ALG 04

VYHLEDÁVÁNÍ (jednorozměrné vyhledávání)

Vyhledávání v poli

naivní, binární, interpolační

Binární vyhledávací strom (BVS)

operace Find, Insert, Delete

AVL strom

rotace L, R, LR, RL

Hledání v seřazeném poli — lineární, POMALÉ

Dané pole

Seřazené pole: →

Velikost = N

363	369	388	603	638	693	803	833	836	839	860	863	938	939	966	968	983	993

najdi 993 !

Testů: N

363	369	388	603	638	693	803	833	836	839	860	863	938	939	966	968	983	993

najdi 363 !

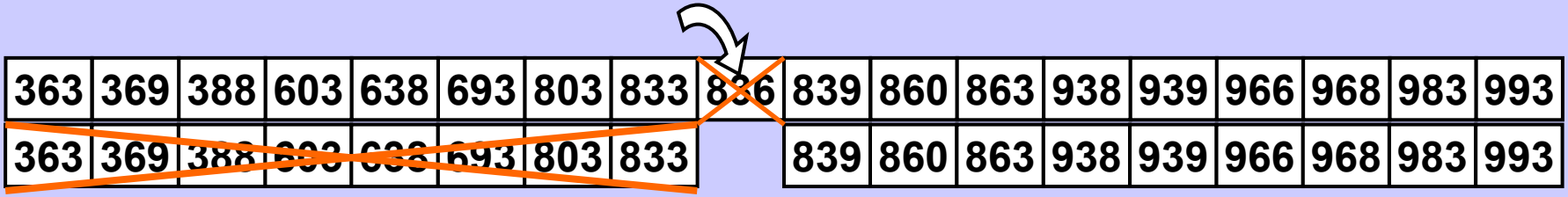
Testů: 1

363	369	388	603	638	693	803	833	836	839	860	863	938	939	966	968	983	993

Hledání v seřazeném poli — binární, RYCHLEJŠÍ

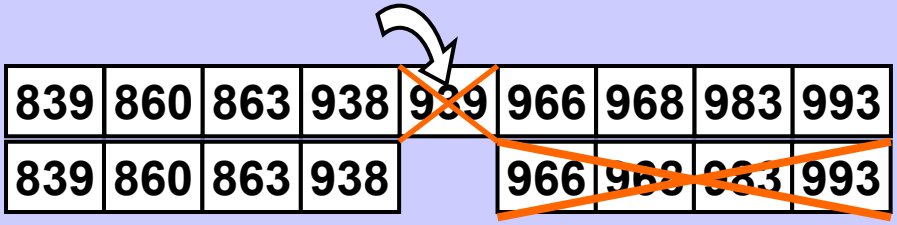
najdi 863 !

2 testy

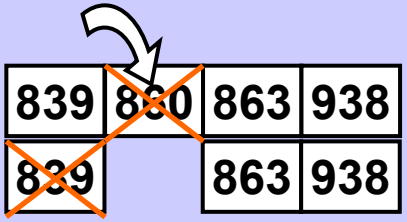


2 testy

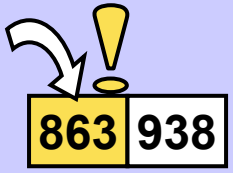
2 testy



2 testy



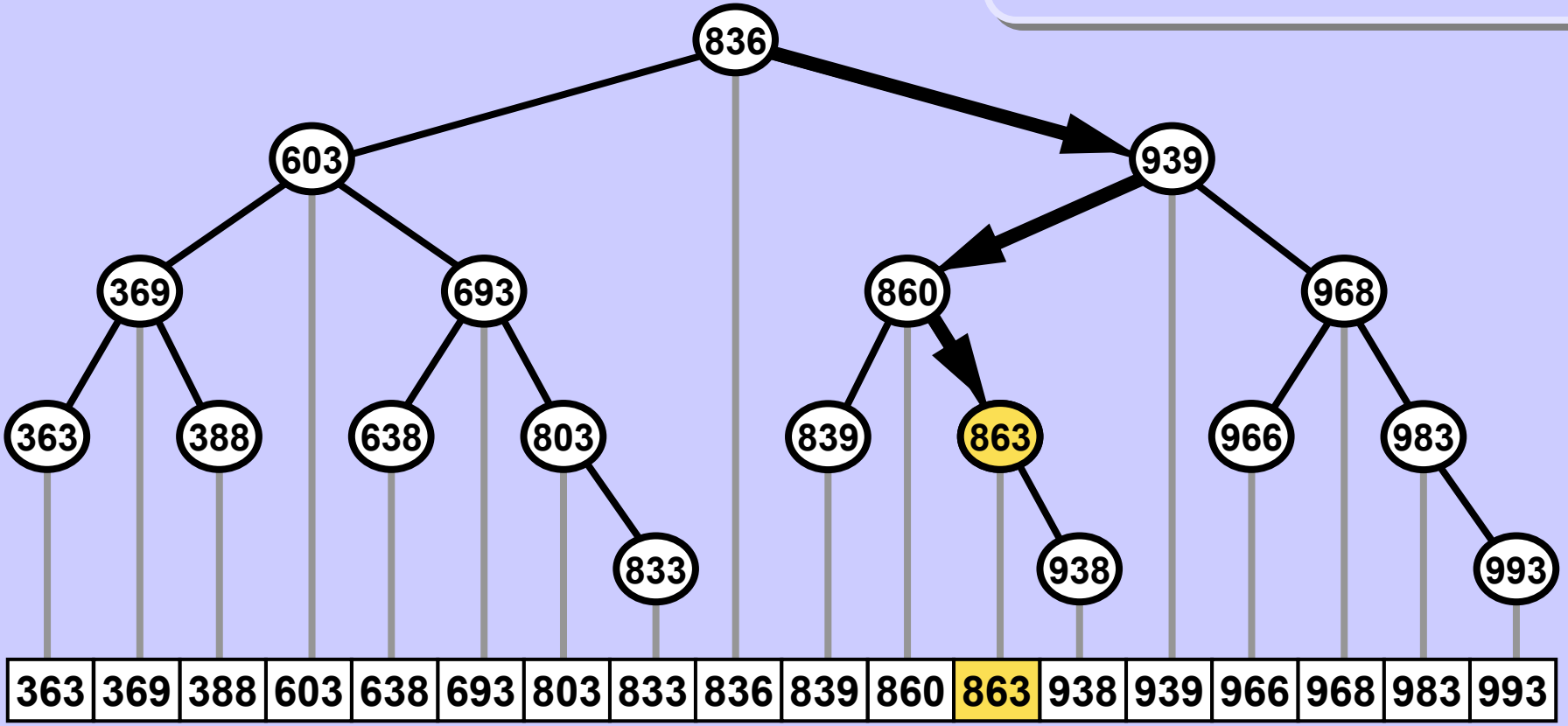
1 test



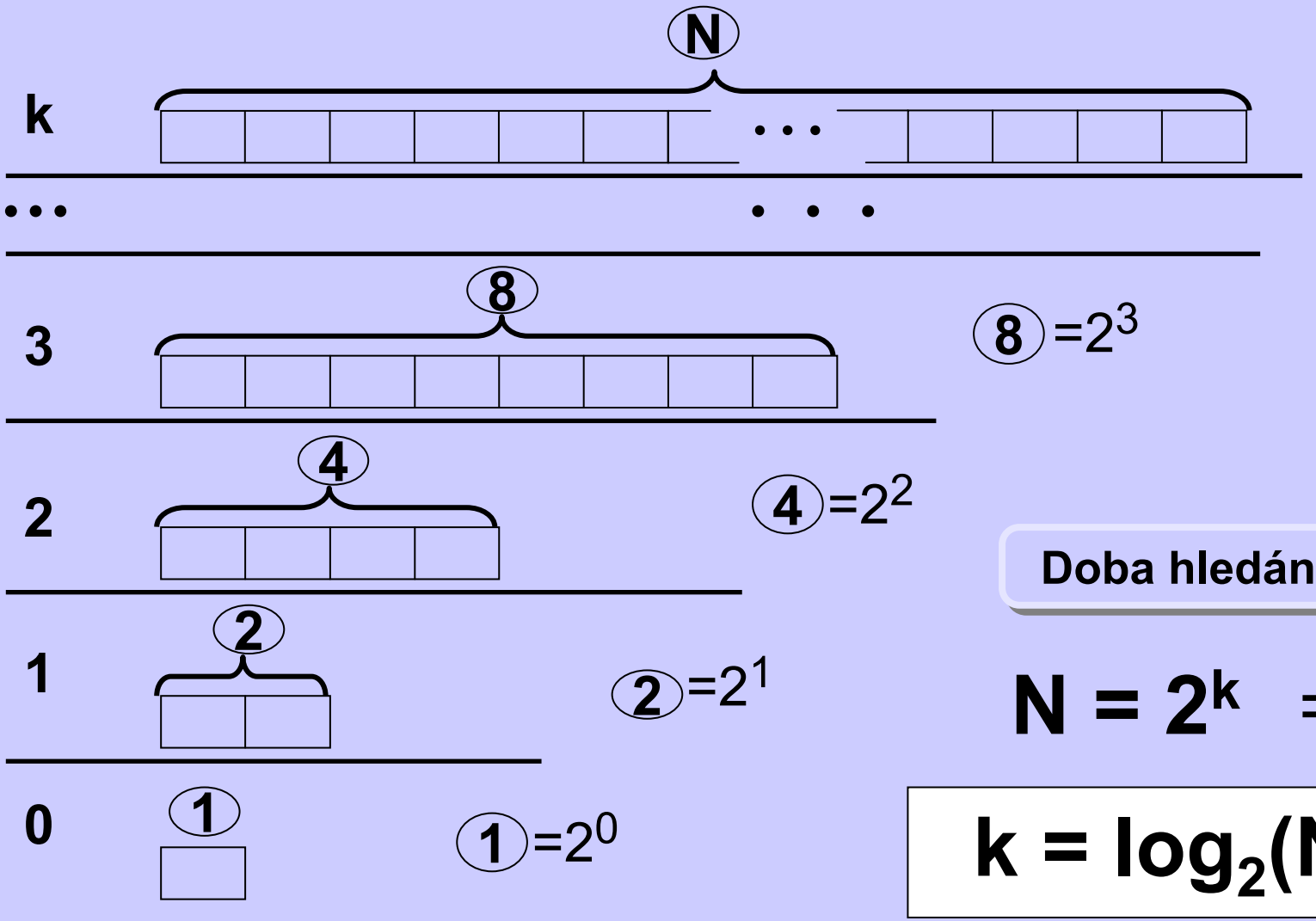
Hledání v seřazeném poli — binární, RYCHLEJŠÍ

najdi 863 !

Hledání kopíruje strukturu binárního stromu



Hledání v seřazeném poli — binární, RYCHLEJŠÍ



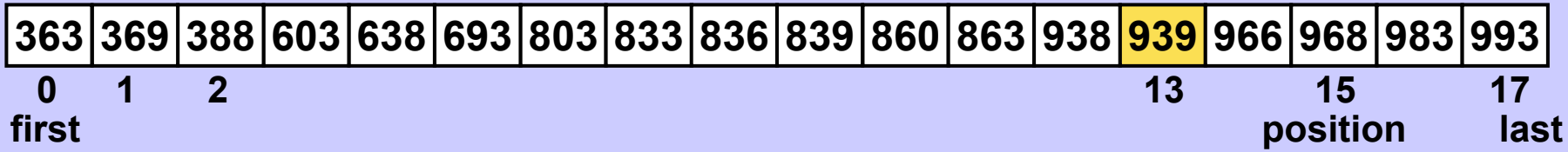
Doba hledání

$$N = 2^k \Rightarrow$$

$$k = \log_2(N)$$

Interpolační hledání

Pole a[ ] **Najdi q = 939**



**Jsou-li hodnoty v poli rozloženy víceméně pravidelně,
je možno použít lineární interpolaci.
Poloha prvku v poli by měla přibližně odpovídat jeho velikosti.**

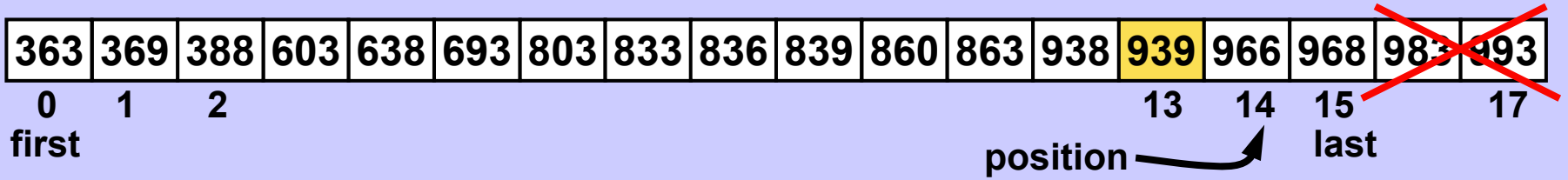
$$position = first + \frac{(q - a[first])}{a[last]-a[first]} * (last - first)$$

Celá část!

$$position = 0 + \frac{939 - 363}{993 - 363} * (17 - 0) = 15.54$$

Interpolační hledání

Pole a[ ] **Najdi q = 939**



Když se na vypočtené pozici prvek nenalézá, je buď vlevo nebo vpravo od ní a pak lze (rekurzivně) vzít za výchozí interval příslušnou levou nebo pravou část pole a výpočet opakovat.

$$position = first + \frac{(q - a[first])}{a[last]-a[first]} * (last - first)$$

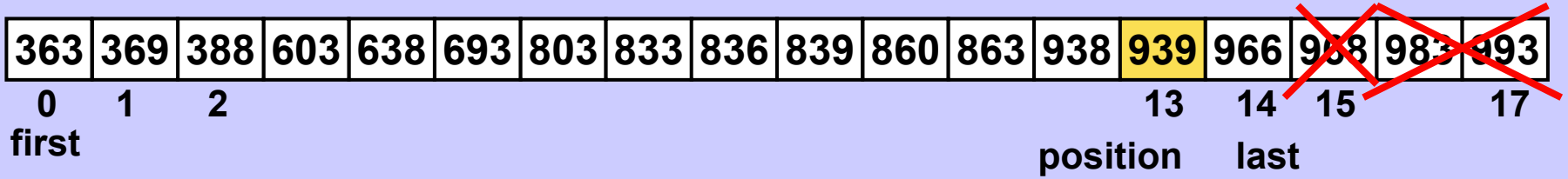
Celá část!

$$position = 0 + \frac{939 - 363}{968 - 363} * (15 - 0) = 14.12$$

Interpolační hledání

Pole a[ ]

Najdi q = 939



Když se na vypočtené pozici prvek nenalézá, je buď vlevo nebo vpravo od ní a pak lze (rekurzivně) vzít za výchozí interval příslušnou levou nebo pravou část pole a výpočet opakovat.

$$position = first + \frac{(q - a[first])}{a[last]-a[first]} * (last - first)$$

Celá část!

$$position = 0 + \frac{939 - 363}{966 - 363} * (14 - 0) = 13.37$$

Hotovo

Interpolační hledání

```
int interpol(int [] arr, int q) {
  int first = 0;
  int last = length(arr);
  do {
     pos = first + round((q–arr[first])/(arr[last]–arr[first])
                         *(last-first) );
     if (arr[pos] < q)       first = pos+1;  // check left side
     else if (arr[pos] > q)  last = pos-1;  //check right side
   } while ((arr[pos] != q) && (first < last));
   return pos;
}
```

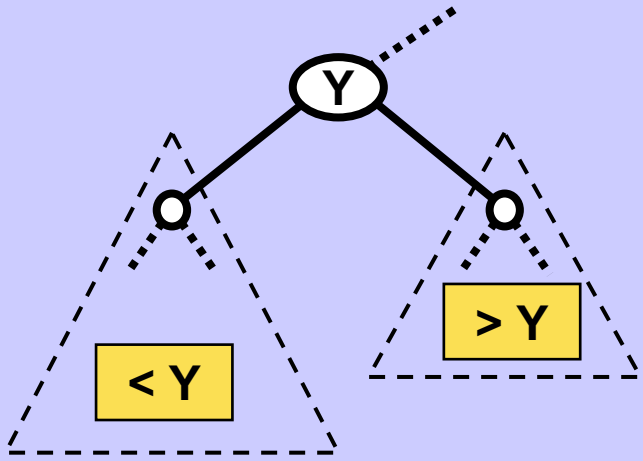
Hledání v seřazeném poli — srovnání rychlostí

