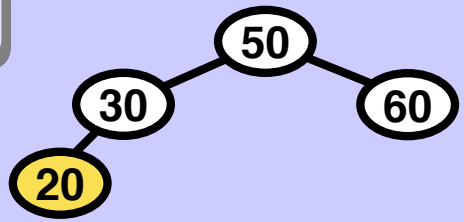
insert 30



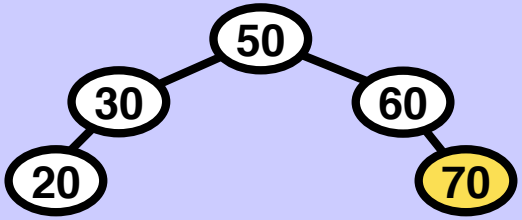
insert 60



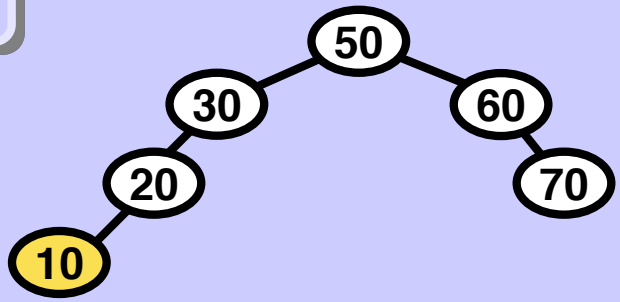
insert 20



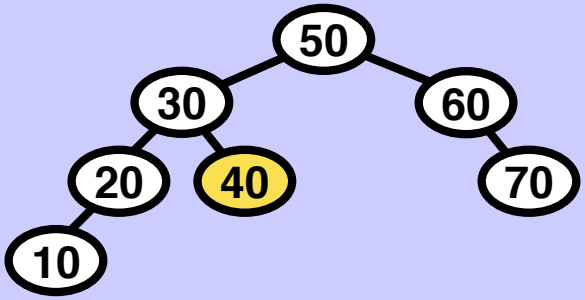
insert 70



insert 10

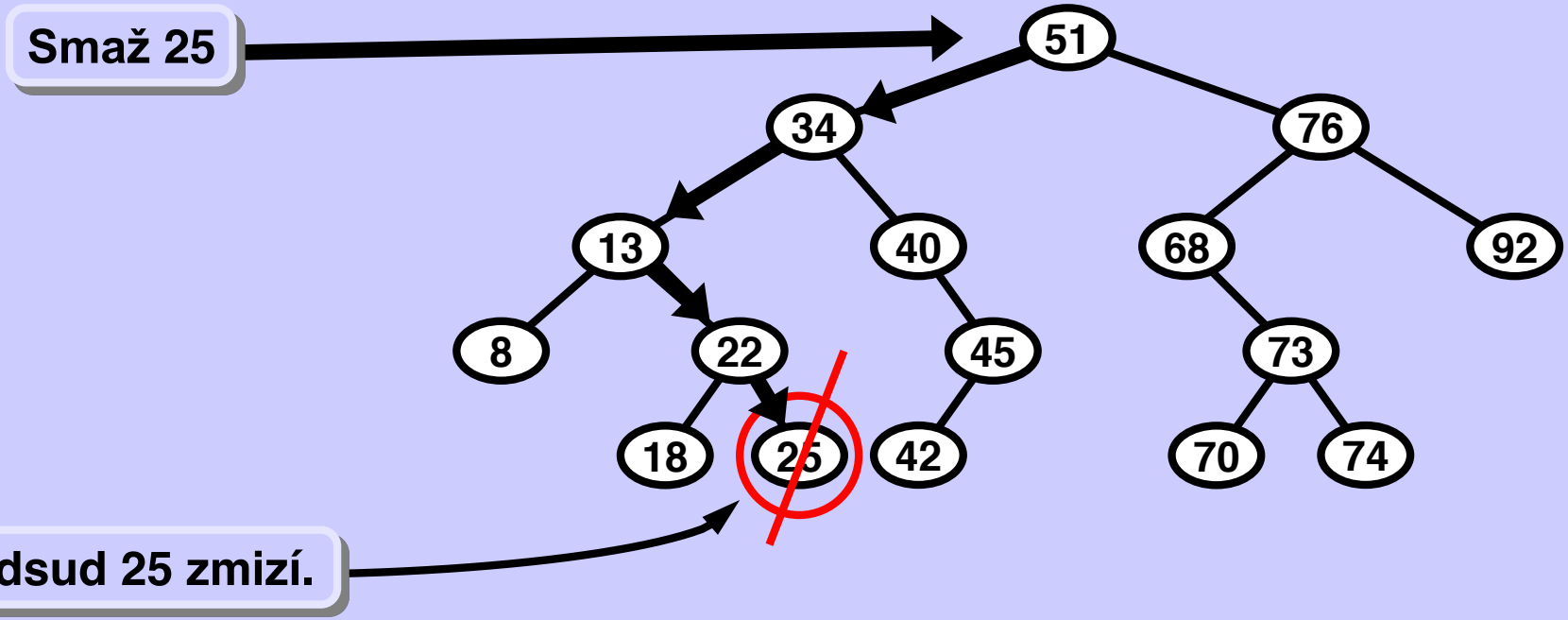


insert 40



