

# Садржај

<b>1 Квалификације (1. круг)</b>	<b>1</b>
Задатак: Цифре се преклапају . . . . .	1
Задатак: Највећа предност домаћих . . . . .	2
Задатак: Промена школе . . . . .	3
Задатак: Топ . . . . .	4
Задатак: Набрајање . . . . .	5
Задатак: Мица штуцалица . . . . .	6
Задатак: Погрешна операција . . . . .	8
Задатак: Разврставање . . . . .	9
Задатак: Најдали најближи . . . . .	10
Задатак: Wifi снага . . . . .	11
Задатак: Max НЗД . . . . .	13
Задатак: Недостајућа страница четвороугла . . . . .	14
Задатак: Ексел . . . . .	15
Задатак: Крађа дијаманта . . . . .	16
Задатак: Број сегмената са производом нула . . . . .	18
Задатак: Један-два-три-четири . . . . .	20
<b>2 Квалификације (2. круг)</b>	<b>23</b>
Задатак: Закуцавање . . . . .	23
Задатак: Непознати број . . . . .	23
Задатак: Ски пас . . . . .	25
Задатак: Гориво . . . . .	26
Задатак: Прва и последња цифра . . . . .	28
Задатак: Најгушћи подскуп . . . . .	29
Задатак: Месец рођења . . . . .	30
Задатак: Врабац и мачка . . . . .	32
Задатак: Неравнотежа . . . . .	34
Задатак: Дорћол опен . . . . .	36
Задатак: Непарни делиоци . . . . .	38
Задатак: Шибице . . . . .	39
Задатак: Биоскоп . . . . .	41
Задатак: Најбрже до циља . . . . .	41
Задатак: Најмање промена . . . . .	43
Задатак: Амљез . . . . .	45
<b>3 Општинско такмичење</b>	<b>50</b>
Задатак: Дани у недељи . . . . .	50
Задатак: Топлотни појас . . . . .	51
Задатак: Робот достављач . . . . .	52
Задатак: Минимах . . . . .	53
Задатак: Обим паралелограма . . . . .	56
Задатак: Адресе . . . . .	57
Задатак: Слепо куцање речи . . . . .	58
Задатак: Одбојкашки резултат . . . . .	60
Задатак: Бејзбол хитац . . . . .	61

Задатак: Годишње доба . . . . .	62
Задатак: Пар-непар у круг . . . . .	64

# Глава 1

## Квалификације (1. круг)

### Задатак: Цифре се преклапају

*Аутор: Душан Попадић*

Маре и Паја играју игрицу и дошли су до нивоа где треба решити шифру. Схватили су да је шифра заправо један петоцифрен број који је био записан на зиду на претходном нивоу игрице. Ниједан од њих не може да се сети тог броја, али се сећају делова. Маре је запамтио прве три цифре, а Паја последње три. Помозите им да пређу на следећи ниво тако што ћете за њих одредити тражени петоцифрен број. Претпоставити да је свако од њих исправно запамтио цифре, односно да су унети подаци исправни.

#### Опис улаза

У првом реду се налази троцифрен број - део који је запамтио Маре. У другом реду се налази троцифрен број - део који је запамтио Паја.

#### Опис излаза

Исписати тражени петоцифрен број.

#### Пример 1

Улаз	Излаз
145	14563
563	

#### Објашњење

Маре је запамтио прве три цифре, тако да су прве три цифре броја 145. Паја је запамтио последње три цифре, тако да су последње три цифре броја 563. Једини петоцифрени број за који ово важи је 14563.

#### Пример 2

##### Улаз

788  
851

##### Излаз

78851

#### Решење

#### Решење задатка

Пошто се тражи петоцифрени број, а дате су прве три и последње три цифре тог броја, то знали да ће последња цифра Маретовог броја уједно бити и прва цифра Пајиног броја. Дакле потребно је изабрати један од следећа два приступа:

На цео Маретов број ћемо налепити последње две цифре Пајиног броја.

```

m = int(input())
p = int(input())

broj = m * 100 + p % 100;

print(broj)

```

Одстранићемо последњу цифру Маретовог броја и на њега налепити цео Пајин број.

```

m = int(input())
p = int(input())

broj = (m // 10) * 1000 + p;

print(broj)

```

## Задатак: Највећа предност домаћих

*Аутор: Милан Вујчелић*

Дат је резултат кошаркашке утакмице након сваке четвртине. Колика је највећа могућа предност домаћих?

### Опис улаза

У сваком од четири реда стандардног улаза налазе се по два цела ненегативна броја раздвојена размаком. Сваки ред улаза одговара резултату након једне четвртине редом. Први број у реду је број поена које је постигла домаћа екипа, а други број је број поена које је постигла гостујућа екипа. Ниједан од бројева на улазу није већи од 200.

### Опис излаза

На стандардни излаз исписати само један цео број, највећу могућу предност домаће екипе. Ако домаћа екипа ни у једном тренутку није могла да има предност, исписати 0.

### Пример 1

Улаз	Излаз
21 18	24
39 40	
63 54	
78 77	

*Објашњење*

Резултат је у једном тренутку могао да буде 78 : 54, што је 24 поена предности.

### Пример 2

Улаз

```

0 1
1 3
2 5
4 9

```

Излаз

```
0
```

*Објашњење*

Домаћа екипа никада није имала предност. Најповољнији резултат за њу је био на почетку утакмице (0 : 0).

### Решење

Нека су резултати по четвртинама редом  $a_1 : b_1$ ,  $a_2 : b_2$ ,  $a_3 : b_3$  и  $a_4 : b_4$ . Најповољнији резултат за домаћу екипу током прве четвртине је могао да буде  $a_1 : 0$ , док су најповољнији могући резултати током друге, треће

---

и четврте четвртине редом  $a_2 : b_1$ ,  $a_3 : b_2$ ,  $a_4 : b_3$ . Ови резултати настају ако током сваке четвртине прво домаћа екипа постигне све своје поене, а затим гостујућа све своје.

Према томе, највећа вођства или најмањи заостаци домаћих у поенима по четвртинама су  $a_1$ ,  $a_2 - b_1$ ,  $a_3 - b_2$ ,  $a_4 - b_3$ . Тражени резултат је највећи од ова четири броја. Приметимо да највећи од ових бројева не може да буде негативан, јер је  $a_1 \geq 0$ .

```
a1, b1 = input().split()
a1, b1 = int(a1), int(b1)

a2, b2 = input().split()
a2, b2 = int(a2), int(b2)

a3, b3 = input().split()
a3, b3 = int(a3), int(b3)

a4, b4 = input().split()
a4, b4 = int(a4), int(b4)

prednost = max(a1, a2-b1, a3-b2, a4-b3)
print(prednost)
```

## Задатак: Промена школе

Аутор: Душан Попадић, Јелена Петровић

На крају првог разреда средње школе Ивана је одлучила да се пребаци из медицинске школе у гимназију. У гимназији су одлучили да ће јој као просек у првом разреду рачунати само одређене предмете - оне који постоје у обе школе. Закључено је да се  $k$  Иваниних оцена неће рачунати за просек у новој школи. Одредити нови Иванин просек оцена ако је познат просек из медицинске школе, укупан број предмета које је слушала у медицинској ( $n$ ) и  $k$  оцена које јој се неће рачунати. Претпоставити да је Ивана завршила разред, тј. да није имала ниједну закључену јединицу.

Просек оцена  $a_1, a_2, \dots, a_n$  рачуна се на следећи начин:

$$\frac{a_1 + a_2 + \dots + a_n}{n}.$$

### Опис улаза

У првом реду стандардног улаза налази се природан број  $n$ , а у другом реду број  $k$  ( $1 \leq k < n \leq 50$ ). У наредних  $k$  редова је дато  $k$  Иваниних оцена (природни бројеви од 2 до 5) које се не рачунају. У последњем реду се налази број  $p$  који представља Иванин просек из медицинске школе који је заокружен на две децимале.

### Опис излаза

Исписати нови Иванин просек, заокружен на 2 децимале.

#### Пример 1

Улаз	Излаз
3	4.50
1	
4	
4.33	
	4
	4.58

#### Пример 2

Улаз	Излаз
12	4.67
3	
4	
5	
4	
4.58	

### Решење

Нека су  $a_1, a_2, \dots, a_n$  Иванине оцене на крају првог разреда средње медицинске школе, где су са  $a_1, \dots, a_k$  означене оцене које се неће рачунати. Просек свих оцена, означен са  $p$ , рачуна се на следећи начин:

$$p = \frac{a_1 + a_2 + \dots + a_n}{n}.$$

Са  $\hat{p}$  ћемо означити број  $p$  заокружен на две децимале. Како су  $\hat{p}$  и  $a_1, \dots, a_k$  познати, збир преосталих оцена можемо добити користећи претходну формулу, водећи притом рачуна о заокруживању јер су оцене природни бројеви:

$$a_{k+1} + \dots + a_n = p \cdot n - (a_1 + \dots + a_k) = \text{round}(\hat{p} \cdot n) - (a_1 + \dots + a_k).$$

Пошто преосталих оцена има  $n - k$ , њихов просек се може израчунати дељењем израчунатог збира преосталих оцена са  $n - k$  (из услова задатка важи  $n - k > 0$ ):

$$\frac{a_{k+1} + \dots + a_n}{n - k} = \frac{\text{round}(\hat{p} \cdot n) - (a_1 + \dots + a_k)}{n - k}.$$

```
n = int(input())
k = int(input())

# ucitati k izostavljenih ocena i sabrati ih
zbirIzostavljenihOcena = 0
for i in range(k):
    izostavljenaOcena = int(input())
    zbirIzostavljenihOcena += izostavljenaOcena

prosek = float(input())

rezultat = (round(prosek*n) - zbirIzostavljenihOcena) / (n - k)
rezultat = '{:.2f}'.format(rezultat)
print(rezultat)
```

## Задатак: Топ

Аутор: Иван Дреџун

Мали Душан је тек почeo да учи како се игра шах. Учитељ му је на шаховску таблу ставио једног топа. Топ је фигура која може да нападне било које поље у врстама и колонама у којој се налази. Врсте на шаховској табли означене су бројевима од 1 до 8, а колоне словима од A до H енглеске абецеде. Малог Душана занима да ли ће топ моћи да нападне његову фигуру ако је стави на неко поље. Напиши програм који даје Малом Душану одговор на то питање за сва поља за која га то занима.

### Опис улаза

Са стандардног улаза се уноси позиција топа у стандардном шаховском запису. У наредном реду се уноси број  $n$  који представља број питања која Мали Душан поставља. Након тога се у наредних  $n$  редова уносе позиције поља за које Мали Душан жели да добије одговор.

### Опис излаза

На стандардни излаз за свако питање исписати DA уколико топ напада то поље, односно NE у супротном. Одговоре исписивати у засебним редовима.

### Пример

Улаз	Излаз
A4	NE
3	DA
B7	NE
D4	
H2	

### Решење

### Опис главног решења

У овом блоку се описује главно решење задатка.

# Задатак: Набрајање

Аутор: Милан Вућелић

Написати програм који учитава речи док не учита реч `gotovo`, а исписује реченицу којом се набрајају све претходне речи.

## Опис улаза

У сваком реду стандардног улаза налази се по једна реч, написана малим словима енглеске абециде. Речи нису дуже од 30 слова. Последња реч је реч “`gotovo`”. Укупан број речи је најмање 2, а највише 30.

## Опис излаза

На стандардни излаз исписати све речи које претходе речи `gotovo`, тако да између узастопних речи стоји запета и размак, осим између последње две, где стоји слово `i` са размасцима око њега.

### Пример 1

Улаз	Излаз
regica	regica
gotovo	

### Пример 2

Улаз	Излаз
lala	lala
sosa	i sosa
gotovo	

### Пример 3

Улаз	Излаз
vuk	vuk, lija, meda, zeka i magagac
lija	
meda	
zeka	
magagac	
gotovo	

## Решење

Пошто прва реч није реч `gotovo` (укупан број речи је бар 2), прву учитану реч можемо одмах да испишемо. У случају да је друга реч `gotovo`, поступак је већ завршен. Ако друга реч није `gotovo`, ипак не можемо да је испишемо пре него што прочитамо и следећу реч, јер тек тада знамо да ли пре друге речи треба да напишемо зарез или слово `i`.

Закључујемо да је потребно да програм у сваком тренутку рада памти бар две последње учитане речи, да би могао да испише прву од те две речи. Назовимо зато последње две учитане речи *претходна* и *следећа* реч. Сада цео поступак можемо да формулишемо овако:

- учитај и испиши прву реч
- учитај следећу реч
- све док та следећа реч није реч `gotovo`:
  - нека претходна реч преузме вредност од следеће
  - учитај нову следећу реч
  - ако је та следећа реч `gotovo`, испиши слово `i` и претходну реч, иначе испиши зарез и претходну реч
- заврши исписивање текућег реда

```
prethodna, sledeca = input(), input()
print(prethodna, end=' ')
while sledeca != 'gotovo':
    prethodna = sledeca
    sledeca = input()
    if sledeca == 'gotovo':
        print(' i ' + prethodna, end=' ')
    else:
        print(', ' + prethodna, end=' ')
print()
```

Претходно решење може да се измени тако да се резултат накупља у једној текстуалној променљивој и да се оспише тек на крају. Такво решење је прихватљиво само у случају да укупан број речи није велики (нпр. мањи је од хиљаду).

```
prethodna, sledeca = input(), input()
odgovor = prethodna
while sledeca != 'gotovo':
    prethodna = sledeca
    sledeca = input()
    if sledeca == 'gotovo':
        odgovor = odgovor + ' i ' + prethodna
    else:
        odgovor = odgovor + ', ' + prethodna

print(odgovor)
```

## Задатак: Мица штуцалица

У соби има  $n$  сијалица, од којих свака има свој прекидач. На почетку су неке сијалице укључене, а неке искључене. Паја је таман толико порастао да може да притиска прекидаче, што га веома забавља. Паја је  $m$  пута притиснуо неки од прекидача. Сваки пут када је у соби био потпуни мрак, мала Милица је штуцнула.

Написати програм, који за дато почетно стање сијалица и редослед притискања прекидача одређује колико пута је Милица штуцнула.

