Poskusno izbirno tekmovanje
Priprave na računalniške olimpijade 2018/19

Vid Kocijan
vid.kocijan@cs.ox.ac.uk

Zlaganje Zrn

Rešitve za delne točke te naloge bodo morda izgledale bolj strašljivo, kot rešitev za vse točke.

Prvo podnalogo lahko rešimo z izčrpnim preiskovanjem vseh možnosti, ali pa kar na roko. Reševanje na roko je verjetno lažja različica.

Pri omejitvah $n \leq 200$ si lahko privoščimo algoritem s časovno zahtevnostjo $O(n^3)$. Posegli bomo po dinamičnem programiranju. Opazimo, da lahko zrna zlagamo od najbolj zunanje vrstice in stolpca proti navznoter. Ukvarjajmo se torej samo s prvo vrstico in prvim stolpcem. Naj bo $f(m, k)$ logična vrednost, ki nam pove, ali se da razporediti m zrn v veljavno obliko, če je najdaljša stranica dolga natanko k zrn. Prva vrstica in stolpec bosta torej skupaj sestavljena iz $2 \cdot k - 1$ zrn. Če iz preostalih $m - 2 \cdot k + 1$ zrn lahko sestavimo veljaven vzorec s stranico krajšo ali enako $k - 1$, lahko to skupaj združimo v veljavno obliko. $f(m, k)$ je torej resnična, če je resnična vsaj ena izmed $f(m-2 \cdot k+1, k-1), f(m-2 \cdot k+1, k-2), \ldots, f(m-2 \cdot k+1, 1)$. Poglejmo robne primere: $f(1, 1)$ in $f(0, 0)$ sta resnična, čim $k > m$, ali pa je kakšen izmed k in m negativen, je $f(k, m)$ neresnična. Relevantnih stanj je $O(n^2)$, vsaka možna kombinacija m in k. Za izračun vsakega od njih rabimo $O(n)$ časa, saj moramo preveriti vsa stanja, našteta zgoraj. Ta rešitev torej porabi $O(n^3)$ časa. Ko izračunamo vsa stanja, poiščemo najmanjši k, pri katerem je $f(n, k)$ resničen. Izris rešitve je prepuščen v premislek.

Pri omejitvah $n \leq 10^4$ si lahko privoščimo algoritem s časovno zahtevnostjo $O(n^2)$. Ponovno bomo uporabili dinamično programiranje, vendar bomo stanje f tokrat definirali drugače. Naj bo $f(m)$ najmanjša dolžina stranice, pri kateri še lahko razporedimo m zrn v veljaven diagram. Vemo, da $f(1) = 1$. Nato induktivno gradimo naprej. Če velja $f(m) = k$, lahko m zrn razporedimo v kvadrat s stranico dolžine k. Okrog tega kvadrata lahko dodamo še eno plast (torej prvo vrstico in prvi stolpec). Stolpec in vrstica bosta dolžine vsaj $k + 1$. Torej vemo, da $f(m + k \cdot 2 + 1) = k + 1$. Seveda sta lahko prvi stolpec in vrstica tudi daljši. Sprehodimo se torej čez vse morebitne njune dolžine in popravimo vrednosti dosedaj izračunanih $f(m+i \cdot 2-1)$ za $k+1 \leq i \leq n$ (sprehodimo se čez vse dolžine zunanjih stranic i). Imamo torej n stanj $f(m)$, pri vsakem izmed njih naredimo $O(n)$ operacij, da popravimo vrednosti za vse $f(m+i \cdot 2-1), n \geq i > f(m)$. Izris rešitve je ponovno prepuščen v premislek.

Pri omejitvi $n \leq 10^5$ si lahko privoščimo zgolj $O(n\sqrt{(}n))$ operacij. Ta rešitev je zelo podobna prejšnji, le da ocenimo, da bo končni diagram zelo podoben kvadratu z nekaj dodatnimi vrsticami in stolpci. S števcem i iz prejšnje rešitve se bomo zato sprehodili samo do $\sqrt{(}n) + \varepsilon$, kjer je ε neka majhna konstanta (npr. 5).

