Domača naloga #2: osnovne podatkovne strukture
Priprave na računalniške olimpijade 2018/19

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

A. Misha and Changing Handles

Recimo, da bi morali izpisati samo seznam uporabniških imen, ki so zasedena na koncu preimenovanj. Vzdrževati bi morali seznam imen, ki so trenutno v uporabi. Ob preimenovanju poiščemo staro ime, ga izbrišemo in dodamo novo. Omejitve so dovolj nizke, da lahko vsakič iščemo po celem seznamu. Če želimo rešitev pohitriti ali skrajšati, pa bi lahko uporabili množico (*set* v standardni knjižnici C++), ki omogoča točno te operacije z logaritemsko časovno zahtevnostjo namesto linearne.

V nalogi pa nas zanima tudi originalno uporabniško ime. Poleg vsakega imena, ki je v uporabi, moramo hraniti tudi pripadajoče originalno ime. To lahko storimo s seznamom parov ali pa uporabimo *map*, ki je namenjen prav temu.

B. Cd and pwd commands

Naloga zahteva simuliranje sprehajanja po datotečni strukturi. Prva ovira je branje podatkov. Pri operaciji `cd` nam prav pride *split*, ki v C++ žal ni tako trivialna operacija kot npr. v Pythonu ali Javi.

Vzdrževati moramo trenutno lokacijo v datotečni strukturi, ki jo lahko opišemo s seznamom map in podmap. Na konec tega seznama moramo dodajati nove podmape ali pa jih s konca brisati (ko naletimo na `..`). Za te potrebe je najbolj primeren *stack*; v C++ pa lahko uporabite tudi *vector* z operacijama *push_back* in *pop_back*.

C. Queue

Zadnja naloga je že bolj algoritmična za razliko od prejšnjih dveh, ki sta bolj tehnični. Za vsakega mroža v vrsti namreč ne moremo preveriti vseh, ki so pred njim (na desni), ker bi to zahtevalo $O(n^2)$ operacij, kar je za $n = 10^5$ preveč.

Smiselno je, da rešujemo problem od mroža na začetku vrste (desno) proti koncu (levo). Recimo, da smo izračunali odgovor za i-tega mroža v vrsti. Pri računanju odgovora za $(i + 1)$-tega se bomo namreč ukvarjali s podobno množico mrožev (ki so pred njim) kot pri i-tem.

Se moramo res ukvarjati z vsemi desnimi mroži, ali lahko kaj izboljšamo? Vzemimo prvi primer iz naloge. Pri računanju nezadovoljstva 5. mroža v vrsti moramo upoštevati mrože na desni: 5, 3, 50, 45. Brez škode lahko ignoriramo mroža s starostjo 5 in 50, ker bolj desno stoji še mlajši mrož, ki bo vir večjega nezadovoljstva pri 5. mrožu in tudi vseh nadaljnjih (bolj levih). Ukvarjati se moramo torej samo s seznamom mrožev, kjer starosti naraščajo proti desni (novega dodamo na levo stran seznama, samo če je mlajši).

V opisanem seznamu moramo poiskati najbolj desnega, ki je še mlajši od trenutnega mroža. Če seznam hranimo npr. v *set*-u, lahko izkoristimo drevesno strukturo, ki nam omogoča izvajati zahtevane operacije v logaritemskem času. Nove elemente dodajamo z metodo *insert*, poizvedbe pa izvajamo z metodo *lower_bound*.

Alternativna rešitev računa odgovore od najmlajšega proti najstarejšemu mrožu. Najmlajši ne more biti nezadovoljen. Nezadovoljstvo starejših pa je odvisno od najbolj oddaljenega mlajšega mroža (te smo že obravnavali). Vzdrževati moramo torej samo najbolj desno pozicijo že obravnavanega mroža. Ta bo namreč najbolj kritičen za vse sledeče. Paziti moramo samo na pravilno obravnavo enako starih mrožev. Ker moramo mrože urediti po starosti, je časovna zahtevnost te rešitve enaka prejšnji.