OBLIKOVANJE I ANALIZA ALGORITAMA — 2. kolokvij

25. 1. 2010.

Rješenje prvog zadatka (magnetska traka)

1. (20) Magnetska traka sadrži n programa. Duljina programa i je l_i, za $i = 1, \ldots, n$. Vjerojatnost da s trake treba učitati program i je p_i, $i = 1, \ldots, n$. Pretpostavljamo da je

$$\sum_{i=1}^{n} p_i = 1.$$

Gustoća snimanja programa na traci je konstantna i brzina kretanja trake prilikom čitanja je konstantna. Svaki puta kada treba učitati neki program, traka se premotava na početak. Ako su programi na traci spremljeni u poretku i_1, i_2, …, i_n, vrijeme potrebno za **čitanje** j-tog programa (s indeksom i_j) je

$$t_{i_j} = c \sum_{k=1}^{j} l_{i_k},$$

gdje je c neka konstanta (ovisi o gustoći pisanja na traku i brzini rada trake). **Prosječno** vrijeme potrebno za učitavanje programa je

$$T = \sum_{j=1}^{n} p_{i_j} t_{i_j} = c \sum_{j=1}^{n} p_{i_j} \sum_{k=1}^{j} l_{i_k}.$$

Redoslijed spremanja na traku je optimalan ako daje **minimalno** prosječno vrijeme T učitavanja programa.

(a) Pokažite primjerom da spremanje programa na traku u redoslijedu rastućem po l_i ne mora biti optimalno. Pokažite primjerom da spremanje programa na traku u redoslijedu padajućem po p_i, također, ne mora biti optimalno.

(b) Nađite redoslijed u kojem treba spremati programe na traku, tako da prosječno vrijeme učitavanja programa bude **minimalno**. Dokažite optimalnost tog redoslijeda. Uputa: analizirajte slučaj $n = 2$.

(c) Sastavite algoritam koji nalazi optimalni redoslijed i nađite njegovu složenost.

Rješenje.

(a) Primjer da spremanje **rastuće** po l_i **ne mora** biti optimalno. Uzmimo samo 2 programa. Ideja: drugi ima **malo** veću duljinu i **puno** veću vjerojatnost od prvog. Na pr., ovako

i	p_i	l_i
1	0.2	1
2	0.8	2

Ako programe spremimo u navedenom redoslijedu $(1, 2)$, pripadno prosječno vrijeme učitavanja je

$$\begin{aligned} T(1,2) &= c\,[\,p_1 l_1 + p_2(l_1 + l_2)\,] \\ &= c\,[\,0.2 \cdot 1 + 0.8 \cdot (1+2)\,] = c(0.2 + 2.4) \\ &= 2.6c. \end{aligned}$$

Ako programe spremimo u obratnom redoslijedu $(2,1)$, pripadno prosječno vrijeme učitavanja je

$$\begin{aligned} T(2,1) &= c\,[\,p_2 l_2 + p_1(l_1+l_2)\,] \\ &= c\,[\,0.8\cdot 2 + 0.2\cdot(1+2)\,] = c(1.6+0.6) \\ &= 2.2c. \end{aligned}$$

Dakle, obratni redoslijed je bolji!

Primjer da spremanje **padajuće** po p_i **ne mora** biti optimalno. Uzmimo samo 2 programa. Ideja: drugi ima **malo** manju vjerojatnost i **puno** manju duljinu od prvog. Na pr., ovako

i	p_i	l_i
1	0.6	2
2	0.4	1

Ako programe spremimo u navedenom redoslijedu $(1,2)$, pripadno prosječno vrijeme učitavanja je

$$\begin{aligned} T(1,2) &= c\,[\,p_1 l_1 + p_2(l_1+l_2)\,] \\ &= c\,[\,0.6\cdot 2 + 0.4\cdot(1+2)\,] = c(1.2+1.2) \\ &= 2.4c. \end{aligned}$$

Ako programe spremimo u obratnom redoslijedu $(2,1)$, pripadno prosječno vrijeme učitavanja je

$$\begin{aligned} T(2,1) &= c\,[\,p_2 l_2 + p_1(l_1+l_2)\,] \\ &= c\,[\,0.4\cdot 1 + 0.6\cdot(1+2)\,] = c(0.4+1.8) \\ &= 2.2c. \end{aligned}$$

Dakle, obratni redoslijed je bolji!

(b) Kako naći "pravi" kriterij optimalnosti? Pogledajmo najjednostavniji slučaj kad imamo samo $n=2$ programa s podacima (p_1,l_1) i (p_2,l_2). Usporedimo prosječna vremena učitavanja za oba moguća rasporeda.

