INFORMATIKA

Ponešto o sortiranju

Ante Ćustić, Zadar*

Sortiranje nekog niza podataka čest je problem u računarstvu. Često se pojavljuje i unutar nekog većeg problema jer sortiranost podataka može olakšati njihovo daljnje korištenje. Zato se o sortiranju govori i u najosnovnijim kursevima o programiranju. Svi smo već čuli za Quicksort, Mergesort, Bubblesort itd. Iako smo puno slušali o njima vjerojatno bi nam trebalo dosta vremena da ih uspijemo iskodirati. Svi ti načini sortiranja su sortiranja uspoređivanjem. Znači, na skupu koji sortiramo imamo definiran totalni uređaj i nizom međusobnih uspoređivanja dvaju elemenata sortiramo niz. Donja granica složenosti sortiranja uspoređivanjem je $O(n \log n)$. Međutim, rijetko kada se spomene da se osim uspoređivanjem, sortiranje može ostvariti i na druge načine te da tako možemo dobiti složenost i bolju od $O(n \log n)$. U ovom članku ću izložititi neka razmišljanja o tim načinima sortiranja.

Počnimo s jednom vrlo jednostavnom, ali moćnom metodom. Pretpostavimo da želimo sortirati niz od n cijelih brojeva čije vrijednosti mogu biti od 0 do k. Sada ćemo za svaki broj od 0 do k prebrojati koliko se puta pojavljuje u našem nizu. Uzmimo polje (nazovimo ga *brojac*) od $k+1$ elemenata čije elemente inicijaliziramo s 0. Sada za svaki naš broj m iz niza povećamo *brojac*[m] za jedan. Na taj način ćemo postići da je na kraju *brojac*[i] jednak broju brojeva iz našeg niza čija je vrijednost i. Sada da bismo dobili sortiran niz potrebno je još samo proći kroz polje *brojac*, od početka do kraja, i slagati brojeve u početno polje jedan iza drugog tako da broj i stavimo *brojac*[i] puta. Da bude jasnije, slijedi primjer takvog programa u C-u:

```
#define N ...                                   //broj elemenata niza kojeg želimo sortirati
#define K ...                                   //brojevi od 0 do K-1

unsigned long int brojevi[N];                   //niz koji želimo sortirati
unsigned long int brojac[K]={0}, i, j, pozicija=0;
for (i=0; i<N; i++) brojac[brojevi[i]]++;
for (i=0; i<K; i++)
   for (j=1; j<=brojac[i]; j++)
      brojevi[pozicija++]=i;
```

Koje su prednosti, a koje mane ovakvog sortiranja? Velika prednost je brzina. Lako se vidi da je složenost ovog sortiranja $O(n+k)$, gdje su n i k konstante iz teksta iznad. Dakle, imamo linearno sortiranje. I zaista, ovakvo sortiranje za veći broj elemenata niza radi daleko brže od bilo kojeg sortiranja uspoređivanjem. Nezanemariva prednost ovog sortiranja je i jednostavnost i kratkoća koda.

---

* Student je 4. godine studija matematike na PMF-u u Zagrebu.

Mana ovog načina skriva se u pomoćnom polju *brojac*. Naime, to polje mora imati onoliko elemenata koliko različitih vrijednosti mogu poprimiti elementi koje sortiramo. Na primjer, ako želimo sortirati prirodne brojeve koji mogu imati do 10 znamenaka, tada moramo imati polje od 10 milijardi elemenata, što današnja osobna računala ne dopuštaju. Razvojem računala, odnosno povećavanjem njihove memorije, povećava se i mogućnost primjene ovakvog načina sortiranja.

