zimska 2024 algo 1

Ivan Janjić/Fran Babić

Veljača

1 Weighting a Tree

Ako je dani graf stablo, krenemo od listova. Težina brida koji povezuje list x sa ostatkom stabla je jednoznačno određena te je jednaka c_x. Sada ažuriramo
$c_{p_x} = c_{p_x} - c_x$ gdje je p_x roditelj lista x. Maknemo čvor x iz stabla zajedno sa njegovim bridom te ponavljamo postupak dok ne maknemo sve bridove. Na kraju ostaje jedan čvor, korijen stabla *root* koji mora imati (ažuriranu) vrijednost $c_{root} = 0$. Raspisivanjem čemu je točno na kraju jednak c_{root} vidimo da vrijedi sljedeća jednakost:
$s \sum_{i=1}^{n_1} c_{a_i} = \sum_{i=1}^{n_2} c_{b_i}$ gdje su a_i čvorovi na parnoj udaljenosti od korijena, a b_i čvorovi na neparnoj udaljenosti. Ako promotrimo bipartitivan graf, njegove čvorove mozemo podijeliti u dva skupa tako da svi bridovi idu između ta dva skupa. Tada svaki brid pridonosi sumi oba skupa jednako, odnosno suma c_i je jednaka za oba skupa. Dakle, ako to ne vrijedi u danom ulazu, rješenja nema. Također, vidimo da jednaki uvijet vrijedi i za stablo, koje smo rješili, pa znamo riješiti svaki bipartitivan graf tako da riješimo neko razapinjuće stablo te na ostale bridove stavimo težinu 0.
Preostaje slučaj kada graf nije bipartitivan, odnosno kada postoji neparni ciklus. Konstrukcijom ćemo pokazati da je u ovom slučaju svaki niz c_i ostvariv. Pronađemo neki neparan ciklus te ukorijenimo neko razapinjuće stablo u nekom čvoru tog neparnog ciklusa. riješimo to razapinjuće stablo nakon čega će c_{root} imati neku vrijednost, recimo c. Tada ćemo na bridove neparnog ciklusa, krenući od korijena, dodavati težinama tih bridova redom $c/2, -c/2, c/2$.... Zbog toga što je ciklus neparan, svi čvorovi, osim čvora *root* ostat će nepromijenjeni, a čvoru *root* ćemo dodati c, što je upravo ono što smo željeli. Preostaje se uvjeriti da ovo uvijek možemo napraviti, to jest da je c paran broj, no c je alternirajuća suma početnih c_i, čija je parnost jednaka parnosti sume stupnjeva svih čvorova po uvjetu zadatka. Suma stupnjeva svih čvorova jednaka je $2m$ to jest paran broj te smo time gotovi.
Napomena
Proces razmišljanja bio bi sljedeći: imam neke čudne uvjete na brojeve, zanemarit ću ih za početak te pokušati riješiti lakšu verziju zadatka. Pošto radimo sa grafom, prirodan odabir je stablo. Rješenje za stablo nije pretjerano teško te ga pokušamo generalizirati. Promatramo karakteristike koje naše rješenje zadovoljava. Vidimo da taj uvjet zadovoljava puno veća klasa grafova, gdje su stabla podskup te klase. Automatski imamo rješenje za cijelu tu klasu. Promotrimo sada ostatak grafova. igrajući se s najjednostavnijim elementima, primjetimo da neparni ciklus uvijek možemo riješiti, upravo zbog danih uvjeta u zadatku. Sada se lako ovo spoji sa prethodnom idejom rješavanja stabla.

2 Commuter Pass

Pustimo Dijsktru iz čvorova S, T, U i V. Izgradimo sljedeći graf koji će imati usmjerene bridove: isti skup vrhova te postoji usmjereni brid $a \to b$ ako postoji neusmjereni brid $a - b$ u originalnom grafu te $dist(S,T) = dist(S,a) + w(a,b) + dist(b,T)$. Ovaj uvjet nam zapravo kaže da se taj brid nalazi na najkraćem putu između čvorova S i T te da je a bliži S. $w(a,b)$ označava cijenu originalnog brida. Očito je dani graf DAG koji ima izvor u S te sve "slijeva" u T. Analogno izgradimo graf koji ima izvor T, a sve se "slijeva" u S(samo obrnemo smjer bridova). Računamo dvije dinamike, za svaki graf posebno. Napravimo topološki poredak grafa. Idemo s kraja, recimo u grafu gdje je izvor S. $dp[T] = dist(T,V)$. Za sve ostale čvorove unatrag po topološkom poretku računamo $dp[x] = min(dist(x,V), dp[y])$ gdje je y takav da postoji brid $x \to y$. Sada su kandidati za rješenje $dist(U,x) + dp[x]$ za svaki x. Uzmemo najmanji takav. Analognu stvar napravimo za drugi graf. Ukupni najmanji takav zbroj je rješenje zadatka.

