Opisi algoritama

Autori zadataka, testnih primjera i rješenja:

Marin Kišić

Adrian Satja Kurdija

Vedran Kurdija

Ivan Paljak

Stjepan Požgaj

Tonko Sabolčec

Paula Vidas

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

Zadatak: Ocjena

Za 50% bodova bilo je dovoljno na sve moguće načine izabrati broj pojavljivanja svake pojedine ocjene u promijenjenom nizu, tj. nizu čija zaključna ocjena odgovara ocjeni O. Među svim opcijama odabiremo onu za čije je postizanje potrebno najmanje promjena u odnosu na niz koji je Mirko prenio. Implementacijski za ovo možemo koristiti četiri ugniježđene for petlje kojima fiksiramo broj pojavljivanja svake od ocjena iz intervala [2, 5]. Primijetimo da nam nije potrebna dodatna for petlja za ocjenu 1, budući da je njezin broj pojavljivanja jednoznačno određen kao $n-$[suma broja pojavljivanja svih ostalih ocjena]. Jasno je da odbacujemo one kombinacije kod kojih bi zaključna ocjena bila različita od O. Sada nam ostaje izračun najmanjeg broja promjena nad nizom koji je Mirko prenio potreban za postići konačni, promijenjeni niz koji smo fiksirali for petljama. Izračun možemo provesti na način da prethodno pohranimo broj pojavljivanja svake od ocjena u nizu koji je Mirko prenio te ga usporedimo sa fiksiranim vrijednostima iz for petlji. Ukupna složenost je $\mathcal{O}(n^4)$.

Za veći broj bodova valjalo je primijetiti da se umjesto isprobavanja svih mogućnosti možemo poslužiti pohlepnim algoritmom. Ako je Mirkova stvarna zaključna ocjena O veća od zaključne ocjene na koju ukazuje niz koji je prenio, intuitivno je da za postizanje najmanjeg broja promjena, ocjene koje je prenio ima smisla samo povećavati. Vrijedi i obrnuto, ako je ocjena O manja od zaključne ocjene prenesenog niza, jasno je da će se najmanji broj promjena postići isključivo umanjivanjem prenesenih ocjena. Postavlja se pitanje kako znati koje ocjene povećavati ili umanjivati. Ovdje se ponovno služimo intuitivnom pohlepnom metodom. Ako ocjene valja povećavati, uvijek ćemo izabrati najmanju ocjenu i povećati je što više možemo (a da time ne premašimo konačni prosjek, tj. zaključnu ocjenu O). Vrijedi, naravno, i obrnuti slučaj, ako ocjene treba umanjivati, uvijek ćemo umanjiti najveću ocjenu što više možemo (pazeći da zaključnom ocjenom ne padnemo ispod konačne ocjene O).

Implementacijski je ovdje za 70% bodova bilo dovoljno u prenesenom nizu for petljom pronalaziti najveću ili najmanju ocjenu (ovisno o slučaju) te ju povećavati ili smanjivati (također ovisno o slučaju), sve dok ne postignemo željenu konačnu ocjenu. Pritom trenutačni prosjek efikasno računamo prateći zbroj ocjena u pomoćnoj varijabli. Pronalazak i promjenu ocjene u najgorem slučaju možemo ponoviti n puta (ako treba promijeniti sve ocjene), a složenost pronalaska najveće ili najmanje ocjene u nizu for petljom jest $\mathcal{O}(n)$, tako da je ukupna složenost $\mathcal{O}(n^2)$.

Za sve bodove trebalo je uočiti da se algoritam korišten u rješenju za 70% bodova može ubrzati tako da najprije sortiramo preneseni niz. Nakon toga možemo ići redom od kraja ili početka (ovisno treba li ocjene smanjivati ili povećavati) i mijenjati ocjene dok ne dostignemo željenu zaključnu ocjenu O. Ukupna složenost je tada $\mathcal{O}(n \log n)$.

