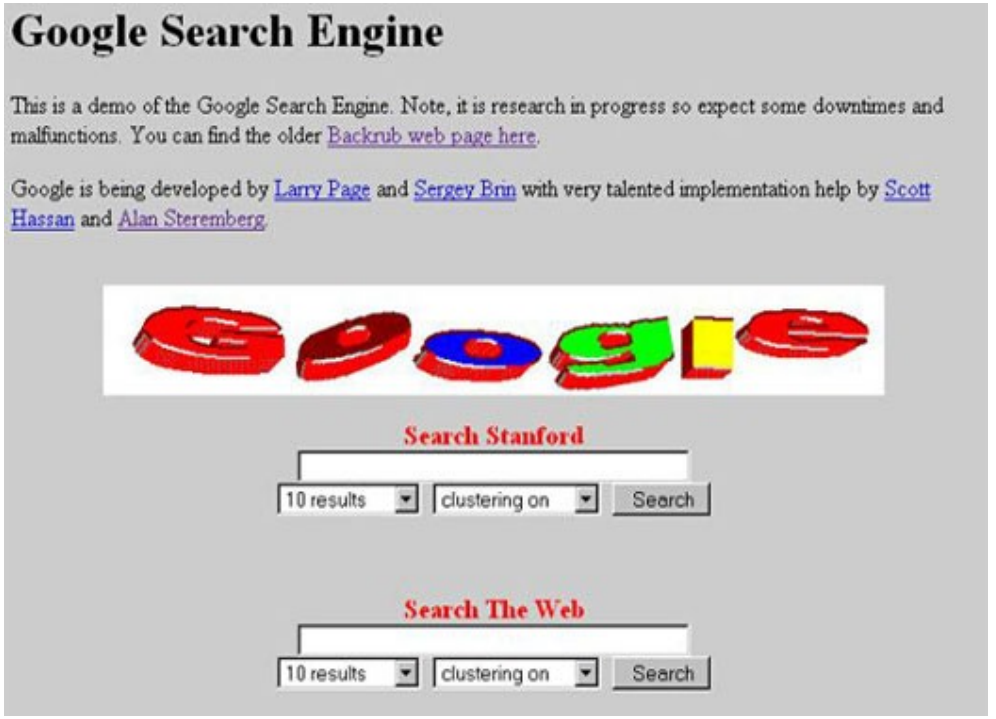
STU FEI

Programovacie techniky

8. indexovanie, array, deque, stack, queue, map, multimap, set, multiset, binárny vyhľadávací strom

Google

google.stanford.edu



STU FEI

The Anatomy of a Large-Scale Hypertextual Web Search Engine

Sergey Brin and Lawrence Page

Computer Science Department,
Stanford University, Stanford, CA 94305, USA
sergey@cs.stanford.edu and page@cs.stanford.edu

Abstract

In this paper, we present Google, a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext. Google is designed to crawl and index the Web efficiently and produce much more satisfying search results than existing systems. The prototype with a full text and hyperlink database of at least 24 million pages is available at http://google.stanford.edu/
To engineer a search engine is a challenging task. Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of millions of queries every day. Despite the importance of large-scale search engines on the web, very little academic research has been done on them. Furthermore, due to rapid advance in technology and web proliferation, creating a web search engine today is very different from three years ago. This paper provides an in-depth description of our large-scale web search engine -- the first such detailed public description we know of to date. Apart from the problems of scaling traditional search techniques to data of this magnitude, there are new technical challenges involved with using the additional information present in hypertext to produce better search results. This paper addresses this question of how to build a practical large-scale system which can exploit the additional information present in hypertext. Also we look at the problem of how to effectively deal with uncontrolled hypertext collections where anyone can publish anything they want.

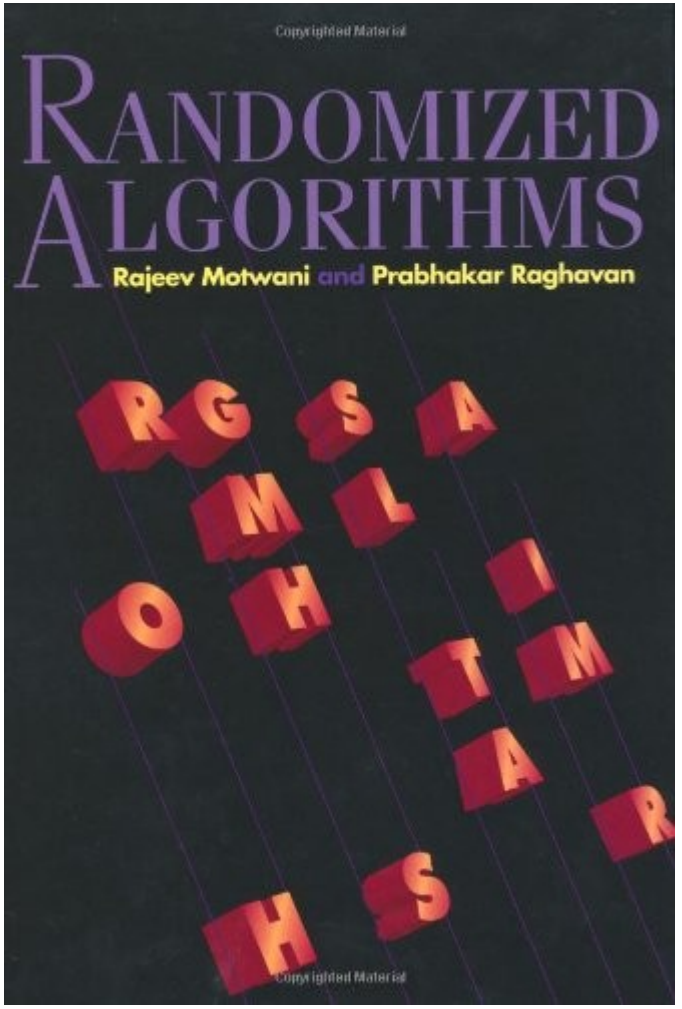
Keywords

World Wide Web, Search Engines, Information Retrieval, PageRank, Google

1. Introduction

(Note: There are two versions of this paper -- a longer full version and a shorter printed version. The full version is available on the web and the conference CD-ROM.)

