Državno natjecanje iz informatike

Srednja škola

Druga podskupina (3. i 4. razred)

14. i 15. ožujka 2018.

OPISI ALGORITAMA

Autori zadataka: Tonko Sabolčec (*Poredak, Palindromi*), Mislav Bradač (*Loptica, Supertetris*), Mislav Balunović (*TurboExcel*), Dominik Gleich (*Kuglobećanje*)

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

Ministarstvo znanosti i obrazovanja

Agencija za odgoj i obrazovanje

HRVATSKA ZAJEDNICA TEHNIČKE KULTURE

HRVATSKI SAVEZ INFORMATIČARA

Postoji mnogo mogućih rješenja.

Jedno rješenje na početak niza stavlja brojeve N, N-1, N-2, ... tim redom, brojeći pritom nastale inverzije, što lako činimo jer se s desne strane stavljenog broja nalaze svi brojevi manji od njega. U trenutku kad bismo stavljanjem idućeg broja X premašili traženi broj inverzija, nastavljamo bez novih inverzija, brojevima 1, 2, 3, ... tim redom, pri čemu posljednji broj X u taj niz umećemo na ono mjesto gdje će broj inverzija biti upravo onaj koji nam je još preostao za postići.

Konačni je niz, dakle, sljedećeg oblika:

N, N-1, N-2, ..., X+2, X+1, 1, 2, 3, ..., X, ..., X-3, X-2, X-1.

Stanje definiramo kao polje i smjer kojim smo ušli u to polje. Održavamo tablicu u kojoj za svako stanje pamtimo u koje ćemo novo stanje prijeći i za koliko sekundi. Na početku, prije simulacije gibanja loptice, u tablici za svako stanje piše prvo sljedeće stanje (nakon jedne sekunde).

Simulacija jednu po jednu sekundu prespora je za veće test podatke. Ideja je povećavati korake: za vrijeme simulacije, ako nas tablica za stanje A upućuje na stanje B, a tablica za stanje B upućuje na stanje C, onda možemo i izravno iz stanja A doći u stanje C upisujući to u tablicu, čime ćemo tijekom simulacije raditi sve veće i veće korake. Valja paziti da ako je stanje B udarac u monitor, ne možemo preusmjeriti A na C jer će sljedeći put ishod (bez monitora) biti drugačiji.

Simulaciju provodimo dok ne dođemo u stanje koje je već posjećeno a da u međuvremenu nije srušen nijedan monitor. U tu svrhu trebamo dodatno pamtiti i broj srušenih monitora pri ulasku u pojedino stanje.

Formula koja opisuje neki stupac zapravo treba generirati do četiri stringa iz tog stupca (po jedan za svaki redak). Zadatak rješavamo dinamičkim programiranjem: zamislimo da smo već izgradili formulu koja generira neke početne dijelove (prefikse) traženih stringova i pitamo se koja je najkraća formula koja generira ostatak tih stringova i koju ćemo nadovezati na trenutačnu (`Concat`).

Stanje u dinamici je, dakle, niz od najviše četiri indeksa (*i1, i2, i3, i4*) koji označavaju dijelove stringova zadanog stupca koje još treba riješiti. U prijelazu na sve načine biramo jednostavnu formulu (bez `Concat`) koja opisuje neki sljedeći komad svakog stringa, tj. formulu kojom prelazimo u novo stanje oblika *(i1+k1, i2+k2, i3+k3, i4+k4*) pri čemu su *k1, k2, k3, k4* duljine komada opisanih novom formulom za retke od prvog do četvrtog.

Spomenute jednostavne formule koje provjeravamo su sljedeće:

- `$var`, `Upper($var)` ili `Lower($var)` za svaki mogući odabir stupca `var`,
- `Const(...)` pri čemu u zagradu može doći svaki zajednički prefiks (ako postoji) trenutačnih komada stringova koje opisujemo,
- `Substr($var, ?, ?)`, `Substr(Upper($var), ?, ?)` ili `Lower(Substr($var, ?, ?))` za svaki mogući odabir stupca `var` i odgovarajućih indeksa substringa.

Primijetite da ne moramo razmatrati varijante `Upper(Substr(...))` ili `Substr(Lower(...))` jer su leksikografski veće od gore navedenih. Kao rješenje za trenutačno stanje biramo fomulu koja, nakon spajanja s rješenjem za ostatak traženih stringova (tj. za novo stanje), ispada najkraća (ili leksikografski najmanja).

Zadatak je implementacijski zahtjevan i postoji mnogo detalja na koje valja paziti. Na primjer, nužna je rekonstrukcija rješenja, ali nadovezivanje (`Concat`) treba raditi samo jednom za cijelu formulu i zato ga ne valja koristiti u rješenjima dinamike za djelomične stringove, nego nadovezane formule privremeno treba npr. samo zarezima odvojiti. Važno je i dalje ispravno računati duljinu formule u svakom stanju radi ispravnog odabira najbolje formule.

Jednostavnom trigonometrijom ili formulama koordinatne geometrije možemo za svaki čunj izračunati u kojem kutnom intervalu ga „rušimo“, tj. koji je najmanji, a koji najveći kut pod kojim bačena kugla ruši taj čunj. Najmanji kut je onaj pod kojim će bačena kugla dodirnuti čunj s desne strane, a najveći kut s lijeve strane.