Napomena

Promatranje ovako izgrađenih grafova često daje plodom. Ključ za rješenje zadatka je uočavanje karakteristike rješenja. Naime, optimalno rješenje krene u čvoru U, dođe do čvora S' koji je na najkraćem putu od S do T, nakon toga dođe do čvora T' koji je na istom tom putu te dođe na kraju do čvora V. Djelomično rješenje je da prođemo po svim parovima S' i T'(mora vrijediti $dist(S,T) = dist(S,S') + dist(S',T') + dist(T',T)$ ili $dist(S,T) = dist(T,S') + dist(S',T') + dist(T',S)$) te uzmemo najmanji izraz oblika $dist(U,S') + dist(T',V)$. To je bila parcijala za $n \leq 300$ gdje možemo primjeniti Floyd-Warshallov algoritam za udaljenost između svaka dva čvora. Za puno rješenje fiksiramo S', recimo da je bliži S. Trebamo naći najbolji T' takav da su S' i T' na istom putu od S do T. No to nam upravo računa spomenuta dinamika, kako je svaki najkraći put od S do T sadržan u tom grafu, T' je čvor koji je dohvatljiv is S' u tom grafu, a ako je čvor T' dohvatljiv, bit ce kandidat za S' kada računamo dinamiku.

3 Long Travels

Rješenje je uzeti nasumično otprilike 200 čvorova, izračunati udaljenost svakog čvora od tih 200 čvorova BFS-om te za upit uzeti minimum od $dist(s,x) + dist(x,t)$ gdje je x jedan od tih čvorova. Zbog uvjeta $dist(s,t) >= n/10$, vjerojatnost da neki čvor nije na tom putu je $9/10$. Vjerojatnost da niti jedan od naših 200 čvorova nije na putu je $(9/10)^{200}) \approx 10^{-10}$ što je dovoljno mala vjerojatnost da možemo biti praktički sigurni da ćemo dobiti točnu vrijednost.

Napomena Ovdje je ključ bio iskoristiti neobičan uvjet iz zadatka da su putevi dosta dugački. To znači da ima puno čvorova između njih, te kada bi znali nekog od njih i udaljenosti prema svim ostalima, znali bi smo rješiti zadatak. Ne znamo točno koji čvor je na našem putu, ali zbog duljine se isplati pogađati.

4 Good Pairs

5 Rivers

6 Strongly Connected Tournament

Zbog načina na koji su određene vjerojatnosti za pojedine partije, za neki skup igrača nije bitan sam skup nego samo njegova veličina. Neka je $dp(n)$ očekivan broj partija za skup veličine n. Očito je $dp(0) = dp(1) = 0$. Neka je $strong(n)$ vjerojatnost da je turnir nad n čvorova jako-povezan. Neka je $loss(n, k)$ vjerojatnost da postoji skup od k igrača koji su izgubili sve partije od preostalih $n - k$ igrača. Fiksirajmo veličinu zadnje jako-povezane komponente, neka je ona i. Njegovi članovi su po definiciji izgubili protiv svih ostalih igrača. Također, po definiciji je ta komponenta turnir koji je jako-povezan. Dakle, vjerojatnost da je zadnja jako-povezana komponenta velicine i je $loss(n, i) \cdot strong(i)$. Po definiciji očekivane vrijednosti, tu vjerojatnost trebamo pomnožiti sa brojem partija koje će se odigrati. Za početak, odigrale su se partije između igrača u jako-povezanoj komponenti i ostalih. Takvih partija ima $(n - i) \cdot i$. Odigrale su se sve partije među tim igračima, njih $\frac{i \cdot (i-1)}{2}$. Rekurzivno se sada postupak ponavlja nad svim jako-povezanim komponentama, pa tako i ovoj. Za nju je "broj" odigranih partija $dp(i)$("broj" pod navodnicima jer nije točno broj nego očekivan broj sto nam u ovom kontekstu znači istu stvar). Ostale su još sve partije koje su odigrane među ostalih $n - i$ igrača i njihovim jako-povezanim komponentama. No to je upravo definicija $dp(n - i)$. Konačan izraz, kada posumiramo po svim mogućim veličinama zadnje komponente je $dp(n) = \sum_{i=1}^{n} strong(i) \cdot loss(n, i) \cdot (i \cdot (n-i) + \frac{i \cdot (i-1)}{2} + dp(i) + dp(n-i))$. Mali problem je sto na ovaj način ne mozemo izračunati $dp(n)$ zato što se on pojavljuje s desne strane za $i = n$. Ako izlučimo taj član iz sume te prebacimo dio uz $dp(n)$ na drugu stranu dobivamo izraz:
$(1 - strong(n) \cdot loss(n, n)) \cdot dp(n) = strong(n) \cdot loss(n, n) \cdot \frac{n \cdot (n-1)}{2} + \sum_{i=1}^{n-1} strong(i) \cdot loss(n, i) \cdot (i \cdot (n - i) + \frac{i \cdot (i-1)}{2} + dp(i) + dp(n - i))$. Kada izračunamo izraz s desne strane, pomnožimo ga s modularnim inverzom od $1 - strong(n) \cdot loss(n, n)$ kako bi dobili $dp(n)$. Preostaje jos izračunati $strong(n)$ i $loss(n, k)$. Izračunat cemo suprotnu stvar, to jest vjerojatnost da turnir nije jako-povezan. To znači da ima barem dvije jako-povezane komponente. Na sličan način kao i prije fiksiramo veličinu zadnje jako-povezane komponente i dobivamo izraz $\sum_{i=1}^{n-1} strong(i) \cdot loss(n, i)$. Sada je $strong(n) = 1 - \sum_{i=1}^{n-1} strong(i) \cdot loss(n, i)$ uz početno stanje $strong(1) = 1$.
$loss(n, 0) = 1$ za svaki n, to jest tehnički gledano, 0 igrača je izgubilo od svih n igrača. Sada računamo $loss(n, k)$ promatrajući igrača sa najvećim indeksom. Ako on nije u skupu koji gubi od svih drugih, on mora sve njih pobjediti te je vjerojatnost za to $loss(n - 1, k) \cdot (1 - p)^k$. Ako on je u skupu, mora izgubiti od svih igrača koji nisu u skupu te je vjerojatnost za to $loss(n - 1, k - 1) \cdot p^{n-k}$. Konačno dobivamo izraz $loss(n, k) = loss(n - 1, k) \cdot (1 - p)^k + loss(n - 1, k - 1) \cdot p^{n-k}$. Zaključak, ako prvo izračunamo sve potrebne $loss(n, k)$, pa sve potrebne $strong(n)$ i na kraju $dp(n)$, dobivamo rješenje u složenosti $O(n^2)$.
Napomena
Ovo je zadatak koji provjerava kako stojite s kombinatorikom, očekivanom vjerojatnosti i

