Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Patrick Pavić, Ivan Paljak i Krešimir Nežmah. Prirmjeri implementiranih rješenja dani su u priloženim izvornim kodovima.

Zadatak A – Alergični Aron

Predložio: Patrick Pavić
Potrebno znanje: teorija grafova, union-find

Probat ćemo za svaki brid naći najveću komponentu u kojoj je upravo on brid s najmanjom težinom.

Soritrajmo sve bridove po težinama. Dodavat ćemo ih od najvećih prema manjima u određenu strukturu na način da možemo saznati veličinu komponente u kojoj se trenutno nalazi čvor te imati operaciju dodavanja brida. Te operacije podržava struktura union-find.

Primijetimo da nakon spajanja s novim bridom e, svi bridovi u komponenti imaju vrijednost veću ili jednaku vrijednosti tog brida. Dakle, nakon spajanja možemo samo pogledati veličinu komponente i pomnožiti s težinom tog brida.

Uzmemo li maksimum tog umnoška po svim bridovima, dobili smo rješenje. Time smo dobili složenost $\mathcal{O}(n \log n)$ zbog sortiranja bridova.

Zadatak B – Bliski Brojevi

Predložio: Patrick Pavić
Potrebno znanje: Moov algoritam, turnirsko stablo

Primjenimo Moov algoritam na upite. Sada trebamo podržati operacije: dodaj broj u strukturu, izbaci broj iz strukture, nađi par s najmanjom apsolutnom razlikom.

To možemo izvesti pomoću turnirskog stabla.

Turnirsko stablo će u čvoru koji prikazuje interval $[l, r]$ održavati najmanji broj l' iz tog intervala, najveći broj r' iz toga intervala te najmanju apsolutnu razliku među dva broja koja se oba nalaze u tom intervalu. Primjetimo da pri spajanju dva djeteta u njihovog roditelja u turnirskom stablu trivijalno možemo izračunati nove l' i r', a najmanji odgovor će biti ili jedan od dva odgovora djece, ili razlika među r' lijevog djeteta i l' desnog djeteta. Time imamo strukturu koja u $\mathcal{O}(\log n)$ podržava ubacivanje, izbacivanje i upit.

U Moovom algoritmu napravit ćemo $\mathcal{O}(n\sqrt{n})$ operacija nad strukturom što daje složenost $\mathcal{O}(n\sqrt{n} \cdot \log n)$. Rješenje je bilo potrebno implementirati s malom konstantom kako bi prošlo ograničenja.

Postoji i $\mathcal{O}(n\sqrt{n})$ rješenje koje se ostavlja čitatelju za vježbu.

Pomoć: podijelite upite na veće i manje od $\sqrt{n}$.

Zadatak C – Crni Ceh

Predložio: Krešimir Nežmah
Potrebno znanje: Fenwickovo ili turnirsko stablo

Na početku je ukupan crni ceh jednak 0. Izračunat ćemo koliko se promijenio odgovor nakon svake promjene u rezultatima.

Ako natjecatelj u crnoj majici dobije bodove, potrebno je pronaći od koliko je natjecatelja u žutim majicama on upravo postao bolji te za toliko povećati odgovor. Sličan postupak radimo ako natjecatelj u žutoj majici dobije bodove.

Preostaje brzo izračunati koliko postoji natjecatelja čiji je broj bodova manji ili jednak nekoj vrijednosti. To možemo koristeći *Fenwickovo stablo*, po jedno za svaku boju majice. Budući da je broj bodova natjecatelja uvijek između 0 i $3 \cdot 10^5$, nema problema s održavanjem te strukture u memoriji.

Vremenska složenost je $\mathcal{O}(n + q \log \text{MAXD})$

Zadatak D – Dramatični Dvoboj

Predložio: Patrick Pavić
Potrebno znanje: Grundyev teorem, Fenwickovo stablo, Gaussova eliminacija

Definirajmo stanje igre kao tepih na vrhu. Izračunajmo Grundyeve brojeve za sve tepihe te njihove zarotirane inačice. Primijetimo da, ako tepih i možemo staviti na tepih j te tepih j na tepih k, onda možemo staviti tepih i na tepih k. Odnosno, uvjet mogućnosti "stavljanja" je tranzitivan. Stoga, umjesto uzimanja *mex*-a svih stanja u koje možemo otići, jednostavno možemo uzeti maksimum te ga uvećati za jedan.

