OBLIKOVANJE I ANALIZA ALGORITAMA — 2. kolokvij

27. 1. 2016.

Skice rješenja prva dva zadatka

1. (20) Zadano je n poslova. Svaki posao je zadan kao vremenski interval realnih brojeva, $P_i = [p_i, k_i]$, za $i = 1, \ldots, n$, gdje je p_i početak, a k_i kraj i-tog posla. Pretpostavljamo da svi poslovi imaju pozitivno trajanje, tj. da je $p_i < k_i$, za $i = 1, \ldots, n$. Različiti poslovi smiju imati presjek pozitivne duljine (trajanja).

Ove poslove treba rasporediti na m izvršitelja, i to tako da se poslovi koje radi svaki **pojedini** izvršitelj smiju preklapati u samo **jednoj** točki (kraj jednog posla smije biti početak drugog posla). Možete zamisliti da jedan posao predstavlja nastavu iz nekog kolegija (zadanu početkom i krajem), a izvršitelji su predavaonice u koje treba rasporediti svu zadanu nastavu, tako da nema preklapanja nastave ni u jednoj predavaonici.

(a) Koliki je **najmanji** broj izvršitelja m_P potreban da se korektno izvrše svi poslovi, uz zadani uvjet? Precizno argumentirajte! Sastavite algoritam koji, za zadani niz poslova P, nalazi i vraća najmanji potreban broj izvršitelja m_P. Nađite složenost tog algoritma u ovisnosti o n.

Zadan je niz poslova P i raspoloživi broj izvršitelja m. Izvršitelje numeriramo brojevima od 1 do m, a posao P_i dodjeljujemo izvršitelju j, tako da u izlaznom polju *radi* postavimo $radi[i] = j$.

(b) Sastavite algoritam koji raspoređuje zadane poslove na zadani broj izvršitelja, tako da vraća polje *radi* od n elemenata. Ako je $m \geq m_P$, algoritam treba iskoristiti samo prvih m_P izvršitelja. U protivnom, za $m < m_P$, onim poslovima P_i koje nije moguće izvršiti treba postaviti $radi[i] = 0$. Nije potrebno minimizirati broj ili trajanje poslova koje je nemoguće izvršiti (samo ih označite). Nađite složenost tog algoritma u ovisnosti o n.

Napomena: oba algoritma moraju imati složenost $O(n^2)$ u ovisnosti o n. Dozvoljeno je koristiti iste pomoćne algoritme (funkcije) u oba algoritma (ne treba ih dva puta pisati, jednom je dovoljno).

Rješenje.

(a) Koliko izvršitelja trebamo?

Gledamo bilo koji vremenski trenutak t između najranijeg početka i najkasnijeg kraja. Izbrojim u koliko intervala se t nalazi (recimo, ne brojim interval ako mu je t krajnja točka). Onda nađem **maksimum** po svim tako dobivenim brojevima, za sve t-ove.

Naravno, to **ne** ide algoritamski — t-ova je (neprebrojivo) beskonačno. Dovoljno je gledati samo **rubove** svih intervala — u koliko intervala se oni nalaze. Dakle, dovoljno za t-ove uzeti točke p_i, k_i, za $i = 1, \ldots, n$ (njih je $2n$, tj. konačno mnogo). Kod pripadnog brojanja "u koliko intervala se točka nalazi", treba samo paziti na to da presjek bude **pozitivne** duljine (recimo, intervale treba gledati kao poluotvorene — početak mu pripada, a kraj ne).

Baš zbog toga, čini se da je zgodnije zaista gledati **presjeke** intervala, tj. brojati presjeke pozitivne duljine. Oprez — kad prelazim na presjeke intervala, **ne** valja brojati ovako:

- Za dani P_i, pogledam broj **svih** intervala P_j, uz $j \neq i$, takvih da je presjek P_i i P_j pozitivne duljine (i još dodam 1, za sam P_i).

Naime, P_i moze sijeći dva intervala koji se međusobno **ne sijeku**, pa je dovoljno 2 izvršitelja (predavaonice), a ne 3.

Kako treba gledati? Odgovor = točno onako kako bih punio predavaonice "na ruke".

