Algoritmizace

Marko Genyk-Berezovskyj, Daniel Průša

2010 – 2023

Dnešní témata

- Vyhledávání v uspořádaném poli
- Binární vyhledávací stromy
- Čtvrtá domácí úloha

slido

Join at slido.com
#429042

ⓘ Start presenting to display the joining instructions on this slide.

slido

Sledujete přednášky z Algoritmizace přes stream?

ⓘ Start presenting to display the poll results on this slide.

Vyhledávání v seřazeném poli

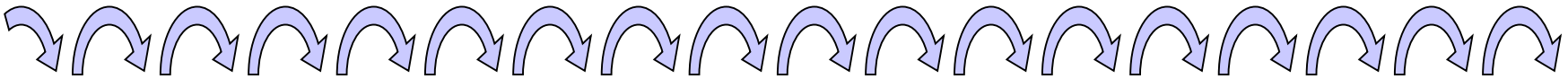
Seřazené pole: → velikost = N

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

- Lineární (sekvenční) vyhledávání je pomalé.

Testů (porovnání hodnot) je 1 (najdi 363) až N (najdi 993),

v průměrném případě $\frac{1}{N}\sum_{i=1}^{N} i = \frac{N+1}{2}$ (předpoklad: hledaná hodnota je v poli)

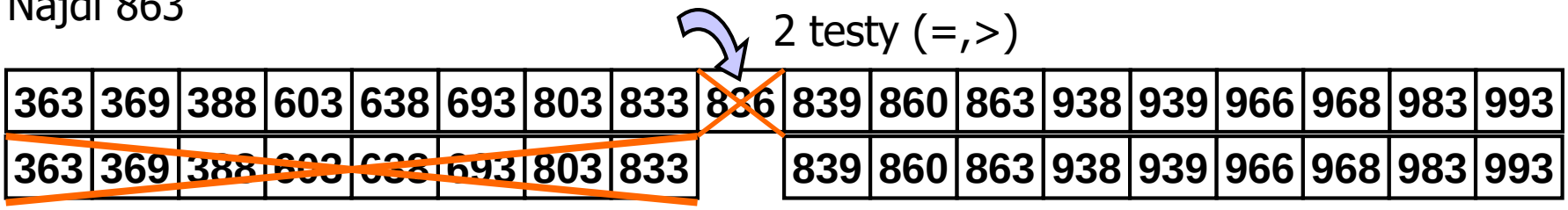


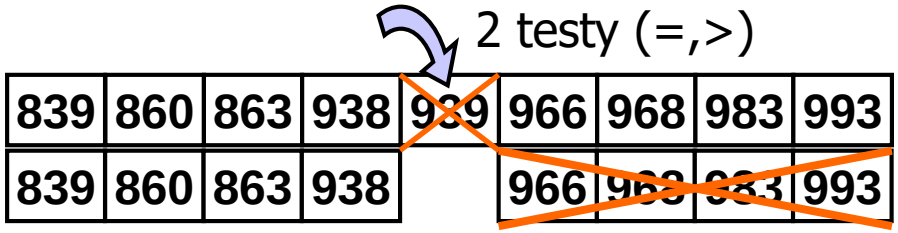
363	369	388	603	638	693	803	833	836	839	860	863	938	939	966	968	983	993

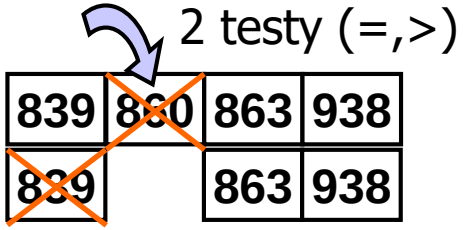
Binární vyhledávání

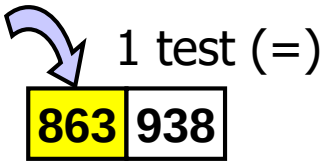
- Též známé jako vyhledávání půlením intervalu.

