urejanje zaporedij

v. batagelj

UDK 681.3.06/07

FNT, VTO matematika in mehanika
Ljubljana

Namen sestavka je seznaniti bralca z dvema učinkovitima primerjalnima postopkoma za urejanje zaporedij znotraj pomnilnika. To sta "heap sort" in "quick sort". Poleg splošnega uvoda v postopke urejanja zaporedij, so podani tudi izpisi ustreznih podprogramov, napisanih v structranu, in časovne značilnosti le-teh pri urejanju slučajno zgeneriranih zaporedij na računalniku Cyber.

INTERNAL SORTING METHODS - In the paper two efficient internal sorting methods, "heap sort" and "quick sort", are presented. Their time characteristics for sorting random generated data on Cyber are also given.

Imejmo zaporedje n podatkov iz linearno urejene množice $(P , \leqslant)$

$$p_1 , p_2 , p_3 , \cdots , p_n$$

Urediti dano zaporedje, pomeni poiskati tako permutacijo indeksov π , da bo zaporedje

$$p_{\pi(1)}, p_{\pi(2)}, p_{\pi(3)}, \cdots , p_{\pi(n)}$$

urejeno v nepadajočem vrstnem redu; kar pomeni, da za vsak par indeksov i in j velja:

$$i < j \Longrightarrow p_{\pi(i)} < p_{\pi(j)}$$

Pogosto pri urejanju preuredimo kar samo zaporedje (permutacija je določena implicitno).

Postopke urejanja zaporedij delimo na
- zunanje : zaporedje je datoteka na zunanjem pomnilniku
- notranje : zaporedje se v celoti nahaja v hitrem pomnilniku.

Postopke urejanja razvrščamo naprej na
- postopke, ki upoštevajo notranjo strukturo podatkov
- primerjalne postopke, ki upoštevajo samo linearno urejenost množice P

V tem sestavku se bomo omejili na primerjalne postopke urejanja znotraj pomnilnika. Ti postopki se v glavne uporabljajo kot sestavni deli drugih postopkov; morda najpogosteje pred izpisom podatkov.

V programih najpogosteje srečujemo takoimenovani "bubble sort" postopek

```
k ← n
for ( i = 1, n-1 ) do
    k ← k - 1
    for ( j = 1, k ) do
        if ( p(j) > p(j+1) ) then
        {premenjamo podatka}
            t ← p(j)
            p(j) ← p(j+1)
            p(j+1) ← t
        endif
    endfor
endfor
```

ali njegove inačice oziroma izboljšave (glej primer v dodatku).

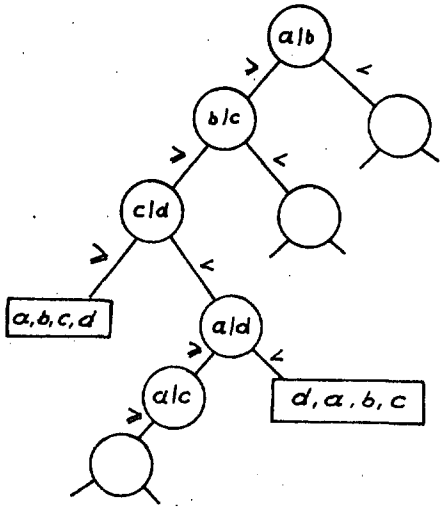
Kompleksnost postopkov urejanja je odvisna od večih parametrov. Za primerjalne postopke je najznačilnejši parameter število primerjanj.

Poglejmo kolikšno je število primerjanj za gornjo verzijo "bubble sort" postopka. Iz programa vidimo, da se primerjanje izvrši

$$\sum_{i=1}^{n-1} \sum_{j=1}^{n-i} 1$$

krat. Torej $n(n-1)/2$ krat. Izboljšani program iz dodatka potrebuje v splošnem manj primerjanj, vendar je v povprečju še vedno reda n^2. V najslabšem primeru, ko je zaporedje urejeno v nenaraščajočem vrstnem redu, pa je enakovreden gornjemu postopku.

