Državno natjecanje iz informatike

Srednja škola

Prva podskupina (1. i 2. razred)

14. i 15. ožujka 2018.

OPISI ALGORITAMA

Autori zadataka: Adrian Satja Kurdija (*Trek, Infokuhar, Jenga, Pjege*), Mislav Bradač (*Ikebana, Supertetris*)

Zadatke, test podatke i rješenja pripremili: Adrian Satja Kurdija, Mislav Bradač, Tonko Sabolčec, Dominik Gleich, Mislav Balunović

Ministarstvo znanosti i obrazovanja

Agencija za odgoj i obrazovanje

HRVATSKA ZAJEDNICA TEHNIČKE KULTURE

HRVATSKI SAVEZ INFORMATIČARA

Zadatak možemo riješiti nekim od standardnih algoritama za traženje najkraćeg puta u (težinskom) grafu, npr. Dijkstrinim algoritmom ili Floyd-Warshallovim algoritmom.

Postoji i jednostavnije rješenje koje razmatra nekoliko slučajeva za traženi put, koji će sigurno biti sljedećeg oblika:

1. prelazimo ili ne prelazimo na drugu kružnicu,
2. putujemo po kružnici,
3. prelazimo ili ne prelazimo na drugu kružnicu.

Zadatak je moguće riješiti tako da brute-forceom pronađemo optimalno rješenje za male N-ove te iz dobivenih rezultata zaključimo kako izgraditi rješenje za bilo koji N.

Općenito, moguće je skuhati N - 2 jela čije su veličine (broj sastojaka) redom 2, 3, ..., N - 1, a konstrukcija za traženi N koristi konstrukciju za manji N.

Preciznije, ako trebamo pronaći jela za N sastojaka, pronađemo ih prvo za N - 2 sastojaka. Tih jela za N - 2 sastojaka bit će N - 4, a njihove veličine bit će 2, 3, …, N - 3.

Svim tim jelima dodamo još sastojak N, čime se veličine mijenjanju na 3, 4, …, N - 2.

Potom dodamo još dva jela: {1, 2, 3, ..., N - 1} i {N - 1, N}.

Dobivenih N - 2 jela imaju međusobno različite veličine 2, 3, …, N - 1 i zadovoljavaju traženi uvjet da nijedno jelo nije podskup drugoga.

Naime, jelo {N - 1, N} nije podskup nijednog drugog jela (jer sadrži "novi" sastojak N - 1) niti je jelo {1, 2, 3, ..., N - 1} nadskup nekog drugog jela (jer ne sadrži sastojak N koji je dodan svim drugim jelima). Za ostala jela svojstvo vrijedi iz konstrukcije za manji problem N - 2.

Promatrajmo neki čvor stabla koji ima barem dvoje djece. Kad bismo ga zalili, morali bismo njegovo novo (treće) dijete onda obrisati, što bi bile dvije operacije. Međutim, isti učinak dobili bismo zalijevajući dvoje njegove djece. Možemo stoga pretpostaviti da čvor s barem dvoje djece nikad nećemo izravno zaliti.

Promatrajmo sada čvor stabla koji ima jedno dijete. Kada zalijemo taj čvor, on će imati dvoje djece i dolazimo u situaciju iz prethodnog odlomka. Zaključujemo da se ne isplati dvaput zaliti neki čvor koji nije list. Također, ako smo zalili neki čvor, ne isplati nam se još jednom zaliti nekog njegovog potomka (koji nije list originalnog stabla) jer je on neizravno već jednom zaliven.

Uz pomoć ovih zaključaka zadatak rješavamo dinamičkim programiranjem. Stanje nam je opisano čvorom originalnog stabla, binarnom zastavicom koja kaže je li taj čvor već jednom (neizravno) zaliven, i dubinom koju želimo dobiti u njegovom podstablu. Ta dubina je, zapravo, jednoznačno određena za svaki čvor originalnog stabla. U prijelazu promatramo nekoliko slučajeva.

- Ako je čvor list originalnog stabla, iz njega „od nule“ (ili „od jedinice“ ako je neizravno zaliven) moramo izgraditi potpuno binarno stablo zadane dubine, za što broj operacija možemo dobiti iz pomoćne (jednostavnije) dinamike ili izvodom formule.

- Ako čvor ima jedno originalno dijete i nije zaliven, možemo ga zaliti, pozivajući dinamiku za njegovo zaliveno dijete, kao i za novorođeno dijete iz kojega gradimo „od nule“ jer je list. S druge strane, možemo odrezati originalno dijete te iz čvora izgraditi potpuno binarno stablo „od nule“.

- Inače, pokušavamo trenutačni čvor zaliti (ako nije zaliven) ili ga ne zaliti, u oba slučaja pozivajući dinamiku koja računa koliko je potrebno njegovoj djeci da nadopune potpuno binarno stablo i brišući svu djecu osim ono dvoje kojima je potrebno najmanje.