Rajeev Motwani



Motwani mal veľký vplyv na Brina a Pagea.
Motwani bol profesor na Stanforde.

Inverzný zoznam

Doc id	Zoznam slov
1	Na, PT, je, nuda
2	Kedy, zacneme, s, C++
3	Este, 2, hod, do, konca
4	Na, PT, nie, je, nuda

Inverzný zoznam (index ako na konci knihy):

Slovo	Doc id
Na	1, 4
PT	1, 4
nie	4
je	1, 4
nuda	1, 4
…	

STL kontajnery

Sekvenčné kontajnery: vector, array, list, forward_list, deque

Sekvenčné adaptéry: stack, queue, priority_queue (heap)

- Využívajú napr. deque ako kontajner
- Bez možnosti iterácie

Asociatívne kontajnery: map, multimap, set, multiset

- Využívajú vyvážené binárne stromy
- Vhodné ako dátové štrukrúry pre indexovanie (map, multimap)

Neusporiadané asociatívne kontajnery:
unordered_set, unordered_multiset, unorderer_map, unordered_multimap

Abstract data type

ADT je model pre dátový typ so sémantikou operácií

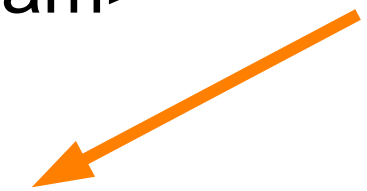
Dátová štruktúra, na rozdiel od ADT, upresňuje spôsob implentácie

Príklady:

- int je ADT s prvkami …, -2, -1, 0, 1, 2, … a definovanými operáciami +, -, *, /.
- stack je ADT s definovanými operáciami top, push, pop a LIFO (last in, first out) štruktúrou

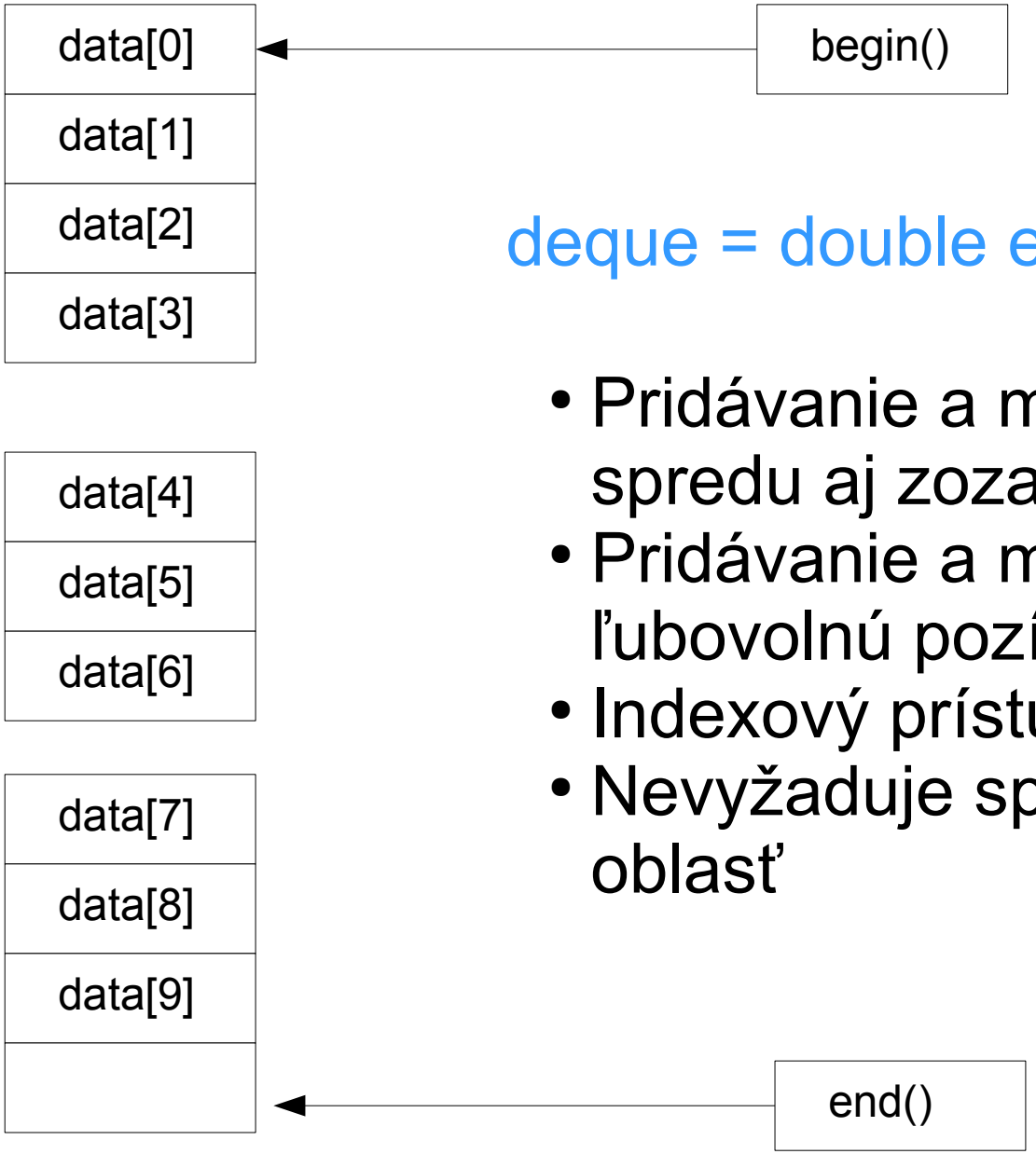
array

```
#include <array>
#include <iostream>

int main() {
  std::array<int, 3> pole = {1, 2, 3};

  std::cout << pole.front(); //1
  std::cout << pole.back(); //3

  return 0;
}
```



