Programski jezik Java

Interno gradivo za predmet

Algoritmi in programski jeziki (4. letnik)

Sortirni algoritmi

(neprečiščeno besedilo)

bostjan.vouk@tsc.si

Urejanje

- Metode sortiranja razvrščamo v dve skupini, in sicer na notranje in zunanje metode.
- ***Notranje sortiranje***
  - kadar se podatki nahajajo v hitrih pomnilnikih z naključnim dostopom,
- ***Zunanje sortiranje***
  - podatki se nahajajo v počasnejših toda večjih zunanjih pomnilnikih
    - trdi diski, ki temeljijo na mehansko gibljivih napravah z zaporednim dostopom

bostjan.vouk@tsc.si

Urejanje

- Pri algoritmih za sortiranje nas vedno zanima tudi kako hitro uredijo elemente in koliko prostora potrebujejo za sortiranje.
- Ločimo algoritme, ki sortirajo elemente
  - na mestu,
    - zavzamejo samo toliko prostora, kolikor ga potrebujejo elementi
  - ki elemente prestavljajo na druge lokacije,
    - potrebujejo več prostora.
- Glede na časovno zahtevnost algoritmov pa ločimo
  - preprostejše,
    - elemente uredijo v času $O(n^2)$, tem pravimo navadne metode
  - hitrejše
    - elemente uredijo v času $O(n \log(n))$, če je

n število elementov

bostjan.vouk@tsc.si

Urejanje

- Kateri algoritem za sortiranje bomo izbrali, pa je seveda odvisno tudi od števila elementov.
- Navadne metode
  - preprostejše, asimptotično počasnejše
  - na majhnem številu elementov prav zaradi svoje preprostosti veliko bolj učinkovite, kot pa metode, ki elemente uredijo v logaritemskem času

bostjan.vouk@tsc.si

Zunanje sortiranje

- Algoritme ne moremo uporabiti, če podatkov, ki jih moramo urediti, ne moremo imeti v hitrem pomnilniku, ampak na zunanjih pomnilnih napravah, za katere vemo, da moramo do njih dostopati zaporedno.
- Na zunanjih pomnilnikih podatke shranjujemo v datotekah, kjer je v vsakem trenutku dosegljiv le en podatek. To pa je kar huda omejitev glede na to, da v polju lahko vsak trenutek dosežemo katerikoli element.
  - To rešujemo z uporabo sortirnih metod, ki jim pravimo zunanje metode.

bostjan.vouk@tsc.si

Zunanje sortiranje

- Pri zunanjih metodah naložimo le majhen del podatkov v glavni pomnilnik, uporabimo notranjo sortirno metodo in nato shranimo podatke na zunanje pomnilnike. Urejene dele nato zlijemo skupaj tako, da uporabimo eno od variant sortiranja z zlivanjem, ki ga imenujemo tudi Merge sort.
- Čeprav smo metodo zlivanja opredelili kot zunanjo metodo, je hkrati tudi notranja metoda in jo lahko uporabljamo za sortiranje polj. Tudi to metodo lahko priredimo tako, da elemente zamenjujemo na mestu, torej ne potrebujemo dodatnega prostora. Vendar je v tem primeru metoda v primerjavi z ostalimi notranjimi metodami precej neučinkovita. Če pa ji ponudimo malo dodatnega prostora, postane ena izmed najhitrejših med notranjimi metodami.

bostjan.vouk@tsc.si

Zunanje sortiranje

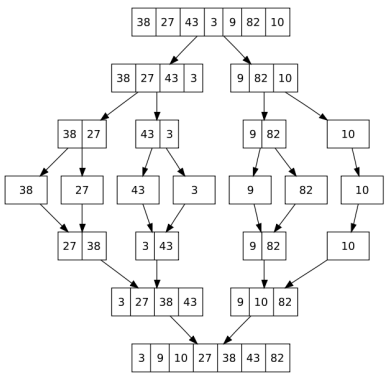
- Metoda Sortiranja z zlivanjem sortira elemente dobesedno z zlivanjem dveh zaporedij v enega, pri čemer izbira med trenutno dosegljivimi elementi. To pa pomeni, da morata biti zaporedji urejeni, da potem lahko s pomočjo primerjanja prvih elementov zaporedja dobimo eno urejeno zaporedje.
- Metoda temelji na strategiji deli in vladaj. Zaporedje najprej razbije na podzapredja, nato pa jih toliko časa zliva skupaj, da dobimo eno urejeno zaporedje.

