Algoritmi i strukture podataka
vežbe 5

Mirko Stojadinović

29. oktobar 2013

1 Algoritmi za rad sa nizovima i skupovima

1. Dat je niz S od n celih brojeva i ceo broj x. Konstruisati algoritam složenosti $O(n \log n)$ koji utvrđuje da li u S postoje dva elementa kojima je zbir jednak x.

Rešenje: Ako za svaki element proverimo da li sa nekim drugim u zbiru daje x onda dobijamo složenost $O(n^2)$, što je lošije od tražene složenosti. Zato koristimo drukčiji postupak:

1. sortirati niz S i neka je rezulat $S1$
2. za svako y iz niza $S1$ binarnom pretragom tražiti član jednak sa brojem $x - y$

U opisanom algoritmu broj koraka u 1. delu je $O(n \log n)$ a u drugom je $O(n \log n)$. Ukupna složenost je dakle $O(n \log n)$.

2. Dat je niz S od n celih brojeva. Konstruisati algoritam složenosti $O(n)$ koji pronalazi broj koji nije u S.

Rešenje: Taj broj može biti broj za jedan veći od najvećeg broja u nizu S. Maksimum svih brojeva se može naći jednim prolazom kroz niz S tako što se inicijalno postavi da je jednak 1. članu, a potom se prolazi kroz svaki sledeći član niza i ukoliko je tekući član veći od maksimuma do tada, onda se promenljiva max ažurira i dobija vrednost tekućeg člana niza S.

Kako se kroz niz S prolazi tačno jednom i za svaki član obavljaju operacije poređenja sa promenljivom max i eventualna ažuriranja njene vrednosti (što je konstantno vreme), onda je ukupna vremenska složenost $O(n)$.

Algoritam manje složenosti ne može da rešava problem za svaki ulaz, jer ne može da pregleda svih n članova niza S.

3. Neka je zadat niz prirodnih brojeva $x[0], x[1], ..., x[n-1]$. Višestrukost broja y u x je broj pojavljivanja y u x. Odrediti element x višestrukosti veće od $n/2$ ("preovlađujući element") ili ustanoviti da takav element ne postoji. Algoritam treba da je što je moguće manje vremenske i prostorne složenosti. Na primer, u nizu 2 3 2 5 5 5 5 2 5 1 5 4 5 5 je preovlađujući element 5.

Ideja 1: Odrediti medijanu (videti u knjizi poglavlje o rangovskim statistikama), te je nađen kandidat za proveru. Ovo se postiže tako što se rekurzivno radi sledeće: 1. Prvo se radi particionisanje poput onog u kviksort algoritmu. 2. Zatim se pretrazuje samo jedna polovina niza. Može li efikasnije od $O(n^2)$? Podsetiti se dobijanja medijane pomoću dva hipa. Koja je tada složenost?

Ideja 2: Izvrši se sortiranje, te ako postoji preovlađujući broj, on je u sredini i izračuna se njegov broj pojavljivanja u sortiranom nizu. Može li efikasnije od $O(n \log n)$?

Ideja 3: Ako je $x[i]$ različito od $x[j]$ i ako se izbace oba ova elementa iz niza, onda ako postoji y koji je preovlađujući element starog niza, onda je on preovlađujući element i novog niza (obratno ne važi).

U algoritmu se u jednom prolazu koriste promenljive C, M, gde je C jedini kandidat za preovlađujući element u nizu $x[0], x[1], \ldots, x[i-1]$. M je broj pojavljivanja elementa C u nizu $x[0], x[1], \ldots, x[i-1]$, bez onih elemenata C koji su izbačeni.

Ako je $x[i] == C$, onda se M uvećava za 1, a inače smanjuje za 1 Ako M postane 0, onda C dobije novu vrednost $x[i]$ i time i M postane 1.

