STRUKTURE PODATAKA I ALGORITMI

Predavanje 11

Ishod 4

1

BINARNA STABLA TRAŽENJA

ALGEBRA BERNAYS SVEUČILIŠTE

2

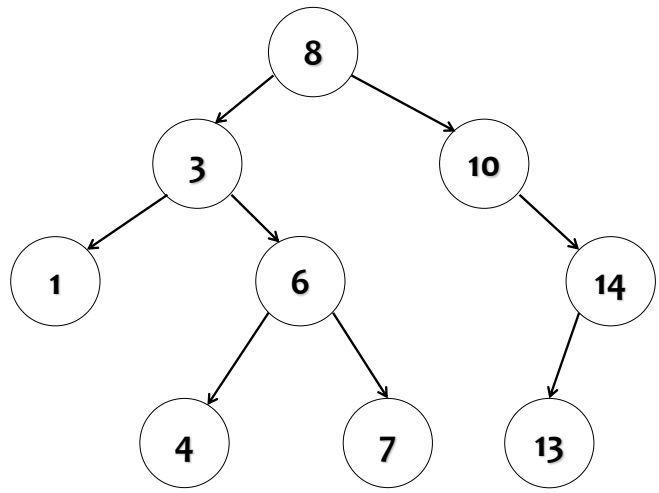
Uvod

- **Binarno stablo traženja** (engl. BST – *binary search tree*) je podvrsta binarnog stabla sa sljedećim svojstvima:
  - Svi podaci u lijevom podstablu su manji od podatka u korijenu podstabla
  - Svi podaci u desnom podstablu su veći ili jednaki podatku u korijenu podstabla
  - Svako podstablo je i samo binarno stablo traženja
- Glavna prednost BST-a je mogućnost efikasnog pretraživanja stabla u potrazi za nekom vrijednošću

ALGEBRA BERNAYS SVEUČILIŠTE

3

Primjer BST-a



ALGEBRA BERNAYS SVEUČILIŠTE

4

Pretraživanje BST-a

- Pretraživanje BST-a obrađuje po jedan čvor na svakoj razini, čime možemo postići logaritamsku složenost (ovisi o izgledu stabla)
  - Recimo da tražimo vrijednost 7
  - Počinjemo od korijena i prema njegovoj vrijednosti (8) znamo da je vrijednost 7 sigurno u lijevom podstablu (jer je 7 < 8)
  - Gledamo korijen lijevog podstabla (3) i znamo da je vrijednost 7 sigurno u desnom podstablu (jer je 7 > 3)
  - Gledamo korijen desnog podstabla (6) i znamo da je vrijednost 7 sigurno u desnom podstablu (jer je 7 > 6)
  - Gledamo korijen desnog podstabla i našli smo 7

ALGEBRA BERNAYS SVEUČILIŠTE

5

Način umetanja u BST

- Način umetanja također može biti vrlo efikasan:
  - Recimo da želimo umetnuti vrijednost 4
  - Počinjemo od korijena i prema njegovoj vrijednosti (8) znamo da vrijednost 4 treba staviti u lijevo podstablo (jer je 4 < 8)
  - Gledamo korijen lijevog podstabla (3) i znamo da vrijednost 4 treba staviti u desno podstablo (jer je 4 > 3)
  - Gledamo korijen desnog podstabla (6) i znamo da vrijednost 4 treba staviti u lijevo podstablo (jer je 4 < 6)
  - Gledamo korijen lijevog podstabla (4) i znamo da vrijednost 4 treba staviti u desno podstablo (jer je 4 = 4)
    - Desno podstablo ne postoji pa kreiramo novi čvor vrijednosti 4 i stavljamo ga kao desno dijete postojećeg čvora vrijednosti 4

ALGEBRA BERNAYS SVEUČILIŠTE

6

AVL STABLA

Uvod

- BST može znatno odstupati od kompletnog stabla
- Uzmimo www.cs.usfca.edu/~galles/visualization/BST.html
  - Dodajmo vrijednosti: 5, 4, 6, 3, 4, 5, 10
    - Savršeno stablo, pretraga je optimalna
  - Resetirajmo stablo i dodajmo iste vrijednosti, ali drugim redoslijedom: 3, 4, 4, 5, 5, 6, 10
    - Dobijemo koso stablo čije performanse su jednake performansama liste

