

Zadatke, test primjere i rješenja pripremili: Frane Kurtović, Tomislav Gudlek, Ivan Katanić, Stjepan Glavina, Goran Žužić, Gustav Matula i Ante Đerek. Primjeri implementiranih rješenja su dani u priloženim izvornim kodovima koji nužno ne odgovaraju u svim detaljima ovdje opisanim algoritmima.

## **KRV**

*Predložio: Ante Đerek*

*Potrebno znanje: naredba if, for petlja, stringovi, isprobavanje svih kombinacija*

Prvi dio zadatka se svodi na pravilno učitavanje ulaza, određivanje izlaza s nekoliko if naredbi te ispisivanje. Za drugi dio nam je potreban jednostavan algoritam koji isprobava sve moguće scenarije za genotipove roditelja.

Ulaz (niz genotipa) možemo pamtit i kao niz stringova, prvi dio zadatka se rješava tako da for petljom prođemo kroz učitani niz, sa if naredbama usporedimo genotip s jednim od šest mogućih te samo ispišemo odgovarajuću krvnu grupu.

Drugi dio zadatka rješavamo tako da s dvije for petlje isprobamo sve moguće kombinacije genotipova roditelja i (s još jednom for petljom) za svaku kombinaciju odredimo broj osoba iz populacije za koje je moguće da im budu djeca. Najveći od pronađenih brojeva je rješenje ovog dijela zadatka. Dakle još je potrebno za genotipove A, B i C odrediti da li je moguće da se C dobije od A i B. To možemo u posebnom potprogramu riješiti opet s dvije petlje (svakom petljom biramo jedan alel) ili direktno s hrpom if naredbi. Efikasna implementacija ovog pristupa će moći odnos genotip / fenotip u konstantnim nizovima stringova.

**Za one koji žele više:** Zadana je populacija zajedno s odnosima roditelj-dijete. Za neke osobe je poznat genotip, za neke fenotip, a za neke nije poznato ništa. Napišite program koji određuje sve nepoznate genotipove/fenotipove koje je moguće odrediti.

## **MREŽE**

*Predložili: Ante Đerek, Ivan Katanić, Gustav Matula*

*Potrebno znanje: stringovi, binarni brojevi, while petlja, pohlepni algoritam*

Prije nego što se krenemo baviti samim algoritmom potrebno je odlučiti koju strukturu podataka odabrati za spremanje adresa. Naravno, postoji više različitih pristupa: najefikasniji način bi bio da adresu pamtimo kao 32-bitni cijeli broj (npr. unsigned int) te manipuliramo direktno s njegovim bitovima. Drugi, možda jednostavniji način, je da se binarni brojevi pamte kao stringovi. Podmrežu, dakle, opisujemo s jednim stringom  $S$  duljine 32 te cijelim brojem  $P$ , koji predstavlja veličinu prefiksa podmreže. Pretvorba broja u string koji sadrži binarnu reprezentaciju se radi pomoću petlje u kojoj broj uzastopno dijelimo s dva te uzimamo sljedeću znamenku kao ostatak pri tom dijeljenju.

**Podzadatak A:** adresu iz upita pretvaramo u string na isti način kao i adrese podmreže te ga uspoređujemo sa svakom podmrežom. Adresa će pripadati podmreži ( $S$ ,  $P$ ) ako i samo ako se string  $S$  i string adrese podudaraju na prvih  $P$  pozicija. Ovo se može provjeriti jednom for petljom ili ugrađenim funkcijama za manipulaciju stringova.

**Podzadatak B:** potrebno je pronaći minimalnu reprezentaciju svih adresa koje su pokrivene zadanim podmrežama. Prvi korak je izbaciti podmreže koje su u potpunosti sadržane u nekoj drugoj podmreži. Podmreža s prefiksom  $P$  sadrži  $2^{32-P}$  adresa pa zaključujemo podmreža ( $S_1$ ,  $P_1$ ) može sadržavati podmrežu ( $S_2$ ,  $P_2$ ) samo ako vrijedi  $P_1 \leq P_2$ . Drugi uvjet je da se adrese  $S_1$  i  $S_2$  podudaraju na prvih  $P_1$  pozicija.