### Опис улаза

У првом реду стандардног улаза је природан број  $n$ ,  $n \leq 50000$ .

У другом реду је  $n$  бројева раздвојених по једним размаком, од којих је сваки једнак 0 или 1. Нуле означавају искључене сијалице, а јединице укључене. Ови бројеви нису сви једнаки нули.

У трећем реду је природан број  $m$ ,  $m \leq 50000$ .

У четвртом реду је  $m$  бројева раздвојених размаком, који представљају редом индексе (тј. редне бројеве, бројећи од 0) сијалица чије прекидаче је редом Паја притискао.

### Опис излаза

На стандардни излаз исписати један неозначен цео број, који означава колико пута је мала Милица штуцнула.

### Пример 1

Улаз	Излаз
5	2
1 1 0 1 0	
9	
0 1 2 3 2 1 0 1 0	

#### Објашњење

Стања сијалица после притискања прекидача су редом

```
0 1 0 1 0
0 0 0 1 0
0 0 1 1 0
0 0 1 0 0
0 0 0 0 0
0 1 0 0 0
1 1 0 0 0
1 0 0 0 0
0 0 0 0 0
```

Према томе, Милица је штуцнула после петог и после деветог притиска прекидача, што је укупно два штуцања.

---

## Пример 2

Улаз

```
3
1 1 1
12
0 1 0 1 0 1 0 1 2 1 2 1
```

Излаз

```
0
```

### Решење

Свакако нам је потребна једна целобројна променљива помоћу које бројимо колико пута су све сијалице биле искључене, тј. колико пута је мала Мица штуцнула. Ову променљиву ћемо звати бројач штуцања. Поред тога, да бисмо могли да утврдимо да ли су све сијалице искључене, треба да имамо неки низ у коме памтимо стање свих сијалица (нуле за искључене, јединице за укључене). Нека је то низ  $a$ .

Када се притисне прекидач са индексом  $p$ , елемент  $a_p$  треба да промени вредност из 0 у 1 или обрнуто, тј. треба да добије вредност  $1 - a_p$ . Након сваке промене, тј. притиска на неки од прекидача, треба да установимо да ли су све сијалице искључене.

Једна могућност је да после сваке промене пребројимо укључене сијалице, па ако укључених сијалица има 0, да повећамо бројач штуцања. Број операција у овом решењу је сразмеран производу броја сијалица и броја притисака на прекидаче. То може да буде до  $50\,000 \cdot 50\,000 = 2\,500\,000\,000$  операција, а тај број операција не може да се изврши у предвиђеном времену.

```
n = int(input())
a = [int(x) for x in input().split()]
m = int(input())
p = [int(x) for x in input().split()]

stuc = 0
for rb in p:
    a[rb] = 1 - a[rb]
    sve_isklj = True
    for i in range(n):
        if a[i] == 1:
            sve_isklj = False
    if sve_isklj:
        stuc += 1

print(stuc)
```

Нешто боље решење је да после сваке промене тражимо само прву укључену сијалицу, па ако је не нађемо, да повећамо бројач штуцања. Нажалост, и даље постоји врло неповољан случај, а то је када је прва укључена сијалица при крају низа.

```
n = int(input())
a = [int(x) for x in input().split()]
m = int(input())
p = [int(x) for x in input().split()]

stuc = 0
for rb in p:
    a[rb] = 1 - a[rb]
    sve_isklj = True
    for i in range(n):
        if a[i] == 1:
            sve_isklj = False
            break
    if sve_isklj:
```

```

stuc += 1

print(stuc)

Бољи начин да после промене установимо да ли су све сијалице искључене је да одржавамо број укључених сијалица. То значи да када Паја укључи неку сијалицу, треба да повећамо број укључених сијалица, а када искључи неку сијалицу да тај број смањимо. После ажурирања броја укључених сијалица треба још да проверимо да ли је тај број нула и ако јесте, да повећамо бројач штуцања. На тај начин долазимо до ефикаснијег решења, јер је у њему укупан број операција након учитавања сразмеран са бројем притисака на прекидаче.

n = int(input())
a = [int(x) for x in input().split()]
m = int(input())
p = [int(x) for x in input().split()]

br_uklj = sum(a)
stuc = 0
for i in p:
    if a[i] == 0:
        a[i] = 1
        br_uklj += 1
    else:
        a[i] = 0
        br_uklj -= 1

    if br_uklj == 0:
        stuc += 1

print(stuc)

```

## Задатак: Погрешна операција

*Аутор: Небојша Варница*

За дати природан број  $n > 1$  одредити разломак који сабран са  $n$  даје исти резултат као и када се тај разломак помножи са  $n$ . Разломак треба да буде потпуно скраћен (бројилац и именилац треба да буду узајамно прости).

### Опис улаза

У првом и једином реду стандардног улаза налази се природан број  $n$ , већи од 1.

### Опис излаза

На стандардни излаз исписати бројилац и именилац траженог разломка у истом реду раздвојене само симболом /.

### Пример

Улаз	Излаз
3	3/2

*Објашњење*

$$3 + \frac{3}{2} = \frac{9}{2}$$

$$3 \cdot \frac{3}{2} = \frac{9}{2}$$

### Решење

Претпоставимо да је разломак  $\frac{a}{b}$ . Тада треба да важи да је

---

$$n + \frac{a}{b} = n \cdot \frac{a}{b}$$

Одатле следи да је:

$$n = n \cdot \frac{a}{b} - \frac{a}{b} = (n - 1) \frac{a}{b}$$

тј.

$$\frac{a}{b} = \frac{n}{n - 1}$$

Пошто су  $n$  и  $n - 1$  узастопни природни бројеви, они су узајамно прости, па је решење разломак  $\frac{n}{n-1}$ .

```
n = int(input())
print(n, n-1, sep="/")
```

## Задатак: Разврставање

Аутор: Душан Попадић

Филип учествује на школском такмичењу у малом фудбалу на ком се ѡаци такмиче по екипама, којих укупно има  $k$ . Пре разврставања су сви поређани у редове, а у сваком реду има по  $m$  ученика. Разврставање креће од почетка првог реда и ѡаци се редом разврставају у екипе  $1, 2, \dots, k, 1, 2 \dots$  Када се разврстају ѡаци из првог реда, прелази се на првог ѡака у наредном реду и тако до краја. Филип се налази у  $i$ -том реду и преbroјајо је да је он  $-$ ти ѡак у том реду (сва бројања крећу од 1). Одредити редни број срећне екипе у коју ће он запasti.

### Опис улаза

Уносе се редом природни бројеви  $k, m, i$  и  $j$  ( $j \leq m$ ), сваки у посебном реду. Сви бројеви су природни и мањи од 500.

### Опис излаза

Исписати један цео број – редни број екипе у којој ће играти Филип.

### Пример

Улаз	Излаз
12	3
8	
4	
3	

### Објашњење

Са слике се види да ће Филип играти у екипи 3. Са  $X$  су означени ученици који стоје у редовима, а у загради пише у којој ће екипи играти. Филип је означен плавом бојом.

### Решење

Да бисмо одредили екипу у којој ће Филип наступати, прво је потребно одредити који по реду ће Филип бити разврстан (односно колико има ученика испред њега). Број ученика испред Филипа одређујемо тако што прво видимо колико има редова испред Филипа и за сваки ред који се налази испред њега на број ученика додајемо  $m$ , што је број ученика у реду. Дакле, ако се Филип налази у првом реду тај број ће бити 0, ако се налази у другом реду биће  $m$ , ако се налази у трећем биће  $2m$  итд. Одатле закључујемо да ако се Филип налази у  $i$ -том реду, број ученика у редовима испред њега ће бити  $(i - 1) * m$ .

На овај број треба додати све ученике који се налазе у истом реду као и Филип, али су испред њега. Поново, ако је Филип први у свом реду то ће бити 0, ако је други биће 1 итд. Дакле, пошто је Филип на  $j$ -том месту у свом реду, треба додати број  $j - 1$ . Коначно долазимо до тога да се испред Филипа налази  $(i - 1) * m + j - 1$  ученика.

Ако сваког ученика “обележимо” бројем ученика испред њега, тада 0. ученик иде у групу 1, 1. у групу 2 …  $k - 1$ -и ученик иде у групу  $k$ , а затим  $k$ -ти поново у групу 1. Пошто видимо да се на сваких  $k$  бројач

група ресетује, закључујемо да можемо посматрати групу као остатак при дељењу броја којим смо обележили ученика са  $k$ . Наравно, пошто групе бројимо од 1, а не од 0, на крају треба додати +1 на број групе.

```

k = int(input())
m = int(input())
i = int(input())
j = int(input())

rBr = (i - 1) * m + j - 1
grupa = rBr % k + 1
print(grupa)

```

## Задатак: Најдаљи најближи

*Аутор: Милан Вујчелић*

Написати програм који од 4 учитана цела броја исписује онај, коме је растојање до најближег од осталих бројева максимално. Ако има више таквих бројева, исписати их све, редом по величини од најмањег од највећег.

### Опис улаза

На стандардном улазу се налазе четири цела броја из интервала  $[-1000000, 1000000]$ , сваки посебном реду.

### Опис излаза

На стандардни излаз исписати оне од учитаних бројева који испуњавају описани услов, сваки у посебном реду.

#### Пример 1

Улаз	Излаз
7	7
2	2
4	7
1	9

#### Пример 2

Улаз	Излаз
4	2
2	4
7	7
9	9

#### Пример 3

Улаз	Излаз
4	1
1	9
6	6
9	9

### Решење

Задатак се лакше решава ако најпре уредимо 4 учитана броја  $a, b, c, d$  по величини. Након што постигнемо да важи  $a \leq b \leq c \leq d$ , можемо лако за сваки од њих да одредимо даљину до најближег од преостала три:

- $a_m = b - a$
- $b_m = \min(b - a, c - b)$
- $c_m = \min(c - b, d - c)$
- $d_m = d - c$

Нека је  $m = \max(a_m, b_m, c_m, d_m)$ . Потребно је још исписати сваки од бројева за који је растојање од њега до најближег од осталих једнако  $m$ .

```

a, b, c, d = sorted([int(input()),int(input()),int(input()),int(input())])
do_a = b - a
do_b = min(b - a, c - b)
do_c = min(c - b, d - c)
do_d = d - c
d_max = max(do_a, do_b, do_c, do_d)
if d_max == do_a:
    print(a)
if d_max == do_b:
    print(b)
if d_max == do_c:
    print(c)
if d_max == do_d:
    print(d)

```

---

Сортирање се још једноставније спроводи ако су бројеви учитани у низ.

# varijanta u kojoj treba ispisati odgovore u redosledu ucitavanja (ako ih je vise)

```
def do_najblizeg(x, y1, y2, y3):
    return min(abs(x-y1), abs(x-y2), abs(x-y3))

a, b, c, d = int(input()), int(input()), int(input()), int(input())
do_a = do_najblizeg(a, b, c, d)
do_b = do_najblizeg(b, a, c, d)
do_c = do_najblizeg(c, a, b, d)
do_d = do_najblizeg(d, a, b, c)
d_max = max(do_a, do_b, do_c, do_d)
if d_max == do_a:
    print(a)
if d_max == do_b:
    print(b)
if d_max == do_c:
    print(c)
if d_max == do_d:
    print(d)
```

## Задатак: Wifi снага

Автор: Филип Марић

Познате су локације кућа дуж једне улице (њихове целобројне  $x$ -координате) и локације WiFi предајника који су постављени дуж те улице (поново њихове целобројне  $x$ -координате). Одредити минималну снагу сигнала коју је потребно подесити свим предајницима (свима се мора подесити иста снага) да би свака кућа имала сигнал. Ако је снага предајника на позицији  $x$  једнака  $d$ , тада се њиме обухватају куће на позицијама из интервала  $[x - d, x + d]$ .

### Опис улаза

Са стандардног улаза се учитава број кућа  $m$  ( $1 \leq m \leq 10^5$ ), а затим локације кућа (бројеви између 0 и  $10^6$ ). Затим се учитава број предајника  $n$  ( $1 \leq n \leq 10^5$ ), а затим локације предајника (бројеви између 0 и  $10^6$ ).

### Опис излаза

На стандардни излаз исписати минималну снагу потребну да се све куће покрију.

### Пример 1

Улаз	Излаз
4	2
1 2 3 4	
2	
1 5	

Објашњење

### Објашњење

Ако је снага предајника 2, тада први обухвата интервал  $[-1, 3]$ , а други покрива интервал  $[3, 7]$ , што покрива све куће. Ако би снага била 1, тада би први обухватао интервал  $[0, 2]$ , а други  $[4, 6]$ , па кућа на позицији 3 не би била покривена.

### Пример 2

Улаз

```
10
13 4 18 9 16 38 25 42 7 19
5
2 16 33 26 10
```

*Излаз*

9

**Решење****Груба сила**

Решење грубом силом подразумева да мало по мало повећавамо снагу све док све куће не буду покривене.

**Сортирање и техника два показивача**

Можемо одредити најмању потребну снагу за сваку кућу и одредити максимум тих вредности. За сваку кућу одређујемо најближи предајник (анализирајући најближи предајник који је испред ње и најближи предајник који је иза ње) и најмања потребна снага за ту кућу је растојање до њој најближег предајника.

Ако сортирамо низ кућа и низ предајника до решења можемо доћи техником два показивача.

```
import bisect

m = int(input())
kuce = list(map(int, input().split()))
n = int(input())
predajnici = sorted(list(map(int, input().split())))

snaga = 0
for kuca in kuce:
    p = bisect.bisect_left(predajnici, kuca)
    if p == n:
        potrebno = kuca - predajnici[n-1]
    elif p == 0:
        potrebno = predajnici[0] - kuca
    else:
        potrebno = min(kuca - predajnici[p-1], predajnici[p] - kuca)
    snaga = max(snaga, potrebno);

print(snaga)
```

**Сортирање и бинарна претрага**

Можемо одредити најмању потребну снагу за сваку кућу и одредити максимум тих вредности. За сваку кућу одређујемо најближи предајник (анализирајући најближи предајник који је испред ње и најближи предајник који је иза ње) и најмања потребна снага за ту кућу је растојање до њој најближег предајника.