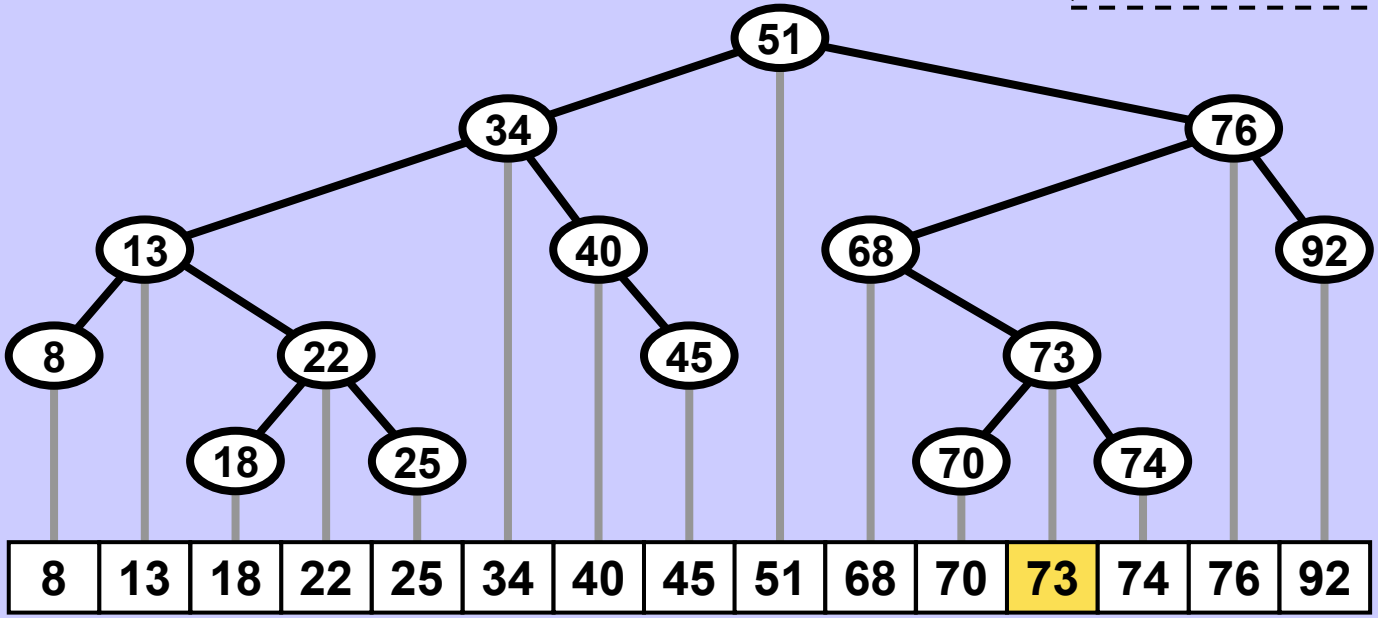
Velikost pole N	Lineární hledání průměrný případ	Interpolační hledání průměrný případ	Binární hledání cca průměrný /nejhorší případ	
10	5.5	1.60	7	9
30	15.5	2.12	9	11
100	50.5	2.56	13	15
300	150.5	2.89	17	19
1 000	500.5	3.18	19	21
3 000	1 500.5	3.41	23	25
10 000	5 000.5	3.63	27	29
30 000	15 000.5	3.80	29	31
100 000	50 000.5	3.96	33	35
300 000	150 000.5	4.11	37	39
1 000 000	500 000.5	4.24	39	41
	Zřejmě $\Theta(n)$	Na náhodném rovnoměrném rozdělení, teoreticky $\Theta(\log(\log(N)))$	Podle struktury binárního stromu $\Theta(\log(n))$	

Binární vyhledávací strom

V levém podstromu každého uzlu jsou všechny klíče menší.

V pravém podstromu každého uzlu jsou všechny klíče větší.

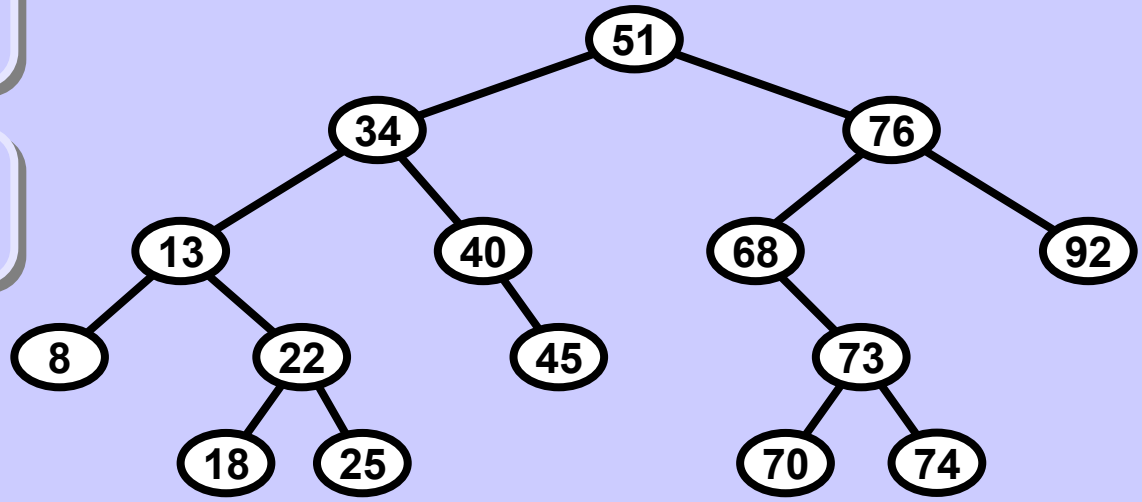




Binární vyhledávací strom

BVS nemusí být
a nebývá vyvážený.

BVS nemusí být
a nebývá pravidelný.

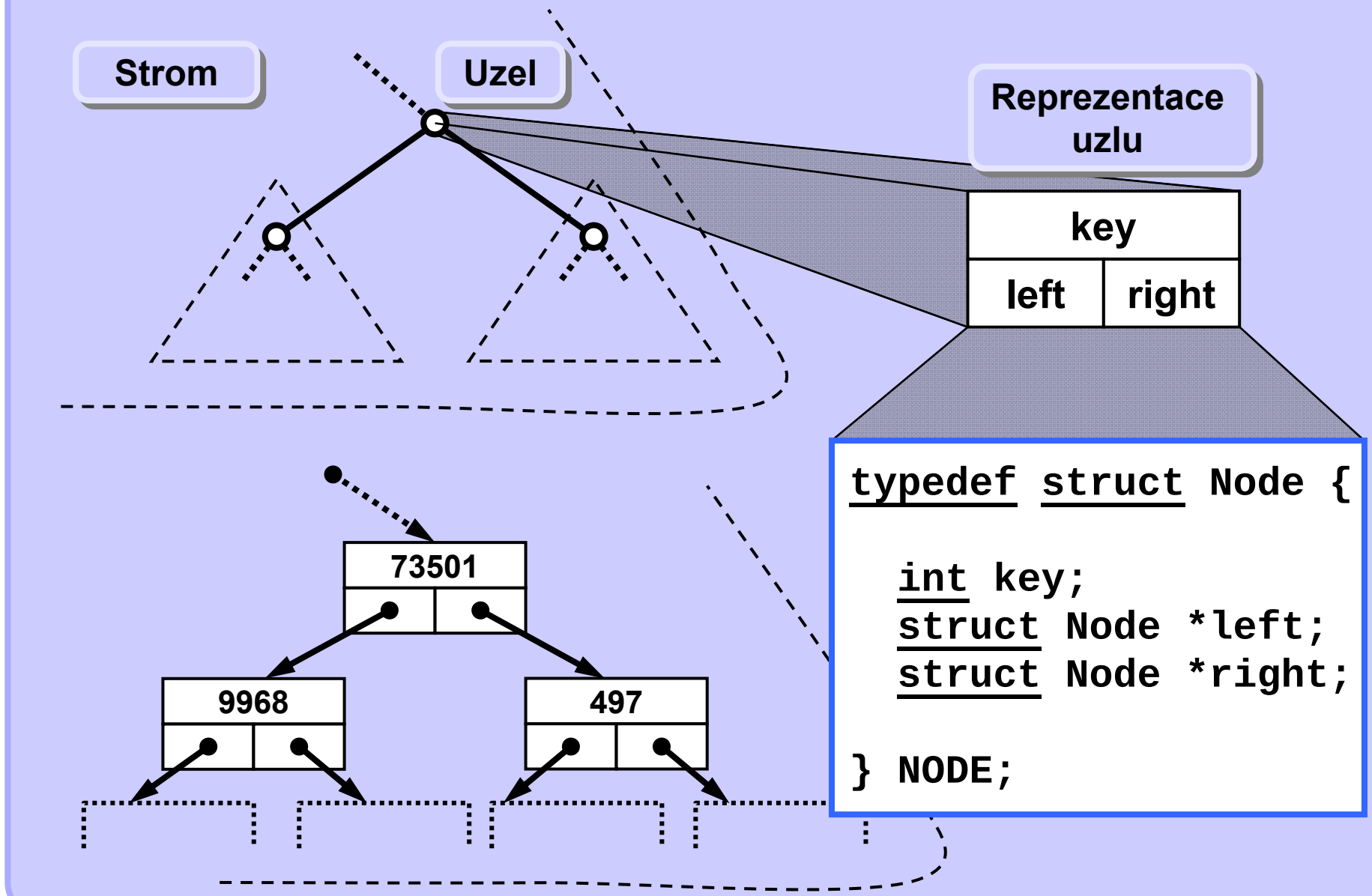


Výpisem prvků BVS
v pořadí INORDER
získáme uspořádané
hodnoty klíčů.

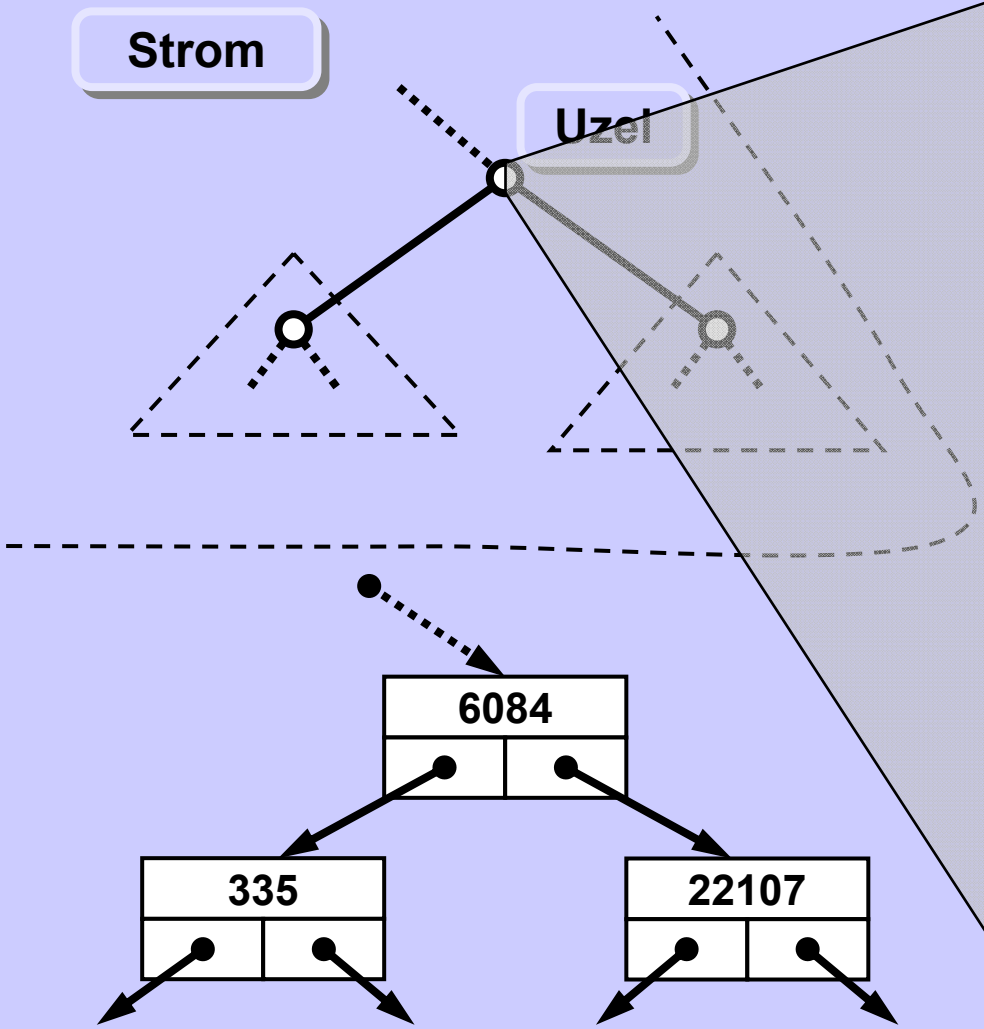
BVS je flexibilní díky operacím:

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

Implementace binárního stromu -- C



Implementace binárního stromu -- java

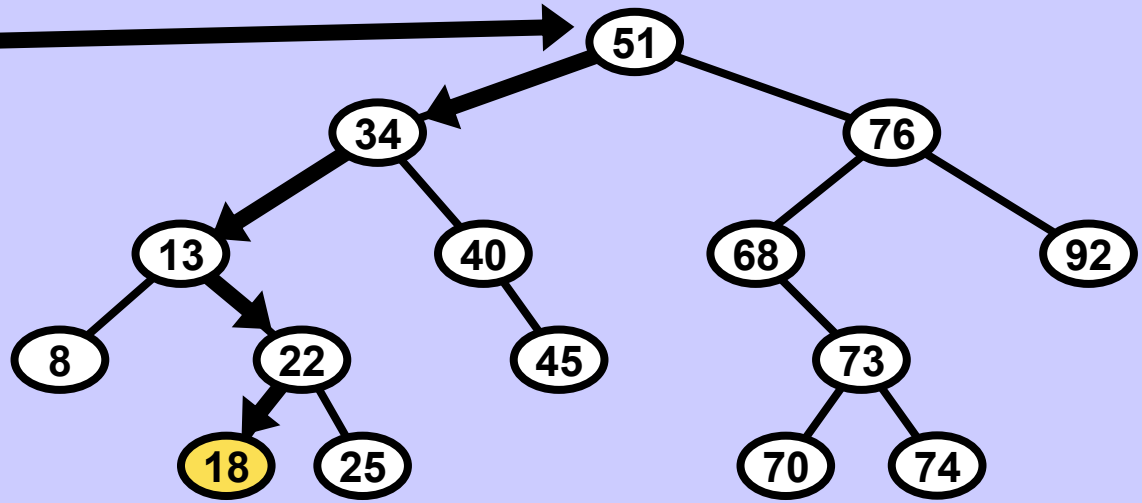



```
public class Node {
  public Node left;
  public Node right;
  public int key;
  public Node(int k) {
    key = k;
    left = null;
    right = null;
  }
}

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

Operace Find v BVS

Najdi 18



Každá operace se stromem začíná v kořeni.

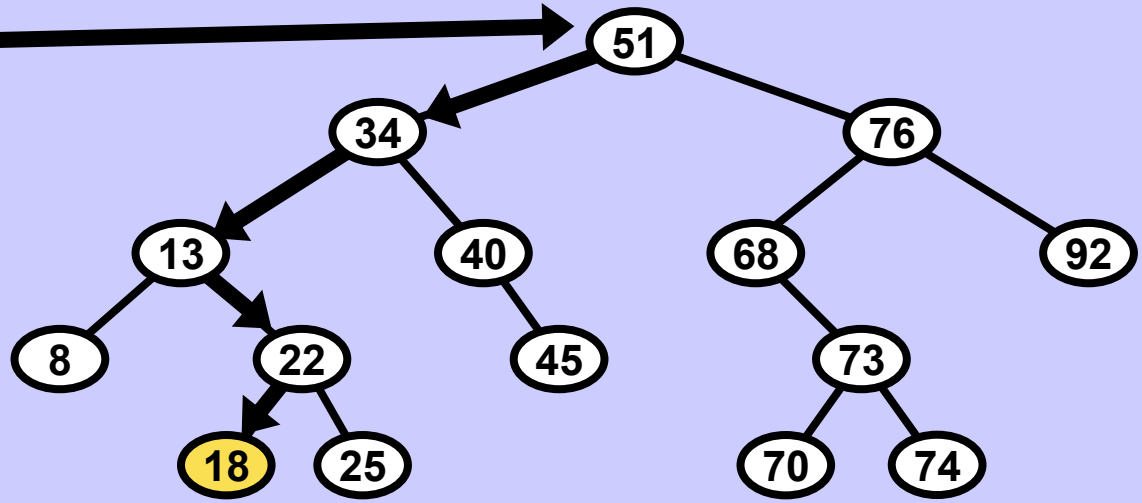
Iterativně

