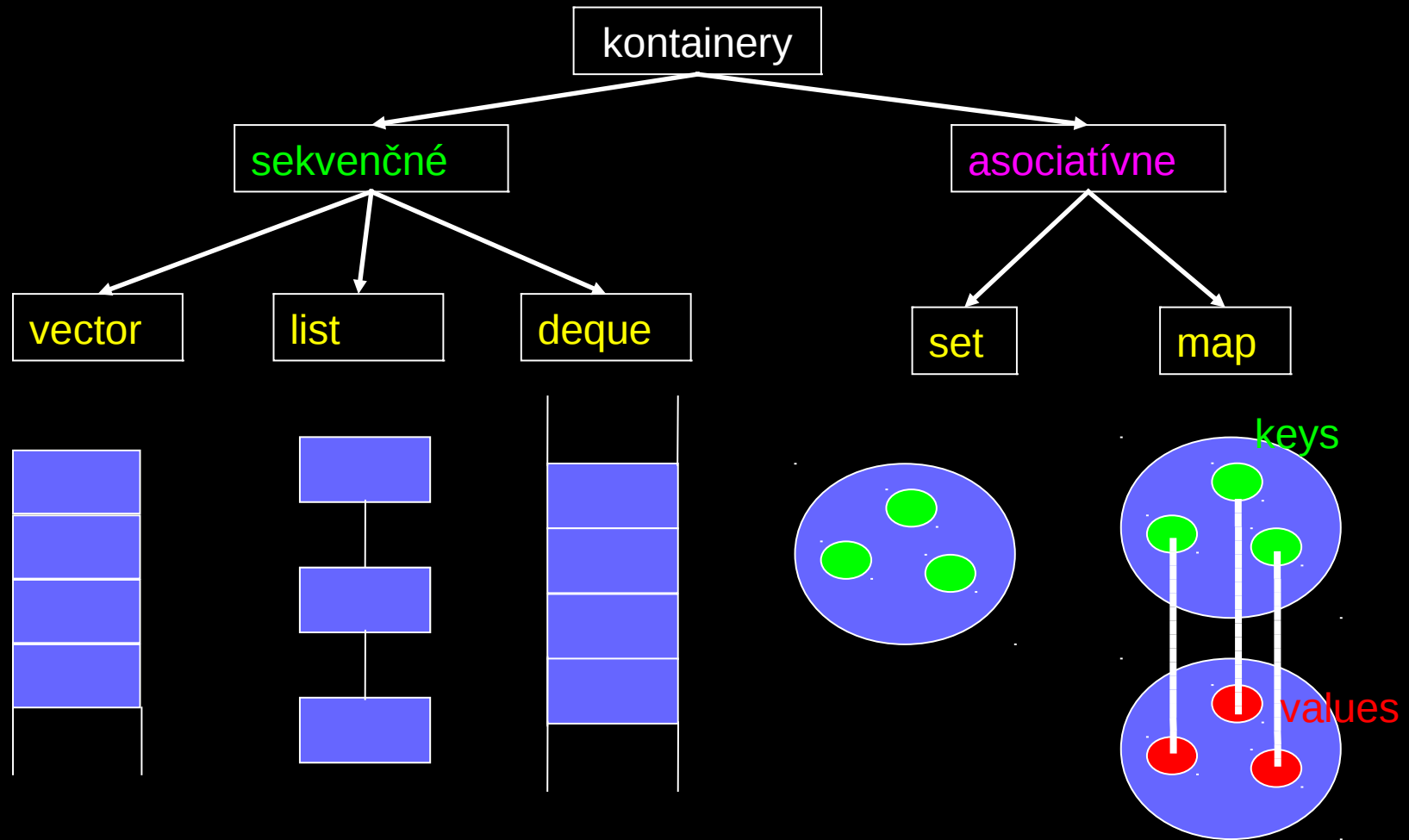


Programovacie techniky

7. set, multiset, deque, bitset, binárny
vyhľadávací strom

Kontainery



list vs deque

List je implementovaný ako dvojmo-zreťazený zoznam

Deque (double-ended queue) je implementovaná ako dynamické pole

Výslovnosť deque: „dek“

set

set = množina

Prvky nie je možné meniť

Obsahuje len rozdielne prvky

Podporuje insert a erase

```
set<int> s;
```

```
s.insert(10); s.insert(20);
```

```
s.insert(20); //nič nie vložené, 20 je už v množine
```

```
s.erase(20);
```

```
s.clear(); //zmaž všetky prvky
```

multiset

set = množina

Prvky nie je možné meniť

Obsahuje aj rovnaké prvky

Podporuje insert a erase

```
multiset<int> s;
```

```
s.insert(10); s.insert(20);
```

```
s.insert(20); //OK, duplicitné hodnoty
```

```
s.erase(20); //zmaž obidva výskyty 20
```

```
s.clear(); //zmaž všetky prvky
```

