STU FEI

Programovacie techniky

5. Google (ako funguje), úvod do C++, triedy, objekty

Heap sort: pseudokód

```
MAX-HEAPIFY(A, i)
1  l = LEFT(i)
2  r = RIGHT(i)
3  if l ≤ A.heap-size and A[l] > A[i]
4       largest = l
5  else largest = i
6  if r ≤ A.heap-size and A[r] > A[largest]
7       largest = r
8  if largest ≠ i
9       exchange A[i] with A[largest]
10      MAX-HEAPIFY(A, largest)
```

1. iterácia: *i* je koreň

l je ľavý potomok

r je pravý potomok

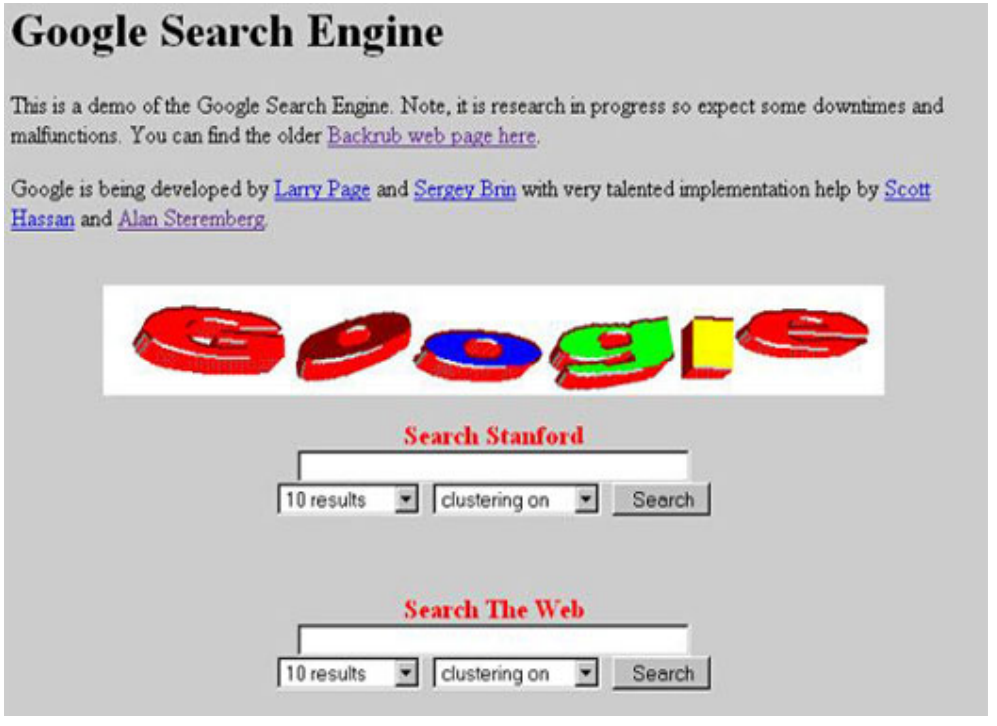
Rekurzia

Heap sort na poli A, uzol *i* má potomkov *l* a *r*

Potreba implementovať funkcie LEFT a RIGHT

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.