Postavi se vprašanje: koliko najmanj primerjanj je potrebnih pri urejanju zaporedja n podatkov z "najboljšim" primerjalnim postopkom? Do odgovora na zastavljeno vprašanje pridemo z naslednjim razmislekom. Vse možne izvršitve poljubnega primerjalnega postopka, lahko prikažemo z binarnim drevesom, katerega notranje točke so primerjanja, končne točke (listi) pa permutacije.
Primer dela mogočega drevesa izvršitve pri urejanju zaporedja a,b,c,d je prikazan na sliki.



Torej je kompleksnost postopka povezana z dolžinami poti po tem drevesu od korena do lista. Zato se vprašamo: koliko je najmanj največja dolžina poti po drevesu? Število listov drevesa (permutacij) je vsaj $n!$. Po poteh dolžine k lahko pridemo v največ 2^k listov. Torej mora veljati

$$2^k \geq n!$$

oziroma po Stirlingovem približku

$$k \geq n \log_2 n - n/\ln 2 + \log_2 n/2 + c$$

Torej je

$$k_{opt} \approx n \log_2 n$$

Od tu vidimo, da je kompleksnost "najboljših" primerjalnih postopkov urejanja zaporedij reda $n \log n$.
Poleg časovne kompleksnosti (število operacij) je pomembna tudi prostorska kompleksnost (zahteve po pomnilniku). Najboljši postopki urejanja bodo za urejanje potrebovali le prostor, ki ga zaseda zaporedje podatkov (in program), drugi pa zahtevajo še dodaten prostor za shranjevanje pomožnih rezultatov.

V dodatku sta opisana dva izmed "najboljših" postopkov.
Prvi postopek, ki ga v literaturi srečamo pod imenom "heap sort", ima časovno kompleksnost (največjo dolžino poti po drevesu) reda $n \log n$ in ne potrebuje dodatnega prostora.

Drugi postopek, imenovan "quick sort", pa je primer postopka, ki potrebuje še $c \log n$ dodatnega prostora in ima povprečno časovno kompleksnost (povprečna dolžina poti po drevesu) reda $n \log n$.
Poskusi na slučajnih podatkih (za dane programe glej tabelo v dodatku), kažejo, da je "quick sort" (v povprečju) boljši od "heap sort-a"; obstajajo pa tudi zaporedja, za katera je reda n^2. Zato previdni programerji dajejo prednost zanesljivemu "heap sort-u".

Več o postopkih urejanja zaporedij lahko bralec prebere v knjigah:

D.E. Knuth: The Art of Computer Programming; vol.3/ sorting and searching; Addison-Wesley, 1973

in

A.V. Aho, J.E. Hopcroft, J.D. Ullman: The Design and Analysis of Computer Algorithms; Addison-Wesley, 1974

T A B E L A : časovne značilnosti za slučajne podatke za Cyber (čas je merjen v sekundah)

dolžina zaporedja	BUBBLE meritve	F_b	HEAP meritve	F_h	QUICK meritve	F_q
0						
10	.001	.001	.001	.001	.002	.001
50	.019	.018	.010	.010	.008	.010
100	.074	.072	.023	.024	.018	.023
200	.27	.29	.055	.055	.045	.052
500	1.8	1.8	.16	.16	.14	.15
1000	7.2	7.2	.36	.36	.34	.34
2000	28.	29.	.79	.79	.78	.74
3000	64.	65.	1.25	1.25	1.2	1.2
5000		180.	2.2	2.2	1.7	2.1
10000		720.	4.8	4.8	4.5	4.5
20000		2900.	10.	10.	9.8	9.7
30000		6500.	16.	16.	15.	15.
100000		72000.		60.		56.