bostjan.vouk@tsc.si

Opis algoritma

- Algoritem uporablja paradigmo deli in vladaj. Začne s podtabelami velikosti 1, ki so same po sebi seveda urejene. Nato začne z zlivanjem(in sprotnim urejanjem) parov podtabel v večjo podtabelo in te tabele nato zopet zliva v večje, dokler ne zlije vseh elementov tabele v eno urejeno tabelo.

bostjan.vouk@tsc.si



bostjan.vouk@tsc.si

Notranje sortiranje

- Vsi algoritmi za urejanje (sortiranje) podatkov slonijo na eni od treh osnovnih tehnik:
  - izbiranju
  - vstavljanju
  - premenami

bostjan.vouk@tsc.si

Urejanje z vstavljanjem

- Algoritem sortiranja z vstavljanjem uredi elemente tako, da po vrsti jemlje elemente in vsakega vstavi na svoje mesto.
- Začne z drugim elementom in ga primerja s prvim. Če je manjši od prvega, potem ga postavi na prvo mesto, sicer na drugo. Nato vzame tretji element in ga primerja z drugim. Če je manjši od drugega, ga primerja še s prvim, in v odvisnosti od rezultata vstavi tretji element na svoje mesto.
- Ta postopek ponavlja do zadnjega elementa.

bostjan.vouk@tsc.si

Opis algoritma

1. Spremenljivki i priredi 1;
2. i-ti element primerjaj z elementi pred njim, začenši z elementom na mestu i-1. Dokler ne naletiš na manjšega oziroma na začetek polja, vsak element prestavi za eno mesto naprej;
3. Vstavi i-ti element na svoje mesto;
4. Spremenljivki i priredi i+1;

- Ponavljaj korake 2, 3 in 4, dokler spremenljivka i ne preseže števila elementov.

bostjan.vouk@tsc.si

Psevdokoda

- Algoritem navadnega vstavljanja lahko zapišemo takole:
- for(i=1; i<a.length; i++){
  x=a[i];
  x vstavi na pravo mesto v $a_1 \ldots a_{i-1}$
  }

Metoda

- public static void urejanjeVstavljanje(Element[] a){
  int i, j;
  Element x;
  for(i=1; i<a.length; i++){
  if(a[i]>a[i-1]) continue; //izboljšava
  x=a[i];
  j=i-1;
  while(j>=0 && x<(a[j])){
  a[j+1]=a[j];
  j--;
  }
  a[j+1]=x;
  }
  }

Primer

- Podana je tabela elementov: 12,1,7,33,14,6,77,8,21,100.
- Napiši, kako se spreminja vsebina tabele pri urejanju z vstavljanjem.
  - 12,**1**,7,33,14,6,77,8,21,100.
    1,12,**7**,33,14,6,77,8,21,100
    1,7,12,**33**,14,6,77,8,21,100
    1,7,12,33,**14**,6,77,8,21,100
    1,7,12,14,33,**6**,77,8,21,100
    1,6,7,12,14,33,**77**,8,21,100
    1,6,7,12,14,33,77,**8**,21,100
    1,6,7,8,12,14,33,77,**21**,100
    1,6,7,8,12,14,21,33,77,**100**
    1,6,7,8,12,14,21,33,77,**100**

Analiza navadnega vstavljanja

- Če upoštevamo, da so vse permutacije elementov enako verjetne, lahko ocenimo število C_i primerjanj med elementi. V i-tem pogrezanju je primerjanj največ i-1 in najmanj 1, torej v povprečju i/2. Število M_i premikov oziroma prirejanj elementov pa je C_{i+2} (če upoštevamo tudi prirejanje vrednosti stražarju).

Analiza navadnega vstavljanja

- Celotno število primerjanj in premikov:
- najmanj $C_{min}=n-1$ $M_{min}=2(n-1)$
- povprečno $C_{ave}=1/4(n^2+n-2)$ $Mave=1/4(n^2+9n-10)$
- največ $C_{max}=1/2(n^2+n)-1$ $M_{max}=1/2(n^2+3n-4)$
- Najmanjše število primerjanj in premikov dobimo v primeru, ko so elementi že urejeni, največje pa v primeru, ko so elementi na začetku nasprotno urejeni. Kar bi logično tudi pričakovali, pa vendar pri vseh algoritmih to ne drži. Vidimo, da je algoritem tudi stabilen, saj medsebojni vrstni red elementov z enakimi ključi ostane nespremenjen.