AVL stabla

- **AVL stabla** su podvrsta binarnog stabla traženja sa sljedećim svojstvima:
  - Oba podstabla čvora su ili jednake dubine ili je razlika u dubini jednaka 1
    - To znači da je svaki lisni čvor približno jednako udaljen od korijena
  - Prilikom umetanja (ili brisanja) čvora može se pojaviti potreba **balansiranja** stabla pomoću jedne ili više rotacija kako bi stablo ostalo balansirano
- AVL stablo je naziv dobilo prema svojim tvorcima: G. M. Adelson-Velskii i E. M. Landis
  - To je bilo prvo balansirajuće stablo

ALGEBRA BERNAYS SVEUČILIŠTE

9

Načini pretraživanja i umetanja u AVL stabla

- Način pretraživanja je jednak onome kod BST-a
  - Pošto je stablo balansirano, vrijeme traženja je uvijek $O(\log n)$, što ga čini odličnim za pretraživanje
  - Performanse umetanja/brisanja pate zbog rotacija
- Umetanje čvora se obavlja u dva dijela:
  - Umetanje se napravi jednako kao kod BST-a
  - Za sve pretke umetnutog čvora se izračunava **faktor balansiranosti** koji je jednak: dubina lijevog podstabla minus dubina desnog podstabla
    - Ako je faktor -1, 0 ili +1 stablo je balansirano
    - Ako je faktor balansiranosti jednak -2 ili +2, radi se balansiranje pomoću rotacija

ALGEBRA BERNAYS SVEUČILIŠTE

10

DEMO

- www.cs.usfca.edu/~galles/visualization/AVLtree.html
- Kreirajte AVL stablo s vrijednostima od 1 do 15

ALGEBRA BERNAYS SVEUČILIŠTE

11

CRVENO CRNA STABLA

ALGEBRA BERNAYS SVEUČILIŠTE

12

Uvod

- Crveno-crna stabla (RB, engl. *red-black trees*) su također podvrsta BST-a sa sljedećim svojstvima:
  - Balansirajuća su
  - Svaki čvor sadrži dodatni bit informacije koji sadrži boju (crvena/crna) koji se koristi kod umetanja/brisanja kako bi stablo ostalo otprilike balansirano
    - Korijen je uvijek crn
    - Ako je neki čvor crven, oba djeteta mu moraju biti crna
    - Svaki put od nekog čvora do listova mora sadržavati jednak broj crnih čvorova
    - Iz gornjeg slijedi da na nekom putu nikad ne smiju biti dva uzastopna crvena čvora (ali crnih smije biti koliko god)

ALGEBRA BERNAYS SVEUČILIŠTE

13

Umetanje čvora

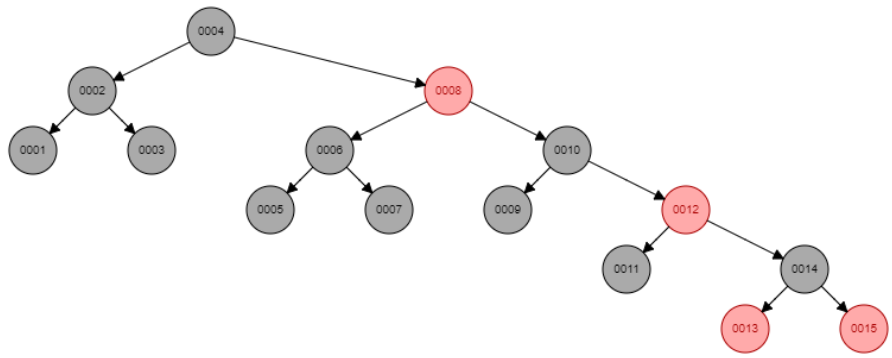
- Umetanje započinje kao kod BST-a, uz:
  - Novi čvor je uvijek crven
  - Kad ga smjestimo na mjesto, postoji šansa da se gubi svojstvo RB stabla
  - Rotacijama i farbanjem se vraća svojstvo RB stabla