Operace Delete v BVS (I.)

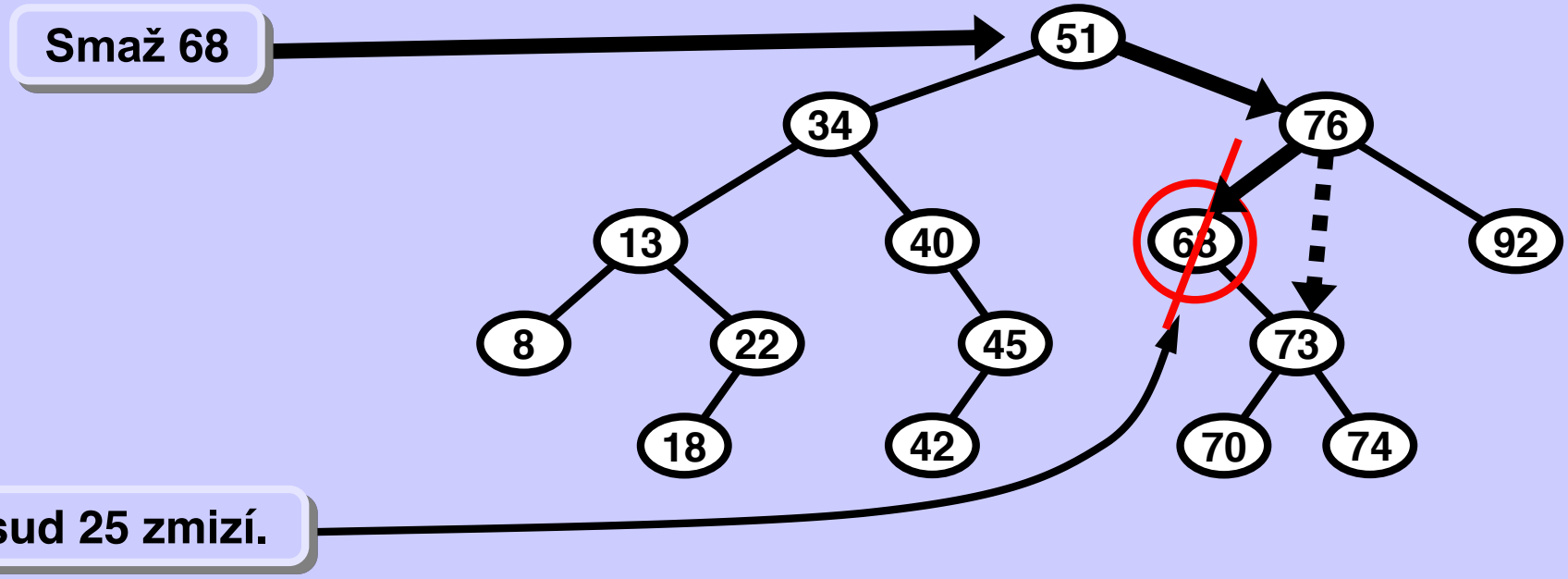
Smazání uzlu s 0 potomky (= listu)



Delete I.
Najdi daný uzel (jako ve Find) a odstraň ukazatel na něj z jeho rodiče.

Operace Delete v BVS (II.)

Smazání uzlu s 1 potomkem



Z ukazatele 76 --> 68 se stane ukazatel 76 --> 73.