Zadatak: Palindromi

Za osvajanje 20% bodova bilo je dovoljno provjeriti mogu li se slova ulazne riječi permutirati na način da ona postane palindrom. Budući da su palindromi simetrični, svakom slovu iz prve polovice palindroma odgovara neko slovo iz druge polovice palindroma. Izuzetak je, dakako, slovo koje se nalazi u sredini palindroma u slučaju da se palindrom sastoji od neparnog broja znakova. Dakle, palindrom je moguće konstruirati ako se najviše jedno slovo u ulaznoj riječi pojavljuje neparni broj puta. Konstrukcija tog palindroma prirodno slijedi iz opisane zamjedbe pa ju ostavljamo čitatelju za vježbu.

Dodatnih 20% bodova mogli ste osvojiti fiksiranjem rješenja te isprobavanjem svih mogućih načina da se to rješenje postigne. Primjerice, ako ste fiksirali da će u rješenju biti k palindroma, tada je bilo potrebno na sve moguće načine raspodijeliti n ulaznih slova u točno k riječi. Za svaku riječ možemo tada iskoristiti algoritam opisan u prethodnom odlomku kako bismo provjerili je li tu riječ moguće pretvoriti u palindrom. Ovakvu iscrpnu pretragu moguće je implementirati u vremenskoj složenosti $\mathcal{O}(n^n)$.

Za osvajanje svih bodova bilo je potrebno dodatno nadograditi zamjedbu iz prvog odlomka. Promatrajmo broj pojavljivanja svakog slova u ulaznoj riječi. Neka je $k > 0$ broj slova koja se u ulaznoj riječi pojavljuju neparan broj puta. Primijetimo da tada moramo konstruirati barem k palindroma. U protivnom će se u nekom palindromu nalaziti dva različita slova koja se u tom istom palindromu pojavljuju neparan broj puta, što je u kontradikciji sa zamjedbom iz prvog odlomka. Jednostavnom konstrukcijom možemo pokazati da je uvijek moguće napraviti točno k palindroma. Naime, napravimo k jednoslovnih riječi od slova koja se pojavljuju neparan broj puta. Broj svakog od preostalih slova sada je paran pa sva ta slova možemo jednostavno dodati bilo kojoj od k jednoslovnih riječi tako da ona ostane palindrom. Ostaje nam još trivijalan slučaj kada je $k = 0$ koji smo već riješili zamjedbom iz prvog odlomka.

Vremenska složenost cijelog rješenja je $\mathcal{O}(n)$.

Zadatak: Taktika

Na početku simulacijom odigramo prvih m poteza i odredimo je li netko već pobjedio. Sada ostaje pitanje može li netko pobjediti i ako može, tko i u koliko najmanje poteza?

Pretpostavimo da gledamo može li Mirko pobjediti, a analogno ćemo poslije rješiti za Slavka. Primijetimo da postoji samo $2(n + 1)$ pobjedničkih linija (n stupaca, n redaka i 2 dijagonale). Fiksirajmo neku pobjedničku liniju i pretpostavimo da će Mirko pobjediti baš na njoj. Ako je na toj liniji Slavko već postavio svoj znak, znamo da Mirko na toj liniji ne može pobjediti te samo preskočimo tu liniju. Inače, neka je x broj znakova koje je Mirko već postavio na tu liniju. Sada znamo da će Mirko odigrati još točno $n - x$ poteza. Također, budući da znamo tko je trenutačno na potezu i da će Mirko odigrati još $n - x$ poteza, lako određujemo i koliko će poteza Slavko odigrati, neka to označava broj y. Sada smo problem sveli na sljedeći: je li moguće postaviti y Slavkovih znakova tako da on ne pobijedi? To možemo riješiti isprobavanjem svih mogućnosti za postavljanje y Slavkovih znakova na preostala polja ploče, npr. s pomoću rekurzije. Ako je te znakove moguće postaviti tako da Slavko ne pobijedi, Mirko može pobijediti nakon $(n - x) + y$ poteza. Ponavljanjem ove provjere za svaku od mogućih pobjedničkih linija dobivamo minimalan broj poteza za Mirkovu pobjedu.