ALGEBRA BERNAYS SVEUČILIŠTE

14

Primjer

- www.cs.usfca.edu/~galles/visualization/RedBlack.html
- Kreirajte RB stablo s vrijednostima od 1 do 15



ALGEBRA BERNAYS SVEUČILIŠTE

15

AVL stablo vs RB stablo

- Radi se dva najpoznatija samobalansirajuća stabla
  - Oba stabla koriste rotacije kako bi stablo ostalo (otprilike) balansirano nakon umetanja/izmjene čvora
- Pretraživanje je generalno brže u AVL stablima jer su svi čvorovi ili jednake dubine ili je razlika u jednoj razini
  - RB stablo malo više odstupa, ali i on ima pretraživanje $O(\log n)$
- Oba stabla garantiraju $O(\log n)$ za umetanje/izmjenu
  - AVL dodatne rotacije garantira u $O(\log n)$
  - RB dodatne rotacije garantira u $O(1)$
  - => Umetanje/izmjena je generalno brža u RB stablima
- C++, Java, C# … koriste RB stabla

ALGEBRA BERNAYS SVEUČILIŠTE

16

RJEČNICI

Rječnici

- Rječnik (engl. *dictionary, associative array, map, symbol table*) je kontejner koji sadrži kolekciju parova (ključ, vrijednost) i koji pruža operacije:
  - Dodavanje novog para
  - Uklanjanja para
  - Modifikaciju vrijednosti postojećeg para (ali ne i ključa)
  - Dohvat vrijednosti prema zadanom ključu (naglasak)
- U nekim programskim jezicima (Python) su ugrađeni tipovi
- Dva glavna smjera implementacije rječnika su:
  - *Hash* tablice (ishod 6)
  - Podvrste binarnih stabala traženja

Usporedba smjerova implementacija rječnika

Underlying data structure	Lookup		Insertion		Deletion		Ordered
	average	worst case	average	worst case	average	worst case	
Hash table	O(1)	O(*n*)	O(1)	O(*n*)	O(1)	O(*n*)	No
Self-balancing binary search tree	O(log *n*)	O(log *n*)	O(log *n*)	O(log *n*)	O(log *n*)	O(log *n*)	Yes
unbalanced binary search tree	O(log *n*)	O(*n*)	O(log *n*)	O(*n*)	O(log *n*)	O(*n*)	Yes
Sequential container of key-value pairs (e.g. association list)	O(*n*)	O(*n*)	O(1)	O(1)	O(*n*)	O(*n*)	No

Preuzeto s:
en.wikipedia.org :/

Rječnici pomoću BST-a

- STL sadrži četiri vrste rječnika implementirana pomoću BST-a: `set`, `multiset`, `map` i `multimap`
  - U implementaciji se koriste RB stabla (odluka implementatora)
- `map` čuva vrijednosti spremljene pod jedinstvenim ključevima
- `multimap` dopušta i vrijednosti s duplim ključevima
- `set` čuva samo jedinstvene ključeve (tj. vrijednost = ključ)
- `multiset` dopušta i duple ključeve
- U svim strukturama elementi su sortirani
  - Kad INORDER algoritmom obiđemo RB stablo

Primjeri rječnika

- Popis svih proizvoda koje dućan nudi spremljenih pod jedinstvenom šifrom proizvoda
  - `map<string, Proizvod>`
- Popis svih računa izdanih nekom OIB-u
  - `multimap<string, Racun>`
- Popis svih JMBAG-ova koji su prijavili MI1 iz Fizike
  - `set<string>`
- Popis svih JMBAG-ova koji ikad prijavili ispit iz Fizike
  - `multiset<string>`

SET I MULTISET

Izrada i uništavanje seta/multiseta (1/2)

- Postoji četiri osnovna načina izrade seta/multiseta:
  - `set<int> jedan;`
    - Kreira prazni set
  - `set<int> dva(jedan.begin(), jedan.end());`
    - Kreira set od svih elemenata unutar raspona [begin, end)
  - `set<int> tri(dva);`
    - Kreira set na način da kopira sve elemente iz drugog seta
  - `set<int> cetiri({ 11, 22, 33, 22, 44 });`
    - Kreira set na temelju inicijalizacijske liste (kopiranjem svake od vrijednosti)
    - Ako setu damo dvije jednake vrijednosti, set će drugu jednostavno ignorirati

