Univerzitetni programerski maraton 2025
2. kolo — rešitve nalog

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

3. april 2025

Tiskanje računa

Implementirati je treba opisani postopek, ki pa vsebuje par nadležnosti. Vsako vrstico lahko razdelite na tabulatorju na ime in ceno. Ceni lahko vejico zamenjate s piko, da jih lahko kasneje primerjate po vrednostih (bodite previdni, da ne pride do kakšne zaokrožitvene napake). Artikle nato uredite glede na ceno in njihov originalni indeks v seznamu. Za izpis morate izračunati najdaljše ime artikla in najdaljšo ceno artikla, da jih lahko primerno poravnate s pikami oz. presledki. Pri izpisu ne pozabite zamenjati decimalne pike nazaj z decimalno vejico.

Slogan

Vsake operacije ne bomo mogli kar izvesti v celoti, ker lahko tako v vsaki izmed K operacij spremenimo vseh $|S|$ znakov niza, kar bo prepočasno. Za vsak indeks v nizu si lahko hranimo, katera črka se nahaja tam. Tako moramo ugotoviti samo še, ali je ta črka velika ali majhna. To pa je odvisno samo od zadnje globalne spremembe velikosti črk v nizu. Če se je ta sprememba zgodila po nastavitvi črke, bo veljala velikost iz te globalne spremembe, sicer pa velikost ob nastavitvi črke. Da lahko to izvedemo moramo poleg črke na vsakem mestu hraniti tudi čas, ko smo jo nastavili.

Popust

Na začetek bi radi premaknili številko x, ki je čim manjša, vendar ne sme biti 0. Ugotoviti pa moramo, katero od najmanjših števk bomo izbrali. Skupine zaporednih števk x lahko obravnavamo kot eno, ker bodo vse izbire iz te skupine dale enak rezultat. Če izbrano številko postavimo pred začetek, se bo na njeno mesto premaknila njena desna soseda. Če je ta soseda $y = 0$, naj se to zgodi čim prej. Če je ta soseda $y > x$, pa se bo na njeno mesto prestavila večja števka. Zato bi bilo bolje, če bi tam ostala števka x in bi raje prestavili kakšno kasnejšo z enako vrednostjo.

Poiščemo torej najmanjšo neničelno številko x. Če se v nizu pojavi x, ki mu sledi 0, izberemo prvi tak x. Sicer pa izberemo zadnji x.

Poravnane kriptovalute

Kriptovalute uredimo po imenih in nato ugotavljamo, ali njihove vrednosti x_i oblikujejo monotono (naraščajoče ali padajoče) zaporedje. Vzdrževali bomo statistiko, koliko zaporednih vrednosti je naraščajočih $x_i \leq x_{i+1}$ in koliko padajočih ($x_i \geq x_{i+1}$). Če je katero od teh dveh števil enako $n - 1$, imamo opravka z monotonim zaporedjem. Ob vsaki spremembi, poiščemo lokacijo kriptovalute (z bisekcijo ali slovarjem) in popravimo statistiko. Najprej zmanjšamo število naraščajočih in padajočih sosedov, ki jih formira stara vrednost na svoji levi in desni strani, nato pa jih primerno povečamo glede na novo vrednost.

Niti

Naloga zahteva kar nekaj implementacije, da sploh deluje, nato pa še nekaj optimizacije, da je rešitev dovolj hitra. Najprej je treba pretvoriti logične izraze v neko prikladno obliko, da jih bomo lahko izračunali pri različnih vrednostih spremenljivk. Prav nam bo prišla predstavitev izraza z drevesno strukturo, ki jo lahko zgradimo z rekurzivnim razbijanjem na mestu zadnje najšibkejše operacije.

Nato lahko rekurzivno simuliramo vse možne vrstne rede izvajanja ukazov in hranimo vrednosti spremenljivk kar v posameznih bitih nekega celega števila. Teh vrstnih redov je povsem preveč, vendar če kdaj dosežemo iste vrednosti spremenljivk pri istem številu obdelanih ukazov v vsakem od treh programov, bomo od tam naprej generirali iste rešitve in lahko generiranje prekinemo. Tako bomo obiskali $O(2^N P^3)$ stanj, kjer je N število binarnih spremenljivk, P pa dolžina posameznega programa.

V vsakem stanju moramo ovrednotiti izraz, ki je lahko precej dolg in vsebuje veliko operacij. Tudi to lahko pohitrimo z vnaprejšnjim izračunom vrednosti vsakega izraza pri vseh možnih 2^N vrednostih spremenljivk. Za to potrebujemo $O(2^N PL)$ časa, kjer je L največje možno število operacij v posameznem izrazu (npr. njegova dolžina). Tako lahko v vsakem stanju kar preberemo že izračunano vrednost izraza v $O(1)$. Skupna časovna zahtevnost je torej $O(2^N P(P^2 + L))$.

Škoda

Ker so škode eksponentne, se v njihovem produktu eksponenti seštevajo. Minimizirati želimo torej vsoto eksponentov. Politike lahko predstavimo s točkami v ravnini. Škoda stranke pa ustreza površini pravokotnika z enim ogliščem v izhodišču, ki pokriva vse politike v tej stranki. Zanima nas torej, kako z minimalno vsoto površin pravokotnikov pokriti vse točke.

Problem si malo poenostavimo z opazovanjem samo Pareto fronte točk - če točke uredimo naraščajoče po x koordinati, bodo njihove y koordinate padale. Problema se lahko lotimo z dinamičnim programiranjem. Naj bo $f(i)$ minimalna vsota površin za pokritje prvih i točk. Recimo, da se bo predhodni pravokotnik začel pri j-ti točki. Potem je $f(i) = \min_{j<i} f(j) + y_{j+1} \cdot x_i$. Ta rešitev najde pravilen odgovor, ni pa dovolj učinkovita, ker ima časovno zahtevnost $O(n^2)$.

Vidimo, da vsaka rešitev pri indeksu j v nadaljevanju vpliva na rešitve pri $i > j$ v obliki linearne funkcije $l(j) = a_j + b_j x$, kjer je $a_j = f(j), b_j = y_{j+1}$, in ki jo ovrednotimo pri različnih vrednostih x. Za izračun $f(i)$ iščemo najmanjšo vrednost linearne funkcije $l(j)$ pri $x = x_i$ za nek j. Ko rešujemo podprobleme za vedno večje indekse, se nakloni b_j manjšajo. Hranili bomo ovojnico linearnih funkcij v seznamu, kjer lahko z bisekcijo najdemo relevanten odsek in izračunamo vrednost. Ko dodamo novo linearno funkcijo, odstranimo nekaj zadnjih odsekov te ovojnice, ki so irelevantni, ter izračunamo od katere vrednosti naprej je relevantna nova linearna funkcija. V kontekstu optimizacije dinamičnega programiranja, je tehnika znana pod imenom "convex-hull optimization". Časovna zahtevnost je $O(n \log n)$.

Alternativna rešitev uporablja drevo linearnih funkcij (t.i. "Li Chao tree"), ki je drevesna podatkovna struktura za učinkovito hranjenje množice linearnih funkcij, njihovo dodajanje in poizvedbe o minimumu/maksimumu pri izbrani vrednosti x.