Ако сортирамо само низ предајника, најближи предајник иза куће можемо наћи бинарном претрагом (тражимо најмањи број у низу који је већи или једнак од дате вредности, што се може урадити библиотечком функцијом).

```
m = int(input())
kuce = sorted(list(map(int, input().split())))
n = int(input())
predajnici = sorted(list(map(int, input().split())))
snaga = 0
p = 0
for k in range(m):
    while p < n and predajnici[p] < kuce[k]:
        p += 1
    if p == 0:
        potrebno = predajnici[p] - kuce[k]
    elif p == n:
        potrebno = kuce[k] - predajnici[p-1]
    else:
```

---

```

potrebno = min(predajnici[p] - kuce[k], kuce[k] - predajnici[p-1])
snaga = max(snaga, potrebno)
print(snaga)

```

### Бинарна претрага по решењу

До оптималног решења се може доћи и процесом бинарне преграге по решењу. Знамо да оно лежи у интервалу  $[0, R]$ , где је  $R$  распон између најближе и најдаље куће. Половимо тај интервал, уз коришћење помоћне функције којом се проверава да ли је текућа снага довољна да све куће буду покривене.

## Задатак: Max НЗД

Аутор: Филип Марић

Напиши програм који на основу познатог производа два позитивна природна броја  $a$  и  $b$  одређује највећу могућу вредност њиховог највећег заједничког делиоца.

### Опис улаза

Са стандардног улаза се читају број  $p = a \cdot b$  ( $1 \leq p \leq 10^{18}$ ).

### Опис излаза

На стандардни излаз исписати максималну могућу вредност за НЗД.

### Пример 1

<i>Улаз</i>	<i>Излаз</i>
600	10

*Објашњење*

Највећи НЗД се добија када се број 600 представи као производ бројева 20 и 30.

### Пример 2

*Улаз*

123456

*Излаз*

8

**Решење**

## Груба сила

Решење грубом силом подразумева да број  $n$  на све начине разложимо на производ два броја  $a \cdot b$ , израчунамо њихов НЗД и одредимо максимум свих тако добијених бројева. Без губитка на општости можемо претпоставити да је  $a \leq b$ , па је довољно испитивати све вредности од 1 до  $\sqrt{n}$ .

```

def nzd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

p = int(input())
f = 2
maxNzd = 1
while f * f <= p:
    if p % f == 0:
        maxNzd = max(maxNzd, nzd(f, p // f))
    f += 1

print(maxNzd)

```

## Растављање на просте чиниоце

Ефикасније решење се добија ако се број растави на просте чиниоце:  $p_1^{k_1} p_2^{k_2} \dots p_m^{k_m}$ . Највећи НЗД се добије ако се сваки чинилац  $p_i$  равномерно раздели у два броја  $a$  и  $b$ . Сваки од бројева ће садржати фактор  $p_i^{\lfloor \frac{k_i}{2} \rfloor}$  (заиста, ако је  $k_i$  паран, оба ће имати тачно тај фактор, а ако је непаран, онда ће један од бројева садржати тај фактор, а други број ће садржати фактор  $p_i^{\lfloor \frac{k_i}{2} \rfloor + 1}$ ).

На пример, важи  $600 = 2^3 \cdot 3 \cdot 5^2$ . Највећи НЗД се може добити ако бројеви  $a$  и  $b$  приме по једну двојку и по једну петицу (ирелевантно је како ће се распоредити преостала двојка и тројка).

```
p = int(input())
max = 1
f = 2
while f * f <= p:
    k = 0
    while p % f == 0:
        p = p / f
        k += 1
    if k % 2 == 0:
        max *= f
    f += 1
print(max)
```

## Задатак: Недостајућа страница четвороугла

Аутор: Милан Вујделића

Темена четвороугла су означена великим словима енглеске абециде. Ознаке страница се добијају читањем темена у круг, увек у истом смеру. На пример, ако су темена редом (идући по контури) X, Y, Z, W, онда су ознаке страница XY, YZ, ZW и WX. Дате су ознаке неке три странице у произвoльнoм редоследу страница, а треба одредити ознаку четврте.

### Опис улаза

У сваком од три реда стандардног улаза налазе се по два велика слова енглеске абециде, која представљају ознаку по једне од страница четвороугла. Редослед страница је произвољан.

### Опис излаза

На стандардни излаз исписати два велика слова енглеске абециде, која представљају ознаку четврте странице.

#### Пример 1

Улаз	Излаз
AB	DA
BC	
CD	

#### Пример 2

Улаз	Излаз
QK	LP
PQ	
KL	

#### Пример 3

Улаз	Излаз
HD	DF
FG	
GH	

### Решење

Нека су учитане странице редом  $a, b, c$ . Означимо непознату страницу са  $d$ . Обилазећи странице четвороугла редом у круг почевши од непознате странице  $d$ , на три дате странице можемо да нађемо у једном од шест могућих редоследа:  $abc, acb, bac, bca, cab, cba$ . Сваки од ових редоследа можемо да "препознамо" поредећи слова у страницама за које претпостављамо да су узастопне. Конкретно:

- ако је  $a_1 = b_0, b_1 = c_0$ , тада је редослед наиласка на странице при обиласку редом у круг  $abc$ , па је  $d_0 = c_1, d_1 = a_0$
- ако је  $a_1 = c_0, c_1 = b_0$ , тада је редослед наиласка на странице при обиласку редом у круг  $acb$ , па је  $d_0 = b_1, d_1 = a_0$
- ако је  $b_1 = a_0, a_1 = c_0$ , тада је редослед наиласка на странице при обиласку редом у круг  $bac$ , па је  $d_0 = c_1, d_1 = b_0$
- ако је  $b_1 = c_0, c_1 = a_0$ , тада је редослед наиласка на странице при обиласку редом у круг  $bca$ , па је  $d_0 = a_1, d_1 = b_0$

- ако је  $c_1 = a_0, a_1 = b_0$ , тада је редослед наиласка на странице при обиласку редом у круг  $cab$ , па је  $d_0 = b_1, d_1 = a_0$
- ако је  $c_1 = b_0, b_1 = a_0$ , тада је редослед наиласка на странице при обиласку редом у круг  $cba$ , па је  $d_0 = a_1, d_1 = c_0$

Од ових шест услова увек је тачно један испуњен. Провером свих шест усова утврђујемо који је испуњен, а када то откријемо, лако израчунавамо непознату страницу.

```
a = input()
b = input()
c = input()

if a[1]==b[0] and b[1]==c[0]:
    d=c[1]+a[0]
elif a[1]==c[0] and c[1]==b[0]:
    d=b[1]+a[0]
elif b[1]==a[0] and a[1]==c[0]:
    d=c[1]+b[0]
elif b[1]==c[0] and c[1]==a[0]:
    d=a[1]+b[0]
elif c[1]==a[0] and a[1]==b[0]:
    d=b[1]+c[0]
elif c[1]==b[0] and b[1]==a[0]:
    d=a[1]+c[0]

print(d)
```

## Задатак: Ексел

*Аутор: Душан Попадић*

Мајкрософт Ексел програм садржи табеле чији су редови означенчи бројевима (1, 2, 3 ...), а колоне словима (A, B, C, ..., Y, Z). Када се у табелу дода првише колона (више од 26 колико слова има енглеска абецеда), колоне почињу да се означавају са два слова (AA, AB, AC ..., AZ, BA, BB, BC ... ZZ), затим са три (AAA, AAB, AAC ... AAZ, ABA, ABB, ... ZZZ) и тако даље. Ако знамо да је у табели последња колона означена карактером (или низом карактера)  $s$ , одредити колико колона има у табели.

### Опис улаза

У првој линији стандардног улаза се уноси ниска карактера  $s$ . Сви карактери су велика слова енглеске абецеде и има их највише 6.

### Опис излаза

У једини ред стандардног излаза исписати цео број који представља укупан број колона у табели.

### Пример 1

Улаз	Излаз
F	6

*Објашњење*

Колоне редом иду: A, B, C, D, E i F - дакле 6 колона.

### Пример 2

Улаз

AC

Излаз

29

**Пример 3***Улаз*

EXCEL

*Излаз*

2708874

**Решење****Опис главног решења**

У овом блоку се описује главно решење задатка.

**Задатак: Крађа дијаманта**

Озлоглашени пљачкаши Тоша и Моша испланирали су да украду драгоценни дијамант. Оно што овај подухват чини тежим него раније јесте нови ласерски безбедносни систем постављен у просторијама где се дијамант налази.

Моша је успео да сазна да су ласери постављени веома специфично; постоји тачно  $n$  вертикалних ласера, где је  $i$ -ти ласер постављен на позицију  $v_i$  и  $m$  хоризонталних ласера, постављени на позицију  $h_i$  (ласере можемо посматрати као праве представљене једначином  $x = v_i$  за вертикалне, односно  $y = h_i$  за хоризонталне ласере).

У међувремену, Тоша је сазнао да један прозор на врху зграде може да се отвори, па ако се спусте низ конопац кроз њега, наћи ће се на координатама  $(x_1, y_1)$  у соби, а да је тачна локација дијаманта на координатама  $(x_2, y_2)$ .

Све што је преостало је да се ласери онеспособе. Зато, Тошу и Мошу интересује колико минимално ласера морају да искључе да би могли несметано да дођу до дијаманта, не прелазећи ни преко једног ласера. Да ли можете да им помогнете?

**Опис улаза**

У првом реду стандардног улаза налазе се целобројне вредности  $x_1$  и  $y_1$  раздвојене размаком, координате места у соби на коме се Тоша и Моша налазе на почетку.

У другом реду налазе се целобројне вредности  $x_2$  и  $y_2$  раздвојене размаком, координате дијаманта у просторији.

У трећем реду налази се природан број  $n$ , број вертикалних ласера.

У четвртом реду налази се  $n$  целобројних вредности раздвојених размаком, где  $i$ -та вредност  $v_i$  представља  $x$  координату вертикалног ласера.

У петом реду налази се природан број  $m$ , број хоризонталних ласера.

У шестом реду налази се  $m$  целобројних вредности раздвојених размаком, где  $i$ -та вредност  $h_i$  представља  $y$  координату хоризонталног ласера.

**Опис излаза**

Природан број који представља минималан број ласера које Тоша и Моша морају да искључују.

**Ограничења**

$$-10^9 \leq x_1, y_1, x_2, y_2 \leq 10^9$$

$$1 \leq n, m \leq 10^5$$

$$-10^9 \leq v_i, h_i \leq 10^9$$

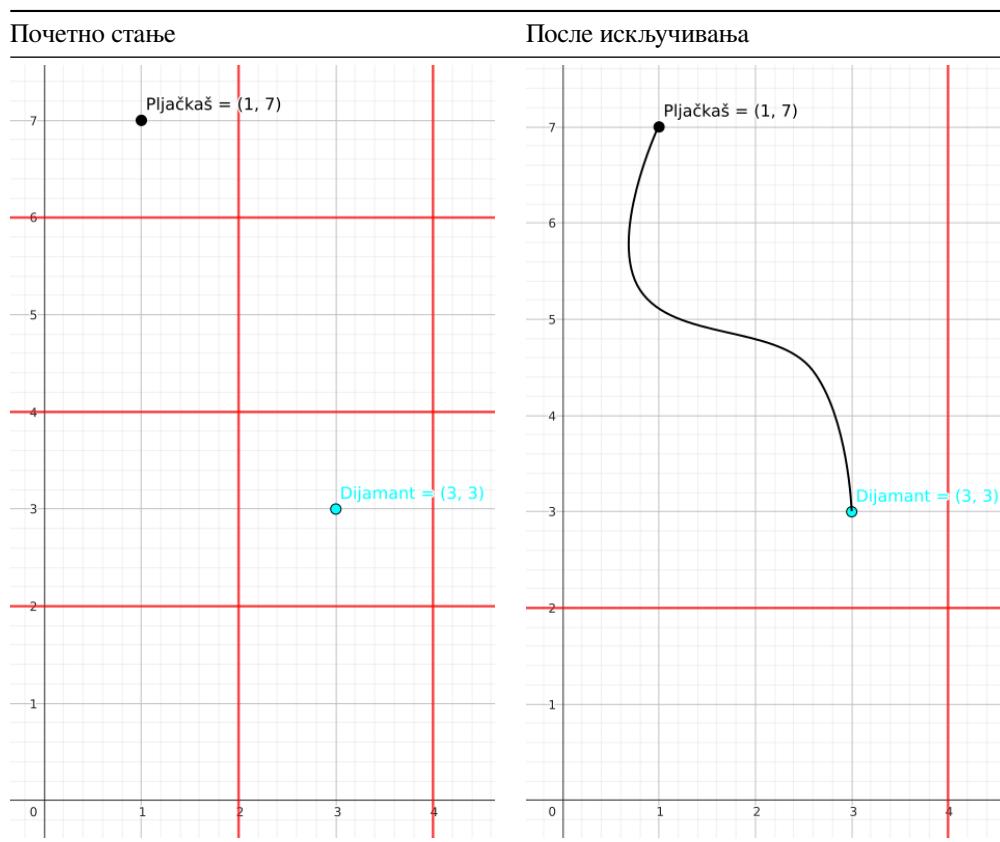
## Пример 1

Улаз      Излаз

1 7  
3 3  
2  
2 4  
3  
2 4 6

Објашњење

Минималан број ласера које морамо да искључимо је 3, на пример хоризонталне ласере на позицијама  $y = 6$  и  $y = 4$  и вертикалан ласер на позицији  $x = 2$ .



