

Zadatke, test primjere i rješenja pripremili: Alen Rakipović <alen.rakipovic@gmail.com>, Goran Gašić <goran.gasic@gmail.com>, Ivo Sluganović <ivo.sluganovic@gmail.com>, Tomislav Gudlek <tgudlek@gmail.com>, Ivan Mandura <ivan.mandura93@gmail.com>, i Ante Đerek <ante.derek@gmail.com>.

Primjeri implementiranih rješenja su dani u priloženim izvornim kodovima koji nužno ne odgovaraju u svim detaljima ovdje opisanim algoritmima.

AUDIO

Predložio: Alen Rakipović

Potrebno znanje: simulacija, jednodimenzionalna polja, cijelobrojno dijeljenje, osnovne petlje

Zadatak se rješava direktnom simulacijom odnosno implementacijom postupka sažimanja koji je opisan u tekstu zadatka - u svakom koraku nalazimo jedan blok te ispisujemo njegov sažetak.

Krenuvši od početka niza, potrebno je naći najduži blok uzastopnih sljedbenika, redom od trenutne pozicije u nizu (sljedbenici su brojevi koji su za točno jedan veći od prethodnog). Na primjer, ako je zadan niz (1, 2, 3, 2) onda će prvi blok biti (1, 2, 3), a ako je zadan niz (2, 1, 2), onda će prvi blok biti samo (2). Blokove nalazimo jednom for petljom koja prolazi kroz cijeli niz:

- Označimo prvi element kao početak i jedini element trenutnog bloka
- Prolazimo kroz niz
 - Ako je trenutni element sljedbenik zadnjeg u trenutnom bloku, dodajemo ga u trenutni blok i nastavljamo
 - Inače, ispisujemo trenutni blok, te počinjemo novi blok i u njega stavljamo trenutni element

Pritom, naravno, nije potrebno pamtit i cijeli blok, već samo prvi i zadnji element u njemu.

Računanje sažetka bloka je također direktna simulacija postupka opisanog u zadatku. Broj znamenaka broja (odnosno same znamenke) možemo naći tako da unutar petlje ponavljamo postupak uzimanja ostataka pri djeljenju sa 10, te potom podijelimo dani broj sa 10 - tako jednu za drugom dobivamo dekadatske znamenke odostraga. Pametna implementacija će ovaj dio napisati u posebnoj funkciji koju će onda pozivati unutar for petlje. Kada smo prvi i zadnji broj u bloku rastavili na znamenke nastavljamo direktno kako je opisano u zadatku:

- Ako im je broj znamenaka različit ili počinju sa različitom prvom znamenkom onda direktno ispisujemo rezultat.
- Inače, jednom for petljom tražimo prefix te ispisujemo prefiks i oba repa. Prilikom ispisa repa potrebno je ispisivati znamenku po znamenku odnosno obratiti pažnju kako bi bile ispisane eventualne vodeće nule.

Za one koji žele više

Popularne metode sažimanja audio signala (na primjer MP3 format) rade tako da najprije 'malo promjene' ulazni signal pa ga tek onda sažimaju. Ideja je da se nakon malih, uhu neprimjetnih, promjena signal može puno bolje sažeti. Ako je dozvoljeno svaki element zadanog signala promijeniti za najviše 10 (plus ili minus) naći najkraće sažimanje takvog, promjenjenog, signala. Duljina sažimanja je ukupni broj znakova u sažetku (uključujući znakove za prijelaz u novi red).

SVEMIR

Predložio: Ivo Sluganović

Potrebno znanje: pretraživanje u širinu (BFS), red (queue)

Ovaj zadatak se rješava primjenom algoritma pretraživanja u širinu (BFS) implementiranog korištenjem strukture podataka red (queue).

Napravimo strukturu **Događaj**, koja u sebi enkapsulira podatke o jednom sletanju, a to su koordinate, vrijeme i rasa. Sortirajmo sva sletanja po vremenu i simulirajmo stanje na ploči kroz **K** koraka. Održavat ćemo dva reda (queue-a) koordinata, **trenutni_red**, i **sljedeci_red**. U redu **trenutni_red**, nalaze se sve točke tj. parovi koordinata čije će se eventualno preuzimanje dogoditi u ovom koraku, **sljedeci_red** jest red koordinatnih točaka čije će se eventualno preuzimanje dogoditi u sljedećem koraku. Za svaku točku (**x**, **y**) održavamo **napadaju[x][y]** -listu rasa koje je napadaju u trenutnom koraku, a za svaku rasu pamtimo koliko je točaka zauzela do ove sekunde.