Pri omejitvi $n \leq 10^6$ potrebujemo rešitev s časovno zahtevnostjo $O(n)$. Rešitev je požrešna in zahteva nekaj opazovanja in analize primerov. Če je n popoln kvadrat, zrna razporedimo v kvadrat. Sicer bo rešitev verjetno še vedno zelo podobna kvadratu. Poiščimo torej največji popoln kvadrat m^2, tako da $m^2 \leq n$ in razporedimo m^2 zrn v kvadrat s stranico m. Ostane nam $l = n - m^2$ zrn. Če je l sodo število, je rešitev preprosta. Dodamo še en stolpec in vrstico dolžine $\frac{l}{2}$. Če je l liho število, iz najbolj spodnjega desnega dela kvadrata vzamemo eno zrno. $l + 1$ je sodo število, zrna lahko spet razporedimo na enak način kot v sodem primeru (glej npr. primer z 10 zrni). Ali smo pokrili vse primere? Izkaže se, da ne. Če ta postopek uporabimo na npr. 7 ali 14 zrnih pridemo v neveljavno situacijo, kjer bi bila v predzadnji vrstici luknja. V specifičnem primeru, kjer je $\frac{l+1}{2} = m$, moramo dodati še eno dodatno vrstico in stolpec dolžine 1. Dokaz optimalnosti je prepuščen v premislek.

Kitajci prihajajo

Recimo, da ima mesto višino v, širino s, v njem pa je c kitajskih družin. Hitro lahko opazimo, da je cilj naloge najti pravokotnik s površino c, v katerem se že nahaja maksimalno število kitajskih družin.

V prvi podnalogi lahko tak pravokotnik iščemo zelo naivno. Za vsako možen zgornji levi kot ($O(vs)$ možnosti) in spodnji levi kot ($O(vs)$ možnosti) preštejemo vse kitajske družine ($O(vs)$ operacij za eno štetje). Neveljavne pravokotnike seveda zavržemo. To skupaj nanese $O(v^3s^3)$, kar je pri danih nizkih omejitvah sprejemljivo.

Za drugo podnalogo je dovolj zelo preprosta optimizacija: pregledujemo zgolj pravokotnike veljavnih velikosti. Določimo enega od robov pravokotnika ($O(vs)$ kombinacij), nato pa zgolj pregledamo možne veljavne pravokotnike (sprehodimo se čez delitelje c) (c kombinacij). Štetje nam še vedno vzame $O(vs)$ operacij. To podnalogo lahko rešimo tudi z algoritmom iz prejšnjega razdelka, če uporabimo preprosto optimizacijo: pregledujemo zgolj vsebino pravokotnikov veljavnih velikosti. Zahtevnost je tako $O(v^2s^2\sqrt{c})$

Pri opisu rešitve za tretjo podnalogo bomo predpostavili, da nismo našli elegantne rešitve za drugo podnalogo. Še vedno bomo pregledali vse možne pravokotnike v mestu, a bomo optimizirali štetje kitajskih družin. Uporabili bomo kumulativne vsote v 2 dimenzijah. Bolj formalno, naračunali bomo ločeno 2D tabelo f, tako da bo $f[a][b]$ število kitajskih družin v kvadratu z enim ogliščem v $(0, 0)$, z drugim pa v (a, b) (vključno). Vrednosti f v zgornji vrstici in levem stolpcu naračunamo kot kumulativne vsote v tabeli, za ostale pa si pomagamo s preprosto formulo: $f[a][b] = f[a-1][b] + f[a][b-1] - f[a-1][b-1] + t[a][b]$, kjer je $t[a][b] = 1$, če je na tem mestu kitajska družina in 0 sicer. Pravilnost te formule je prepuščena v premislek (skica). Isti premislek nam bo pomagal tudi pri izpeljevanju formule za štetje kitajskih družin v poljubnem pravokotniku. Za pravokotnik z zgornjim levim ogliščem v (a, b) in spodnjim desnim v (c, d) je število družin enako $f[c][d]-f[a-1][d]-f[c][b-1]+f[a-1][b-1]$ (paziti je treba na robne primere). Štetje kitajskih družin v nekem pravokotniku tako lahko opravimo v konstantnem času. Skupaj s preverjanjem vseh možnih pravokotnikov to nanese $O(v^2s^2)$ operacij.