algorithm: set_intersection

```
#include <iostream>
#include <algorithm>    // std::set_intersection, std::sort
#include <vector>

int main () {
    int first[] = {5,10,15,20,25};
    int second[] = {50,40,30,20,10};
    std::vector<int> v(10);
    std::vector<int>::iterator it;

    std::sort (first,first+5);    // 5 10 15 20 25, triedenie nutné
    std::sort (second,second+5); //10 20 30 40 50

    it=std::set_intersection (first, first+5, second, second+5,
v.begin()); // 10 20, v je prienik
    return 0; }
```

algorithm: set_union

```
#include <iostream>
#include <algorithm>    // std::set_union, std::sort
#include <vector>

int main () {
    int first[] = {5,10,15,20,25};
    int second[] = {50,40,30,20,10};
    std::vector<int> v(10);
    std::vector<int>::iterator it;

    std::sort (first,first+5);    // 5 10 15 20 25
    std::sort (second,second+5);  // 10 20 30 40 50

    it=std::set_union (first, first+5, second, second+5,
v.begin()); // 5 10 15 20 25 30 40 50, v je zjednotenie
    return 0;}
```

bit set

```
#include <iostream>
#include <string>
#include <bitset>          // std::bitset

int main () {
    std::bitset<4> foo (std::string("1001"));
    std::bitset<4> bar (std::string("0011"));

    std::cout << (foo^=bar) << '\n';      // 1010 (XOR,assign)
    std::cout << (foo&=bar) << '\n';      // 0010 (AND,assign)
    std::cout << (foo|=bar) << '\n';      // 0011 (OR,assign)

    std::cout << (foo<<=2) << '\n';        // 1100 (SHL,assign)
    std::cout << (foo>>=1) << '\n';        // 0110 (SHR,assign)
```


bit set

```
std::cout << (~bar) << '\n';           // 1100 (NOT)
std::cout << (bar<<1) << '\n';          // 0110 (SHL)
std::cout << (bar>>1) << '\n';          // 0001 (SHR)

std::cout << (foo==bar) << '\n';         // false (0110==0011)
std::cout << (foo!=bar) << '\n';         // true  (0110!=0011)

std::cout << (foo&bar) << '\n';          // 0010
std::cout << (foo|bar) << '\n';          // 0111
std::cout << (foo^bar) << '\n';          // 0101

return 0;
}
```

bitset: set

```
#include <iostream>
#include <bitset>      // std::bitset
```

```
int main () {
    std::bitset<4> foo;
```

```
    std::cout << foo.set() << '\n';      // 1111
    std::cout << foo.set(2,0) << '\n';    // 1011
    std::cout << foo.set(2) << '\n';      // 1111
```

```
    return 0;
}
```

Najnižší bit



bitset: reset

```
#include <iostream>
#include <string>
#include <bitset>

int main () {
    std::bitset<4> foo (std::string("1011"));

    std::cout << foo.reset(1) << '\n';    // 1001
    std::cout << foo.reset() << '\n';     // 0000

    return 0;
}
```

bitset: flip

```
#include <iostream>
#include <string>
#include <bitset>

int main () {
    std::bitset<4> foo (std::string("0001"));

    std::cout << foo.flip(2) << '\n';    // 0101
    std::cout << foo.flip() << '\n';    // 1010

    return 0;
}
```

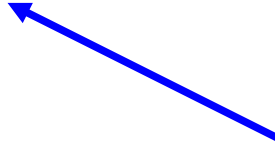
bitset: out_of_range

```
#include <iostream>
#include <string>
#include <bitset>
```

```
int main () {
    std::bitset<4> foo (std::string("0001"));

    try {
        std::cout << foo.flip(20) << '\n';    // 0101
    }
    catch (std::out_of_range& oor) {
        std::cerr << "Out of Range error: " << oor.what() << '\n';
    }

    return 0;
}
```



Konverzia int na string

```
#include <string>
#include <sstream>
```

```
int main () {
    stringstream ss;
    int i = 10;
    string s;

    ss << i;
    s = ss.str();

    return 0;
}
```

A navyše je to SAFE! (bad_alloc?)

stringstream reset

```
stringstream ss;  
ss << "nieco"; //vloží do stringstreamu
```

```
ss.str(std::string()); //stringstream je prázdny  
ss.clear(); //reset stavu streamu
```

Potrebné pre viacnásobné použitie
toho istého stringstreamu!