pojmovima vezanim uz turnire. Ako sto dobro upoznati sa njima, ovaj zadatak je praktički rutinski, sto ne znači da nije težak. Lukav dio zadatka je bio uočavanje da je rekurzivna relacija ciklička, no da se može popraviti izlučivanjem i modularnim dijeljenjem.

7 The Maximum Prefix

8 Divisors

Kako ovdje radimo s nasumično generiranim brojevima, želimo pronaći strategiju kojom ćemo doći to traženog broja n u najmanje moguće (očekivano) broja koraka. Nakon svakog koraka broj koji trenutno imamo će biti veći ili jednak prethodnom tj. ako smo imali x i nakon jednog koraka $x \longrightarrow x'$ vrijedi $x' \geq x$, stoga zaključujemo da strategiju i očekivanje računamo redom od većih prema manjim x-evima.

Promatrajmo neki $1 \leq x < n$ i pokušajmo izgraditi strategiju u slučaju da je trenutni broj x. Možemo pretpostaviti da smo već izračunali očekivane vrijednosti za sve brojeve veće od x. Za neki $y \leq k$ će vrijediti da ga uvijek odbijamo tj. ostajemo na trenutom broju ili ćemo prvom pojavom tog broja povećati broj za y. To se jednostavno vidi iz činjenice da nam se strategija neće mijenjati nakon poteza ako nismo promijenili broj. Neka je S podskup djelitelja od x manjih ili jednakih k, za koje vrijedi da ukazivanjem nekog broja iz skupa S uvećavamo x za ukazani broj. Očekivani broj poteza se može izraziti kao

$$E(x) = \frac{\sum_{y \in S} E(x+y)}{k} + \frac{k - |S|}{k} E(x) + 1$$

tj. pojednostavljivanjem se dobije $E(x) = \frac{\sum_{y \in S} E(x+y)}{|S|} + \frac{k}{|S|}$. Sad je očito da za svaki fiksirani $|S| = l$ najisplativije je uzeti l najmanjih $E(x+y)$. Uzimanjem minumuma od navedenih, dobivamo optimalni $E(x)$.

Za kraj, potrebno je odrediti algoritam odlučivanja. Za $y|x$ dovoljno je provjeriti $E(x+y) < E(x)$.