## Пример 2

Улаз

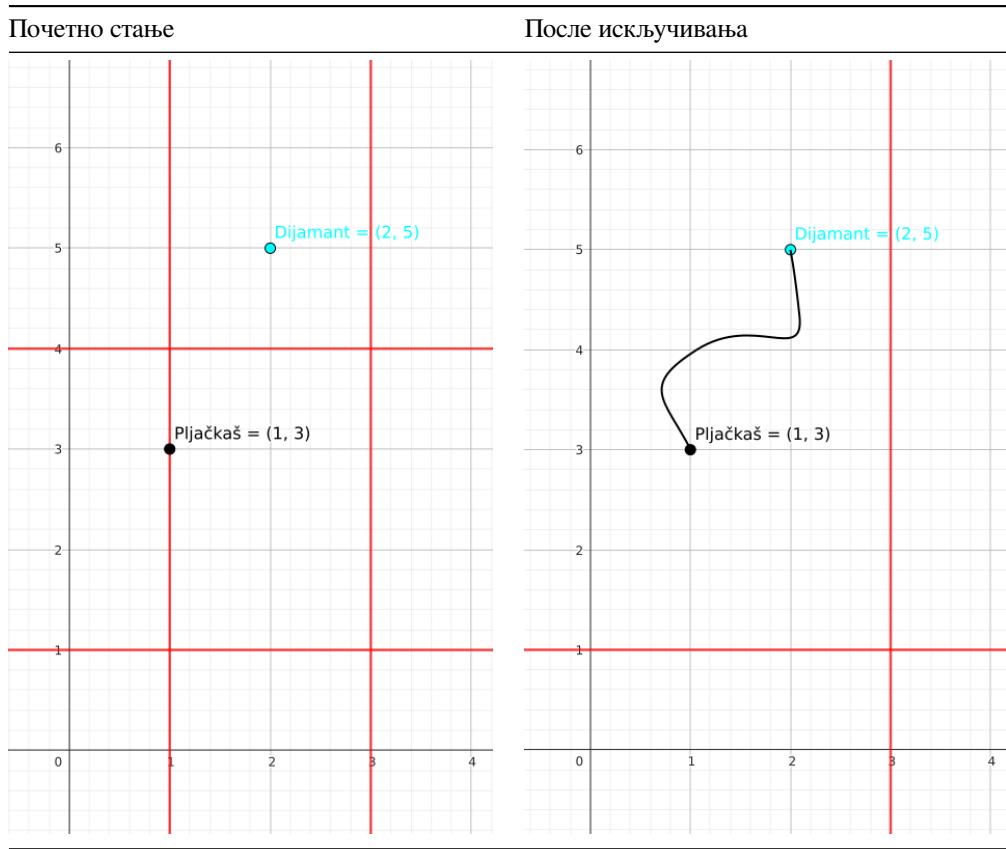
1 3  
2 5  
2  
1 3  
2  
1 4

Излаз

2

Објашњење

Минималан број ласера које морамо да искључимо је 2, на пример хоризонтални ласер на позицији  $y = 4$  и вертикалан ласер на позицији  $x = 1$ .



## Решење

### Опис главног решења

У овом блоку се описује главно решење задатка.

## Задатак: Број сегмената са производом нула

*Аутор: Милан Вујделића*

Написати програм који за дати низ целих бројева одређује број сегмената (поднизова са узастопним елемен-тима) датог низа, којима је производ елемената једнак нули.

### Опис улаза

У првом реду стандарданог улаза је број  $n$  ( $1 \leq n \leq 100000$ ), број елемената низа. У другом реду је  $n$  целих бројева из интервала  $[-100, 100]$  раздвојених по једним размаком, елементи низа.

### Опис излаза

На стандардни излаз исписати један цео број, тражени број сегмената.

### Пример 1

Улаз	Излаз
5	8
1 2 3 0 5	

*Објашњење*

То су сегменти  $1 2 3 0$ ,  $1 2 3 0 5$ ,  $2 3 0$ ,  $2 3 0 5$ ,  $3 0$ ,  $3 0 5$ ,  $0$ ,  $0 5$ .

### Пример 2

Улаз

---

```
3
0 1 0
```

Излаз

```
5
```

Објашњење

То су сегменти 0 (прва нула), 0 1, 0 1 0, 1 0, 0 (друга нула).

### Решење

Производ сегмента је нула ако и само ако је бар један од бројева у том сегменту једнак нули. Према томе, потребно је одредити број сегмената који садрже бар једну нулу.

Сваки сегмент је одређен индексом свог почетка и краја. Зато задатак можемо да решимо помоћу двоструке `for` петље, где променљиву спољне петље користимо као индекс почетка сегмента, а променљиву унутрашње петље као индекс краја. Логичка променљива `ima_nula` означава да ли у текућем сегменту низа постоји елемент једнак нули.

```
n = int(input())
a = [int(x) for x in input().split()]
br = 0
for i in range(n):
    ima_nula = False
    for j in range(i, n):
        if a[j] == 0:
            ima_nula = True
    if ima_nula:
        br += 1
print(br)
```

Мана претходног решења је да се две `if` наредбе извршавају за сваки сегмент, а сегмената има нешто више од  $n^2/2$ , где је  $n$  дужина низа. То може да буде преко  $50\,000 \cdot 50\,000/2 = 1\,250\,000\,000$  операција, што је превише за расположиво време.

Претходно решење можемо да побољшамо тако што од сваког почетка сегмента (позиција  $i$  у низу) тражимо први елемент низа једнак нули. Нека је та прва нула на позицији  $j$ . Тада од свих сегмената који почињу на позицији  $i$ , нулу садрже сегменти  $\$[i..j]$ ,  $[i..j + 1]$ ,  $[i..j + 2]$ , ...,  $[i..n - 1]$ . Таквих сегмената има  $n - j$ , па резултат (бројач тртажених сегмената) можемо да увећамо за  $n - j$  и тиме смо обрадили све сегменте са почетком  $i$ . У случају да почевши од позиције  $i$  нема нула у низу, променљива  $j$  ће достићи вредност  $n$ , па додавање  $n - j = 0$  на резултат не мења вредност резултата.

```
n = int(input())
a = [int(x) for x in input().split()]
br = 0
for i in range(n):
    j = i
    while j < n and a[j] != 0:
        j += 1
    br += n-j
print(br)
```

Неповољан случај за претходно решење је да у низу има врло мало нула, или их нема уопште. У том случају за сваки почетак  $i$  сегмента, унутрашња петља која тражи нулу иде до краја низа, па је број операција поново једнак укупном броју сегмената.

Да бисмо добили још боље решење, треба да приметимо да приликом тражења нула у низу, променљива  $j$  не мора за сваки нови почетак  $i$  да почне од тог  $i$ , јер смо већ утврдили да између позиција  $i$  и  $j$  нема нула у низу. Захваљујући томе, када је  $j > i$  можемо да уштедимо време не враћајући вредност  $j$  на почетак интервала. Пошто се у овом решењу  $j$  никад не смањује, укупан број операција је сразмеран са дужином низа.

```

n = int(input())
a = [int(x) for x in input().split()]
br = 0
j = 0
for i in range(n):
    if j < i:
        j = i

    while j < n and a[j] != 0:
        j+=1

    br += n-j

print(br)

```

## Задатак: Један-два-три-четири

*Аутор: Милан Вујелић*

Написати програм, који за дате природне бројеве  $N$  и  $P$  редом исписује  $P$  лексикографски најмањих уређених  $N$ -торки природних бројева, таквих да важи:

- сваки елемент  $N$ -торке је један од бројева 1, 2, 3, 4
- у свака 4 узастопна елемента  $N$ -торке јављају се бар три различите вредности

Нека је  $B_N$  број свих  $N$ -торки које испуњавају ове услове. Ако је  $B_N < P$ , исписати свих тих  $B_N$   $N$ -торки.

### Опис улаза

У првом реду стандарданог улаза број  $N$ ,  $4 \leq N \leq 50$ . У другом реду стандарданог улаза број  $P$ ,  $1 \leq P \leq 300$ .

### Опис излаза

У сваком од  $\min(B_N, P)$  редова исписати по једну  $N$ -торку бројева 1, 2, 3, 4. Елементе  $N$ -торке исписати без размака између њих.

### Пример 1

Улаз	Излаз
5	11231
17	11232 11233 11234 11241 11242 11243 11244 11321 11322 11323 11324 11341 11342 11343 11344 11421

### Објашњење

Дати улаз значи да је потребно да се испише првих 17 уређених енторки за  $n = 5$ , тј. првих 17 уређених петорки.

Уређене петорке бројева 1, 2, 3, 4 лексикографским редом су:

---

11111, 11112, 11113, 11114, 11121, 11122, 11123, 11124, 11131, 11132, 11133, 11134, 11141, 11142, 11143, 11144,

11211, 11212, 11213, 11214, 11221, 11222, 11223, 11224, 11231, 11232, 11233, 11234, 11241, 11242, 11243, 11244,

11311, 11312, 11313, 11314, 11321, 11322, 11323, 11324, 11331, 11332, 11333, 11334, 11341, 11342, 11343, 11344,

11411, 11412, 11413, 11414, 11421, 11422, 11423, 11424, 11431, 11432, 11433, 11434, 11441, 11442, 11443, 11444...

Многе од ових петорки не испуњавају услов да међу свака четири узастопна броја у петорци буду бар три различите вредности. Такве су, на пример, све оне петорке које у прва четири броја имају бар три јединице, или две јединице и две двојке, или две јединице и две тројке. Зато је лексикографски прва петорка која испуњава поменути услов 11231. Следеће петорке које испуњавају услов су (лексикографским редом) 11232, 11233 и даље како је наведено у излазу.

## Пример 2

Улаз

14  
10

Излаз

```
11231123112311  
11231123112312  
11231123112313  
11231123112314  
11231123112321  
11231123112324  
11231123112331  
11231123112334  
11231123112341  
11231123112342
```

## Решење

Тражене  $N$ -торке је најлакше генерисати помоћу рекурзивне функције. Довољно је да као параметар функцији проследимо број генерисаних елемената  $N$ -торке.

Одмах на уласку у функцију проверавамо да ли је исписан тражени број  $N$ -торки. Ако јесте, нема потреба да се даље генеришу  $N$ -торке и функција може да заврши са радом.

Ако је број генерисаних елемената текуће  $N$ -торке једнак  $N$ , функција исписује елементе  $N$ -торке и завршава са радом. У противном, за сваки од наставака 1, 2, 3, 4 проверавамо да ли је одговарајући, а за оне који су одговарајући рекурзивно настављамо са генерисањем.

```
def dobar_sufiks(i):  
    if i < 2:  
        return True  
    if i == 2:  
        return a[i] != a[i - 1] or a[i] != a[i - 2]  
  
    br = [0, 0, 0, 0, 0]  
    for k in range(i-3, i+1):  
        br[a[k]] = 1  
    return br[1] + br[2] + br[3] + br[4] > 2
```

```
def produzi_niz(i):  
    global ispisati  
    if ispisati == 0:
```

```
return

if i == n:
    for k in range(n):
        print(a[k], end = ' ')
    print()
ispisati -= 1
return

for c in range(1, 5):
    a[i] = c
    if dobar_sufiks(i):
        produzi_niz(i + 1)

n = int(input())
ispisati = int(input())
a = [0] * n
produzi_niz(0)
```

## Глава 2

# Квалификације (2. круг)

### Задатак: Закуцавање

Аутор: Милан Вујелић

Кошаркашки обруч за децу је на висини 240 центиметара, а пречник кошаркашке лопте је 24 центиметара. Виктор са пода може да дохвати висину  $h$  центиметара. Колико Виктор треба да скочи да би могао да закуца лопту у кош?

Претпоставља се да је за закуцавање довољно досегнути висину која је за пречник лопте изнад висине обруча.

#### Опис улаза

На стандардном улазу је цео број  $h$  ( $100 \leq h \leq 280$ ), висина у центиметрима коју Виктор дохвата са пода.

#### Опис излаза

На стандардни излаз исписати један цео број, потребну висину скока.

#### Пример 1

Улаз	Излаз
220	44

#### Пример 2

Улаз	Излаз
270	0

#### Решење

На основу текста задатка закључујемо да Виктор треба да досегне висину од  $240 + 24 = 264$  центиметара да би могао да закуца. Нека је  $h$  висина коју Виктор дохвата са пода.

Потребна висина скока је  $264 - h$ , осим када је овај број негативан. У том случају потребан скок је нула (не може се скакати на висину ниже од пода). Према томе, одговор је већи од борђева  $264 - h$  и 0.

```
h = int(input())
potrebno = max(264 - h, 0)
print(potrebno)
```

Ако висину  $h$  или већу висину Виктор може да досегне са пода, потребан скок је 0, а у противном је потребан скок  $264 - h$ . Ова два случаја можемо да разликујемо у програму помоћу једне `if` наредбе.

```
h = int(input())
if h > 264:
    print(0)
else:
    print(264 - h)
```

### Задатак: Непознати број

Аутор: Филип Марић

Миша је замислио један природан број. Након тога га је помножио са 8, добијени производ је сабрао са 10, добијени збир је поделио са 2 и од добијеног количника је одузео 8. Помози Цеци, Мишиној драгарици, да на основу резултата који је Миша добио погоди број који је Миша замислио.

### Опис улаза

Са стандардног улаза се уноси резултат који је Миша добио (природан број  $n$  између 1 и 1000).

### Опис излаза

На стандардни излаз исписати број који је Миша замислио, или текст **greska**, ако је негде погрешио у рачуну и не постоји начин да се од неког природног броја добије резултат који је он добио.

### Пример 1

<i>Улаз</i>	<i>Излаз</i>
17	5

*Објашњење*

Миша је замислио број 5. Затим га је помножио са 8 и добио 40. На 40 је додао 10 и добио 50. Затим је 50 поделио са 2 и добио 25. На крају је од 25 одузео 8 и добио је 17.

### Пример 2

*Улаз*

28

*Излаз*

greska

### Пример 3

*Улаз*

1

*Излаз*

1

### Решење

Да бисмо решили задатак потребно је да прво учитамо природни број  $r$ . Број  $r$  настаје тако што је Миша применио описане рачунске операције над оригиналним природним бројем  $x$ . Из текста задатка видимо да је редослед операција следећи:

$$r = (8x + 10)/2 - 8.$$

Дакле, знајући број  $r$  и редослед примењених операција, потребно је да одредимо број  $x$ . Задатак се практично своди на примењивање свих рачунаских операција из текста задатка, али обрнутим редоследом. Корак по корак, креирати решење:

$$\begin{aligned} r + 8 &= (8x + 10)/2 \\ (r + 8) \cdot 2 &= 8x + 10 \\ (r + 8) \cdot 2 - 10 &= 8x. \end{aligned}$$

На овом месту је кључни део задатка. Последњи израз који нам треба је:

$$x = ((r + 8) \cdot 2 - 10)/8.$$

Резултат  $x$  мора бити природни број. Међутим, то неће бити могуће ако десна страна израза није потпун количник: Прецизније разликујемо два случаја:

- 
- Израз  $(r+8) \cdot 2 - 10$  је дељив са 8. У том случају имамо решење које је облика:  $x = ((r+8) \cdot 2 - 10)/8$ .
  - Израз  $(r+8) \cdot 2 - 10$  није дељив са 8. У том случају, зnamо да решење не постоји и да је Миша погрешио у рачуну, па као резултат треба да испишемо `greska`.