```
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

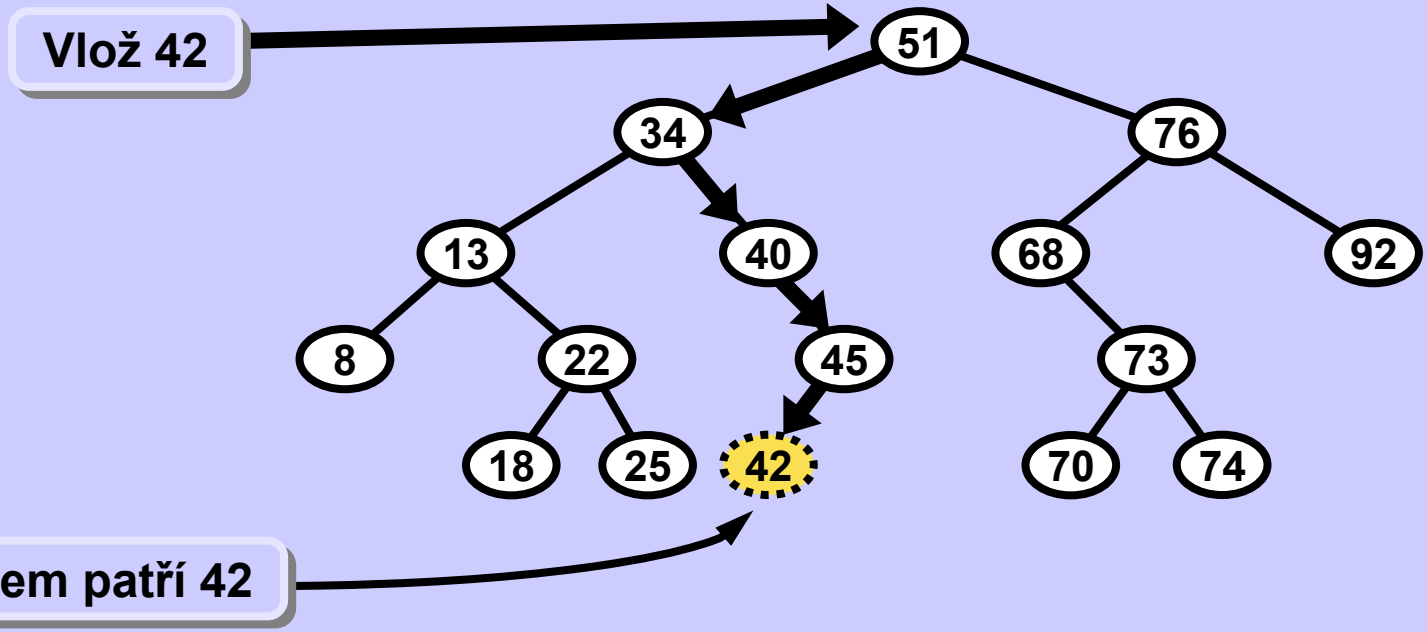
Každá operace se stromem začíná v kořeni.



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; }
} }
```

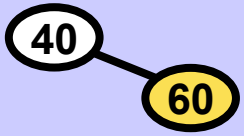
Operace Insert v BVS rekurzivně