Grundyeve brojeve možemo izračunati tako da sve tepihe sortiramo uzlazno po S_i te održavamo Fenwickovo stablo u koje ubacujemo njihove Grundy brojeve na poziciju D_i. Na taj način kada dođemo do nekog tepiha, pogledano maksimum pomoću Fenwickovog stabla u intervalu $[0, D_i >$ te ga uvećamo za jedan i time dobijemo Grundy broj tog tepiha. Ako taj tepih nije zarotirana inačica, onda je on moguće stanje u koje možemo doći iz nekog drugog stanja te ga dodajemo u Fenwickovo stablo.

Neka je za i-ti tepih na tlu, G_i njegov Grundy broj, a G'_i Grundy broj njega zarotiranog. Po Grundy teoremu, stanje cijele igre je xor svih pojedinačnih stanja, a drugi igrač pobjeđuje ako je broj nula. Stoga, želimo odabrati brojeve na način da je taj xor nula. Prvo odabiremo svugdje broj G_i te xoranjem tih brojeva dobijemo broj X. Sada, ako pomoću brojeva (G_i xor G'_i) možemo dobiti podskup čiji je xor X, onda na tim mjestima koja su u skupu zarotiramo tepih i time smo dobili rješenje. To možemo riješiti Gaussovom eliminacijom, ako operaciju xora promatramo kao zbrajanje vektora modulo 2.

Zadatak E – Elokventni Evaluator

Predložio: Ivan Paljak
Potrebno znanje: brute-force

Ovo je u potpunosti implementacijski zadatak. Da biste na našem, manje elokventnom evaluatoru ugledali poruku **Accepted**, bilo je potrebno pažljivo pročitati tekst zadatka, implementirati ono što piše u tekstu zadatka i pokriti sve slučajeve. Standardnu (natjecateljsku) implementaciju možete vidjeti u izvornom kodu `E.cpp`, a nešto kompaktniju implementaciju koja koristi regularne izraze i funkciju `eval` u Pythonu, možete vidjeti u izvornom kodu `E.py`.

Zadatak F – Fantastični Fožgaj

Predložio: Krešimir Nežmah
Potrebno znanje: dinamičko programiranje, trie, potenciranje matrica

Pogledajmo prvo naivno rješenje. Koristimo dinamičko programiranje. Neka je $dp[s]$ broj mogućih govora ako smo dosad već izgovorili string s. Ako s sadrži neku od riječi s popisa onda je $dp[s] = 0$, a inače, ako je duljina od s jednaka m, tada je $dp[s] = 1$. U prijelazu pokušavamo produljiti s sa svakim slovom.

Uočimo da je jedini problem ako u nekom trenutku pri prijelazu napravimo riječ čiji je sufiks neka riječ s popisa. U stanju je, dakle, dovoljno pamtiti duljinu dosad izgrađene riječi te najdulji sufiks od s koji je neki prefiks riječi s popisa. Konkretno, ako ubacimo sve riječi s popisa u *trie* (stoblo), stanje postaje trenutna duljina te pozicija u trieu.

Da bi prijelaz bio brz, potrebno je za svaki čvor u trieu znati dvije stvari:

- sadrži li on kao podstring neku riječ s popisa.
- kada bismo probali dodati slovo na kraj tog čvora, koji je najdulji sufiks tako dobivene riječi koji se nalazi u trieu.

Dakle, potrebno je označiti čvorove u trieu koje u dinamici ne smijemo posjetiti te dodati dodatne bridove prema stanjima koje možemo posjetiti. Takav graf moguće je izgraditi u linearnoj složenosti, no kako je ukupni broj znakova u trieu najviše 100, bilo koja naivna konstrukcija takvog grafa trebala bi stati unutar vremenskog ograničenja.

