Univerzitetni programerski maraton 2021
1. kolo — rešitve nalog

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

15. april 2021

Oglaševanje

Začnimo s primerom $N = 1$, ko imamo samo en televizijski kanal s T termini. Poglejmo, kakšen dobiček od oglaševanja lahko dosežemo, če ne bi bilo pristopnih stroškov X. V vsakem terminu je smiselno oglaševati, če so pričakovani prihodki večji od stroškov oglaševanja v tistem terminu. Zaslužili bi torej $D' = \sum_{j=1}^{T} max(P_j - S_j, 0)$. Če se odločimo za oglaševanje, moramo temu odšteti še pristopne stroške, če niso preveliki. Dobiček enega kanala je torej $D = max(D' - X, 0)$. Oglaševanje na različnih kanalih je med seboj neodvisno, zato lahko v primeru več kanalov dobičke po kanalih enostavno seštejemo.

Periode

Kako pretvorimo periodično decimalno število v ulomek? Vzemimo za primer število $x = 12{,}345\overline{67}$. Število primerno zamaknemo in odštejemo, da se znebimo periode.

$$\begin{aligned} 1000x &= 12345{,}\overline{67} \\ 100000x &= 1234567{,}\overline{67} \\ \hline 99000x &= 1222222 \\ x &= 1222222/99000 \\ x &= 12\,\frac{17111}{49500} \end{aligned}$$

Poiskati moramo torej začetek in dolžino periode. Števila so dovolj kratka, da lahko opravimo vse izračune s 64-bitnimi celimi števili. Ulomek okrajšamo z največjim skupnim deliteljem, ki ga izračunamo z Evklidovim algoritmom, ali pa to delo prepustimo kakšni funkcije iz standardne knjižnice vašega programskega jezika. Pozorni moramo biti na vse robne primere (npr. 0), ki pa jih dobro pokriva že podani primer vhodnih podatkov.

Drsna sestavljanka

Najmanjše število potez za rešitev posamezne sestavljanke lahko poiščemo z iskanjem najkrajše poti v grafu stanj, kjer vsako vozlišče grafa ustreza stanju sestavljanke (razporedu števil v mreži). Za iskanje najkrajše poti uporabimo kar iskanje v širino, ker povezave niso uteženе oz. dolžino poti predstavlja kar število povezav. Sočasno z iskanjem najkrajše poti lahko izračunamo tudi število teh poti. Stanje opišemo z 2D tabelo ali pa ga predstavimo kar z 9-mestnim celim številom, kot bi ga prebrali po vrsticah od zgoraj navzdol in znotraj vrstic od leve proti desni (prazno mesto lahko nadomestimo npr. z 0). Že obiskana stanja hranimo v slovarju (`map`, `TreeMap`, `HashMap`, `dict`, `...`), ki omogoča vstavljanje in iskanje v logaritemskem ali konstantnem času.

Ker nas zanima več najkrajših poti do istega urejenega stanja, je smiselno na začetku izračunati najkrajše poti od urejenega stanja do vseh ostalih, nato pa samo izpisovati odgovore na poizvedbe.

Sestop

Za izračun iskane K-te poti bi nam prav prišlo število sestopov $f_{i,j}$ iz celice v i-ti vrstici in j-tem stolpcu do najnižje celice. Če poznamo te vrednosti, lahko zgradimo K-to pot od najvišje proti najnižji. Na vsakem koraku obravnavamo premike na jug, sever, vzhod in zahod (kot si sledijo po abecedi) in opazujemo, koliko poti obstaja v določeni smeri.

Vrednosti $f_{i,j}$ lahko računamo od najnižje proti najvišji celici (celice v mreži uredimo glede na njihovo višino). Število poti z neke celice je enako vsoti števila poti s sosednjih nižjih celic, ki pa so že izračunane. Pazljivi moramo biti, ker lahko te vrednosti presežejo 64-bitna števila. Vrednosti večje od K, nas ne zanimajo, zato jih lahko nadomestimo kar s K in ostanemo v 64-bitnih številih.

Mikrobiologija

Za simulacijo dogajanja moramo učinkovito računati maksimum in vsoto (za izračun povprečja) na različnih intervalih tabele N števil. To lahko dosežemo z uporabo statičnega drevesa (*segment tree*), ki ga zgradimo nad tabelo. Vsako notranje vozlišče hrani podatek o maksimumu in vsoti na pripadajočem intervalu tabele.

Poleg poizvedb pa izvajamo tudi spremembe tabele na intervalih: množenje z 2, prištevanje in nastavljanje na dano vrednost. Nastavljanje vrednosti lahko simuliramo z zaporedjem operacij množenja z 0 in prištevanjem. Izračunane vrednosti maksimuma in vsote v notranjih vozliščih lahko enostavno popravimo ob množenju ali prištevanju. Če bi imeli opravka samo s prištevanjem, bi lahko spremembe označevali v notranjih vozliščih drevesa, ki pokrivajo interval, na katerega se nanaša sprememba. Ko pa dodamo tudi množenje, postane pomemben vrstni red operacij. To rešimo z uporabo standardne tehnike *lazy propagation*, kjer spremembe po potrebi propagiramo iz vozlišča globlje po drevesu in s tem ohranjamo vrstni red operacij. Časovna zahtevnost rešitev je $O(N + M \log N)$ s precejšnjim konstantnim faktorjem.

Cepljenje

Stanje lahko modeliramo z grafom, v katerem vozlišča predstavljajo osebe, povezave pa kontakte med njimi. V nalogi iščemo minimalno število vozlišč, z odstranitvijo katerih ločimo dve skupini vozlišč (okužene in ranljive). Ločevanje vozlišč oz. razpad na ločene komponente je povezan s konceptom prerezov (*cut*) v grafih. Vsa okužena vozlišča lahko povežemo s izvorom S, vsa ranljiva vozlišča pa s ponorom T. Pri prerezih običajno odstranjujemo povezave in ne vozlišča. Problem lahko prevedemo na odstranjevanje povezav z modeliranjem vsakega originalnega vozlišča s pomožnim vhodnim in izhodnim vozliščem, ki sta med seboj povezana z usmerjeno povezavo. Minimalen S-T prerez (*minimum s-t cut*) bo v tem primeru množica povezav med pomožnimi vhodnimi in izhodnimi vozlišči. Prerez poiščemo z uporabo maksimalnega pretoka (*max flow*) v grafu. Dovolj učinkoviti so že najbolj osnovni algoritmi kot sta npr. Ford-Fulkerson ali Edmonds-Karp. Največja možna vrednost pretoka v tem grafu namreč ne presega števila vozlišč.