```
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);
}
```

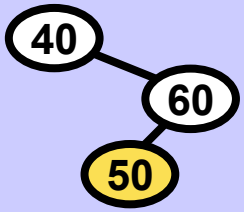
Stavba BVS operací Insert

insert 40

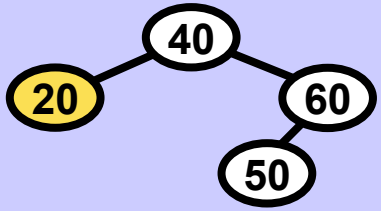
insert 60



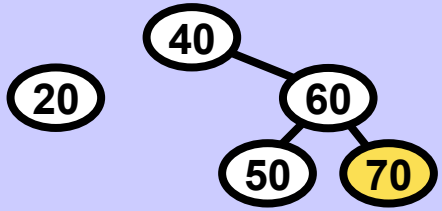
insert 50



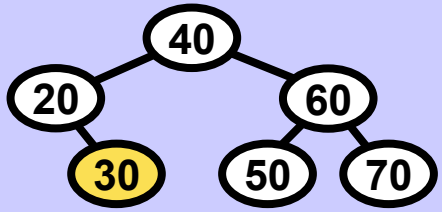
insert 20



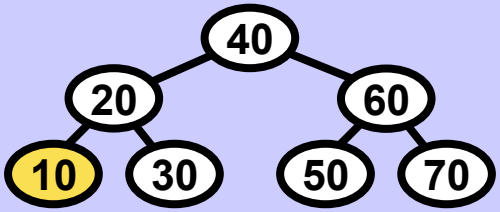
insert 70



insert 30



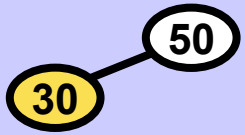
insert 10



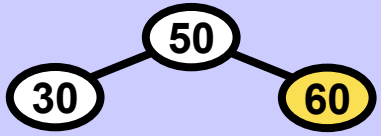
Tvar budovaného BVS závisí na pořadí vkládání dat.

insert 50

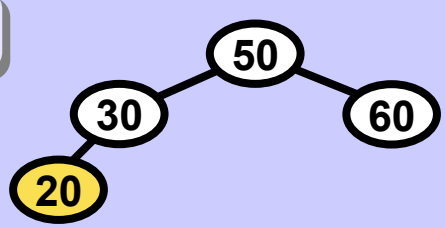
insert 30



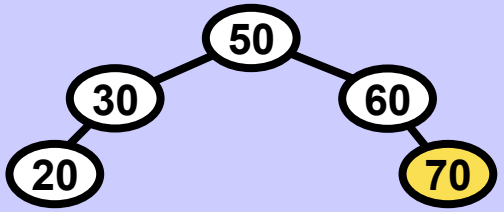
insert 60



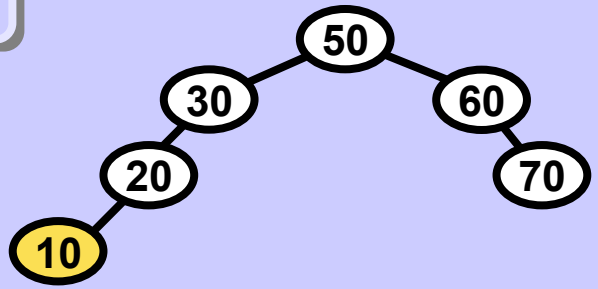
insert 20



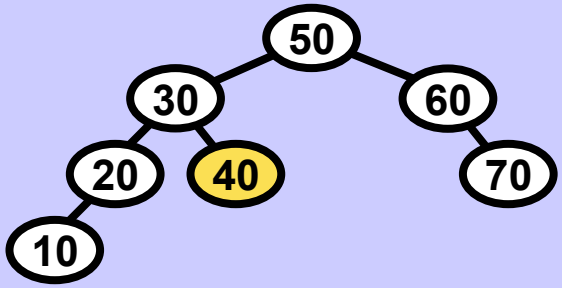
insert 70



insert 10

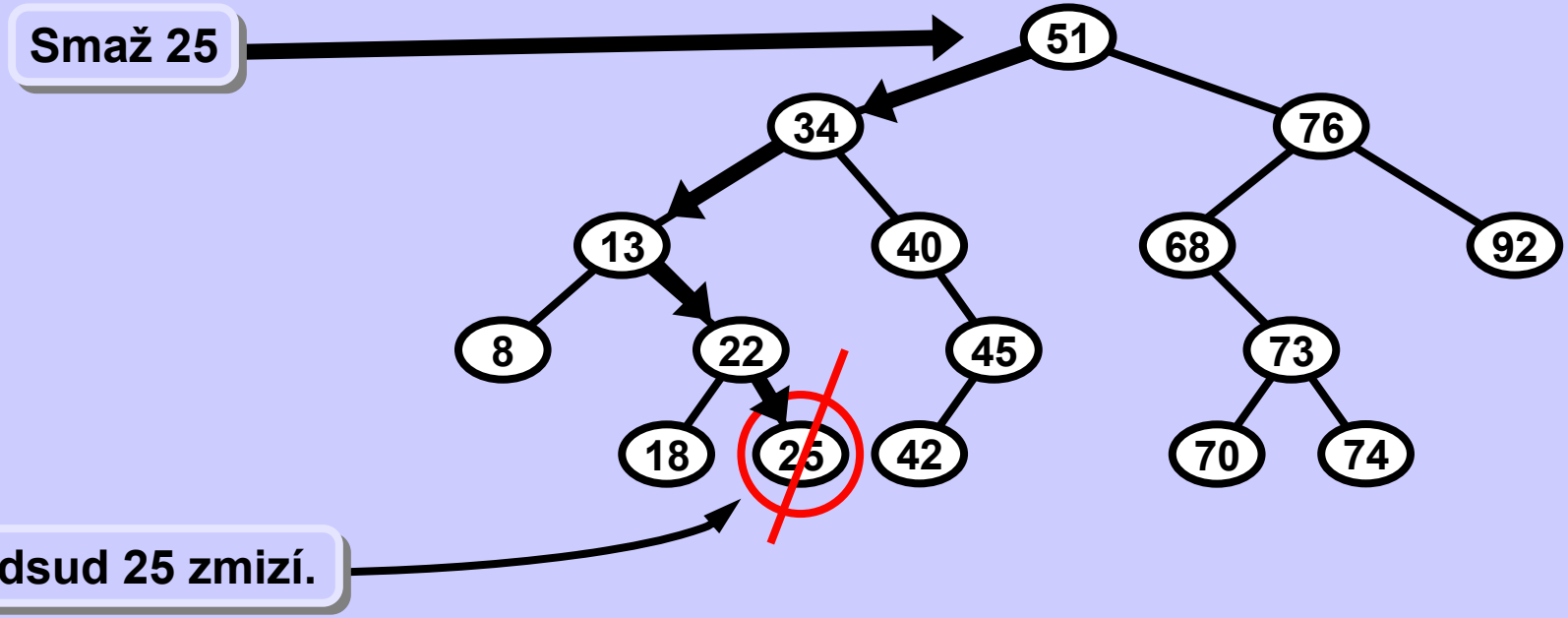


insert 40



Operace Delete v BVS (I.)

Smazání uzlu s 0 potomky (= listu)

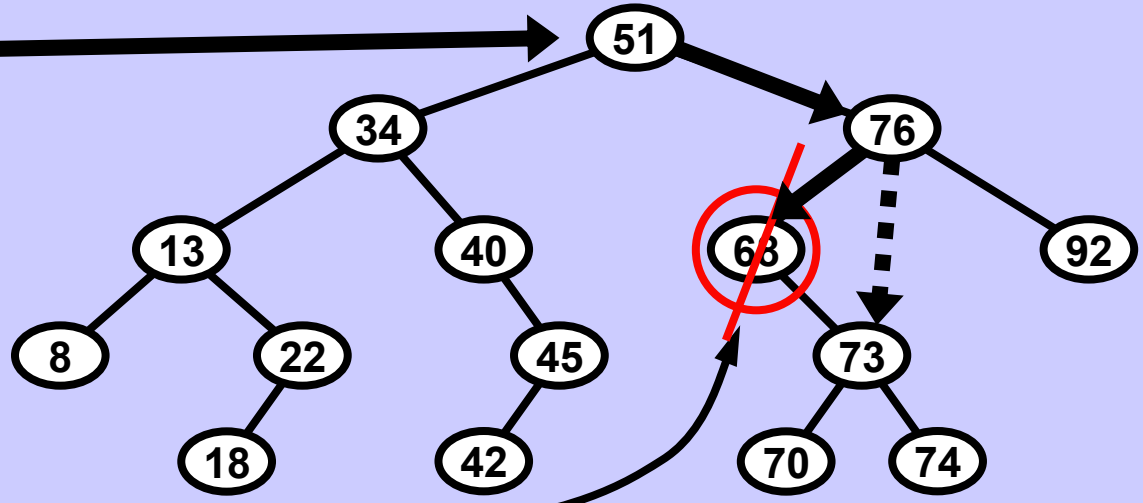


Delete I.
Najdi daný uzel (jako ve Find) a odstraň ukazatel na něj z jeho rodiče.

Operace Delete v BVS (II.)

Smazání uzlu s 1 potomkem

Smaž 68



Odsud 25 zmizí.

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

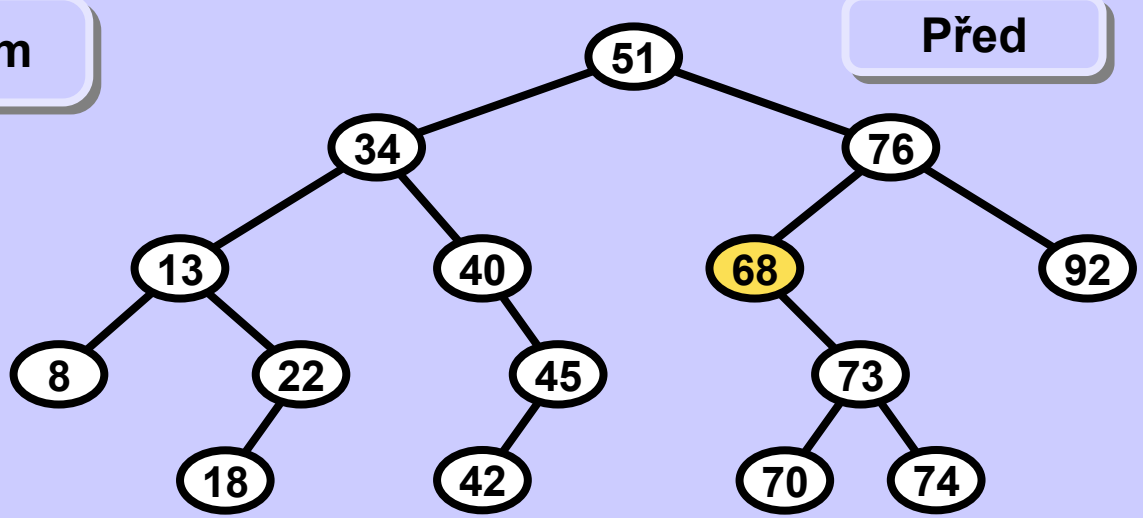
Delete II.
Najdi daný uzel (jako ve Find) a ukazatelem z jeho rodiče na něj ukaž na jeho (jediného!) potomka.

Operace Delete v BVS (II.)

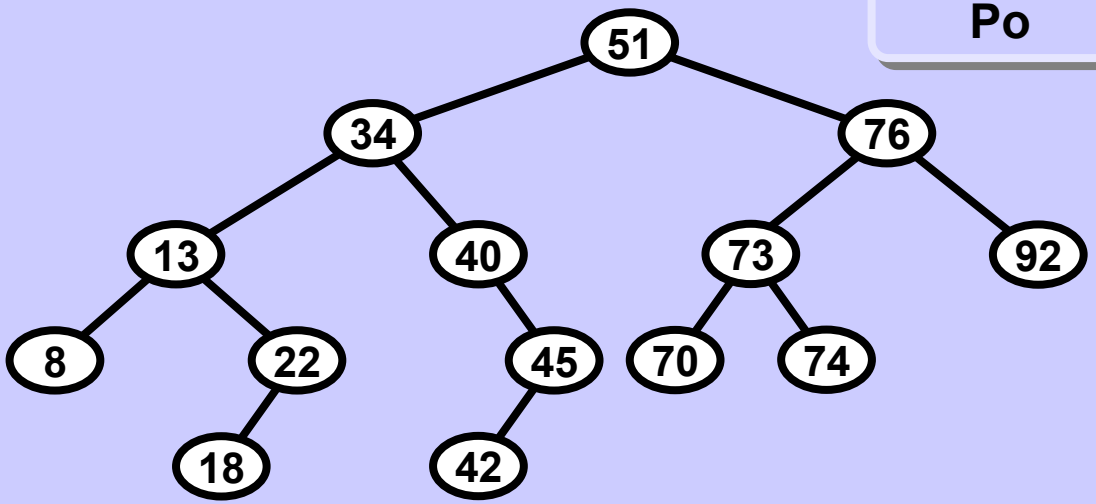
Smazání uzlu s 1 potomkem

Smaž 68

Před

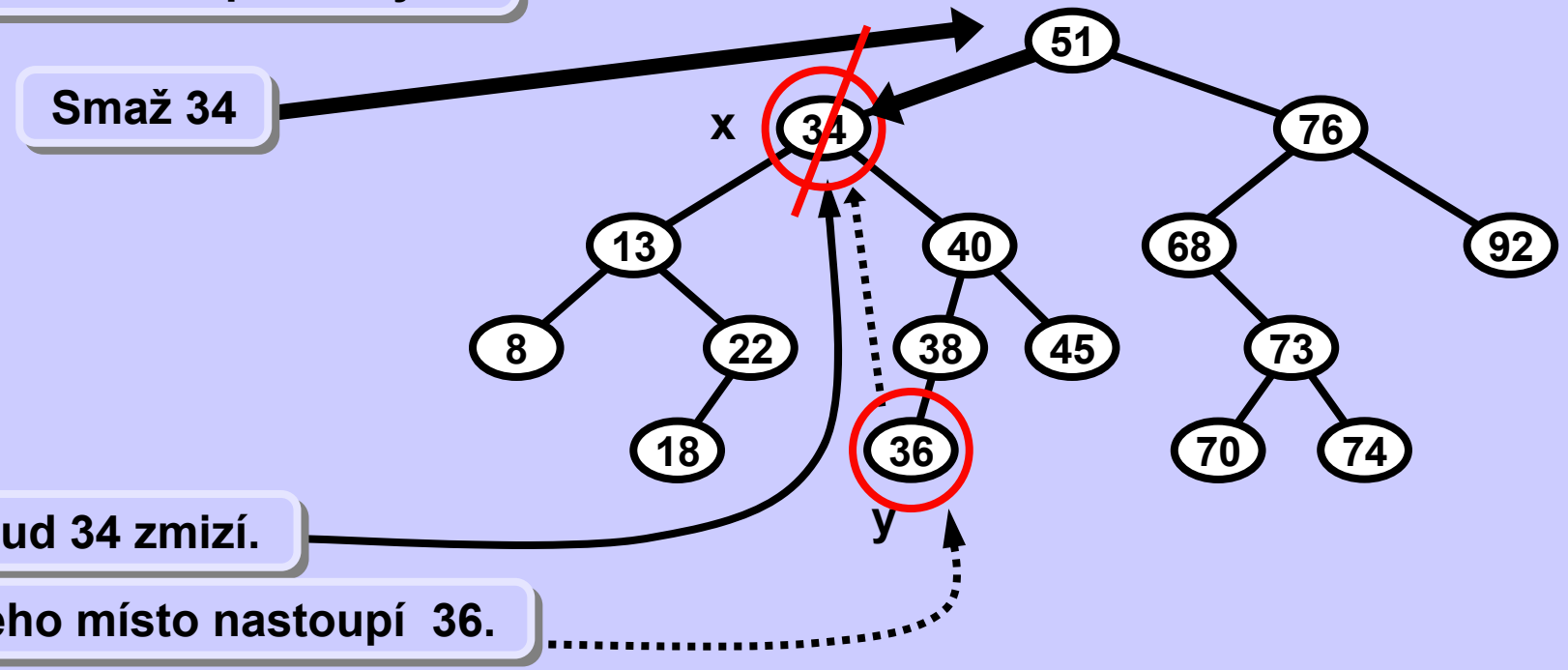


Po



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.

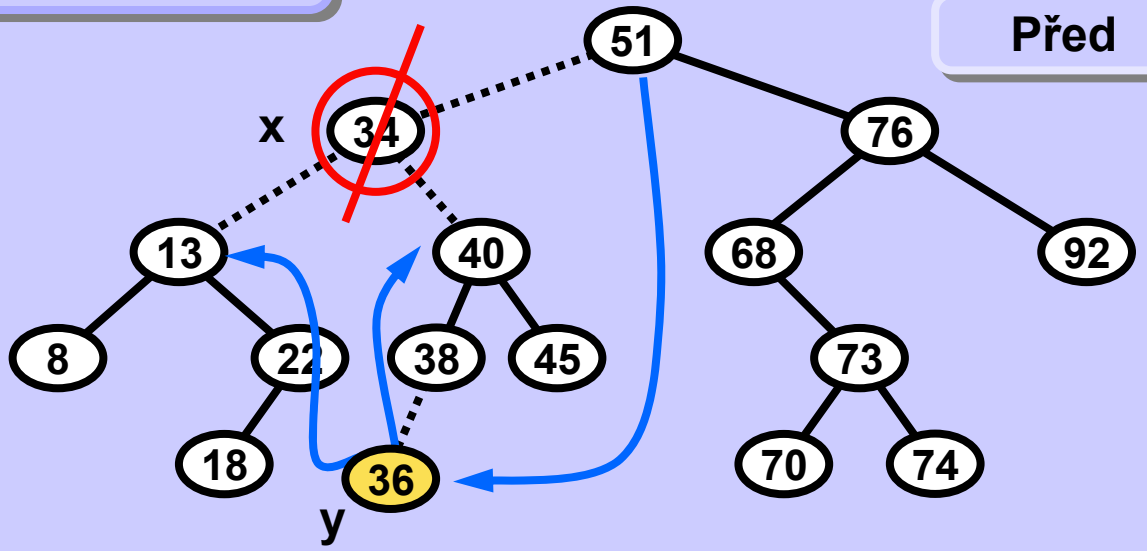
Operace Delete v BVS (IIIa.)

Smaž 34

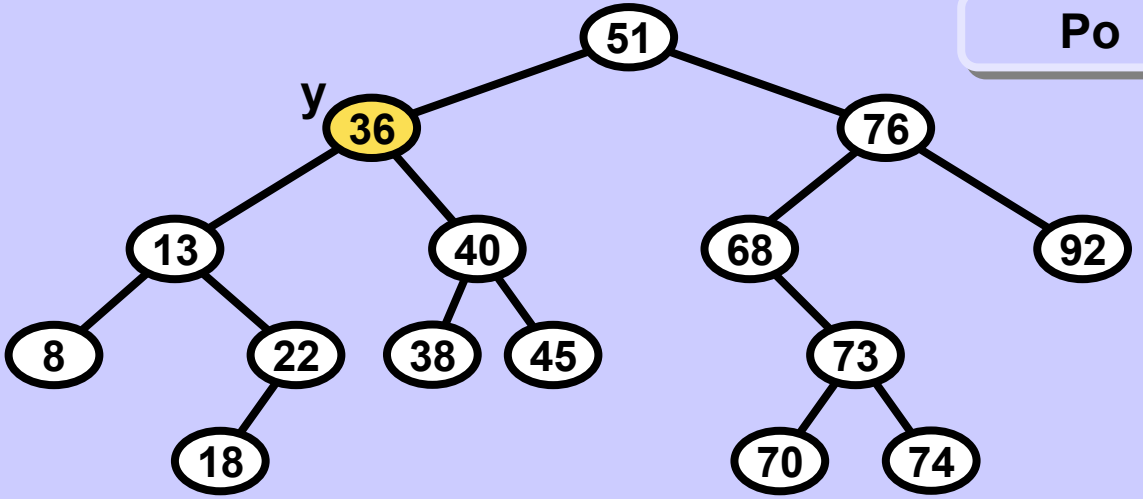
zanikající hrany/ukazatele/reference .....

vznikající hrany/ukazatele/reference ——→

Před

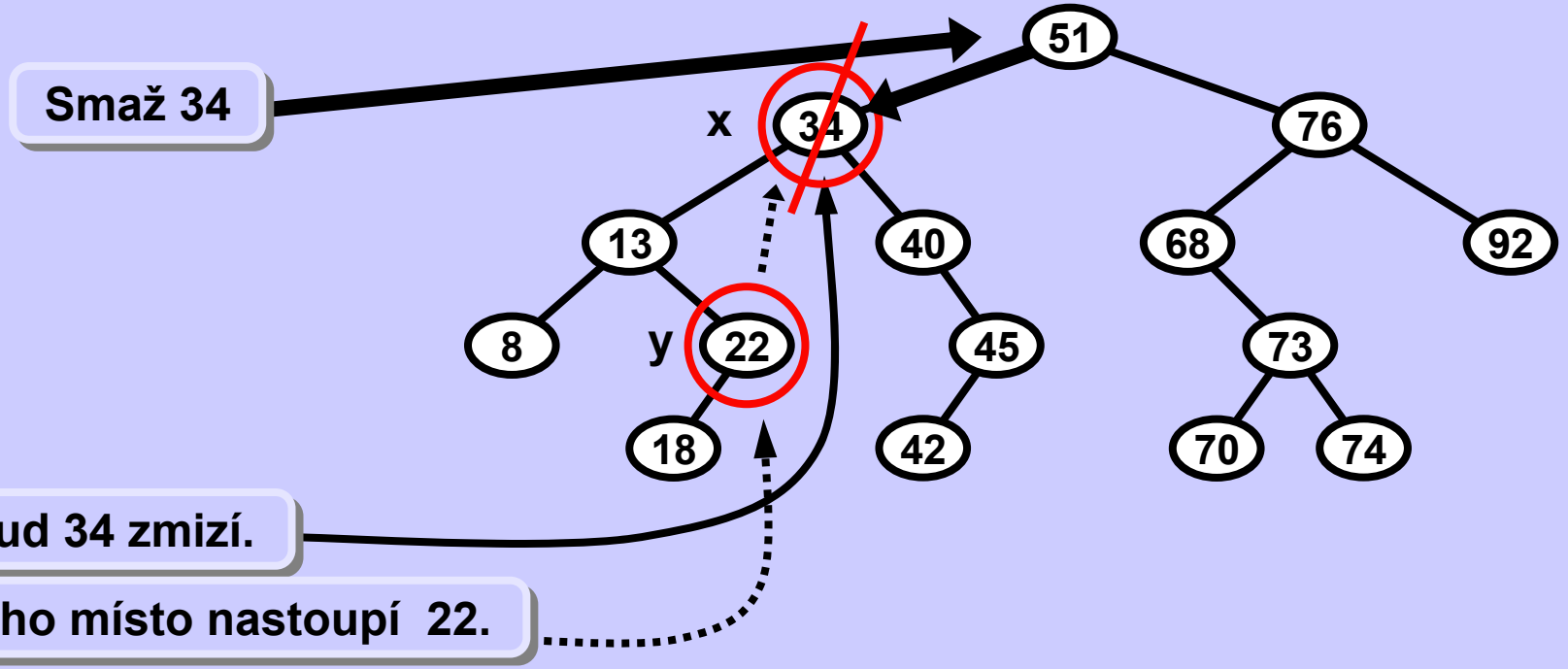


Po



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

Smazání uzlu s 2 potomky



Delete IIIb.

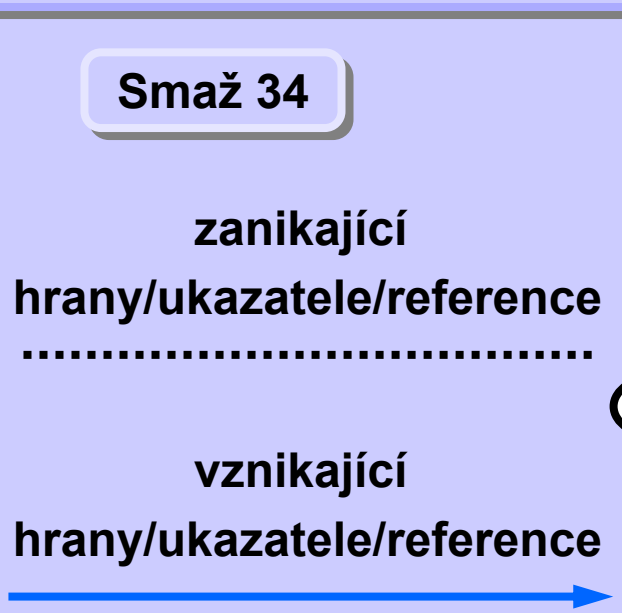
1. Najdi daný uzel x (jako ve Find) a dále
   najdi ***nepravější*** (=největší klíč) uzel y v ***levém*** podstromu x.
2. Z uzlu y ukaž na potomky uzlu x, z rodiče y ukaž na potomka y místo y,
   z rodiče x ukaž na y.

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

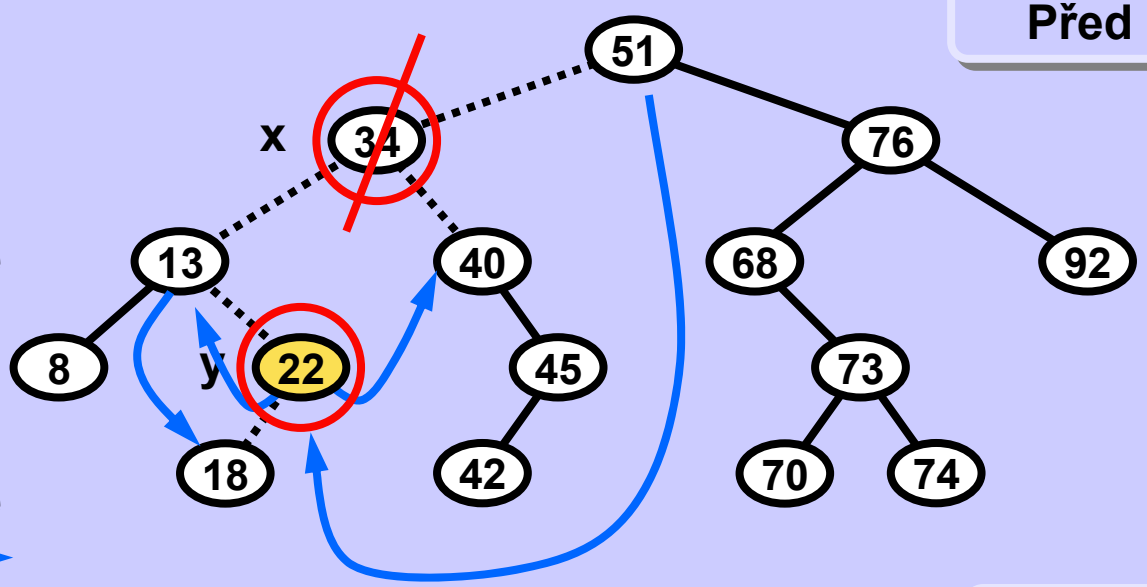
Smaž 34

zanikající hrany/ukazatele/reference

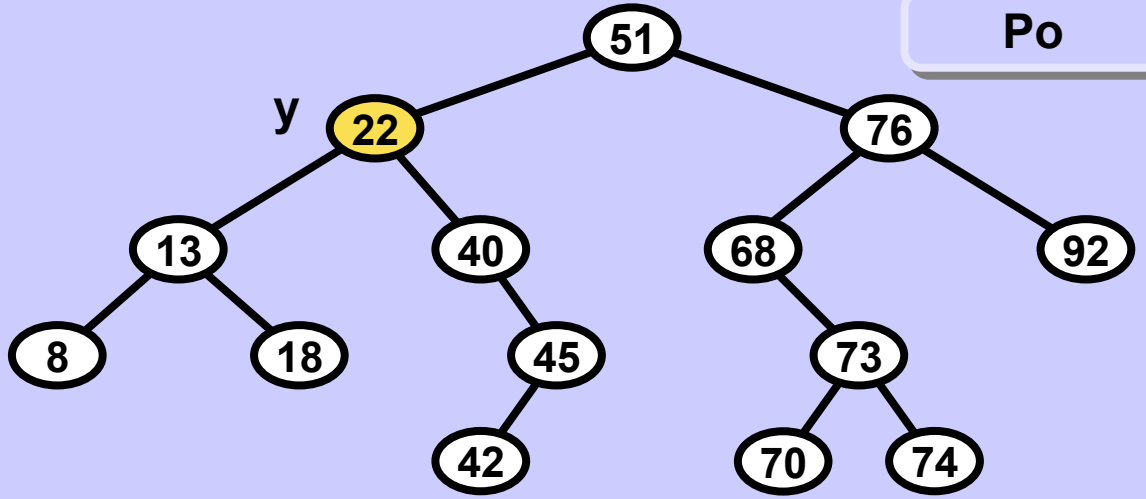
vznikající hrany/ukazatele/reference



Před



Po



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...
}
```