1. Sortirati sve intervale uzlazno po vremenu pocetka p_i. Nakon toga prolazim po P_i u tom **uzlaznom** poretku.
2. Za svaki i (idemo od 1 do n), nađem broj m_i = s koliko **prethodnih** intervala P_j ($j \leq i$), interval P_i ima neprazan presjek (pozitivne duljine) — tu brojim i njega samog! Ova funkcija za presjek s prethodnima se lako napiše.

Nađem m_P = maksimum od m_i, po svim $i = 1, \ldots, n$. To je **najmanji** broj izvršitelja. Očito je da manje **ne može**.

Tvrdim da je to dovoljno, tj. da s m_P ide. Dokaz ide algoritmom za raspored (pretpostavljamo da su P_i već sortirani). U nastavku su dvije varijante algoritma (bitna razlika je u poretku petlji).

(b1) Varijanta 1: Imam jedno polje `kraj`, u kojem pamtim trenutni kraj aktivnosti za k-tog izvršitelja (do tada). Inicijaliziram broj izvršitelja `kmax` na 1 i njegov `kraj` na kraj prvog posla P_1, tj. stavim `kraj`[`kmax`] = 1.

Petlja po poslovima P_i, za $i = 2, \ldots, n$:
  Stavi `radi`[`i`] = 0;
  Nađi slobodnog izvršitelja među postojećim, tj. za $k = 1, \ldots,$ `kmax`
    Ako je `kraj`[`k`] $\leq$ početak p_i // dodijeli ga k-tom
      `radi`[`i`] = k; `kraj`[`k`] = k_i; `break`;
  Ako je (nakon petlje) `radi`[`i`] = 0, onda treba novi izvršitelj (ako ga ima)
    Ako je `kmax` $< m$ onda // ima još izvršitelja
      `kmax`++; `radi`[`i`] = `kmax`; `kraj`[`kmax`] = k_i;
// kraj petlje po i

Ako je `radi`[`i`] = 0 nakon prve petlje, to upravo znači da P_i ima presjek pozitivne duljine s točno `kmax` prethodnih intervala (onih s $j < i$) i onda nam treba novi izvršitelj (to je onaj "+1" za njega samog). Potreban broj izvršitelja je m_P = `kmax`.

(b2 Varijanta 2: Inicijalizacija = svima stavim `radi`[`i`] = 0, za $i = 1, \ldots, n$. Okrenem petlje, tako da **vanjska** ide po izvršiteljima.

Onda nam **ne** treba cijelo polje `kraj` (za sve izvršitelje), već je dovoljna jedna varijabla `kraj`, koja pamti kraj za trenutnog izvršitelja. Varijabla `ima_nerasp` (logičkog tipa) pamti ima li još neraspoređenih poslova ili ne.

Petlja po izvršiteljima, za $k = 1, \ldots, m$
  `ima_nerasp` = `false`;
  Petlja po poslovima P_i, za $i = 1, \ldots, n$:
    Ako je `radi`[`i`] = 0 // neraspoređen
      Ako je `ima_nerasp` = `false` // ovo je prvi takav
        `radi`[`i`] = k; `kraj` (k-tog) = k_i; `ima_nerasp` = `true`;
      `else` // nije prvi takav
        Ako mu je $p_i \geq$ `kraj` rasporedi ga na k-tog (inače ga preskoči)
          `radi`[`i`] = k; `kraj` (k-tog) = k_i;
  // kraj petlje po i

  Ako je `ima_nerasp` = `false`
    $m_P = k - 1$; `break`; // prekid petlje po k
// kraj petlje po k

Može i malo elegantnije, tako da brojim sve do tada raspoređene (kad ga rasporedim, povećam globalni brojac za 1). Ako je taj broj = n (na kraju petlje za dani k), onda stavim $m_P = k$ i `break`.

2. (30) Promatramo problem "segmentnih najmanjih kvadrata": zadan je niz od n točaka u ravnini, $P = (P_1, \ldots, P_n)$, gdje je $P_i = (x_i, y_i)$, za $i = 1, \ldots, n$, s tim da je $x_1 < x_2 < \cdots < x_n$. Ovaj niz treba podijeliti (particionirati) u neki broj segmenata. Svaki **segment** je neki podniz uzastopnih točaka iz P (gledano po indeksima ili x-koordinatama), tj. dovoljno je gledati prvu i zadnju točku

$$S_{i,j} = (P_i, P_{i+1}, \ldots, P_{j-1}, P_j), \quad i \leq j.$$

Segment smije biti i jednočlan.

