

Univerzitetni programerski maraton 2025

3. kolo — rešitve nalog

Tomaž Hočevar

tomaz.hocevar@fri.uni-lj.si

21. maj 2025

Komunikacija

Za vsak (urejen) par vozlišč moramo izračunati vsoto uteži povezav med njima. Vendar ne moremo obravnavati kar vseh $O(N^2)$ možnih parov in tudi ne moremo zgraditi matrike sosednosti velikosti $O(N^2)$. Zato obravnavamo vseh M povezav in v nekem slovarju hranimo vsoto uteži od manjšega do večjega vozlišča. Slovar je lahko realiziran kot razpršena tabela (*unordered_map*, *HashMap*, *dict*) ali pa kot uravnotežena drevesna struktura (*map*, *TreeMap*).

Študentje in asistentje

Učinkovito moramo simulirati premikanje asistentov med študenti. V vsakem trenutku moramo biti zmožni hitro poiskati asistenta, ki je najdlje prost. V ta namen jih lahko hranimo urejene po času, ko so postali ali bodo postali prosti. Da bo to dovolj učinkovito, uporabimo prioriteto vrsto z dvojiško kopico (*priority_queue*, *PriorityQueue*, *heapq*) ali kakšno drugo uravnoteženo drevesno strukturo. Asistenta z najbolj zgodnjim prostim časom vzamemo iz vrste, ga premaknemo do študenta in nato vstavimo nazaj v vrsto s časom, ko bo zaključil delo pri temu študentu.

Rebus

Na prvi pogled izgleda, da bo naloga zahtevala res veliko implementacije, vendar ni tako hudo. Uganko moramo razdeliti na praznih stolpcih v več slikovnih namigov. Vsak slikovni namig moramo obrezati oz. določiti očrtani pravokotnik. Vse možne slike hranimo v slovarju, da učinkovito najdemo pripadajoči tekstovni namig. Ko združimo vse tako dobljene tekstovne namige, moramo v slovarju besed poiskati tisto, ki je namigu najbolj podobna. Za vsako besedo izračunamo razdaljo med namigom in njo glede na dovoljene operacije brisanja črk. To lahko počnemo rekurzivno z memoizacijo ali pa z dinamičnim programiranjem od spodaj navzgor. Če se prva znaka ujemata, se ukvarjamo z malo krajšimi nizi; če pa se prva znaka ne ujemata, je treba izbrisati vsaj enega od njiju. Razdalja med nizoma $s[i:]$ in $t[j:]$ je enaka

$$d(i, j) = \begin{cases} d(i+1, j+1), & \text{če je } s[i] = t[j] \\ \min(1 + d(i+1, j), 1 + d(i, j+1)), & \text{sicer} \end{cases}$$

Digitalni pano

Iz začetne in končne slike lahko določimo, katere piksele je treba obrniti in katerih ne. Vsak ukaz na izbrani koordinati bomo očitno izvedli kvečjemu enkrat. Število pikslov na panoju ne presega 200, zato krajša stranica ne bo daljša od $w = 14$ pikslov. Orientirajmo pano tako, da širina krajša od obeh dimenzij. Če bi vedeli, katere ukaze je treba izvesti v prvi vrstici, lahko enolično določimo preostanek. Ker tega ne vemo, bomo preizkusili vseh 2^w možnosti. Če je treba piksel $(0, x)$ spremeniti, ga lahko samo tako, da izvedemo ukaz na $(1, x)$. Tako so vse potrebne operacije povsem definirane in če se pri tem zadnja vrstica reši pravilno, je bila izbira prve vrstice veljavna. Med vsemi veljavnimi rešitvami izberemo tisto z najmanj potrebnimi ukazi.

Quasi-sort

Podan imamo naključen graf, v katerem imajo vozlišča eksponentne stopnje (1, 2, 4, 8 ali 16). Izračunali bi radi približek topološkega vrstnega reda s čim manj kršitvami. Vozlišča brez vhodnih povezav lahko damo na začetek. Če jih kdaj zmanjka, izberemo vozlišče s čim manj vhodnimi povezavami, ki bodo prispevale h kršitvam. Tako lahko pridemo do deleža 31%. Med tistimi, ki imajo enako vhodnih povezav, je smiselno izbrati tiste, ki imajo veliko izhodnih. Taka rešitev doseže približno 25%.

Glavno opažanje je, da je kdaj smiselno izbrati tudi vozlišče, ki ima malo več vhodnih povezav, vendar ima zato veliko več izhodnih. S tako izbiro trenutno malo bolj povečamo število kršitev, vendar se tudi znebimo velikega števila povezav, ki bi v prihodnosti prispevale h kršitvam. Enostavna razlika med vhodno in izhodno stopnjo vozlišča doseže 20%.

Izboljšanje rešitve zahteva nekaj eksperimentiranja s cenilnimi funkcijami, katero vozlišče izbrati glede na vhodno in izhodno stopnjo. Z njimi lahko eksperimentirate na kakšnem naključno sestavljenem primeru. Izkaže se, da izbira vozlišča z največjo vrednostjo $\text{out}(x) - 3 * \text{in}(x)$ vodi do rezultata približno 17.3%. S podobno sočasno obravnavo vozlišč, ki jih lahko postavimo na konec seznama (npr. takih, ki imajo izhodno stopnjo nič ali pa majhno izhodno stopnjo ter veliko vhodno), lahko rezultat znižamo celo na 17.1%. Da je rešitev dovolj učinkovita, hranimo vozlišča v prioritetni vrsti glede na izbrano cenilno funkcijo ter jih posodablamo ob spremembah zaradi odstranitve vozlišč.

Oddajnik

Iščemo krog s polmerom D , ki pokriva čim več izmed podanih N točk. Vsako rešitev lahko malo premaknemo, dokler se s krožnico ne dotika vsaj dveh točk. Vsaki dve točki (recimo A in B) torej določata dva možna kroga z danim polmerom (enega s središčem na levi strani daljice AB in drugega na desni). Središče X takega kroga lahko izračunamo vsaj na dva načina, ki oba vključujeta razpolovišče daljice AB , recimo mu C .

1. Izračunamo razdaljo od C do središča kroga iz polmera D in dolžine daljice AC . Določimo pravokotnico na AB skozi C in se premaknemo iz C za izračunano razdaljo v tej smeri, da dobimo središče kroga.
2. Izračunamo kot $\angle BAX$ iz dolžine AC in polmera D ter se pod tem kotom premaknemo za D enot iz točke A .

Težava pa je v tem, da ne moremo za vseh $O(N^2)$ parov preveriti vseh N točk, da bi prešteli vsebovane.

Lahko pa izberemo eno točk A in opazujemo, kaj se dogaja pri izbiri različnih točk B . Vsaka definira središče kroga na desni strani daljice AB , ki se nahaja pod kotom α glede na izhodišče v točki A . Če si predstavljamo, da zavrtimo krog okoli točke A , bodo pri izračunanih kotih nekatere točke vstopile v krožnico, druge pa izstopile (nekatero pa so predaleč, da bi sploh igrale kakšno vlogo). Če te dogodke uredimo, lahko nato v linearnem času izračunamo, koliko točk pokriva krog s središčem pri različnih naraščajočih kotih. Paziti moramo, ker med koti obstaja krožna urejenost (krog pri največjih kotih zajema tudi točke pri najmanjših kotih). Časovna zahtevnost te rešitve je $O(N^2 \log N)$.