Najdi 863









Rozděl a panuj

$$T(1) = \Theta(1)$$

$$T(N) = T\left(\frac{N}{2}\right) + \Theta(1), N > 1$$

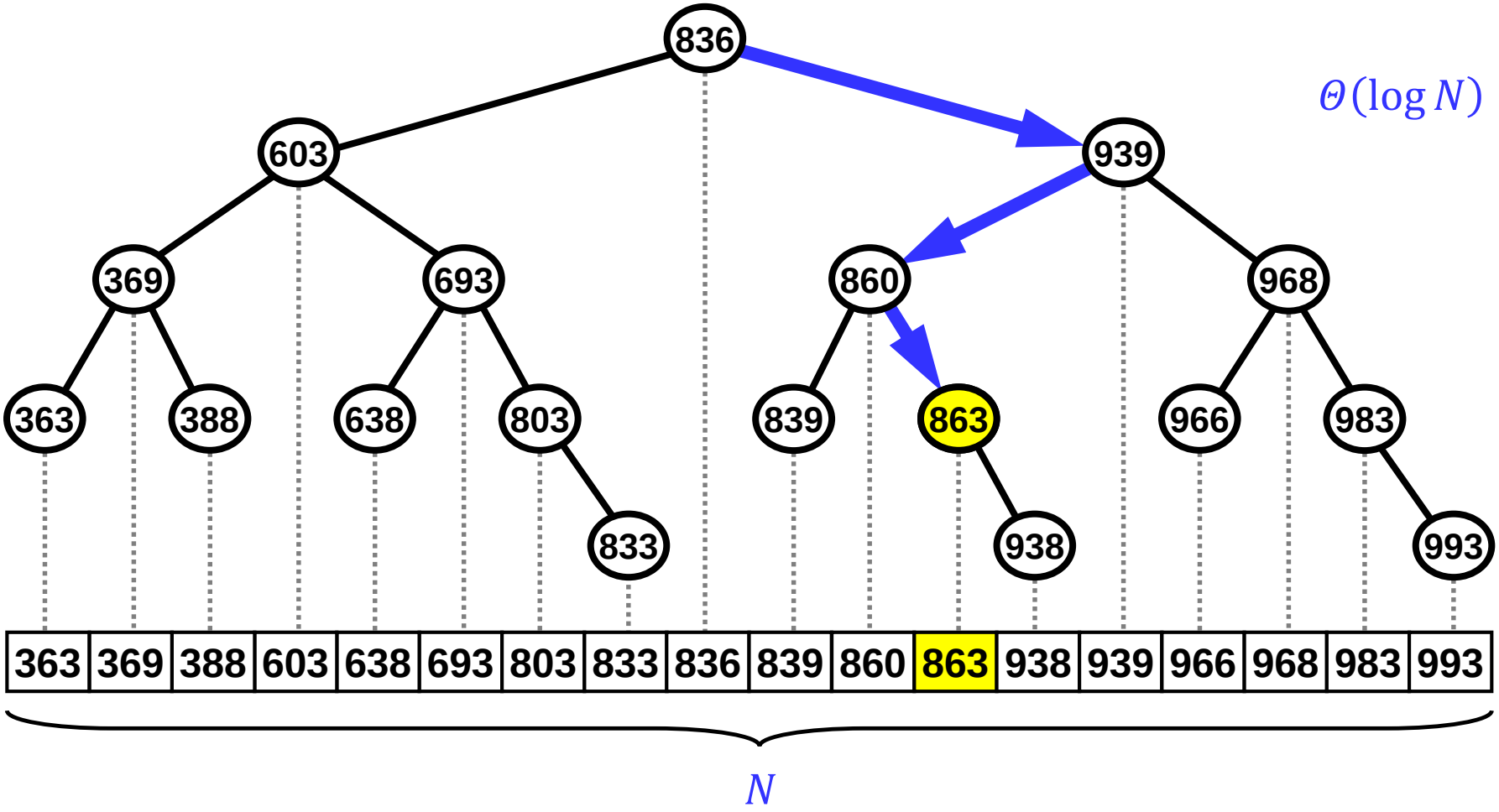
$$\Rightarrow T(N) \in \Theta(\log N)$$

podle mistrovské věty

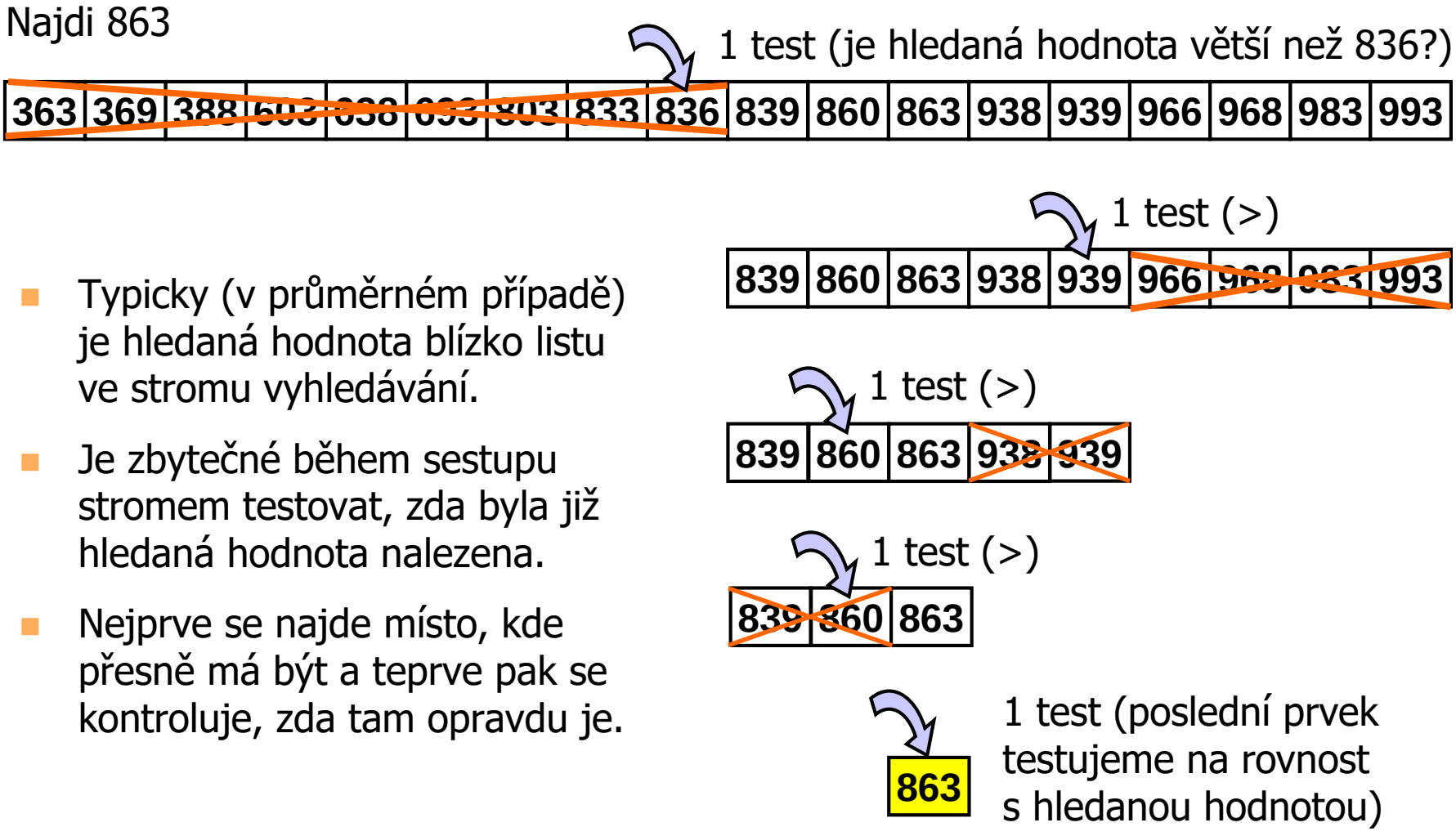
Binární vyhledávání

Najdi 863

Hledání kopíruje strukturu vyváženého binárního stromu



Binární vyhledávání – vylepšení



- Typicky (v průměrném případě) je hledaná hodnota blízko listu ve stromu vyhledávání.
- Je zbytečné během sestupu stromem testovat, zda byla již hledaná hodnota nalezena.
- Nejprve se najde místo, kde přesně má být a teprve pak se kontroluje, zda tam opravdu je.

Binární vyhledávání

```
int binarySearch(int[] arr, int val) {
    int low = 0, high = arr.length - 1, mid;
    while (low < high) {
        mid = low + (high - low) / 2; // better than
        if (val > arr[mid])           // mid = (low+high)/2;
            low = mid + 1;
        else
            high = mid;
    }
    if (arr[low] == val)
        return low;
    else
        return -1;
}
```

slido

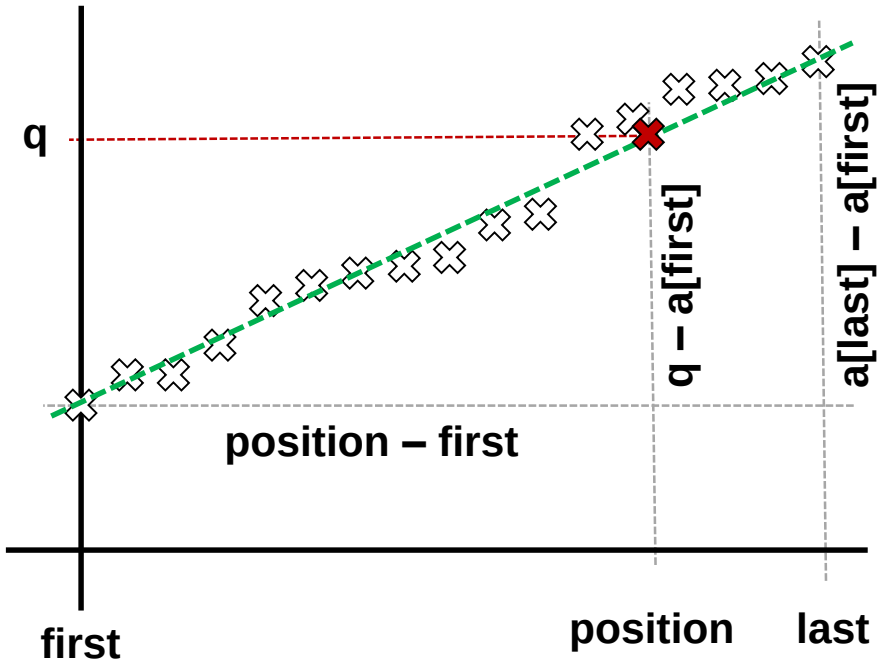
Audience Q&A Session

ⓘ Start presenting to display the audience questions on this slide.

Interpolační vyhledávání

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

363	369	388	603	638	693	803	833	836	839	860	863	938	939	966	968	983	993
0	1	2											13		15		17
first															position		last



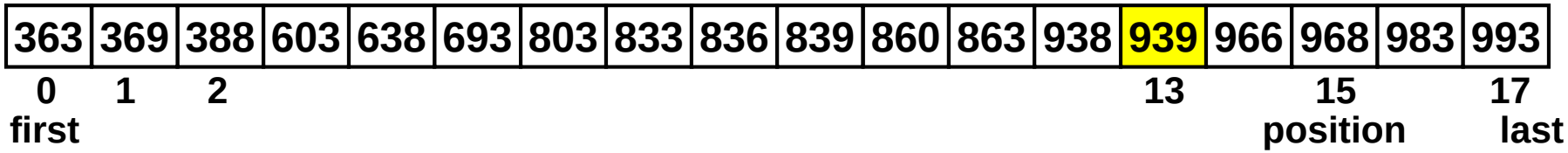
Jsou-li hodnoty v poli víceméně rovnoměrně rozložené, je možno použít lineární interpolaci pro odhad hledané pozice.

$$\frac{q - a[first]}{position - first} = \frac{a[last] - a[first]}{last - first}$$

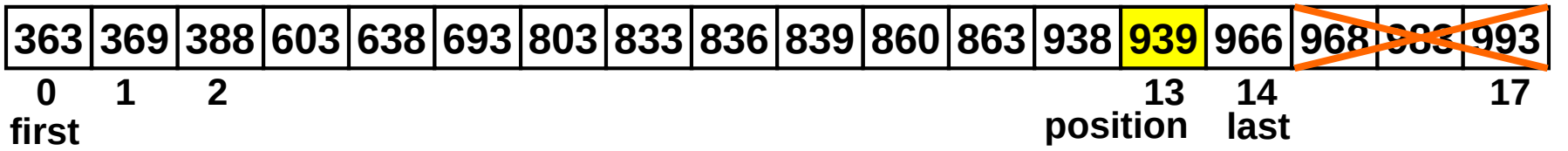
$$position = first + \frac{q - a[first]}{a[last] - a[first]} * (last - first)$$

Interpolační vyhledávání

Najdi 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.



- Časová složitost pro rovnoměrně rozložená data: $O(\log \log N)$

slido

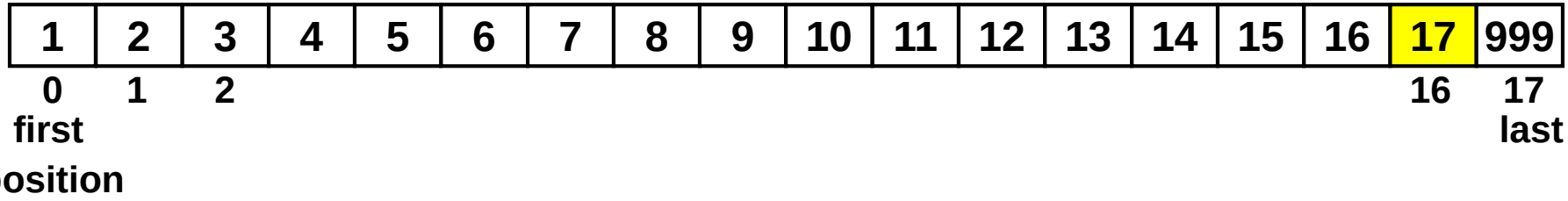
Jakou časovou složitost má Interpolační vyhledávání v nejhorším případě?