9 Kraljice

Rješenje je induktivno. Posebni slučajevi su $n = 1$ i $n = 2$ kada je moguće postaviti samo jednu kraljicu. Za sve ostale neparne n mozemo popuniti cijelu plocu, a za parne ce ostati dva polja nepopunjena. Dokaz da ne možemo bolje od dva prazna polja za parne n ide analizom slučaja. Pretpostavimo da smo popunili sva polja osim dva koja mogu biti: u istom retku/stupcu ili niti jedno od tog dvoje. Za svaki slučaj se lako provjeri da je parnost broja kraljica koji napada ta dva polja neparan. Baza indukcije su $n = 3$ i $n = 4$ čiji su primjeri bili dani te čije detalje možete vidjeti u priloženom kodu. Cilj je proširiti $(n-2) \times (n-2)$ ploču do $n \times n$ ploče. Pretpostavimo da smo popunili koliko možemo gornju lijevu $(n-2) \times (n-2)$ ploču. Napišimo na polje 0 ako ga napada paran broj kraljica, a 1 ako ga napada neparan broj kraljica. Može se pokazati da će polja $(1, n-1)$ i $(1, n)$ činiti skup $\{0, 1\}$. Dodamo kraljicu na polje na kojem piše 0. Sada će na onom drugom polju pisati 0 pa na njega stavimo kraljicu. Analogna tvrdnja vrijedi za

polja $(n-1,1)$ i $(n,1)$. Oni će samo zamijeniti mjesta napravljenim operacijama zato što stavljanjem kraljice na polje $(1,n)$ promjenimo broj na polju $(n,1)$, a stavljanjem kraljice na polje $(1,n-1)$ promjenimo broj na polju $(n-1,1)$. Sada kad smo popunili ta polja, analogno nastavljamo popunjavati polja $(i,n-1),(i,n),(n,i),(n-1,i)$ dok nam u donjem desnom kutu ne ostane 3×3 ploča sa gornjim lijevim popunjenim poljem. Ovaj dio na kraju uvijek izgleda jednako te se popunjavanje može ručno izvesti za parne i neparne n posebno čije detalje također možete pronaći u priloženom kodu. Time je korak indukcije gotov pa i rješenje zadatka.

Napomena

Ovo je klasičan primjer "naporne" konstrukcije. Moguće je sve izvesti na papiru, no pošto ne morate rigorozno stvari dokazivati, puno je brže napisati program koji postavlja kraljice i ispisuje dozvoljena polja te se igrati s njim i uočavati uzorke. Iz test primjera je moguće naslutiti da će mozda uvijek biti moguće popuniti cijelu ploču, no ako se poigrate sa slučajem $n=4$ ili ako skucate brut, vidi se da to nije istina, ali je blizu istine. Sa malo logičkog uvjeravanja, za parne n ne možemo bolje od n^2-2.

Sada je na redu konstrukcija. Konstrukcije ovog tipa mogu biti eksplicitne, to jest moguće je naci formulu ili lijep uzorak kojim se postavljaju kraljice, ili su implicitne, najčešće induktivne. Induktivan pristup se nameće kao prirodniji. Pokušaj proširenja sa $(n-1)\times(n-1)$ ploče na $n\times n$ ploču neće uroditi plodom, pa motivirani različitim rješenjima za $n=3$ i $n=4$ pokušavamo proširiti za 2 retka i stupca što čistom silom uspijevamo.

10 Half Queen Cover

Prepostavimo da rješenje koristi k polu-kraljica. Postoji barem $n-k$ redova i stupaca koje ne sadrže niti jednu polu-kraljicu. Neka su $r_1,r_2,...,r_a$ prazni redovi, a $s_1,s_2,...,s_b$ prazni stupci. Promotrimo sljedećih $a+b-1$ polja: $(r_a,c_1),(r_{a-1},c_1),...,(r_1,c_1),(r_1,c_2),...,(r_1,c_b)$. Očito neka dijagonala sadrži najviše jedno polje iz ovog skupa, dakle različite polu-kraljice napadaju ova polja. Kako polu-kraljice pokrivaju sva polja vrijedi $a+b-1\leq k$. Također vrijedi $n-k\leq a$ i $n-k\leq b$. Zbrajanjem ovih nejednakosti dobivamo $2\cdot n\leq 3\cdot k+1$, odnosno $k\geq\lceil\frac{2\cdot n-1}{3}\rceil$. Pokazat ćemo konstrukcijom da je ovu ogradu moguće postići.

Za $n=3\cdot k+2$, $k+1$ polu-kraljica stavljamo na sporednu dijagonalu dornjeg lijevog $(k+1)\times(k+1)$ kvadrata, a preostalih k polu-kraljica stavljamo na sporednu dijagonalu donjeg desnog $k\times k$ kvadrata. Ako je $n=3\cdot k$ ili $n=3\cdot k+1$ dodamo jednu ili dvije polu-kraljice u donji desni kut te ostatak popunimo kao u prethodnom slučaju.

Napomena

Više matematički zadatak nego informatički. Opet, analizom karakteristike optimalnog rješenja dolazimo do donje ograde. Kako smo dobili netrivijalnu donju ogradu, vrijedi pokušati pronaći konstrukciju. Pošto polu-kraljice napadaju glavne dijagonale, ima smisla pokušati ih postaviti na sporedne sto sa malo igranja dovodi do konačne konfiguracije. Postoji i više algoritamsko rješenje, naime postavimo polu-kraljicu u gornji lijevi kut te nastavljamo postavljati na prvo polje koje nitko ne napada. Ispada da će kraljice biti razmaknute za "L"(2×1 pomak), te činiti lijepe stepenice. Razradu ovog rješenja je ostavljeno za vježbu.

