

Osnovne podatkovne strukture in algoritmi

Uvod

Podatkovne
strukture

Algoritmi

Matevž Jekovec, Tomaž Hočevar

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Programiranje v višji prestavi
2. julij 2018

GimnazijaVič



EVROPSKA UNIJA
EVROPSKI
SOCIALNI SKLAD
NALOŽBA V VAŠO PRIHODNOST



REPUBLIKA SLOVENIJA
MINISTRSTVO ZA IZOBRAŽEVANJE,
ZNANOST IN ŠPORT



- ▶ CodeForces
 - ▶ Ogromna zbirka tekmovalnih nalog iz programiranja
 - ▶ Tekmovanja iz programiranja, običajno brez denarnih nagrad
- ▶ TopCoder
 - ▶ Tekmovanja iz kodiranja, uporabniških vmesnikov in analize podatkov
 - ▶ Denarne nagrade
- ▶ CodeChef
 - ▶ Zbirka tekmovalnih nalog in organizirana tekmovanja (nekaj na mesec)
 - ▶ Manjše denarne nagrade
- ▶ Letno tekmovanje iz programiranja z bogatimi nagradami:
 - ▶ Google Code Jam
 - ▶ Google Hash Code
 - ▶ Facebook Hacker Cup

- ▶ Tekmovanja iz programiranja v Sloveniji:
 - ▶ ACM RTK (dijaki)
 - ▶ ZOTKS Festival inovativnih tehnologij (dijaki)
 - ▶ ACM UPM (študenti)
- ▶ Mednarodna tekmovanja iz programiranja:
 - ▶ CEOI (dijaki) — leta 2017 prvič v Ljubljani!
 - ▶ IOI (dijaki)
 - ▶ ACM CERC (študenti)
 - ▶ ACM ICPC (študenti)
 - ▶ ACM SIGMOD programming contest (študenti, raziskovalci s področja obdelave velikih podatkov)

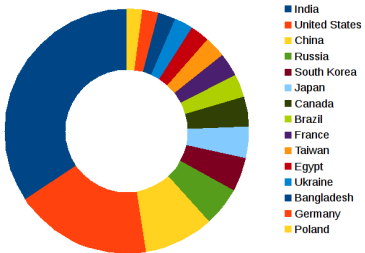
► Google Code jam 2017

Uvod

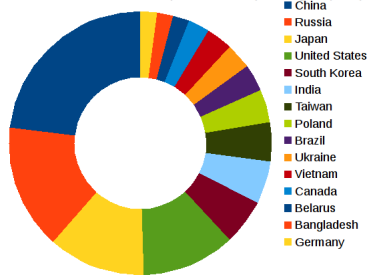
Podatkovne
strukture

Algoritmi

Delež oddanih rešitev po državah (kvalifikacije, top 15)



Delež oddanih rešitev po državah (2. krog, top 15)



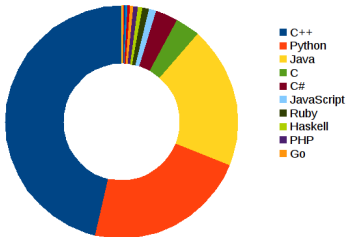
► Google Code jam 2017

Uvod

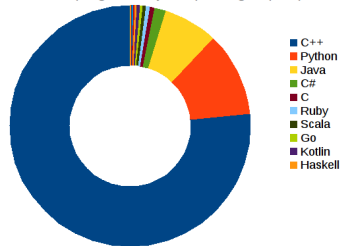
Podatkovne
strukture

Algoritmi

Delež oddanih rešitev glede na
programski jezik (kvalifikacije, top 15)



Delež oddanih rešitev glede na
programski jezik (2. krog, top 15)



Učbeniki v angleščini:

- ▶ Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms (3rd ed.)
- ▶ Sedgewick, Wayne: Algorithms (4th ed.)
- ▶ Berg, Cheong, Kreveld, Overmars: Computational Geometry: Algorithms and Applications (3rd ed.)