Za zadnjo podnalogo preprosto kombiniramo rešitvi druge in tretje podnaloge, da dobimo algoritem zahtevnosti $O(vs\sqrt{c})$, kar zadošča za rešitev naloge.

Lučkar Jože

Za prvo nalogo si lahko privoščimo rešitev reda $O(n \cdot m)$. V tabeli si lahko hranimo stanje vseh lučk in pri vsaki spremembi popravimo vsa stanja.

Za drugo podnalogo je število lučk preveliko, da bi lahko opazovali vse. Uporabili bomo kompresijo koordinat, na kar namiguje že format izpisa. Preberemo vse intervale sprememb (torej intervale, na katerih se luči ugasnejo ali prižgejo). Bolj konkretno, če se luč a in luč b pojavita na robu nekih intervalov (ne nujno na robu istega intervala), števila med $a + 1, \ldots, b - 1$ pa ne, vemo, da morajo biti skozi celoten postopek enake (torej vse prižgane ali ugasnjene). Glede na to, ali sta se a ali b pojavila kot desni oz. levi rob intervala, sta mogoče del te množice enakih žarnic – prepuščeno bralcu v premislek. Vse te žarnice lahko torej obravnavamo kot eno žarnico. Število žarnic, ki jih moramo tako opazovati smo zreducirali na red velikosti $O(n)$. Sedaj si lahko privoščimo navaden naiven algoritem, kot v prvi podnalogi, za skupno časovno zahtevnost $O(n^2)$.

V tretji podnalogi trik iz druge podnaloge ni dovolj. Dodatno bomo uporabili metodo veder (buckets) in vse lučke (po kompresiji koordinat) razdelili v vedra velikosti $\sqrt{n}$. Oglejmo si interval neke posodobitve. Če interval v celoti vsebuje neko vedro, posodobimo celotno vedro (torej ločen števec, ki označuje vedro), namesto posameznih žarnic. Sicer pregledamo žarnice. Posodobitev ima torej časovno zahtevnost $O(\sqrt{n})$, celoten algoritem pa časovno zahtevnost $O(n\sqrt{n})$.

Četrta podnaloga ima dve možni rešitvi. Podrobneje si bomo ogledali eno izmed njiju. Uporabimo kompresijo koordinat, kot opisano v rešitvi druge podnaloge. Lučke bomo "analizirali" od leve proti desni. Opazimo, da nas izmed vseh posodobitev zanima zgolj zadnja. Inicializiramo prazno kopico (priority queue) in se premikamo od leve proti desni. Vsakič, ko naletimo na spremembo (torej na levi del nekega intervala) v kopico dodamo "dogodek", ki predstavlja en interval. Za vsak dogodek si hranimo, ali so se luči prižgale, ali ugasnile, kje je konec tega intervala in kateri po vrsti je ta dogodek bil. Kopica naj vse dogodke sortira po tem, kdaj so se zgodili (na vrhu se torej nahaja zadnji dogodek, tisti, ki nas zanima). Vrh kopice nam torej predstavlja zadnje stanje intervala lučk. Marsikatere intervale, ki predstavljajo zgodnje spremembe, bomo vstavili, čeprav jih morda nikoli ne bomo uporabili. Prav zato si za vsak interval hranimo, kdaj se zaključi. Če se na vrhu kopice pojavi dogodek, ko bi se moral že davno zaključiti, ga pre-

prosto izbrišemo in ignoriramo. Dodajanje in odstranjevanje dogodkov v kopici ima časovno zahtevnost $O(\log(n))$, skupna zahtevnost je torej $O(n \log(n))$.

Alternativna rešitev te naloge je nadgradnja tretje podnaloge z vedri, kjer lučke razdelimo v liste binarnega drevesa, namesto v vedra. Vsako točko na daljici, v kateri se luči prižgejo ali ugasnejo, shranimo v binarno drevo (npr. v set) (teh bo največ $2 \cdot n$). Pri posodobitvi intervala (a, b) najdemo vse točke med a in b in jih izbrišemo, dodamo pa točki a in b (pri čemer upoštevamo prižganost in ugasnjenost). S pravilno uporabo množice (set) in funkcij upper_bound, lower_bound in erase, lahko iskanje meja in brisanje opravimo v časovni zahtevnosti $O(\log(n))$, kar nam da skupno časovno zahtevnost $O(n \log(n))$.