Domača naloga #9:
Priprave na računalniške olimpijade 2018/19

Filip Koprivec
filip.koprivec@student.fmf.uni-lj.si

A. Dungeons and Candies

Naloga je preprosta implementacija najmanjšega vpetega drevesa, ki pa je malo zakamuflirano. Precej hitro lahko ugotovimo, da stanja predstavljajo vozlišča, teža posamezne povezave med njimi pa je 'cena' pretvorbe iz enega v drug nivo. Zanima nas najcenejši način, kako lahko v igri zgeneriramo vsa stanja, kjer je ena izmed možnosti tudi to da stanje zgeneriramo na novo.

Če želimo priti v nivo a lahko nanj pridemo na dva različna načina. Lahko ga zgeniriramo na novo(z neko ceno), lahko pa ga 'popravimo' iz nekega novega stanja (z drugačno ceno). Ker želimo priti do vseh nivojev (in vsaj enega popolnoma zgenerirati), iščemo množico povezav, ki poveže vse nivoje (vsak nivo si predstavljamo kot vozlišče, cene generiranja novih nivojev pa kot povezave z ustrezno težo) in tudi nivo 0 (ki predstavlja začetno stanje, ko še nismo začeli z igro). Iščemo torej najcenejšo podmnožico povezav, ki poveže vsa vozlišča, ki pa je seveda najmanjše vpeto drevo.

Ko dobimo najmanjše vpeto drevo, je potrebno zgolj še poskrbeti, da vozlišča izpišemo v pravilnem vrstnem redu, pri čemer si lahko pomagamo z dejstvom, da izpis vedno začnemo v stanju 0 in izpisujemo drevo. Vprašanje za ponovitev: Kateri izmed algoritmov za MST se teoretično obnese bolje, glede na to, kako redek/gost je graf?

B. Swaps in Permutation

Zanima nas največja leksiografska permutacija danega niza (dejstvo da gre za permutacijo je povsem nepomembno), kjer lahko med seboj menjamo samo elemente na določenih indeksih. Če bi lahko vse menjali z vsemi, bi bila rešitev očitna -> permutacijo uredimo in to je že kar rezultat. Ker pa lahko med seboj menjamo samo določene indekse, si celoten niz predstavljamo kot graf, kjer so vozlišča indeksi posameznih črk (številk permutacije), povezave med njimi pa 'dovoljeni' prehodi. V tem nepovezanem grafu najdemo komponente (elegantno lahko uporabimo DSU, ali pa poljubno iskanje) in elemente vsake izmed teh komponent uredimo padajoče, da dobimo največjo leksikografsko ureditev (Ker v komponenti lahko med seboj menjamo posamezne elemente ni težko videti, da lahko z dovolj menjavami, pridemo do poljubnega vrstnega reda v komponenti).

Preostane nam le še da komponente zložimo skupaj. Za vsako komponento si zapomnimo, na katerih indeksih so nastopali njeni elementi in jih nato zložimo nazaj na pravilna mesta (glede na urejeno stanje).

C. The Child and Zoo

Če imamo graf s samo dvema vozliščema je rešitev očitna, $f(p, q)$ je kar minimum števila živali, ki so v kateremkoli izmed vozlišč. Kaj pa če imamo namesto dveh vozlišč dve (disjunktni/različni) komponenti grafa, med katerima je samo ena povezava? Prav bi nam prišlo, če bi lahko kaj povedali o ceni dodajanja te povezave glede na njeno ceno (kot ceno povezave si predstavljamo kar minimum cen vozlišč ne obeh koncih povezave, saj bo 'pot' čez njo gotovo imela za nimimum največ minimum enega izmed njih). Če je ta cena cenejša od vseh povezav v komponentah, potem so vse dodatne cene $f(p', q')$ (kjer sta p' in q' poljubni vozlišči iz različnih komponent) gotovo minimum od števila živali, ki jih obiščemo v p ali q. Cena povezovanja obeh komponent je v takem primeru: $2 \times \min(a_p, a_q) \times |P| \times |Q|$, kjer sta Q in P množici vozlišč posameznih komponent (v obeh smereh, prek nove povezave iz vsake v vsako). Da pa lahko zagotovimo moramo povezave obdelovati od največje do najmanjše, in preverjati, ali nova povezave poveže že povezani

komponenti. Pri tem si pomagamo s podatkovno strukturu DSU (v resnici iščemo največje vpeto drevo, kar preprosto dosežemo, da v Kruskalovem algoritmu povezave uredimo padajoče; namig, poglejte kako lahko to najhitreje napišete s pomočjo uporabe standardne knjižnice vašega jezika).

Poleg tega pa moramo za vsako trenutno komponento vedeti tudi kako velika je. Poleg starša tako za vsako vozlišče hranimo še velikost njegovega poddrevesa in ga od združevanju ustrezno posodobimo. Podobno kot pri DSU je pomembno, da je velikost poddrevesa pravilna zgolj za njegov koren, ostale pa pridobimo tako, da rekurzivno preverimo velikost njegovega nadrejenega.

Pri izpisu pazimo na zahtevano natančnost in celoštevilsko deljenje.