Univerzitetni programerski maraton 2020
1. kolo — rešitve nalog

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

16. april 2020

Ponaredki

Če potrebujemo X originalov, to pomeni $X - 1$ obiskov pri ponarejevalcu. Stranski učinek tega bo tudi $X - 1$ ponaredkov, torej mora veljati $Y >= X - 1$. Preostale ponaredke nam bo ponarejevalec izdelal v kosih po 2, zato mora biti $Y - (X - 1)$ deljivo z 2.

Posebna primera sta $X = 0$ in $X = 1$. Ker že začnemo z enim originalom, za $X = 0$ ni rešitve. Pri $X = 1$ pa še nimamo nobenega ponaredka, ki bi ga lahko razmnožili s ponarejanjem, zato je edina rešitev $Y = 0$. Ta dva primera sta bili podana v javnih testnih primerih, sicer bi bila naloga precej bolj frustrirajoča.

Pet v vrsto

Na kvadratno mrežo moramo postavljati simbole in preverjati, ali jih je slučajno 5 v vrsti (vodoravno, navpično, diagonalno). Malenkost se zaplete, ker je ta mreža zelo velika. Hranimo lahko seznam že postavljenih simbolov in ob postavitvi novega na mesto (r, c) preverimo, koliko simbolov se zapored razteza v vsaki od 8 smeri. Če jih je npr. skupaj v levo in v desno dovolj, smo našli vodoravno rešitev. Za levo smer bi preverjali, ali obstaja simbol na mestu $(r, c - 1)$, $(r, c - 2)$, itd. Ta operacija je na seznamu počasna (linearna). Lahko pa simbole hranimo v kakšni drugi strukturi s konstantno ali logaritemsko časovno zahtevnostjo, kot so npr. slovarji in množice (dict, map, HashMap, TreeSet, ...)

Druga možnost je, da vse simbole uredimo najprej po vrsticah, znotraj vrstice pa po stolpcih. Potem lahko za vsakih pet zaporednih simbolov preverimo, ali pripadajo istemu igralcu, so del iste vrstice in se držijo skupaj. Tako najdemo vodoravne rešitve. Podobno logiko uporabimo še za odkrivanje rešitev v stolpcih in na diagonalah.

Tikparske napake

Naloga je dokaj enostavna s stališča, kaj in kako narediti, je pa z njo kar nekaj dela. Najprej potrebujemo seznam sosednjih tipk. Lahko napišemo ločen program, ki nam seznam iz besedila preoblikuje v obliko, ki jo lahko uporabimo v našem programu. Nato moramo poiskati besede. Besedilo lahko procesiramo po znakih in sproti sestavljamo besede ali pa si pomagamo z regularnimi izrazi. Vzdržujemo lahko množico pravilnih besed in množico zatipkanih različic. Če nove besede ni med pravilnimi, jo poskusimo poiskati med zatipkanimi. Če je ni tudi tam, je to nova beseda in jo dodamo v množico pravilnih, vse njene zatipkane različice pa v množico zatipkanih. Kapitalizacija je neodvisna od tipkarskih napak, zato lahko popravljeni besedi pred izpisom priredimo še zahtevano kapitalizacijo, ki jo prenesemo iz nepopravljene. Podani primeri so bili kar obsežni, zato pride prav kakšno orodje za primerjanje vsebine vašega izpisa in pravilne rešitve.

Trgovec s kriptovalutami

Vsak dan se nam splača investirati v največ eno valuto. Če bi nek dan investirali del denarja v A, del pa v B, bi lahko pogledali, iz katerega dela v končni rešitvi dobimo več in raje vse vložili vanj. Drugo pomembno

dejstvo je, da lahko vedno investiramo samo do naslednjega dne, ko lahko trgujemo s to valuto. Ker ni provizij, lahko namreč še isti dan vložimo denar nazaj v isto valuto, če želimo vložek držati dlje.