Vremenska složenost ovog rješenja ograničena je s $\mathcal{O}\left(n^3\binom{n^2}{n}\right)$, iako se teorijski maksimum nikad neće postići pa je ona zapravo i manja.

Zadatak: Tisak

Potrebno je simulirati postupak koji je opisan u zadatku.

S obzirom da broj riječi u odlomku nije unaprijed poznat, u jeziku C++ učitavanje jednog retka ulaza možemo napraviti pomoću funkcije `getline`.

Zadatak: Banka

Za 30% bodova bilo je dovoljno u složenosti $\mathcal{O}(2^n)$ isprobati sve moguće odabire gradova u kojima otvaramo banke, izračunati odgovarajuće gužve te ispisati najmanju.

Za veći broj bodova potrebno je razmišljati na sljedeći način. Otvorimo banke u svih n gradova i izračunajmo dobivenu gužvu g. Pretpostavimo da je ta gužva nastala u gradu A. Očito je g jedan kandidat za rješenje. Htjeli bismo pronaći rješenje manje od g. Primijetimo da je točna sljedeća tvrdnja: *Ako postoji rješenje s gužvom manjom od g, ono sigurno ne sadrži grad A.* Ta tvrdnja vrijedi jer će bilo koji odabir koji uključuje grad A imati u tom gradu gužvu veću ili jednaku g.

Iz gornje tvrdnje slijedi: izbacivanjem grada A (tj. zatvaranjem banke u njemu) ne gubimo nijedno potencijalno rješenje s gužvom manjom od g (ako ono postoji), dakle, možemo iz odabira izbaciti grad A. Za preostali skup od $n-1$ gradova ponovimo isti postupak računanja gužve i izbacivanja najfrekventnijeg grada. Tako postupno smanjujemo broj gradova u odabiru i pamtimo najmanju otkrivenu gužvu.

Ovaj se postupak ponavlja n puta. Ako opterećenja pojedinih gradova u svakom "krugu" računamo naivno, u složenosti $\mathcal{O}(nk)$, ukupna je složenost $\mathcal{O}(n^2k)$, što je dovoljno za 70% bodova. Za sve bodove potrebno je uočiti da opterećenja gradova možemo efikasno ažurirati, ako pamtimo pozicije prve otvorene banke za svakog trgovca i samo ih povećavamo do sljedeće banke kad se neka zatvori. Ukupno ćemo te indekse povećati najviše nk puta. Najveće opterećenje u svakom od n krugova možemo računati prolaskom po (ažuriranom) nizu opterećenja u $O(n)$. Time dobivamo ukupnu složenost od $\mathcal{O}(nk+n^2)$ što je dovoljno za sve bodove.

Zadatak: Integram

Označimo kategorije s $c_1, c_2, ..., c_k$, a pripadajuće elemente u kategoriji c_i s $v_{i,1}, v_{i,2}, ..., v_{i,n}$.

Prvo ćemo opisati rješenje dovoljno za dobivanje svih bodova za slučaj $k = 2$. Za svaki par vrijednosti iz različitih kategorija definiramo *povezanost* koja može poprimiti jednu od sljedećih vrijednosti: `NEPOZNATO`, `POVEZANO` ili `NEPOVEZANO`. Na početku su sve vrijednosti postavljene na one koje odgovaraju ulaznim spoznajama. Vrijednosti ćemo postupno ažurirati prolazeći po svim elementima te koristeći sljedeća pravila:

- Ako je element povezan s nekim elementom iz suprotne kategorije, onda za sve ostale elemente iz suprotne kategorije postavimo povezanost na `NEPOVEZANO`.
- Ako postoji $n-1$ elemenata (tj. svi osim jednog) iz druge kategorije za koji povezanost s trenutnim elementom iznosi `NEPOVEZANO`, tada postavimo povezanost s preostalim elementom na `POVEZANO`.