ⓘ Start presenting to display the poll results on this slide.

Interpolační vyhledávání

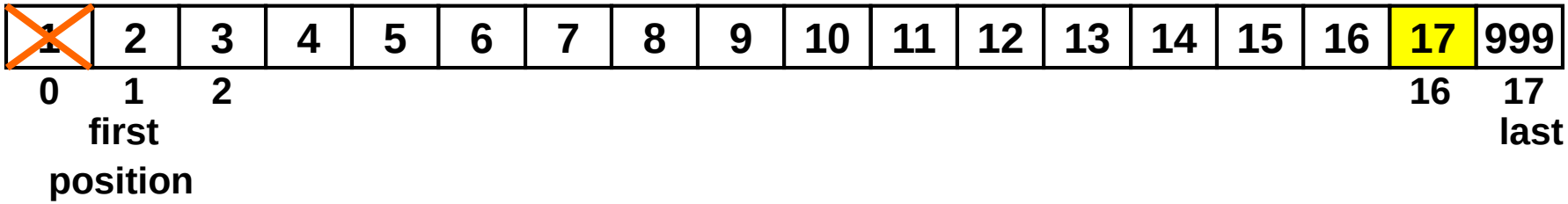
- Nejhorší případ – lineární složitost

Najdi 17



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

$$0 + \left\lfloor \frac{17 - 1}{999 - 1} \cdot (17 - 0) \right\rfloor = 0$$



Jak zařídit, aby nejhorší případ měl složitost $O(\log N)$?

Interpolační vyhledávání

```
int interpolationSearch(int[] arr, int key) {
    int low = 0, high = arr.length - 1;
    while (low <= high && key >= arr[low] && key <= arr[high]) {
        if (low == high) {
            if (arr[low] == key) return low;
            return -1;
        }
        int pos = low + (key - arr[low])*(high-low) /
            (arr[high] - arr[low]);
        if (arr[pos] == key)
            return pos;
        if (arr[pos] < key)
            low = pos + 1;
        else
            high = pos - 1;
    }
    return -1;
}
```

slido

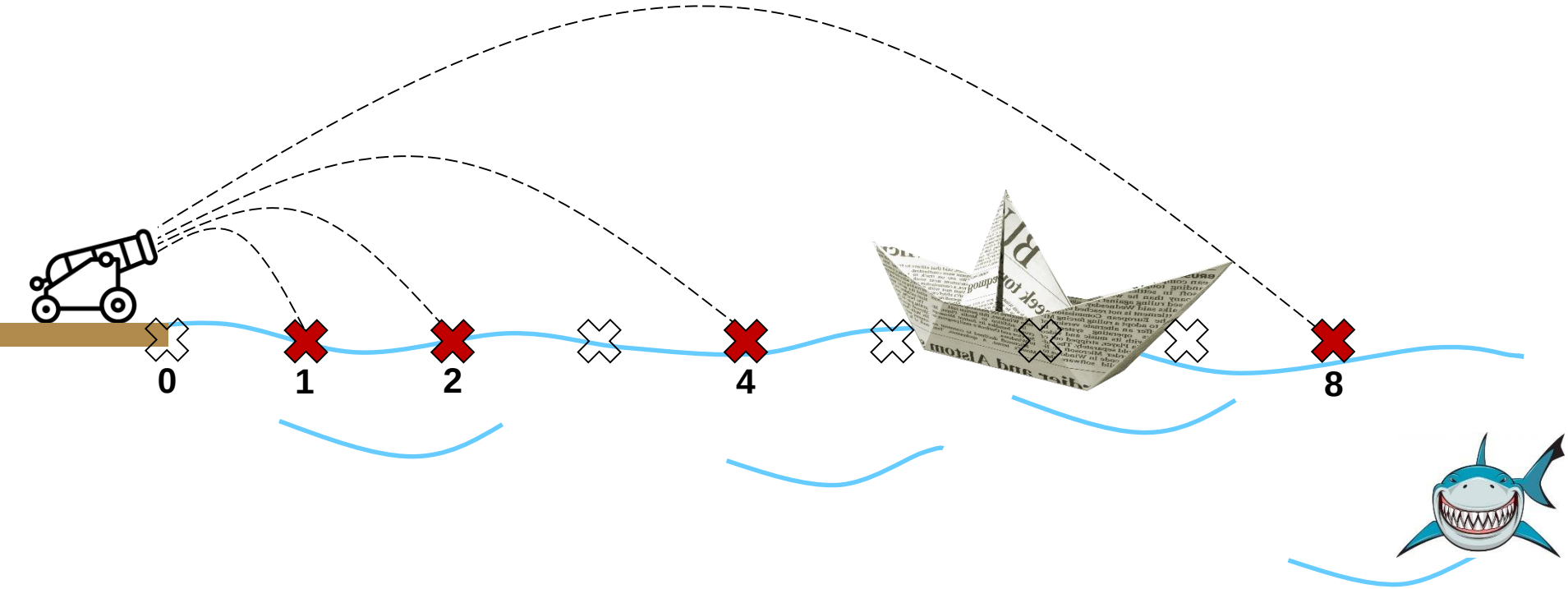
Audience Q&A Session

ⓘ Start presenting to display the audience questions on this slide.

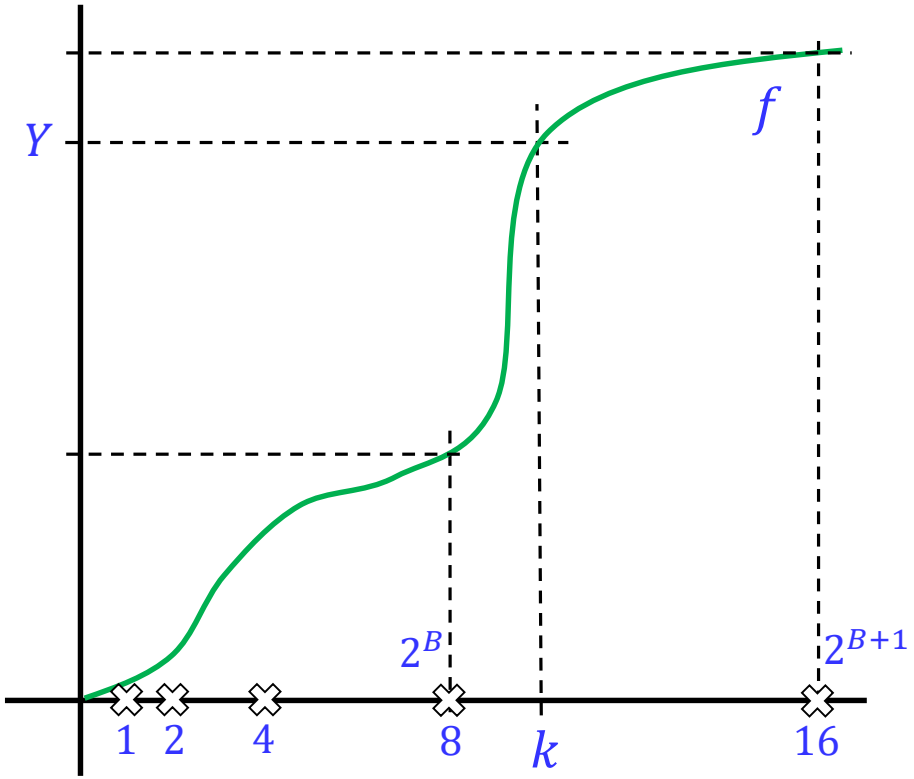
Exponenciální vyhledávání

Jak trefit loď?

- dostřel můžeme nastavit libovolně
- nevíme ale, jak je loď daleko (pro vzdálenost neexistuje ani odhad)
- po výstřelu vidíme pouze šplouchnutí před/za lodí, nebo zásah