ALGEBRA BERNAYS SVEUČILIŠTE

23

Izrada i uništavanje seta (2/2)

- Set/multiset se automatski uništava završetkom funkcije
  - Ako čuva objekte, na svakom se poziva destruktor
- `operator=` kopira sadržaj jednog seta/multiseta u drugi
  - Prethodni sadržaj drugog seta/multiseta se uništava (prepisivanjem ili otpuštanjem)
- Vrijednosti stavljene u set/multiset se ne mogu mijenjati

ALGEBRA BERNAYS SVEUČILIŠTE

24

Iteratori seta/multiseta

- Najvažniji iteratori su:
  - `set<T>::iterator` je pokazivač čiji ++ pomiče prema kraju
  - `set<T>::reverse_iterator` je pokazivač čiji ++ pomiče prema početku
- Pošto su setovi sortirani:
  - Prolaskom iteratorom u jednom smjeru idemo od manjih prema većim vrijednostima
  - Prolaskom iteratorom u drugom smjeru idemo od većih prema manjim vrijednostima
- Ako u setu čuvamo objekte, preopterećenjem `operator<` definiramo koji objekt je manji, a koji veći

Struktura `pair<T1,T2>`

- Struktura `pair<T1,T2>` predstavlja par vrijednosti
  - Prva se zove `first` i tipa je `T1`
  - Druga se zove `second` i tipa je `T2`
- Primjer:

```
pair<int, string> p(17, "Miro Miric");
cout << p.first << " " << p.second << endl;
p.first++;
cout << p.first << " " << p.second << endl;
```

Umetanje u set

- U set možemo vrijednosti umetati na tri glavna načina:
  - `s.insert(x)` kopira x i smješta ga na njegovo mjesto u setu
    - Vraća objekt tipa `pair<iterator, bool>`
      - `first` pokazuje ili na friško umetnuti element ili na element koji već postoji u setu
      - `second` sadrži `true` (ako je umetanje uspjelo) ili `false` (ako je element već postojao)
  - `s.insert(begin, end)` kopira elemente [begin, end) i smješta ih na njihovo mjesto u setu
    - Ne vraća ništa
  - `s.insert({ 11, 22, 33 })` kopira brojeve i smješta ih na njihovo mjesto u setu
    - Ne vraća ništa

Brisanje iz seta

- Iz seta možemo vrijednosti brisati na četiri glavna načina:
  - `s.erase(val)` briše element jednak `val`
    - Vraća broj obrisanih elemenata (0 ili 1)
  - `s.erase(position)` briše koji god element se nalazi na zadanoj poziciji
    - Vraća iterator na element koji se nalazi iza obrisanog elementa
  - `s.erase(begin, end)` briše elemente u zadanom rasponu [begin, end)
    - Vraća iterator na element koji se nalazi odmah iza zadnje obrisanog elementa
  - `s.clear()` uklanja i uništava sve elemente seta

Primjer

```
set<int> s({ 55, 11, 55, 33, 22, 44 });
cout << s.size() << endl;

auto ir = s.insert(11);
cout << "Umetnuo: " << ir.second << endl;
ir = s.insert(66);
cout << "Umetnuo: " << ir.second << endl;

s.erase(s.begin());
s.erase(66);

for (auto it = s.begin(); it != s.end(); ++it) {
    cout << *it << endl;
}
```


Razlike multiseta u umetanju i brisanju

- Multiset se ponaša jednako kao i set, uz razlike:
  - `s.insert(x)` kopira x i smješta ga na njegovo mjesto u multisetu
    - Uvijek uspijeva
    - Vraća iterator na umetnuti element
  - `s.erase(val)` vraća broj obrisanih elemenata (0, 1, 2, …)

Ostale važnije metode seta

- `s.find(x)` traži element x u setu i vraća njegovu poziciju
  - Ako nema elementa, vraća `s.end()`
- `s.count(x)` vraća broj pojavljivanja elementa x u setu
  - Može vratiti 0 ili 1 jer su vrijednosti jedinstvene
- `s.size()` vraća broj elemenata u setu
- `s.empty()` vraća je li set prazan