Podobné ako std::vector, ale s fixnou dĺžkou.
Lepšie ako C pole, podporuje výnimky, typová bezpečnosť.

deque



deque = double ended queue

- Pridávanie a mazanie prvkov spredu aj zozadu, O(1)
- Pridávanie a mazanie prvkov na ľubovolnú pozíciu, O(n)
- Indexový prístup
- Nevyžaduje spojitú pamäťovú oblasť

deque

```
#include <deque>
#include <iostream>

int main() {
  std::deque<int> dq;
  dq.push_back(1);
  dq.push_front(0);

  std::cout << dq.front(); //0
  std::cout << dq.back(); //1

  std::cout << dq.at(1); //1

  return 0;
}
```

stack

```
#include <iostream>
#include <stack>

int main () {
    std::stack<int> mystack;

    for (int i = 0; i < 5; ++i) {
       mystack.push(i);
    }

    std::cout << mystack.size(); //5

    return 0;
}
```

V kontajneri sú int objekty

stack

```
#include <iostream>
#include <stack>

int main () {
    std::stack<int> mystack;

    for (int i = 0; i < 5; ++i) {
        mystack.push(i);
    }

    while (!mystack.empty()) { //empty() = je prázdny?
     mystack.pop();
    }

    std::cout << mystack.size(); //0
    return 0; }
```

stack

```
#include <iostream>
#include <stack>
#include <utility>

int main() {
  std::stack<int> s0, s1;
  s0.push(0);
  s1.push(1);

  std::swap(s0, s1); //výmena

  std::cout << s0.top(); //1, v s0 je 1

  return 0;
}
```

queue

```
#include <iostream>
#include <queue>

int main () {
    std::queue<int> myq;

    for (int i = 0; i < 5; ++i) {
       myq.push(i);
    }

    std::cout << myq.front(); //0, front(), nie top()

    return 0;
}
```

priority_queue (heap)

```
#include <queue>
#include <iostream>

int main() {
  std::priority_queue<int> q;

  for(int n : {1,8,5,6,3,4,0,9,7,2}) {
    q.push(n);
  }

  while(!q.empty()) {
    std::cout << q.top() << " "; //9 8 7 6 5 4 3 2 1 0
    q.pop();
  }

  return 0; }
```

map

```
#include <map>
#include <iostream>
using namespace std;

int main () {
    std::map<string, int> mMap;

    mMap["Martin"] = 1;
    mMap["Imro"] = 4;
    mMap["Walter"] = 5;

    cout << mMap["Imro"] << endl;  //4
    return 0;
}
```

Výnimka

```
#include <map>
#include <iostream>
using namespace std;

int main () {
  std::map<string, int> mMap;

  mMap["Martin"] = 1; mMap["Imro"] = 4;
  mMap["Walter"] = 5;

  try {
    cout << mMap.at("Anna");  //neobsahuje "Anna"
  } catch(...) {              //zachytí všetky výnimky
    cout << "Exception!";
  }

  return 0; }
```

map

mMap["Martin"] = 1

- reťazec (string)"Martin" je kľúč (key)
- 1 je asociovaná hodnota (value)

map

```
#include <map>
#include <iostream>
using namespace std;

int main () {
  std::multimap<string, int> mMap;

  mMap.insert({"Martin", 1}); mMap.insert({"Martin", 1});
  mMap.insert({"Imro", 4}); mMap.insert({"Walter", 5});

  for(auto& i : mMap) {
    cout << i.first << " " << i.second << endl;
  }

  return 0;
}
```

set

```
#include <set>
#include <iostream>
using namespace std;

int main () {
  std::set<int> s;

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

  for(auto& i : s) {
    cout << i << " "; //10 20
  }

  return 0;
}
```

Zoradené a bez duplikátov

multiset

```
#include <set>
#include <iostream>
using namespace std;

int main () {
  std::multiset<int> s;

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

  for(auto& i : s) {
    cout << i << " "; //10 10 20 20
  }

  return 0;
}
```

Zoradené a s duplikátmi

ordered, unordered

Find, insert, erase O(log n), pomocou binárneho vyhľadávacieho stroma:

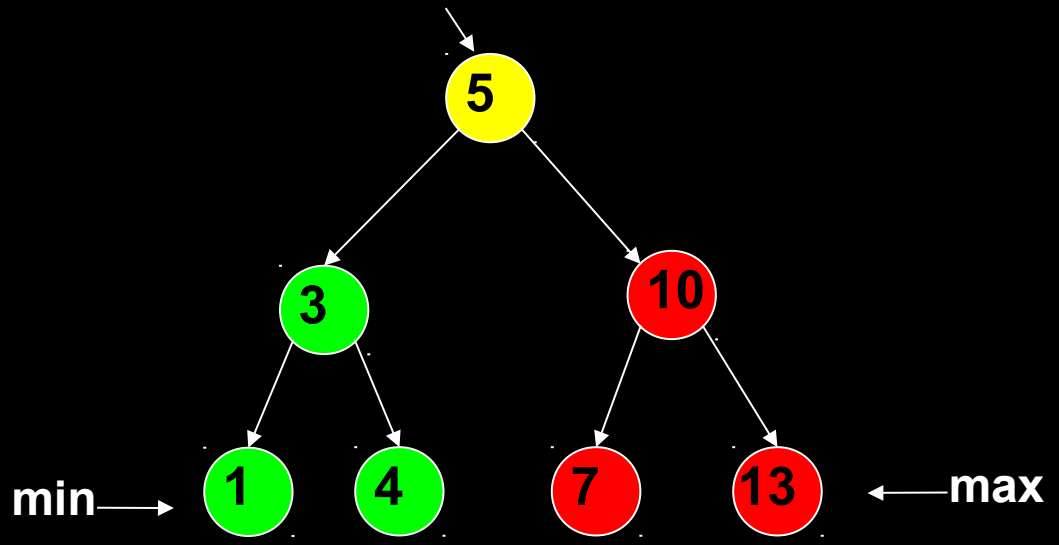
map
multimap
set
multiset

Find, insert, erase O(1), ale je potrebné hašovanie (neskôr):

unordered_map
unordered_multimap
unordered_set
unordered_multiset

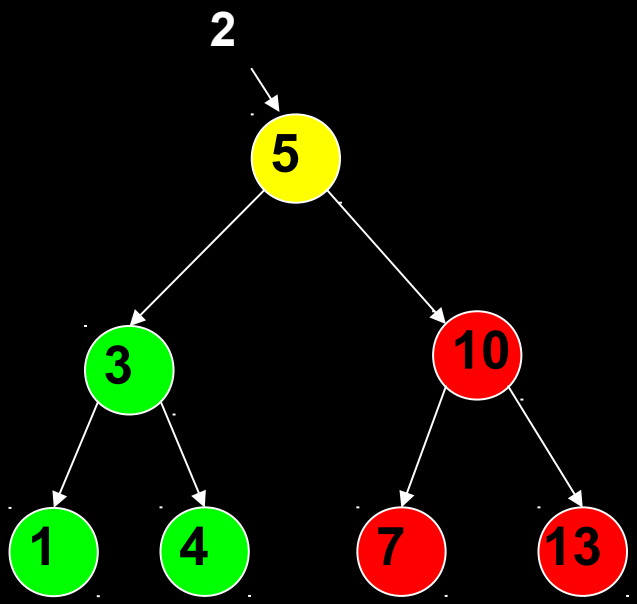
Binárny vyhľadávací strom

Dátová štruktúra na rýchle vyhľadávanie: O(log n)
Binárny vyhľadávací strom: prvky v ľavom/pravom podstrome ľubovoľného uzla sú menšie/väčšie



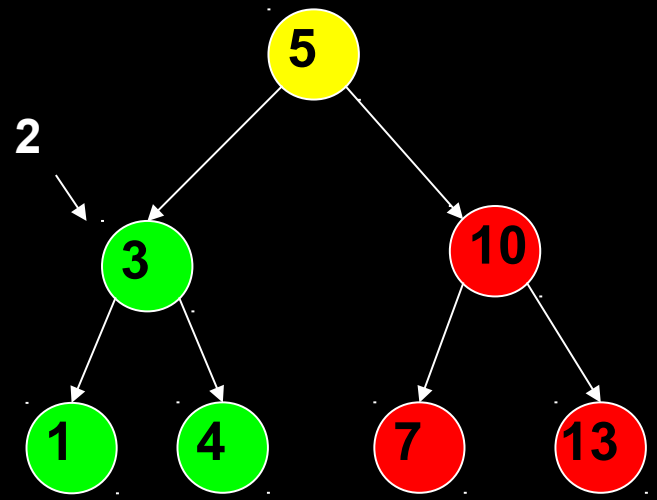
Binárny vyhľadávací strom

Obsahuje strom číslo 2?