Vremenska složenost dinamike trenutno je $\mathcal{O}(nm)$, što je zasad očito previše. Da bismo je ubrzali, prijelaze možemo bilježiti u 100×100 matrici. Potenciranjem te matrice na m-tu potenciju možemo dobiti odgovor za govor duljine m. Koristeći brzo potenciranje matrica, dobivamo vremensku složenost $\mathcal{O}(n^3 \log m)$.

Zadatak G – Golema Gozba

Predložio: Krešimir Nežmah
Potrebno znanje: teorija grafova, konstruktivni algoritmi, dfs

Tvrdimo da rješenje uvijek postoji (gospodin Malnar ipak uspijeva bez problema naći raspored). Dokaz je konstruktivan.

Napravimo graf u kojemu su osobe od 1 do $2n$ čvorovi. Ako su dvije osobe u istom paru, povezat ćemo ih crvenim bridom. Dodatno, povezat ćemo plavim bridom osobe u parovima $(1, 2)$, $(3, 4)$, $\ldots$, $(2n - 1, 2n)$. Uočimo da ne postoje dva brida iste boje koji dijele krajnju točku. Zato će svaki ciklus u ovakvom grafu imati alternirajuće boje (crvena, plava, crvena, plava, $\ldots$), pa je svaki ciklus parne duljine. Iz toga slijedi da je graf bipartitan te bojenje možemo pronaći jednostavnim dfs algoritmom.

Uvjeti zadatka sada su očito zadovoljeni – crveni bridovi osiguravaju da dvije osobe iz istog para nemaju isto jelo, a plavi bridovi osiguravaju da među tri uzastopne pozicije u krugu nema tri ista jela.

Vremenska složenost je $\mathcal{O}(n)$.

Zadatak H – Herojski Histogram

Predložio: Krešimir Nežmah
Potrebno znanje: convex hull trick

Izračunat ćemo za svaki stupac j, koja je najveća površina pravokutnika čija je desna granica upravo stupac j. Nakon što smo to napravili, odgovor za prvih j stupaca je maksimum među tim vrijednostima od 1 do j.

a svaki stupac pronađimo prvi stupac lijevo od njega koji je manji od njega, za stupac j neka je to $left[j]$, gdje uzimamo $left[j] = 0$ ako takav stupac ne postoji. Promatrajmo neki stupac na indeksu p_1 s visinom h_1. Možemo pratiti padajući niz stupaca $p_2 = left[p_1]$, $p_3 = left[p_2]$, ..., $p_m = left[p_{m-1}]$ dok ne dođemo do kraja ($p_m = 0$). Njihove visine neka su redom h_1, h_2, ..., h_m. Svakom pravokutniku čija je desna granica p_1, možemo pomicati lijevu granicu dok ne dođemo do nekog p_i, čime samo povećavamo površinu. Dakle, dovoljno je razmatrati samo one pravokutnike čija je lijeva granica za jedan desno nekog od indeksa p_1, p_2, ..., p_m. Odgovor za stupac p_1 je maksimum od vrijednosti $(p_1 - p_{i+1}) * h_i$, za i između 1 i $m-1$, što možemo napisati i kao $h_i * p_1 - h_i * p_{i+1}$.

Definirajmo zato za svaki stupac funkciju – za stupac i imamo $f_i(x) = h_i * x - h_i * left[i]$. Tada vidimo da je odgovor za stupac p_1 zapravo maksimum od vrijednosti $f_{p_1}(p_1), f_{p_2}(p_1), \ldots, f_{p_{m-1}}(p_1)$. Budući da su sve navedene funkcije linearne, ovo je poznati problem u kojem je potrebno održavati gornji envelope danih pravaca (tzv. *convex hull trick*).

No problem je što za različite početne p_1, imamo različite ljuske i ne možemo izgraditi svaku ispočetka. Konkretno, ako promatramo vezu j i $left[j]$ kao dijete i roditelj, dobivamo stablo ukorijenjeno s pozicijom 0, a da bismo dobili odgovor za neku poziciju, potrebno je promatrati konveksnu ljusku dobivenu prateći put od te pozicije do korijena. Uočimo dodatno da na putu od korijena do bilo kojeg čvora visine stupaca rastu, dakle nagibi pravaca su rastući.