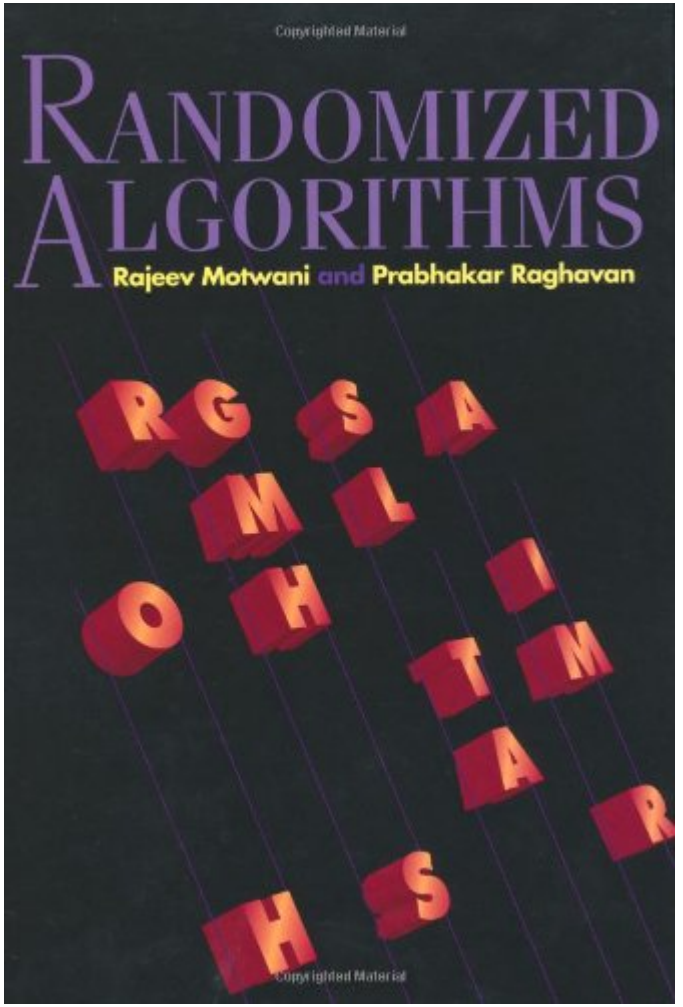
Keywords

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

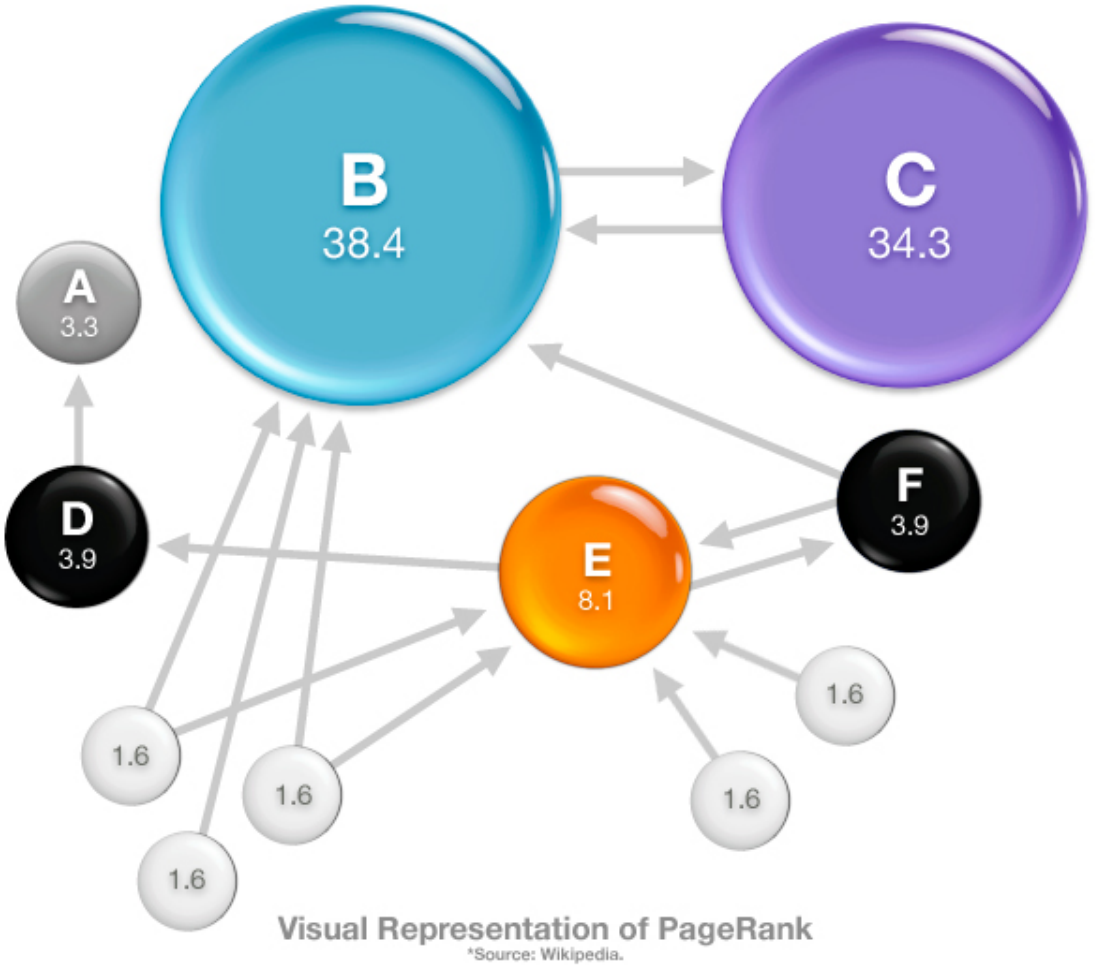
Doc id	Zoznam slov
1	Na, PT, je, nuda
2	Kedy, zacneme, s, C++
3	Este, 2, hod, do, konca
4	Na, PT, nie, je, nuda

Inverzný zoznam (index ako na konci knihy):

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

…

Page rank



Výsledky hľadania treba zoradiť podľa Page ranku

Indexovanie

Flex:

- the fast lexical analyzer
- Používaný pri kompilácii
- Podporuje regulárne výrazy
  - (a|b)* = {"", "a", "b", "aa", "ab", "ba", "bb", "aaa", ...}
- Ideálne pre extrakciu URL z dokumentu a pre extrakciu slov z dokumentu

http://en.wikipedia.org/wiki/Flex_lexical_analyser

CRC hashing

CRC = cyclic redundancy check

CRC32, CRC64: 32-bitová, 64-bitová verzia

Prekonvertuje:

- „nuda“ na 1897896858
- „http://www.sme.sk/stranka/nieco.htm“ na 68768766878
- „Kedy“ na 98977979

Reťazec prekonvertuje na číslo

hashing

Hashing má nenulovú pravdepodobnosť konfliktu: dva reťazce môžu byť prekonvertované na to isté číslo

CRC32: nevyhovujúce

CRC64: lepšie

Výzvy

Google za 3 mesiace?

Vo verzii z roku 1998-2000: OK

Morálne ponaučenie: treba hľadať zaujímavý biznis model

Hello world!

Hello world v C

```
#include <stdio.h>

int main() {

    fprintf(stdout,”Hello world!”);


    return 0;
}
```

Hello world v C++

```
#include <iostream>

int main() {

    std::cout << “Hello world!”;   //vypis na std. vystup

    return 0;
}
```

Menný priestor (namespace)

```
name::nieco;  //nieco je premenná
                  alebo funkcia/metóda

namespace Stack {

    void push(char);
    char pop ();
}
```

namespace: príklad

```
void f() {

    Stack::push('c');
    if(Stack::pop() != 'c')
        std::cout << „Impossible!“;
}
```

namespace: príklad

```
namespace N1 { int x; }
namespace N2 { int x; }
int globalna_premenna;

int main() {
   int n;
   n = 4;     //netreba písať menný priestor
   N1::x = 1;
   N2::x = 2;
   ::globalna_premenna = 5;
}
```

#include

Názvy hlavičkových súborov v C++ neobsahujú príponu .h

Môžeme využívať aj C-čkové knižnice

<stdio.h> - bez menného priestoru

<cstdio> - v mennom priestore std

```
std::printf(„Hello“);
```

i/o operátory

cin,cout - identifikátory štandardného vstupu/výstupu (stdin,stdout v C)

std::cin, std::cout

>>, << operátory vstupu/výstupu

Možnosť formátovania podľa typu premennej

Možnosť riadiť manipulátormi

i/o operátory

```
#include <iostream>
#include <iomanip>  //pre manipulátory

int main() {
    int n;
    std::cout << “zadaj cislo: ”;    //vypíše na std. výstup
    std::cin   >> n;          //prečíta číslo zo std. vstupu

    std::cout << “desiatkova hodnota n je”  << n <<
std::endl;
    std::cout << “hexa hodnota n je” << std::hex << n <<
std::endl;

    return 0;
}
```

using namespace

```
#include <iostream>



int main() {
    int n;
    std::cout << “Hello
world!”;

    return 0;
}
```

```
#include <iostream>

using namespace std;

int main() {
    int n;
    cout << “Hello world!”;


    return 0;
}
```

using

```
#include <iostream>

using std::cout;

int main() {

    cout << “I love PT!” << std::endl;

    return 0;
}
```

endl je to isté ako '\n' + flush stream

Dynamická alokácia pamäte

```
double *pvalue = new double;
float  *pole = new float[5];
char   *str = new char[20];
```

Uvoľnenie pamäte:

```
delete pvalue ;
delete[] pole;     //zátvorky [] sú nutné!!!
delete[] str;
```

Dynamická alokácia pamäte

Alokácia dvojrozmerného poľa 2x4

```
int **dvojrozmerne_pole;

dvojrozmerne_pole = new *int[2];

dvojrozmerne_pole[0] = new int[4];

dvojrozmerne_pole[1] = new int[4];
```

Preťaženie funkcií

C++ umožňuje použiť rovnaké meno pre rôzne funkcie
Preťažené funkcie sa musia odlišovať typmi a/alebo počtom argumentov.

