



ZADATAK	PERIOD	SORT	SLOVA	KLIZA
ulazni podaci	standardni ulaz			
izlazni podaci	standardni izlaz			
vremensko ograničenje	1 sec	1 sec	1 sec	1 sec
memorijsko ograničenje	64 MB	64 MB	64 MB	64 MB
broj bodova	100	100	100	100
	400			



Ovako definiran zadatak omogućuje izradu slojevitih rješenja koja mogu donijeti odgovarajući broj bodova.

Za početak, decimalni broj s ulaza ćemo učitati u obliku stringa. Ako se ne sjetimo općeg rješenja za traženje perioda ponavljanja decimala, tada možemo riješiti samo prvi slučaj opisan u zadatku.

Prvi slučaj: Kako znamo da je $x=0$ i da će se period ponavljati točno tri puta, tada je dovoljno odrediti duljinu podstringa desno od decimalne točke. Naime, tada znamo da se u prvoj trećina tog podstringa nalaze decimale koje se ponavljaju. Sada još treba uočiti da je traženi razlomak oblika:

$$\text{razlomak} = \text{period ponavljanja} / 10^{\text{duljina perioda ponavljanja} - 1}.$$

Drugi slučaj: U općem slučaju, krenuvši od kraja učitano stringa, treba tražiti pravilnost u ponavljanju decimala. Zbog uvjeta iz zadatka na broj decimala, prvo gledamo jesu li zadnje tri znamenke jednake, zatim gledamo jesu li zadnja tri para znamenki jednaka i jesu li jednake zadnje tri trojke. Kada odredimo period, znamenke koji se ne ponavljaju lako odredimo promatrajući odnos ukupne duljine podstringa iza decimalne točke i trostruke duljine perioda znamenki koje se ponavljaju. S dva tako određena slijeda znamenki, pratimo opisani algoritam iz zadatka i određujemo traženi razlomak.

Napomena:

Skraćivanje razlomka do kraja: U 60% test primjera razlomak nije skraćen do kraja. Za skraćivanje razlomka, trebamo odrediti najveću zajedničku mjeru brojnika i nazivnika. U tu svrhu možemo koristiti Euklidov algoritam.

Primjer implementacije Euklidovog algoritma:

```
int gcd (long long a, long long b)
{
    long long x,y;
    x = min(a,b);
    y = max(a,b);

    if (x == 0) return y;

    return gcd(x,y % x);
}
```



U zadatku Sort potrebno je osmisliti način kako usporediti dvije riječi iz ulaznog riječnika. Budući da se slova od kojih su riječi sastavljene ne sastoje nužno od jednog znaka, tradicionalna usporedba dvije riječi neće raditi očekivano.

Kao prvi korak u rješenju potrebno je riječ rastaviti na slova. Struktura podataka koja pamti riječ bit će niz string-ova u kojem će svaki string biti sastavljen od jednog znaka osim string-ova koji predstavljaju slova <LJ> i <NJ>. To radimo tako da obilazimo riječ slovo po slovo i provjeravamo nalazi li se slovo <J> neposredno iza nekog slova <L> ili <N>. Ako da, onda u niz dodamo dva slova od jednom, a u svakom drugom slučaju dodajemo po jedno slovo.

Sada za svaku riječ imamo niz stringova u kojem svaki string predstavlja jedno hrvatsko slovo. Na kraj svake riječi možemo dodati i string "0" koji će označavati kraj riječi kako bi nam usporedba bila lakša.

U usporedbi dvaju nizova koristimo jednostavnu for petlju. Ako je prvo slovo različito, manja riječ je ona čije je prvo slovo manje. Ako je prvo slovo isto, idemo na sljedeće slovo i ponavljamo algoritam.

Pri sortiranju riječnika možemo koristiti ugrađenu funkciju za sortiranje.



U zadatku Slova imamo posla s permutacijama. Permutacija je neki niz znakova duljine točno N u kojem se svako slovo pojavljuje točno jednom. Za početak, prebrojimo koliko različitih permutacija možemo sastaviti od niza od N znakova.

