Državno natjecanje iz informatike

Srednja škola

Prva podskupina (1. i 2. razred)

20. i 21. ožujka 2019.

OPISI ALGORITAMA

Autori zadataka: Domagoj Bradač, Mislav Bradač, Adrian Satja Kurdija, Tonko Sabolčec

Eventualna pitanja, mišljenja ili prijedloge vezane uz zadatke uputite na adrian.kurdija@fer.hr.

Ministarstvo znanosti i obrazovanja

Agencija za odgoj i obrazovanje

HRVATSKI SAVEZ INFORMATIČARA

Ako je $N \leq 20$, možemo isprobati svih 2^N mogućnosti za dobitnike svih poena te za svaku od tih mogućnosti (npr. Mirko, protivnik, Mirko, Mirko, protivnik) provjeriti rezultira li zadanim nizom servisnih polja. Sve mogućnosti možemo generirati rekurzivno ili (jednostavnije) kao binarne brojeve, tzv. bitmaske. Ovo rješenje nosilo je 50% bodova.

Ako nema upitnika, ispravan je sljedeći pohlepni algoritam: ako Mirko može osvojiti trenutačni poen tako da zadano servisno polje u idućem poenu bude odgovarajuće, onda Mirko osvaja poen; inače ga osvaja protivnik. Ako zadano servisno polje u idućem poenu ni u jednom slučaju nije odgovarajuće, ispisujemo -1. Ovo rješenje nosilo je 50% bodova, a u kombinaciji s prethodnim rješenjem nosilo je 70% bodova.

Budući da upitnika ima najviše 15, možemo isprobati sve mogućnosti za slova L i R (servisna polja) koja odgovaraju tim upitnicima. Mogućnosti generiramo rekurzivno ili koristeći binarne brojeve, tzv. bitmaske. Za svaku od tih mogućnosti provodimo gornji pohlepni algoritam te pamtimo maksimalan broj bodova koji je Mirko ostvario u nekom od tih scenarija, što daje potpuno rješenje zadatka.

Alternativno, moguć je pohlepni algoritam koji odabire i upitnike, uz nešto složeniju provjeru tko može osvojiti trenutačni poen. Razradu ovog rješenja ostavljamo čitateljici za vježbu.

U testnim primjerima vrijednim 50% bodova vrijedilo je R, S ≤ 300. Te primjere bilo je moguće riješiti tako da za svaku dijagonalu izvedemo idući postupak:

- označimo polja na dijagonali i njima susjedna kao plodna
- s jednog polja dijagonale pokrenemo DFS ili BFS obilazak koji broji do koliko je plodnih polja moguće doći iz početnog
- ako je vrijednost dobivena obilaskom veća od do sad zapamćene vrijednosti, tada ćemo nju zapamtiti, a staru zaboraviti

Ostale testne primjere bilo je moguće riješiti tako da preprocesiramo određene podatke čime ćemo za svaku dijagonalu smanjiti broj operacija koje moramo napraviti kako bismo izbrojili na koliko polja će izrasti biljka ako nju odabere Željko. Prije odabira dijagonala možemo pronaći komponente plodnih kvadratića te za svaku komponentu zapamtiti kolika je njezina površina. Također ćemo za svaku plodnu ćeliju zapamtiti redni broj komponente u kojoj se nalazi. Komponente je moguće pronaći koristeći već spomenuti DFS ili BFS obilazak.

Sada za svaku dijagonalu možemo izračunati broj izraslih biljaka tako što ćemo proći po svim ćelijama na dijagonali i njima susjednim ćelijama. Ako je ćelija neplodna, ona postaje plodna i moramo povećati broj biljaka za 1. Ako je ćelija plodna te pripada komponenti koju još nismo pribrojili, na broj biljaka dodajemo veličinu pripadne komponente te pamtimo da smo tu komponentu pribrojili.

Opisano rješenje je složenosti $O(N^2)$.

Primijetimo da, ako neka osoba B ima više manipulatora nad sobom, tada svaka poruka koja stiže do B prolazi svim njegovim manipulatorima, redom od najviše do najniže rangiranog. Zadnjeg od tih manipulatora nazovimo "neposrednim manipulatorom" od B.

Ako među svim osobama nacrtamo samo veze neposredne manipulacije, dobit ćemo stablo. To vrijedi zato što svatko (osim direktora, koji je korijen stabla) ima barem jednog manipulatora, pa stoga i točno jednog neposrednog manipulatora, roditelja u tom stablu. Cilj nam je izgraditi to stablo, nakon čega će traženi broj osoba kojima netko manipulira biti veličina njegovog podstabla.

Stablo manipulacije gradimo dodavajući u njega jednu po jednu osobu (1, 2, 3, …). Kada osobu br. B dodajemo u stablo, njegovog roditelja (neposrednog manipulatora) tražimo tako da uzmemo sve osobe koje njemu mogu poslati poruku i pogledamo gdje se one nalaze u dosad izgrađenom stablu. Ako je samo jedna takva, to je traženi manipulator. Inače, manipulator od B mora manipulirati svim navedenim osobama. Stoga je neposredni manipulator od B prvi zajednički predak navedenih osoba u stablu manipulacije, kojega možemo efikasno pronaći standardnim LCA algoritmom.

