	Linijski programi

Zadaci

1.

Beleiver

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	64 MB	standardni ulaz	standardni izlaz

Za date prirodne brojeve a i b napisati program kojim se dati nepravi razlomak $\frac{a}{b}$ prevodi u mešovit broj n $\frac{c}{b}$, takav da važi da je $\frac{c}{b} < 1$.

Ulaz

U prvoj liniji standardnog ulaza nalazi se prirodan broj a koji predstavlja brojilac nepravog razlomka, a u drugoj liniji prirodan broj b različit od nule koji predstavlja imenilac razlomka (a≥=b).

Izlaz

Prva i jedina linija standardnog izlaza sadrži mešoviti zapis razlomka, preciznije prirodan broj, brojilac i imenilac mešovitog broja međusobno odvojeni sa po jednom prazninom (blanko znakom).

Primer

Ulaz

```
23
8
```

Izlaz

```
2 7 8
```

REŠENJE

IDEJA:
1. Izračunati celobrojni deo količnika a/b
2. Izračunati ostatk pri deljenju a sa b kako bi se dobio brojilac c razlomljenog dela $\frac{c}{b}$ mešovitog razlomka

C++ rešenje

```cpp
#include <iostream>

using namespace std;

int main() {
  int a, b;
  cin >> a >> b;
  int n = a / b;
  int c = a % b;
  cout << n << " " << c << " " << b << endl;
  return 0;
}
```

Python rešenje

```
a = int(input())
b = int(input())
n = a // b
c = a % b
print(n, c, b);
```

2.

Natural

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	64 MB	standardni ulaz	standardni izlaz

Napiši program koji određuje poslednje tri cifre zbira i poslednje tri cifre proizvoda četiri uneta cela broja.

Ulaz

U svakom redu standardnog ulaza unosi se po jedan ceo broj iz intervala [1..999].

Izlaz

Na standardni izlaz se ispisuju broj određen sa poslednje tri cifre zbira i broj određen sa poslednje tri cifre proizvoda unetih brojeva (eventualne vodeće nule se ne moraju ispisati).

Primer

Ulaz

```
999
999
999
999
```

Izlaz

```
996
1
```

REŠENJE

IDEJA:
Ideja koja prirodno prva padne napamet je da se izračunaju zbir i proizvod unetih brojeva, a onda da se poslednje tri cifre odrede izračunavanjem ostatka pri celobrojnom deljenju sa 1000. Kada se u obzir uzme raspon brojeva koji se unose, takvo rešenje bi davalo korektne rezultate u slučaju zbira, međutim, proizvod brojeva može biti mnogo veći od samih brojeva i zato je za izračunavanje poslednje tri cifre proizvoda potrebno primeniti malo napredniji algoritam koji koristi činjenicu da su za poslednje tri cifre proizvoda relevantne samo poslednje tri cifre svakog od činilaca.

Generalno, važe sledeće relacije:
$(a+b) \bmod n=(a \bmod n + b \bmod n) \bmod n$
i
$(a \cdot b) \bmod n=(a \bmod n \cdot b \bmod n) \bmod n$

Funkcije za množenje i sabiranje po modulu zasnovaćemo direktno na prikazanim relacijama, a onda ćemo ih primenjivati tako što ćemo proizvod (tj. zbir) po modulu n svih prethodnih brojeva primenom funkcija kombinovati sa novim brojem.