Ideja je naći takvu particiju zadanih točaka da točke u svakom pojedinom segmentu približno leže na istom pravcu, a ti pravci se smiju razlikovati za razne segmente.

Svakom segmentu $S_{i,j}$ **pridružen** je realni broj $e_{i,j} \geq 0$, kojeg možemo interpretirati kao **grešku** najbolje aproksimacije pripadnih točaka iz $S_{i,j}$ — na primjer, pravcem po metodi najmanjih kvadrata. Grešku $e_{i,j}$ dobivamo pozivom funkcije *err* na sljedeći način

$$e_{i,j} = err(i, j, P), \quad i \leq j.$$

Funkciju **ne** treba napisati! Pretpostavljamo da je složenost ovog poziva linearna u broju točaka u segmentu, tj. $\Theta(j - i + 1)$.

Segmenti u nekoj **particiji** od P moraju sadržavati sve točke iz P i ne smiju se sjeći. Za bilo koju particiju od P, **kaznena** funkcija te particije definira se kao **zbroj** grešaka po svim segmentima u toj particiji i tom zbroju još dodamo "kazneni" član = **broj** segmenata u particiji, pomnožen s unaprijed zadanom realnom konstantom $C > 0$. Broj C je kazna za "lomljenje" jednog komada pravca u dva komada (bez te kazne, segmentima duljine najviše 2, dobivamo ukupnu grešku nula). Sastavite algoritme koji nalaze:

(a) **najmanju** vrijednost kaznene funkcije po svim particijama od P,

(b) neku particiju koja daje najmanju vrijednost kaznene funkcije (dovoljno je vratiti broj segmenata u toj particiji i indekse početnih točaka tih segmenata, uzlazno ili silazno sortirano).

Složenost ovih algoritama **mora** biti polinomna u n. Analizirajte složenost algoritama i pokažite da zadovoljavaju ovaj uvjet.

Uputa: Iskoristite dinamičko programiranje, tj. nađite rekurziju za minimalnu vrijednost kaznene funkcije na odgovarajućim potproblemima. U algoritmima iskoristite "memoizaciju" = pamćenje poznatih najmanjih vrijednosti kaznene funkcije za potprobleme.

Rješenje.

Prema uputi, rješenje ide **dinamičkim programiranjem**. Prvo parametriziramo problem, tako da uvijek krećemo od **prve** točke P_1, idemo redom po zadanim točkama, do točke P_j, tj. imamo **varijabilni kraj**. Za $1 \leq j \leq n$, definiramo

- $\mathrm{opt}(j)$ = minimum kaznene funkcije na potproblemu za točke $P_1, \ldots, P_j$, tj. zadnji segment u particiji završava u točki P_j.

Ako je zadnji segment u **optimalnoj** particiji za potproblem do P_j jednak $S_{i,j}$ (pokriva točke od P_i do P_j), onda je (princip optimalnosti)

$$\text{opt}(j) = e_{i,j} + C + \text{opt}(i-1),$$

tj. morao sam optimalno stići do P_{i-1}, inače ne može biti optimalno do točke P_j (jer je iza P_{i-1}). Inicijalizacija je $\text{opt}(0) = 0$.

Sad **minimiziram** gornji izraz po svim mogućim vrijednostima za i, tj. po $i = 1, \ldots, j$. Onaj indeks i na kojem se minimum **dostiže** $\Longleftrightarrow$ zadnji segment mora biti $S_{i,j}$.

(a) Dalje je očito — petlja po j od 1 do n, uz pamćenje optimalne vrijednosti (memoizacija za potprobleme), tj. redom računamo tablicu vrijednosti $\text{opt}(j)$, za sve j. Rezultat je $\text{opt}(n)$. Složenost je sigurno u $O(n^3)$.

(b) Ovdje treba, za svaki j, pamtiti i pripadni indeks $i = i_j$, na kojem se dostiže minimum za $\text{opt}(j)$.