Najvažnija je zamjedba da pozlaćene pločice imaju prioritet u izvlačenju jer njih potencijalno možemo još koji put izvući iz tornja. Također, u većini slučajeva bolje je izvući dvije rubne pločice nego jednu srednju. Kombinacijom ovih zamjedbi dobivamo jednostavan pohlepni algoritam koji rješava katove (i gradi nove) odozdo prema gore dok god postoji kat iz kojeg je moguće izvući pločicu.

Jedino je dvojbeno pitanje treba li izvući pozlaćenu ili dvije zlatne pločice u slučaju kata ZPZ. U nekim situacijama bolje je izvući dvije zlatne, a u nekim situacijama srednju pozlaćenu pločicu jer ćemo njome možda osigurati još dva nova izvlačenja. Zbog malog ograničenja na broj katova, možemo na sve moguće načine unaprijed odlučiti u koliko ćemo situacija iz kata ZPZ prednost dati zlatnim, a u koliko situacija pozlaćenoj pločici. Na kraju ispisujemo rješenje one strategije koja je dala najveći ukupan broj izvlačenja.

Za svaku pjegu na jednostavan način možemo izračunati kada najranije, a kada najkasnije mora početi snimka koja „rješava“ tu pjegu. Dakle, za svaku pjegu znamo u kojem vremenskom intervalu mora započeti odgovarajuća snimka.

Sada se zadatak svodi na odabir najmanjeg broja vrijednosti (početaka snimaka) takvih da svaki od dobivenih intervala sadrži neku od odabranih vrijednosti. Problem nam stvara činjenica da intervali idu „u krug“.

Kad intervali ne bi išli u krug, nego kad bi npr. svi bili prije podne (kao u jednom od podzataka), rješenje bi bio pohlepni algoritam koji odabire prvi završetak nekog intervala, izbacuje sve intervale koje pritom rješava, i to ponavlja. Ovaj postupak može se efikasno provesti sortiranjem intervala po njihovom završetku i jednim prolaskom po dobivenom nizu intervala.

Sada za „krug“ možemo provesti isti postupak za svaki mogući početni interval, tj. za svaki mogući odabir vremenske „granice“ od koje ćemo sortirati intervale po njihovom završetku, ispisujući onaj postupak koji je dao najbolje rješenje.

Rješenje se sastoji od nekoliko zamjedbi. Najprije primjećujemo da postoji samo jedna pozicija na koju možemo spustiti figuru tako da ukloni bilo koji red. Naime, očito moramo najprije ukloniti donji (prvi) red jer znamo da u njemu ima praznih polja, što znači da naša figura mora pokriti sva prazna polja u donjem redu, i to ne može učiniti na više načina jer bi to značilo da ih neće sve pokriti.

U figuri pronalazimo najviši stupac, a ako postoji više njih onda prvi lijevi. U početnom izgledu ekrana nalazimo prvi stupac visine nula. Figuru ćemo probati postaviti tako da se poklope ti pronađeni stupci. Sada kada smo odredili poziciju, pretpostavljamo da će figura na toj poziciji dodirnuti dno ekrana. Tada moramo odrediti koliko će redova biti potpuno popunjeni te hoće li se figura negdje preklapati s već spuštenim figurama. U slučaju da se figura preklapa s već spuštenim figurama, naša pretpostavka da figura pada na dno ekrana je pogrešna te se figurom ne može ukloniti nijedan red.

To određujemo tako da ekran dijelimo na intervale tako da je u svakom intervalu visina stupaca figure koje spuštamo na taj interval jednaka. Nazovimo tu visinu H_{gore}. U takvom intervalu pronalazimo najveću visinu stupca već spuštenih figura – nazovimo je H_{dolje}. Ako je $H_{dolje} + H_{gore}$ veće od visine figure (tj. njenog najvišeg stupca), tada bi se figura preklapala s već spuštenim figurama te njome ne možemo ukloniti nijedan red.

Za svaki interval treba pronaći i najmanju visinu stupca već spuštenih figura – nazovimo je h_{dolje}. Ako je $h_{dolje} + H_{gore}$ manje od visine figure (tj. njenog najvišeg stupca), tada broj redova koji uklanjamo ne može biti veći od h_{dolje} jer iznad tog stupca ostaje „rupa“. Također, broj redova koji uklanjamo ne može biti veći od visine figure, niti od visine stupaca početne strukture koji se nalaze lijevo ili desno od figure koju spuštamo. Broj redova koji možemo ukloniti jednak je minimumu ovih ograničenja.

Minimume i maksimume na intervalu možemo efikasno pronaći uporabom sažimanja i tournament stabla ili sparse tournament stablom.