Algoritmi in podatkovne strukture – 2

Uvod

uvod in matematične osnove

Algoritmi in podatkovne strukture – 2

predavanja: dr. Andrej Brodnik, andrej.brodnik@upr.si

vaje: Tine Šukljan, tine.sukljan@upr.si

e-viri:

- LMS: https://e.famnit.upr.si/course/view.php?id=315

Opis predmeta

Literatura:

- **Cormen, Leiserson, Rivest: *Introduction to Algorithms***,
- Sedgewick: Algorithms in Java.
- Kozak: *Podatkovne strukture in algoritmi*.
- `www.brodnik.org/andy/PoStrInA`.

Program:

- Prvi polovica *podatkovne strukture* in druga polovica *algoritmi*.

Ocenjevanje:

40%: sprotno delo

60%: končna ocena

obvezno: zapiski

Opis predmeta (nadalj.)

domače naloge:

- šest domačih nalog s po štirimi vprašanji, od katerih bo eno programersko (empirično)
- programersko vprašanje na način kot pri ACM ICPC (`http://cm2prod.baylor.edu/`), oziroma UPM (`http://www.upm.si/`)
- uporaba okolja za oddajo nalog
- oddaja preko e-učilnice

izpit: lahko nadomestita kolokvija, vendar mora biti skupna ocena pozitivna in nobeden od kolokvijev ne sme biti nižji od 40%

zapiski: oddajate jih e-učilnico

- pomembno, da se jih odda pravočasno: v enem tednu po predavanjih
- za to, da se semester pozitivno oceni, morate oddati vsaj ene zapiske

Okvirni program – prvi del

Uvod

- Uvod in matematične osnove.
- Osnovne podatkovne strukture: (i) Implicitne podatkovne strukture: polje, sklad, vrsta, kopica; (ii) Eksplicitne podatkovne strukture: povezan seznam, drevo

Številska drevesa

- Osnove, *trie*.
- Patricijina in LC drevesa.

Disjunktne množice

- Disjunktne množice.

Program – prvi del (nadalj.)

Slovar

- Osnove in izvedba s seznamom.
- Drevesa: osnove, iskalna, uravnotežena, AVL.
- B-drevesa in 2-3 drevesa.
- Rdeče-črna drevesa.
- Razpršene tabele.
- Preskočni seznami.

Vrste s prednostjo

- Osnove in kopica.
- Binomska in Fibbonacijeva kopica.
- Plastovita drevesa.

prvi delni izpit 4. april

Algoritem

- Vsak dobro definiran računski postopek, s katerim kaj izračunamo ali rešimo kak problem, imenujemo *algoritem*.
  Algoritem vhodne podatke (*input*) spremeni v izhodne podatke (*output*).
- Na kaj moramo biti posebej pozorni:
  - pravilnost,
  - časovna zahtevnost (število korakov v odvisnosti od velikosti problema).
- Algoritme – korake postopka – bomo opisavali v *psevdokodi*.
- Primer – urejanje.
  Vhod: Zaporedje n števil $(a_1, a_2, \ldots, a_n)$,
  Izhod: Premutacija $(a'_1, a'_2, \ldots, a'_n)$ vhodnega zaporedja, tako da velja $a'_1 \leq a'_2 \leq \cdots \leq a'_n)$.
  Za zaporedje $(31, 41, 59, 26, 41, 58)$ naj algoritem vrne zaporedje $(26, 31, 41, 41, 58, 59)$.

Urejanje z vstavljanjem

Eden izmed algoritmov za urejanje – *urejanje z vstavljanjem*:

INSERTION-SORT(A)

1: **for** $j \leftarrow 2$ **to** A.length **do**
2: $\quad key \leftarrow A[j]$
3: $\quad i \leftarrow j - 1$
4: $\quad$ **while** $i > 0$ and $A[i] > key$ **do**
5: $\quad\quad A[i + 1] \leftarrow A[i]$
6: $\quad\quad i \leftarrow i - 1$
7: $\quad$ **end while**
8: $\quad A[i + 1] \leftarrow key$
9: **end for**

Model računanja

- Za model računanja bomo vzeli *random-access machine (RAM)*, oziroma točneje *primerjalni model*.
  - Ukazi se izvajajo eden za drugim.
  - Predpostavljali bomo, da so vse operacije enako drage, čeprav to ne odraža dejanskega stanja, saj so množenja ter posebej še deljenja dražja.
- Obstajajo še drugi modeli, npr. *dostopni model*, kjer štejemo samo dostope do pomnilnika

Pravilnost delovanja

Dokaz z indukcijo.

