# Univerzitetni programerski maraton 2021
# 2. kolo — rešitve nalog

Tomaž Hočevar
tomaz.hocevar@fri.uni-lj.si

6. maj 2021

## Posplošen Pitagorov izrek

Naloga sprašuje po številu celoštevilskih rešitev $1 \leq a \leq b \leq c \leq$ enačbe $a^n + b^n = c^n$. Če je $n > 2$, nam Fermatov zadnji izrek pove, da enačba nima rešitev. Nalogo pa lahko rešimo tudi brez tega dejstva. Preverimo vse možne pare števil $a \leq b$. Pri tem nas zanima, ali je $a^n + b^n$ slučajno n-ta potenca nekega celega števila. Vsoto lahko korenimo in preverimo, ali je koren celo število. Lahko pa si vnaprej pripravimo množico $n$-tih potenc celih števil in namesto korenjenja naredimo poizvedbo v tej množici.

## Plaža

Problem modeliramo z grafom. Vozlišča predstavljajo osebe, povezave pa sprte osebe. Vozlišča moramo pobarvati z dvema barvama, tako da nista povezani dve osebi iste barve. Gre za iskanje dveh ločenih skupin vozlišč v dvodelnem grafu. Če ima prva oseba določeno barvo, lahko enolično določimo barve vsem osebam, ki so sprte z njo, itd. To lahko izvedemo učinkovito v času $O(m)$ s sprehodom po grafu v globino ali širino, lahko pa tudi v več prehodih čez povezave v $O(nm)$. Naloga zagotavlja, da barvanje obstaja (ni ciklov lihe dolžine) in je enolično (obstaja samo ena povezana komponenta).

## Davki

Natančno moramo slediti opisanemu postopku izračuna davkov. Podani primeri so temeljiti in dobra pomoč pri razhroščevanju rešitve. Največja ovira je oblika izpisa števil z ločili za tisočice. V večini programskih jezikov obstaja podpora za tako obliko izpisa. Če se lotimo lastne implementacije, pa moramo biti pozorni na posebne primere, kot je npr. 0,00.

## Oklepajski izrazi

Pri ugotavljanju pravilnosti celotnega oklepajskega izraza vodimo sklad odprtih oklepajev, ki še nimajo pripadajočega zaklepaja. Če je podniz od $i$-tega do $j$-tega znaka pravilen oklepajski izraz, bo sklad pred dodatkom $i$-tega znaka enak skladu po dodatku $j$-tega. Vmesni oklepaji se bodo namreč pozaprli v prazen seznam. Stanje sklada lahko enolično določimo z indeksom zadnjega znaka, ki se nahaja na njem. Tako lahko v $O(1)$ med seboj primerjamo sklade v različnih trenutkih.

## Zrcaljenje

Točko $P$ lahko zrcalimo čez vektor $v = B - A$ z uporabo projekcije na njegov enotski normalni vektor $n$: $P' = P - 2\,\text{proj}_n(P - A) = P - 2(n^T(P - A))n = (I - 2nn^T)P + 2n^TAn = ZP + D$. Transformacijo lahko zapišemo z matriko $M$ dimenzije 3x3:

$$\begin{bmatrix} Z_{11} & Z_{12} & D_1 \\ Z_{21} & Z_{22} & D_2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix} = \begin{bmatrix} P'_x \\ P'_y \\ 1 \end{bmatrix}$$

Več zaporednih transformacij opišemo s produktom pripadajočih matrik. Trenutno koordinato točke tako poiščemo z enostavnim množenjem izračunanega produkta transformacijskih matrik z vektorjem točke.

Težava pa nastane pri vmesnem dodajanju nove točke. Izračunamo lahko, kakšna bi morala biti začetna lokacija točke, da bi po trenutnih transformacijah pristala na svojem mestu in jo od tam naprej obravnavamo kot vse ostale začetne točke. Za izračun potrebujemo inverz produkta transformacijskih matrik, ki pa je v primeru zrcaljen enostaven. Transformacijske matrike samo množimo v obratnem vrstnem redu.

# Škrati

Podano imamo drevesno strukturo po kateri se premikajo škrati in puščajo odtise na povezavah ali pa jih čistijo. Na pot se odpravijo ob različnih časih. Vsako škratovo pot od vozlišča $A$ do $B$ bomo razbili na dve poti do njunega najnižjega skupnega prednika (lowest common ancestor $L = lca(A, B)$). LCA je klasičen problem, kjer si lahko v $O(N \log N)$ pripravimo podatkovno strukturo, s pomočjo katere lahko v $O(\log N)$ odgovarjamo na LCA poizvedbe.

Ker nas zanima samo končno stanje, si bomo označili vse spremembe na drevesu, nato pa te spremembe združevali od listov proti korenu drevesa. Za škrata, ki se giblje od vozlišča $A$ do $L$ si pri vozlišču $A$ označimo, da tam začne svojo pot proti korenu škrat tipa $T$ ob času $C$, v vozlišču $L$ pa pot ta isti škrat zaključi. Spremembe hranimo v množicah parov (čas, tip škrata). Učinkovito jih združujemo npr. v vozlišču $V$ tako, da vzamemo rezultat največjega poddrevesa in vanj vstavimo množice iz ostalih poddreves (small to large merging). Po tem dodamo ali odstranimo spremembe, ki se nanašajo neposredno na vozlišče $V$. Skupna časovna zahtevnost teh operacij je $O(N \log N)$.

V tako zgrajenih množicah nas zanima škrat z največjim časom, ker se bo ta zadnji premaknil po povezavi od vozlišča $V$ proti korenu in edini prispeval h končnemu rezultatu. Če so te množice implementirane z drevesno strukturo (npr. set, TreeSet), je ta poizvedba prav tako $O(\log N)$.