Opisi algoritama

Primjeri implementiranih rješenja dani su u priloženim izvornim kodovima koji nužno ne odgovaraju u svim detaljima ovdje opisanim algoritmima.

Zadatak: Rang (P1 i P2)

autor: Adrian Satja Kurdija
pripremili: Adrian Satja Kurdija i Vedran Kurdija

Natjecatelj koji je rekao da je k-ti sigurno govori istinu ako i samo postoji točno $n - k$ natjecatelja koji su sigurno zauzeli lošija mjesta, točnije, čije su izjave veće od k.

Naime, ako vrijedi navedeni uvjet, onda je svih $n - k$ mjesta nakon k-tog mjesta sigurno zauzeto pa promatrani natjecatelj ne može biti lošiji od k-tog mjesta. Obrnuto, ako navedeni uvjet ne vrijedi (broj natjecatelja čije su izjave veće od k manji je od $n - k$), onda barem jedno mjesto nakon k-tog nije zauzeto izjavama ostalih natjecatelja pa je promatrani natjecatelj mogao zauzeti to mjesto, tj. lagati.

Ovu ideju moguće je implementirati u linearnoj složenosti tako da najprije za svako mjesto zapišemo koji su natjecatelji izjavili to mjesto; potom prolazimo unatrag po mjestima ($k = n, n - 1, \ldots, 2, 1$) te pribrajamo natjecatelje, tj. pratimo ukupan broj natjecatelja koji su zauzeli lošija mjesta od trenutačnog mjesta k. Ako je taj broj jednak $n - k$, natjecatelj na k-tom mjestu govori istinu.

Postoji i alternativno rješenje (u prilogu) koje kreira listu od 1. do n-tog mjesta tako da prati skup natjecatelja kojima još nije dodijeljeno mjesto te jednog od njih stavlja na trenutno mjesto. Ako to može biti samo jedan natjecatelj – i to onaj koji je izjavio da je osvojio trenutno mjesto – onda taj natjecatelj sigurno govori istinu.

Zadatak: Lektira (P1)

autor: Pavel Kliska
pripremili: Pavel Kliska, Vedran Kurdija i Adrian Satja Kurdija

Rješenje je isprobati sve mogućnosti za broj $k = 1, 2, 3, \ldots$. Za određeni k, koristeći postotke p_i računamo donju i gornju granicu za broj pročitanih stranica s_i. Konkretno, mora vrijediti

$$\left\lfloor k \cdot \frac{p_i - 1}{100} \right\rfloor + 1 \leq s_i \leq \left\lfloor k \cdot \frac{p_i}{100} \right\rfloor.$$

Ako je zbroj donjih granica manji ili jednak k, a zbroj gornjih veći ili jednak k, onda je odgovarajući k rješenje, a konkretne s_i računamo tako da ih inicijalno postavimo na donje granice i povećavamo dok im zbroj ne postane jednak k.

Moguće je dokazati da, ako rješenje postoji, onda je i broj $k = 100n$ rješenje. To znači da ako nijedan $k \leq 100n$ ne prođe gore opisanu provjeru, zaključujemo da nema rješenja.

Zadatak: Tenis (P1 i P2)

autor: Adrian Beker
pripremili: Josip Klepec, Vedran Kurdija i Adrian Beker

Ako je neki igrač dobio k setova, onda je pobijedio u $\lfloor k/2 \rfloor$ mečeva. Iz te zamjedbe zaključujemo koji su igrači sudjelovali u kojem kolu. Također, ako je k neparan, onda je igrač u svom posljednjem (gubitničkom) meču dobio set, a inače nije.