Ako programe spremimo u redoslijedu $(1,2)$, pripadno prosječno vrijeme učitavanja je

$$T(1,2) = c\,[\,p_1 l_1 + p_2(l_1+l_2)\,],$$

a ako ih spremimo u obratnom redoslijedu $(2,1)$, pripadno prosječno vrijeme učitavanja je

$$T(2,1) = c\,[\,p_2 l_2 + p_1(l_1+l_2)\,].$$

Razlika ovih prosječnih vrijednosti je (članovi $p_1 l_1$ i $p_2 l_2$ se skrate)

$$T(1,2) - T(2,1) = c\,[\,p_2 l_1 - p_1 l_2)\,].$$

Gledamo kad je prvi raspored **bolji** od drugog, tj. kad je $T(1,2) \le T(2,1)$. To je ekvivalentno s tim da je razlika $T(1,2)-T(2,1)$ negativna, odakle izlazi (koristimo $c>0$)

$$p_2 l_1 \le p_1 l_2 \quad \Longleftrightarrow \quad \frac{p_2}{l_2} \le \frac{p_1}{l_1}.$$

Dakle, poredak $(1, 2)$ je bitno bolji od $(2, 1)$ ako i samo ako vrijedi

$$\frac{p_1}{l_1} > \frac{p_2}{l_2}.$$

Ako vrijedi jednakost, oba poretka su jednako dobra (imaju isto prosječno vrijeme učitavanja), a ako vrijedi obratna nejednakost ($<$), onda je obratni poredak bolji. Drugim riječima, **optimalni** poredak mora biti **padajući** po omjeru "vjerojatnost po duljini".

Tvrdimo da isto mora vrijediti i općem slučaju, za bilo koji n.

Tvrdnja. Poredak $(i_1, i_2, \ldots, i_n)$ je **optimalan**, tj. daje minimalno prosječno vrijeme učitavanja T, ako i samo ako su omjeri "vjerojatnost po duljini" poredani **silazno** u tom poretku

$$\frac{p_{i_1}}{l_{i_1}} \geq \frac{p_{i_2}}{l_{i_2}} \geq \cdots \geq \frac{p_{i_n}}{l_{i_n}}.$$

Prije dokaza, uočimo da je optimalni poredak **jedinstven**, ako i samo ako na svim mjestima vrijede **stroge** nejednakosti ($>$). Ako na nekom mjestu (ili na više mjesta) vrijedi **jednakost**, onda takvih optimalnih poredaka ima **više**:

- u bilo kojem **uzastopnom** bloku jednakosti, pripadne indekse možemo permutirati bilo kako, što **neće** promijeniti optimalnu vrijednost za T (pogledajte slučaj jednakosti za $n = 2$).

Tvrdnju možemo dokazati na više različitih načina. Recimo:

- Uspoređujemo "našu" permutaciju (iz algoritma) i neku drugu permutaciju, koja je navodno "optimalna", tj. **bolja** od naše. Nađemo prvo mjesto razlike u ta dva poretka (permutacije) — na tom mjestu, ako je razlika bitna, nije znak jednakosti, već "optimalna" permutacija ima (strogu) obratnu nejednakost ($<$). Zatim dokažemo da zamjena pripadnih **susjednih** indeksa, tako da dobijemo znak $>$, strogo smanjuje pripadni T — primjenom analognog argumenta kao za $n = 2$. Time dobivamo kontradikciju s pretpostavkom da je "optimalna" permutacija zaista optimalna, jer smo našli bolju.
- Slično prethodnom, na prvom mjestu razlike od "naše" permutacije, u navodno "optimalnoj" se "naš" element mora nalaziti **iza** onog kojeg uzme "optimalna" (jer nema razlike prije toga). Zatim zamijenimo ta dva (ne moraju biti susjedi) i dokažemo da to smanjuje T (osim ako su sve jednakosti između pripadnih omjera za ta dva elementa). Tu možemo iskoristiti da se svaka zamjena (transpozicija) **neka** dva elementa može prikazati kao kompozicija zamjena (transpozicija) **susjednih** elemenata.
- Svaka permutacija je kompozicija transpozicija (zamjena neka dva elementa) = kompozicija transpozicija susjednih elemenata, pa ispravljamo poredak, sve dok ima "krivih" znakova nejednakosti u poretku omjera (susjednih ili udaljenih elemenata). Stvarno, na ovaj "bubblesort" argument se svodi i varijanta dokaza u nastavku!