Exponenciální vyhledávání

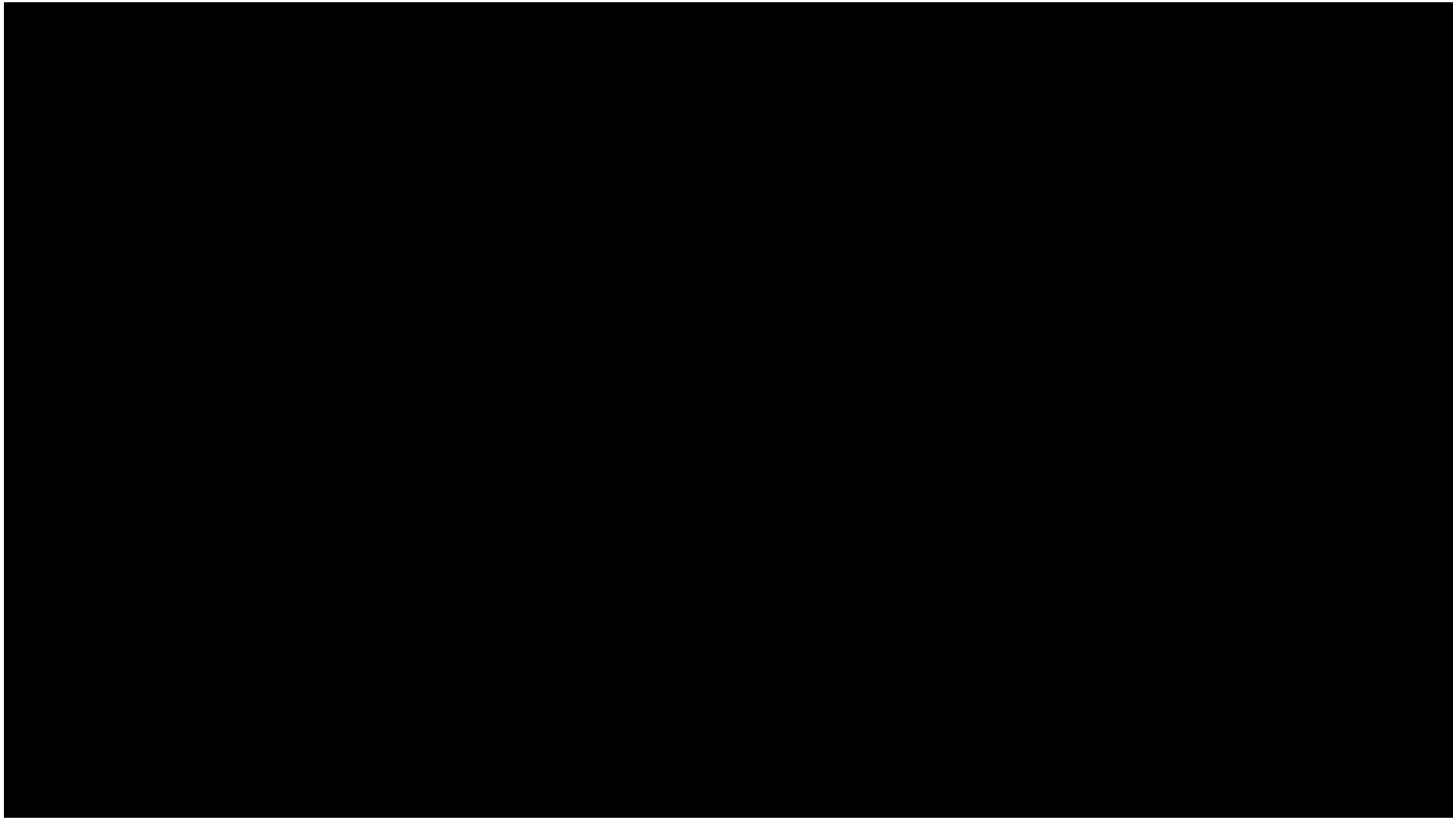


Je dána rostoucí funkce $f: \mathbb{N} \to \mathbb{N}$ (např. algoritmem) a číslo $Y \in \mathbb{N}$.

Hledáme nejmenší číslo $k \in \mathbb{N}$, pro které platí $f(k) \geq Y$.

	Časová složitost:
1. Inkrementálně nalezneme B, pro které platí $f(2^B) < Y \leq f(2^{B+1})$.	$\Theta(B+1) = \Theta(\log k)$
2. Aplikujeme binární vyhledávání pro posloupnost $f(2^B), \ldots, f(2^{B+1})$.	$\Theta(\log(2^{B+1} - 2^B + 1)) = \Theta(\log k)$

Vyhledejte pandu



https://www.youtube.com/watch?v=AbfdWxOSqtg

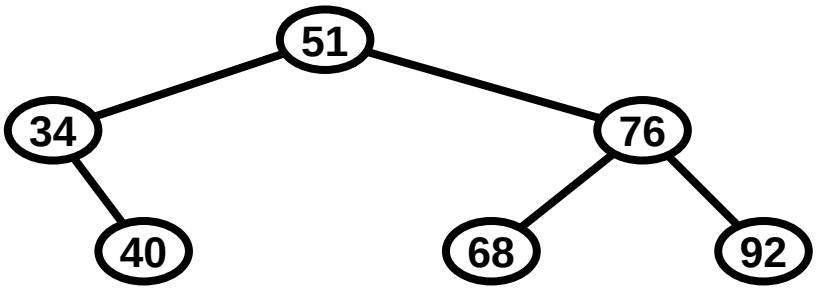
slido

Audience Q&A Session

ⓘ Start presenting to display the audience questions on this slide.

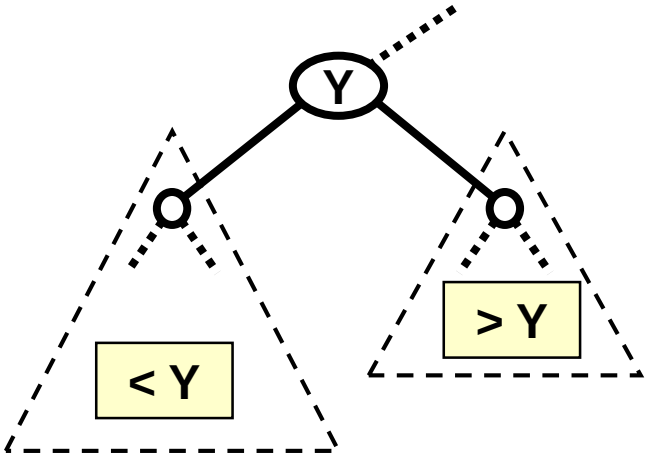
Binární vyhledávací strom (BVS)

- Binary search tree (BST).
- Datová struktura reprezentující množinu klíčů.
- Operace Find, Insert a Delete.

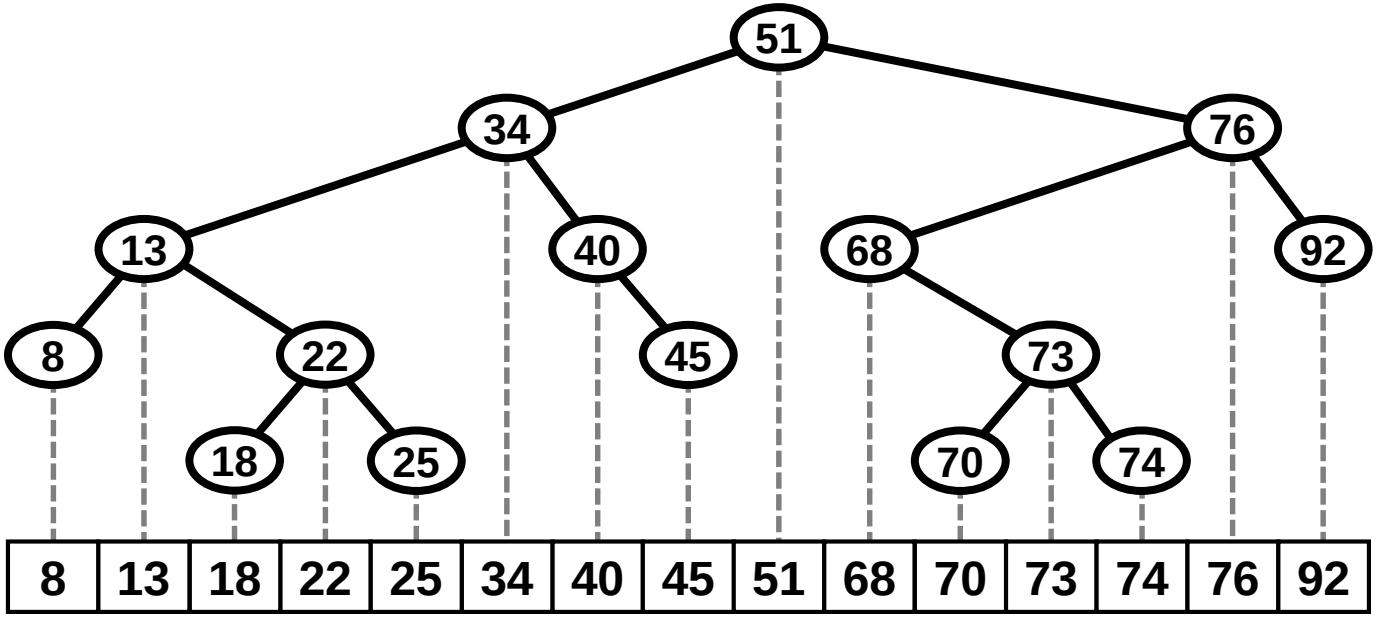


- V levém podstromu každého uzlu jsou všechny klíče menší.
- V pravém podstromu každého uzlu jsou všechny klíče větší.

Binární vyhledávací strom (BVS)



- V levém podstromu každého uzlu jsou všechny klíče menší.
- V pravém podstromu každého uzlu jsou všechny klíče větší.