ALGEBRA BERNAYS SVEUČILIŠTE

31

Razlike multiseta

- `ms.count(x)` vraća broj pojavljivanja elementa x u multisetu
  - Može ih biti 0 ili više
- `ms.find(x)` traži prvi element x u multisetu i vraća iterator na njegovu poziciju
  - Ako nema elementa, vraća `s.end()`
- Ako želimo dohvatiti sva pojavljivanja x u multisetu:
  - `ms.equal_range(x)`
    - Vraća `pair<iterator, iterator>`
      - `first` je iterator na prvu vrijednost
      - `second` je iterator na prvu vrijednost iza zadnje

ALGEBRA BERNAYS SVEUČILIŠTE

32

Primjer

```
multiset<int> ms({ 22, 11, 55, 22, 33, 22, 44 });

auto it = ms.find(22);
cout << *it << endl;

auto range = ms.equal_range(22);
for (auto it = range.first; it != range.second; ++it) {
    cout << *it << endl;
}
```


MAPA I MULTIMAPA

Uvod

- Mapu i multimapu možemo shvatiti kao set gdje su ključ i vrijednost međusobno različiti
  - U mapi ključevi moraju biti jedinstveni, u multimapi ne moraju
  - Parovi su sortirani prema ključevima
  - Ključevi su nepromjenjivi, vrijednosti možemo mijenjati

Specifičnosti mape i multimape (1/2)

- Sučelje za korištenje je vrlo slično, uz nekoliko posebnosti:
  - Set/multiset čuva ključeve, dok mapa/multimapa čuva parove
    - `first` čuva ključ, `second` čuva vrijednost
  - Iterator pokazuje na par
  - Parametar metodi `insert` je par
    - Primjerice:

```
map<char, string> m;
m.insert({ 'c', "Canada" });
m.insert(pair<char, string>('a', "America"));
m.insert(pair<char, string>('j', "Japan"));

for (auto it = m.begin(); it != m.end(); ++it) {
   cout << it->first << " " << it->second << endl;
```

Specifičnosti mape i multimape (2/2)

- `s[key]` dohvaća vrijednost pohranjenu pod ključem ili umeće novu praznu vrijednost ako ključ ne postoji
  - Ne postoji na multimapi
- `s.at(key)` radi istu stvar, ali baca iznimku ako ključ ne postoji
  - Ne postoji na multimapi
- Primjerice:

```
map<char, string> m;
m.insert(pair<char, string>('c', "Canada"));
m.insert(pair<char, string>('a', "America"));

cout << m['c'] << endl;
cout << m['a'] << endl;
cout << m['r'] << endl;
cout << m.size() << endl;
cout << m.at('c') << endl;
cout << m.at('f') << endl;
```


Zadatak

- Ubacite brojeve od 1 to 100.000 u vektor i u set. Ispišite koliko traje traženje broja 100.000 u vektoru, a koliko u setu.
  - Traženje u vektoru traje: 172567 mikrosekundi
    - Potrebno odraditi 100.000 operacija
  - Traženje u setu traje: 426 mikrosekundi
    - Potrebno odraditi log(100.000) = 17 operacija

Rješenje (1/2)

```
// Priprema.
int n = 100000;
vector<int> v(n);

for (int i = 1; i <= n; i++) {
    v.push_back(i);
}

set<int> s(v.begin(), v.end());

// Mjerenje.
auto begin = chrono::high_resolution_clock::now();
for (auto it = v.begin(); it != v.end(); ++it) {
    if (*it == n) {
        cout << "Pronasao u vektoru" << endl;
        break;
    }
}
```

ALGEBRA BERNAYS SVEUČILIŠTE

39

Rješenje (2/2)

```
auto end = chrono::high_resolution_clock::now();
cout
    << "Vektor: "
    << chrono::duration_cast<chrono::microseconds>(end -
begin).count() << " us" << endl;

begin = chrono::high_resolution_clock::now();
if (s.find(n) != s.end()) {
    cout << "Pronasao u setu" << endl;
}
end = chrono::high_resolution_clock::now();
cout
    << "Set: "
    << chrono::duration_cast<chrono::microseconds>(end -
begin).count() << " us" << endl;
```

ALGEBRA BERNAYS SVEUČILIŠTE

40