11 Voting

Poredamo listove prema vremenu kada ih posjetimo dfs-om. Nije teško pokazati da ima neparan broj listova. Prvi list je poseban, a ostale podjelimo u parove tako da su 2. i 3. u paru, 4. i 5. itd. Odigramo potez u posebnom čvoru. Ako protivnik odigra potez u nekom čvoru, mi ćemo odigrati potez u čvoru s kim je on u paru.

Napomena

Pošto nam je dano da prvi igrač uvijek može pobijediti, često je tada nešto prvi potez, a svaki sljedeći je odgovor(često simetričan u nekom smislu) na protivnikov potez. Analizirajmo malo, na intuitivnoj razini, zašto dano rješenje funkcionira. Kada se odaberu sva djeca nekog čvora kojem su djeca samo listovi, on odabire za koga glasa te se ovaj postupak moze nastaviti kroz lanac nadređenih. Na taj način imamo "balansirane" i "pobjedničke" parove. Ako je nadređeni oba lista jednak, onda je par "balansiran", inace je "pobjednički". Kada protivnik napravi potez u "balansiranom" paru, mi takodjer napravimo tamo potez te vratimo stanje kao da protivnik nije ni napravio potez. Ako protivnik napravi potez u "pobjedničkom" paru, on je "osvojio" taj čvor i neke njegove nadređene, no mi potezom na drugi list u tom paru "osvajamo" za sebe neke nadređene. Pošto smo prvi krenuli i "osvojili" posebni list te u budućnosti njegove nadređene, na kraju ćemo imati više "osvojenih" čvorova i osvojiti korijen stabla.

12 Bishops

Postoji više rješenja ovog zadatka. Jedan je bio dati ekspicitnu konstrukciju za sve slučajeve. Slučajevi su $n = m$, n i m parni te barem jedan n ili m neparan. u prva dva slučaja rješenje je $n + m - 2$, a u trećem je $n + m - 1$. Drugo moguće rješenje je slično prethodnom bez potpunih opservacija te s algoritamskom konstrukcijom. Posebno se rješi slucaj $n = m$, a za ostale pretpostavimo $n < m$. Stavimo $2n$ lovaca u prvi i zadnji stupac. Sada su ostale neke "glavne" i neke "sporedne" dijagonale slobodne. Na nekoj glavnoj dijagonali vrijednost $x - y$ je konstantna gdje su x i y koordinate polja na toj dijagonali. Slično su na sporednim dijagonalama konstante $x + y$. Ako uzmemo jednu sporednu i jednu glavnu dijagonalu te riješimo sustav s dvije jednadžbe i dvije nepoznanice dobit ćemo polje na koje možemo staviti lovca. Moramo paziti da uparimo dijagonale iste parnosti inace rješenje sustava neće biti cjelobrojno. Također ćemo upariti najveće brojeve s najmanjim kako bi se rješenje sustava nalazilo unutar granica ploče. Treće rješenje promatra bipartitivan graf gdje su s "lijeve" strane čvorovi koji predstavljaju sporedne dijagonale, a s "desne" glavne, a brid između dva čvora znači da postoji polje na ploči koje je presjek te dvije dijagonale. Ovim reformulacijom je zadatak postao pronaći maksimalno uparivanje. Postoji algoritam koji rješava ovaj problem u složenosti $O(V * E)$ gdje je V broj čvorova, a E broj bridova što neće rješiti zadatak u našem slučaju. Promotrimo bolje strukturu našeg grafa. Vraćajuci se na to što čvorovi predstavljaju, vidimo da svaka sporedna dijagonala zapravo pokriva interval glavnih dijagonala, odnosno svaki čvor s lijeve strane povezan je sa intervalom čvorova s desne. Ispada da je ovo moguće rješiti dovoljno brzo. Mi zapravo imamo $n + m - 1$ intervala te trebamo uzeti najveći podskup u kojem se za svaki interval može odabrati jedinstveni reprezentant. Obrnimo uloge te za neki reprezentant biramo interval s kojim će biti uparen. Uzmimo najmanji

broj koji se nalazi u nekom intervalu. Njega uparimo sa intervalom koji ima najmanju desnu granicu a da ga sadrži. Postupak ponavljamo dokle možemo. Ovaj algoritam može se implementirati na više načina, naprimjer koristeći set ili priority queue čija je razrada ostavljena za vježbu.