Map: zložitosť

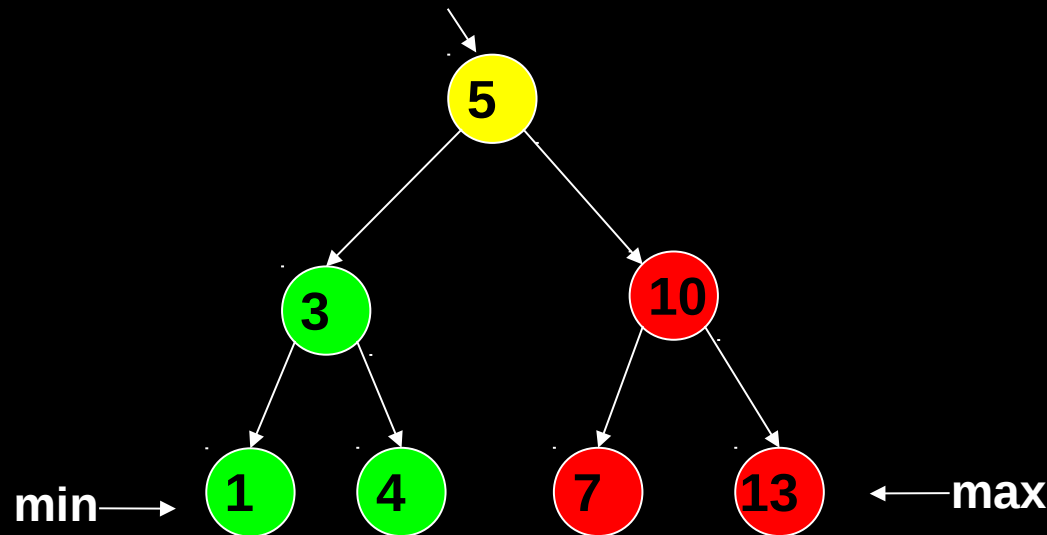
hodnotenie[„Imro“] = 4; *//O(?)*

Map je implementovaná ako binárny
vyhľadávací strom

Binárny vyhľadávací strom

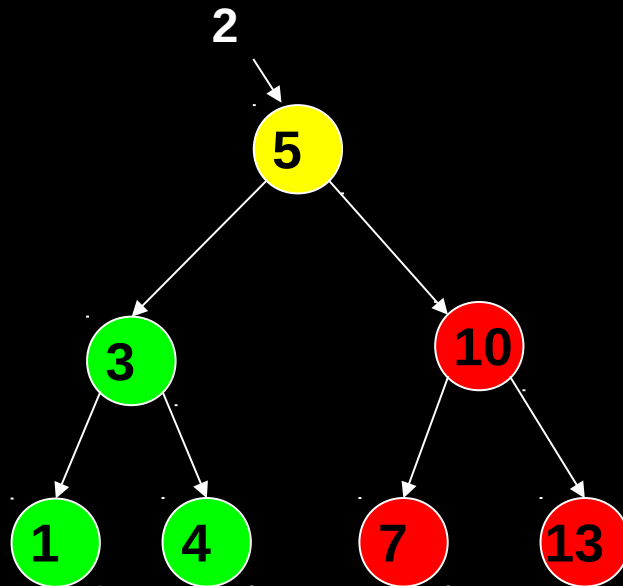
Dátová štruktúra na rýchle vyhľadávanie: $O(\log n)$

Binárny vyhľadávací strom: prvky v ľavom/pravom podstrome ľubovoľného uzla sú menšie/väčšie



Binárny vyhľadávací strom

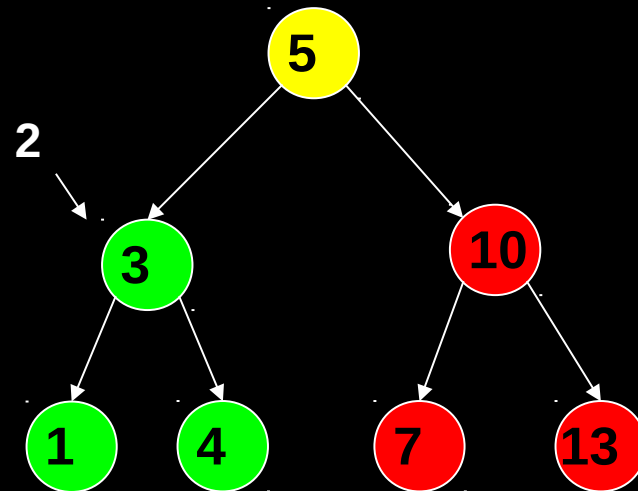
Obsahuje strom číslo 2?



Binárny vyhľadávací strom

Obsahuje strom číslo 2?