Pokrenut ćemo dfs obilazak stabla i održavati trenutnu ljusku pravaca. Moramo podržati dvije operacije:

- prelazak s roditelja na dijete – u ljusku dodajemo novi pravac, a prije toga smo ih nekoliko morali izbrisati (uvijek brišemo sufiks jer na putu od korijena su nagibi pravaca rastući).
- prelazak s djeteta na roditelja – moramo vratiti ljusku u originalno stanje.

Naivno brisanje pravaca daje kvadratnu složenost u slučaju da u stablu postoji dugački lanac koji završava s čvorom koji ima veliki stupanj.

Da bismo efikasno napravili promjene, pri ubacivanju možemo binarnim pretraživanjem pronaći poziciju u ljuski gdje je potrebno ubaciti novi pravac. Potrebno je zabilježiti tu promjenu te ažurirati veličinu ljuske, a pri povratku je onda lagano rekonstruirati originalni hull.

Za više detalja pogledajte i zadatak *Harbingers* sa CEOI 2009 koji je popularizirao ovaj algoritam u svijetu natjecateljskog programiranja.

Vremenska složenost je $\mathcal{O}(n \log n)$.

Zadatak I – Idilični Instagram

Predložio: Ivan Paljak
Potrebno znanje: brute-force

Zamislimo da su sve slike s Alenkinog instagram profila poredane u niz duljine n redoslijedom objava. Lagano je uočiti da su slike dobro preklopljene ako za svako putovanje osim kronološki najstarijeg (sufiks niza) vrijedi da je broj objavljenih slika s tog putovanja višekratnik broja tri.

Budući da broj različitih putovanja ne može biti veći od 26 (broj slova u engleskoj abecedi), možemo za svako putovanje pretpostaviti da će upravo ono biti najstarije nakon brisanja nekih slika. Broj slika na novom profilu u tom je slučaju jedinstveno određen. Sva starija putovanja od fiksiranog najstarijeg moraju biti u potpunosti obrisana, a za svako novije putovanje je potrebno obrisati najmanji broj slika tako da broj slika koje ostanu bude višekrantik broja tri. Jasno, taj broj odgovara ostatku pri dijeljenju broja slika sa tri.

Ovaj postupak je konstruktivan pa je i rješenje zadatka potpuno. Ovaj algoritam moguće je implementirati u vremenskoj složenosti $\mathcal{O}(n)$.

Zadatak J – Jaki Jovsi

Predložio: Patrick Pavić
Potrebno znanje: olbat stablo, dinamičko programiranje

Izgradimo *olbat stablo* (*palindromsko stablo*, *eertree*) nad nizom znakova. Sada ćemo dinamičkim programiranjen izračunati odgovor za svaki čvor (tj. palindrom) te ga pomnožiti s brojem pojavljivanja u nizu. Za svaki čvor X postoji čvor BK – najveći palindrom koji je sufiks palindroma X te PAR što je palindrom koji nastaje micanjem prvog i zadnjeg znaka iz tog palindroma. U olbat stablu to su čvor u koji pokazuje backedge i čvor koji je roditelj tog čvora, zato oznake BK i PAR. Stanje dinamike će se sastojati od oznake između 0 i 2 te čvora X.

Oznaka 1: rješenje kad bismo u prvom potezu ostavili baš taj palindrom
Oznaka 0: kada bismo ostavili ili taj palindrom ili bilo koji palindrom koji je prefiks početnog.
Oznaka 2: kada bismo u prvom potezu ostavili ili taj palindorm ili bilo koji podpalindorm.

Podpalindrom palindroma X je svaki palindrom koji se nalazi kao podriječ unutar X.

Prijelazi dinamike su:

$$dp(X,1) = dp(PAR,2) + 1$$
$$dp(X,0) = dp(BK,0) + dp(X,1)$$
$$dp(X,2) = dp(PAR,2) + dp(X,1) + 2 \cdot dp(BK,0)$$

Vremenska složenost je $\mathcal{O}(n)$