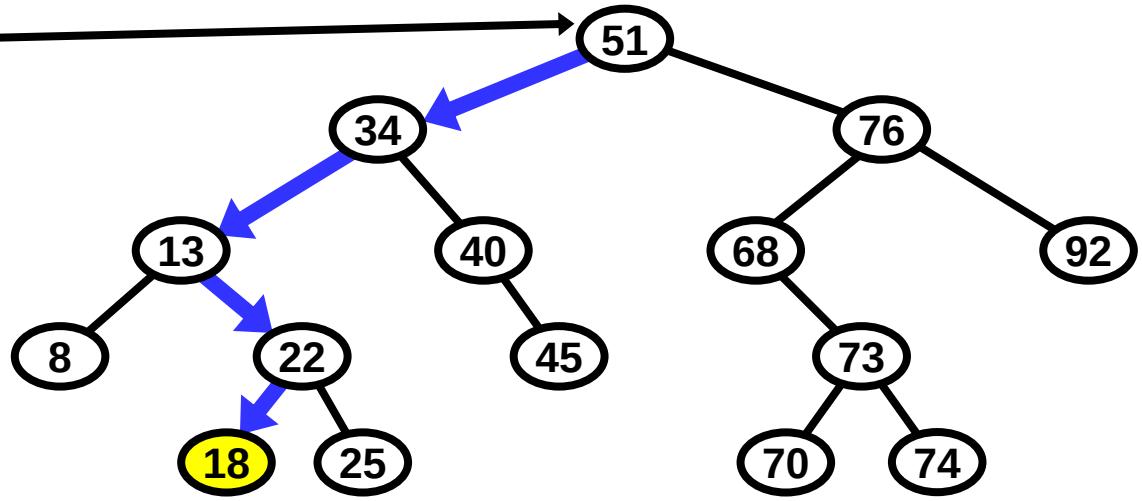


pořadí inorder = vzestupně uspořádané klíče

Operace Find v BVS

Najdi 18

Každá operace se stromem začíná v kořeni.

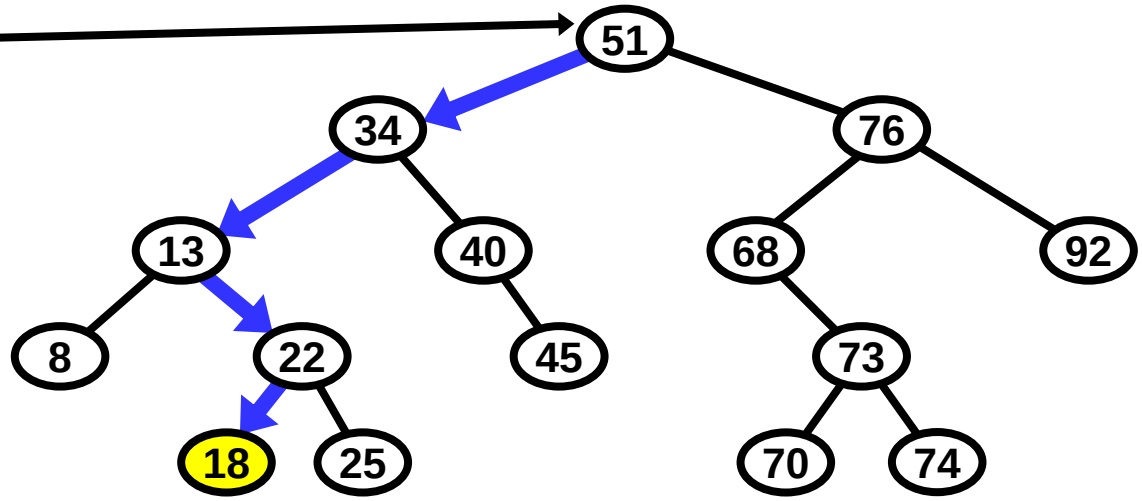



```
Node find(int key, Node node) {
    if (node == null || node.key == key)
        return node;
    if (key < node->key)
        return find(key, node->left);
    else
        return find(key, node->right);
}
```

Operace Find v BVS

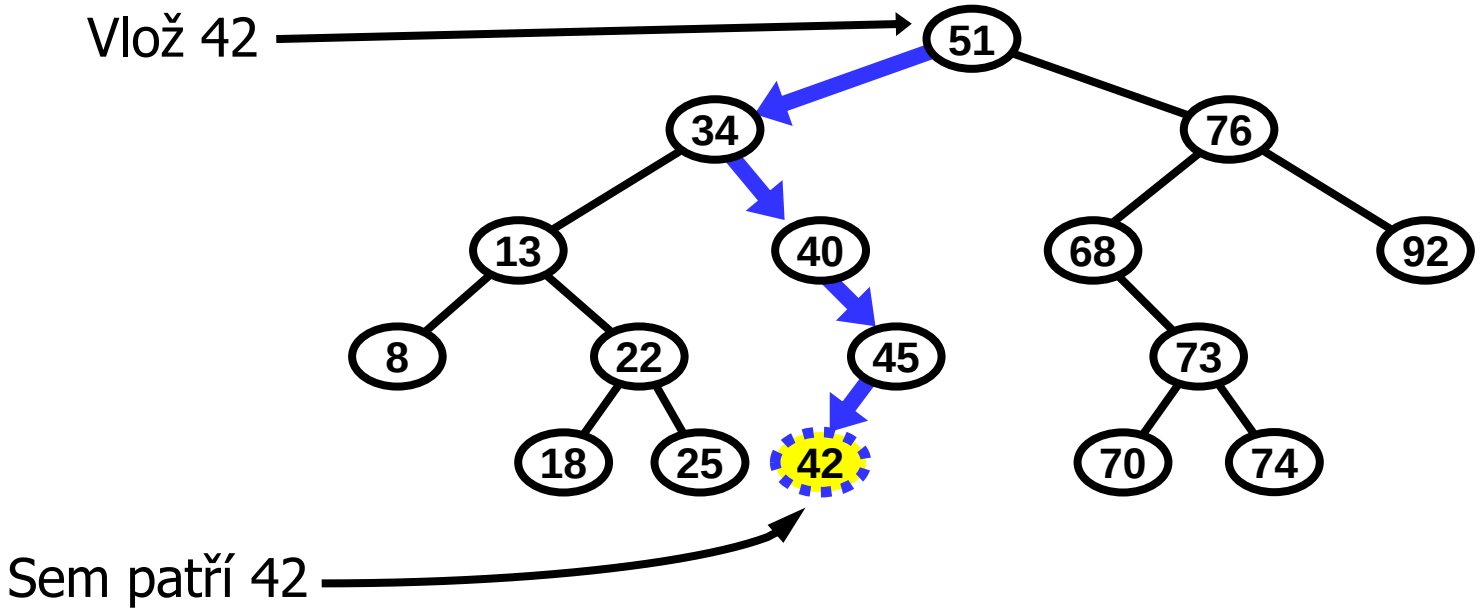
Najdi 18

Každá operace se stromem začíná v kořeni.




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

Operace Insert v BVS



Insert

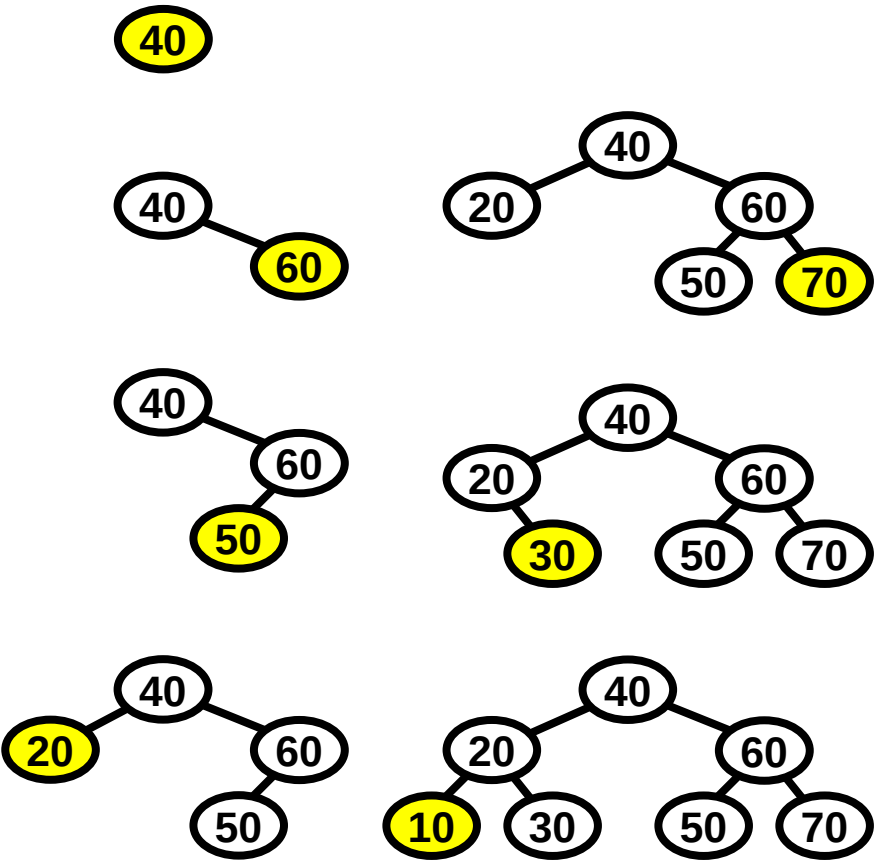
1. Najdi místo (jako ve Find) pro list, kam patří uzel s daným klíčem.
2. Vytvoř tento uzel a vlož jej do stromu.