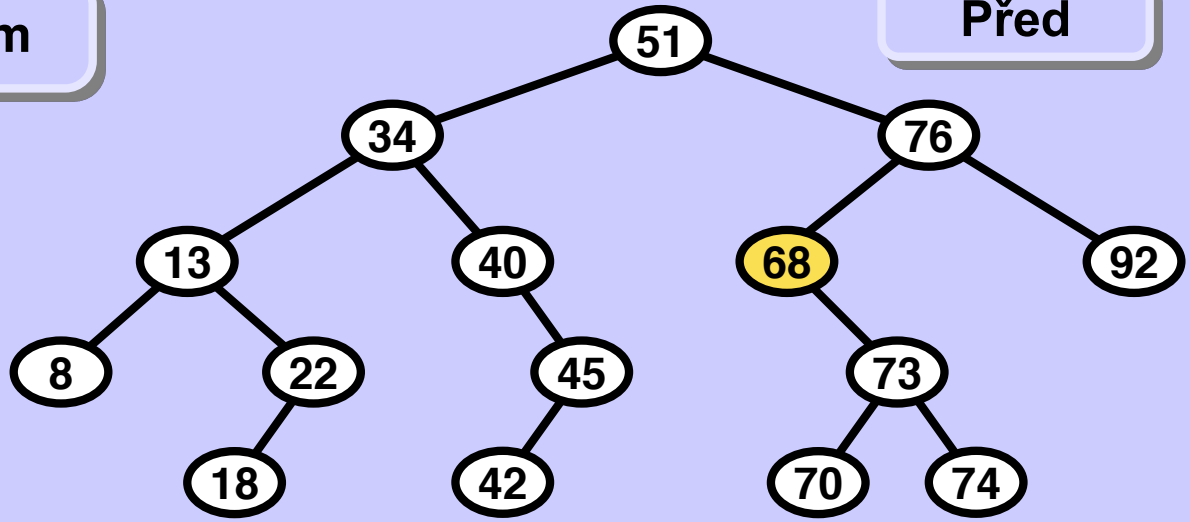
Delete II.
Najdi daný uzel (jako ve Find) a ukazatelem z jeho rodiče na něj ukaž na jeho (jediného!) potomka.

Operace Delete v BVS (II.)