Analiza navadnega vstavljanja

- Algoritem navadnega vstavljanja seveda lahko izboljšamo, saj je končno zaporedje, v katerega vstavljamo nove elemente, že urejeno. Kar pa pomeni, da bi lahko uporabili kakšno hitrejšo metodo za iskanje mesta za vstavljanje naslednjega elementa. Na primer dvojiško iskanje, kjer naredimo primerjavo z elementom na sredi končnega zaporedja in nadaljujemo z razpolavljanjem dokler ne najdemo ustreznega mesta za vstavljanje.

Analiza navadnega vstavljanja

- To lastnost ima algoritem, ki ga imenujemo *Sortiranje z vstavljanjem s padajočim prirastkom* oziroma Shell sortiranje po njegovem avtorju.
- Lahko rečemo, da ima algoritem Navadnega izbiranja prednost pred Navadnim vstavljanjem, je pa Navadno vstavljanje nekoliko hitrejše takrat, ko so elementi na začetku že sortirani ali pa skoraj urejeni.

bostjan.vouk@tsc.si

Urejanje z izbiranjem

- Sortiranje z Navadnim izbiranjem temelji na načelu iskanja oziroma izbiranja najmanjšega elementa.
- V vsakem prehodu skozi polje poiščemo najmanjši element in ga postavimo na prvo mesto, kar pomeni, da so začetni elementi polja urejeni in teh v naslednjih prehodih ne preverjamo več.

bostjan.vouk@tsc.si

Opis algoritma

1. Najdi najmanjši element in ga zamenjaj s prvim elementom polja.
2. Med ostalimi elementi najdi drugi najmanjši element v polju in ga zamenjaj z drugim elementom polja.
3. Ponavljaj do predzadnjega elementa polja.

bostjan.vouk@tsc.si

Psevdokoda

- Program, ki uredi n elementov, bi lahko zapisali takole:
- for (int i=0; i<a.length-1; i++){
  spremenljivki k priredimo indeks najmanjšega elementa med $a_i..a_n$
  zamenjava elementov na mestih a_i in a_k
  }

Metoda

-
```
public static void uredi_z_izbiranjem(int[] a) {
 int j,t;
    for (int i = 0; i < a.length-1; i = i + 1) {
       j = i;
       for (int k = i; k < a.length; k = k + 1) {
        if (a[k] < a[j]) {
            j = k;
        }
       }
       t = a[i];
       a[i] = a[j];
       a[j] = t;
    }
}
```


Analiza Navadnega izbiranja

- Število C primerjanj med ključi je neodvisno od začetne urejenosti elementov, saj moramo v vsakem prehodu poiskati najmanjši element, torej moramo narediti primerjavo z vsemi še neurejenimi elementi. Ugotovimo lahko: $C= ½(n^2 - n)$.

Analiza Navadnega izbiranja

- Število M premikov
  - najmanj $M_{min}=3(n-1)$, če so ključi na začetku urejeni
  - največ $M_{max}=trunc(n^2/4)+3(n-1)$, če so nasprotno urejeni.
- Čeprav je algoritem preprost, je povprečje M_{ave} težko oceniti, saj pri iskanju najmanjšega elementa ne moremo natančno vedeti, kolikokrat smo morali element zamenjati z drugim, da smo naleteli na pravega.
- Z matematičnimi postopki dobimo približno vrednost: $M_{ave}=n(ln(n)+\gamma)$, kjer je Eulerjeva konstanta $\gamma=0,577216\ldots$.

bostjan.vouk@tsc.si

Primer

- Podana je tabela elementov: 'G' 'C' 'S' 'B' 'W' 'c' 'P' 'r' 'Q' 'U'.
- Napišite, kako se spreminja vsebina tabele pri urejanju z izbiranjem.
  - 'G' 'C' 'S' 'B' 'W' 'c' 'P' 'r' 'Q' 'U'
    'B'|'C' 'S' 'G' 'W' 'c' 'P' 'r' 'Q' 'U'
    'B' 'C'|'S' 'G' 'W' 'c' 'P' 'r' 'Q' 'U'
    'B' 'C' 'G'|'S' 'W' 'c' 'P' 'r' 'Q' 'U'
    'B' 'C' 'G' 'P'|'W' 'c' 'S' 'r' 'Q' 'U'
    'B' 'C' 'G' 'P' 'Q'|'c' 'S' 'r' 'W' 'U'
    'B' 'C' 'G' 'P' 'Q' 'S'|'c' 'r' 'W' 'U'
    'B' 'C' 'G' 'P' 'Q' 'S' 'U'|'r' 'W' 'c'
    'B' 'C' 'G' 'P' 'Q' 'S' 'U' 'W'|'r' 'c'
    'B' 'C' 'G' 'P' 'Q' 'S' 'U' 'W' 'c'|'r'