V slovenščini:

- ▶ Slovenski prevod prosto dostopne knjige Open Data Structures (<http://sl.opendatastructures.org/>)
- ▶ Zborniki rešenih tekmovalnih nalog ACM RTK od leta 1988 dalje. (<http://rtk.ijs.si>)

C++ in STL dokumentacija:

- ▶ <http://cplusplus.com/reference>
- ▶ <http://en.cppreference.com>

Osnovne podatkovne strukture in algoritmi

Uvod

Podatkovne
strukture

Algoritmi

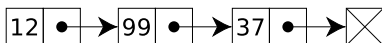
- ▶ tabela/polje (vector)
- ▶ seznam (list)
- ▶ vrsta (queue)
- ▶ sklad (stack)
- ▶ vrsta s prednostjo (priority_queue)
- ▶ množica (set)
- ▶ slovar (map)

- ▶ Hiter dostop do poljubnega elementa.
- ▶ Velikosti so znane vnaprej (statična alokacija).

```
#define N 10000  
int tab[N];
```

vector je 1D tabela dinamične velikosti. Uporaba (STL):

```
vector<int> v = {5,8,2};  
v.push_back(3);  
cout << v[v.size()-1] << endl;
```

- ▶ Počasen dostop do poljubnega elementa.
- ▶ Vstavljanje/Brisanje: i) dinamična alokacija in ii) prevezava sosednjih elementov.

Uporaba (STL):

```
list<int> l = {12,99,37};
l.insert(l.end(),4);
auto it = find(l.begin(),l.end(),99);
l.insert(it, 1);
l.erase(it);
for (int x : l) cout << x << endl;
```

- ▶ FIFO: first-in, first-out
- ▶ Operacije: push, front, back, pop

Uporaba (STL):

```
queue<int> q;  
q.push(9); q.push(1);  
q.pop();  
cout << q.front() << endl;
```

S pomočjo krožne tabele (*circular buffer*):

```
int q[N], h=0, t=0;  
q[t++]=9; q[t++]=1;  
h++;
```

Dosegljivost v grafu (iz točke 0)

```
int n,e;
vector<int> sosedi[100000];
int mark[100000], st=0;
```

```
int main() {
    scanf("%d %d",&n,&e);
    for (int i=0;i<3;i++) {
        scanf("%d %d",&a,&b);
        sosedi[a].push_back(b);
        sosedi[b].push_back(a);
    }
    ...
}
```

```
queue<int> q;
q.push(0);
mark[0]=1;
while (!q.empty()) {
    int x=q.front();
    q.pop();
    st++;
    for (int y : sosedi[x])
        if (!mark[y]) {
            q.push(y);
            mark[y]=1;
        }
}
cout << st << endl;
return 0;
}
```

- ▶ LIFO: Last-in, first-out
- ▶ Operacije: push, top, pop

Uporaba (STL):

```
stack<int> s;  
s.push(9); s.push(1);  
s.pop();  
cout << s.top() << endl;
```

Izvedba s tabelo:

```
int s[N], n=0;  
s[n++]=9; s[n++]=1;  
n--;  
cout << s[n-1] << endl;
```

- ▶ Rekurzivni klici funkcij.
- ▶ Sprehod po drevesu.

```
struct Node {  
    int value;  
    vector<Node*> children;  
}
```

```
void preorderTraversal(Node *root) {  
    stack<Node*> nodes;  
    nodes.push(root);  
    while (!nodes.empty()) {  
        Node *current = nodes.top();  
        cout << current->value;  
        nodes.pop();  
        for (Node *c : current->children) nodes.push(c);  
    }  
}
```

- ▶ Vrsta s prioritetaami elementov — najpomembnejši naprej.

Uporaba (STL):

```
priority_queue<int> p;  
p.push(1); p.push(9); p.push(3);  
p.pop();  
cout << p.top() << endl;
```

Izvedba s seznamom (počasi!), bolje drevesna struktura —
kopica (*heap*).