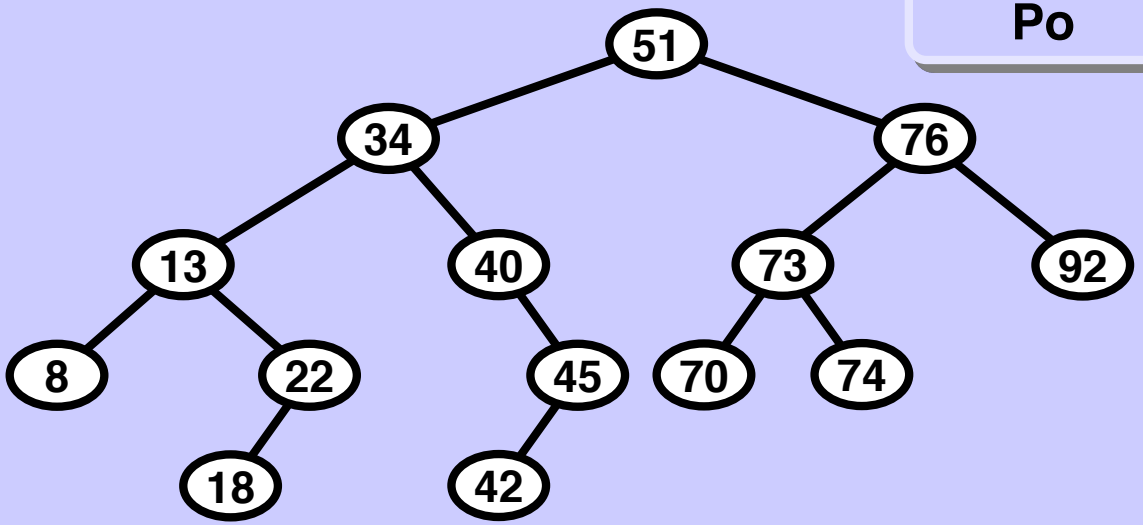
Smazání uzlu s 1 potomkem

Smaž 68

Před

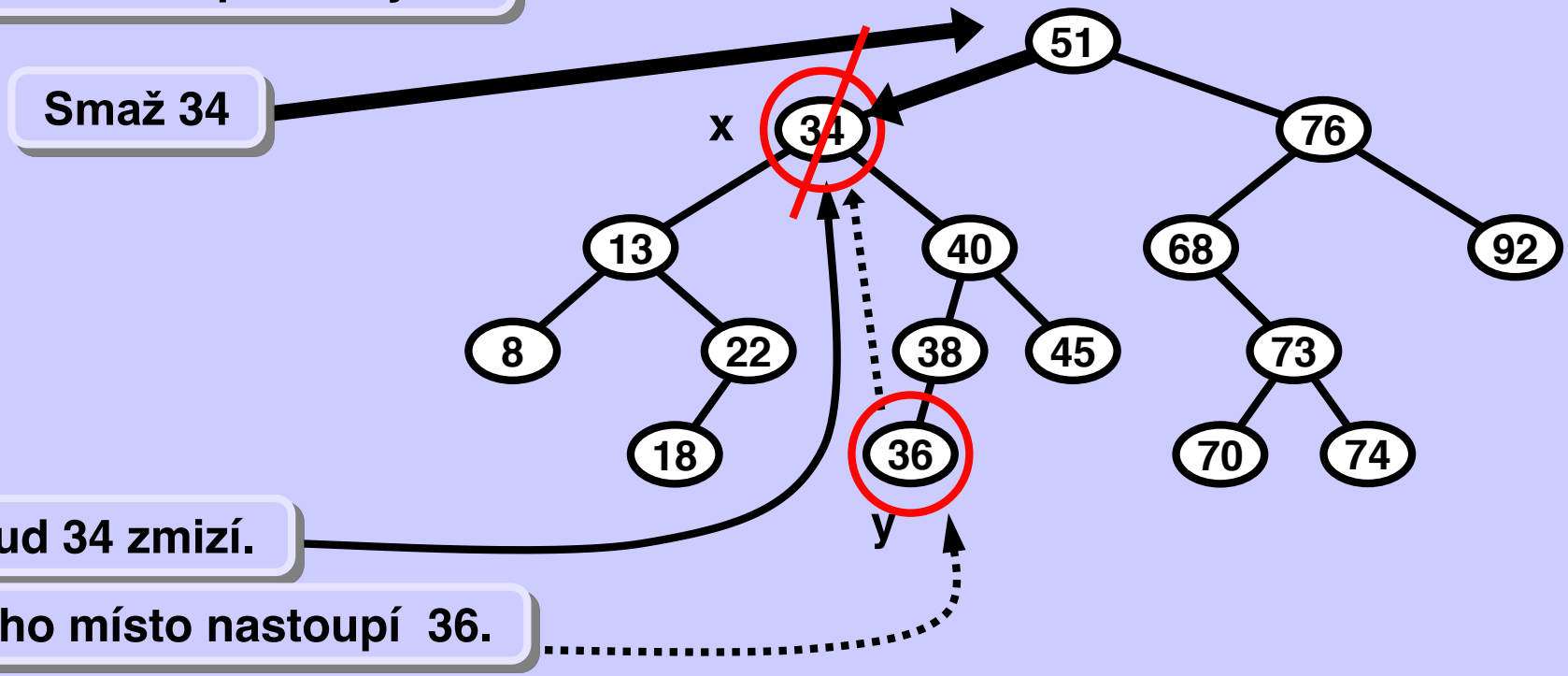


Po



Operace Delete v BVS (IIIa.)

Smazání uzlu s 2 potomky



Delete IIIa.

1. Najdi daný uzel x (jako ve Find) a dále
   najdi ***nejlevější*** (=nejmenší klíč) uzel y v ***pravém*** podstromu x.
2. Z uzlu y ukaž na potomky uzlu x, z rodiče y ukaž na potomka y místo y,
   z rodiče x ukaž na y.