Brez škode za splošnost predpostavimo, da so elementi, ki jih urejamo, med seboj različni.

Osnova: Če je samo en element (a_1), je polje že urejeno.

Hipoteza: Predpostavimo, da algoritem zna urediti elemente $(a_1, a_2, \ldots, a_{j-1})$ v $(a'_1 < a'_2 < \ldots < a'_{j-1})$.

Korak: Sedaj imamo polje elementov $(a_1, a_2, \ldots, a_j)$. Prvih $j - 1$ elementov po predpostavki znamo urediti. v $(a'_1 < a'_2 < \ldots < a'_{j-1})$. Za zadnji element se notranja while izvaja dokler je `key` manjši. Ustavi se pri i-tem elementu. Ko vstavimo ključ na to mesto, kar se zgodi v koraku 8, so elementi $(a'_1 < a'_2 < \ldots < a'_i < \texttt{key})$ urejeni. Poleg tega velja, zaradi oblike WHILE zanke, da ostajajo elementi $(a'_{i+1} < a'_{i+2} < \ldots < a'_{j-i})$ urejeni, hkrati pa velja $(\texttt{key} < a'_{i+1}, a'_{i+2}, \ldots, a'_{j-i})$. Zatorej $(a'_1 < a'_2 < \ldots < a'_i < \texttt{key} < a'_{i+1} < a'_{i+2} < \ldots < a'_{j-i})$.
$\mathcal{QED}$

Analiza časovne zahtevnosti algoritma

- Naj bo $T_{\mathcal{A}}(n)$ število izvedenih osnovnih operacij ali $\gg$korakov$\ll$ algoritma $\mathcal{A}$, pri vhodnih podatkih velikosti n.

  Funkciji $T_{\mathcal{A}}(n)$ rečemo *časovna zahtevnost ali čas izvajanja algoritma* $\mathcal{A}$.
- Predpostavljali bomo, da za izvajanje posameznega koraka potrebujemo konstantno časa, t.j. izvajanje i-te vrstice v vzame c_i časa.
- Izraz *velikost podatkov* je odvisen od problema, ki ga rešujemo.
  - V primeru urejanja je to kar *število elementov*, ki jih moramo urediti.
  - Pri množenju celih števil je to *število bitov* (števk), ki jih potrebujemo za dvojiški zapis vhodnih števil.

Analiza *urejanja z vstavljanjem*

INSERTION-SORT(A)	čas izvajanja	število izvajanj
1: **for** $j \leftarrow 2$ **to** A.length **do**	c_1	n
2: $\quad key \leftarrow A[j]$	c_2	$n-1$
3: $\quad i \leftarrow j-1$	c_3	$n-1$
4: $\quad$ **while** $i > 0$ and $A[i] > key$ **do**	c_4	$\sum_{j=2}^{n} t_j$
5: $\quad\quad A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(t_j-1)$
6: $\quad\quad i \leftarrow i-1$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7: $\quad$ **end while**		
8: $\quad A[i+1] \leftarrow key$	c_8	$n-1$
9: **end for**		

(n je dolžina niza A, t_j naj bo število ponovitev zanke **while** pri danem j.)

Analiza *urejanja z vstavljanjem 2*

Naj bo $T(n)$ čas izvajanja algoritma INSERTION-SORT. Velja

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1)$$

$$+ c_6 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Funkcija $T(n)$ je odvisna od podatkov (od t_j, ta števila pa so odvisna od podatkov).

Izračunajmo $T(n)$ v dveh primerih: v *najboljšem* (podatki so že urejeni) in v *najslabšem primeru* (podatki so urejeni v obratnem vrstnem redu).

Analiza *urejanja z vstavljanjem* – najboljši primer

V najboljšem primeru (podatki so že urejeni) velja $t_j = 1$:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_3 + c_4 + c_8)n - (c_2 + c_3 + c_4 + c_8) \\ &= an + b \end{aligned}$$

Torej je $T(n)$ linearna funkcija. Pravimo, da ima algoritem *v najboljšem primeru linearno časovno zahtevnost*.

Analiza *urejanja z vstavljanjem* – najslabši primer

V najslabšem primeru (podatki so urejeni obratno) velja $t_j = j$:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\frac{n(n+1)}{2} + \\ &\quad c_6\frac{n(n+1)}{2} + c_8(n-1) \\ &= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2} + c_8\right) n - \\ &\quad (c_2 + c_3 + c_4 + c_8) \end{aligned}$$

Torej je $T(n)$ kvadratna funkcija. Pravimo, da ima algoritem *v najslabšem primeru kvadratno časovno zahtevnost*.