1. int max(int array[], int len);
2. long max(long array[], int len);

Linker vyberie správnu funkciu na základe počtu a typov skutočných argumentov vo volaní

Nestačí ak sa funkcie líšia typom návratovej hodnoty!

```
long   average(long array[], int len);
double average(long array[], int len);    //linking error
```

Implicitné hodnoty

```
void vytlac (int n = 1){
    cout << n;
}

int main() {

    vytlac(3);   // vytlaci sa 3
    vytlac();    // vytlaci sa 1

}
```

Od štruktúry k triede

```
struct Item {
    int value;
    struct Item *next;
};


struct Item polozka;
```

```
class Item {
   public:
        int value;
        Item *next;
};

Item polozka;
```

Špecifikátory prístupu

public – k členom triedy je možné pristupovať zvonku

private - členy su viditeľné iba zvnútra

protected (viacej pri dedení tried)

Defaultne je člen private

public, private: príklad

```
class Rectangle {
  public:
    int width, height;
};

int main () {
  Rectangle r;
  r.width = 10;
  r.height = 12;
}
```

public, private: príklad

```
class Rectangle {
  private:
    int width, height;
  public:
    int get_w() {return width;}
    int get_h()  {return height;}
};
int main () {
Rectangle r;
int w = r.get_w();
}
```

Príklad

```
class Point {
    public:
       int x,y;
       void  posun(int a, int b) {  // metóda  triedy Point
              x = x+a;
              y = y+b;
            }
};

int main() {
    Point bod;    //vytvorenie objektu triedy(“typu”) Point
    bod.x = 1;
    bod.y = 2;

    bod.posun(3,4); // čo sa stane ??
}
```

Class definuje namespace

```
class Point {  // definuje tiež menný priestor Point
public:
    int x;
    int y;
    void posun(int a,int b);  //deklarácia funkcie
};

void Point::posun(int a, int b) {
     x= x + a;
     y = y + b;
}
```

Definícia funkcie/metódy posun je mimo triedu

Private vs public

```
class Point {
private:
    int x,  int y;
public:
    void nastav (int a, int b) {
        x = a;   y = b;
    }
};

int main() {
    Point bod;
    bod.x = 1;              //compile error, x je private
    bod.nastav(3,4);        // OK
}
```

this

this = smerník na seba t.j. objekt, ktorého metóda je práve vykonávaná

```
class Point {
  public:
    int x,y;
    void posun ( int x, int  y) {
      this->x = x;
      this->y = y;
  }
};
```

Pomocou this možno pristúpiť k objektu, z ktorého voláme metódu. Rieši prekrytie atribútu lokálnou premennou!!

Konštruktor

```
class Point {
public:
    int x;
    int y;
    Point(int x, int y) {
        this->x = x;
        this->y = y;
    }
};
```

```
int main() {
   Point bodA(3,4);
    Point bodB; //error,
neexistuje default
konštruktor
}
```

Konštruktor sa zavolá keď sa vytvára objekt

Default konštruktor

```
class Point {
public:
    int x;
    int y;
    Point() {
        x = -1;
        y = -1;
    }
    Point(int x, int y) {
        this->x = x;
        this->y = y;
    }
};
```

```
int main() {
    Point bodA(3,4);
     Point bodB; //OK
}
```

Ďalšia možnosť

```
class Point {
public:
    int x;
    int y;

    Point(int mx=-1, int
my=-1) {
        x = mx;
        y = my;
    }
};
```

```
int main() {
    Point bodA(3,4);
     Point bodB; //OK
}
```

Dynamický vznik objektu

```
Point *pbod, *pbod2, *pbody, *pbody2;

pbod = new Point();
pbod2 = new Point(1,2);

pbody = new Point[10];
//volá sa 10 krát defaultný konštr. (bez
parametrov)

pbody2 = new Point(1,2)[5];  // compile error
//nemožno inicializovať dynamické pole
```

Kopírovací konštruktor

Voláme ho keď potrebujeme skopírovať objekt

```
Point(Point& Bod){
    //Point& je referencia(odkaz) priamo na objekt
    x = Bod.x; y = Bod.y;
}

Point Bod1(1,2);
Point Bod2(Bod1); //zavolá sa kopírovací konštruktor

Point Bod3 = Bod1; //kopíruje sa obsah Bod1 do Bod3
zložka  po zložke, ak neexistuje kopírovací konštruktor
```

Pozor! Tu sa vytvára plytká kópia - problém s poľami nekopíruje sa celé pole ale len odkaz na pole.

Deštruktor ~

```
Point::~Point()
{
    cout << "Object is being deleted" << endl;
}

int main() {

Point* p = new Point();

delete p;
}
```

Deštruktor sa zavolá pri delete a keď ide premenná out-of-scope (napr. pri skončení funkcie).

Deštruktor: príklad

```
class Line {
public:
     double *vector;
      int dimension;
      //ďalšie premenné a
metódy
};

Line::Line(int dim=1) {
    dimension = dim;
    vector = new double[dim];
}

Line::~Line(void) {
    delete[] vector;
}
```

```
int main() {

Line *line = new Line(10);

delete line;
}
```

Úloha

Pre triedu Line napíš copy constructor, ktorý prekopíruje všetky elementy poľa vector (a nie len smerník na toto pole).