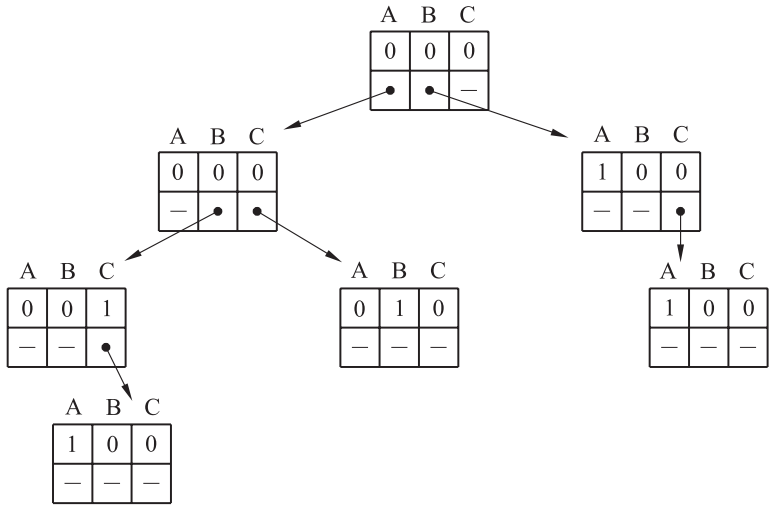
Mogu li se na ovaj način sortirati i drugi tipovi podataka? Nije baš jasno kako bi konstruirali naše pomoćno polje kad bi morali sortirati negativne brojeve, stringove, realne brojeve itd. No, na ovaj način moguće je sortirati elemente svih skupova X za koje postoji monotona bijekcija $f : X \to \{0, 1, ..., k-1\}$, gdje je k prirodan broj takav da je moguće konstruirati pomoćno polje od k elemenata. Tada ćemo za x iz X u *brojac*[f(x)] spremiti broj pojavljivanja od x u nizu kojeg želimo sortirati. Na primjer, ako želimo sortirati cijele bojeve između $-100\,000$ i $100\,000$, tada u *brojac*[0] izračunamo broj pojavljivanja $-100\,000$, u *brojac*[1] broj pojavljivanja $-99\,999$, i tako dalje. Tada je $f(x) = x + 100\,000$, a $k = 200\,001$. I zatim inverznom funkcijom popunimo početni niz koji tako postane sortiran.

Probajmo sortirati stringove. Možemo sortirati stringove npr. duljine 5 sastavljene od malih slova engleske abecede tako da f definiramo ovako: `f(str)=str[4]-'a'+(str[3]-'a')*26+(str[2]-'a')*26^2+(str[1]-'a')*26^3 +(str[0]-'a')*26^4`, pa je $k = 26^5$, što je dovoljno mali broj za konstrukciju pomoćnog polja. Tada *brojac*[0] odgovara stringu "aaaaa", brojac[1] stringu "aaaab" i tako dalje. No, ako želimo sortirati stringove različitih duljina koji mogu biti i dulji od 5, na primjer prezimena, tada to nećemo moći napraviti jer već različitih stringova duljine 6 ima previše za naše pomoćno polje.

Čini se da realne brojeve nečemo moći sortirati. Naime, koliko god mali interval uzeli on će i dalje biti neprebrojivo beskonačan. Međutim, to nije baš tako jer u računarstvu je sve konačno. Na primjer, ako tip float u C-u ima veličinu 32 bit-a, jasno je da on može prikazati najviše 2^{32} različitih brojeva. Jako neprecizno govoreći, realni brojevi se u računalu prikazuju u obliku $\pm s \cdot 10^e$, gdje je s decimalni broj iz intervala $[0.1, 1\rangle$ s preciznošću na d decimala, a e je cijeli broj čija vrijednost može biti od $-ex$ do $+ex$, gdje su d i ex konstante ovisne o tipu realnih brojeva (float, double...). Tada bi našu funkciju f mogli definirati tako da *brojac*[0] broji $-s \cdot 10^e$, gdje su s i e maksimalni. *brojac*-u dalje pridružujemo brojeve tako da se s smanjuje za $1/(10^d)$. Kad s dođe do 0.1, e se smanji za jedan, a s opet postane maksimalan. Kad e dođe do minimuma radimo analogno za pozitivne brojeve samo u obrnutom redoslijedu. Iako neprecizno opisano, ovim je dana ideja za pravo rješenje ovog problema. Ovakav f se moze izračunati "čačkanjem" po bitovima broja.

Znači, možemo sortirati i neke intervale realnih brojeva. Sada vidimo da se ovaj način može koristiti puno češće nego što se na početku činilo. Jedini problem nam je veličina pomoćnog polja i činjenica da elementi koje sortiramo moraju moći poprimiti samo ograničen broj različitih vrijednosti. Taj problem ćemo probati riješiti u daljnjim razmišljanjima.