Rješenje će naravno imati N mjesta i na svako mjesto možemo staviti po jedan znak. Na prvo mjesto možemo staviti svako od N slova. Kada postavimo slovo na prvo mjesto, na drugo mjesto možemo staviti sva slova osim slova koje je postavljeno na prvo mjesto, odnosno $N - 1$ mogućnosti. Na treće mjesto moguće je staviti $N - 2$, itd. dok na zadnje mjesto, kada su sva slova ranije odabrana možemo staviti samo preostalo slovo što je jedna jedina mogućnost. Množenjem svih ovih izbora dobivamo broj $N * (N-1) * (N-2) * \dots * 2 * 1$ što je $N!$.

Budući da je popis riječi leksikografski uređen, na početku se nalaze sve permutacije koje počinju sa slovom A i njih ima $(N-1)!$. Nakon toga slijedi $(N-1)!$ permutacija koje počinju sa slovom B , itd.. Kako bi izračunali udaljenost dviju riječi možemo odrediti koja je po redu manja riječ u riječniku, veća riječ u riječniku i tada oduzeti te dvije vrijednosti. Opišimo algoritam koji određuje redni broj neke permutacije.

U nizu čuvamo sva slova koja još nismo vidjeli. Inicijalno je to svih N slova. Rezultat postavimo na nulu. Promatramo prvo slovo u permutaciji. Određujemo broj slova koja su manja od trenutnog slova i koja još nismo vidjeli prije. Za svako to slovo znamo da postoji $(N-1)!$ permutacija koje sigurno dolaze prije nas. U rješenje dodajemo prebrojeni broj slova pomnožen s $(N-1)!$. Iz niza slova koje još nismo vidjeli brišemo trenutno slovo i nastavljamo algoritam sa sljedećim slovom.

Konačno rješenje dobivamo oduzimanjem rednog broja leksikografski veće permutacije od rednog broja manje permutacije. Iako će redni broj permutacije možda premašiti broj 2^{64} , ako se on pamti u cjelobrojnom tipu bez predznaka, dobiveni preljev neće uzrokovati pogrešku.



U svemu što slijedi prazno polje ćemo predstavljati s brojem 9. Na taj način možemo svako stanje tablice predstaviti kao permutaciju brojeva od 1 do 9, gdje je permutacija niz duljine 9 u kojem se svaki broj od 1 do 9 pojavljuje točno jednom.

Zadatak rješavamo na sljedeći način:

Svaku permutaciju označavamo s njenim rednim brojem definiranim npr. u zadatku Slova.

Koristimo pomoćni niz `bio[i]` u kojem pamtimo jesmo li permutaciju pod rednim brojem `i` već posjetili i to na način da označavamo u `bio[i]` korak kojim smo prvi put posjetili stanje `i`.

Pseudokod rješenja koje koristi strukturu `red`:

Ubacujemo početno stanje u `red`.

Dok `red` nije prazan:

 Uzimamo prvo stanje u `red`u i označavamo s `x`.

 Izbacujemo prvo stanje iz `red`a.

 Promatramo sva stanja do kojih možemo doći u jednom koraku iz stanja `x`, a da nisu već posjećena

 Označimo da smo ta stanja posjetili i zapišemo u njihov `bio` koji broj smo pomaknuli na prazno polje da dođemo do tog stanja iz stanja `x`. Zatim ubacujemo stanje u `red`.

Na posljetku ispisujemo krenuvši iz početnog stanja redom brojeve zapisane u `bio` trenutnog stanja te se pomičemo prema toj uputi.

Primijetimo da različitih permutacija od 9 elemenata ima $9! = 362880$ te se iz svakog stanja možemo proširiti u maksimalno 4 druga stanja u jednom koraku. Što nam daje zadovoljavajuću složenost.

Alternativno zadatak je moguće riješiti i rekurzijom s memoizacijom dok rekurzija bez memoizacije daje 60% bodova, a uz par optimizacija moguće je postići i do 80% bodova.