bostjan.vouk@tsc.si

Sortiranje s premenami

- *Mehurčno sortiranje (Bubblesort)*
- public static void bubbleSort(Element[] a){

```
int i,j;
Element x;
for(i=1; i<a.length; ++i)
    for(j=a.length-1; j>=i; --j)
        if(a[j]<(a[j-1]))
        {
        x=a[j];
        a[j]=a[j-1];
        a[j-1]=x;
        }
}
```

bostjan.vouk@tsc.si

Primer

- Podana je tabela elementov: 'G' 'C' 'S' 'B' 'W' 'c' 'P' 'r' 'Q' 'U'. (GCSBWcPrQU)
- Napišite, kako se spreminja vsebina tabele pri urejanju s premenami.
  - 'G' 'C' 'S' 'B' 'W' 'c' 'P' 'r' 'Q' 'U'
    'B'|'G' 'C' 'S' 'P' 'W' 'c' 'Q' 'r' 'U'
    'B' 'C'|'G' 'P' 'S' 'Q' 'W' 'c' 'U' 'r'
    'B' 'C' 'G'|'P' 'Q' 'S' 'U' 'W' 'c' 'r'
    'B' 'C' 'G' 'P'|'Q' 'S' 'U' 'W' 'c' 'r'
    'B' 'C' 'G' 'P' 'Q'|'S' 'U' 'W' 'c' 'r'
    'B' 'C' 'G' 'P' 'Q' 'S'|'U' 'W' 'c' 'r'
    'B' 'C' 'G' 'P' 'Q' 'S' 'U'|'W' 'c' 'r'
    'B' 'C' 'G' 'P' 'Q' 'S' 'U' 'W'|'c' 'r'
    'B' 'C' 'G' 'P' 'Q' 'S' 'U' 'W' 'c'|'r'

bostjan.vouk@tsc.si

Analiza urejanja s premenami

- Na prejšnjem primeru vidimo, da zadnji trije prehodi niso vplivali na urejenost elementov, ker so bili elementi že sortirani.
- Zato lahko algoritem hitro izboljšamo tako, da si med vsakim prehodom zabeležimo, če je prišlo do kake premene. Potem potrebujemo samo en zadnji prehod brez premen, ki nam pove, da je polje urejeno, torej lahko postopek ustavimo.
- Še bolje je, če si zapomnimo tudi mesto oziroma indeks zadnje premene, saj vemo, da so vsi pari sosednih elementov za tem indeksom že pravilno urejeni. Kasnejše prehode lahko ustavimo pri tem indeksu, namesto da bi nadaljevali s pregledovanjem do konca polja.

bostjan.vouk@tsc.si

Analiza urejanja s premenami

- Napačno postavljen težji mehurček prispe na svoje mesto v enem samem prehodu, napačno postavljen lažji mehurček pa se bo proti svojemu mestu pomikal očitno bolj počasi, in sicer le za eno mesto v vsakem prehodu.
- Na primer, izboljšano Sortiranje z mehurčki bo za sortiranje polja 94 06 12 18 42 44 55 67 porabilo en prehod, za polje 12 18 42 44 67 94 06 pa kar sedem. Ta ugotovitev nas popelje do novega izboljšanja. Če menjamo smer prehodov, nam en napačno postavljen mehurček na težji strani ne more več narediti take škode. Ta algoritem ima primerno ime *Sortiranje s stresanjem.*

bostjan.vouk@tsc.si

Povezave do simulacij

- http://maven.smith.edu/~thiebaut/java/sort/demo.html
- http://www.sorting-algorithms.com/
- http://people.cs.ubc.ca/~harrison/Java/sorting-demo.html
- http://www.brian-borowski.com/Software/Sorting/

bostjan.vouk@tsc.si