REŠENJE 1 – koje ne koristi model izložen u IDEJA

C++

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main() {
  int a, b, c, d;
  cin >> a >> b >> c >> d;

  cout << (a + b + c + d) % 1000 << endl;
  cout << (a * b * c * d) % 1000 << endl;

  return 0;
}
```

Python

```python
a = int(input())

b = int(input())
c = int(input())
d = int(input())

print((a + b + c + d) % 1000)
print((a * b * c * d) % 1000)
```

`REŠENJE 2 – potprogrami koji koriste` funkcije za množenje i sabiranje po modulu

Ipak, naglasimo da je izračunavanje celobrojnog količnika i ostatka vremenski zahtevna operacija (najčešće se izvršava dosta sporije nego osnovne aritmetičke operacije), tako da je u praksi poželjno izbeći je kada je to moguće. Na primer, ako ste potpuno sigurni da će a+b biti moguće reprezentovati odabranim tipom podataka, tada je umesto (a mod n+b mod n) mod n ipak bolje koristiti (a+b) mod n

C++

```cpp
#include <iostream>
#include <cmath>

using namespace std;
```

```cpp
int plus_mod(int a, int b, int n) {
  return ((a % n) + (b % n)) % n;
}

int puta_mod(int a, int b, int n) {
  return ((a % n) * (b % n)) % n;
}

int main() {
 int a, b, c, d;
 cin >> a >> b >> c >> d;

 cout << plus_mod(plus_mod(plus_mod(a, b, 1000), c, 1000), d, 1000) << endl;
 cout << puta_mod(puta_mod(puta_mod(a, b, 1000), c, 1000), d, 1000) << endl;

 return 0;
}
```

Python

```python
def plus_mod(a, b, n):
  return ((a % n) + (b % n)) % n


def puta_mod(a, b, n):
  return ((a % n) * (b % n)) % n;

a = int(input())
b = int(input())
c = int(input())
d = int(input())

print(plus_mod(plus_mod(plus_mod(a, b, 1000), c, 1000), d, 1000))
print(puta_mod(puta_mod(puta_mod(a, b, 1000), c, 1000), d, 1000))
```

3.

Thunder

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	64 MB	standardni ulaz	standardni izlaz

Monopol je igra u kojoj se igrači kreću po poljima koja su postavljena u krug. Igrači se uvek kreću u smeru kazaljke na satu. Pretpostavimo da su polja označena rednim brojevima koji kreću od 0, da su na početku igre oba igrača na tom polju i da se tokom igre igrači nisu kretali unatrag. Ako se zna broj polja koje je prvi igrač prešao od početka igre i broj polja koje je drugi igrač prešao od početka igre napiši program koji izračunava koliko koraka prvi igrač treba da napravi da bi došao na polje na kom se nalazi drugi igrač.

Ulaz

Sa standardnog ulaza se unose tri prirodna broja. U prvoj liniji dat je broj polja na tabli, u drugoj broj polja koje je od početka igre prešao prvi, a u trećoj broj polja koje je od početka igre prešao drugi igrač.

Izlaz

Na standardni izlaz treba ispisati koliko koraka unapred igrač treba da napravi da bi stigao na željeno polje.

Primer 1

Ulaz

```
10
3
7
```

Izlaz

```
4
```

Primer 2

Ulaz

```
10
7
3
```

Izlaz

```
6
```

Rešenje

PODACI

Označimo

ukupan broj polja sa n,

broj polja koje je prvi igrač prešao sa A i

broj polja koje je drugi igrač prešao sa B.
Označimo broj polja na kome se prvi igrač trenutno nalazi sa a,
broj polja na koje treba da stigne sa b.

MATEMATIČKI MODEL KRETANJA
PRVI IGRAČ: Tokom svog kretanja, 1.igrač je prešao k_a punih krugova ($k_a \geq 0$) u kojima je prešao po n polja i nakon toga još a polja, tako da je $A = k_a \cdot n + a$ tj., pošto je $0 \leq a < n$, važi $a = A \bmod n$.

DRUGI IGRAČ: Slično je i $b = B \bmod n$.

ODNOS tekućih polja oba igrača
PRVI SLUČAJ: Ako je $b \geq a$, tada se broj koraka može izračunati kao $b - a$.
DRUGI SLUČAJ: Međutim, moguće je i da važi $b < a$ i u tom slučaju igrač mora preći preko polja Start koje je označeno brojem 0.
Broj koraka koje igrač treba da napravi da bi od polja na kom se nalazi stigao do polja start je $n - a$, a broj koraka potrebnih da od polja start stigne do željenog polja je b, tako da je ukupan broj koraka jednak $n - a + b$.

Na osnovu ove diskusije jednostavno je napraviti program koji analizom ova dva slučaja stiže do rešenja.

C++ rešenje 1, TERNARNI OPERATOR poređenje ?:

```cpp
#include <iostream>

using namespace std;

int main() {
  int brojPolja;
  int presaoPrvi, presaoDrugi;
  cin >> brojPolja >> presaoPrvi >> presaoDrugi;
  int saPolja = presaoPrvi % brojPolja;
  int naPolje = presaoDrugi % brojPolja;
  cout << (naPolje >= saPolja ?
         naPolje - saPolja  :
         naPolje - saPolja + brojPolja)
       << endl;
  return 0;
}
```

Python rešenje 1, kompaktna naredba print i grananje kao argument naredbe

```python
brojPolja = int(input())

presaoPrvi = int(input())
```

```
presaoDrugi = int(input())
saPolja = presaoPrvi % brojPolja
naPolje = presaoDrugi % brojPolja
print(naPolje - saPolja if naPolje >= saPolja else naPolje - saPolja +
brojPolja)
```

C++ rešenje 2, NAREDBA if

```
#include <iostream>

using namespace std;

int main() {
  int brojPolja;
  int presaoPrvi, presaoDrugi;
  cin >> brojPolja >> presaoPrvi >> presaoDrugi;
  int saPolja = presaoPrvi % brojPolja;
  int naPolje = presaoDrugi % brojPolja;
  if (naPolje < saPolja)
    naPolje += brojPolja;
  cout << naPolje - saPolja << endl;
  return 0;
}
```

Python rešenje 2, GRANANJE

```
brojPolja = int(input())
presaoPrvi = int(input())
presaoDrugi = int(input())
saPolja = presaoPrvi % brojPolja
naPolje = presaoDrugi % brojPolja
if naPolje < saPolja:
    naPolje = naPolje + brojPolja
print(naPolje - saPolja)
```

MODULARANA ARITMETIKA ili kako da rešenje bude linijski program, a ne program sa grananjem

Još jedno gledište koje dovodi do istog rešenja je mogućnost da se u slučaju da je $b<a$ promene oznake na poljima iza starta.
Tako bi se polje start umesto sa 0 označilo sa n, sledeće bi se umesto sa 1 označilo sa $1+n$ itd., dok bi se polje b označilo sa $b+n$. Dakle, rešenje dobijamo tako što umanjilac uvećavamo za n ukoliko je manji od umanjenika, i izračunavamo razliku.

Posmatrano još iz jednog ugla u ovom zadatku se traži da se odredi razlika brojeva B i A po modulu n. To sugeriše da je umesto analize slučajeva rešenje moguće dobiti izračunavanjem vrednosti izraza $(b-a+n) \bmod n$ tj.
$(B \bmod n - A \bmod n + n) \bmod n$.

Zaista, ako je $b \geq a$, vrednost izraza $b-a+n$ biće veća ili jednaka n i njen ostatak pri deljenju sa n biće jednak vrednosti $b-a$ (traženje ostatka će praktično poništiti prvobitno dodavanje vrednosti n).
Sa druge strane, ako je $b<a$ tada će $b-a$ biti negativan broj, pa će se dodavanjem vrednosti n dobiti broj iz intervala između 0 i $n-1$. Zato pronalaženje ostatka neće na kraju promeniti rezultat i dobiće se vrednost $b-a+n$ za koju smo rekli da je tražena vrednost u ovom slučaju.

U prethodnom smo se oslonili na činjenicu da su i a i b brojevi koji su veći i jednaki od 0 i strogo manji od n. Pronalaženje vrednosti razlike po modulu n moguće je i u opštem slučaju i važi
$(B-A) \bmod n = (B \bmod n - A \bmod n + n) \bmod n$

Dokažite ovo !!!

C++ rešenje 3