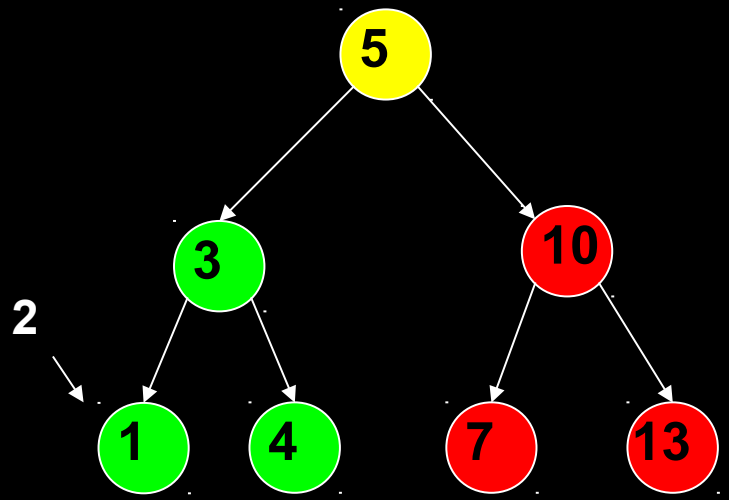
Binárny vyhľadávací strom

Obsahuje strom číslo 2?
2<5: pokračuj v ľavom podstrome



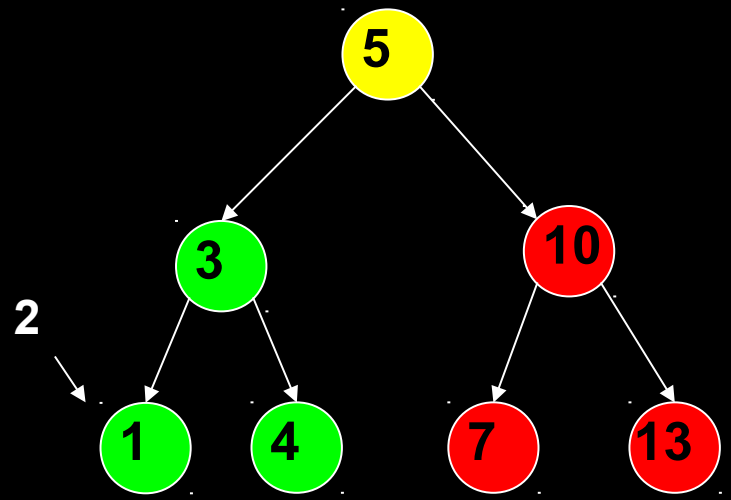
Binárny vyhľadávací strom

Obsahuje strom číslo 2?
2<3: pokračuj v ľavom podstrome



Binárny vyhľadávací strom

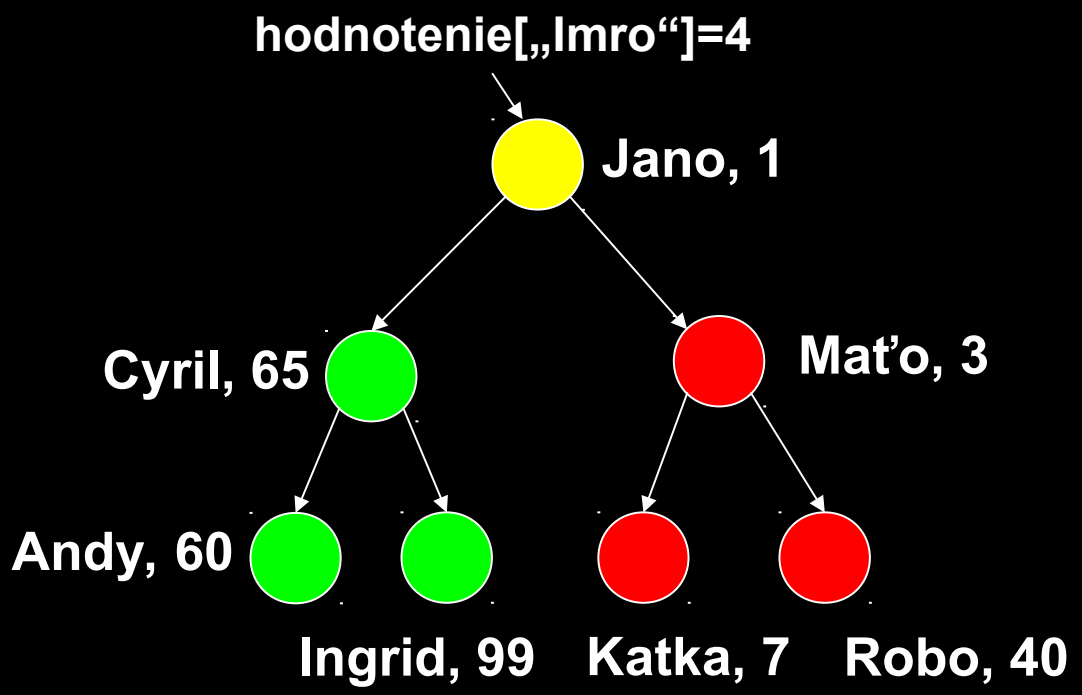
Obsahuje strom číslo 2?
Uzol 1 nemá potomkov
1≠2, 2 sa nenachádza v strome