Operace Insert v BVS

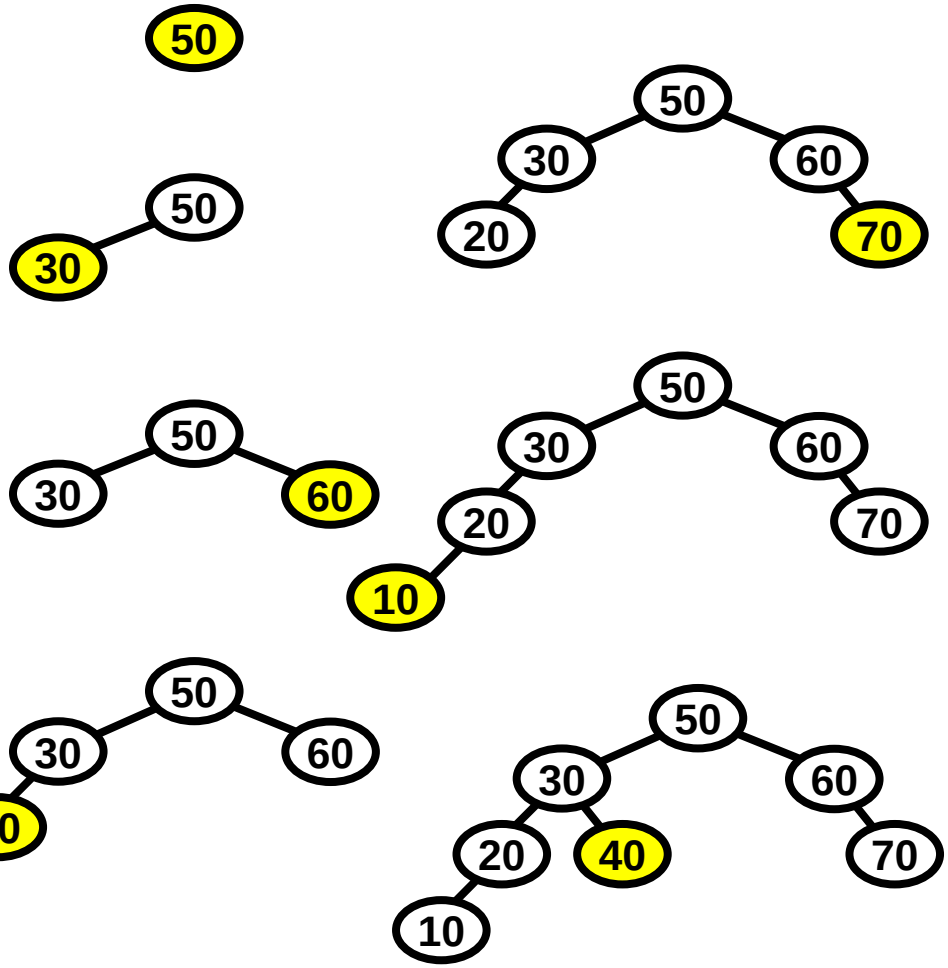
```
Node insert(Node node, int key) {
    /* If the tree is empty, return a new node */
    if (node == null)
        return new Node(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    /* return the (unchanged) node pointer */
    return node;
}
```

Různé tvary BVS

Insert: 40, 60, 50, 20, 70, 30, 10



Insert: 50, 30, 60, 20, 70, 10, 40



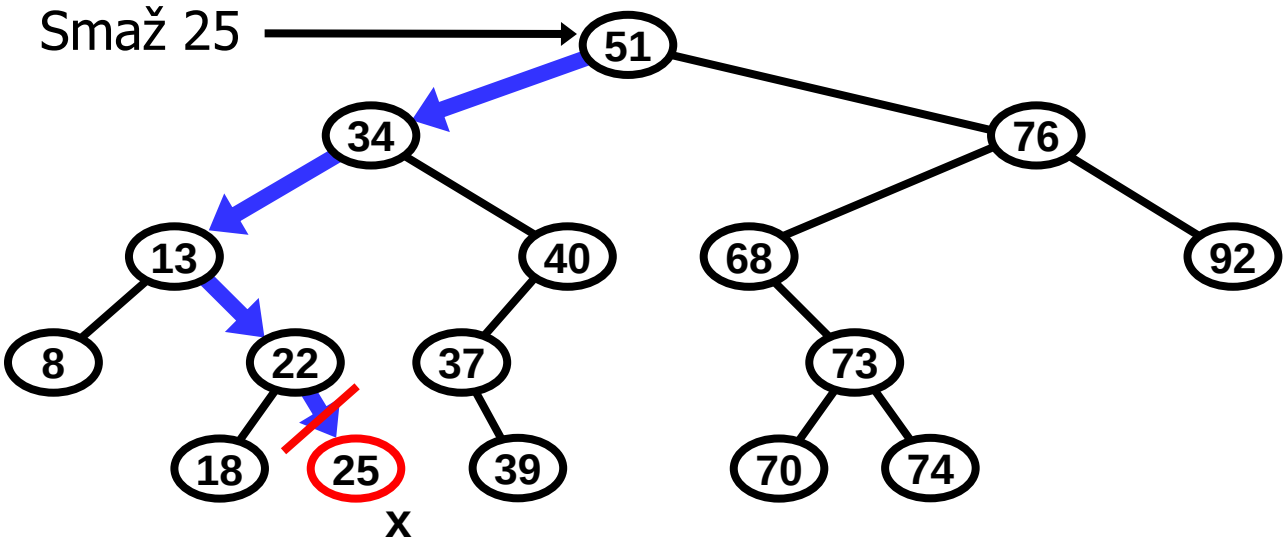
slido

Audience Q&A Session

ⓘ Start presenting to display the audience questions on this slide.

Operace Delete v BVS

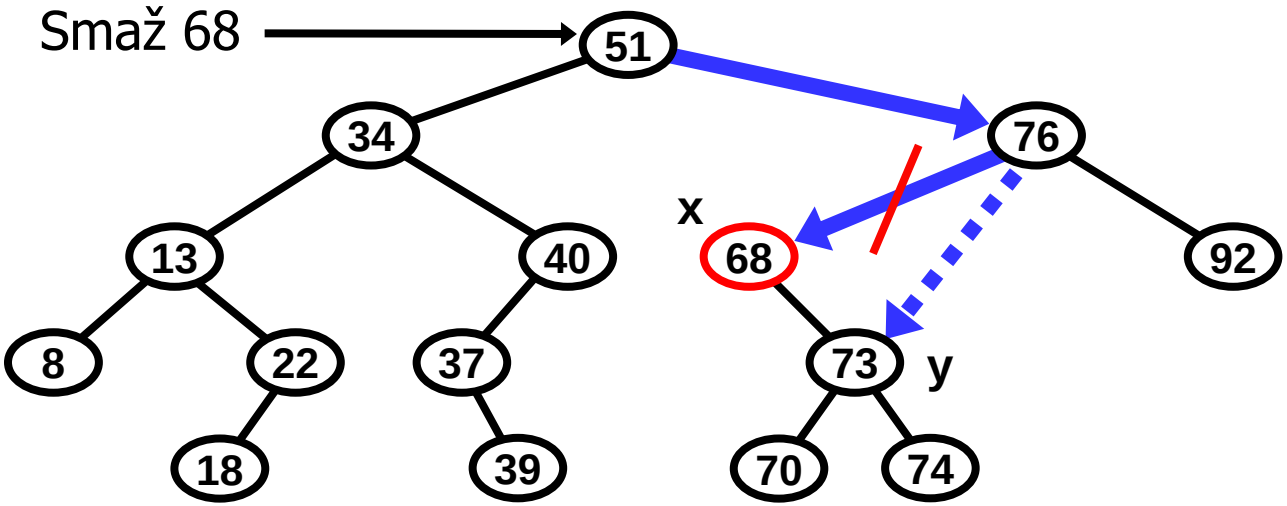
1. Smazání listu



- Najdi daný uzel *x* (Find) a odstraň ukazatel na něj z jeho rodiče.