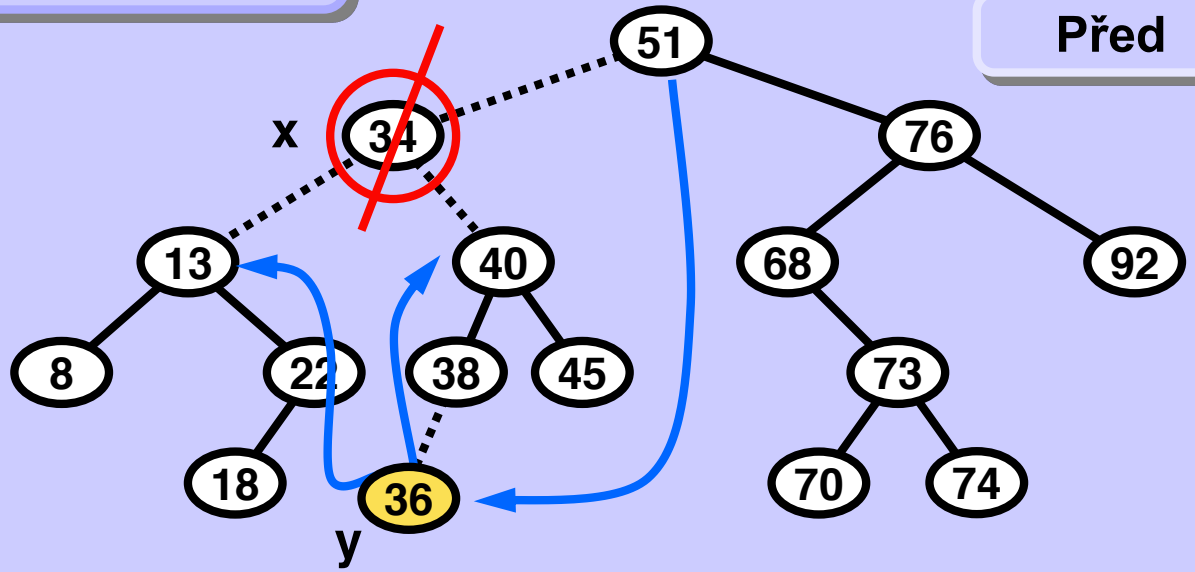
Operace Delete v BVS (IIIa.)

Smaž 34

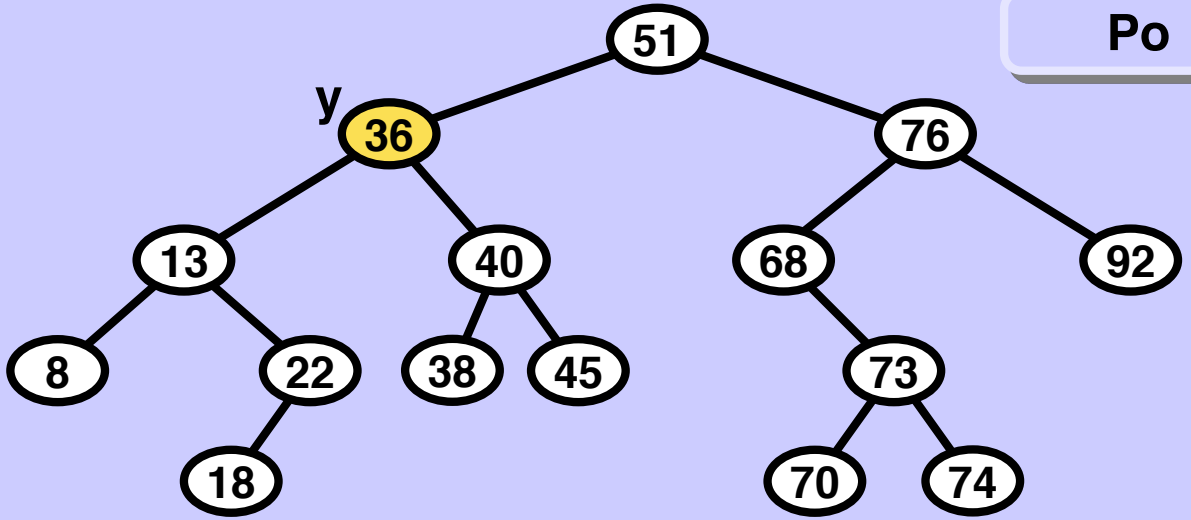
zanikající hrany/ukazatele/reference .....