13 Yet Another Minimization Problem

Ovo je standardna primjena divide-and-conquer dp optimizacije sa malim twistom kod računanja *cost* funkcije. Generalno, ovu optimizaciju primjenjujemo ako imamo dinamiku sljedećeg oblika:

$$dp(n,k) = min_{n'<n}(dp(n',k-1) + cost(n',n))$$

te *cost* funkcija zadovoljava nejednakost četverokuta, to jest vrijedi:

$$a < b < c < d \implies cost(a,c) + cost(b,d) \leq cost(a,d) + cost(b,c)$$

Označimo sa $opt(n,k)$ indeks n' za koji se postiže optimalna vrijednost od $dp(n,k)$. Može se pokazati da ako vrijedi nejednakost četverokuta onda vrijedi i sljedeća tvrdnja:

$$a < b \implies opt(a,k) \leq opt(b,k)$$

Fiksirajmo k. Želimo ozračunati sve vrijednosti $dp(n,k)$ gdje je $1 \leq n \leq N$. Prvo ćemo izračunati $dp(mid,k)$ gdje je $mid = \frac{N}{2}$. Nakon toga znamo $opt(mid,k)$. Za sve $n < mid$ znamo $opt(n,k) \leq opt(mid,k)$, odnosno ne moramo proći kroz sve prijelaze. Analogna tvrdnja vrijedi za sve $mid < n$. Svaki od ta dva intervala rješavamo rekurzivno na isti način. Ukupna složenost je $O(NKlogN)$ gdje je K ukupan broj različitih k zato što za svaki k izračunamo sve vrijednosti dinamike u $O(NlogN)$. To je zato što na svakoj razini napravimo najviše $O(N)$ prijelaza, a razina ima $logN$.
Ova analiza složenosti je dobra ako je računanje *cost* funkcije $O(1)$. U našem slučaju to nije direktno moguće, a nemamo vremena sve izračunati unaprijed. Koristit ćemo trik koji se pojavljuje u Mo-ovom algoritmu, a to je lijeno pomicanje granica. Ako trenutno znamo odgovor za $cost(a,b)$ onda možemo saznati odgovor za $cost(a',b')$ u $|a-a'|+|b-b'|$ operacija pomicanja granica. Ispada da je ovo dovoljno brzo, a analiza složenosti je slična kao prethodna analiza za složenost optimizacije.

14 Joker

Neka je za neki $1 \leq l \leq M$ $opt(l)$ najmanji indeks takav da je graf bez bridova u intervalu $[l, opt(l)]$ bipartitivan(ili $M+1$ ako takav indeks ne postoji), a graf bez bridova u intervalu $[l, opt(l)-1]$ nije bipartitivan. Može se pokazati da vrijedi $l_1 < l_2 \implies opt(l_1) \leq opt(l_2)$.
Kako bi rješili zadatak koristit ćemo isti trik kao u prethodnom zadatku.
Imamo rekurzivnu funkciju koja prima parametre l_1, l_2, r_1, r_2. Oni znače da je za sve indekse l u intervalu$[l_1, l_2]$ $opt(l)$ u intervalu $[r_1, r_2]$. Za početak pozivamo fukncju za parametre $1, M, 1, M+1$. Pretpostavimo da smo u pozivu funkcije za parametre l_1, l_2, r_1, r_2. Održavamo globalnu union-find strukturu koja trenutno ima u sebi sve bridove s indeksom manjim od l_1 i većim od r_2. Struktura treba moći provjeriti bipartitivnost ako pokušamo

dodati neki brid, moći dodati brid ako je bipartitivnost očuvana te imati mogućnost *rollbackanja*, odnosno poništavanje zadnje operacije. Neka je $mid = \frac{l_1+l_2}{2}$ te želimo izračunati $opt(mid)$. To radimo tako da dodamo u union-find bridove $[l_1, mid-1]$ te pokušamo dodati bridove $r_2, r_2-1, ...$, dok ne dođemo do indeksa r_{mid} čijim dodavanjem graf neće biti bipartitivan. Zaključujemo da $opt(mid) = r_{mid}$ te rollbackamo sve dosada dodane bridove. Sada radimo sljedeće:
1. dodamo bridove $r_{mid}+1, ..., r_2$ te se pozovemo rekurzivno za parametre $l_1, mid-1, r_1, r_{mid}$. Rollbackamo dodane bridove.
2. dodamo bridove $l_1, ..., mid$ te se pozovemo rekurzivno za parametre $mid+1, l_2, r_mid, r_2$.
U svakom pozivu radimo $(l_2-l_1)+(r_2-r_1)$ dodavanja i micanja bridova koji, zbog nemogućnosti path compression-a u union-findu imaju složenost $O(logN)$. Kako je ukupan broj dodavanja i micanja po razini $O(M)$, razina $O(logM)$, ukupna složenost je $O(MlogMlogN)$.
Napomena
Jedna nestandardna primjena "divide-and-conquer" optimizacije, to jest iskorištavanje istog pravila na *opt* niz koji "čuva" poredak na neki način. Ako ikad u divljini uočite to svojstvo, osim razmišljanja u smjeru two-pointera, provjerite ima li smisla raditi nekakav offline divide-and-conquer na upitima kao u ovom zadatku. Za detalje implementacije ovog union-finda pogledajte priloženi kod.