- ▶ Urejanje — *heap sort*
- ▶ Iskanje najkrajših poti — *Dijkstra*
- ▶ Razvrščanje internetnih paketkov (zvok/slika ima prednost pred spletom)

Naloga:

Imamo n palic z dolžinami a_i , ki bi jih radi razrezali na čim manjše kose. Naredimo lahko k rezov – izberemo neko palico in jo na poljubnem mestu razrežemo na dva kosa. Tako dobimo $n + 1$ palic, ostane pa nam še $k-1$ rezov. Želimo doseči, da bo na koncu najdaljša palica čim krajša! Kako?

$$n = 5, k = 3$$

$$a = [10, 7, 4, 16, 4]$$

- ▶ Množica elementov, elementi se ne podvajajo (izjema multiset), vrstni red vstavljanja ni ohranjen.
- ▶ Operacije: insert, remove, contains, size

Uporaba (STL):

```
set<int> s = {6,8,2,8,4};  
s.insert(6); s.insert(7);  
s.erase(8);  
cout << s.size() << " " << s.count(6);
```

Običajno implementiran z **urejeno podatkovno strukturo**, kar lahko izkoristimo!

- ▶ lower_bound (bisekcija),
- ▶ iteratorji.

- ▶ Preslika ključ \rightarrow vrednost.
- ▶ Podoben množici, vsak ključ se pojavi kvečjem 1-krat (z izjemo `multimap`).

Uporaba (STL):

```
map<int, string> m;  
m[1000]="Ljubljana";  
m[4000]="Kranj";  
cout << m[1000] << endl;
```

- ▶ Polje več različnih vendar anonimnih tipov (naslavljanje z indeksi).
- ▶ Se ustvari enkrat in se ne spreminja (vse vrednosti so konstantne).

Uporaba (STL):

```
tuple<float, float> point1 = {1.0, 2.4};  
cout << "x=" << get<0>(point1) << " y=" <<  
    << get<1>(point1) << endl;
```

Iterator je ovoj (*wrapper*) okoli kazalca na element v podatkovni strukturi.

Večina operacij (brisanje, vrivanje, iskanje) zahteva pozicijo v obliki iteratorja in ne indeksa elementa.

Operacije:

- ▶ Iterator dobimo prek obstoječe podatkovne strukture:

```
vector<int> myvector;  
vector<int>::iterator a = myvector.begin();  
vector<int>::iterator b = myvector.end();
```
- ▶ Naslednji/Prejšnji element: ++a, a++, --a, a--
- ▶ Skok: a+7
- ▶ Kako dobiti element, na katerega kaže iterator?

```
cout << *a;
```

```
#include <iostream>
#include <vector>

int main ()
{
    std::vector<int> myvector;
    for (int i=1; i<=5; i++) myvector.push_back(i);

    std::cout << "myvector contains:";
    for (auto it = myvector.begin();
         it != myvector.end(); ++it)
        std::cout << ' ' << *it;
    std::cout << '\n';

    return 0;
}
```

- ▶ Min/Max: `min`, `max`, `min_element`, `max_element`
- ▶ Masovno spreminjanje podatkov: `copy`, `move`, `replace`, `fill`, `unique`, `reverse`, `rotate`, `partition`
- ▶ Urejanje: `sort`, `stable_sort`, `is_sorted`
- ▶ Iskanje: `binary_search`, `lower_bound`, `upper_bound`
- ▶ In še več: `lexicographical_compare`, `next_permutation`

Izgradnja priponskega polja (urejanje+podan comparator).

```
const char *text="abcabc";
const int N=6;
```

```
bool suffixCompare( int a, int b ) {
    int i;
    for (i=0; (i<N - max(a,b)) &&
          (text[a+i]==(text[b+i]); i++);
        if ( i == N - max(a,b) ) { return a>b; }
        else { return (text[a+i] < text[b+i]); }
    }

    int main() {
        vector<int> sa = {0,1,2,3,4,5};
        std::sort(sa.begin(), sa.end(), suffixCompare);
    }
```