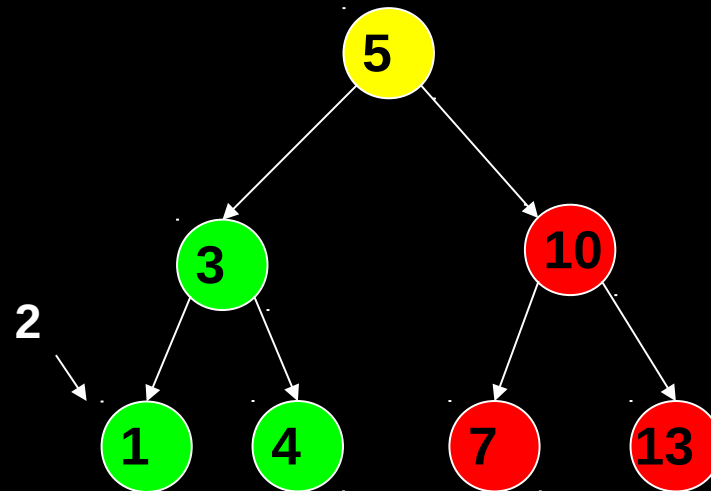
$2 < 5$: pokračuj v ľavom podstrome



Binárny vyhľadávací strom

Obsahuje strom číslo 2?

$2 < 3$: pokračuj v ľavom podstrome

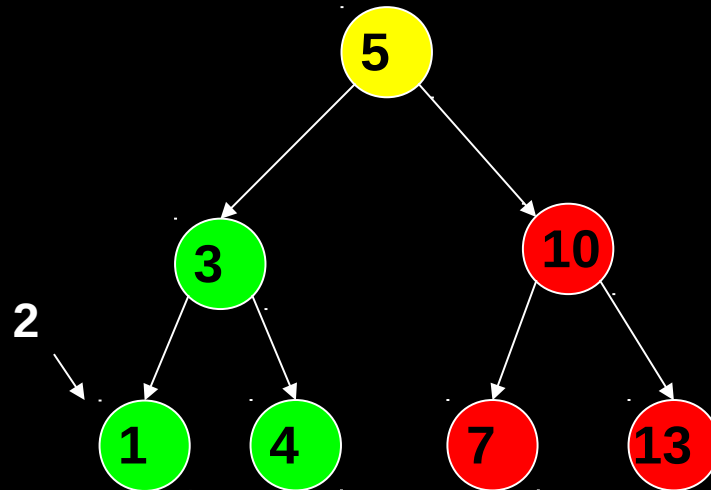


Binárny vyhľadávací strom

Obsahuje strom číslo 2?