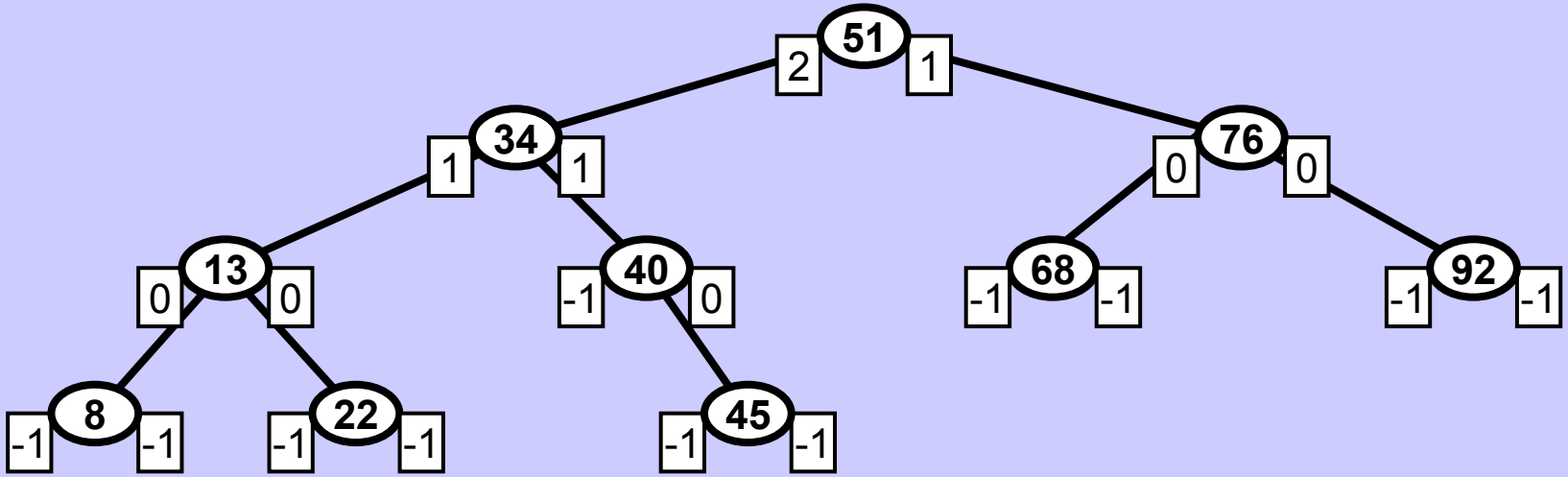
Asymptotické složitosti operací Find, Insert, Delete v BVS

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

AVL strom

AVL strom je BVS s přidanými vlastnostmi, které jej udržují (téměř) vyvážený.

AVL má také operace Find, Insert, Delete.



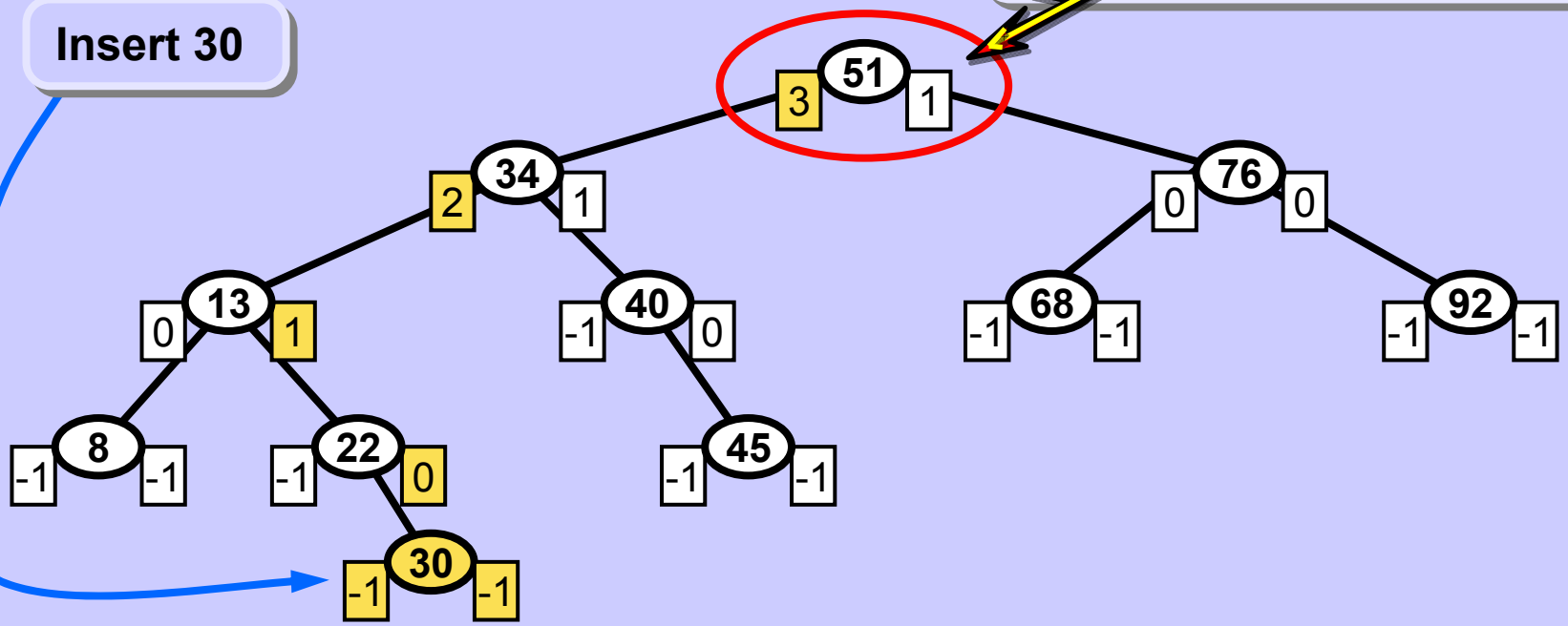
Každý uzel registruje hloubku svého levého a pravého podstromu, hloubka prázdného stromu je -1.

V každém uzlu je rozdíl výšek obou podstromů roven -1, 0, 1.

Vložení uzlu může způsobit rozvážení AVL stromu.

V každém uzlu je rozdíl výšek obou podstromů roven -1, 0, 1 !!

Insert 30

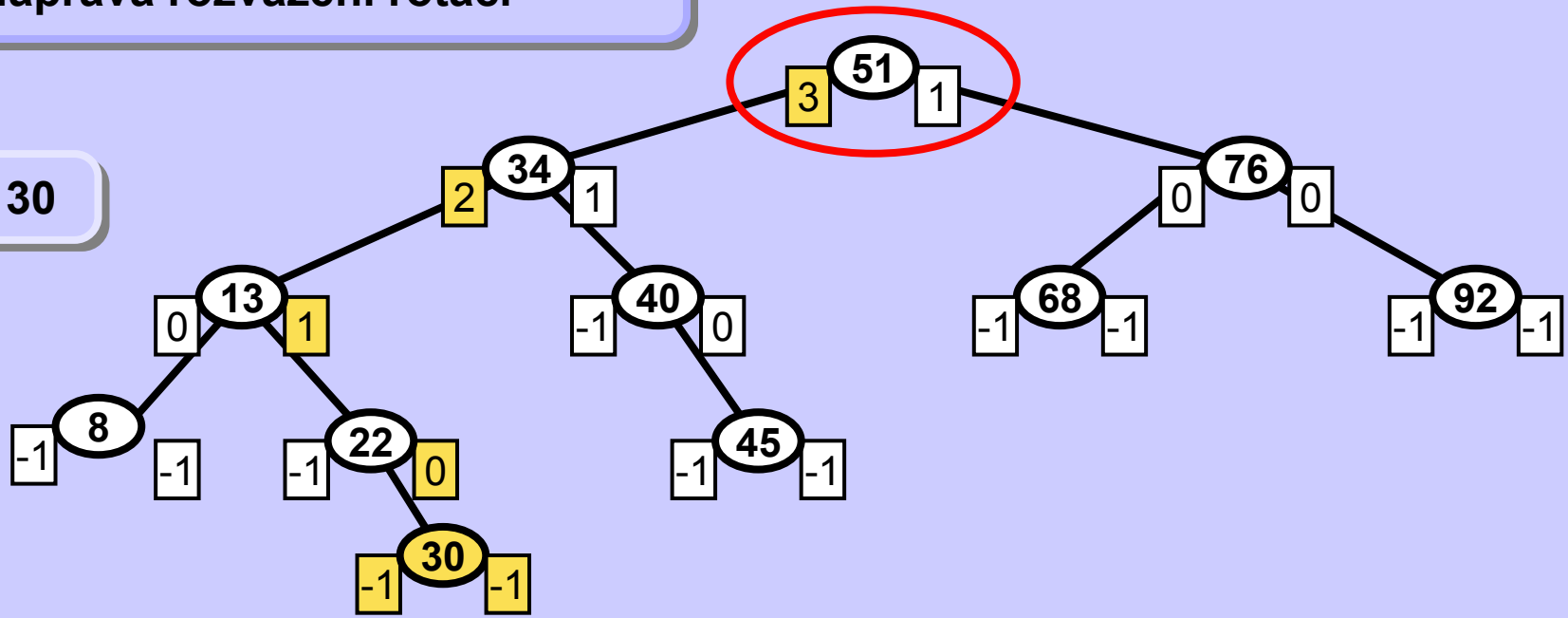


Změněné hloubky

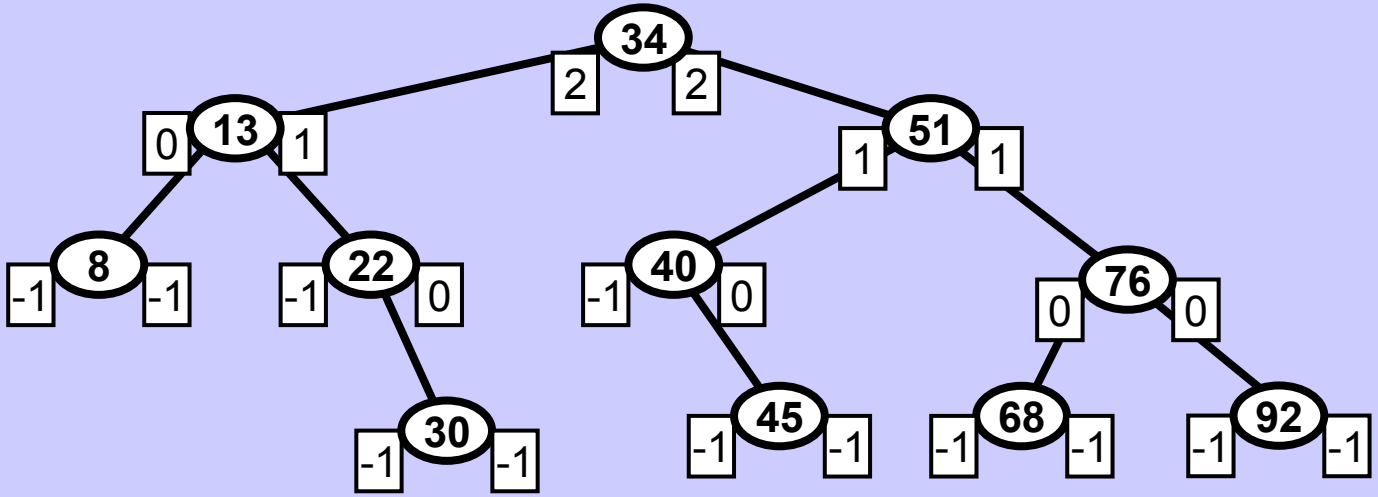
Levý podstrom uzlu 51 je příliš hluboký, strom přestal být AVL.

