Domača naloga #8: Najkrajše poti v grafih
Priprave na računalniške olimpijade 2018/19

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

A. Dijkstra?

Naloga pred nami je preprosta implementacija Dijkstra algoritma. V knjižnici svojega programskega jezika najdemo ustrezno implementacijo vrste s prednostjo (`priority_queue` v C++, `heapq` v Pythonu, `PriorityQueue` v Javi). Pri vhodu pazimo, da gre za neusmerjen graf (povezave gredo v obe smeri) in na to, da lahko dolžine povezav presežejo `int`. Ko imamo zgrajeno tabelo predhodnikov jo samo prehodimo in izpišemo v obrnjenem vrstnem redu.

B. Roads in Berland

Podane imamo najkrajše povezave med vsemi mesti. Zanima nas, kako se ob dodatku nove povezave spremenijo najkrajše povezave med vsemi pari mest. Če bi v vsakem koraku na novo izračunali vse najkrajše povezave (FW: n^3, ali večkratna Dijkstra: $n^3 \log n$) bi porabili preveč časa $kn^3 \approx 8 * 10^9$. Zato poskušamo izoristiti dejstvo, da imamo pred vsakim dodajanjem nove povezave že pravilno izračunane najkrajše povezave med vsemi pari.

Dodajanje nove povezave uv je seveda smiselno zgolj če je cenejša kot že obstoječa. V tem primeru si lahko novo povezavo predstavljamo kot povezavo, ki jo Bellman-Ford pregleda v zadnjem koraku in ustrezno posodobimo najkrajše povezave. Ker nas zanimajo najkrajše povezave med vsemi pari, za vsako vozlišče s izvedemo zadnji korak algoritma: Za vsako preostalo vozlišče a preverimo, ali je pot preko novo dodane povezave cenejša kot do sedaj obstoječa pot, torej ali je $|su| + |ua| < |sa|$ in enako še za v. Skupno za vsako dodano pot opravimo približno n^2 operacij, kar na koncu znese $kn^2 = 2.7 * 10^7$

Na koncu vsakega koraka preprosto seštejemo najdaljše povezave med vsemi pari in pazimo da povezav ne štejemo večkrat. Brez težav pa lahko sprotno vsoto posodabljamo kar v popravljanju najcenejših poti.

C. Jzzhu and Cities

Zanima nas, katere izmed železniških povezav lahko odstranimo, če želimo še vedno ohraniti cene najcenejših poti od prestolnice. Prvo manjšo težavo predstavlja območje cen, kjer moramo paziti, da uporabljamo dovolj velik podatkovni tip (recimo: `long long`). Ključna opazka, ki nam pomaga, je dejstvo, da so vse železniške povezave zgolj v smeri od prestolnice do mest in ne med poljubnimi mesti.

Pomagamo si z manjšo modifikacijo Dijkstre. Najkrajše poti od prestolnice iščemo popolnoma običajno, le pri vsakem koraku si za posamezno vozlišče namesto predhodnega vozlišča zapomnimo povezavo, po kateri smo do novega vozlišča prišli (v resnici je dovolj, da si to zapomnimo zgolj za vozlišča do katerih smo prišli po železniških povezavah, pri ostalih je dovolj, da si zgolj zapomnimo, da smo do njih prišli po cesti).

Na koncu se sprehodimo po predhodnikih vseh vozlišč in preverimo, koliko različnih železnic nikoli ni udeleženih kot predhodna povezava. Take železnice lahko odstranimo, saj je najkrajša pot tudi brez njih še vedno enaka. Dejstvo zakaj jih lahko odstranimo sledi iz vrstnega reda procesiranj, v algoritmu sproti 'popravljamo' najkrajše poti od začetka do ostalih vozlišč. Če je med dvema vozliščema železniška povezava, a ni predhodnik v najkrajši poti med njima pomeni da najkrajša pot med njima (in poti, ki to vozlišče vsebujejo) gotovo ne potrebuje železnice. Paziti moramo še na dejstvo, da če sta dve povezavi enako dolgi, potem raje vzamemo cestno, železniško pa lahko odrežemo.