Zadatak možemo riješiti korištenjem C++ strukture *set*, u koju na početku stavimo zadane pozicije mrvica, a mrvice najbliže zadanoj poziciji mrava pronalazimo korištenjem funkcije *lower_bound* (ili *upper_bound*) te ih brišemo iz seta, sve u složenosti O(log N).

Rješenje bez uporabe seta, koje je moguće implementirati i npr. u Pythonu, binarno pretražuje niz pozicija hrane da bi pronašao one najbliže mravu s lijeve i desne strane. Putovanje mrava tada se simulira jednostavnim pomacima pronađenih dviju kazaljki u nizu, lijevo i desno. "Brisanje" mrvica ostvarujemo tako da nakon svakog putovanja ažuriramo interval mrvica koji je još uvijek nepojeden, sljedeće binarno pretraživanje radimo samo unutar tog intervala, a izlaskom iz tog intervala odmah detektiramo završetak putovanja.

Optimalan broj brodova možemo pronaći sljedećim pohlepnim algoritmom: u prvi brod stavimo najdulji mogući dobar prefiks životinja tako da su sve međusobno različite. U drugi brod stavimo najveći mogući interval životinja počevši od prve životinje koju nismo stavili u prvi brod, i tako dalje dok ne stavimo sve životinje u brodove. Ovo je jednostavno napraviti u linearnom vremenu. Ponavljajući ovaj algoritam za svaki upit, dobit ćemo algoritam ukupne složenosti O(NQ) kojim je moguće ostvariti 30% bodova.

Nazovimo intervale koji odgovaraju pojedinim brodovima u prethodno opisanom algoritmu prije promijene ijednog broja *inicijalnim intervalima*, a intervale koje bi dobili algoritmom primijenjenim na promijenjeni niz *pohlepnim intervalima*. Kako bismo izbjegli linearnu složenost po upitu, potrebno je ubrzati ovaj pohlepni algoritam. Ideja je sljedeća: neka P predstavlja poziciju upita. Tada će pohlepni intervali biti jednaki inicijalnim intervalima do točke P. Izračunat ćemo završetak pohlepnog intervala koji sadrži P, a preostali dio niza možemo *preprocessati* s obzirom da na tome dijelu niza nije napravljena promjena.

Traženi *preprocess* je oblika: koliko minimalno treba brodova Noi ako ne postoji prvih *i-1* životinja? Ovo je moguće izračunati dinamičkim programiranjem u linearnoj složenosti. Označimo tu vrijednost s *DP*(i).

Ovisno o složenosti u kojoj naš algoritam izračuna početak i kraj pohlepnog intervala koji sadrži P, dobit ćemo rješenja vrijedna 60% ili 100% bodova.

Za 60% bodova, primijetimo da intervali ne mogu biti veći od M, što je u drugom podzadatku najviše 100. Linearnim prolaskom od početka inicijalnog intervala koji sadrži P možemo pronaći kraja pohlepnog intervala koji sadrži P, te koristeći *preprocessane* informacije dobiti algoritam ukupne složenosti O(N + min(m, N) * Q)

Koristeći dinamičko programiranje u linearnoj složenosti također možemo izračunati sljedeće dvije informacije: kolika je duljina najvećeg intervala početnog niza koja počinje na danoj poziciji *i* koja sadrži samo različite brojeve. Tu vrijednost označimo s *F(i)*. Druga informacija koja nam je potrebna je kolika je duljina najvećeg intervala početnog niza koja počinje na *i* te sadrži najviše jedan par istih brojeva, pri čemu ne smije naravno sadržavati 3 ista broja. Ovu vrijednost označavat ćemo s *G(i)*.

Neka Z kao i u zadatku označava novi broj na poziciji P, W staru vrijednost na poziciji P te X neka označava broj na početku incijalnog intervala neposredno nakon inicijalnog intervala koji sadrži P.

Analiza svih slučajeva je delikatna, no rutinska. Ispostavlja se da promatrajući je li W = X, pronalaskom brojeva s vrijednošću Z koji su najbliže poziciji P (što je moguće učiniti binarnim pretraživanjem u složenosti O(log N)), te korištenjem funkcija *DP*, *F* i *G* moguće odgovoriti na svaki upit u složenosti O(log N) što daje algoritam ukupne složenosti O(N + Q log N).

Mnogi implementacijski detalji su izostavljeni te vas potičemo da ih sami osmislite, a za bolje razumijevanje možete pogledati i službeno rješenje.