Svake korak simuliramo u sljedeće tri faze:

- 1) Obavimo sva sletanja koja se dešavaju u tom koraku. Koordinate sletanja stavimo u **trenutni_red**.
- 2) Prolazimo kroz **trenutni_red** te za svaku točku iz trenutnog reda računamo pobjednika na način da iz liste **napadaju[x][y]** uzmemo rasu koja ima najviše zauzetih pozicija do ove sekunde. Ukoliko točka (**x**, **y**) biva zauzeta (ne postoje dvije rase sa jednako zauzetih pozicija do ove sekunde), u **sljedeci_red**, dodajmo susjedne točke koje mogu biti zauzete.
- 3) Zamijenimo **trenutni_red** i **sljedeci_red**.

Primjetimo da ćemo neku točku (**x**, **y**) staviti u red maksimalno 4 puta, odnosno lista napadaju će uvijek imati maksimalno 4 elementa. Također, primijetimo najprije da ćemo napraviti najviše **N*M** izmjena na mapi jer se svaki od **N*M** pojedinih kvadratića može izmijeniti najviše jednom. Stoga je algoritam je vremenske složenosti $O(N*M + K)$, što je dostatno za sve bodove na ovom zadatku.

Za one koji žele više

Prilagodi algoritam tako da brzina izvršavanja ne ovisi o **K**. Nađi implementaciju koja koristi samo jedan umjesto predložena dva reda.

HAKERI

Predložio: Goran Gašić

Potrebno znanje: dinamičko programiranje, binarno pretraživanje

Riješimo najprije jednostavniji problem bez šumova. Ovo lako ostvarujemo rekursivnim algoritmom koji za zadani sufiks presretnute poruke određuje može on li se rastaviti na riječi iz rječnika te maksimalan broj riječi kojima je to moguće ostvariti. Za neki sufiks presretnute poruke dovoljno je promotriti njegove prefikse te za svaki koji se pojavljuje kao riječ u rječniku (pohranjenom kao niz stringova) rekursivno pozvati funkciju na ostatku sufiksa. Konačno, uzimamo rastav s maksimalnim brojem riječi. Brojčano rješenje dobivamo pozivanjem opisane funkcije na cijeloj presretnutoj poruci. Kako bismo odredili konkretan rastav, za svaki sufiks pamtimo riječ koja nas je dovela do najboljeg rješenja. S tim informacijama lako možemo rekonstruirati najbolji rastav poruke.

Ako s **L** označimo maksimalnu duljinu riječi u rječniku, ovo rješenje ima vremensku složenost $O(N! * L^2 * M)$ te nosi 30% bodova. Rekursivan algoritam možemo ubrzati memoizacijom

(pamćenjem brojčanog rezultata za svaki sufiks na kojemu pozivamo funkciju) ili prevođenjem u rješenje dinamičkim programiranjem. Time vremensku složenost spuštamo na $O(N * L^2 * M)$ te ostvarujemo sljedećih 30% bodova.

Uvođenjem šuma potrebno je neznatno promijeniti postojeći algoritam. Funkcija osim sufiksa treba imati još dva parametra: broj preostalih šumova te je li prethodan niz znakova bio šum. Osim promatranja prefiksa zadanog sufiksa, dodatno će provjeriti ostvarujemo li najbolji rastav ako je prvi znak sufiksa šum, rekursivnim pozivanjem funkcije na ostatku sufiksa sa smanjenim brojem preostalih šumova ako prethodni niz znakova nije bio šum.

Ovime uz vremensku složenost $O(N * K * L^2 * M)$ ostvarujemo sljedećih 20% bodova. Za posljednjih 20% bodova potrebno je na brži način napraviti provjeru da li se riječ nalazi u rječniku (da se u gornjem produktu riješimo broja M). Jedan način da se to napravi je da rječnik pohranimo kao niz sortiranih riječi, na kojemu radimo upite binarnim pretraživanjem čime ostvarujemo složenost $O(N * K * L^2 * \log M)$. Alternativno, rječnik možemo pohraniti u neku od naprednih struktura podataka kao što su hash tablica ili balansirano binarno stablo (npr. map ili set u jeziku C++).