Náprava rozvážení rotací

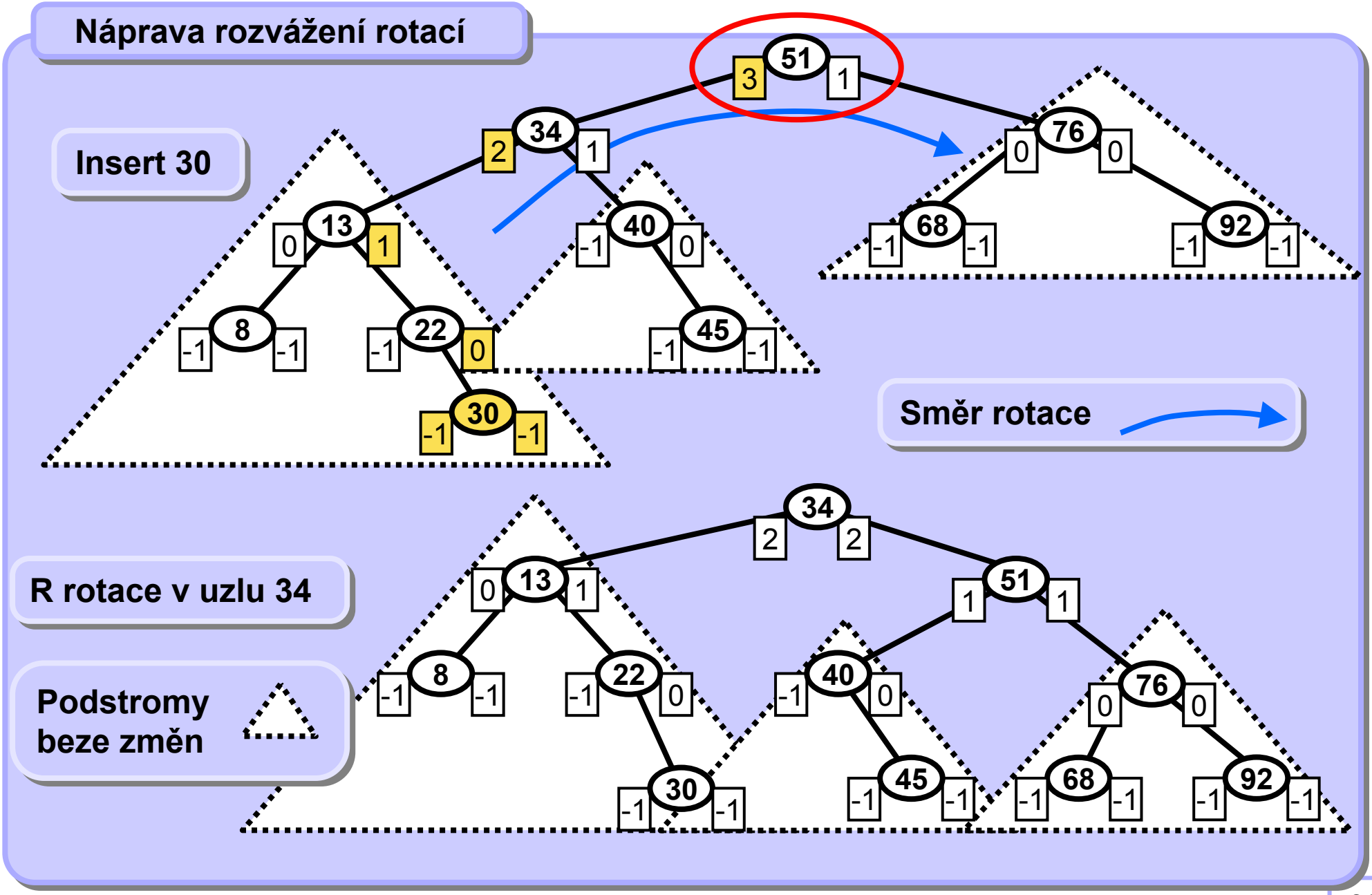
Insert 30



Vyvážený strom po pravé jednoduché rotaci, tzv. R rotaci

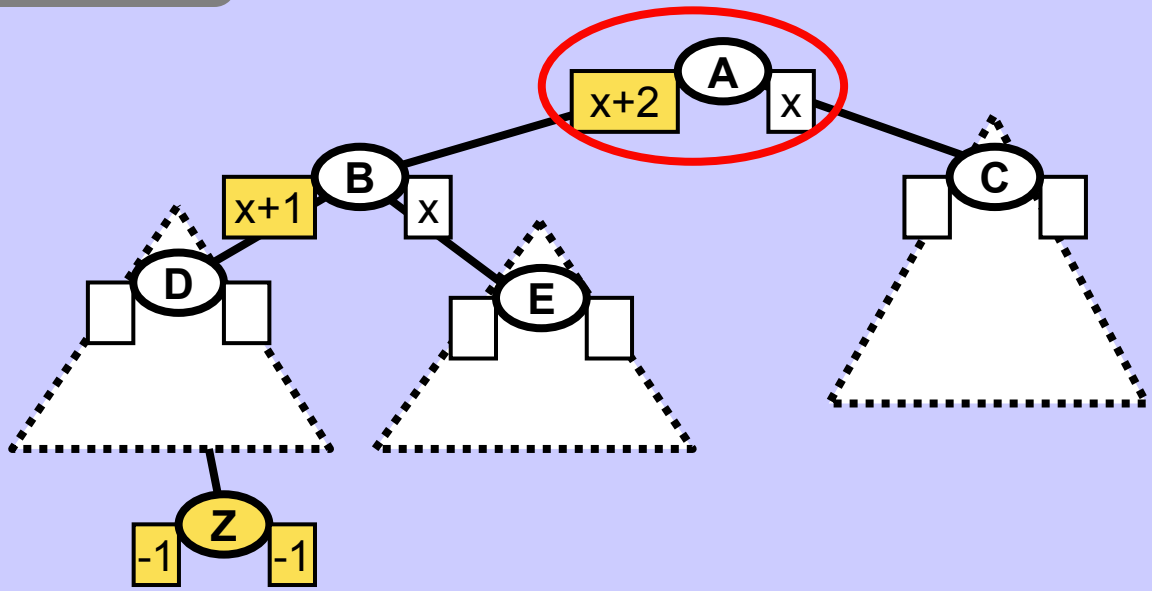


Náprava rozvážení rotací

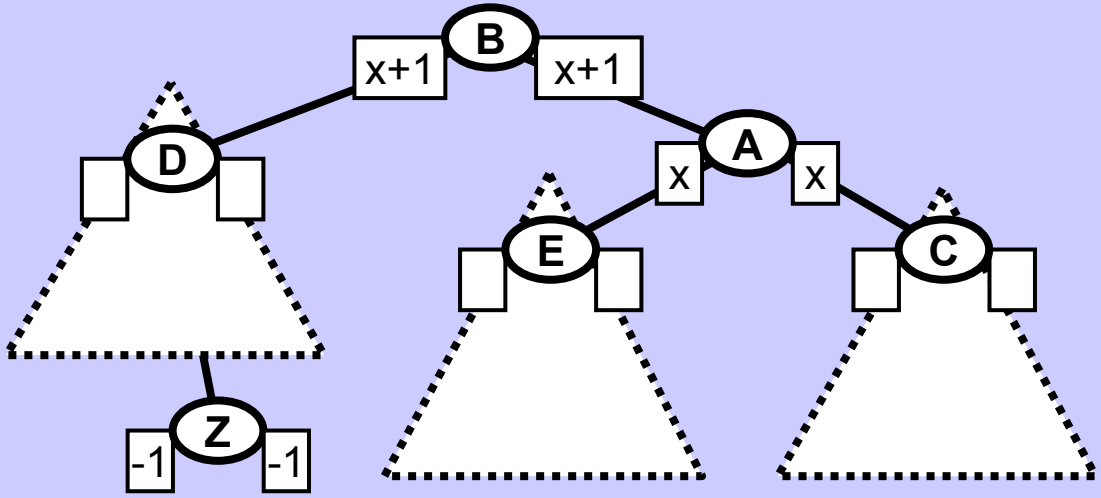


Rotace R obecně

Před

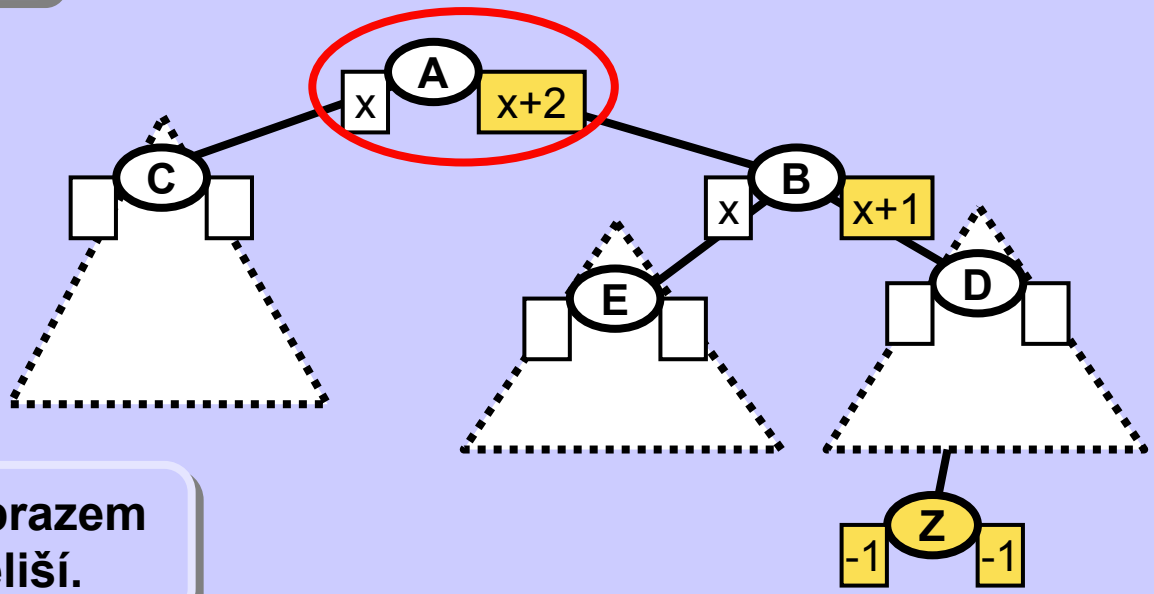


Po



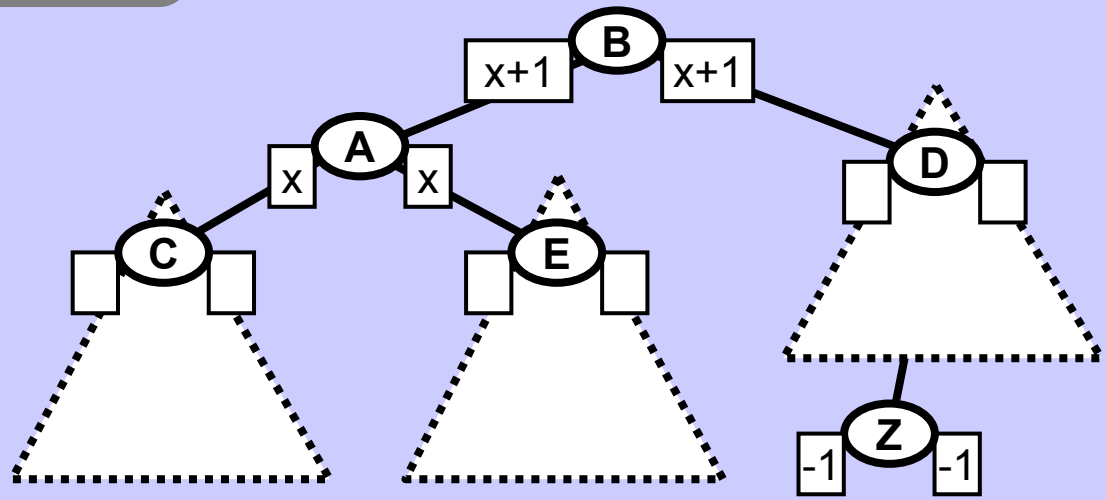
Rotace L obecně

Před

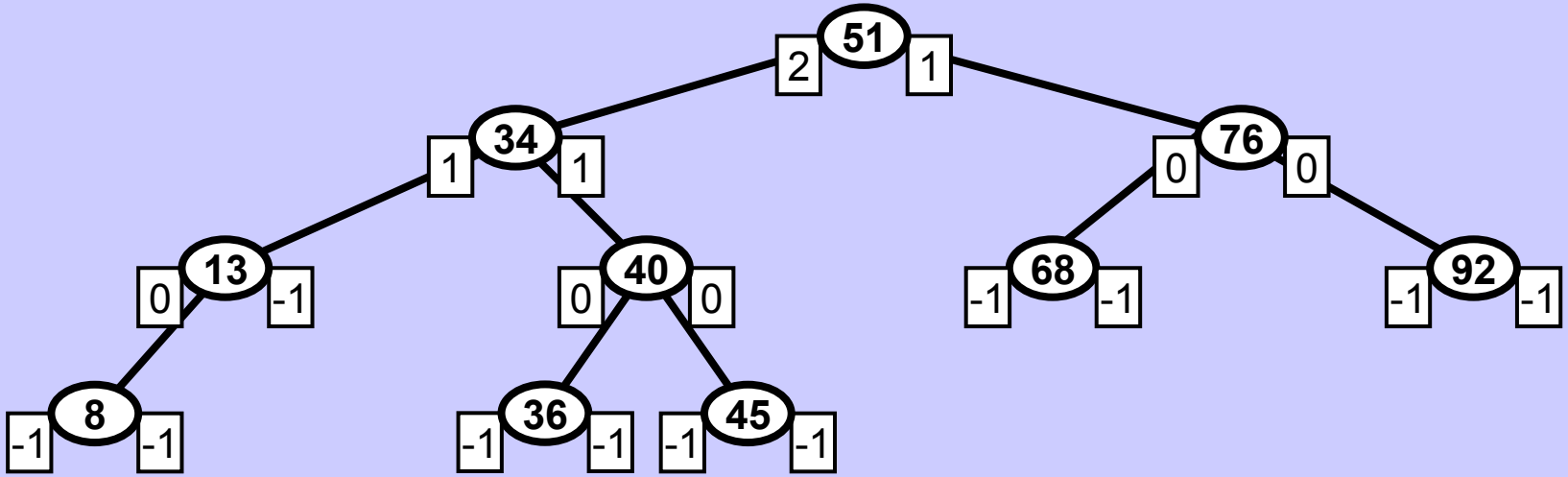


Rotace L je symetrickým obrazem rotace R, jinak se od ní neliší.

Po



AVL strom

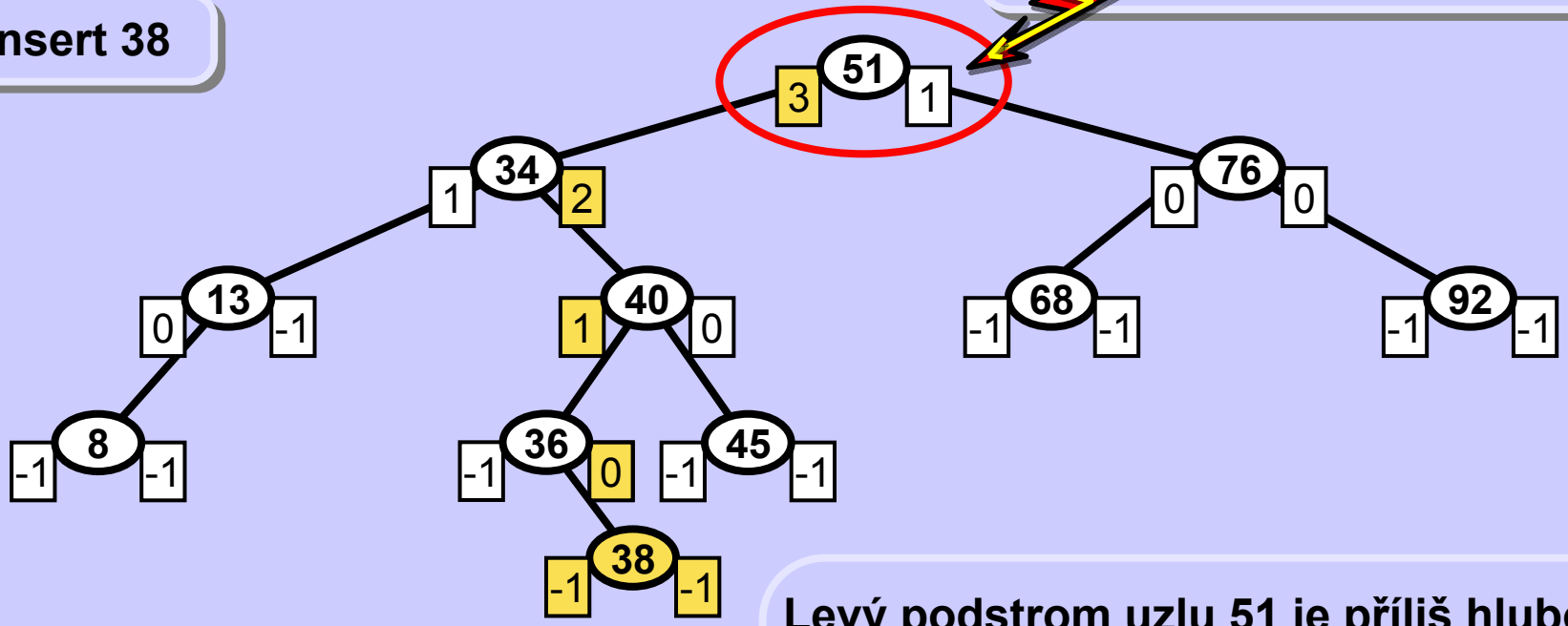


Strom pro demonstraci LR rotace

Vložení uzlu může způsobit rozvážení stromu.

V každém uzlu je rozdíl výšek obou podstromů roven -1, 0, 1 !!

Insert 38

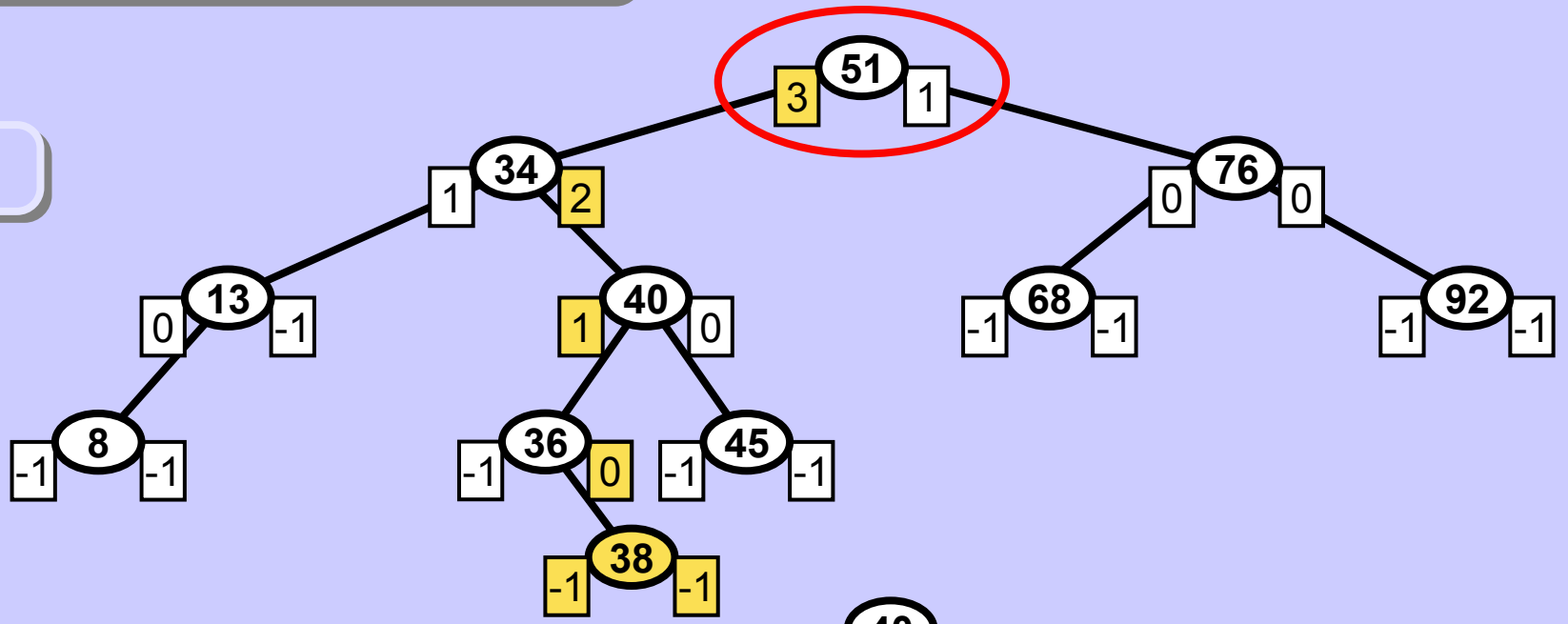


Levý podstrom uzlu 51 je příliš hluboký, strom přestal být AVL.
Použití rotace R by nepomohlo, příliš hlubokým by se stal pravý podstrom uzlu 34.