Ovaj postupak ponavljamo dokle god ima promjena, a naposljetku na upite odgovaramo provjerom povezanosti dva elementa.

Za slučaj $n = 2$ izgradit ćemo graf čiji čvorovi odgovaraju elementima $v_{i,j}$. Za zadanu spoznaju oblika $v_{i,j}$ `je` $v_{k,l}$ dodat ćemo brid između čvorova $v_{i,j}$ i $v_{k,l}$ te između čvorova $v_{i,\overline{j}}$ i $v_{k,\overline{l}}$ (pri čemu je $\overline{1}$ oznaka za 2, a $\overline{2}$ oznaka za 1). Ako pak je spoznaja oblika $v_{i,j}$ `nije` $v_{k,l}$, tada ćemo dodati brid između čvorova $v_{i,j}$ i $v_{k,\overline{l}}$ te između $v_{i,\overline{j}}$ i $v_{k,l}$. Prilikom odgovaranja na upite dovoljno je provjeriti nalazi li se koji element iz tražene kategoriji u istoj komponenti grafa kao i zadani element. U slučaju da element nije pronađen, ispisujemo `-1`.

Konačno rješenje za sve bodove svodi se na isprobavanje svih mogućnosti, tj. svih mogućih spajanja elemenata. Za početak ćemo odrediti koliko različitih mogućnosti postoji. Za $k = 2$ postoji $n!$ (n faktorijela) načina na koji možemo upariti elemente dvije kategorije. Npr. možemo fiksirati poredak u prvoj kategoriji te za svaku moguću permutaciju elemenata druge kategorije povezati elemente na istim pozicijama. Dodavanjem još jedne kategorije opet je potrebno isprobati sve permutacije elemenata sa svim kombinacija prije dodavanja, pa je ukupan broj kombinacija jednak $n!^{k-1}$. Sljedeća tablica prikazuje

jednu moguću kombinaciju za $k = 3$ i $n = 4$, pri čemu smatramo da su svi elementi u istom stupcu međusobno povezani.

$$\begin{array}{cccc} v_{1,1} & v_{1,2} & v_{1,3} & v_{1,4} \\ v_{2,3} & v_{2,4} & v_{2,2} & v_{2,1} \\ v_{3,2} & v_{3,1} & v_{3,4} & v_{3,3} \end{array}$$

Generiranje svih mogućih kombinacija moguće je implementirati pomoću rekurzije i korištenja ugrađenih funkcija za generiranje permutacija kao što je `std::next_permutation` u C++ (za detalje možete pogledati priloženi programski kod). Sljedeći korak je provjera ispravnosti neke mogućnosti, tj. zadovoljava li neka mogućnost zadani skup spoznaja. To je moguće provjeriti tako da za svaku spoznaju oblika $v_{i,j}$ `je` $v_{k,l}$ provjerimo nalaze li se elementi u istom stupcu, odnosno za spoznaju oblika $v_{i,j}$ `nije` $v_{k,l}$ jesu li oni u različitim stupcima. Tako ćemo filtrirati sve zadovoljavajuće kombinacije. Za svaki element $v_{i,j}$ i kategoriju c_k ($k \neq i$) izgradit ćemo listu mogućih vrijednosti iz kategorije c_k koje se mogu povezati s elementom $v_{i,j}$ a da pritom ne narušimo konzistentnost sa zadanim spoznajama. To ćemo napraviti tako da za svaki element u svakoj zadovoljavajućoj kombinaciji provjerimo s kojim je elementima iz drugih kategorija povezan. Prilikom odgovaranja na upite, ako postoji točno jedan mogući element, onda ispisujemo oznaku tog elementa, inače ispisujemo `-1`.