Решење је у наставку.

```
x = int(input())
y = (x + 8) * 2 - 10
```

```
if y % 8 == 0:
    print(y // 8)
else:
    print('greska')
```

Приликом решавања једначине изрази се могу упрощавати. Важи:

$$x = ((r+8) \cdot 2 - 10)/8 = (2r + 16 - 10)/8 = (2r + 6)/8 = (r + 3)/4.$$

```
x = int(input())
if (r + 3) % 4 == 0:
    print((r + 3) // 4)
else:
    print('greska')
```

## Задатак: Ски пас

Аутор: Душан Попадић

На једном скијалишту у Србији цена дана скијања зависи од узраста. Деца млађа од 5 година не плаћају ски карту, деца од 5 до 12 година плаћају 2800 динара дан, људи од 13 до 64 године плаћају 3500 динара дан, а сениори (65+ година) плаћају 3200 динара дан. Такође, за сваки дан након 10. остварује попуст и плаћа се половина цене дана. Уколико је познато да Надежда има  $n$  година и жели да уплати  $k$  дана скијања, одредити колико укупно динара треба да плати.

### Опис улаза

У првом реду улаза се налази природан број  $n$  ( $1 \leq n \leq 140$ ). У другом реду се налази приородан број  $k$  ( $1 \leq k \leq 90$ ).

### Опис излаза

У једином реду излаза исписати колико новца треба Надежда да плати своје скијање.

### Пример

Улаз	Излаз
85	35200
12	

Објашњење

Пошто је Надежда сениор, њена цена је 3200 динара по дану. За првих 10 дана треба да плати  $10 \cdot 3200 = 32000$ , а за наредна 2 дана  $2 \cdot (3200/2) = 3200$ .

### Решење

Да бисмо решили задатак потребно је да прво учитамо природне бројеве  $n$  и  $k$  који редом означавају број година скијаша и број дана за које потребна ски карта.

Решење задатка можемо да поделимо у две целине. Прва целина је одређивање цене ски карте по дану на основу броја година скијаша. Нека се та променљива назива *сепа*. Можемо претпоставити да је основна цена заправо деција, па променљиву *сепа* можемо иницијализовати на 0. Да бисмо до краја одредили цену ски карте по дану, треба нам једноставно гранање у којем ћемо испитати припадност вредности  $n$  датим интервалима и доделити променљивој *сепа* одговарајућу вредност. Укратко, вршимо следеће провере:

- Ако је број година између 5 и 12, цена по дану је 2800.

2. Ако је број година између 13 и 64, цена по дану је 3500.
3. Ако је број година већи или једнак 65, цена по дану је 3200.

Друга целина у задатку је одређивање укупне цене ски карте. Да бисмо то урадили, потребно је да испитамо број дана за које се купује ски карта.

1. Ако је број дана мањи или једнак 10, укупна цена ће бити  $k \cdot cena$ .
2. Ако је број дана већи од 10, првих десет дана се рачуна по пуној цени, док за сваки следећи дан треба урачунати попуст од 50%. Због тога, укупну цену рачунамо по формулама  $10 * cena + (k - 10) \cdot cena / 2$ .

```
n = int(input())
k = int(input())

cena = 0
if (n >= 5 and n <= 12):
    cena = 2800
elif (n>12 and n <= 64):
    cena = 3500
elif (n >= 65):
    cena = 3200

if (k <= 10):
    print(int(k*cena))
else:
    print(int(10*cena + (k-10)*cena/2))
```

Алтернативно елегантније решење може се добити издвајањем логике којом се рачуна укупна цена у посебну функцију. Поред тога, могуће је и користити уграђене функције `max` и `min` приликом одређивање укупне цене.

```
def cena(c, k):
    return int(min(10, k)*c + max(0, k-10)*c/2)

n = int(input())
k = int(input())

if (n < 5):
    print(0)
elif (n >= 5 and n <= 12):
    print(cena(2800, k))
elif (n>12 and n <= 64):
    print(cena(3500, k))
elif (n >= 65):
    print(cena(3200, k))
```

## Задатак: Гориво

*Аутор: Владимир Кузмановић*

Пера, Жика и Мика су одлучили да иду на море колима и да се сменjuју за воланом током пута. Први је за воланом био Пера и превезао је  $p$  километара, затим је Жика превезао  $z$  километара и на крају је возио Мика који је превезао  $m$  километара. Познато је да на пут крећу са пуним резервоаром горива и да аутомобил са пуним резервоаром може да пређе  $k$  километара. Написати програм који одређује ко је био за воланом на сваком стајању за допуну горива. Подразумева се да је приликом сваког стајања за воланом био возач који довози аутомобил на пумпу. У случају да Пера, Жика и Мика треба да се зауставе на пумпи на крају путовања, подразумевати да ће прво сипати гориво, па тек онда ићи у хотел и да се у том случају занемарује растојање до хотела.

### Опис улаза

У сваком од 4 реда стандардног улаза налази се по један природан број. Бројеви који се учитавају редом представљају вредности  $p, z, m, k$ . Бројеви  $p, z, m, k$  су између 1 и 100000.

---

## Опис излаза

На стандардни излаз исписати имена возача који су били за воланом у тренутку сваког доласка на пумпу. Свако име исписати у посебном реду. Имена возача треба исписати малим словима енглеског алфабета.

### Пример 1

Улаз	Излаз
700	рега
900	зика
800	мика
600	мика

#### Објашњење

Прву паузу за сипање горива треба да направе након 600km, када је за воланом Пера. Следећу паузу праве након следећих 600km, када је за воланом Жика. Наредну паузу праве након још 600km када је за воланом Мика. Последњу паузу праве након још 600km, тј. на самом крају пута, када је опет за воланом Мика.

### Пример 2

#### Улаз

100  
100  
100  
100

#### Излаз

рега  
зика  
мика

#### Објашњење

Сваки возач је превезао тачно 100km и аутомобил има дomet од 100km са пуним резервоаром. Прво стајање ће бити након 100km, што значи да на пумпу Пера довози аутомобил и завршава својих 100km вожње. Затим, Жика седа за волан и вози својих 100km, чиме ће потрошити цео резервоар горива, па је он следећи који довози аутомобил на пумпу. Након сипања горива, Мика седа за волан и вози својих 100km. Мика ће потрошити цео резервоар горива и он је следећи који довози аутомобил на пумпу. Након што сипају гориво, стигли су на своје одредиште.

### Пример 3

#### Улаз

300  
500  
750  
400

#### Излаз

зика  
зика  
мика

#### Решење

Да бисмо решили задатак потребно је прво да учитамо све неопходне податке са стандардног улаза. Потребно је да користимо целе бројеве да бисмо учитане вредности  $p$ ,  $z$ ,  $m$  и  $k$  представили на исправан начин у програму.

Затим, потребно је да уочимо да нам треба укупан пут који су Пера, Мика и Жика прешли на свом путовању. Дакле, рачунамо збир:

$$ukupno = p + z + m.$$

Након тога, потребно је да уочимо да на сваких  $k$  пређених километара Пера, Жика и Мика морају да направе паузу да би допунили резервоар. Прецизније, треба да откријемо ко је био за воланом на сваких  $k$  километара када су правили паузу.

Да бисмо то постигли, потребно је бројач у петљи иницијализујемо са вредности  $k$ , да га након сваке итерације увећавамо за  $k$  све док не пређемо вредност *ukupno* и да у свакој итерацији петље проверавамо ко је био за воланом у датом тренутку. Провере вршимо тако што испитамо да ли вредност бројача припада одговарајућем интервалу:

1. Ако је вредност бројача мања или једнака  $p$ , онда је за воланом био Пера.
2. Ако је вредност бројача већа од  $p$  и мања или једнака  $p + z$ , онда је за воланом био Жика.
3. Ако је вредност бројача већа од  $p + z$ , онда је за воланом био Мика.

Одговарајуће решење је у наставку.

```
p = int(input())
z = int(input())
m = int(input())
k = int(input())

ukupno = p + z + m
for s in range(k, ukupno+1, k):
    if s <= p:
        print("pera")
    elif p < s and s <= p + z:
        print("zika")
    else:
        print("milka")
```

Алтернативно решење прати исту логику као и главно решење, само је уместо петље `for` искоришћена петља `while`.

```
p = int(input())
z = int(input())
m = int(input())
k = int(input())

ukupno = p + z + m
kilometraza = k
while (kilometraza <= ukupno):
    if kilometraza <= p:
        print("pera")
    elif p < kilometraza and kilometraza <= p + z:
        print("zika")
    else:
        print("milka")
    kilometraza += k
```

## Задатак: Прва и последња цифра

*Аутор: Филип Марић*

На капији древног града налази се загонетка у којој се крије лозинка за улазак у тај град. Загонетка се састоји од слова и бројева поређаних у више редова. Међутим, пуно симбола је наведено само да би збунило оне који желе да уђу у град. Наиме, за одређивање лозинке битне су само прва и последња цифара у сваком реду (а у сваком реду постоје бар две цифре). Од њих се у сваком реду добија по један двоцифрени број, а збир свих тих двоцифрених бројева нам даје лозинку за улаз.

### Опис улаза

Са стандардног улаза се уноси број  $n$  ( $1 \leq n \leq 100$ ), а затим  $n$  редова који садрже речи које се састоје од највише 50 слова и цифара (у сваком реду сигурно постоје бар две цифре).

---

## Опис излаза

На стандардни излаз исписати лозинку.

### Пример 1

Улаз	Излаз
3	39
12345	
a1b2c3	
abc121abc	

#### Објашњење

- У првом реду је прва цифра 1, а последња 5, па је двоцифрен број 15.
- У другом реду је прва цифра 1, а последња 3, па је двоцифрен број 13.
- У трећем реду је прва цифра 1, а последња 1, па је двоцифрен број 11.

Збир ова три двоцифrena броја је  $15+13+11=39$ .

### Пример 2

#### Улаз

1  
7a1

#### Излаз

71

#### Решење

У петљи учитавамо и обрађујемо једну по једну реч са улаза. Потребно је да за сваку реч одредимо прву и последњу цифру. За то ћемо употребити две променљиве `prva_cifra` и `poslednja_cifra` и анализираћемо један по један карактер текуће речи. Променљиву `poslednja_cifra` ћемо ажурирати сваки пут када нађемо на цифру, тако да ће њена завршна вредност заиста бити последња цифра речи. Променљивој `prva_cifra` ћемо иницијално доделити вредност -1 и ажурираћемо је само ако јој је вредност -1. Након тога она ће добити вредност прве цифре и надаље неће бити ажурирана. Од издвојених цифара лако формирајмо број и додајемо га на укупан збир.

```
n = int(input())
lozinka = 0
for i in range(n):
    # ucitavamo novu rec
    rec = input()

    # odredjujemo prvi i poslednju cifru te reci
    prva_cifra = -1
    for c in rec:
        if c.isdigit():
            poslednja_cifra = int(c)
            if prva_cifra == -1:
                prva_cifra = int(c)

    # formiramo broj od te dve cifre i uvećavamo ukupan zbir za taj broj
    lozinka += 10*prva_cifra + poslednja_cifra

print(lozinka)
```

## Задатак: Најгушћи подскуп

Аутор: Иван Дреџун

Дат је скуп природних бројева. Густина његовог подскупа рачуна се као број елемената подскупа подељен разликом између његовог највећег и најмањег елемента. Напиши програм који одређује подскуп са највећом густином чији је број елемената тачно  $k$ . Уколико постоји више подскупова који испуњавају овај услов, исписати онај чији су елементи најмањи.

### Опис улаза

Са стандардног улаза се уносе величина датог скупа  $n$  ( $1 \leq n \leq 2 \cdot 10^5$ ) и величина траженог подскупа  $k$  ( $1 \leq k \leq n$ ). У наредном реду се уносе елементи скупа  $a_i$  ( $1 \leq a_i \leq 10^9$ ) одвојени размаком.

### Опис излаза

На стандардни излаз исписати елементе траженог подскупа у растућем поретку. Уколико постоји више подскупова који испуњавају услов задатка, исписати онај чији су елементи најмањи.

#### Пример 1

<i>Улаз</i>	<i>Излаз</i>
6 3	6 8 10
8 3 10 1 13 6	

#### Пример 2

<i>Улаз</i>	<i>Излаз</i>
5 2	4 5
8 5 1 7 4	

### Решење

Како се у задатку тражи подскуп фиксне величине, доволно је пронаћи такав подскуп да му се највећи и најмањи елемент разликују што мање. Ако имамо скуп  $\{1, 3, 4, 6, 8\}$  и тражимо подскуп величине 3, нема потребе да разматрамо подскуп  $\{1, 3, 6\}$ , када знамо да подскуп  $\{1, 3, 4\}$  сигурно има већу густину. Дакле, доволно је да сортирамо дати скуп и посматрамо само поднизове узастопних елемената тражене дужине у добијеном низу. Тражимо такав подниз да му се последњи и први елемент разликују што мање.

Пошто број елемената у скупу може бити велик, потребно је користити неки од ефикасних алгоритама за сортирање (најједноставније је да употребимо библиотечку функцију за сортирање).

```

n, k = [int(x) for x in input().split()]
a = [int(x) for x in input().split()]
a.sort()

mind = a[n - 1] - a[0]
mini = 0
for i in range(k - 1, n):
    if a[i] - a[i - k + 1] < mind:
        mind = a[i] - a[i - k + 1]
        mini = i - k + 1

for i in range(mini, mini + k):
    print(a[i], end=' ')
print()

```

## Задатак: Месец рођења

Автор: Милан Вујделић

У матичном броју грађана трећа и четврта цифра означавају месец рођења.