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

map

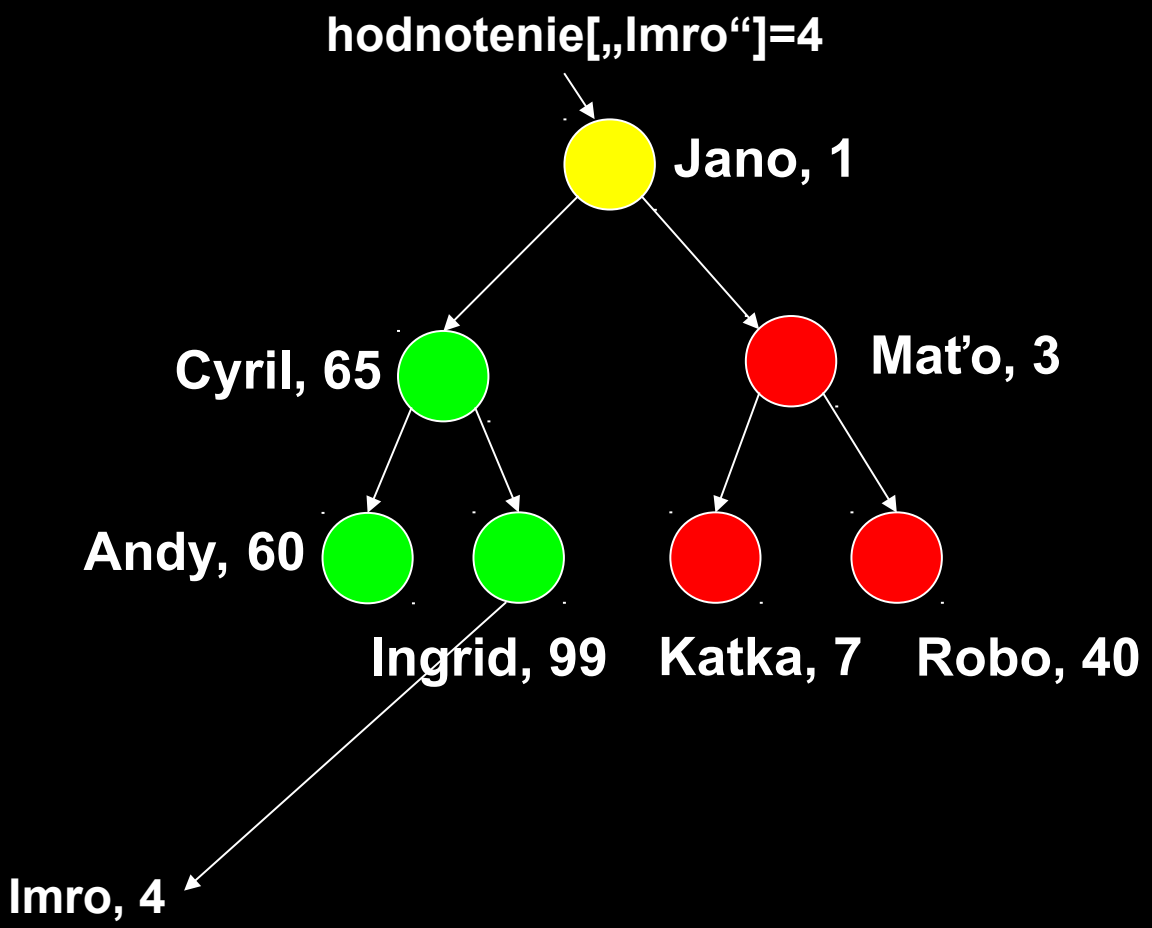
Čo spravíme s Imrom?



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

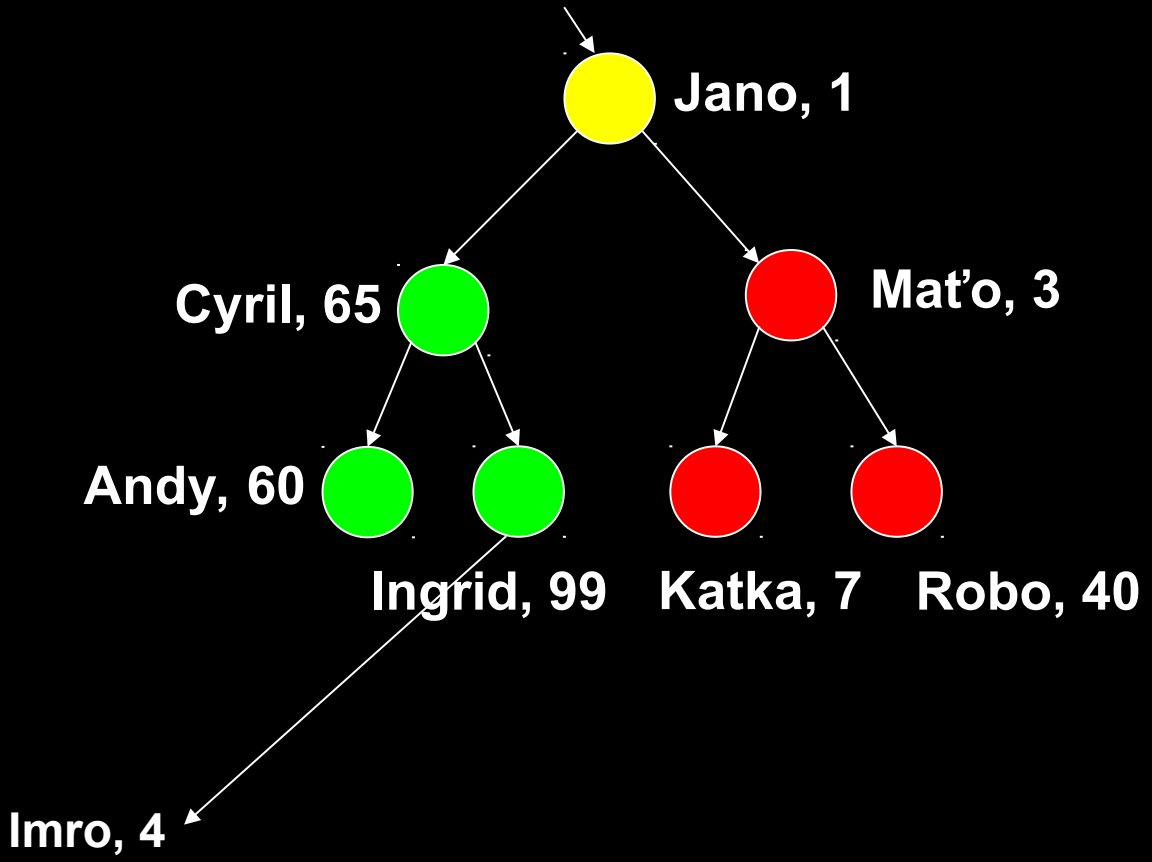
map

Čo spravíme s Imrom?



insert

Balancing??



Hľadanie v usporiadanom poli

Je 2 v poli?