vznikající hrany/ukazatele/reference ——►

Před

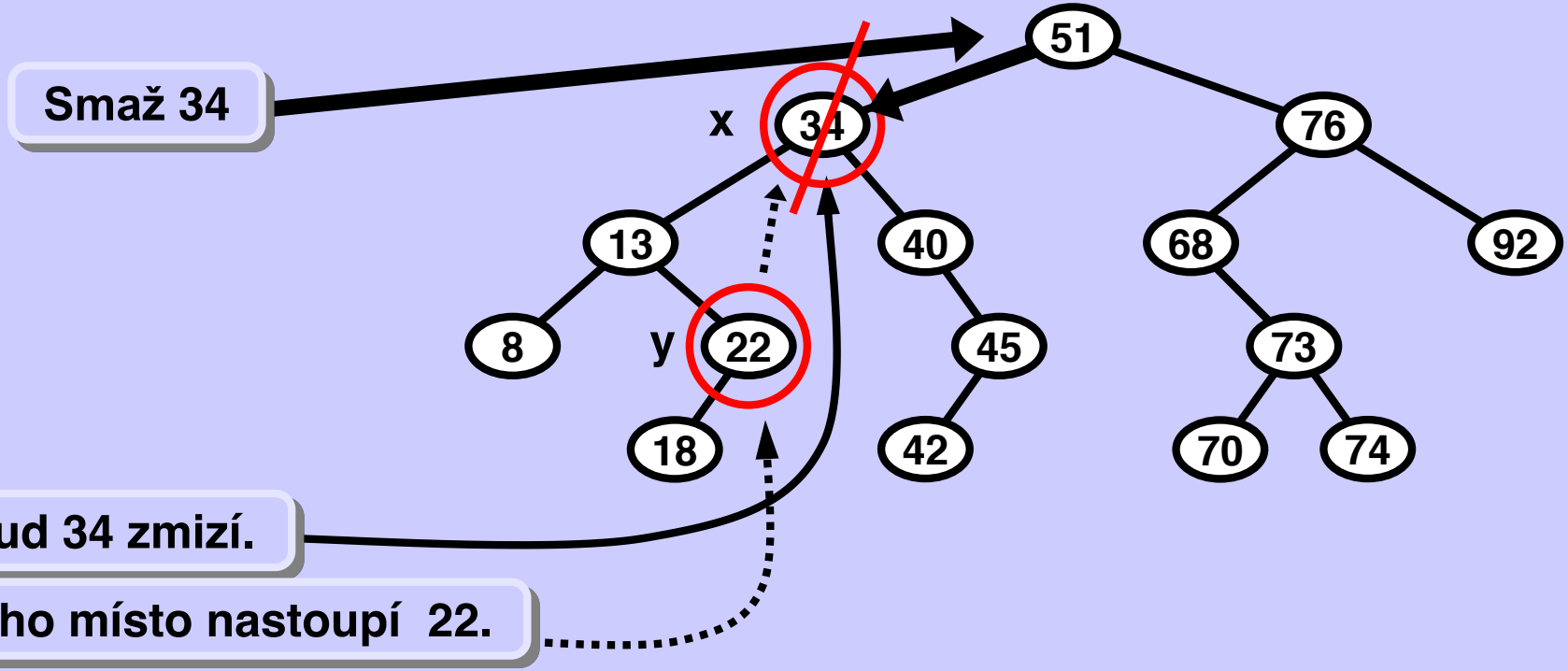


Po



Operace Delete v BVS (IIIb.) je ekvivalentní Delete IIIa.

Smazání uzlu s 2 potomky



Delete IIIb.