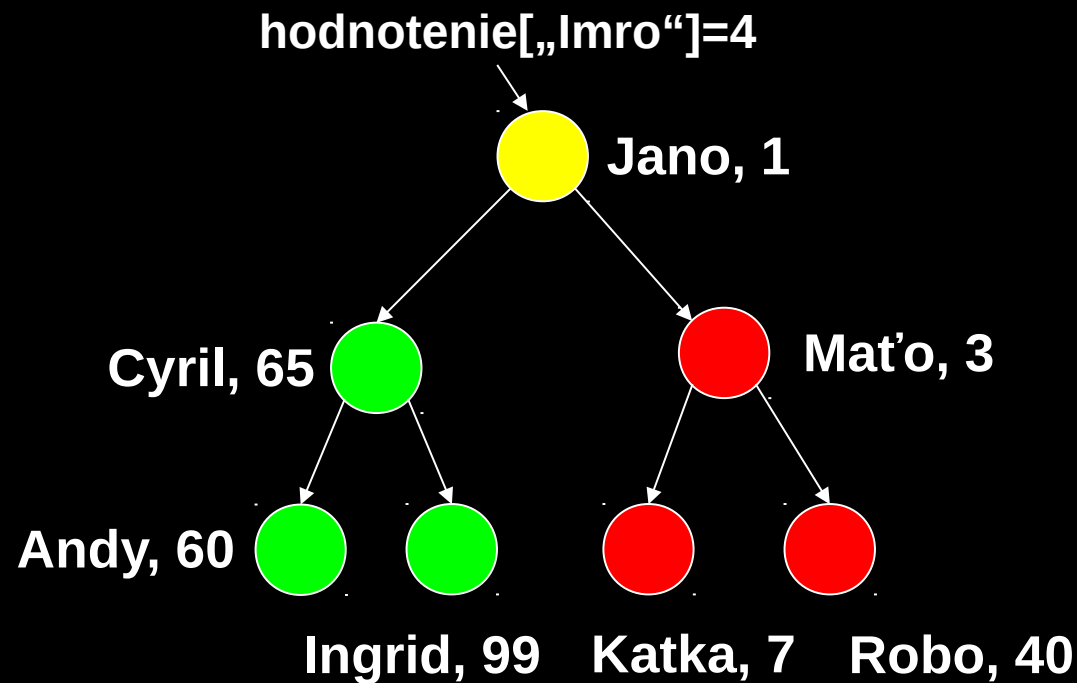
Uzol 1 nemá potomkov

$1 \neq 2$, 2 sa nenachádza v strome



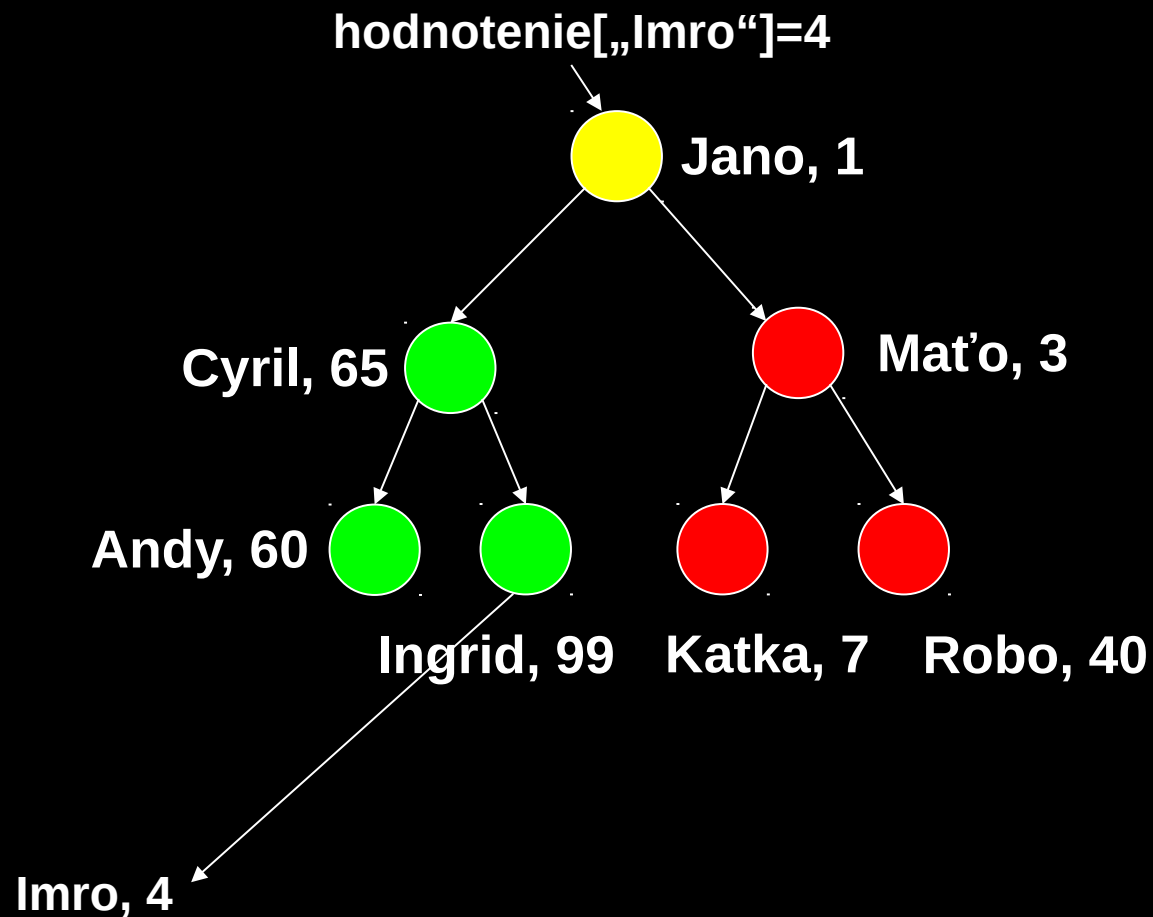
Počet porovnaní: $O(\log n)$, teda hĺbka stromu

Čo spravíme s Imrom?