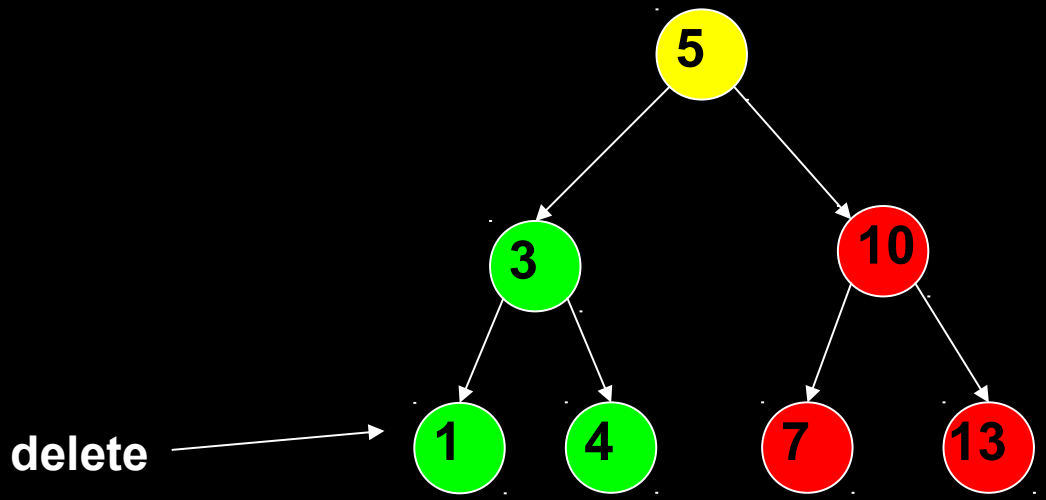
-60, -12, 0, 4, 8, 9, 56, 98, 100, 267, 667
2<9

