Univerzitetni programerski maraton 2022
1. kolo — rešitve nalog

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

14. april 2022

Skalarni produkt

Ker lahko elemente poljubno množimo z -1, se bomo osredotočili samo na njihove absolutne vrednosti, saj lahko elementom vedno spremenimo predznak tako, da se bo s svojim parom zmnožil v pozitiven rezultat. Ker iščemo poljubni vrstni red komponent drugega vektorja, lahko prvi vektor uredimo po komponentah od manjših proti večjim po absolutnih vrednostih. Zapomnimo pa si, od kod je posamezna komponenta prišla, da bomo lahko na koncu vrstni red rekonstruirali. Sedaj lahko opazimo, da je smiselno tudi drugi vektor urediti na enak način, da se velike vrednosti zmnožijo med seboj. To lahko dokažemo z razmislekom, kaj bi se zgodilo, če bi v neki neurejeni rešitvi popravili vrstni red dveh neurejenih elementov in ugotovimo, da se lahko rezultat kvečjemu izboljša. Tako smo ugotovili, kateri pari elementov naj se zmnožijo med seboj, sedaj pa samo še rekonstruiramo prvotni vrstni red.

Ladjice potapljat

Ločeno poiščemo vodoravne in navpične ladjice. Začetek vodoravne ladjice iščemo znotraj vrstic od leve proti desni z iskanjem dveh zaporednih zasedenih polj. Od tam naprej preštejemo število zaporednih zasedenih polj, da dobimo dolžino ladjice, hkrati pa morajo biti polja nad in pod ladjico prazna, sicer postavitev ni pravilna. Tako odkrito ladjico lahko izbrišemo s polja. Na enak način poiščemo tudi navpične ladjice. Izjema so še ladjice oblike 1x1, ki jih lahko obravnavamo ločeno. To so zasedena polja, ki imajo prosto mesto na vseh štirih sosednjih poljih. Sedaj samo primerjamo število pojavitev ladjic posamezne dolžine s podanim številom pojavitev.

Pravilo srečanja

Stanje na poti lahko opišemo s koordinatama x in y ter smerjo d. Smeri lahko oštevilčimo od 0 do 3 v smeri urinega kazalca. Iščemo torej najkrajšo pot v grafu z $n \cdot m \cdot 4$ stanji, kar lahko naredimo z enostavnim iskanjem v širino. Iz vsakega stanja se lahko premaknemo v tri smeri (polkrožno ne smemo obrniti v križišču), kar predstavimo s prištevanjem -1, 0 ali 1 (po modulu 4) trenutni smeri d. Če obstaja nasprotna povezava, ne smemo upoštevati zavijanja levo in nam ostaneta samo dve možni smeri. Seznam obstoječih povezav je smiselno hraniti v množici, da lahko enostavno testiramo, ali neka povezava obstaja, ko želimo zaviti v njeno smer. Ostane nam samo klasična implementacija iskanja v širino z uporabo vrste.

Razvijanje zank

Imena spremenljivk v posamezni vrstici lahko enostavno zamenjamo z njihovimi vrednostmi. To lahko storimo z iskanjem imena spremenljivke v vrstici ali pa si vse skupaj poenostavimo z uporabo regularnega izraza. Celotno vsebino programa obravnavamo po vrsticah. Izjema sta for zanka ter if-else pogojni stavek, kjer moramo najti vsebino pripadajočega bloka z uporabo zamika ukazov. Tako najdene bloke razvijemo rekurzivno z enakim postopkom. V primeru for zanke to storimo večkrat z uporabo različnih vrednosti zančne spremenljivke in zmanjšamo zamik rezultata, da nadomestimo for zanko z več kopijami bloka.

Fevdalci

Če poznamo delež x za začetek mobilizacije, lahko izračunamo čas mobilizacije. Za izračun časa mobilizacije fevdalca najprej izračunamo čase mobilizacije njegovih neposrednih vazalov. Te čase uredimo in ugotovimo, kdaj je mobiliziranih prvih x odstotkov njegovih neposrednih vazalov. K temu prištejemo čas mobilizacije fevdalca. Čase mobilizacije posameznih oseb lahko torej računamo od listov proti korenu drevesa. Opazimo lahko, da večjih kot je delež x, dlje časa bodo morali fevdalci čakati pred začetkom mobilizacije, kasneje bo kralj zaključil svojo mobilizacijo. Z bisekcijo poiščemo največji x, pri katerem kralj zaključi mobilizacijo do časa T. Težavo predstavljajo še ulomki. Ugotovimo, da bo rezultat oblike $\frac{a}{b}$, kjer je b število neposrednih vazalov nekega fevdalca, a pa je med 0 in b. Možne rezultate si pripravimo v naprej, jih uredimo po velikosti in z bisekcijo iščemo med njimi. Rezultat na koncu še okrajšamo.

Prečrtanka

V nalogi iščemo množico besed v besedilu. Težavo pa imamo, ker ne moremo za vsako besedo preiskati celega besedila, tudi če bi to naredili v linearnem času. Tudi če bi vsako pojavitev besede v besedilu našli v konstantnem času, je teh pojavitev lahko preveč. Učinkovitemu sočasnemu iskanju več vzorcev v nizu je namenjen algoritem Aho-Corasick. Gre za posplošitev algoritma Knuth-Morris-Pratt za iskanje več vzorcev hkrati. Aho-Corasick zgradi drevo vzorcev, s katerim lahko v času $O(n + m + k)$ poiščemo vse pojavitve vzorcev v besedilu. Pri tem je n dolžina besedila, m vsota dolžin vzorcev, k pa število pojavitev. Algoritem za vsako mesto v besedilu poišče vse vzorce, ki se zaključijo na tem mestu. Namesto naštevanja vseh, lahko algoritem priredimo, da vrne samo najdaljšega. Še vedno pa ne smemo kar prečrtati vseh mest v besedilu, saj so lahko pojavitve dolge in se med seboj prekrivajo. Namesto tega si pripravimo seznam intervalov in na koncu izračunamo njihovo unijo v linearnem času. Črke, ki ležijo izven unije, so iskani rezultat.