15 Xor

Pronađemo najmanji broj tako da pošaljemo upit za broj 0, neka se zove *lo*. Najveći broj pronađemo tako da pošaljemo upit za broj $2^{30}-1$ te taj broj xor-amo sa $2^{30}-1$. Neka se zove *hi*. Pozovemo rekurzivnu funkciju $DnC(lo, hi)$. Funkcija DnC ce otkriti sve brojeve u intervalu $[lo, hi]$ s pretpostavkom da su *lo* i *hi* u skupu koji želimo otkriti. Ako $lo = hi$ gotovi smo. Inače pronađemo prvi bit u kojem se brojevi *lo* i *hi* razlikuju. Očito ce on biti 0 u broju *lo*, a 1 u broju *hi*. Neka je $suff$ broj koji ima isti prefiks bitova koji su jednaki u brojevima *lo* i *hi*, sljedeci bit 1, a a svi ostali bitovi neka su 0. Npr. za $lo = 1010100$ i $hi = 1011010$, $suff = 1011000$. $ask(a)$ označava odgovor na upit za broj a. Konstruiramo tri broja, $z = 2^i - 1$, gdje je i pozicija bita gdje se *lo* i *hi* razlikuju, $x = ask(suff + z) + suff + z$, $y = ask(suff) + suff$. Očito su ovi brojevi unutar skupa zato sto smo dobili rezultat za neki broj i točno taj broj dodali tom rezultatu, "negirajući" utjecaj tog broja. x je najveći broj koji ima 0 na mjestu razlike, a y je najmanji broj koji ima 1 na mjestu razlike. Zato između njih nema niti jedan broj u skupu. Pozovemo $DnC(lo, x)$ i $DnC(y, hi)$. Za svaki broj ćemo najviše 2 puta poslati upit, jednom da ga otkrijemo i drugi puta da ga "potvrdimo" to jest, ako $lo = x$ ili $hi = y$, u sljedećem koraku smo gotovi.
Napomena Igranjem i rješavanjem 3. podzadatka nije teško pronaći način za odrediti najveći i najmanji broj u skupu. Jedna od stvari koje se pitamo dok rješavamo zadatak, može li se ovo riješiti nekom dinamikom ili divide-and-conquer pristupom, to jest možemo li svesti zadatak na sličnu, ali jednostavniju instancu. Ako pokušamo divide-and-conquer, željeli bi smo odrediti brojeve u "sredini" te se onda granati. Gledajući brojeve izmedju *lo* i *hi*, uočavamo da neki prefiks ima zajednicki prefiks sa *lo* i *hi*, a na sljedecem bitu 0. Svi brojevi nakon toga na tom bitu imaju 1. Dakle, granica gdje se taj bit "pretvara" iz

0 u 1 čini se kao prirodna granica. No sada vidimo da nije teško generalizirati postupak za nalazenje najvećeg i najmanjeg u skupu na naš slučaj. Imamo dva skupa, jedan gdje je bitan bit 0 a jedan gdje je on 1. Za naći naveći broj u prvom skupu želimo iskoristiti trik gdje ce najveći broj biti "najsličniji" broju sa svim jedinicama. Postavimo sve manje bitove na 1, no moramo ostaviti isti prefiks kako bi xor na tim mjestima bio 0, to jest niti jedan broj van našeg skupa neće imati bolju vrijednost. Analogno rjesavamo drugi skup te dobijamo podjelu $[lo, x]$ i $[y, hi]$ koja sadrži sve naše brojeve. Lagano se uvjeriti da je ovo rješenje dovoljno dobro.

16 Remont

Pretvaramo zadatak u ekvivalentnu instancu 2-SAT-a. Varijable su "razmaci" između brojeva, označimo ih sa $x_1, x_2, ..., x_{n-1}$. Za poziciju $1 < i < n$ dodamo uvjet $x_{i-1} \vee x_i$. Dodatno, dodamo uvjete $x_1 \vee x_1$ i $x_{n-1} \vee x_{n-1}$ za bojanje rubova. Još trebamo dodati uvjete da jedan roler smijemo iskoristiti najviše jednom. Skupimo sve varijable koje odgovaraju istom roleru i nazovimo ih $a_1, a_2, ..., a_m$. Za 60 bodova dovoljno je proći po svim $1 \leq i < j \leq m$ i dodati uvjet $\neg a_i \vee \neg a_j$. Za sve bodove ćemo dodati nove "dummy" varijable pomoću kojih ćemo smanjiti broj bridova u grafu. Svaku vrstu rolera opet zasebno rješavamo. Ako su varijable koje koriste taj roler $a_1, a_2, ..., a_m$ dodajemo varijable $p_1, p_2, ..., p_m$ u graf. p_i će biti istina ako postoji barem jedna varijabla među $a_1, a_2, ..., a_i$ koja je istina. To ćemo napraviti tako da dodamo bridove $a_i \implies p_i$ i $p_i \implies p_{i+1}$ te njihove kontrapozicije. Još trebamo dodati uvjete koji ce biti ekvivalentni uvjetu iz zadatka. Ako postoji više od jedne varijable koje su istinite, postojat ce indeks i takav da je a_i istina te da je p_{i-1} istina. Dakle moramo ovaj slučaj spriječiti. To radimo dodavanjem uvjeta $\neg a_i \vee \neg p_{i-1}$.