Mečeve i njihove rezultate određujemo redom od finala, polufinala i tako dalje prema prvom kolu. Za promatrano kolo znamo koji igrači su pobjednici, a koji gubitnici, te za svakog gubitnika znamo je li u tom

kolu dobio set. Pobjednike i gubitnike u promatranom kolu kolu sparujemo na sljedeći način: pobjednike sortiramo po ukupnom broju izgubljenih setova u kolima koje još nismo riješili, te ih redom od onih s najviše izgubljenih setova pridružujemo gubitnicima koji su u promatranom kolu dobili set. (Budući da svi pobjednici igraju isti broj mečeva u kolima koja još nismo riješili, njihove izgubljene mečeve "trošimo" od onog koji ih ima najviše.) Preostale gubitnike, koji u promatranom nisu dobili set, pridružimo preostalim pobjednicima proizvoljno. Tijekom opisanog algoritma treba naravno paziti da, u slučaju da naiđemo na nekonzistencije u danim podacima (npr. nemamo odgovarajući broj igrača u nekom kolu ili ne postoji igrač koji je dobio $2N$ setova), javimo da rješenje ne postoji.

Spomenimo još da je zadatak moguće riješiti i svođenjem na problem najvećeg protoka kroz graf (engl. *max flow*), no u tom slučaju treba koristiti dovoljno efikasan algoritam za rješenje tog problema.

Zadatak: Brodovi (P2)

autor: Adrian Beker
pripremili: Adrian Beker i Adrian Satja Kurdija

Ponude kompanije 1 nazovimo bijelima, a ponude kompanije 2 crnima.

U prvim dvama podzadatcima možemo unaprijed postaviti pitanja za sve parove gradova. Za prvi podzadatak dovoljno je rekurzivno isprobati sve mogućnosti. Kako složenost ne bi bila prevelika, pritom treba paziti da se u rekurziji ne nastavljamo granati kada se trenutni skup odabranih linija ne može proširiti do skupa sa željenim svojstvom. Drugi podzadatak rješavamo dinamičkim programiranjem: za svaka dva grada a i b te svaku boju c imamo dinamiku koja pamti možemo li naći skup linija sa željenim svojstvom ako se ograničimo na gradove između a i b (u smjeru kazaljke na satu) te linije boje c. Prijelaz radimo tako da na sve moguće načine linijom boje c spojimo neki od gradova a i b s nekim gradom na luku između a i b (te na taj način svedemo problem na dva manja ekvivalentna potproblema). Ovaj pristup ima složenost $O(N^3)$.

U trećem i četvrtom podzadatku postupamo na sljedeći način. Postavimo upite za sve rubne bridove $k \leftrightarrow k+1$. Ako su svi ti bridovi iste boje, imamo rješenje. Inače, postoji vrh k takav da su bridovi koji ga povezuju s njegovim susjedima na kružnici raznobojni: primjerice, brid do lijevog susjeda je bijeli, a do desnog crni. Promatrajmo skup od preostalih $N-1$ vrhova i rekurzivno riješimo problem za taj skup. Ako smo dobili bijelo rješenje, vrh k spajamo s lijevim susjedom, a ako smo dobili crno rješenje, vrh k spajamo s desnim susjedom i tako dobivamo rješenje za cijeli skup od N vrhova.

Da bi opisano rješenje postavilo najviše $2N$ pitanja, nakon "izbacivanja" vrha k (da bismo sveli problem na manji) ne smijemo ponovno postaviti pitanja za sve rubne bridove. Primijetimo da dva rubna brida nestaju (oni iz vrha k), a jedan se pojavljuje – treba postaviti upit za izravan brid između lijevog i desnog susjeda vrha k. Treba nam struktura podataka koja će pamtiti kružni niz boja, dovoljno brzo dvije susjedne zamijeniti jednom novom, te dovoljno brzo pronaći dvije susjedne različite boje. U tu svrhu možemo koristiti dvostruko vezanu listu, gdje za svaki vrh pamtimo trenutačne susjede lijevo i desno te trenutačne boje lijevo i desno. Dodatno održavamo skup "raznobojnih" vrhova u koji ubacujemo/izbacujemo vrhove za koje se situacija mijenja.

Ukupna je složenost rješenja $O(N \log N)$ ili $O(N)$, ovisno o implementaciji.