Sortiranje podataka

Goran Delač

U *PlayMath*-u br. 8 upoznali smo se s time kako ubrzati potenciranje nekog broja. Sada ćemo obraditi jedan vrlo čest praktični problem s kojim se susrećemo u životu, a to je **sortiranje podataka**.

Imamo nekakav niz od $n \in \mathbb{N}$ podataka u nekom vektoru A:

$A[0]$	$A[1]$	...	$A[n-1]$
a_0	a_1	...	a_{n-1}

Vrijednosti $a_0, \ldots, a_{n-1}$ vrlo često možemo uspoređivati na neki način (abecedno, brojčano i sl.). Naprimjer, ako su $a_0, \ldots, a_{n-1}$ imena, želimo ih poredati (sortirati) po abecedi ili ako su u pitanju godine, želimo ih poredati po starosti,...

S tom svrhom razvijeni su brojni algoritmi koji počivaju na različitim idejama. Ovdje ćemo navesti dvije.

Selection sort

Jedan od jednostavnijih algoritama za sortiranje je *selection sort*.

za $i = 0$ *do* $n - 1$ **radi**
  $min = i$;
  za $j = i + 1$ *do* $n - 1$ **radi**
    ako $A[j] < A[min]$
      $min = j$;
  $A[i] = pom$;
  $A[i] = A[min]$;
  $A[j] = pom$;

Ovaj pseudokod preveden na programski jezik C izgledao bi ovako:

```
for (i = 0; i < n; i++) {
    min = i;
        for (j = i+1; j < n; j++) {
    if (A[j] < A[min]) min = j;
        }
        A[i]= pom;
        A[i]= A[min];
        A[j]=pom;
    }
```

Što ovaj algoritam zapravo radi? Za svaki i u skupu $\{a_i, \ldots, a_{n-1}\}$ traži najmanji element i stavlja ga na i-to mjesto, dok element koji se nalazio na i-tom mjestu stavlja na mjesto gdje je bio najmanji element.

Npr. neka je $n = 3$ i $A = (1.2, 3.4, 0)$.
Za $i = 0$, $\min\{A[0], A[1], A[2]\} = A[2] = 0$. Stoga zamjenjujemo brojeve na 0-tom i 2. mjestu u A te je sada $A = (0, 3.4, 1.2)$. (*Uočimo kako je najmanji element došao na 0. mjesto.*)
Za $i = 1$ dobivamo $\min\{A[1], A[2]\} = A[2] = 1.2$. Stoga zamjenjujemo 1. i 2. član iz A te je sada $A = (0, 1.2, 3.4)$.
Za $i = 2$ postupak je završen.

Algoritam će se izvršiti u najgorem i u najboljem slučaju jednaki broj puta. Možemo jednostavno odrediti složenost. Imamo dvije petlje, jednu po varijabli i, a drugu po j. Vanjska petlja će se izvršiti n puta, a unutarnja petlja će se izvršavati ovisno o i. Za $i = 0$ unutarnja petlja će se izvršiti $n - 1$ puta, za $i = 2$ unutarnja petlja $n - 2$ puta, $\ldots$ za $i = n - 2$ izvršit će se 0 puta. Dakle ukupno

$$(n-1) + (n-2) + \ldots + 1 + 0 = \frac{n(n-1)}{2} \text{ puta.}$$

Lako dolazimo do zaključka da je složenost *selection sort*-a kvadratična, a to pišemo

$$c(n) = \mathcal{O}(n^2).$$

Pokažimo sada jedan učinkovitiji algoritam sortiranja.

Merge sort

Listu od n brojeva (ili znakova) najprije podijelimo u dvije podliste s (pod)jednakim brojem elemenata. Ako je broj elemenata $n = 2r + 1$ neparan, onda u jednu listu stavljamo r, a u drugu $r + 1$ elemenata. Isti postupak ponavljamo sve dok liste u potpunosti ne rastavimo.

Ova metoda poznata je još pod nazivom *”podijeli pa vladaj”*. Princip je da se početni problem raščlani na dva lakša problema.

Dobivene liste treba spojiti na odgovarajući način tako da u konačnici dobijemo jednu listu koja sadržava sortirane podatke. Neka su zadane dvije liste : $(a_1, \ldots, a_i)$ i $(b_1, \ldots, b_j)$. Spajanje vršimo tako da uspoređujemo prvi element u lijevoj listi s prvim u desnoj. Manji od tih dvaju elemenata stavljamo u novu listu i brišemo ga iz pripadne liste. U svakom koraku uspoređujemo samo prva dva elementa u listama. Za gornje dvije liste moguća je najviše $i + j - 1$ usporedba.

Pokažimo sada na primjeru kako se izvršava *merge sort* na listi brojeva $(1.2, 3.4, 0)$:

$$\begin{array}{c}
(1.2, 3.4, 0) \\
\swarrow \quad \searrow \\
(1.2, 3.4) \qquad (0) \\
\swarrow \quad \searrow \qquad\quad \downarrow \\
(1.2) \quad (3.4) \quad (0) \\
\searrow \quad \swarrow \qquad\quad \downarrow \\
(1.2, 3.4) \qquad (0) \\
\searrow \quad \swarrow \\
(0, 1.2, 3.4)
\end{array}$$

Pokazuje se da je složenost *merge sort*-a:

$$c(n) = \mathcal{O}(n \log_2(n))$$

To je bitno poboljšanje u odnosu na *selection sort*. Da bismo sortirali 1000 podataka, *selection sort* bi trebao ponavljanja reda 10^6, dok bi *merge sort* to isto napravio za najviše 9965 ponavljanja.

Zaključak

U računarstvu često nije dovoljno da algoritam samo obavi zadani posao, cilj je da to napravi što *efikasnije* u vremenskom i memorijskom smislu i sa što manje računskih operacija.

Literatura

[1] Žubrinić D.: *Diskretna matematika*, Element, Zagreb, 2002.

[2] Knuth D. E.: *The Art of Computer Programming*, Volume 1, Addison-Wesley,1973.

[3] *Dictionary of Algorithms and Data Structures*, `http://www.nist.gov/dads/`