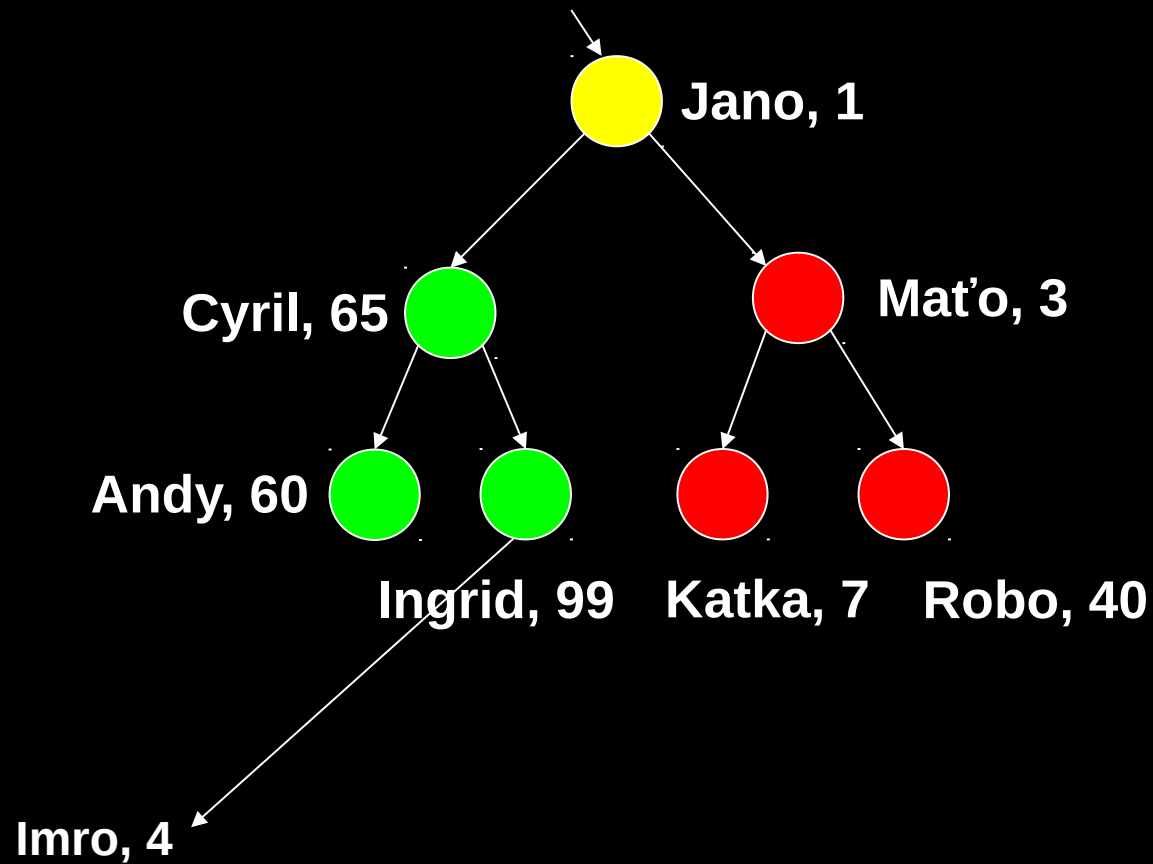
„Andy“ < „Cyril“ < „Imro“ < „Ingrid“ < „Jano“

Čo spravíme s Imrom?



insert

Balancing??



Hľadanie v usporiadanom poli

Je 2 v poli?