Ko rečemo, da *ima algoritem . . . časovno zahtevnost*, mislimo na časovno zahtevnost v NAJSLABŠEM PRIMERU.

Ocena zahtevnosti problema

Imamo algoritem, ki uredi elemente v kvadratičnem času. Pa se dâ to narediti hitreje? Če smo se prej pogovarjali o *zahtevnosti algoritma*, se sedaj pogovarjamo o *zahtevnosti problema*.

RAZMISLEK:

- če imamo n elementov, obstaja $n!$ permutacij le-teh
- katerikoli algoritem za urejanje mora znati pretvoriti katerokoli permutacijo elementov v tisto pravo, urejeno permutacijo – z drugimi besedami, mora znati *razlikovati* med katerimakoli permutacijama
- recimo, da z eno primerjavo znamo razlikovati med dvema permutacijama in če želimo razlikovati med poljubnima permutacijama, lahko nad vsemi $n!$ permutacijami izgradimo drevo primerjav
- višina drevesa primerjav je najkrajši ≫čas≪, ki je potreben za urejanje in je $\lg n!$
- ker je (Stirling)

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \cdot C$$

potem $\lg n! \approx n \lg n + n \cdot C_1$.

Ocena velikosti funkcij in *veliki O*

Računanje časovne zahtevnosti v prejšnjem primeru je ≫zoprno≪. V praksi nas zanima le *red velikosti* (ocena velikosti): ali je funkcija linearna, kvadratična, kubična, $\gg \log n \ll$, $\gg n \log n \ll$, . . . funkcija.

Funkcije, nam popisujejo lahko *časovna zahtevnost* algoritma ali problema, ali pa *prostorsko zahtevnost* algoritma ali problema.

Za dano funkcijo $g(n)$ označimo z $O(g(n))$ *množico* funkcij

$$O(g(n)) = \left\{ f(n) \mid \begin{array}{l} \text{obstajata pozitivni konstanti } c \text{ in } n_0, \\ \text{tako da } f(n) \leq cg(n) \text{ za vse } n \geq n_0. \end{array} \right.$$

Če je $h(n) = O(g(n))$ potem rečemo, da $g(n)$ *asimptotično od zgoraj omejuje* $h(n)$.

Opomba: predpostavljamo, da so vse funkcije nenegativne.

Pravzaprav bi morali zapisati $h(n) \in \mathbf{O}(g(n))$, toda držali se bomo ustaljene (zlo)rabe!

Ocena velikosti funkcij in *veliki O* – primer

- $3n = O(n)$, $n = O(n^2)$.
- naš algoritem za urejanje je potreboval $O(n^2)$ primerjav.

Ocena velikosti funkcij in *velika* Ω

Za dano funkcijo $g(n)$ označimo z $\Omega(g(n))$ *množico* funkcij

$$\Omega(g(n)) = \left\{ f(n) \mid \begin{array}{l} \text{obstajata pozitivni konstanti } c \text{ in } n_0, \\ \text{tako da } f(n) \geq cg(n) \text{ za vse } n \geq n_0. \end{array} \right.$$

Če je $h(n) = \Omega(g(n))$ potem rečemo, da je $g(n)$ *asimptotično od spodaj omejuje* $h(n)$.

Z Ω zapisom običajno opisujemo *časovno zahtevnost problema* in ne algoritma.

Zahtevnost urejanja je $\Omega(n \log n)$; t.j. *katerikoli* algoritem za urejanje potrebuje *vsaj* $\gg n \log n \ll$ primerjav

Ocena velikosti funkcij in *velika* Θ

Za dano funkcijo $g(n)$ označimo z $\Theta(g(n))$ *množico* funkcij

$$\Theta(g(n)) = \left\{ f(n) \mid \begin{array}{l} \text{obstajajo pozitivne konstante } c_1, c_2 \text{ in } n_0, \\ \text{tako da } c_1 f(n) \geq c_2 g(n) \text{ za vse } n \geq n_0. \end{array} \right.$$

kar je enakovredno

$$(f(n) = O(g(n))) \wedge (f(n) = \Omega(g(n))) \equiv (f(n) = \Theta(g(n)))$$

Dokažite to!

Če je $h(n) = \Theta(g(n))$ potem rečemo, da je $h(n)$ *asimptotično omejena z* $g(n)$.