Napomena

Prvi dio zadatka je bilo uočiti da se može riješiti 2-SAT-om što je često najteži dio zadatka. Drugi dio je bio smanjiti broj bridova dodavanjem "dummy" varijabli radeći na neki način prefiks sume. Ovo, kada ste jednom vidjeli, postaje puno lakši dio zadatka.

17 Magneti

Sortiramo magnete prema radijusu od najmanjeg do najvećeg. Računamo dinamiku $dp(i, j, l)$ koja označava koliko rasporeda postoji u prvih i magneta rasporedjenih u j komponenti ukupne duljine l. Odgovor na pitanje iz zadatka je $\sum_{d=1}^{l} \binom{l-d+n}{n} \cdot dp(n, 1, d)$. Kada dodajemo novi magnet, on sam može biti nova komponenta te se ukupna duljina ne promjeni. To možemo napraviti na 1 način. Možemo ga dodati na kraj neke komponente, što možemo napraviti na $2 \cdot j$ načina te se ukupna duljina poveća za r_i. Na kraju, možemo spojiti dvije komponente u jednu s novim magnetom u sredini. To možemo napraviti na $j \cdot (j + 1)$ način te se ukupna duljina poveća za $2 \cdot r_i - 1$.

Napomena

Standarna primjena takozvane "connected component" dinamike. U početku sortiramo magnete po radijusu zbog toga što kad dodajemo novi magnet, sigurni smo za koliko ce se promijeniti duljina. Ne moramo paziti hoće li ga pokriti neki drugi magnet zato što

je njegov radijus veći ili jednak svim dodanim, pa ako osiguramo da naš novi magnet ne prekriva niti jedan drugi magnet znamo da je to dovoljno.

18 Trokuti

Želimo za početak saznati neke bridove te induktivno graditi nad onime što imamo. Postoji više načina za ovo postići, ovdje ću opisati dva. Prvi je proći po svim grafovima na 5 čvorova kojih ima 2^{10}. Provjerom se utvrdi da ako napravimo upit za svaku trojku, jedinstveno je određen graf na temelju tih upita. Drugi način je pitati sve trokute u grafu na 6 cvorova. Postoji teorem koji kaže da u grafu na 6 čvorova postoji trokut koji ima sve bridove ili postoji trokut koji nema niti jedan brid. Očito s tom informacijom za neki trokut znamo stanje svih bridova.
Sada gradimo graf induktivno, to jest za novi čvor ćemo saznati stanje bridova prema svim prethodno dodanim čvorovima. Podjelimo već dodane čvorove u parove. Sada postavimo upit za svaki par plus čvor koji dodajemo. Pošto znamo stanje brida između čvorova u paru, među 2 brida koja povezuju novi čvor sa parom možemo dobiti odgovor 0, 1 ili 2. Očito 0 ili 2 jedinstveno određuju stanje tih bridova. Ako je odgovor 1, proslijedimo jedan čvor iz para na sljedeću razinu. Ponavljamo postupak dok više nema čvorova na sljedećoj razini. Sada se vraćamo unatrag i rješavamo čvorove koji ostali nerješeni, to jest čvorove koje nismo poslali na nižu razinu. No za njih znamo stanje brida od drugog čvora u paru, pa time znamo i njih.
Napomena
Preostaje pokazati zašto je dano rješenje dovoljno dobro. bridovi iz novog čvora prema dva čvora u paru mogu se naći u 4 stanja: oba brida postoje, oba ne postoje, postoji samo prema prvom čvoru ili postoji samo prema drugom. U 2 od ta 4 slučaja proslijedit ćemo jedan od njih na nižu razinu. Dakle, ako uzmemo da su svaki od ova 4 slučaja nasumični s jednakom vjerojatnosti, u polovini slučajeva pošaljemo polovicu čvorova na nižu razinu, to jest broj čvorova se promjeni za faktor $\frac{1}{4}$. Ako na razini ima m čvorova, na njoj napravimo otprilike $\frac{m}{2}$ upita, odnosno ukupno napravimo otprilike $\frac{1}{2}\sum_{k=1}^{\infty}\frac{n}{4^k} = \frac{2n}{3}$ upita ako dodajemo $n+1$. čvor. Zaključak je da otpilike napravimo $\frac{2}{3}\frac{n(n-1)}{2}$ upita što za n=100 daje vrijednost 3300. U realnosti ćemo biti ili s jedne ili s druge strane ovog broja što je dovoljno s obzirom da je granica u zadatku 3400.