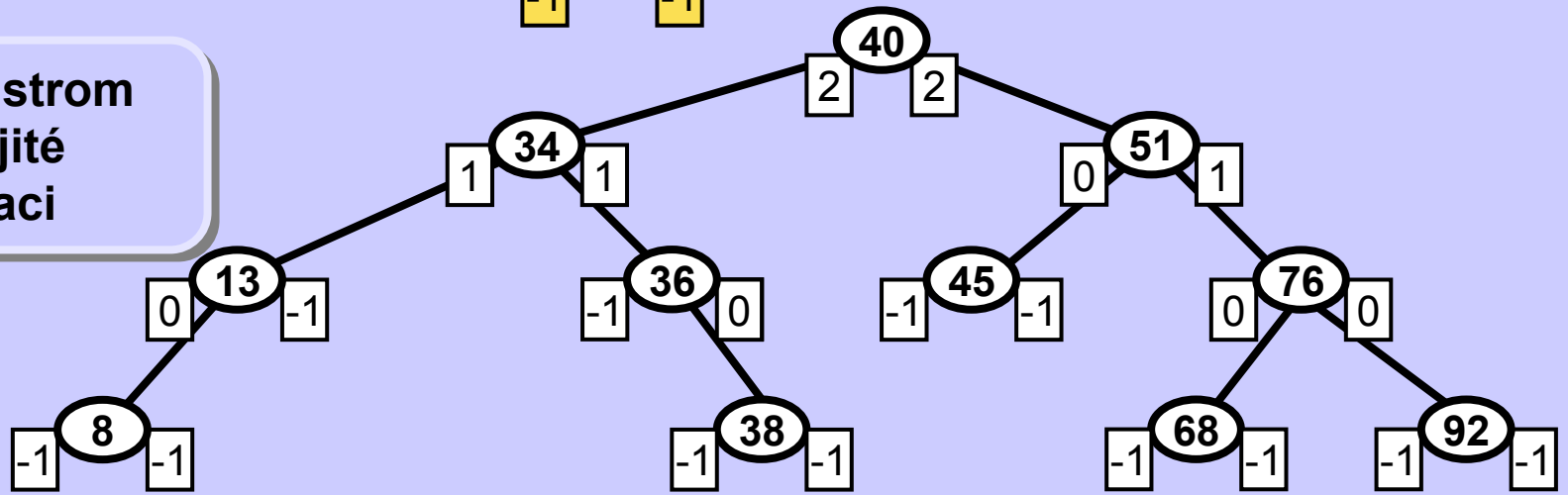
Změněné hloubky

Náprava rozvážení rotací

Insert 38

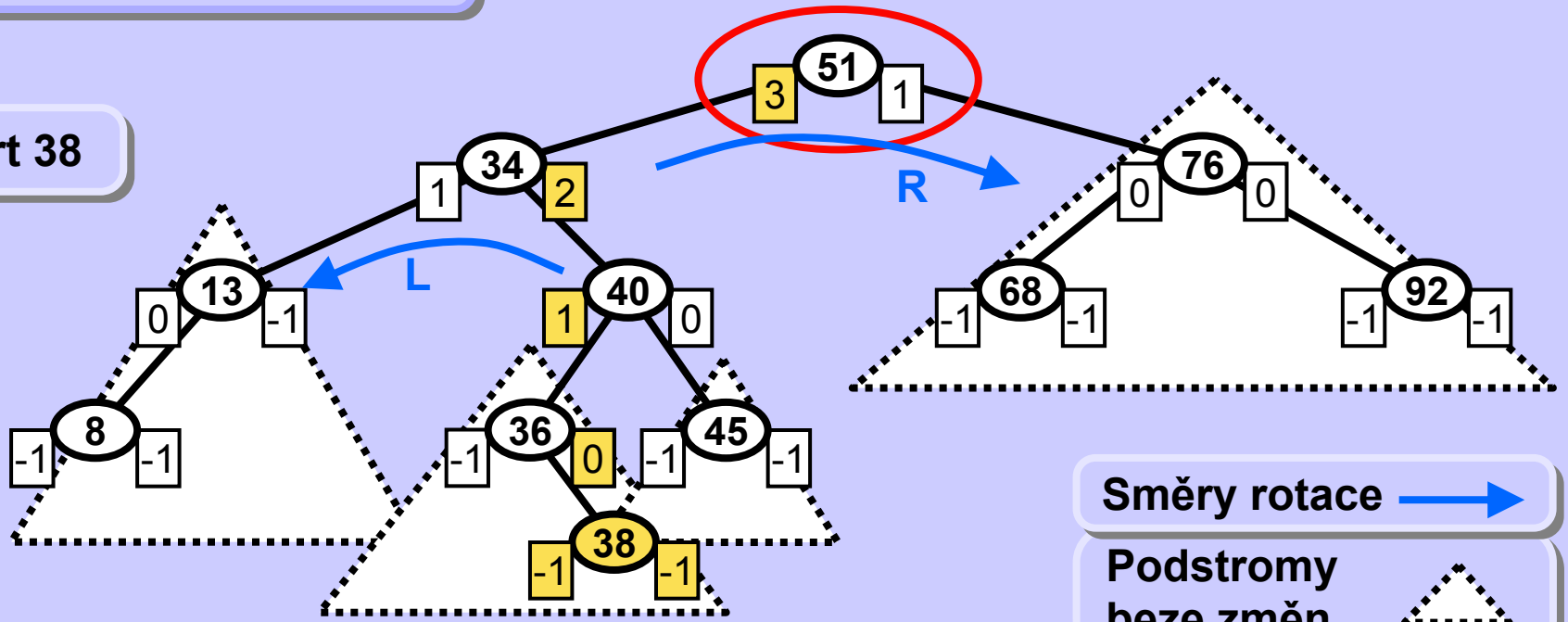


Vyvážený strom po dvojité LR rotaci



Náprava rozvážení rotací

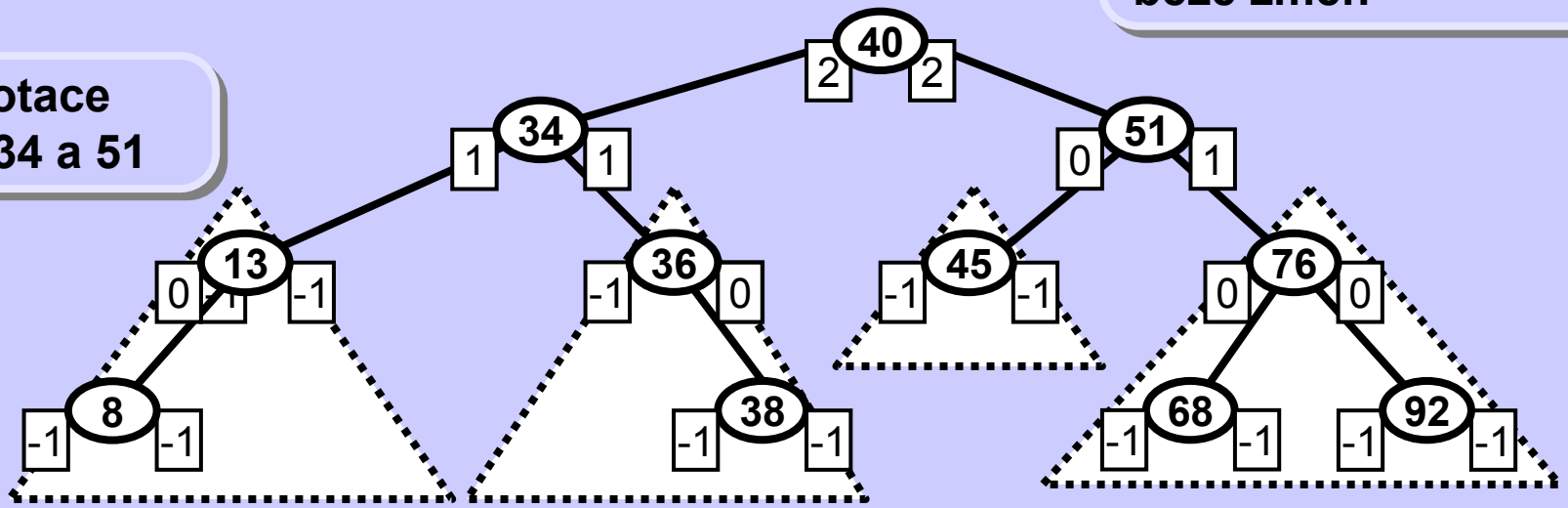
Insert 38



Směry rotace →

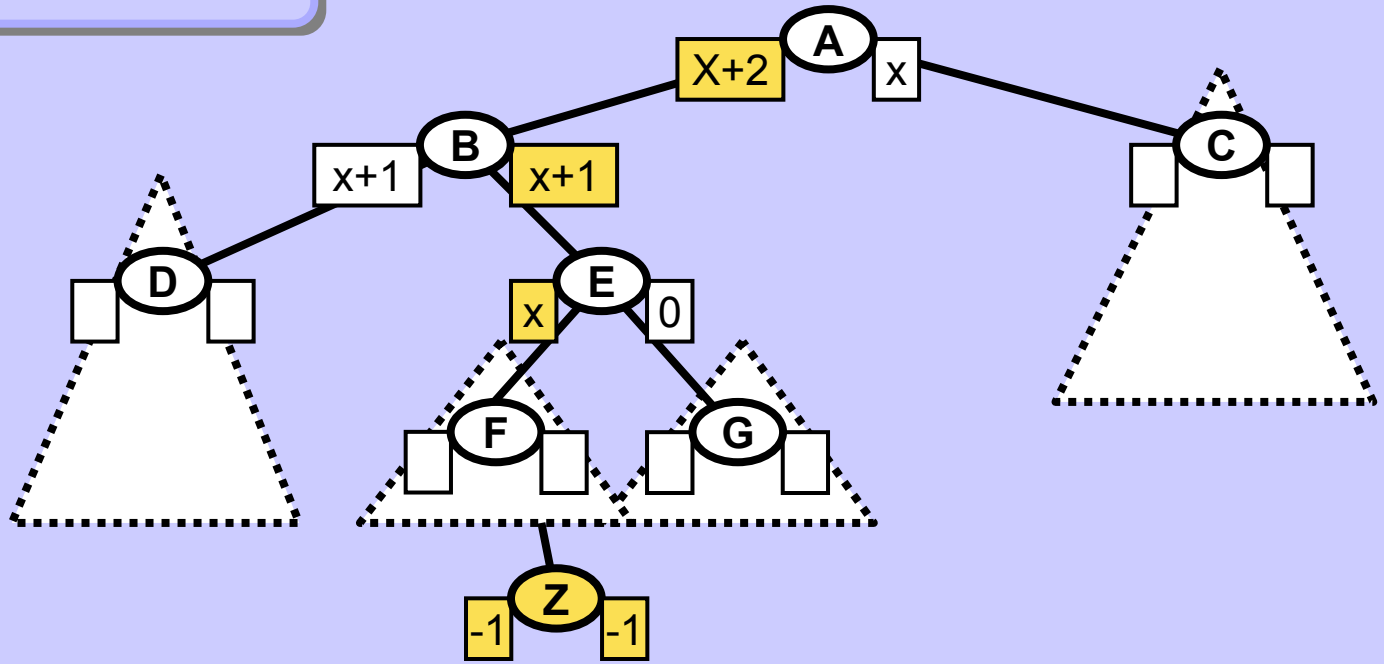
Podstromy beze změn

LR rotace v uzlu 34 a 51

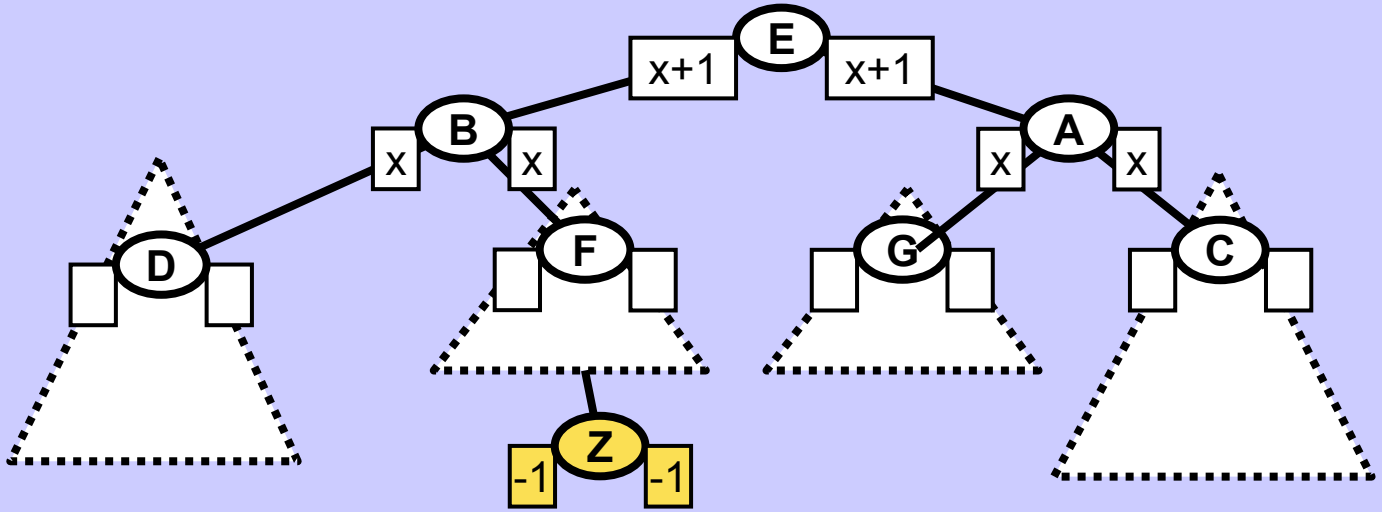


Rotace LR obecně

Před

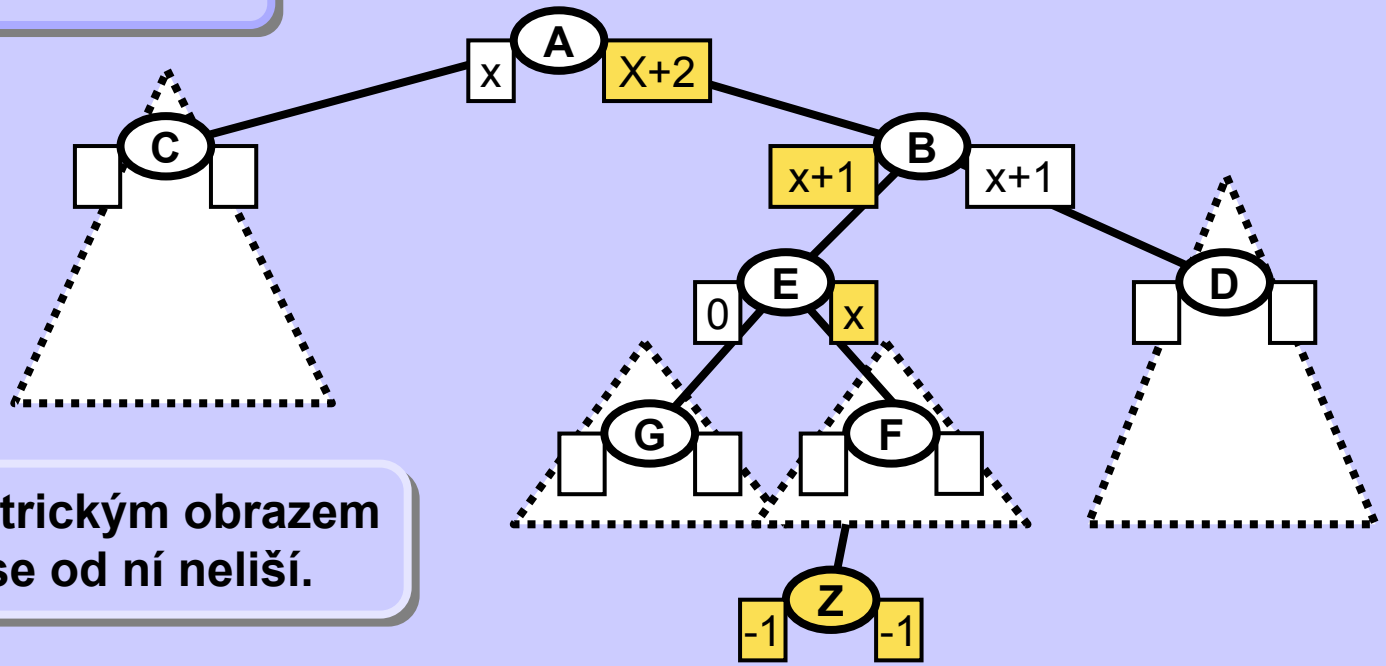


Po



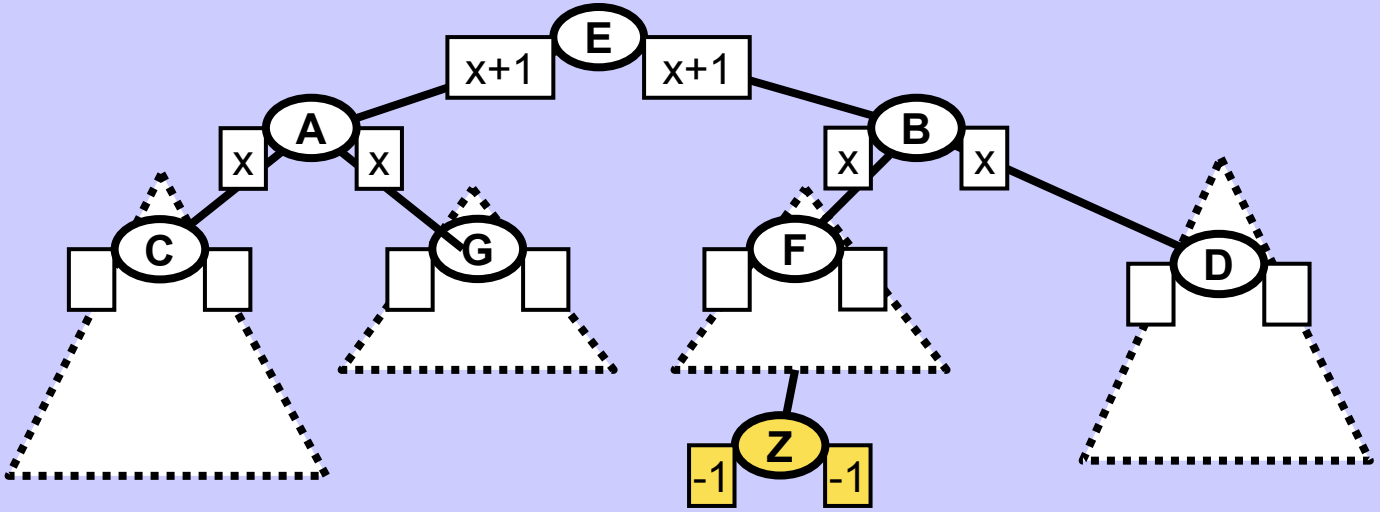
Rotace RL obecně

Před



Rotace RL je symetrickým obrazem rotace LR, jinak se od ní neliší.

Po



Pravidla pro aplikaci L, R, LR nebo RL rotací

Od přidaného (nebo smazaného, viz dále) uzlu postupujeme směrem ke kořeni a aktualizujeme hloubky podstromů v každém navštíveném uzlu.

**Když narazíme na rozvážený uzel,
do kterého jsme bezprostředně došli dvěma hranami *doprava* nahoru,
provedeme v tomto uzlu R rotaci.**

**Když narazíme na rozvážený uzel,
do kterého jsme bezprostředně došli dvěma hranami *doleva* nahoru,
provedeme v tomto uzlu L rotaci.**

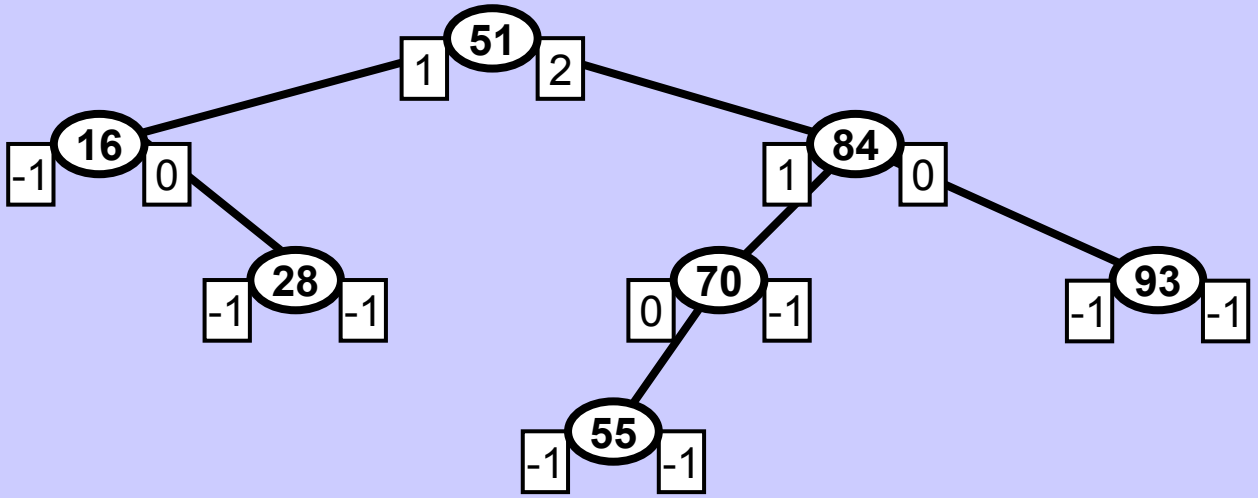
**Když narazíme na rozvážený uzel,
do kterého jsme bezprostředně došli hranami *doleva a pak doprava* nahoru,
provedeme v tomto uzlu LR rotaci.**

**Když narazíme na rozvážený uzel,
do kterého jsme bezprostředně došli hranami *doprava a pak doleva* nahoru,
provedeme v tomto uzlu RL rotaci.**