Ocena velikosti funkcij in *velika* Θ – primer

- $\frac{1}{2}n^2 - 3n = \Theta(n^2)$
- Recimo, da bi nadomestili naš algoritem za urejanje, ki zahteva $O(n^2)$ primerjav z algoritmom, ki bi zahteval $O(n \log n)$ primerjav. Ker je toliko primerjav tudi najmanj potrebnih (zahtevnost problema je $\Omega(n \log n)$, potem bi ta, najdeni algoritem bil *optimalen* in bi tekel v resnici v času $\Theta(n \log n)$. Zakaj?

Ocena velikosti funkcij in *mali* o

Za dano funkcijo $g(n)$ označimo z $o(g(n))$ množico funkcij

$$o(g(n)) = \left\{ f(n) \mid \begin{array}{l} \text{za poljubno konstanto } c > 0 \text{ obstaja} \\ \text{konstanta } n_0 > 0, \text{ tako da } f(n) < cg(n) \text{ za vse } n \geq n_0. \end{array} \right.$$

Intuitivni pomen: če je $f(n) = o(g(n))$, potem je $f(n)$ vedno manjša od $g(n)$, ko n večamo čez vse meje.

Enakovredna definicija:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \ .$$

Primer: $2n = o(n^2)$, $2n^2 \neq o(n^2)$, $7n = o(n^{1+\epsilon})$.

Ocena velikosti funkcij in *mali* ω

Za dano funkcijo $g(n)$ označimo z $\omega(g(n))$ množico funkcij

$$\omega(g(n)) = \left\{ f(n) \mid \begin{array}{l} \text{za poljubno konstanto } c > 0 \text{ obstaja} \\ \text{konstanta } n_0 > 0, \text{ tako da } f(n) > cg(n) \text{ za vse } n \geq n_0. \end{array} \right.$$

Intuitivni pomen: če je $f(n) = \omega(g(n))$, potem je $f(n)$ vedno večja od $g(n)$, ko n večamo čez vse meje

Enakovredna definicija:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \ .$$

Primer: $n^2/2 = \omega(n)$, $n^2/2 \neq \omega(n^2)$, $7n = \omega(n^{1-\epsilon})$.

Ocena velikosti funkcij in primerjava s števili

$$
\begin{aligned}
f(n) = &\quad \omega(g(n)) &&\approx a > b \\
f(n) = &\quad \Omega(g(n)) &&\approx a \geq b \\
f(n) = &\quad \Theta(g(n)) &&\approx a = b \\
f(n) = &\quad O(g(n)) &&\approx a \leq b \\
f(n) = &\quad o(g(n)) &&\approx a < b
\end{aligned}
$$

Še en algoritem

X-FUNCTION(p, r)

1: **if** $p < r$ **then**
2: $\quad q \leftarrow \lfloor (p + r)/2 \rfloor$
3: $\quad$ X-FUNCTION(p, q)
4: $\quad$ X-FUNCTION$(q + 1, r)$
5: $\quad$ Y-FUNCTION(p, q, r)
6: **end if**

Če je $r - p$ časovna zahtevnost funkcije Y-FUNCTION(p, q, r), kakšna je potem časovna zahtevnost $T(n)$ funkcije X-FUNCTION$(1, n)$?

Očitno velja:

$$T(n) = \begin{cases} \Theta(1) & n = 1, \\ 2T(n/2) + \Theta(n) & n > 1. \end{cases}$$

In $T(n)$ je potem koliko?

Splošno pravilo

Izrek. Naj bosta a, b konstanti, $a \geq 1$, $b > 1$, naj bo $f(n)$ funkcija in naj bo $T(n)$ funkcija definirana na nenegativnih celih številih, ki zadošča enačbi

$$T(n) = aT(n/b) + f(n) \ ,$$

kjer interpretiramo n/b kot $\lfloor n/b \rfloor$ ali $\lceil n/b \rceil$.

Potem velja:

1. Če je $f(n) = O(n^{\log_b a - \epsilon})$ za nek $\epsilon > 0$, potem $T(n) = \Theta(n^{\log_b a})$.
2. Če je $f(n) = \Theta(n^{\log_b a})$, potem $T(n) = \Theta(n^{\log_b a} \log n)$.
3. Če je $f(n) = \Omega(n^{\log_b a + \epsilon})$ za nek $\epsilon > 0$ in, če velja $af(n/b) \leq cf(n)$ za neko konstanto $c < 1$ in vse dovolj velike n, potem $T(n) = \Theta(f(n))$.

Glej Cormen, *Master theorem*.