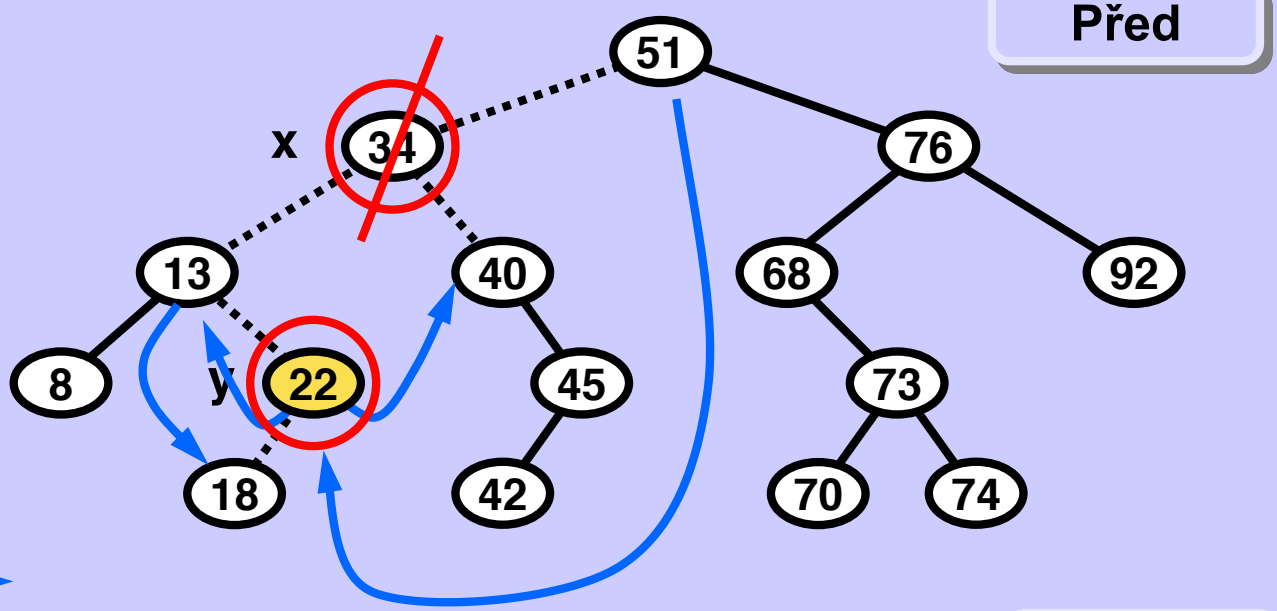
1. Najdi daný uzel x (jako ve Find) a dále
   najdi ***nepravější*** (=největší klíč) uzel y v ***levém*** podstromu x.
2. Z uzlu y ukaž na potomky uzlu x, z rodiče y ukaž na potomka y místo y,
   z rodiče x ukaž na y.

Operace Delete v BVS (IIIb.) je ekvivalentní Delete IIIa.

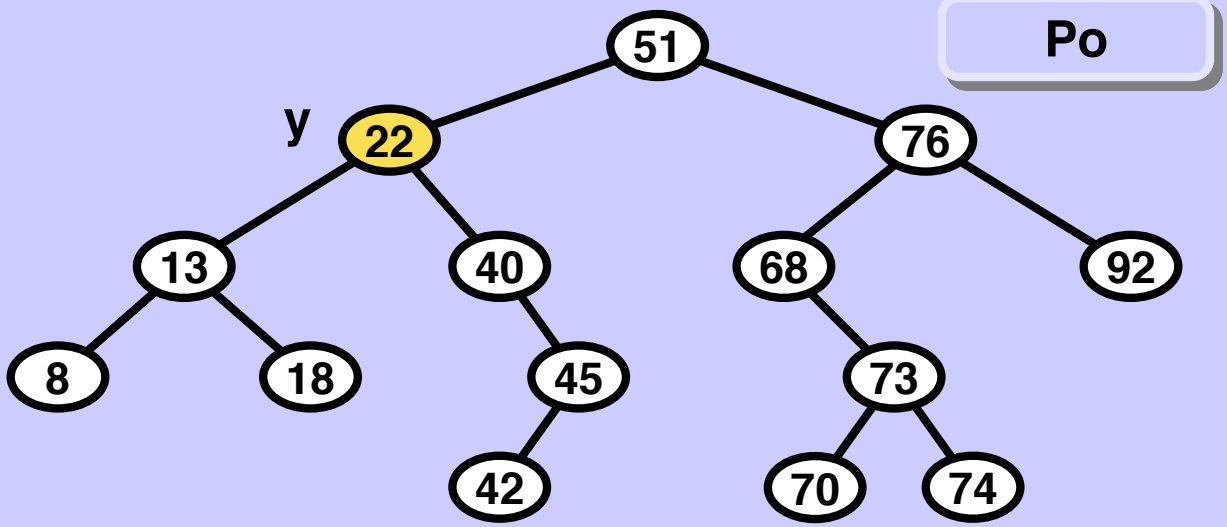
Smaž 34

zanikající hrany/ukazatele/reference

vznikající hrany/ukazatele/reference



Je nutno ošetřit případ, kdy přesouvaný uzel y má sám následníka, tedy je na něj nutno aplikovat variantu Delete II.



Operace Delete v BVS

```
Node delete (int k, Node node) {
   ...                              // homework...
}
```

Asymptotické složitosti operací Find, Insert, Delete v BVS

	BVS s n uzly	
Operace	**Vyvážený**	**Možná nevyvážený**
Find	$O(\log(n))$	$O(n)$
Insert	$\Theta(\log(n))$	$O(n)$
Delete	$\Theta(\log(n))$	$O(n)$