Sada ćemo naše elemente sortiranja promatrati malo drugačije. Rastavljat ćemo ih na manje dijelove i onda sortirati po dijelovima stvarajući uređeno stablo. Pogledajmo to na primjeru stringova. Stvarat ćemo stablo čiji čvorovi će predstavljati jedan znak u stringu. Ubacivat ćemo stringove u naše stablo tako da će za svaki string postojati jedan posebno označen čvor za kojeg će se, putujući iz korijena do tog čvora i čitajući oznake čvorova kroz koje se prolazi, dobiti baš taj pripadni string. Pojasnimo to na jednom jednostavnom primjeru prikazanom na sljedećoj slici. U ovom primjeru, radi jednostavnoti slike, sortirat ćemo stringove koji mogu sadržavati samo znakove A, B i C. Ovo stablo nastalo je od stringova ABCD, BCA, ACB, ABC, BA.



Vidimo da se svaki čvor sastoji od tri stupca – svaki stupac za jedan mogući znak. Prvi red u stupcu označava broj stringova koje smo uronili u stablo koji završavaju na tom čvoru i točno tim znakom. Sada je putem od korijena do tog čvora jedinstveno određen taj string. Drugi red u stupcu sadrži pointer na podstablo u slučaju da postoji string koji se dobije dodavanjem još znakova na string koji je određen mjestom na kojem se nalazimo.

Uočimo da nam za ubacivanje novog stringa duljine l u stablo treba l koraka, od kojih će dio biti putovanje po već postojećim čvorovima, a kasnije ćemo eventualno dodavati nove čvorove. Po tome se vidi da će složenost stvaranja stabla od n stringova biti $O(n * k)$, gdje je k duljina najvećeg stringa. Uočimo da naše stablo može primati stringove proizvoljne duljine, na primjer dodavanje stringa duljine 100 ne bi predstavljao nikakav problem za naše stablo. Dakle, nemamo problem koji smo imali kod prošlog načina sortiranja. Sada, da bismo od ovakvog stabla dobili sortirani niz stringova sve što moramo napraviti je preorder obilazak stabla i to tako da na mjestima u čvorovima gdje nisu nule spremimo pripadni string odgovarajući broj puta. Preorder obilazak je obilazak gdje prvo posjetimo korijen, a zatim prvo dijete i po redu svu ostalu djecu. Preorder obilaskom prođemo svaki čvor točno jednom i ako pamtimo pripadni string čvora na kojem se nalazimo nećemo morati za svaki čvor prijeći put od korijena do njega da bismo znali pripadni string. Zato je složenost ovakvog preorder obilaska $O(m)$, gdje je m broj čvorova u stablu, što je sigurno manje od $O(n * k)$, pa je ukupna složenost ovog načina sortiranja $O(n * k)$. I zaista, na ovaj način možemo, za relativno velik broj elemenata, sortirati brže od Quicksorta, ali kod sortiranja još više elemenata gubi se efikasnost zbog velike alokacije memorije relativno velikih čvorova. Ipak ovaj način daje nam dobru teorijsku podlogu za daljnja razmišljanja.

Probajmo sada povezati vrline, a izbaciti mane dva opisana načina sortiranja. Opet ćemo naše elemente, kao u prošlom načinu, dijeliti na manje dijelove, i tada naš cijeli niz sortirati s obzirom na te dijelove nekoliko puta. Ta sortiranja radit ćemo metodom prvog opisanog sortiranja. Da bude jasnije objasnit ćemo to na jednom primjeru. Sortirat ćemo prirodne brojeve 149, 252, 478, 73, 672, 106. Naše brojeve ćemo sortirati rastavljanjem na znamenke. Prvo sortiramo s obzirom na znamenku najmanje važnosti, tj. znamenku jedinica, zatim desetica i na kraju stotica. Na kraju ćemo dobiti sortirani niz.

Ovako:

25<u>2</u>	1<u>0</u>6	_73
67<u>2</u>	1<u>4</u>3	<u>1</u>06
14<u>3</u>	2<u>5</u>2	<u>1</u>43
7<u>3</u>	6<u>7</u>2	<u>2</u>56
10<u>6</u>	<u>7</u>3	<u>4</u>78
47<u>8</u>	4<u>7</u>8	<u>6</u>72

Primijetimo da za svako međusortiranje oni elementi koji imaju jednaku pripadnu znamenku po kojoj se sortira, moraju ostati u jednakom međusobnom poretku kao i prije tog sortiranja. To svojstvo je ključno! (Zašto?) Svako naše međusortiranje sortira puno brojeva po jednoj znamenci koja može poprimiti samo 10 različitih mogućnosti. Jasno je da nam za to savršeno odgovara metoda iz prvog opisanog sortiranja. Međusortiranje se ponavlja onoliko puta koliko najveći broj ima znamenaka. Naše elemente, u ovom slučaju prirodne brojeve, možemo dijeliti i na druge načine, npr. na bitove ili po dvije, tri ili više znamenaka.