$F_b = 7.2\ n^2\ 10^{-6}$ $F_h = 3.6\ n \log_2 n\ 10^{-5}$ $F_q = 3.4\ n \log_2 n\ 10^{-5}$

B U B B L E - SORT

```
      SUBROUTINE SORT(TAB,LENGTH)
      INTEGER TAB    ( 1       )
      INTEGER BOUND , CHANGE
*
*  TABELO  TAB(1..LENGTH)  UREJAMO TAKO, DA NA KONEC NEUREJENEGA DELA
*  TABELE SPRAVIMO NJEGOV NAJVECJI ELEMENT. TO PONAVLJAMO VSE DOKLER NI
*  TABELA UREJENA.
*
      IND = LENGTH
      REPEAT
*
*     NEUREJENI DEL TABELE PREGLEDUJEMO OD ZACETKA PROTI KONCU. CE TE-
*     KOCA ZAPOREDNA ELEMENTA NISTA V PRAVEM VRSTNEM REDU, JU PREMENJAMO
*
         BOUND = IND - 1
         IND   = 0
         FOR (I = 1,BOUND) DO
            NEXT = I + 1
            IF (TAB(I).GT.TAB(NEXT)) THEN
*
*              TEKOCA DVOJICA NI V PRAVEM VRSTNEM REDU, ZATO ELEMENTA
*              PREMENJAMO
*
                 CHANGE    = TAB(I)
                 TAB(I)    = TAB(NEXT)
                 TAB(NEXT) = CHANGE
*
*              ZAPOMNIMO SI INDEKS PREMENE. PO KONCANEM PREGLEDOVANJU
*              DOLOCA INDEKS ZADNJE PREMENE KONEC NEUREJENEGA DELA TABELE.
*
                 IND = I
              ENDIF
           ENDFOR
      UNTIL (IND.LE.1) ENDREPEAT
*
*  TABELA JE UREJENA
*
      RETURN
*
      END
```