Drugi korak je spajanje neke dvije podmreže u jednu veću. Kada možemo spojiti podmreže ( $S_1$ ,  $P_1$ ) i ( $S_2$ ,  $P_2$ )? Ako je  $P_1 = P_2$  i  $S_1$  i  $S_2$  se podudaraju u prvih  $P_1-1$  pozicija, ali na poziciji  $P_1$  se razlikuju, onda njihovu uniju možemo opisati jednom podmrežom čija je adresa jednaka stringovima  $S_1$  i  $S_2$  na prvih  $P_1-1$  pozicija, a prefiks joj je jednak  $P_1-1$ .

Primijetimo da npr. nakon što smo spojili neke dvije pod mreže u jednu moguće je da se ta nova pod mreža može spojiti s nekom drugom, ili da u potpunosti sadrži neku treću itd.

To nas navodi na ponavljanje opisana 2 koraka dok god to ima smisla, a to ima smisla ponavljati dokle god uspijevamo uspješno provesti neku od redukcija. Neka je jedna iteracija algoritma prolazak po svim parovima pod mreža i uklanjanje onih koje su u potpunosti sadržane u drugima, te pokušaj spajanja parova. Iteracija se prekida onog trenutka kad je uspješno napravljena neka promjena na skupu pod mreža te se iteracija ponavlja ispočetka. Onog trenutka kad cijela iteracija ne napravi niti jednu promjenu, sa sigurnošću da mogućih promjena više nema možemo prekinuti algoritam.

Budući da se u svakoj iteraciji koja napravi promjenu broj pod mreža smanji za 1, algoritam će se završiti u najviše  $N+1$  iteracija. Svaka iteracija isprobava sve parove pod mreža za moguću redukciju što daje složenost  $O(N^2)$  po iteraciji ili  $O(N^3)$  ukupno.

**Za one koji žele više::** Pokušajte poboljšati složenost opisanog algoritma na  $O(N^2)$ .

Uz vizualnu reprezentaciju adresa pod mreža u obliku binarnog stabla gdje svaki vrh sadrži neku binarnu znamenku moguće je doći do linearnog algoritma koji rješava zadatak.

## **BOMBE**

*Predložio: Frane Kurtović*

*Potrebno znanje: polja, prebrojavanja, optimizacije*

Naivno rješenje koje postavlja pokušava postaviti dvije bombe na svih  $N^4/2$  načina nije dovoljno brzo da dobije sve bodove, stoga ga ideju potrebno poboljšati.

Pretpostavimo da smo postavili jednu bombu na poziciju  $(x, y)$  i sada želimo naći optimalan način za postaviti drugu bombu. Možemo razmatrati dva odvojena slučaja: a) kada se središte druge bombe nalazi unutar  $21 \times 21$  pravokutnika centriranog oko  $(x, y)$  te b) kada se središte druge bombe nalazi izvan njega.

**Slučaj a):** Broj konfiguracije dvije bombe koje odgovaraju ovom slučaju ima relativno malo. Točnije, ima ih najviše  $N^2 * 21^2$ , što je dovoljno malo da možemo ručno iterirati po svakom od tih slučaja posebno. Ipak, budući da je moguće da se bombe preklapaju u ovome slučaju potrebno je obratiti posebnu pažnju na njega.

**Slučaj b)** ćemo riješiti upotrebom odgovarajuće strukture podataka. Ovdje je olakotna okolnost što neće biti apsolutno nikakvog preklapanja između dviju bombi. Potrebna nam je struktura koja će nam u svakom trenutku od svih pozicija izvan  $21 \times 21$  kvadrata centriranog oko  $(x, y)$  pronaći najbolju poziciju (s kojom ćemo pokupiti najveći broj bodova ako na njoj postavimo bombu). To možemo jednostavno postići tako da pamtimo niz frek veličine  $0..21$  (jer je 21 najveći rezultat kojeg možemo dobiti sa samo jednom bombom) koji nam na mjestu  $frek[k]$  pamti koliko ima različitih pozicija za drugu bombu s kojom ćemo dobiti točno  $k$  bodova. Ukoliko znamo efikasno održavati takav niz, gotovi smo.