Pogledajmo detaljnije kako bi izgledao jedan program za sortiranje prirodnih brojeva ovom metodom. Najprije prođemo kroz cijeli niz i u pomoćno polje *brojac* izračunamo koliko se puta koja znamenka pojavljuje. Tada ne možemo sortirati naš niz samo prolaskom po pomoćnom polju jer ne znamo kojem broju je pripadala ta znamenka. Za to ćemo imati još jedno pomoćno polje (nazovimo ga *pomak*) veličine iste kao i *brojac* koji u *pomak*[i] računa mjesto u nizu gdje treba biti spremljen sljedeći broj kojem je pripadna znamenka i. *pomak* popunimo sljedećom rekurzijom:

```
pomak[0]=0
pomak[i]=pomak[i-1]+brojac[i-1]
```

Sada prođemo po početnom nizu i svaki broj kojem je pripadna znamenka i, spremamo na mjesto *pomak*[i], i nakon toga povećamo *pomak*[i] za jedan. Spremati moramo u drugi niz. Kasnije će taj drugi niz biti sortiran, a onaj prvi će služiti za spremanje. Uloge ta dva niza će se ciklički mijenjati. Sada ponavljamo međusortiranja na tako nastalom nizu, ali na sljedećoj znamenci, sve dok ne bude i posljednja znamenka najvećeg broja obrađena. Evo jednog jednostavnog primjera u C-u koji koristi tu metodu. U ovom primjeru nećemo rastavljati na po jednu znamenku nego po tri.

```
#define N 10000000                  //broj elemenata niza kojeg sortiramo
#define VZ 1000                     //veličina 'znamenke', 1000 znaci da sortiramo po tri znamenke
unsigned long int polje[2][N];      //polje koje sortiramo i polje u koje spremamo, mjenjaju uloge
long int dj=1,i,broj,predjeni;      //pomoćne varijable
int prvi=0,drugi=1;                 //varijable koje određuju koje polje ima koju ulogu
long int brojac[VZ],pomak[VZ];

while (1){
  predjeni=0;                       //broji za koliko elemenata niza smo pregledali sve znamenke
   for (i=0; i<VZ; i++) brojac[i]=0;
  for (i=0; i<N; i++){
     broj=polje[prvi][i]/dj;
     brojac[broj%VZ]++;             //punimo polje 'brojac' (metoda prvog načina)
     if (broj==0) predjeni++;
  }
  if (predjeni==N) break;           //svim elementima presli sve znamenke, program se zaustavlja
  pomak[0]=0;
```

```
  for (i=1; i<VZ; i++) pomak[i]=pomak[i-1]+brojac[i-1]; //računanje pomaka
  for(i=0;i<N;i++)polje[drugi][pomak[(polje[prvi][i]/dj)%VZ]++]=polje[prvi][i];
                        //spremanje
  prvi=(prvi+1)%2;      //varijable se zamjenjuju, tj. polja mijenjaju ulogu
  drugi=(drugi+1)%2;
  dj*=VZ;               //dj se množi sa VZ, tj. gledamo sljedeće (tri) znamenke
}
```

Ovaj kratak i jednostavan program, iako se može još optimizirati, vrlo je efikasan. Očito je da se ovako mogu sortirati i stringovi na analogan način. Inače, iako to nije očito niti jako poznato, na ovaj način mogu se sortirati i realni brojevi, iako malo kompliciranije. Ova metoda se može i modificirati ako je potrebno, tako da se prvo sortira od najvažnijeg dijela. Lako se vidi da je složenost ovog načina $O(n*k)$ gdje je k broj dijelova na koliko se mora rastaviti najveći element niza. Više o ovakvim načinima sortiranja potražite na internetu pod nazivom radix sort. Metoda je vrlo efikasna, probajte i usporedite je s nekim sortiranjem uspoređivanjem i vidjet ćete da radi bolje.