Univerzitetni programerski maraton 2023
1. kolo — rešitve nalog

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

21. april 2023

Račun

Podani šifrant izdelkov morate v čim bolj primerni obliki spraviti v svoj program. Lahko kar skopirate večvrstični niz, lahko ga spravite v primerno obliko s kakšnim urejevalnikom besedila, v najslabšem primeru ga lahko celo pretipkate. Za izdelek na računu poiščete njegovo ceno v šifrantu in te cene seštejete. Vrednosti lahko zaokrožite glede na ostanek pri deljenju s 5 ali z 10. Bodite pozorni pri pretvorbi decimalnih števil v cela števila. Uporabljajte zaokroževanje, saj decimalne vrednosti v dvojiškem zapisu niso povsem natančne. Če npr. za $x = 0.29$ izpišete $100 \cdot x$ na dovolj decimalk, boste ugotovili, da je rezultat manjši od 29 (28.99999999999999644729) in če ga pretvorite v celo število, dobite 28.

Rok

Naloga zahteva, da maksimiziramo $\sum r_i - t_i = \sum r_i - \sum t_i$. Vsota rokov $\sum r_i$ je konstantna, zato moramo minimizirati vsoto časov oddaj $\sum t_i$. To dosežemo tako, da rešujemo naloge od krajših proti daljšim. V to se lahko prepričate z argumentom o zamenjavi, kjer bi v primeru drugačne rešitve lahko zamenjali dve sosednji nalogi (kjer se daljša reši pred krajšo) in tako izboljšali rezultat.

Kuhinjski pribor

Vhodni podatki nam podajo več vrstnih redov stolpcev v datoteki. Iz tega lahko sestavimo usmerjen graf, ki predstavlja relacije med stolpci, vsaka povezava pa nam pove, da se prvi stolpec nahaja pred drugim. Zanima nas topološki vrstni red vozlišč, kar je klasičen problem, ki ga lahko rešimo tudi z malo razmisleka. Stolpec oz. vozlišče brez vhodnih povezav lahko brez škode postavimo na prvo mesto in pobrišemo izhodne povezave. To logiko nato ponavljamo, dokler ne uredimo vseh vozlišč ali pa nam zmanjka vozlišč brez vhodnih povezav - v tem primeru imamo opravka s ciklom in iskani topološki vrstni red ne obstaja. Paziti moramo, da ne iščemo vozlišč brez vhodnih povezav vsakič znova, ampak jih sproti dodajamo v nek seznam ali vrsto in nato pobiramo iz njega.

Osmerosmerka

Na mreži črk moramo poiskati vse pojavitve podanih besed. Namesto iskanja besed na vseh možnih lokacijah, bomo generirali vse besede, ki se pojavijo na mreži in preverili, ali obstajajo v naši množici podanih besed. Za vsako začetno celico v mreži in za vseh 8 smeri sestavljamo vedno daljše besede po mreži. Če smo sestavili kakšno obstoječo besedo, si to označimo na mreži. Vprašanje je samo še, kako ugotoviti, ali je trenutno zgenerirana beseda v naši množici iskanih besed. To lahko naredimo zelo učinkovito, če sestavimo črkovno drevo (angl. trie) iskanih besed.

Bivaki

Nalogo lahko neposredno modeliramo s pretoki v grafu. Vsaka enota pretoka ustreza enemu turistu, ki ga moramo spraviti od izvora do ponora. Pri tem pa ne sme iti skozi isto vozlišče (bivak) več turistov od njegove kapacitete. Iščemo torej maksimalen pretok z omejitvami pretoka na vozliščih. Vsako vozlišče lahko razdelimo v vhodno in izhodno, ter med njiju dodamo usmerjeno povezavo s kapaciteto vozlišča. Ker so ostale povezave dvosmerne, jih predstavimo s po dvema usmerjenima povezavama, kjer vsaka povezuje izhodno vozlišče enega bivaka z vhodnim vozliščem drugega.

Ker so omejitve kapacitet v grafu in posledično tudi iskani pretoki velika števila, običajen algoritem Ford-Fulkerson za iskanje maksimalnega pretoka ne bo primeren. Dovolj dober pa bo že Edmonds-Karp, ki ni odvisen od velikosti pretoka, temveč samo od števila vozlišč in povezav v grafu.

Stiskanje nizov

Zanimajo nas pogostosti različnih podnizov. V tem primeru pogosto pridejo prav priponske podatkovne strukture, kot je npr. priponsko polje (suffix array), ki ima vse pripone niza urejene po abecede. Uporabna lastnost pripon je ta, da so predpone pripon ravno podnizi. Obstajajo učinkoviti algoritmi za sestavljanje priponskih polj s časovno zahtevnostjo, ki je boljša od kvadratne. Poleg običajno sodi tudi izračun najdaljše skupne predpone sosednjih pripon v priponskem polju (LCP - longest common prefix).

Postopek je morda lažje razumljiv na priponskem drevesu (suffix tree). V priponskem drevesu bi za vsako vozlišče morali poznati oddaljenost od korena (dolžina podniza) in število pripon v njegovem poddrevesu (število podnizov). Podoben postopek pa lahko implementiramo z uporabo priponskega polja, ki vsebuje po vrsti urejena vozlišča v priponskem drevesu. Pri premikanju po polju bomo vzdrževali sklad naraščajočih dolžin skupnih predpon. Ko dodamo novo pripono, moramo zaradi nje odstraniti nekatere s sklada. Takrat si označimo število pojavitev, ki jih prispeva nova pripona v intervalu do prejšnje, ki jo odstranjujemo.