Primijetimo sada da pri premještanju prve bombe s pozicije  $(x, y)$  na poziciju  $(x, y+1)$  potrebno je točno 21 novu poziciju dodati u niz frek, te točno 21 poziciju izbaciti iz istog niza. Ovo nam omogućava da efikasno implementiramo pomak prve bombe u sljedeći stupac i tako efikasno održavamo strukturu. Složenost održavanja ove strukture je otprilike  $N^2 * 21 * 2$ , što je dovoljno dobro da dobije sve bodove.

## **RUTA**

*Predložio: Ante Derek*

*Potrebno znanje: stringovi, binarni brojevi, for petlja*

Prije nego što se krenemo baviti samim algoritmom potrebno je odlučiti koju strukturu podataka odabrati za spremanje adresa i maski. Naravno, postoji više različitih pristupa: najefikasniji način bi bio da adresu pamtimo kao 32-bitni cijeli broj (npr. unsigned int) te manipuliramo direktno s njegovim bitovima. Drugi, možda jednostavniji način, je da se binarni brojevi pamte kao stringovi. Svaku adresu i masku, dakle, opisujemo s jednim stringom  $S$  duljine 32 koji se sastoji od znamenki nula i jedan. Pretvorba broja u string koji sadrži binarnu reprezentaciju se radi pomoću petlje u kojoj broj uzastopno dijelimo s dva te uzimamo sljedeću znamenku kao ostatak pri tom dijeljenju.

Drugi korak u implementaciji je da napišemo potprogram koji će za zadanu adresu i masku pod mreže i adresu odredišta paketa, odrediti da li adresa odredišta pripada toj pod mreži. Ovo implementiramo pomoću jedne for petlje doslovno prepisujući logiku iz teksta zadatka.

Zadnji korak je da, za svaku učitane adresu odredišta paketa, prođemo for petljom kroz tablicu usmjeravanja, pomoću potprograma provjeravamo pripadnost pod mreži i ispisujemo oznaku usmjeritelja.

### **Za one koji žele više**

Riješi zadatak tako da adresu pamtiš kao cijeli broj (npr. unsigned int) i koristiš logičke operacije na bitovima.

## **RIBE**

*Predložio: Stjepan Glavina*

*Potrebno znanje: nizovi, pohlepni algoritam*

Zadatak rješavamo pohlepnim algoritmom koji u svakom koraku traži prvu sljedeću (najmanju) ribu koja može progutati prethodnu.

- Recimo da je riba  $A$  najmanja riba (veličine  $velA$ ).
- Riba  $B$  je najmanja koja može progutati ribu  $A$ . Vrijedi:  $velB > velA$ .
- Riba  $C$  je najmanja koja može nakon toga progutati  $B$ . Vrijedi:  $velC > velA + velB$ .
- Riba  $D$  je najmanja koja može nakon toga progutati  $C$ . Vrijedi:  $velD > velA + velB + velC$ .
- I tako dalje.

Sada se za ribu  $X$  pitamo koliko najviše riba može imati u trbuhu. Ključna stvar koju treba primijetiti je da je dovoljno provjeriti samo koju najveću ribu među gore opisanim ribama  $A, B, C, D, \dots$  (nakon što su narasle gutanjem manjih) riba  $X$  može progutati.

Točnije. dovoljno je provjeriti sljedeće:

- Je li  $velX > velA$ ? Ako da, riba  $X$  može imati jednu u trbuhu.
- Je li  $velX > velA + velB$ ? Ako da, riba  $X$  može imati dvije u trbuhu.
- Je li  $velX > velA + velB + velC$ ? Ako da, riba  $X$  može imati tri u trbuhu.
- I tako dalje.