Написати програм који за дати матични број исписује прва три слова месеца рођења.

### Опис улаза

На стандардном улазу налази се низ од 13 цифара без размака.

### Опис излаза

На стандардни излаз исписати један од текстова jan, feb, mar, apr, maj, jun, jul, avg, sep, okt, nov, dec.

#### Пример 1

<i>Улаз</i>	<i>Излаз</i>
0312998714216	dec

---

## Објашњење

Трећа и четврта цифара датог броја (записане заједно) су 12, па је месец рођења децембар.

### Пример 2

Улаз

2103009718495

Излаз

мар

### Решење

Најпре треба одредити редни број месеца на основу матичног броја. Поступак одређивања овог броја види се из програма.

Када имамо редни број месеца, можемо да га искористимо као индекс у низу свих могућих одговора, којих има 12. Пошто могући бројеви месеца почињу од 1, а индекси у низу од 0, можемо да додамо један празан стринг на почетак низа. На тај начин одговор јави се налази на позицији са индексом 1, а слично важи и за остале месеце, тако да је индекс сваког могућег одговора управо редни борј месеца.

```
nazivi_meseci = (
    '', 'jan', 'feb', 'мар', 'апр', 'май', 'јун',
    'јул', 'авг', 'сеп', 'окт', 'нов', 'дек')  
  
jmbg = input()  
mesec = int(jmbg[2:4])  
print(nazivi_meseci[mesec])
```

Друго решење:

Најпре треба да издвојимо из матичног броја део који садржи цифре на позицијама 3 и 4. За вредност тог дела има 12 могућности, па случајеве можемо да разликујемо помоћу 12 надовезаних наредби if. Остаје да у сваком од случајева испишемо правилан одговор.

```
jmbg = input()  
mesec = jmbg[2:4]  
if mesec == '01':  
    print('jan')  
elif mesec == '02':  
    print('feb')  
elif mesec == '03':  
    print('мар')  
elif mesec == '04':  
    print('апр')  
elif mesec == '05':  
    print('май')  
elif mesec == '06':  
    print('јун')  
elif mesec == '07':  
    print('јул')  
elif mesec == '08':  
    print('авг')  
elif mesec == '09':  
    print('сеп')  
elif mesec == '10':  
    print('окт')  
elif mesec == '11':  
    print('нов')  
elif mesec == '12':  
    print('дек')
```

## Задатак: Врабац и мачка

*Аутор: Милан Вујделић*

Врабац лети унутар интервала  $[a, b]$  на бројевној правој ( $a < b$ ), а мачка шета унутар интервала  $[c, d]$  ( $c < d$ ). Интервали садрже своје крајње тачке. Врабац зна да, када слети, мачка ће му прићи најближе што може. Зато врабац жeli да слети на такво место да након што му мачка приђe, он буде што даљe од њe. Којe јe највећe растојањe на коме врабац може да буде након приласка мачке?

### Опис улаза

На стандардном улазу су цели бројеви  $a, b, c, d$ , сваки у посебном реду. Сви бројеви су позитивни и нису већи од 2000000000 (две милијарде).

### Опис излаза

На стандардни излаз исписати један цео број, највеће растојање које врабац може да обезбеди.

#### Пример 1

Улаз	Излаз
2	6
20	
4	
14	

*Објашњење*

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
.	.	<b>в</b>	.	.	.																		
.	.	.	<b>м</b>	.	.	.																	

Врабац може да се креће по словима **в** (и да лети изнад њих), а мачка да се креће по словима **м**. За врабца је најбоље да слети у тачку 20, а мачка ће након тога да му приђe до тачке 14. Тада је растојање између њих једнако 6 и то је тражено растојање.

#### Пример 2

*Улаз*

5
12
4
15

*Излаз*

0
---

#### Пример 3

*Улаз*

100
200
120
300

*Излаз*

20
----

#### Пример 4

*Улаз*

400
700
100
200

---

Излаз

500

### Решење

Ако макча може да стане у сваку тачку доступну врапцу, онда је врапцу свеједно где ће да стане, јер ће га мачка свакако ухватити, а тражени резултат је у том случају 0.

Ако постоји тачка која је врапцу доступна а мачки није, онда је та тачка или лево или десно од целог мачкиног интервала. Од свих таквих тачака лево од мачкиног интервала најбоља је крајња лева тачка интервала врапца, а од свих таквих тачака десно од мачкиног интервала најбоља је крајња десна тачка интервала врапца.

Према томе, постоје само две тачке које су кандидати за најбољу тачку врапца, а то су крајеви његовог интервала. Једна или обе ове тачке могу да буду врло лоше за врапца, али сигурни смо да никаква трећа тачка не може да буде боља од обе ове тачке и зато нема потребе да размишљамо о било којим додатним тачкама осим ове две.

Остаје још само да се одреди колико највише врабац може да буде удаљен од мачке у свакој од својих крајњих тачака, када му мачка приђе најближе што може. Начин на који можемо да израчунамо поменута растојања и њихов максимум може да се разуме читањем програма.

```
v_poc = int(input())
v_kraj = int(input())
m_poc = int(input())
m_kraj = int(input())

if v_poc < m_poc:
    d_levo = m_poc - v_poc
else:
    d_levo = 0

if v_kraj > m_kraj:
    d_desno = v_kraj - m_kraj
else:
    d_desno = 0

d = max(d_levo, d_desno)
print(d)
```

Други начин да напишемо програм на основу истог размишљања је овај. Нека су  $[v_{poc}, v_{kraj}]$  и  $[m_{poc}, m_{kraj}]$  интервали врапца и мачке редом.

- ако је интервал врапца у потпуности садржан у интевалу мачке, тј. ако је  $m_{poc} \leq v_{poc} < v_{kraj} \leq m_{kraj}$ , резултат је 0
- у противном, бар једна од разлика  $m_{poc} - v_{poc}$ ,  $v_{kraj} - m_{kraj}$  је позитивна, па је резултат једнак већој од њих (није битно што друга разлика може да буде негативна).

```
v_poc = int(input())
v_kraj = int(input())
m_poc = int(input())
m_kraj = int(input())

if v_poc > m_poc and v_kraj < m_kraj:
    print(0)
else:
    print(max(m_poc - v_poc, v_kraj - m_kraj))
```

Лоше решење би било да се за сваку тачку доступну врапцу одређује најближи положај мачке, а затим најдали од свих тих најближих положаја. Резултат који се добија јесте тачан, али због много рачунања програм је врло спор.

```
# Lose rešenje
def d(x):
```

```

if x < m_poc:
    return m_poc - x
elif x < m_kraj:
    return 0
else:
    return x - m_kraj

v_poc = int(input())
v_kraj = int(input())
m_poc = int(input())
m_kraj = int(input())

rez = 0
for x in range(v_poc, v_kraj+1):
    rez = max(rez, d(x))
print(rez)

```

## Задатак: Неравнотежа

*Аутор: Милан Вујделић*

Четворо ћака првака при повратку из школе пролазе кроз парк и увек сврата на клацкалицу. Осим што воле да се клацкају, они радо покушавају да претегну једни друге, јер је клацкалица нова и добро подмазана. При томе се ћаци труде да подела буде што равноправнија, да би игра била занимљивија.

Наши ћаци прво седну мирно по двоје на сваку страну клацкалице без својих школских торби, а затим, ако је клацкалица у неравнотежи, ћаци са лакше стране узму неколико школских торби. Након тога почиње игра претезања.

Написати програм који за дате масе четворо првака одговара на питање да ли они могу да постигну да збир маса ћака и могућих торби буде исти на обе стране и колико **најмање** торби је потребно за то у случају потврдног одговора. Свака торба је тешка два килограма.

### Опис улаза

На стандардном улазу су четири цела позитивна броја, мања од 100, сваки у посебном реду. Ови бројеви представљају масе ћака.

### Опис излаза

У први ред стандардног излаза исписати **да** или **не**. Уколико је у први ред исписана реч **да**, у други ред исписати **најмањи могућ број** потребних торби, а то мора да буде **0, 1, 2, 3** или **4**.

### Пример 1

Улаз	Излаз
40	da
35	1
37	
44	

### Објашњење

Када на једну страну седну деца од 35 и 44 килограма, а на другу од 37 и 40, неравнотежа је  $(35 + 44) - (37 + 40) = 79 - 77 = 2$ , па равнотежа може да се постигне тако што лакши пар узме једну школску торбу.

### Пример 2

#### Улаз

44  
41  
30  
43

---

*Излаз*

*не*

*Објашњење*

Како год да се ђаци распореде без торби, збир њихових маса са једне стране је за више од 8 килограма већи од збира са друге стране, па равнотежу није могуће постићи додавањем торби.

### Пример 3

*Улаз*

35  
35  
35  
35

*Излаз*

*да*  
0

### Решење

Нека је дат било који распоред деце на клацкалици, по два детета са сваке стране. Разлику збирива маса на левој и десној страни зваћемо неравнотежом.

Можемо да приметимо да у случају да је неравнотежа при једном распореду непарна, онда је неравнотежа при сваком распореду непарна, па додавањем торби од по два килограма ниједна неравнотежа не може да постане парна, а тиме ни нула. То значи да у случају једне непарне неравнотеже нема решења.

У случају да је најмања неравнотежа већа од укупне масе свих торби, тј. од 8 килограма, поново нема решења. Преостаје случај када је најмања неравнотежа парна и мања од 10. Тада равнотежа може да се постигне, а тражени број торби добијамо дељењем најмање неравнотеже са 2.

Остаје да одредимо најмању неравнотежу.

Могући распореди су:

- да ђаци маса  $a$  и  $b$  седе наспрам ђака маса  $c$  и  $d$
- да ђаци маса  $a$  и  $c$  седе наспрам ђака маса  $b$  и  $d$
- да ђаци маса  $a$  и  $d$  седе наспрам ђака маса  $b$  и  $c$

Најмању неравнотежу можемо да израчунамо тако што израчунамо све три могуће неравнотеже и нађемо њихов минимум.

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

neravnoteza = min(abs(a+b-c-d), abs(a+c-b-d), abs(a+d-b-c))
if neravnoteza <= 8 and neravnoteza % 2 == 0:
    print('да')
    print(neravnoteza // 2)
else:
    print('не')
```

Најмања неравнотежа се добија када дете најмање масе и дете највеће масе седе на једној страни клацкалице, а остала два детета на другој страни. Докажимо ово тврђење.

Доказ: означимо масе ђака словима  $a, b, c, d$ , тако да важи  $a \leq b \leq c \leq d$  (тако бирамо ознаке). Користећи ове ознаке, тврђење је да је неравнотежа најмања ако ђаци маса  $a$  и  $d$  седе наспрам ђака маса  $b$  и  $c$ .

Као што већ знајмо, осим наведеног распореда седења постоје још само два: да  $a$  и  $c$  седе наспрам  $b$  и  $d$ , или да  $a$  и  $b$  седе наспрам  $c$  и  $d$ . Доказаћемо да неравнотежа при овим распоредима не може да буде мања од неравнотеже при распореду  $a$  и  $d$  наспрам  $b$  и  $c$ .

Разликоваћемо два случаја:  $a + d \geq b + c$  и  $b + c > a + d$ .

1. Нека је  $a + d \geq b + c$ . Тада:

$$\begin{aligned} N_{ab} &= |(c+d) - (a+b)| = (d-b) + (c-a) \geq (d-b) - (c-a) = (d+a) - (b+c) = |(d+a) - (b+c)| = N_{ad} \\ N_{ac} &= |(b+d) - (a+c)| = (d-c) + (b-a) \geq (d-c) - (b-a) = (d+a) - (b+c) = |(d+a) - (b+c)| = N_{ad} \end{aligned}$$

1. Нека је  $b + c > a + d$ . Тада:

$$\begin{aligned} N_{ab} &= |(c+d) - (a+b)| = (c-a) + (d-b) \geq (c-a) - (d-b) = (b+c) - (a+d) = |(b+c) - (a+d)| = N_{ad} \\ N_{ac} &= |(b+d) - (a+c)| = (b-a) + (d-c) \geq (b-a) - (d-c) = (b+c) - (a+d) = |(b+c) - (a+d)| = N_{ad} \end{aligned}$$

Овим је доказ завршен. Следи да најмања неравнотежа може да се добије и као  $|(b+c) - (a+d)|$ , где су  $a$  и  $d$  најмања и највећа маса детета.

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())
levo = max(a, b, c, d) + min(a, b, c, d)
desno = a+b+c+d-levo
negavnoteza = abs(levo-desno)
if negavnoteza % 2 == 0 and negavnoteza < 10:
    print('da')
    print(negavnoteza // 2)
else:
    print('ne')
```

## Задатак: Дорћол опен

Два другара Маре и Вук организују тениски турнир Дорћол опен се игра по систему “на испадање” (губитник меча испада, победник пролази у наредну рунду) и има  $k$  кола и на њему учествује  $2^k$  такмичара. Договорили су се да Маре буде главни судија, а Вук директор турнира. Маре није превише систематично бележио резултате и Вуку их је донео у следећој форми: за сваки меч пише ко је у њему победио и ко је изгубио, али не пише у ком је колу одигран. Помозите Вуку да на основу ових података направи поредак играча на турниру.

### Опис улаза

У првом реду се налази природни број  $k$  - број кола које турнир има ( $1 \leq k \leq 15$ ). У сваком наредном реду се налази резултат једног од мечева на турниру у формату идентификациони број победника и идентификациони број губитника, раздвојени размаком. Сваки такмичар има једнствен идентификациони број у распону од 0 до  $2^k - 1$ .

### Опис излаза

У првом реду исписати победника турнира. У другом реду исписати другопласираног. У сваком наредном реду исписати идентификационе бројеве играча који су испали у колу пре (у трећем реду полуфинале, у четвртом четвртфинале, у петом осмина финала итд. колико год да их има). У сваком реду је распоред идентификационих бројева произвољан.

### Пример 1

Улаз	Излаз
2	3
3 0	0
3 1	1 2
0 2	

*Објашњење*

---

У првом колу је играч 3 победио играча 1, а играч 0 играча 2. У другом колу је играч 3 победио играча 0. Дакле победник турнира је играч 3, играч 0 је други, а играчи 1 и 2 су испали у полуфиналу.

