Opisi algoritama

Zadatke, test primjere i rješenja pripremili: Ante Đerek i Ivan Katanić. Primjeri implementiranih rješenja su dani u priloženim izvornim kodovima koji nužno ne odgovaraju u svim detaljima ovdje opisanim algoritmima.

Zadatak: Mao

Predložio: Ante Đerek
Potrebno znanje: naredba `if`, `for` petlja, stringovi

Zadatak rješavamo direktnom implementacijom opisane logike. Najprije učitamo kartu s vrha gomile te je pohranimo u string. Nakon toga pomoću `for` petlje čitamo ostalih pet karata, svaku uspoređujemo s kartom na vrhu gomile pomoću jedne `if` naredbe te ispisujemo odgovarajuću poruku. Prvom odnosno drugom znaku stringa pristupamo primjerice pomoću uglatih zagrada.

Zadatak: Traka

Predložio: Ante Đerek
Potrebno znanje: `for` petlja, simulacija

Najprije je potrebno razmisliti kako na najbolji način odabrati početnu boju trake. Jedan pristup je da za početnu boju odaberemo onu koja se najčešće pojavljuje u uzorku, ali se iz brzo vidi da ta strategija ne vodi do optimalnog rješenja. Najjednostavnije je izračunati cijenu bojanja *za svaki odabir početne* trake te jednostavno ispisati najmanju cijenu. Dakle, s jednom `for` petljom prolazimo kroz sve boje, zovemo funkciju koja za zadani uzorak i početnu boju računa vrijeme bojanja i pamtimo najmanje vrijeme.

Funkcija koja računa vrijeme bojanja direktno implementira pravila iz teksta zadatka. Jednom `for` petljom prolazimo kroz uzorak, a u varijablama pamtimo boju zadnje obojenog kvadratića, broj obojanih kvadratića od zadnjeg čišćenja te ukupno vrijeme.

Zadatak: Craft

Predložio: Ante Đerek
Potrebno znanje: stringovi, queue, pretraživanje u širinu, pohlepni algoritam

Jedan pristup rješavanju zadatka je da krenemo od inventara, probamo primijeniti svako od pravila te dobijemo nove inventare, onda na te nove inventare probamo primijeniti pravila te taj postupak ponavljamo sve dok ne dobijemo niti jedan novi inventar. Dakle, pretražujemo u širinu prostor koji se sastoji od svih mogućih inventara koji se mogu dobiti od početnog. Obzirom da primjena recepta smanjuje veličinu inventara, prostor koji pretražujemo je konačan. Međutim, prostor je još uvijek jako velik pa stoga nije lagano (ali je moguće) ovaj pristup iskoristiti kako bi se dobilo dovoljno efikasno rješenje.

Drugi pristup je da za svaki mogući tip c pokušamo krenuti od jednog predmeta tog tipa te pretraživanjem *unazad* odrediti je li moguće da ga dobijemo iz inventara. Pritom će za efikasnost postupka biti nužno da za svaki tip postoji najviše jedan recept. Koristimo dvije liste predmeta: I je trenutni inventar, a R je popis predmeta koje je potrebno izraditi kako bi dobili barem jedan predmet tipa c. Na početku, I odgovara početnom inventaru, a R sadrži samo jedan predmet tipa c. Algoritam ponavlja sljedeće korake:

1. Za svaki predmet d iz liste R:
   (a) Ukoliko postoji d u inventaru I onda ukloni d i iz I i iz R.
   (b) Inače, pronađi recept za d te ukloni d iz R i dodaj sve predmete s desne strane recepta u R.
2. Ukoliko je lista R prazna onda je moguće dobiti predmet tipa c.
3. Ukoliko lista R sadrži više predmeta nego lista I nije moguće dobiti predmet tipa c.