Sada se zadatak svodi na odabir najmanjeg broja vrijednosti takvih da svaki od dobivenih intervala sadrži neku od odabranih vrijednosti. To možemo riješiti pohlepnim algoritmom 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.

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.

Rješenje za 1. podzadatak

Svaki broj u zadanom intervalu možemo rastaviti na sve moguće načine te izbrojiti načine kod kojih je svaki dio palindrom. Složenost je ovog rješenja O((B-A)*2^N) gdje je N broj znamenki brojeva.

Rješenje za 2. podzadatak

Za svaki broj računamo broj načina da se rastavi na palindrome s pomoću dinamičkog programiranja. Neka je dp[i] broj načina na koji možemo rastaviti prefiks duljine *i* na palindrome. Tada vrijedi relacija: dp[i] = suma {dp[j]} za svaki j < i za koji vrijedi da je podstring između j-tog i i-tog znaka palindrom.

Rješenje za 3. podzadatak

Rastavimo broj X ($A \leq X \leq B$) na dva dijela tako da se desni dio sastoji od 5 znamenki, a lijevi od preostalih znamenki. Primijetimo da će se lijevi dijelovi ponavljati za skoro sve X-eve između A i B jer se uglavnom mijenja samo posljednjih 5 znamenki broja X kako ga povećavamo od A do B. Prema tome, možemo unaprijed izračunati vrijednost niza dp opisanog u rješenju prethodnog podzadatka za sve lijeve dijelove od X (kojih ima najviše 2). Također izračunamo niz dp2 za desne dijelove brojeva X. Na kraju samo možemo "spojiti" dva dijela po sredini pazeći na još neke detalje.

Rješenje za 4. i 5. podzadatak

Neka je f(X) ljepota broja X, a S(X) = f(0) + f(1) + f(2) + ... + f(X). Tada je potrebno odrediti vrijednost S(B) - S(A - 1).

Pokušajmo odrediti S(X) za proizvoljni X.

Za početak pretpostavimo da brojevi mogu započeti s 0 i da tražimo f(000...00) + f(000...01) + ... + f(999...99) gdje svi brojevi imaju jednak broj znamenki N.

Ovo možemo riješiti dinamičkim programiranjem. Neka je dp[i] = f(0...00) + f(0...01) + ... + f(9...99) ukupan broj rastava na palindrome svih brojeva (argumenata od f) koji imaju *i* znamenki. Tada vrijedi relacija:

$$dp[i] = \text{suma}\{\ dp[j] * h(i - j)\ \} \text{ za svaki } j < i,$$

gdje je h(x) broj palindroma duljine x (npr. "0..000", "0..010..0", "0..020..0", ... "9..989..9", "9..999..9").

Drugim riječima, u prijelazu dinamike biramo duljinu sljedećeg palindroma i odredimo na koliko načina možemo odabrati taj palindrom, što je u ovom slučaju broj h(x), gdje je x duljina palindroma. Primijetite da je h(x) = 10^((x+1)/2), pri čemu je dijeljenje s 2 cjelobrojno.

Međutim, brojevi ne smiju započeti s 0. Da bismo to popravili, računat ćemo dp[i] = f(10...00) + f(10...01) + ... + f(999...99) kao ukupan broj rastava na palindrome svih brojeva koji imaju *i* znamenki te nemaju vodećih nula. Relacija je sada drugačija:

$$dp[i] = \text{suma}\{dp[j] * h1(i - j) \text{ za sve } 0 < j < i\} + h2(i),$$

pri čemu je h1(x) = 10^((x+1)/2 broj palindroma duljine x kojima početna znamenka može biti 0, a h2(x) = 9 · 10^((x-1)/2) broj palindroma duljine x kojima početna znamenka mora biti različita od nule.

Vrijednost f(0) + f(1) + f(2) + ... f(999...99) sada možemo dobiti tako da pokrenemo dinamiku za svaki broj znamenki n ($0 \leq n \leq N$).

Na kraju još moramo ograničiti brojeve s gornje strane tako da zbrajamo vrijednosti do f(X) umjesto do f(999...99). Dinamici ćemo dodati još jedno stanje - bound - koje poprima vrijednost 1 ako moramo paziti na gornju granicu ili 0 ako ne moramo paziti na gornju granicu. Sada je stanje dinamike dp[pozicija][bound]. Početno stanje je dp[0][1]. Radi jednostavnije implementacije, dp će nam označavati broj rastava od trenutačnog stanja do kraja broja (tj. desno), za razliku od prethodnih relacija koje su računale broj rastava lijevo od trenutačne pozicije *i*. Relacije stoga treba prilagoditi ovisno o smjeru u kojem računamo rješenje.

U prijelazu dinamike biramo duljinu sljedećeg palindroma u rastavu. Ako je bound = 1, računamo ukupan broj palindroma odabrane duljine pazeći na gornju granicu. Ako je bound = 0, ne moramo paziti na gornju granicu nego računamo kao što je prije opisano. U oba slučaja treba paziti na detalje i rubne slučajeve.

Za implementacijske detalje pogledajte priloženu implementaciju i komentare koji je pojašnjavaju.