H E A P - SORT

```
      SUBROUTINE SORT(TAB,LENGTH)
      INTEGER CHANGE, FATHER, ROOT , SON
      INTEGER TAB    ( 1      )
      DATA    ROOT   / 1      /
*
*  NAD TABELO TAB(1..LENGTH) RAZPNEMO BINARNO DREVO, TAKO DA VELJA:
*  ROOT = 1  IN ZA POLJUBNO TOCKO  NODE  :  FATHER = ENTIER(NODE/2) ,
*  LEFTSON = 2*NODE  TER  RIGHTSON = LEFTSON + 1 . DREVO, V KATEREM JE
*  VRH VSAKEGA PODDREVESA VECJI OD VSEH SVOJIH NASLEDNIKOV IMENUJEMO
*  KOPICA (HEAP). TABELO  TAB  UREJAMO TAKO, DA NAJPREJ SESTAVIMO IZ
*  NJE KOPICO. NA VRHU KOPICE JE NAJVECJI ELEMENT TABELE. ODSTRANIMO GA.
*  PREOSTALE ELEMENTE ZOPET PREUREDIMO V KUPICO. TO PONAVLJAMO VSEDOKLER
*  NI TABELA UREJENA.
*
*  SESTAVIMO KOPICO
*
      FOR (NODE = 2*LENGTH) DO
*
*     PREDPOSTAVIMO, DA JE  TAB(1..NODE-1)  KOPICA. PREDELAJMO V KOPICO
*     TAB(1..NODE) , TAKO DA POMAKNEMO ELEMENT  TAB(NODE)  NAVZGOR NA
*     NJEGOVO MESTO V HIERARHIJI.
*
        SON = NODE
        REPEAT
          FATHER = SON/2
          IF (TAB(FATHER).GE.TAB(SON)) EXIT
*
*       POMAKNEMO SE NAVZGOR
*
          CHANGE      = TAB(FATHER)
          TAB(FATHER) = TAB(SON)
          TAB(SON)    = CHANGE
          SON         = FATHER
        UNTIL (SON.LE.ROOT) ENDREPEAT
      ENDFOR
*
*  UREDIMO TABELO
*
      NODE = LENGTH
      LOOP
*
*     PREMENJAMO  TAB(NODE)  IN TAB(ROOT) . DEL TABELE  TAB(NODE..
*     LENGTH)  JE TAKO UREJEN.
*
        CHANGE      = TAB(NODE)
        TAB(NODE)   = TAB(ROOT)
        TAB(ROOT)   = CHANGE
*
*     ELEMENT  TAB(ROOT)  POMAKNEMO PO DELU TABELE  TAB(1..NODE-1)
*     PRIPADAJOCEM DREVESU NAVZDOL, TAKO DA ZASEDE V HIERARHIJI
*     PRIPADAJOCE MU MESTO. DEL TABELE  TAB(1..NODE-1)  JE POSTAL
*     KOPICA.
*
        NODE = NODE - 1
        IF (NODE.LE.ROOT) RETURN
        FATHER = ROOT
        LOOP
          SON = 2*FATHER
          IF (SON.GT.NODE) EXIT
*
*       DOLOCIMO VECJEGA OD BRATOV
*
          IF ((TAB(SON).LT.TAB(SON+1)).AND.(SON.LT.NODE)) SON = SON+1
          IF (TAB(FATHER).GE.TAB(SON)) EXIT
*
*       POMAKNEMO SE NAVZDOL
*
          CHANGE      = TAB(FATHER)
          TAB(FATHER) = TAB(SON)
          TAB(SON)    = CHANGE
          FATHER      = SON
        ENDLOOP
      ENDLOOP
*
      END
```

```
      Q U I C K - SORT


      SUBROUTINE SORT(TAB,LENGTH)
      INTEGER CHANGE, DELTA , DEPTH , RIGHT
      INTEGER TAB    ( 1     ), STACK ( 3     )
*
*  TABELO  TAB(1..LENGTH)  UREJAMO TAKO, DA JO GLEDE NA IZBRANI DELILNI
*  ELEMENT RAZBIJEMO NA DVA DELA, PRI ČEMER DELILNI ELEMENT ZASEDE SVOJE
*  MESTO V UREJENI TABELI IN JE VSAK ELEMENT PRVEGA DELA MANJŠI OD VSA-
*  KEGA ELEMENTA IZ DRUGEGA DELA. ZA VSAK DEL POSEBEJ, ČE NI UREJEN, PO-
*  STOPEK PONOVIMO. TABELA JE UREJENA, KO SO UREJENI VSI DOBLJENI DELI.
*
      DEPTH =                 ... GLOBINA LASTNEGA SKLADA
      LEFT  = 1               ... SPODNJI INDEKS DELA TABELE, KI GA UREJAMO
      RIGHT = LENGTH          ... ZGORNJI INDEKS DELA TABELE, KI GA UREJAMO
      LOOP
*
*     ČE TEKOČI DEL TABELE SE NI UREJEN, GA RAZBIJEMO IN NADALJUJEMO Z