Naravno, moguće je da drugom strategijom riba  $X$  u trbuhu ima i neke druge kombinacije riba. Međutim takve slučajeve uopće nije potrebno razmatrati jer na taj način sigurno neće imati više riba u trbuhu.

Ovaj pohlepni algoritam može se jednostavno implementirati pomoću jedne for petlje.

## **LOGOS**

*Predložili: Ante Đerek, Ivan Katanić*

*Potrebno znanje: stringovi, parsiranje izraza, dinamičko programiranje*

Kako se logički izrazi definiraju rekurzivno tako ima smisla tražiti i rekurzivnu formulu za rješenje problema. Tražimo dakle formulu koja će računati rješenje za izraz na temelju rješenja za podizraze od kojih je on sagrađen.

Dizajnirajmo funkciju 'rijesi' koja kao argument prima logički izraz u obliku niza znakova (string) i cijene promjena pojedinih znamenaka, a vraća dva broja ( $V$ ,  $C$ ):

- $V$  - logička vrijednost izraza
- $C$  - minimalna cijena potrebna za promjenu vrijednosti izraza.

U glavnom programu funkciju pozivamo s ulaznim podacima i ispisujemo rješenje ovisno o vraćenim brojevima  $V$  i  $C$ :

- 0 - ukoliko je  $V = 1$
- $C$  - inače.

Temeljni slučajevi funkcije 'rijesi' su logički izrazi koji se sastoje od samo jedne znamenke, u tom slučaju povratna vrijednost funkcije bit će (0,  $X$ ) ako je to znamenka 0, odnosno (1,  $X$ ) ako je to znamenka 1, gdje je  $X$  cijena promjene znamenke.

Inače, izraz se sastoji od više podizraza i operatora. Operatore i podizraze možemo pronaći na više načina, jedan neoptimalan ali dovoljno efikasan za ovaj zadatak je pronalaženje znakova koji su okruženi samo jednim parom zagrada, npr.  $(A|B|C)$  gdje su  $A$ ,  $B$  i  $C$  logički izrazi. Označimo operator s  $R$ , a podizraze s  $V_1, V_2, \dots, V_N$ , svaki od njih rješavamo rekurzivno funkcijom riješi, a povratne vrijednosti označimo s  $(V_1, C_1) \dots (V_N, C_N)$ . Povratnu vrijednost izraza  $V$  računamo primjenom operatora  $R$  na logičke vrijednosti  $V_1 \dots V_N$ .

Minimalnu cijenu promjene  $C$  možemo pronaći analiziranjem svih slučajeva.

- $R = \text{AND}$  i  $V = 0$  onda je  $C = \text{suma } C_i$ , gdje je  $V_i = 0$ 
  - kako bi izraz promijenio vrijednost i postao istinit svi podizrazi koji nisu istiniti moraju promijeniti vrijednost pa je cijena zbroj svih takvih promjena.
- $R = \text{AND}$  i  $V = 1$  onda je  $C = \min C_i$ 
  - kako bi izraz promijenio vrijednost i postao neistinit dovoljno je da samo jedan podizraz promjeni vrijednost u 0 pa je cijena minimalan  $C_i$  jer su svi podizrazi istiniti.
- $R = \text{OR}$  i  $V = 0$  onda je  $C = \min C_i$ 
  - kako bi izraz promijenio vrijednost i postao istinit barem jedan podizraz mora postati istinit.
- $R = \text{OR}$  i  $V = 1$  onda je  $C = \text{suma } C_i$  gdje je  $V_i = 1$ 
  - kako bi izraz promijenio vrijednost i postao neistinit svi podizrazi moraju biti neistiniti pa zbrajamo cijene promjena onih koji to jesu.
- $R = \text{XOR}$  onda je  $C = \min C_i$ 
  - kako bi izraz promijenio vrijednost potrebno je samo promijeniti vrijednost bilo kojeg podizraza neovisno o trenutnoj istinitoj vrijednosti, pa je optimalno uzeti onog s najmanjom cijenom promjene.

Ovu logiku računanje funkcije riješi možemo direktno implementirati rekurzivnim potprogramom.