Naj $f(d)$ predstavlja faktor, s katerim se ob optimalnem trgovanju pomnoži naš vložek, če začnemo s trgovanjem na dan d. Zadnji dan ne moremo več povečati vložka, torej je $f(n) = 1$. Sicer pa lahko vsak dan ne naredimo ničesar, ali pa vložimo denar v eno od valut, ki so na voljo tisti dan, ter jo prodamo ob prvi možnosti na dan d'. Dneve d' lahko izračunamo v naprej, ali pa jih iščemo sproti. V obeh primerih bomo vse skupaj potrebovali $O(n)$ operacij.

$$f(d) = \max \begin{cases} f(d+1) & \text{ne vlagamo na ta dan} \\ \max_k(v_k(d')/v_k(d) \cdot f(d')) & \text{vložimo v valuto } k\text{, če je na voljo, torej } v_k(d) \neq -1 \end{cases}$$

Plovba

V osnovi iščemo optimalno plovbo po celotni obali jezera. Ob izboru neke direktne poti, pa se nam nabor razpoložljivih pristanišč vedno bolj krči na nek strnjen interval vzdolž obale. Definirajmo podproblem s trojico (s, e, d), ki pomeni, da začnemo v pristanišču s in imamo na razpolago pristanišča do e, če se premikamo za $d \in \{-1, 1\}$; torej $\{s, s+d, s+2d, ..., e-d, e\}$. Preverimo vse možne plovbe iz s na x na tem intervalu in dobimo dva podproblema, lahko iz x obrnemo proti s, ali pa nadaljujemo proti e.

$$f(s, e, d) = max_x(c_{s,x} + max(f(x, s+d, -d), f(x, e, d)))$$

S takim dinamičnim programiranjem dobimo rešitev s časovno zahtevnostjo $O(n^3)$, kar je prepočasi. Opazimo pa lahko, da so si podproblemi precej podobni. Razlika med $f(s, e, d)$ in $f(s, e+d, d)$ je samo v enem x-u, ki ga moramo dodatno upoštevati. Obravnavajmo primere, glede na to, kaj se zgodi z zadnjim pristaniščem e. Lahko ga sploh ne uporabimo. Sicer pa bomo naredili nekaj direktnih plovb v isti smeri, preden pridemo z zadnjo do e. Ker je cena potovanja ob obali vsaj tako draga kot direktna bližnjica, se nam splača vse razen zadnjega skoka na e opraviti ob obali. Naredimo torej korak naprej ob obali, ali pa plujemo direktno do e. Ta rešitev ima časovno zahtevnost $O(n^2)$. Paziti moramo samo na cikličnost indeksov.

$$f(s, e, d) = max \begin{cases} f(s, e-d, d) & \text{zadnjega pristanišča sploh ne uporabimo} \\ c_{s,e} + f(e, s+d, -d) & \text{direktna plovba na zadnjega} \\ c_{s,s+d} + f(s+d, e, d) & \text{naredimo en korak ob obali} \end{cases}$$

Koledar

Ploščino nekonveksnega večkotnika lahko izračunamo kot vsoto predznačenih ploščin trapezov, ki se raztezajo pod vsako stranico. Ker bomo rezali koledar na različnih višinah, nam v tem primeru bolj prav pride seštevanje trapezov, ki se raztezajo v levo. $P = \sum_i (y_{i+1} - y_i) \cdot (x_i + x_{i+1})/2$. Za vsako stranico lahko izračunamo njen prispevek. Paziti moramo tudi na negativno orientacijo večkotnika ($P < 0$).

Poizvedbe je smiselno urediti in računati odgovore po vrsti, saj je rešitev s časovno zahtevnostjo $O(nq)$ prepočasna. Koledar lahko razrežemo na vodoravne pasove na vseh različnih višinah, ki se pojavijo v vhodnih podatkih. Za vsak pas moramo izračunati prispevek delov večkotnikov, ki se pojavijo v njem. Daljice se tipično raztezajo čez več pasov. Če poznamo spodnje oglišče i-te stranice (x_1, y_1), lahko površino trapeza, ki se razteza od tam do neke druge višine y ($y_1 \leq y \leq y_2$) zapišemo s kvadratno funkcijo oblike $f(y) = Ay^2 + By + C$. Ob premikanju po pasovih vzdržujemo vsoto teh funkcij za daljice, ki se pojavijo v trenutnem pasu. Daljice je enostavno dodati in odstraniti. Doprinos trapezov v trenutnem pasu od Y_1 do Y_2 je torej enak $F(Y_2) - F(Y_1)$, kjer je $F(Y) = \sum f(Y)$ enak vsoti funkcij aktivnih daljic.