```cpp
#include <iostream>

using namespace std;

int main() {
  int brojPolja;
  int presaoPrvi, presaoDrugi;
  cin >> brojPolja >> presaoPrvi >> presaoDrugi;
  int saPolja = presaoPrvi % brojPolja;
  int naPolje = presaoDrugi % brojPolja;
  cout << (naPolje - saPolja + brojPolja) % brojPolja << endl;
  return 0;
}
```

Python rešenje 3

```python
brojPolja = int(input())

presaoPrvi = int(input())
presaoDrugi = int(input())
saPolja = presaoPrvi % brojPolja
naPolje = presaoDrugi % brojPolja
print((naPolje - saPolja + brojPolja) % brojPolja)
```

Sečenje

vreme	memorija	ulaz	izlaz	test p
1 s	64 Mb	standardni ulaz	standardni izlaz	

Pravougaonu terasu dimenzija $d \times s$ centimetara kvadratnih treba popločati korišćenjem pločica kvadratnog oblika stranice p centimetara, koje se postavljaju tako da su im stranice paralelne stranicama terase. Napisati program kojim se određuje koliko se pločica mora seći radi popločavanja, kao i površinu dela terase koji zauzimaju sečene pločice. Od svake sečene pločice koristi se jedan deo, a drugi odbacuje.

Ulaz

- p - stranica pločice u cm ($10 \leq p \leq 50$)
- d - dužina prostorije u cm ($200 \leq d \leq 10000$)
- s - širina prostorije u cm ($200 \leq s \leq 10000$)

Izlaz

Broj pločica koje se moraju seći i površina koju pokrivaju sečene pločice.

Primer

Ulaz

```
20
310
270
```

Izlaz

```
29
5700
```

REŠENJE

Ako je

- p - dužina stranice pločice u `cm`
- d - dužina prostorije u `cm`
- s - širina prostorije u `cm`

tada se

1. traženi broj pločica koje se moraju seći dobija kao razlika minimalnog broja pločica koje pokrivaju pravougaonu oblast: $\lceil d/p \rceil \cdot \lceil s/p \rceil$ i maksimalnog broja pločica koje su sadržane u pravougaonoj oblasti: $\lfloor d/p \rfloor \cdot \lfloor s/p \rfloor$

2. površina koju pokrivaju sečene pločice dobija se kao razlika površine terase: d·s i površine koju pokrivaju pločice koje nisu sečene: ⌊d/p⌋·p·⌊s/p⌋·p

C++ rešenje

```cpp
#include <iostream>

using namespace std;

int main() {
  // dimenzija polocice i duzina i sirina prostorije
  int p, d, s;
  cin >> p >> d >> s;

  int dfloor = d / p, dceil = (d + p - 1) / p;
  int sfloor = s / p, sceil = (s + p - 1) / p;
  // dceil * sceil - minimalni broj plocica koje pokrivaju oblast
  // dfloor * sfloor - maksimalni broj plocica koje su sadrzane u oblasti
  int brojSecenihPlocica = (dceil * sceil) - (dfloor * sfloor);
  int povrsinaPokrivenaSecenimPlocicama = (d*s) - (dfloor*p*sfloor*p);
  cout << brojSecenihPlocica << endl;
  cout << povrsinaPokrivenaSecenimPlocicama << endl;
}
```

Python rešenje

```python
p = int(input())

d = int(input())
s = int(input())
dfloor = d // p
dceil = (d + p - 1) // p
sfloor = s // p
sceil = (s + p - 1) // p
# dceil*sceil - minimalni broj plocica koje pokrivaju oblast
# dfloor*sfloor - maksimalni broj plocica koje su sadrzane u oblasti
brojSecenihPlocica = (dceil * sceil) - (dfloor * sfloor)
povrsinaPokrivenaSecenimPlocicama = (d*s) - (dfloor*p*sfloor*p)
print(brojSecenihPlocica)
print(povrsinaPokrivenaSecenimPlocicama)
```