Algoritam Preovladjuje (x,n):
Ulaz x, n /* niz brojeva x, dimenzije n*/
Izlaz /* preovladjujuci element (ako postoji) ili -1 */

```
C=x[0];
M=1;
for(i=1; i<n; i++){
    if (M==0) {
      C=x[i];
      M=1;
    }
    else {
      if (x[i]==C) M++;
      else M--;
    }
}

if (M==0) return -1;   /*nema preovladjujuceg elementa*/
else brojac=0;
```

```
for (i=0; i < n; i++)
    if (x[i]==C) brojac++;
// moze i bez petlje ako se proverava da li je C jednak x[0] ili x[n-1]

if (brojac > n/2) return C;
else return -1;
```

4. Data su dva niza celih brojeva S1 i S2 i celi broj x. Ustanoviti da li postoje element y1 niza S1 i element y2 niza S2 takvi da im je zbir jednak x. Vremenska složenost algoritma treba da bude $O(n \log n)$, gde je n ukupan broj elemenata u oba niza.

Rešenje:

Neka u prvom nizu ima k, a u drugom n-k elemenata.

1. Sortirajmo oba niza u rastućem redosledu i označimo elemente prvog niza posle sortiranja sa $a_1, a_2, \ldots a_k$. Označimo elemente drugog niza posle sortiranja sa $b_1, b_2, \ldots, b_{n-k}$.

2. Za svako i takvo da 1 <= i <= n - k: tražimo u nizu a element $x - b_i$ binarnom pretragom pomoću $O(\log n)$ (preciznije [log k]+1) upoređivanja među ovim brojevima pronalazi x, ili se utvrđuje da ni jedan od njih nije jednak x. Znači, za proveru za svaki element niza b dovoljno je $O(n \log n)$ upoređivanja.

SLOŽENOST algoritma: složenost početnog sortiranja u koraku 1 je $O(n \log n)$ za proveru svih parova u koraku 2 dovoljno je $O(n \log n)$ upoređivanja, te je ukupna složenost algoritma $O(n \log n)$. Prostorna slozenost algoritma je $O(1)$. Zašto?

5. Kutije su numerisane redom od 1 do n, gde je n paran broj. U i-toj kutiji je smeštena količina od a[i] objekata. Konstruisati algoritam koji će spojiti sadržaje po dve kutije, ali tako da maksimalna količina objekata u po dve kutije bude što je moguće manja (potrebno je da kada se posmatraju novonastale kutije, i izračuna maksimum količine objekata medju njima, taj maksimum bude što je moguće manji).

Rešenje: Prvo sortirati niz a i rezultat je niz b. Zatim formirati parove (b[1],b[n]), (b[2], b[n-1]),..., (b[i], b[n-i+1]), i <= n/2

Dokaz korektnosti postupka: => dokaz da ma koje spajanje sadržaja po dve kutije različito od spajanja iz koraka 2, ima maksimalnu količinu u po dve kutije ne manju od dobijene maksimalne količine u koraku 2. Pretpostavimo suprotno, tj. uočimo spajanje sadržaja po dve kutije koje nema par (b[1], b[n]) već ima neka (proizvoljna) druga dva para (b[1], b[i]), (b[j], b[n]). Ako se ta dva para zamene parovima (b[1], b[n]), (b[i], b[j]), onda se može samo smanjiti maksimum zbirova, jer važi da:

a) b[1]+b[n] <= b[j] + b[n]

b) b[i]+b[j] <= b[j] + b[n]

Slično, ako se i na ostalim parovima primeni analogan postupak, onda zamene parova mogu dovesti samo do smanjivanja maksimuma zbirova. Dakle za bilo koje drugo uparivanje, navedenim postupkom dobijamo ono koje smo mi predložili i čiji maksimum može biti samo manji. Zato su parovi (b[1],b[n]), (b[2], b[n-1]),..., (b[i], b[n-i+1]), i <= n/2 optimalno rešenje zadatka.