Svi dokazi svode se na primjenu osnovnog argumenta za $n = 2$ (transpozicija susjeda), a gledanje permutacija ponešto komplicira argument, ako u optimalnom poretku ima jednakosti (optimalna permutacija nije jedinstvena).

U nastavku ide (po mom mišljenju) najjednostavniji korektni dokaz tvrdnje za "korektno" sortirani poredak. Argument je isti kao i "Uvod u Bubblesort" (pogledati 13. predavanje iz Prog1):

- Ako niz vrijednosti (u ovom slučaju — navedenih omjera) **nije** sortiran (silazno), onda u njemu postoji bar jedan par **susjeda** koji su u "obratnom" = nekorektnom poretku (u ovom slučaju, uzlaznom).

Dokaz tvrdnje. Uzmimo bilo koji poredak $(i_1, i_2, \ldots, i_n)$ programa na traci. Prosječno vrijeme učitavanja za taj poredak je

$$T_0 = c \sum_{j=1}^{n} p_{i_j} \left(\sum_{k=1}^{j} l_{i_k} \right).$$

Pretpostavimo da pripadni niz "bitnih" vrijednosti = omjera "vjerojatnost po duljini" **nije silazno** sortiran, tj. da **postoji** indeks $m \in \{1, \ldots, n-1\}$, takav da za **susjedne** omjere na mjestima m i $m+1$ vrijedi obratni = nekorektni (uzlazni) poredak

$$\frac{p_{i_m}}{l_{i_m}} < \frac{p_{i_{m+1}}}{l_{i_{m+1}}}.$$

Trenutno, uopće **nije** bitno u kojem poretku su ostali susjedni omjeri!

Pogledajmo što će se dogoditi ako u polaznom poretku samo **zamijenimo** mjesta susjeda i_m i i_{m+1}. Radi jednostavnosti, za indekse novog poretka koristimo novu oznaku i'_j, a nakon osnovnog objašnjenja, sve pišemo u terminima polaznih indeksa. Novi poredak programa na traci je

$$(i'_1, \ldots, i'_{m-1}, i'_m, i'_{m+1}, i'_{m+2}, \ldots, i'_n) = (i_1, \ldots, i_{m-1}, \underbrace{i_{m+1}, i_m}_{\text{zamjena}}, i_{m+2}, \ldots, i_n).$$

U pripadnom prosječnom vremenu učitavanja T_1

$$T_1 = c \sum_{j=1}^{n} p_{i'_j} \left(\sum_{k=1}^{j} l_{i'_k} \right),$$

zbog komutativnosti sume (u zagradi), različiti su **samo** članovi za "okrenute" indekse $j = m, m+1$ — jer je $i'_m = i_{m+1}$, $i'_{m+1} = i_m$, a za sve ostale indekse vrijedi $i'_j = i_j$, $i'_k = i_k$, gdje je $j, k \neq m, m+1$.

Za **razliku** prosječnih vremena $T_0 - T_1$ **podijeljenu** s c (smatramo da je $c > 0$, kao i ranije), onda vrijedi — krate se svi članovi za $j \neq i_m, i_{m+1}$, a zatim za indekse vjerojatnosti uvrstimo $i'_m = i_{m+1}$, $i'_{m+1} = i_m$,

$$\begin{aligned}
\frac{T_0 - T_1}{c} &= p_{i_m} \left(\sum_{k=1}^{m} l_{i_k} \right) + p_{i_{m+1}} \left(\sum_{k=1}^{m+1} l_{i_k} \right) \\
&\qquad - p_{i'_m} \left(\sum_{k=1}^{m} l_{i'_k} \right) - p_{i'_{m+1}} \left(\sum_{k=1}^{m+1} l_{i'_k} \right) \\
&= p_{i_m} \left(\sum_{k=1}^{m} l_{i_k} \right) + p_{i_{m+1}} \left(\sum_{k=1}^{m+1} l_{i_k} \right) \\
&\qquad - p_{i_{m+1}} \left(\sum_{k=1}^{m} l_{i'_k} \right) - p_{i_m} \left(\sum_{k=1}^{m+1} l_{i'_k} \right).
\end{aligned}$$

Kad "spojimo" članove s **istim** vjerojatnostima, dobivamo

$$\frac{T_0 - T_1}{c} = p_{i_m}\left(\sum_{k=1}^{m} l_{i_k} - \sum_{k=1}^{m+1} l_{i'_k}\right) + p_{i_{m+1}}\left(\sum_{k=1}^{m+1} l_{i_k} - \sum_{k=1}^{m} l_{i'_k}\right)$$