## Пример 2

Улаз

```
3
7 2
6 7
6 5
1 4
6 1
3 0
1 3
```

Излаз

```
6
1
3 7
0 4 2 5
```

## Решење

```
k = int(input()) # broj kola
br_igraca = 2**k
broj_meceva = br_igraca - 1
a = [[0, i] for i in range(br_igraca)]
for i in range(broj_meceva):
    x, y = [int(s) for s in input().split()]
    a[x][0] += 1

a.sort(reverse=True)
for i in range(br_igraca):
    if a[i][0] <= k:
        print()
        k -= 1
    print(a[i][1], end=' ')
print()

# vise prolaza kroz niz brojaca pobeda

k = int(input()) # broj kola
br_igraca = 2**k
broj_meceva = br_igraca - 1
a = [0] * br_igraca
for i in range(broj_meceva):
    dobio, izgubio = [int(s) for s in input().split()]
    a[dobio] += 1

for br_pobeda in range(k, -1, -1):
    for i in range(br_igraca):
        if a[i] == br_pobeda:
            print(i, end=' ')
    print()

# vise prolaza kroz niz brojaca pobeda

k = int(input()) # broj kola
br_igraca = 2**k
broj_meceva = br_igraca - 1
```

```

a = [0] * br_igraca
for i in range(br_mecova):
    dobio, izgubio = [int(s) for s in input().split()]
    a[dobio] += 1

for br_pobeda in range(k, -1, -1):
    print(''.join([str(i) for i in range(br_igraca) if a[i] == br_pobeda]))

```

## Задатак: Непарни делиоци

*Автор: Ођићен Тешлић*

Напиши програм који за унети природни број  $n$  одређује најмањи природан број  $m$  такав да  $n \cdot m$  има непаран број делилаца у скупу природних бројева (укључујући 1 и  $n \cdot m$ ).

### Опис улаза

У једином реду стандардног улаза се налази један природан број  $n$  ( $n \leq 10^{12}$ ).

### Опис излаза

У једином реду стандардног излаза исписати један природан број  $m$ , који представља најмањи природан број такав да  $n \cdot m$  има непаран број делилаца у скупу природних бројева.

Тест примери су подељени у четири независне (дисјунктне) групе: - У тест примерима вредним 10 поена:  $n \leq 10^2$ . - У тест примерима вредним 10 поена:  $n \leq 10^3$ . - У тест примерима вредним 30 поена:  $n \leq 10^6$ . - У тест примерима вредним 50 поена:  $n \leq 10^{12}$ .

### Пример 1

Улаз	Излаз
3	3

*Објашњење*

Делиоци броја  $3 \cdot 3 = 9$  су 1, 3, 9. Лако се види да бројеви  $3 \cdot 1 = 3$  и  $3 \cdot 2 = 6$  имају 2, односно 4 делиоца у скупу природних бројева.

### Пример 2

*Улаз*

12

*Излаз*

3

*Објашњење*

Делиоци броја  $12 \cdot 3 = 36$  су 1, 2, 3, 4, 6, 9, 12, 18, 36. Лако се види да бројеви  $12 \cdot 1 = 12$  и  $12 \cdot 2 = 24$  имају 6, односно 8 делиоца у скупу природних бројева.

### Решење

#### Решење када $n \leq 10^2$

У овом случају је довољно испитати све бројеве  $1, 2, 3, \dots, n$  као кандидате за  $m$ , проверити петљом од 1 до  $n \cdot m$  колико бројева у интервалу  $[1, n \cdot m]$  дели број  $n \cdot m$  и одредити најмањи  $m$  такав да непаран број бројева у интервалу  $[1, n \cdot m]$  дели број  $n \cdot m$ . Ово решење ради у сложености  $\mathcal{O}(n^3)$ .

#### Решење када $n \leq 10^3$

У овом случају треба урадити малу модификацију првог случаја. Делиоце ћемо избројати петљом од 1 до  $\sqrt{n \cdot m}$ . Ово решење ради у сложености  $(n^2)$ .

Да би се решиле и остале групе тест примера, потребно је приметити следеће: *Број има непаран број делилаца ако и само ако је квадрат природног броја*.

---

Ово се може закључити на два начина.

*Први начин.* Ако природан број  $d$  дели  $n$ , тада и  $\frac{n}{d}$  дели  $n$ . Из тога можемо закључити да, уколико су за свако  $d$ , бројеви  $d$  и  $\frac{n}{d}$  различити, сваки број  $d$  има свог пару, па би то резултирало парним бројем делилаца. Бројеви су једнаки ако важи  $n = d^2$ , тј. ако је  $n$  квадрат природног броја (тада и даље сви делиоци осим  $\sqrt{n}$  имају свог пару).

*Други начин.* Посматрајмо канонску факторизацију броја  $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ . Сваки делилац броја  $n$  има облик  $d = p_1^{s_1} p_2^{s_2} \cdots p_k^{s_k}$ , где је  $0 \leq s_i \leq \alpha_i$  за свако  $i = 1, 2, \dots, k$ . Сваки од експонената  $s_i$  може се изабрати на  $\alpha_i + 1$  начина. То нам даје укупно  $(\alpha_1 + 1)(\alpha_2 + 1) \cdots (\alpha_k + 1)$  могућности за делилац  $d$ . То је уједно и број делилаца броја  $n$ . Приметимо да је број делилаца непаран ако и само ако је  $\alpha_i + 1$  непаран, односно  $\alpha_i$  паран за свако  $i = 1, 2, \dots, k$ . То управо значи да је  $n$  потпун квадрат.

Према томе, задатак смо свели на еквивалентан задатак - *да одредимо најмање  $m$  такво да је  $n \cdot m$  потпун квадрат*.

### Решење када $n \leq 10^6$

У овом случају је доволно испитати петљом све бројеве  $1, 2, 3, \dots, n$  као кандидате за  $m$  и проверити да ли је тренутни број помножен са  $n$  потпун квадрат. Ово решење ради у сложености  $\mathcal{O}(n)$ .

## Комплетно решење

Да бисмо задатак решили за 100 поена, потребно је да посматрамо канонску факторизацију броја  $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ . Циљ нам је да број  $n \cdot m$  буде потпун квадрат тј. да сви експоненти буду парни бројеви. Стога, број  $m$  ћемо потражити у облику  $m = p_1^{\beta_1} p_2^{\beta_2} \cdots p_k^{\beta_k}$  (тада  $\alpha_i + \beta_i$  треба да буде паран број за свако  $i = 1, 2, \dots, k$ ). Уколико је експонент (простог фактора  $p_i$ )  $\alpha_i$  паран, тада ће  $m$  бити најмањи за  $\beta_i = 0$ , а уколико је експонент (простог фактора  $p_i$ )  $\alpha_i$  непаран, тада ће  $m$  бити најмањи за  $\beta_i = 1$ .

Дакле,  $m$  се добија као производ простих бројева са непарним експонентом у факторизацији броја  $n$ . Ово решење ради у сложености факторизације броја тј.  $\mathcal{O}(\sqrt{n})$ .

## Задатак: Шибице

Аутор: Владан Ковачевић

Дато је  $n$  шибица (палидрвца) које не морају бити исте дужине. Испитати да ли се коришћењем свих датих шибица може формирати квадрат. Шибице не смеју да се ломе на мање делове, већ само да се надовезују.

### Опис улаза

Са стандардног улаза уноси се број  $n$  ( $1 \leq n \leq 16$ ), затим  $n$  бројева који представљају дужине шибица (дужине су природни бројеви мањи или једнаки 500).

### Опис излаза

На стандардни излаз исписати 1 ако је могуће формирати квадрат, у супротном исписати 0.

### Пример 1

Улаз	Излаз
6	1
3 2 4 1 2 4	

### Објашњење

Можемо формирати квадрат чија је дужина једне ивице 4, тако што ће прве две ивице да буду шибице дужине 4, трећа ивица ће да буде сачињена од 2 шибице дужине 2, док ће четврта ивица да буде сачињена од шибица дужина 1 и 3.

### Пример 2

Улаз

```
6
3 2 4 2 2 4
```

*Излаз*

```
0
```

### Пример 3

*Улаз*

```
6
2 4 6 8 10 12
```

*Излаз*

```
0
```

### Решење

Ако укупна дужина свих шибица није дељива са 4, од њих се не може направити квадрат. Ако јесте дељив са 4, задатак решавамо рекурзивном функцијом којом набрајамо све могуће распореде шибица на 4 странице троугла. Жељена дужина сваке странице је четвртина укупне дужине свих страница. Функција прима текућу шибицу и тренутне дужине страница квадрата (добијене распоређивањем свих претходних шибица). Покушавамо да текућу шибицу приједоримо свакој страници квадрата (то је могуће ако се додавањем текуће шибице и даље добија дужина странице која је мања или једнака од текуће дужине). Након приједоривања текуће шибице некој страници, рекурзивно покушавамо да распоредимо све остале шибице. Ако нема више шибица које треба да се распореде, успешно смо распоредили све шибице.

Приметимо да се програм може мало убрзати тако што прву шибицу увек доделимо само првој страници квадрата.

```
def moze_kvadrat_rek(sibice, i, stranica1, stranica2, stranica3, stranica4, zeljena_stranica):
    if i == len(sibice):
        return True
    if (sibice[i] + stranica1 <= zeljena_stranica and
        moze_kvadrat_rek(sibice, i+1, sibice[i] + stranica1, stranica2, stranica3, stranica4, zeljena_s
            return True
    if (sibice[i] + stranica2 <= zeljena_stranica and
        moze_kvadrat_rek(sibice, i+1, stranica1, sibice[i] + stranica2, stranica3, stranica4, zeljena_s
            return True
    if (sibice[i] + stranica3 <= zeljena_stranica and
        moze_kvadrat_rek(sibice, i+1, stranica1, stranica2, sibice[i] + stranica3, stranica4, zeljena_s
            return True
    if (sibice[i] + stranica4 <= zeljena_stranica and
        moze_kvadrat_rek(sibice, i+1, stranica1, stranica2, stranica3, sibice[i] + stranica4, zeljena_s
            return True
    return False

def moze_kvadrat(sibice):
    ukupna_duzina = sum(sibice)
    if ukupna_duzina % 4 != 0:
        return False
    return moze_kvadrat_rek(sibice, 1, sibice[0], 0, 0, 0, ukupna_duzina // 4)

n = int(input())
sibice = list(map(int, input().split()))
if moze_kvadrat(sibice):
    print(1)
else:
    print(0)
```

---

## Задатак: Биоскоп

Аутор: Оњен Тешић

У једном чудном биоскопу се налази  $n$  редова у којима **не мора** да се налази исти број седишта. У првом реду се налази  $a_1$  седишта, у другом реду се налази  $a_2$  седишта, …, у  $n$ -том реду се налази  $a_n$  седишта. Парови долазе на пројекцију филма, двоје људи који су пар треба да седе једно поред другог (у истом реду). Колико највише парова може да буде у биоскопу у исто време?

### Опис улаза

Прва линија стандардног улаза садржи природан број  $n$  који представља број редова у биоскопу (у биоскопу је највише 1000 редова). Наредних  $n$  линија стандардног улаза садрже по један природан број –  $i$ -ти члан низа  $a$ , који представља број седишта у  $i$ -том реду биоскопа.

### Опис излаза

На стандардни излаз исписати тражени број парова.

### Пример

Улаз	Излаз
5	14
7	
8	
4	
5	
6	

### Решење

Ако у реду са индексом  $i$  има паран број седишта, тада у тај ред може да седне  $\frac{a_i}{2}$  парова. Слично, ако у реду са индексом  $i$  има непаран број седишта, тада у тај ред може да седне  $\frac{a_i - 1}{2}$  парова. Приметимо да је у оба случаја тај број једнак количнику целобројно подељених бројева  $a_i$  и 2. Стога, одговор ће бити  $\left\lfloor \frac{a_1}{2} \right\rfloor + \left\lfloor \frac{a_2}{2} \right\rfloor + \left\lfloor \frac{a_3}{2} \right\rfloor + \dots + \left\lfloor \frac{a_n}{2} \right\rfloor$ .

```
n = int(input())
rezultat = 0

for i in range(n):
    broj_sedista = int(input())
    rezultat += broj_sedista // 2

print(rezultat)
```

## Задатак: Најбрже до циља

Аутор: Милан Вугделија

Тачке у матрици представљају празна поља, а слова T представљају капије система телепорта. Након стајања на капију, кретање може да се настави из исте или било које друге капије. За употребу телепорта не троши се никакво додатно време.

У матрици још постоји једно слово S (старт) и једно слово C (циљ). Дозвољени су кораци у сваком од 4 смера, тј. по једно поље горе, доле, лево или десно, ако се тиме не излази из матрице.

Написати програм који за дату матрицу одређује најмањи могућ број корака од старта до циља.

### Опис улаза

У првом реду стандардног улаза налазе се два цела позитивна броја  $R$  и  $K$ , мања или једнака 200, раздвојена једним размаком. Број  $R$  је број редова, а  $K$  број колона матрице.

У сваком од наредних  $R$  редова налази се по  $K$  знакова раздвојених по једним размаком. Знакови могу да буду ., T, S или C, с тим да се појављује укупно тачно једно S и тачно једно C.

**Опис излаза**

На стандардни излаз исписати један неозначен цео број, најмањи могућ број корака од старта до циља.

**Пример 1**

Улаз	Излаз
4 7	4
. . . . T . .	
. S . . . . .	
. . . . . . C	
T . . T . . T	

*Објашњење*

Један низ од 4 корака који доводе од старта до циља су: доле, доле, лево, горе.

**Пример 2**

Улаз

```
3 8
. . . . T T .
. S . . . . .
T . . C . . . T
```

Излаз

```
3
```

*Објашњење*

Један низ од 3 корака који доводе од старта до циља су: десно, десно, доле.

**Решење**

Након (или током) учитавања матрице, могу у једном пролазу кроз њу да се одреде координате старта  $S_i, S_j$  и циља  $C_i, C_j$ .

У другом пролазу кроз матрицу, за свако слово T може да се одреди растојање (број потребних корака) од старта до тог слова, као и од тог слова до циља. Успут, у истом пролазу, може да се израчуна минимум свих растојања од старта до неког слова T, означен са  $ST_{min}$ , као и минимум свих растојања од неког слова T до циља, означен са  $TC_{min}$ .