Prethodni algoritam spada u grupu pohlepnih algoritama: brzo i intuitivno se dolazi do rešenja problema. Intuicija često može da vara, pa je kod konstrukcije ovakvih algoritama potrebno izvesti i dokaz korektnosti, koji je često vrlo komplikovan. Više reči o ovakvim algoritmima na jednom od narednih časova.

6. Dat je niz od n prirodnih brojeva sa više ponavljanja elemenata, tako da je broj različitih elemenata u nizu $O(\log n)$.

Konstruisati algoritam za sortiranje ovakvih nizova, u kome se izvršava najviše $O(n \log \log n)$ upoređivanja brojeva.

REŠENJE:

Svaki element niza umetnuće se kao jedno polje čvora balansiranog binarnog stabla pretrage. Drugo polje čvora će čuvati broj pojava tog elementa u nizu. Ako je broj različitih elemenata u nizu $O(\log n)$, onda je broj čvorova u stablu $O(\log n)$, te je visina stabla $O(\log \log n)$. Zato je broj upoređivanja najviše $O(n \log \log n)$ pri unosu svih elemenata u stablo.

Zatim se stablo obilazi inorder obilaskom i njegovi elementi se kopiraju u neopadajućem redosledu u izlazni niz dužine n tako što se svaki element kopira onoliko puta kolika je vrednost odgovarajućeg brojačkog polja.

7. Dat je hip sa n elemenata tako da je najveći element u korenu. Dat je i ceo broj x. Konstruisati algoritam koji samo utvrđuje da li je k-ti najveći element hipa veći od x. Nije nužno odrediti k-ti najveći element hipa. Vremenska složenost algoritma mora da (nezavisno od veličine hipa) bude $O(k)$. Dozvoljeno je koristiti memorijski prostor veličine $O(k)$ (u slučaju da se koristi rekurzija, dozvoljeno je koristiti memorijski prostor veličine $O(k)$ za pamćenje rekurzivnih poziva, tj. za stek).

Rešenje: k-ti najveći element hipa veći od x <=> postoji k elemenata hipa koji su veći od x, k >= 0

```
broj=0;  /* promenljiva koja cuva broj elemenata vecih od x; ova promenljiva mora ziveti, tj
                 te je staticka ili globalna ili ...*/
odgovor=false;  /* odgovor da li je k-ti najveci element hipa manji ili jednak od x. Kada se
```

Algoritiam Utvrdi (Hip H sa korenom v, x, k)
ulaz: v, x, k
izlaz: odgovor DA/NE

```
{
if (!v)
  return odgovor; // prazno stablo
if (v->broj > x)
  broj++;
if (broj >= k){
  odgovor = true;
  return odgovor;
}

if (v->levi && v->levi->broj > x)
   Utvrdi (v->levi, x,k);
if (v->desni && v->desni->broj > x)
   Utvrdi (v->desni, x,k);

return odgovor;
}
```

Bez obzira na veličinu hipa, broj rekurzivnih poziva f-je Utvrdi je $O(k)$, jer jer se rekurzivni pozivi vrše samo za ona podstabla za koja je koren veći od x, a ukoliko se nađe k odgovarajućih elemenata, prekida se funkcija. Za pamćenje rekurzivnih poziva potrebno je koristiti stek, tj. memorijski prostor $O(k)$

8. Napisati algoritam koji određuje najveći i najmanji broj u nizu. Dozvoljeno je maksimalno 3n/2 poređenja.

9. Napisati pseudokod algoritma za zadatak 3, ideja 1.

10. Za rastući niz n celih brojeva A, napisati pseudokod koji proverava da li postoji indeks i takav da je A[i] = i. Vremenska složenost treba da je $O(\log n)$ a prostorna $O(1)$.

11. Napisati pseudokod algoritma koji vraća sve duplikate u datom nizu. Vremenska složenost treba da je $O(n \log n)$