Stred

-60, -12, 0, 4, 8, 9, 56, 98, 100, 267, 667

$2 < 9$

-60, -12, 0, 4, 8

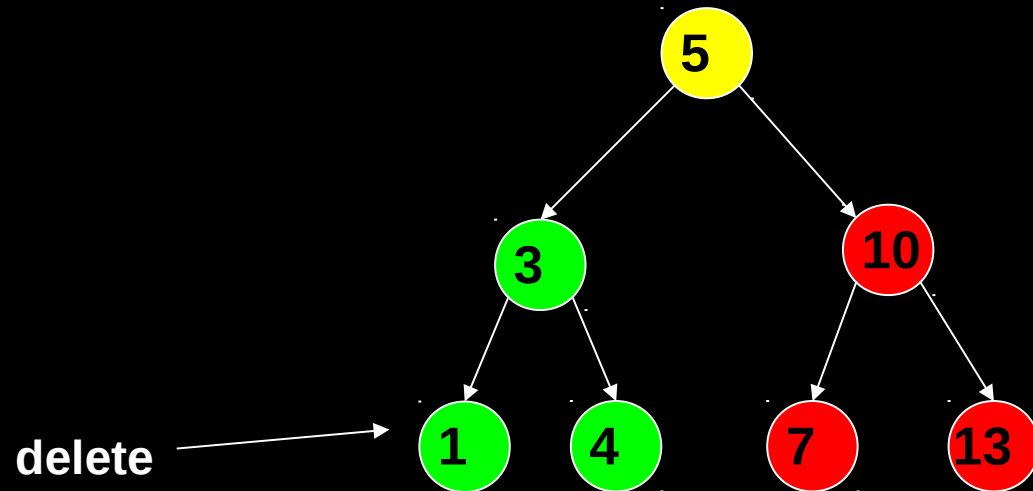
$2 > 0$

4, 8

$2 < 4$, koniec, 2 nie je v poli

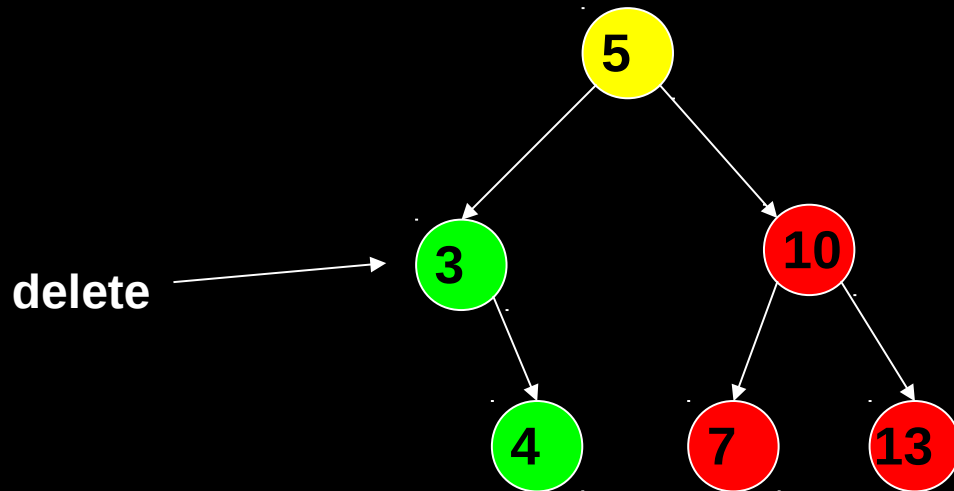
delete

delete leaf: jednoduché



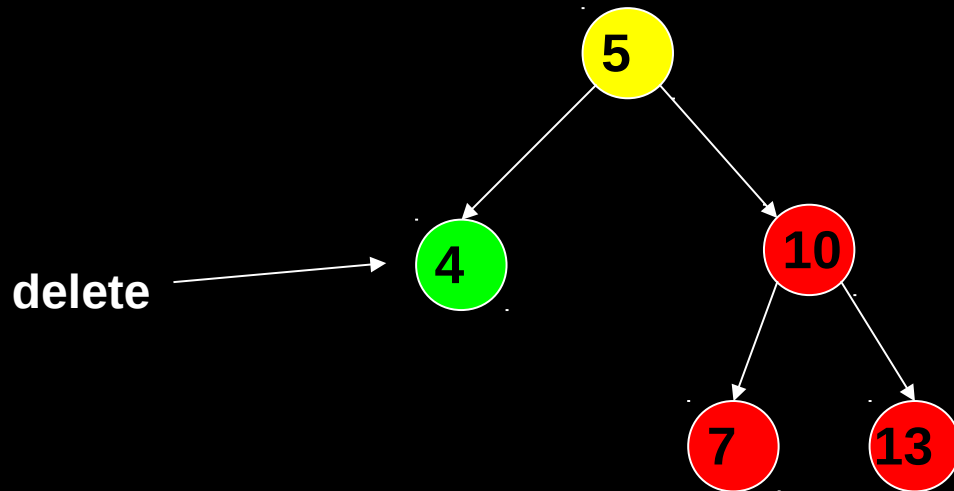
delete

delete node s jedným potomkom



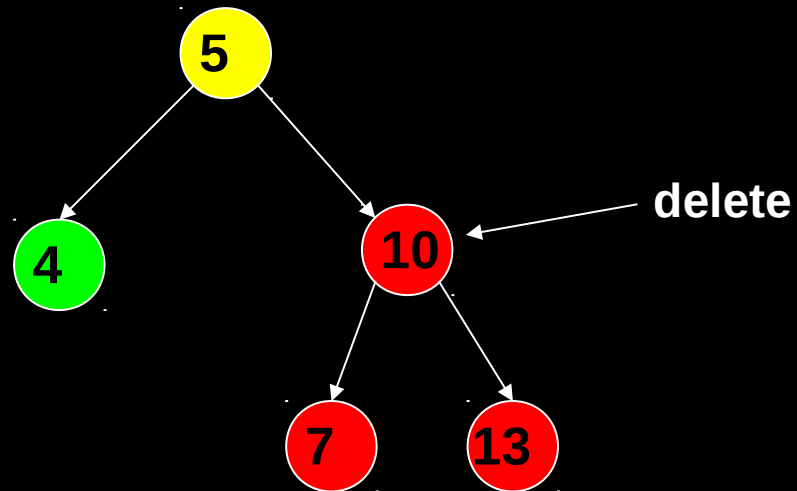
delete

delete node s jedným potomkom



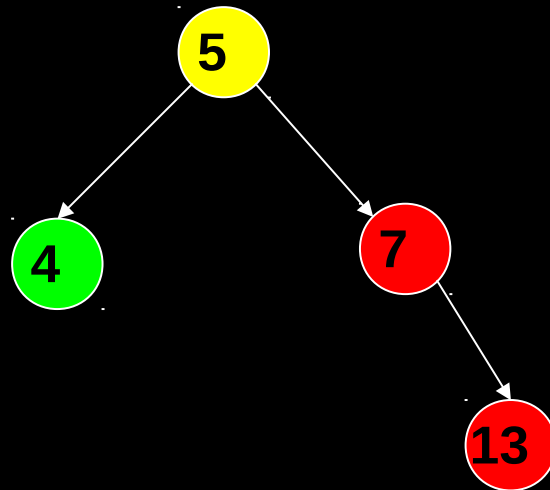
delete

delete node s dvoma potomkami



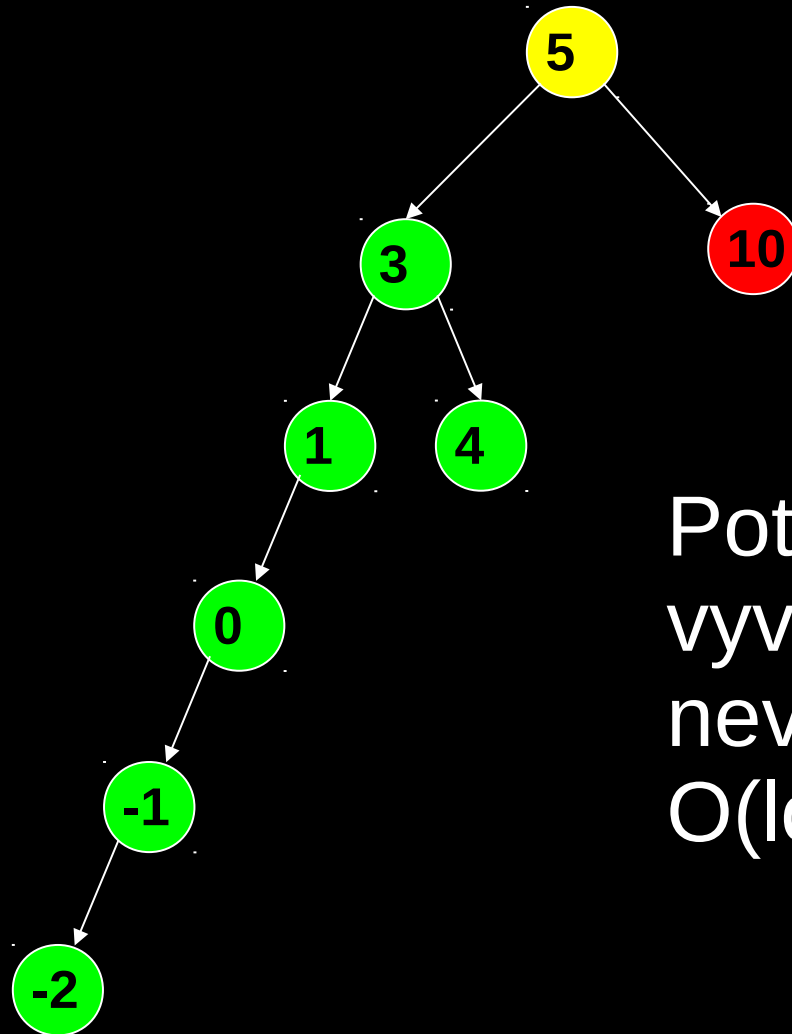
delete

delete node s dvoma potomkami
nahrad' jedným potomkom (možná voľba)



Binárny vyhľadávací strom

Nevyvážený binárny vyhľadávací strom



Potrebuje vyvažovanie, ináč
nevieme dosiahnuť
 $O(\log n)$

Zložitosť bez vyvažovania

	Average	Worst-case
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$

Zložitosť: C++ špecifikácia

	Average	Worst-case
Search	$O(\log n)$	$O(\log n)$
Insert	$O(\log n)$	$O(\log n)$
Delete	$O(\log n)$	$O(\log n)$

Vyvážený binárny vyhľadávací strom: **red-black tree**

Binary search tree (BST)

1960: P.F. Windley, A.D. Booth, A.J.T. Colin, T.N. Hibbard



A.D. Booth



A.J.T. Colin

Timeline

1730, Stirling (UK): aproximácia faktoriálu