Operace Delete v BVS

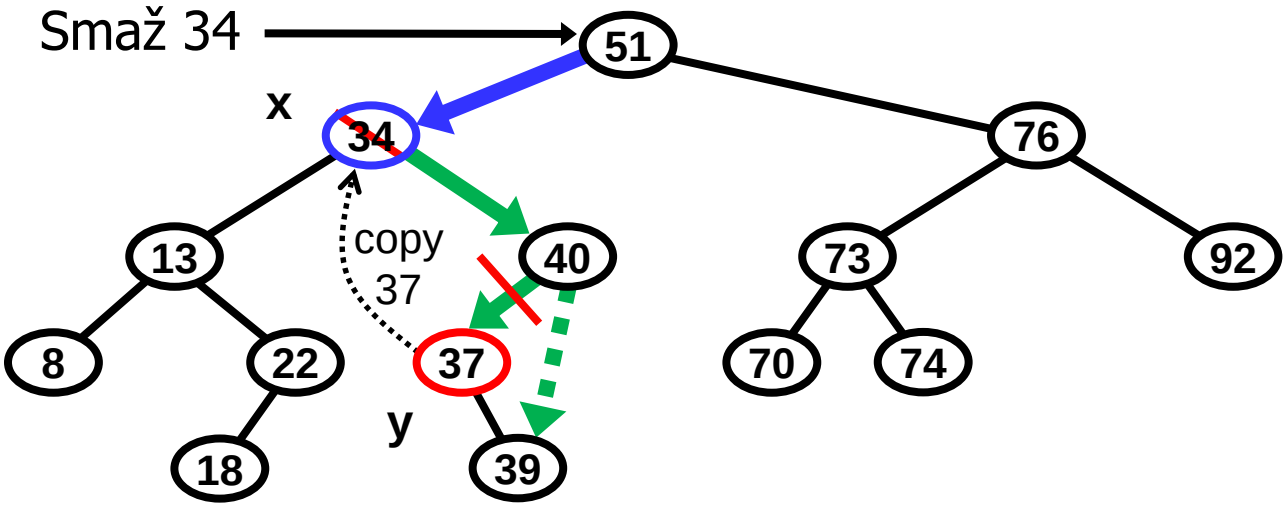
2. Smazání uzlu s 1 potomkem



- Najdi daný uzel *x* (operace Find) s potomkem *y*.
- V jeho rodiči změň ukazatel, který ukazoval na *x*, aby nově ukazoval na *y*.

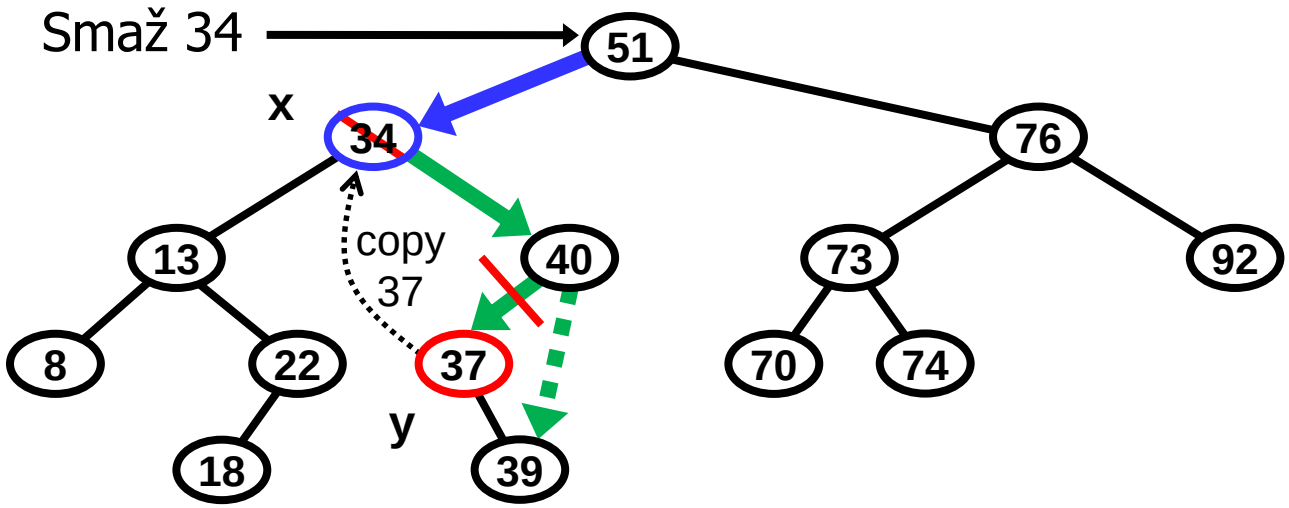
Operace Delete v BVS

3. Smazání uzlu se 2 potomky

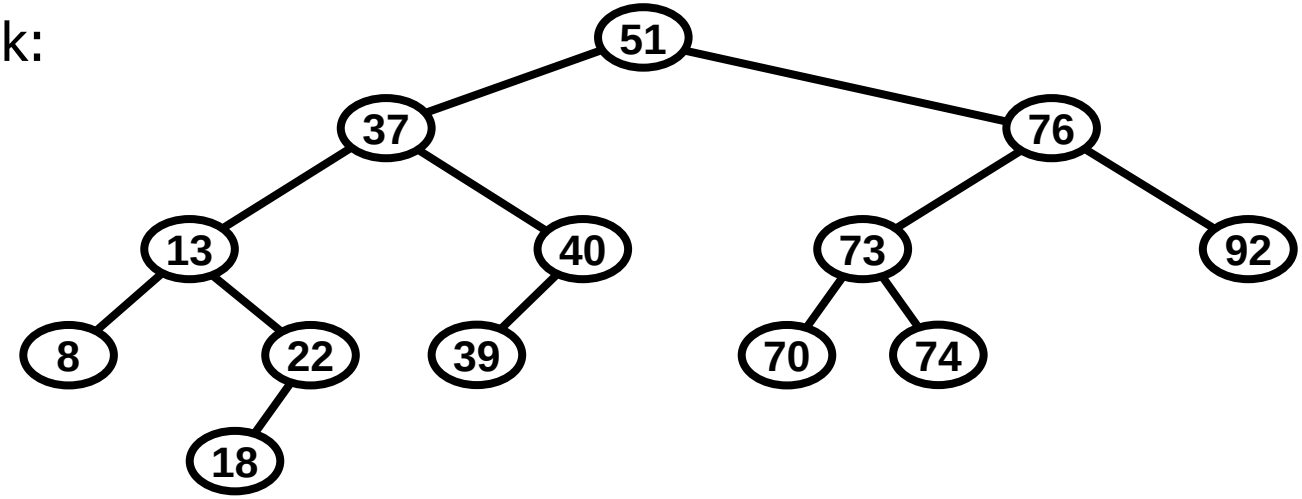


- Najdi daný uzel *x* (Find) a dále najdi uzel *y* s nejmenším klíčem v pravém podstromu uzlu *x*.
- Zkopíruj klíč uzlu *y* do uzlu *x*.
- Smaž uzel *y*, který má maximálně 1 potomka (viz předchozí případy).

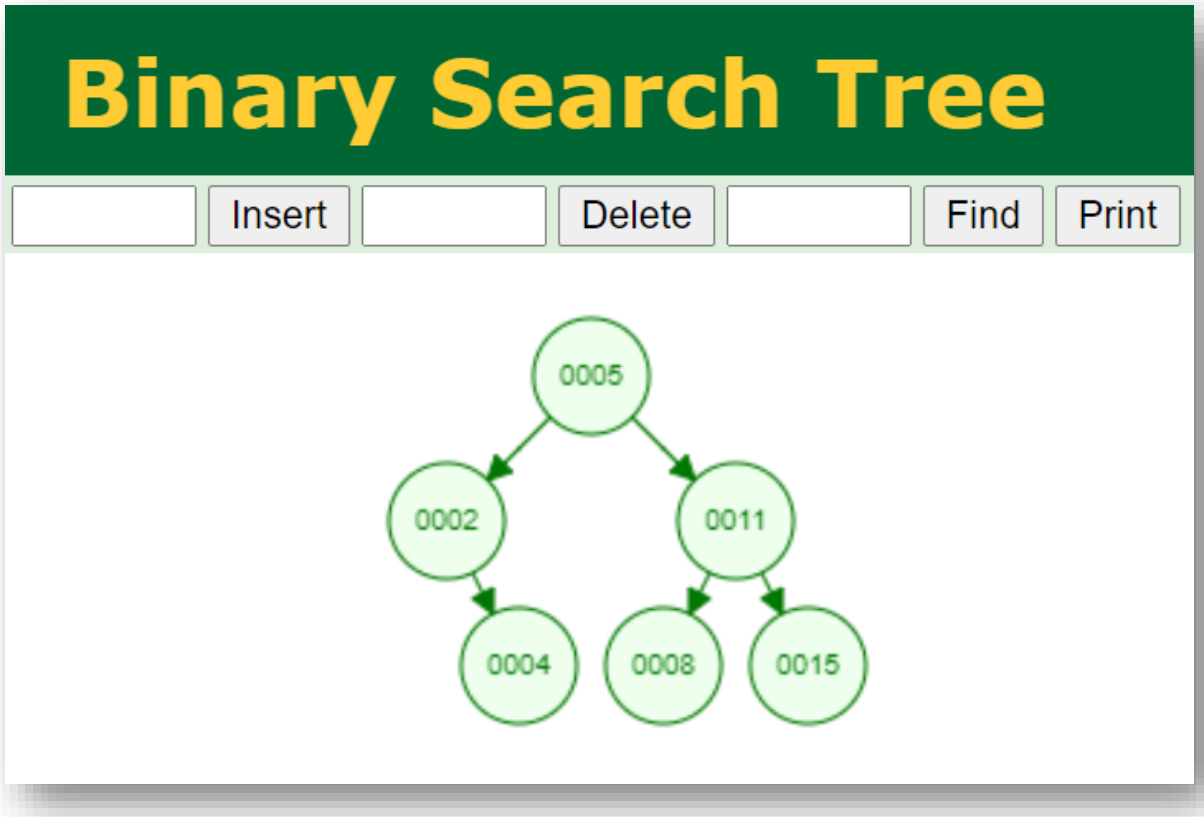
Operace Delete v BVS



Výsledek:



Vizualizace operací v BVS

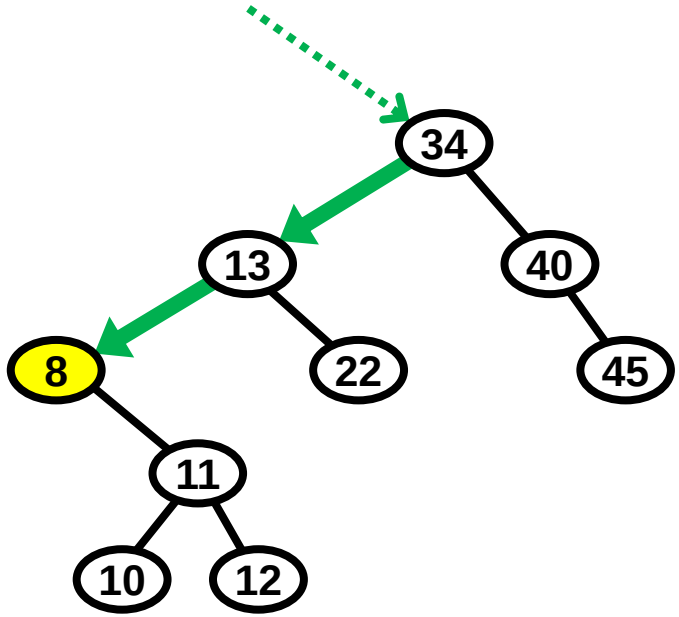


https://www.cs.usfca.edu/~galles/visualization/BST.html

Operace Delete v BVS

Nalezení nejmenšího klíče v podstromě:

```
int minValue(Node node) {
    while (node.left != null)
        node = node.left;
    return node.key;
}
```



Operace Delete v BVS

```
Node delete(int key, Node root) {
    if (root == null) return root; // if the tree is empty
    if (key < root.key) // otherwise, recur down the tree
        root.left = delete(key, root.left);
    else if (key > root.key)
        root.right = delete(key, root.right);
    else {
        // node with only one child or no child
        if (root.left == null)
            return root.right;
        else if (root.right == null)
            return root.left;
        else {
            // node with two children
            root.key = minValue(root.right);
            // delete the inorder successor
            root.right = delete(root.key, root.right);
        }
        return root;
    }
}
```

slido

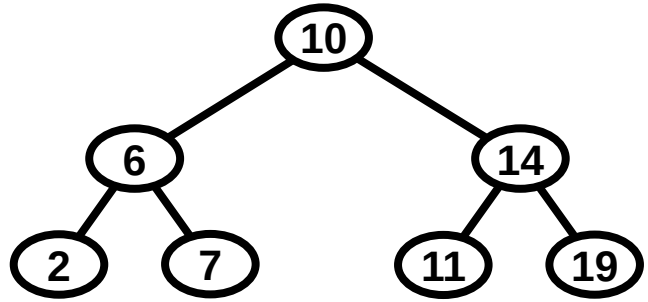
Jakou časovou složitost má operace Delete v BVS s 'n' uzly?

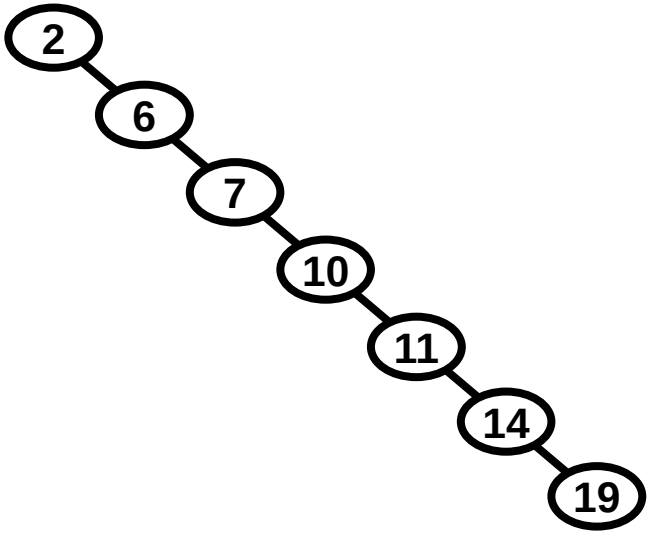
ⓘ Start presenting to display the poll results on this slide.

Asymptotická složitost operací

Asymptotická složitost operací v BVS je úměrná maximální hloubce uzlů, které navštívíme.

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



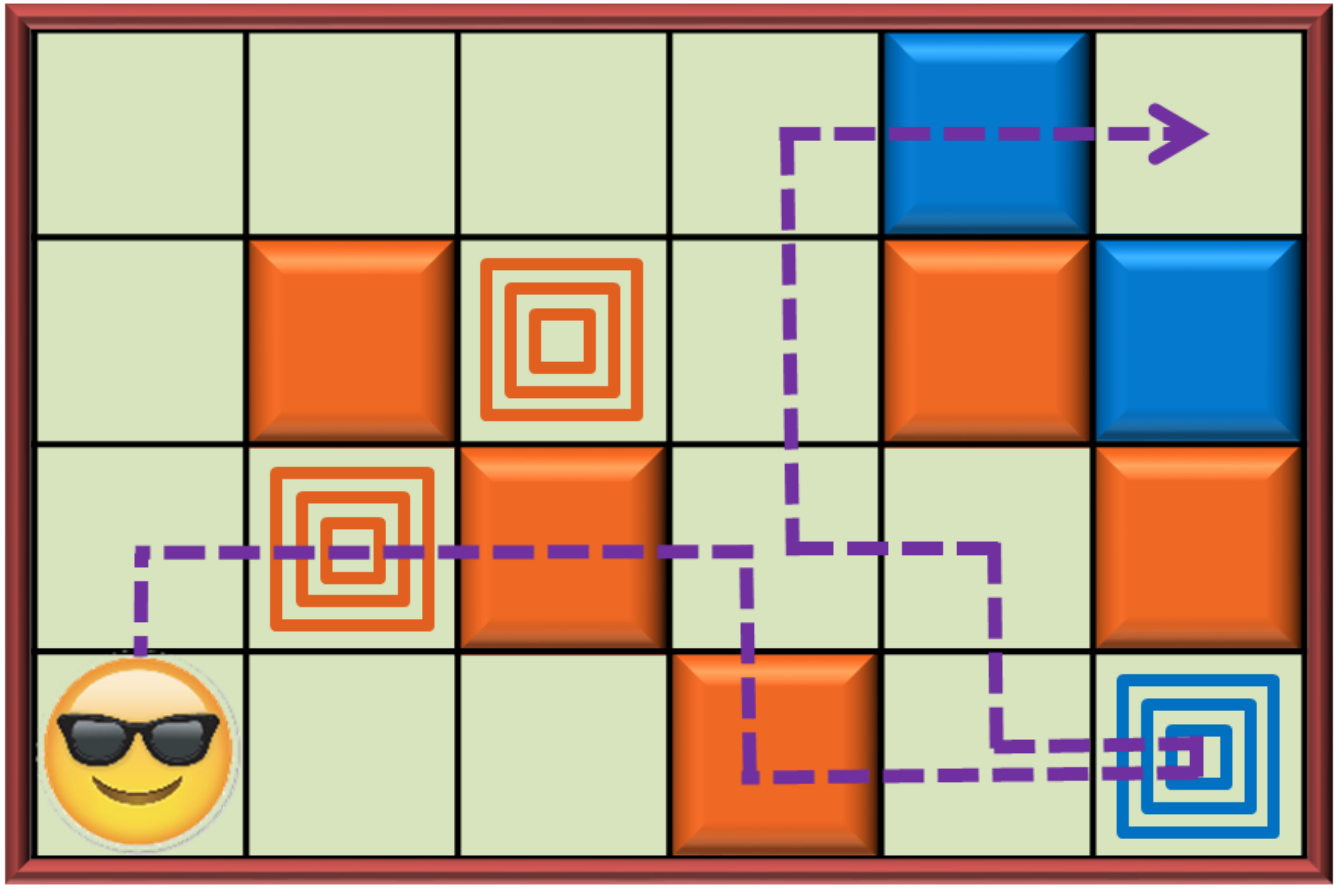


slido

Audience Q&A Session

ⓘ Start presenting to display the audience questions on this slide.

Čtvrtá domácí úloha



slido

Audience Q&A Session

ⓘ Start presenting to display the audience questions on this slide.