Po provedení jedné rotace je AVL strom opět vyvážen .

Delete v AVL stromu

Strom pro demonstraci rotace po smazání uzlu



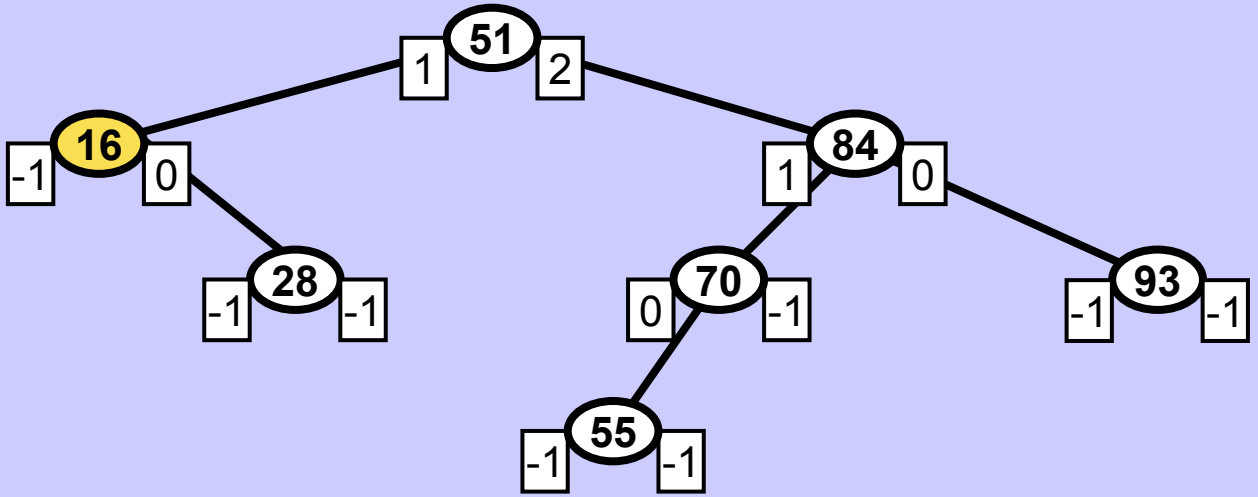
Delete 16

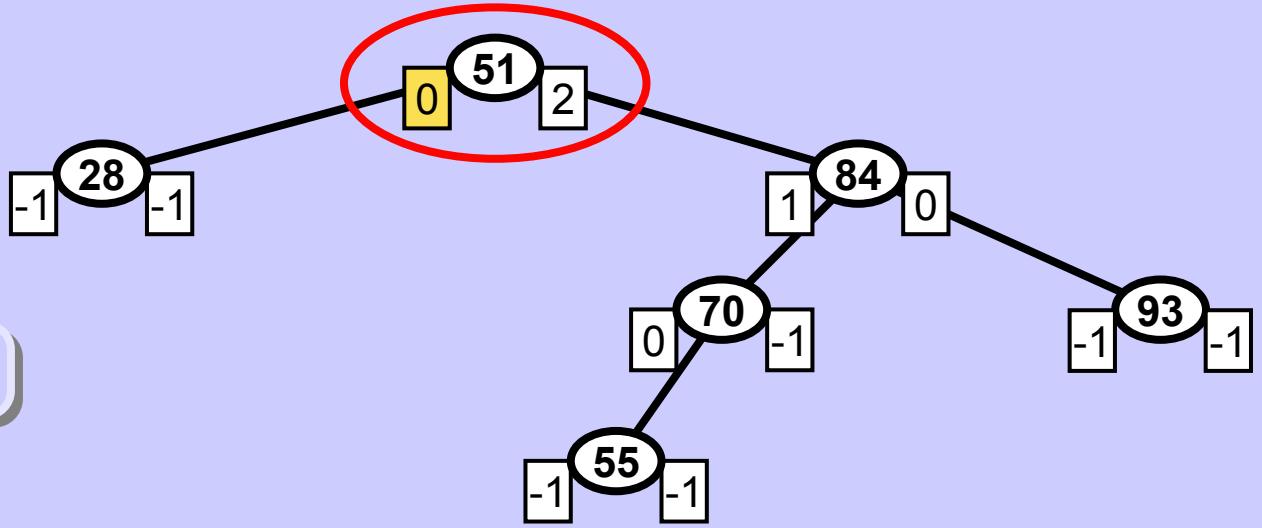
Delete proběhne standardně jako v obyčejném BVS.

Poté postupujeme od místa smazání nahoru ke kořeni a aktualizujeme výšky podstromů v každém uzlu. Při rozvážení aplikujeme rotaci podobně jako při vkládání.

Delete v AVL stromu

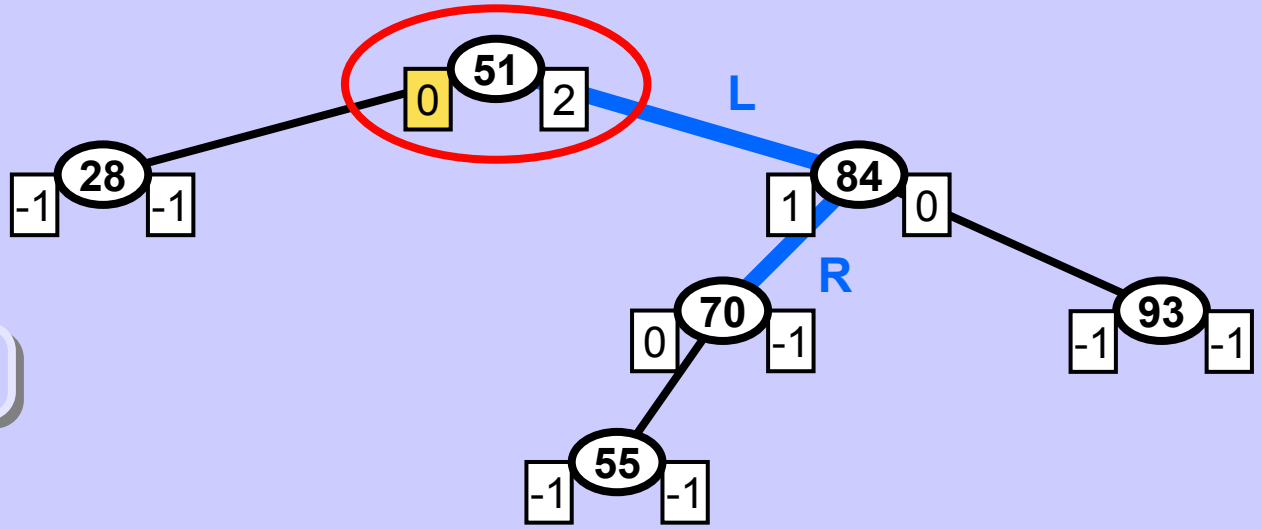
Delete 16





Změněné hloubky

Delete v AVL stromu



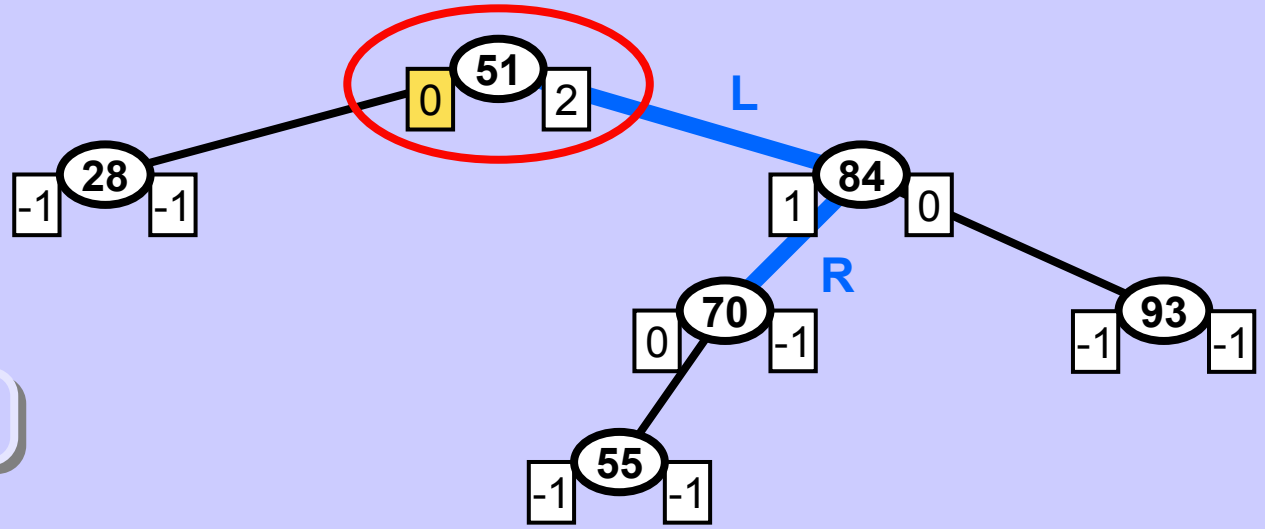
Změněné hloubky

Z rozváženého uzlu 51 prozkoumáme kořen sousedního podstromu, než ze kterého jsme přišli, v tomto případě uzel 84. Má-li tento oba své podstromy stejně hluboké použijeme jednoduchou L nebo R rotaci. Má-li je různě hluboké (nejvýše se liší o 1), rozhodneme, zda použijeme L, R, LR, RL rotaci, jako kdyby rozvážení (uzel 51) vzniklo naopak přidáním uzlu do tohoto sousedního podstromu (s kořenem 84). V tomto případě použijeme RL.

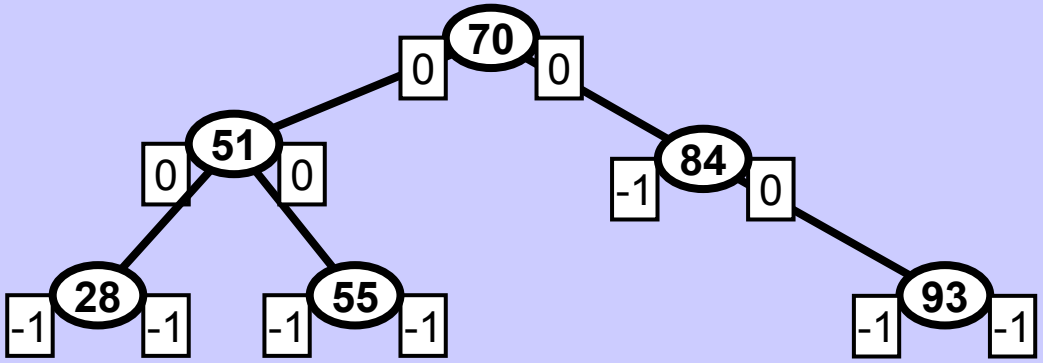
Delete v AVL stromu

Delete 16

Změněné hloubky



Po rotaci RL
v uzlu 84 a 51



Implementace operací v BVS a AVL stromu

```
...                                    // homework...
```

Asymptotické složitosti operací Find, Insert, Delete v BVS

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

ALG 04
Fin