1945, von Neumann (USA): merge sort

1960, Hoare (UK): quick sort

1960, P.F. Windley, A.D. Booth, A.J.T. Colin,
T.N. Hibbard (UK): binary search tree

1964, J. W. J. Williams (UK, Canada): heap
sort

1972, Dennis Ritchie (USA): jazyk C

1983, Bjarne Stroustrup (Denmark, USA):
jazyk C++

1995, James Gosling (USA): jazyk Java

1998, Larry Page, Sergey Brin (USA): Google

Pseudocode

Pseudocode je štandardná forma zápisu algoritmu

BINARY-SEARCH(x, T, p, r)

```
1   $low = p$ 
2   $high = \max(p, r)$ 
3  while  $low < high$ 
4       $mid = \lfloor (low + high) / 2 \rfloor$ 
5      if  $x \leq T[mid]$ 
6           $high = mid$ 
7      else  $low = mid + 1$ 
8  return  $high$ 
```

Zaokrúhľovanie
nadol

x : prvok, ktorý treba nájsť

T : pole prvkov

p : pozícia prvého prvku

r : pozícia posledného prvku

Overflow pri sčítaní

Môže nastat' overflow, ak low a high sú veľké čísla:

$\text{mid} = (\text{low} + \text{high}) / 2;$

Lepšie:

$\text{mid} = \text{low} + (\text{high} - \text{low}) / 2;$

Úloha

Naimplementuj binary search:

- Iteratívne
- Rekurzívne

Zložitosť: C++11

hodnotenie[„Imro“] = 4; $O(\log n)$

roztrieď vector $O(n \log n)$

Pre znudených...