*     UREJENJEM KRAJŠEGA DELA. DALJŠI DEL SI ZAPOMNIMO; DRUGAČE NADALJU-
*     JEMO Z UREJANJEM ENEGA IZMED PREOSTALIH NEUREJENIH DELOV; ČE NI
*     NOBENEGA VEČ, JE TABELA UREJENA.
*
         DELTA = RIGHT - LEFT
         IF (DELTA.GT.1) THEN
*
*        TEKOČI DEL TABELE VSEBUJE VEČ KOT 2 ELEMENTA. DELITEV NADA-
*        LJUJEMO. SIRŠI DEL TABELE SI ZAPOMNIMO (MEJI SPRAVIMO V SKLAD)
*
            CALL SPLIT(TAB,LEFT,RIGHT,NLEFT,NRIGHT)
            IF (NLEFT.LT.NRIGHT) THEN
               DEPTH = DEPTH + 2
               STACK(DEPTH-1) = NLEFT
               STACK(DEPTH)   = NRIGHT
            ENDIF
         ELSE
*
*        TEKOČI DEL TABELE VSEBUJE NAJVEČ 2 ELEMENTA. ČE STA 2,
*        JU, ČE JE POTREBNO, UREDIMO.
*
            IF ((DELTA.EQ.1).AND.(TAB(LEFT).GT.TAB(RIGHT))) THEN
               CHANGE      = TAB(LEFT)
               TAB(LEFT)   = TAB(RIGHT)
               TAB(RIGHT)  = CHANGE
            ENDIF
*
*        PRIPRAVIMO NOV DEL TABELE (MEJI VZAMEMO Z VRHA SKLADA); ČE GA
*        NI KONČAMO
*
            IF (DEPTH.LE.0) RETURN
            LEFT  = STACK(DEPTH-1)
            RIGHT = STACK(DEPTH)
            DEPTH = DEPTH - 2
         ENDIF
      ENDLOOP
*
      END
```

```
      SUBROUTINE SPLIT(TAB,LEFT,RIGHT,NLEFT,NRIGHT)
      INTEGER TAB(  1     )
      INTEGER CHANGE, CUT   , RIGHT , RSCAN , TEMP
*
*  SPLIT JE POMOZNA RUTINA PODPROGRAMA (QUICK)SORT. DEL TABELE
*  TAB(LEFT..RIGHT) RAZBIJE NA DALJSI DEL  TAB(NLEFT..NRIGHT)  IN
*  KRAJSI DEL  TAB(LEFT..RIGHT) .
*
*  DEL TABELE  TAB(LEFT..RIGHT)  PREUREDIMO TAKO, DA SO NA LEVI STRANI
*  DELILNEGA ELEMENTA VSI (OD NJEGA) MANJSI; NA DESNI PA VSI VECJI ALI
*  ENAKI ELEMENTI. S TEM DELILNI ELEMENT ZASEDE MESTO, KI GA ZASEDA V
*  UREJENI TABELI.
*
      CUT    = (LEFT + RIGHT)/2
      TEMP   = TAB(CUT)          ... DELILNI ELEMENT
      RSCAN  = RIGHT             ... ZGORNJI INDEKS SE NEPREUREJENEGA DELA
      LSCAN  = LEFT              ... SPODNJI INDEKS SE NEPREUREJENEGA DELA
*
*  TABELO PREUREJAMO TAKO, DA POISCEMO PRVI ELEMENT Z LEVE, KI JE VECJI,
*  IN PRVI ELEMENT Z DESNE, KI JE MANJSI OD DELILNEGA. ELEMENTA PREME-
*  NJAMO. TO PONAVLJAMO VSE DOKLER OBSTAJA NEPREUREJENI DEL TABELE.
*
      LOOP
        WHILE (TAB(LSCAN).LT.TEMP) DO
          LSCAN = LSCAN + 1
        ENDWHILE
        WHILE (TAB(RSCAN).GT.TEMP) DO
          RSCAN = RSCAN - 1
        ENDWHILE
*
*       DOBILI SMO DVA ELEMENTA, KI STOJITA NA NAPACNI STRANI DELILNEGA
*       ELEMENTA. PREMENJAMO JU.
*
        IF (LSCAN.GT.RSCAN) EXIT
        CHANGE     = TAB(RSCAN)
        TAB(RSCAN) = TAB(LSCAN)
        TAB(LSCAN) = CHANGE
        RSCAN      = RSCAN - 1
        LSCAN      = LSCAN + 1
      ENDLOOP
*
*  DOLOCIMO NOVA DELA TABELE
*
      IF ((LEFT+RIGHT).LT.(LSCAN+RSCAN)) THEN
*
*       LEVI DEL JE VECJI
*
        NLEFT  = LEFT
        NRIGHT = RSCAN
        LEFT   = LSCAN
      ELSE
*
*       DESNI DEL JE VECJI
*
        NLEFT  = LSCAN
        NRIGHT = RIGHT
        RIGHT  = RSCAN
      ENDIF
      RETURN
*
      END
```