-60, -12, 0, 4, 8
2>0

4, 8
2<4, koniec, 2 nie je v poli

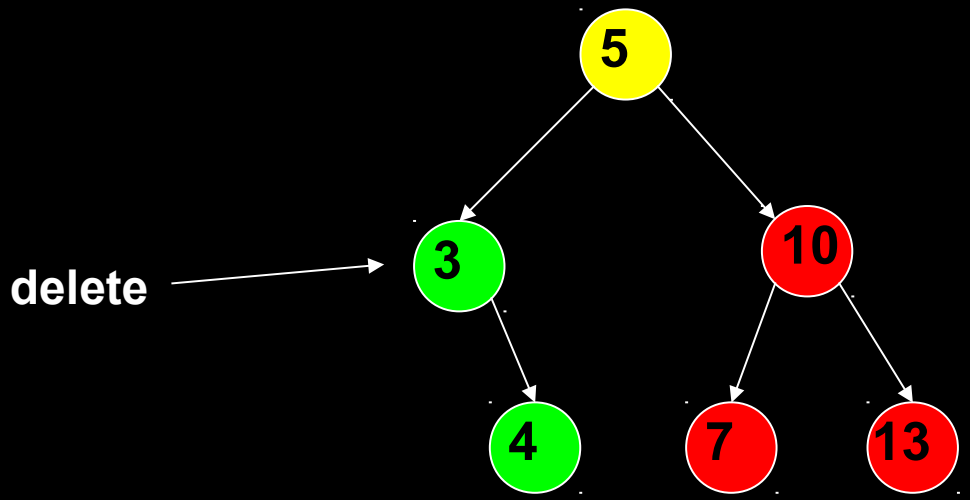
delete

delete leaf: jednoduché



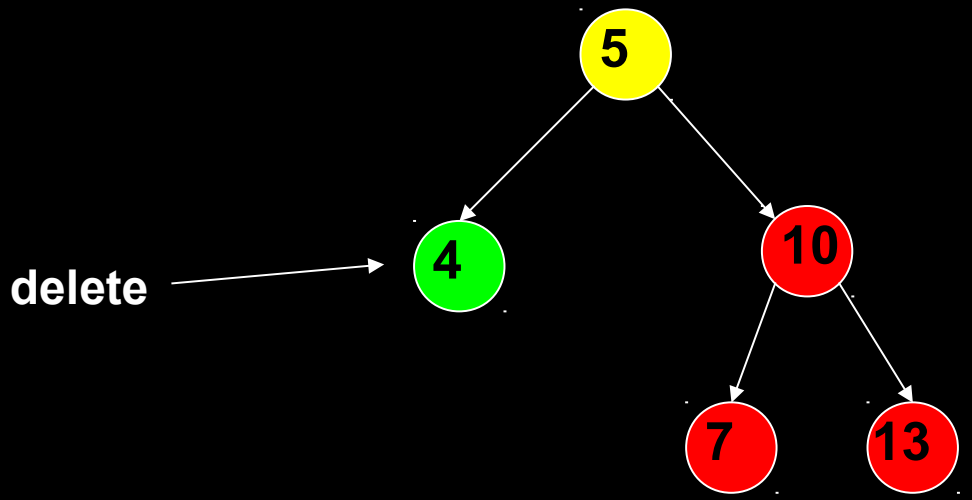
delete

delete node s jedným potomkom



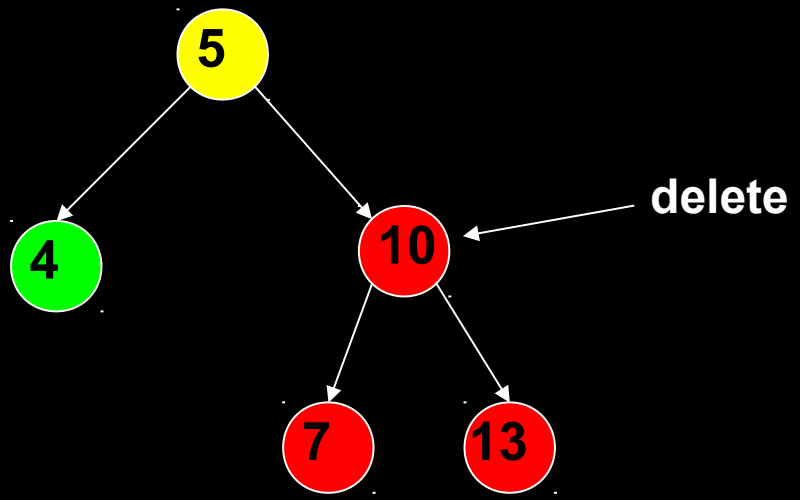
delete

delete node s jedným potomkom



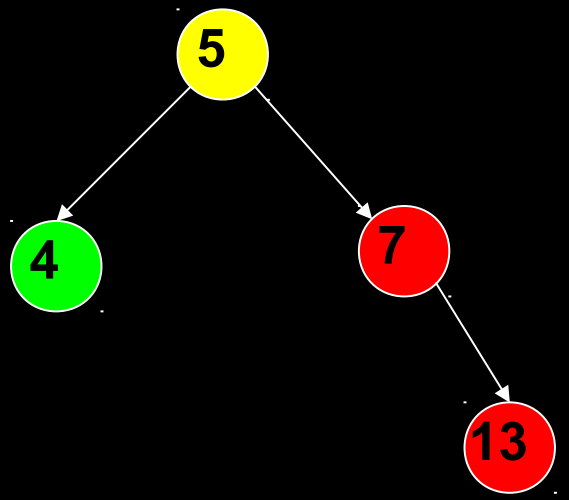
delete

delete node s dvoma potomkami



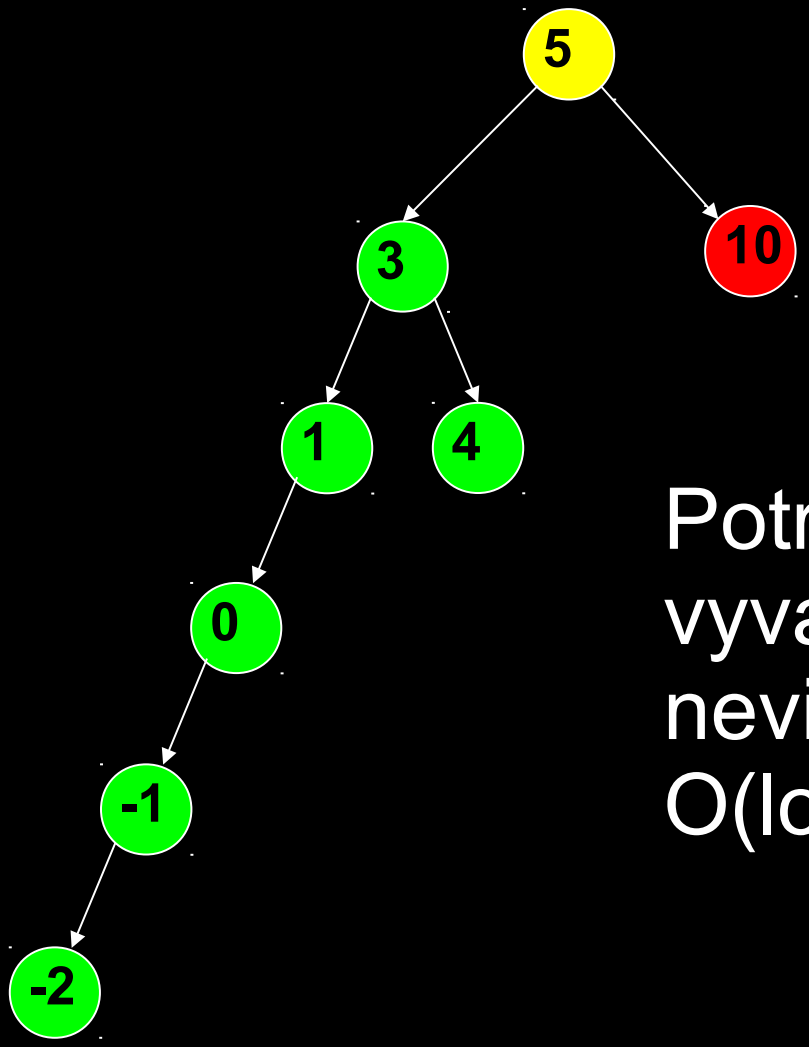
delete

delete node s dvoma potomkami
nahraď jedným potomkom (možná voľba)



Binárny vyhľadávací strom

Nevyvážený binárny vyhľadávací strom



Potrebujeme 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)

Vyvážený binárny vyhľadávací strom: red-black tree

Binárny vyhľadávací strom

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

Binárne hľadanie

```
BINARY-SEARCH(x, T, p, r)
1  low = p
2  high = max(p, r      )
3  while low < high
4        mid = ⌊(low + high)/2⌋
5        if x ≤ 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 nastať overflow, ak low a high sú veľké čísla:

```
mid = (low + high) / 2;
```

Lepšie:

```
mid = low + (high - low) / 2;
```

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

Úloha

Ako urobiť vektor zoznamov?

Alebo zoznam zásobníkov?

Alebo … be happy u dont have to do it in C!

Úloha

Máme zdrojový kód (textový súbor) a máme zistiť, či v kóde sú korektne párované zátvorky { }

Príklad: {…{…}..}..} : zátvorky nie sú párované

Ako použiť stack??