Sad iskoristimo da je $i'_k = i_k$ za sve $k < m$ i skratimo pripadne članove u sumama u **okruglim** zagradama, a zatim još uvrstimo $i'_m = i_{m+1}$, $i'_{m+1} = i_m$,

$$\begin{aligned}\frac{T_0 - T_1}{c} &= p_{i_m}\big(l_{i_m} - (l_{i_m} + l_{i_{m+1}})\big) + p_{i_{m+1}}\big(l_{i_m} + l_{i_{m+1}} - l_{i_{m+1}}\big) \\ &= -p_{i_m} l_{i_{m+1}} + p_{i_{m+1}} l_{i_m}.\end{aligned}$$

No, iz polazne pretpostavke "obratnog" = nekorektnog (uzlaznog) poretka

$$\frac{p_{i_m}}{l_{i_m}} < \frac{p_{i_{m+1}}}{l_{i_{m+1}}}$$

slijedi

$$p_{i_m} l_{i_{m+1}} - p_{i_{m+1}} l_{i_m} < 0.$$

Kad ovdje okrenemo predznak, zbog $c > 0$, izlazi $T_0 - T_1 > 0$, odnosno, $T_0 > T_1$.

Zaključak: Ako **postoji** par **susjeda** sa **strogo** "obratnim" poretkom bitnih vrijednosti za problem (ovdje su to omjeri "vjerojatnost po duljini"), od korektnog poretka (ovdje, korektan poredak je silazni), onda **zamjena** nekorektnog para susjeda sigurno **poboljšava** kriterij optimalnosti (ovdje, strogo smanjuje prosječno vrijeme učitavanja T).

Dakle, u (bilo kojem) **optimalnom** poretku **ne smije** postojati par susjeda u obratnom poretku, po **bitnim** vrijednostima za problem, tj. svi parovi susjeda, a onda (po "Bubblesort" argumentu) i **sve** vrijednosti moraju imati **ispravan** poredak — sortiran po istom kriteriju kao za samo **dva** "objekta" (ovdje, objekti su programi).

Uočite da prethodni argument ne gleda "permutacije", već samo pripadni poredak "bitnih" vrijednosti — tj. dozvoljava razne permutacije (ako postoje) koje daju **isti** (optimalni = sortirani) poredak "bitnih" vrijednosti!

(c) Za algoritam je **nužno** potrebno sortiranje (u propisnom **smjeru** = silazno ili uzlazno), po odgovarajućem kriteriju optimalnosti ("bitne" vrijednosti) za dani problem.

Sasvim općenito, obzirom na ulazne podatke (ovdje su to p_i, l_i), za sortiranje je potrebno

- polje **struktura** (C-aški rečeno), koje sadrži odgovarajuće "bitne" vrijednosti (ovdje p_i/l_i) po kojima treba sortirati objekte (u **propisnom** smjeru, tj. silazno ili uzlazno)
- i još treba omogućiti pamćenje pripadne **permutacije** objekata, nakon sortiranja. Inicijalna permutacija je $(1, \ldots, n)$, kako su zadani objekti.

Dakle, kad se zamjenjuju objekti prilikom soritranja, potrebno je mijenjati i **indekse** u pripadnoj permutaciji, a ne samo "bitne" vrijednosti.

Dozvoljeno je iz ulaznih podataka kreirati polje struktura, u kojem i-ta struktura sadrži (ako treba, ulazne vrijednosti — ovdje p_i, l_i), "bitne" vrijednosti za problem (ovdje p_i/l_i) i inicijalne vrijednosti permutacije (i) za

"sortirani" (= optimalni) poredak, s tim da se kod sortiranja "zamjenjuju" bitne vrijednosti (p_i/l_i) i pripadni indeksi i.

Možete eksplicitno napisati neki algoritam za (propisno) sortiranje odgovarajućih struktura (na pr., izborom ekstrema). Umjesto toga, smijete pozvati neki "standardni" algoritam za sortiranje, ali **morate** navesti **koji** (recimo, `qsort` iz standardne C biblioteke), i morate navesti **kriterij** po kojem se sortira ("ključ" za usporedbe i u kojem smjeru, odnosno, poretku = **uzlazno** ili **silazno**). Nadalje, morate navesti što se sve **mijenja** prilikom zamjena objekata (za korektnu permutaciju).

Kod argumentacije (tražene) složenosti, morate **korektno** iskoristiti odgovarajuću složenost napisanog/pozvanog algoritma.