Primijetimo da je zadatak ekvivalentan sljedećemu: potrebno je odabrati neparan broj K i K permutacija brojeva od 1 do N tako da za M zadanih uređenih parova (A, B) vrijedi da se A pojavljuje nakon B u više od pola permutacija. Pritom k-ta permutacija predstavlja relativan poredak snaga k-tog najjačeg igrača iz timova.

Za dani niz permutacija, neka *f(A, B)* označava razliku broja permutacija u kojima se A pojavljuje prije B i broja permutacija u kojima se B pojavljuje prije A. Ideja je sljedeća: želimo pronaći mali broj permutacija kojima ćemo promijeniti vrijednost *f(A, B)* za neke parove (A_1, B_1), (A_2, B_2), … (A_s, B_s) na željeni način, a da se pritom ne promijeni vrijednost funkcije f za ostale parove. Koliko smo uspješni da s malim brojem permutacija "riješimo" velik broj parova, ovisit će o broju bodova koji algoritam nosi. Vidjet ćemo pritom da će naši algoritmi napraviti 200 permutacija, tj. 200 igrača, ali da će *f(A, B)* biti barem 2 za tražene parove. Tada ćemo izbacivanjem proizvoljne permutacije dobiti traženo rješenje.

Kako bismo opisali algoritam do kraja, uvedimo sljedeću notaciju: za dani skup A neka R(A) označava niz brojeva iz A u rastućem poretku, a P(A) neka označava niz brojeva iz A u padajućem poretku. Također, prisjetimo se notacije za skupovnu razliku: (A / B) označava skupovnu razliku skupova A i B, tj. skup koji sadrži sve brojeve koji se nalaze u skupu A, a ne nalaze se u skupu B. Također označimo s S skup svih brojeva od 1 do N.

Opišimo rješenje koje pravi dvije permutacije po svakom zadanom paru, što će izbacivanjem jedne davati rješenje vrijedno 30% bodova. Za par (A, B) dodat ćemo permutacije:

(R(S \ {A, B}), A, B)

(A, B, P(S \ {A, B})).

Primjerice, za N = 5 te A = 2, B = 5, dane permutacije bile bi:

(1, 3, 4, 2, 5)

(2, 5, 4, 3, 1)

Primijetite da se pritom *f(A, B)* povećao za 2, a *f* se nije promijenila ni za jedan drugi par čime smo rješenje ako jednostavno napravimo po dvije opisane permutacije za svaki uređeni par iz ulaza.

Za još 30% bodova potrebno je napraviti najviše četiri permutacije za svaki broj od 1 do N.

Neka je odabrani broj x, te neka je A skup svih brojeva tako da je (A, x), uređeni par u ulazu (tj. A je skup onih brojeva a za koje želimo da vrijedi *f(a, x)* > 0), a B = S \ A \ {x}, tj. skup svih ostalih brojeva različitih od x. Ove četiri permutacija dat će nam traženo rješenje:

(P(A), x, P(B))

(R(A), x, R(B))

(x, P(B), P(A))

(R(B), R(A), x)

Za 100% bodova, podijelit ćemo sve neuređene parove brojeva u najviše 100 skupina tako da se svaki broj pojavljuje u najviše jednom paru u skupini. Za početak možemo pretpostaviti da je N = 100, jer u suprotnom ćemo izgraditi "veći" turnir od 100 ekipa pri čemu možemo jednostavno ne ispisati sve njih.

Traženu podjelu parova na skupine naći ćemo na sljedeći način: promatrajmo potpuni neusmjereni graf od 101 čvorova. Podijelit ćemo njegove bridove na 50 Hamiltonovih ciklusa tako da se svaki brid nalazi u točno jednom Hamiltonovom ciklusu. Neka su vrhovi čvorova označeni brojevima od 0 do 100.

Ovaj problem moguće je riješiti u većoj općenitosti (za više detalja pogledajte zadatak Hyperloop). S obzirom da je 101 prost broj, postoji i jednostavnije rješenje. Naime sljedeći niz Hamiltonovih ciklusa zadovoljava teražene uvjete:

(0, k mod N, 2k mod N, 3k mod N, … (N-1)k mod N) za sve k od 1 do 50.

Uzimajući svaki drugi brid te ignorirajući pomoćni čvor s indeksom 0, dobit ćemo dvije skupine bridova za svaki Hamiltonov ciklus i svih 100 skupina zadovoljavat će traženi uvjet.

Preostaje za dani skup uređenih parova (A_1, B_1), (A_2, B_2), … (A_s, B_s), pri čemu se nijedan broj ne pojavljuje dvaput, konstruirati dvije permutacije koje će promijeniti samo *f(A, B)* za dane parove. Neka je T skup svih brojeva koji nisu ni u jednom paru. Tražene su permutacije:

(A_1, B_1, A_2, B_2, … A_s, B_s, R(T))

(P(T), A_s, B_s, … A_2, B_2, A_1, B_1).

Zadatak je moguće riješiti i na druge (jednostavnije) načine. Primjerice, jedan natjecatelj osvojio je 100% bodova induktivnim pristupom.