Zadatak: Glasovi

Predložili: Ante Đerek
Potrebno znanje: `for` petlja, simulacija, sortiranje

Umjesto da određujemo jednog po jednog kandidata koji ulaze u parlament, možemo *sortirati* sve kandidate po prioritetu pa jednostavno označiti prvih m. Sortiranje možemo napraviti bilo kojim standardnim algoritmom (selection sort, bubble sort) ili primjenom funkcija iz standardnih biblioteka programskog jezika. Prilikom sortiranja uspoređujemo dva kandidata A i B i naprijed stavljamo onoga koji ima veći *prioritet* prema sljedećim pravilima:

- Ako su oba kandidata preferirana onda prioritet ima onaj s većim brojem glasova (odnosno onaj s manjom oznakom ako oba kandidata imaju jednak broj glasova).
- Ako je samo jedan kandidat preferiran onda on ima veći prioritet.
- Ako niti jedan kandidat nije preferiran onda prioritet ima onaj s manjom oznakom.

Zadatak: Trokut

Predložio: Ante Đerek
Potrebno znanje: `for` petlja, graf, stringovi, polja

Rješenje možemo podijeliti na dva dijela: Najprije odredimo listu svih imena, sortiramo je uzlazno po abecedi te graf prijateljstava spremimo u obliku dvodimenzionalnog polja — $g[a][b]$ je 1 ako je a-ti po redu korisnik prijatelj s b-tim po redu korisnikom. U drugom dijelu rješenja za svakog korisnika računamo potencijal direktnom simulacijom postupka opisanog u tekstu zadatka.

Postoji nekoliko načina da se prvi dio rješenja efikasno implementira. Možemo koristiti napredne strukture podataka poput `set` ili `map` u C++-su odnosno `set` i `dict` u Python-u za pohranjivanje imena. Također, imena možemo pohraniti u običan niz kojeg sortiramo te izbacimo duplikate. Nakon što su sortirana imena pohranjena u nizu, jednostavno za svako ime odredimo njegov redni broj te konstruiramo polje g.

Drugi dio rješavamo pomoću nekoliko `for` petlji. Najprije, moramo odabrati korisnika A za kojeg računamo potencijal. Nakon toga, kopiramo polje g u novo polje h koje će predstavljati graf prijateljstava nakon što povežemo prijatelje od A. Sada s dvije `for` petlje povežemo svaka dva prijatelja od A te s tri `for` petlje prebrojimo trokute u novom grafu.

Zadatak: Papir

Predložio: Ante Đerek
Potrebno znanje: dinamičko programiranje

Zadatak rješavamo dinamičkim programiranje, ovdje opisujemo rješenje koje nije najefikasnije, ali je još uvijek dovoljno efikasno da riješi zadatak u zadanim ograničenjima te se dade nešto jednostavnije opisati.

Za niz znakova s, i znak c označimo sa $F(s, c)$ najmanji broj koraka potreban da traku s obojimo cijelu u boju c. Primjerice $F(\texttt{aabbaaa}, \texttt{a})$ je očigledno 1 dok je $F(\texttt{aabbaaa}, \texttt{d})$ jednako 2. Razmatramo zadnje bojanje u tom optimalnom postupku bojanja te razlikujemo nekoliko slučaja za prethodni izgled trake s':

1. s' je oblika $xxxxxxxx$ za neku boju x.
2. s' je oblika $ccccxxxx$ za neku boju x.
3. s' je oblika $xxxxxcccc$ za neku boju x.
4. s' je oblika $cccxxxccc$ za neku boju x.

Možemo zaključiti sljedeće: Optimalni postupak je ili da najprije cijelu traku obojimo u proizvoljnu boju x pa zatim još jednim korakom u boju c (slučaj 1) ili da traku podijelimo u dva dijela $s = s_1 s_2$, obojimo

jedan dio u c, drugi u proizvoljnu boju x te još jednim korakom cijelu traku obojimo u boju c (slučajevi 2, 3 i 4). Pomoću ove logike možemo direktno izgraditi rekurzivne relacije za F te implementirati rješenje zadatka dinamičkim programiranjem. Jedan tehnički detalj je da $F(s, c)$ ovisi o $F(s, x)$ koja opet ovisi od $F(s, c)$ pa dolazi do cirkularnosti koju je potrebno pažljivo razriješiti. Detalje implementacije ostavljamo čitatelju.

Za one koji žele više: Osmislite rješenje koje radi za trake duljine 1000 znakova. Možete li osmisliti rješenje za trake duljine 10^5 znakova?