Дужина најкраћег пута од старта до циља користећи систем телепортаже је  $ST_{min} + TC_{min}$ . Дужина најкраћег пута уопште је или ова, или број корака од старта до циља, па се коначан одговор добија као мања од ове две вредности, тј. као  $\min(ST_{min} + TC_{min}, |S_i - C_i| + |S_j - C_j|)$

```
r, k = [int(x) for x in input().split()]
a = [input().split() for j in range(r)]

for i in range(r):
    for j in range(k):
        if a[i][j] == 'S':
            i_start = i
            j_start = j
        elif a[i][j] == 'C':
            i_cilj = i
            j_cilj = j

min_st, min_tc = r+k, r+k

for i in range(r):
    for j in range(k):
        if a[i][j] == 'T':
            min_st = min(min_st, abs(i_start - i) + abs(j_start - j))
            min_tc = min(min_tc, abs(i_cilj - i) + abs(j_cilj - j))
```

---

```
dist_sc = abs(i_start - i_cilj) + abs(j_start - j_cilj)
print(min(dist_sc, min_st + min_tc))
```

## Задатак: Најмање промена

Аутор: Иван Дрецун

Најпознатији анонимни хакер Душан се запрепастио када је прочитao рачун за Интернет. Душан је својим невероватним хакерским способностима (комшија Мића му ради у Интернету) сазнао да ако своју потрошњу интернета представи као низ нула и јединица, Интернет ће му наплатити по један динар за сваку промену са 0 на 1 или обрнуто, читајући низ са лева на десно. На пример, ако му је потрошња интернета 000110, рачун за тај месец ће бити 2 динара.

Одлучио је да следећег месеца хакује самог себе у нади да ће му стићи мањи рачун. Пошто хаковање није лако, он може да одабере само један број  $k$  и затим промени (са 0 на 1 или обрнуто) сваки  $k$ -ти елемент низа. На пример, ако је низ 000110 и он одабере број 2, након измене низ постаје 010011. Напиши програм који за Душанову потрошњу следећег месеца одређује колики је најмањи могући рачун који ће му стићи након хаковања.

### Опис улаза

Са стандарданог улаза се уноси један стринг нула и јединица који представља потрошњу за наредни месец. Дужина стринга је највише 100000.

### Опис излаза

На стандардни излаз исписати два природна броја одвојена размаком. Први број представља најбоље  $k$  (између 1 и  $n$ ) које Душан може да одабере, а други број представља рачун за такво  $k$ . Ако постоји више таквих вредности за  $k$ , исписати најмању.

### Пример 1

Улаз	Излаз
0010111	4 1

Објашњење

Рачуни за  $k = 1, k = 2, \dots$  редом су 1101000, 0111101, 0000101, 0011111, 0010011, 0010101 и 0010110.

### Пример 2

Улаз

00101

Излаз

2 1

Објашњење

Рачуни за  $k = 1, k = 2, \dots$  редом су 11010, 01111, 00001, 00111 и 00100.

### Решење

#### Претрага по $k$

Задатак је могуће решити испробавањем сваке дозвољене вредности броја  $k$  и одређивањем износа рачуна. Приликом претраге памтимо најбоље решење.

Сложеност оваквог решења је  $O(n^2)$  зато што за сваку вредност броја  $k$  морамо одредити вредност рачуна проласком кроз цео низ.

```
def flip(c):
    return 1 - c

def diff(s):
```

```

d = 0
for i in range(1, len(s)):
    d += s[i] != s[i - 1]
return d

s = [int(x) for x in input()]
n = len(s)

mind = diff(s)
mink = 1
for k in range(1, n + 1):
    for i in range(k - 1, n, k):
        s[i] = flip(s[i])

    d = diff(s)
    if d < mind:
        mind = d
        mink = k

for i in range(k - 1, n, k):
    s[i] = flip(s[i])

print(mink, mind)

```

## Ефикасно решење

Ефикасније решење је могуће постићи ако запазимо да приликом одређивања вредности рачуна није потребно пролазити кроз цео низ. Довољно је једном, пре почетка претраге, израчунати укупан рачун. Затим, када желимо да одредимо вредност рачуна за неко  $k$ ,овољно је да израчунамо за колико ће се број разлика повећати или смањити приликом обртања свих  $k$ -тих елемената. Што је вредност броја  $k$  већа, то ће бити све мање елемена на којима се прави измена. Приметимо да већ на половини претраге, када је  $k > \frac{n}{2}$ , обрћемо само по један елемент низа.

Прецизније, сложеност овог алгоритма је  $O(n \log n)$ . У првом кораку обрћемо свих  $n$  елемената, у другом кораку  $\frac{n}{2}$  елемената, у трећем  $\frac{n}{3}$ , итд. Укупан број корака ће бити мањи од  $n + \frac{n}{2} + \frac{n}{3} + \dots + \frac{n}{n-1} + \frac{n}{n} = n(1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}) < n(1 + \frac{1}{2} + \frac{1}{3} + \dots) = O(n \log n)$ . Образложење последње једнакости може се наћи на Википедији (потражите чланак [Harmonic series \(mathematics\)](#)).

```

s = input()
n = len(s)

diff = 0
for i in range(1, n):
    diff += s[i] != s[i - 1]

diff_min = diff
diff_k = 1

for k in range(2, n + 1):
    flip_diff = diff
    for i in range(k - 1, n, k):
        if s[i] == s[i - 1]:
            flip_diff += 1
        else:
            flip_diff -= 1

    if i < n - 1:
        if s[i + 1] == s[i]:
            flip_diff += 1

    if flip_diff < diff_min:
        diff_min = flip_diff
        diff_k = k

print(diff_k, diff_min)

```

---

```

else:
    flip_diff -= 1

if flip_diff < diff_min:
    diff_min = flip_diff
    diff_k = k

print(diff_k, diff_min)

```

## Задатак: Амљез

*Аутор: Душан Пойагић*

На чаробној планети Аљmez објекти могу да имају позитивну и негативну масу. На тој планети се налази велики координатни систем на који је Анави поставила лагане лопте дуж у-осе. Све лопте које је поставила имају апсолутну вредност масе између 0,1 и 0,15 гк-а (гк је мера за масу на Аљmezу), с тим што је Анави водила рачуна да буде приближно једнак број лопти са позитивном и негативном масом (ти бројеви се разликују за највише 1). Анавин млађи брат Пилиф тренира лабдуф (спорт у коме је дозвољено само трчање уназад и додиривање лопте ногама) и жели да вежба свој шут. Он прилази свакој лопти и шутира је истом јачином од 126 А (А је мерна јединица за јачину шута лопте) у правцу одређеном x-осом (дакле не мења се у-координата лопте). Лопта лети и пада на место чија се x-координата добија када се јачина шута (126) подели масом лопте. На пример ако имамо лопту чија је у-координата 20,3 и маса -0,15, она ће слетети на место са координатама (-840, 20,3) јер је  $126/-0,15 = -840$ . Када је Анави видела шта је Пилиф у радио, прво се јако најутила, а онда му рекла да мора да покупи све лопте. Пошто Пилифа наравно mrзи да то уради, замолио је да му помогне тако што ћеш одредити којим редом да купи лопте тако да укупна дистанца коју пређе буде најкраћа могућа. Занемари дистанцу коју Пилиф треба да пређе од своје тренутне позиције до прве лопте са твог списка. У случају више тачних решења исписати било које.

*Помоћ:* Ако имамо две лопте на координатама  $(x_1, y_1)$  и  $(x_2, y_2)$ , растојање између њих се рачуна као  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ .

### Опис улаза

У првом реду стандардног улаза се налази  $n$  ( $3 \leq n \leq 12$ ), број лопти које је Анави поставила. У наредних  $n$  редова се налазе по два броја раздвојена размаком:  $y$ -координата лопте ( $-50 \leq y_i \leq 50$ ) и маса лопте ( $0.1 \leq |m| \leq 0.15$ ).

*Ограничења:*

- У тест примерима вредним 20 поена важи  $y_i = 0$ .
- У тест примерима вредним 50 поена додатно важи  $n \leq 8$ .

### Опис излаза

У једином реду стандардног излаза исписати  $n$  различитих бројева одвоједних размаком – тражени редослед лопти. На пример, ако Пилиф треба да оде прво по 3. лопту која је унета на улазу, па по прву, па по другу исписати 3 1 2.

### Пример

Улаз	Излаз
6	1 2 3 6 4 5
5 -0.1	
5 -0.15	
0 -0.15	
0 0.1	
-5 0.1	
0 0.15	

*Објашњење*

Са слике се види да је тражени распоред оптималан. Алтернативно решење је 5 4 6 3 2 1.

### Решење

Проблем пред нама је добро познат проблем путујућег трговца у ком је потребно одредити најкраће растојање при обиласку свих тачака у равни.

### Решење сортирањем по $x$ оси

Уколико наивно приступимо решавању овог проблема можемо помислiti да је довољно да обилазимо тачке тако што кренемо од најлевље и онда идемо редом по  $x$  оси. Ово решење, наравно, није тачно у општем случају и доноси само 10 поена.

#### Сложеност решења

Сложеност овог решења је једнака сложености сортирања низа, односно  $n \cdot \log n$

```
import math

lopte = []
n = int(input())
for i in range(n):
    y, m = map(float, input().split())
    x = 126 / m
    lopte.append([x, y, i + 1])

lopte.sort(key=lambda x:x[0])

for lopta in lopte:
    print(lopta[2], end=" ")
```

### Решење генерирањем свих распореда лопти

Класично решење за овај проблем је генерирање свих распореда (пермутација) тачака, одређивање укупног растојања за сваки од тих распореда и онда одређивање распореда који има најмање растојање. Пермутације генеришемо користећи рекурзивну методу *permutacije*. Растојање између две тачке у равни се одређује коришћењем Питагорине теореме. Ово решење доноси 50 поена.

#### Сложеност решења

Генерирање свих распореда има сложеност која је једнака броју распореда, дакле  $O(n!)$ . Пошто је за сваки распоред додатно потребно проћи кроз њега и одредити укупно растојање које се пређе ходајући од лопте до лопте, то додаје додатни фактор на сложеност и укупна сложеност је  $O(n \cdot n!)$ . Ово решење је могуће додатно побољшати тако што ће се при генерирању пермутација успут рачунати и укупно растојање и онда би укупна сложеност била  $(n!)^2$ .

```
import math

# funkcija za određivanje rastojanja izmedju dve tacke u ravni
# Pitagorinom teoremom
def rastojanje(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

n = int(input())
ys = []
xs = []

for _ in range(n):
    y, m = map(float, input().split())
    xs.append(126 / m)
    ys.append(y)

# najbolja permutacija do sada
najPerm = []
# najmanje rastojanje do sada
```

---

```

najR = float('inf')

# lista perm predstavlja jednu permutaciju inicijalno ucitanih lopti.
# U njemu se ne nalaze lopte vec indeksi iz originalnog niza
perm = list(range(n))

def permutacije(i, n):
    global najPerm, najR, perm
    if i == n:
        # Ako smo stigli do kraja odredujemo ukupno rastojanje za
        # generisani permutaciju i pamtimo je ako je najmanja do sada.
        p = list(perm)
        r = 0
        for k in range(n):
            r += rastojanje(xs[p[k]], ys[p[k]], xs[p[k-1]], ys[p[k-1]]) if k > 0 else 0
        if r <= najR:
            najR = r
            najPerm = p
    else:
        for j in range(i, n):
            perm[i], perm[j] = perm[j], perm[i]
            permutacije(i + 1, n)
            perm[i], perm[j] = perm[j], perm[i]

permutacije(0, n)

for i in range(n):
    print(najPerm[i] + 1, end=" ")

```

## Решење раздвајањем на две групе

Можемо приметити да се све лопте групишу у два правоугаоника: један који је лево од  $y$  осе и један који је десно. Та два правоугаоника су међусобно доста удаљена: Ако имамо најтеже могуће лопте различитих знакова (-0,15 и 0,15) које имају исту  $y$  координату, оне ће обе пасти на удаљеност од  $126/0.15 = 840$  од  $y$  осе, односно биће на међусобном растојању од тачно  $840 + 840 = 1680$ . Са друге стране, две лопте које су са исте стране  $y$  осе ће бити међусобно најудаљеније ако падну у два темена квадрата која су по дијагонали, то растојање биће  $\sqrt{(420^2 + 100^2)} = 408$  што је значајно мање од до било које лопте која је са супротне стране  $y$  осе. Одавде је интуитивно јасно да се оптимална путања добија тако што се прво обиђу тачке са једне, а затим са друге стране  $y$  осе, тј. Пилиф само једном треба да пређе преко  $y$  осе. Ово се може и лако доказати тако што се сваки наредни (осим првог) прелазак преко  $y$  осе може заменити увећењем додатне (краће) везе између две лопте са исте стране и променом почетне лопте. Дакле решење је одредити све распореде са једне и са друге стране  $y$  осе којих има по  $7! = 5040$  и онда пробати комбинацију сваке леве са сваком десном и тако одредити укупно најкраћи пут.

## Сложеност решења

Генерирање свих распореда има сложеност која је једнака броју распореда, дакле  $O((n/2)!)$  са сваке стране. Затим је потребно пробати сваку леву са сваком десном, па се добија коначна сложеност од  $((n/2)! \cdot (n/2)!)$ . Поређења ради за  $n = 14$ ,  $n! 87 \cdot 10^9$ , а  $(n/2)! \cdot (n/2)! 25 \cdot 10^6$ . Ово решење доноси максималан број поена.

```

import math
from itertools import permutations

ysl = []
xsl = []
il = []
nl = 0

yld = []

```

```

xsd = []
id = []
nd = 0

perm1 = []
rast1 = []
n1 = 0

perm2 = []
rast2 = []
n2 = 0

def rastojanje(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1))

def rastojanje1(i, j):
    return rastojanje(xsl[i], ysl[i], xsd[j], ysd[j])

def printVektor(v, n):
    for i in range(n):
        print(v[i], end=" ")
    print()

def printVektorD(v, n):
    for i in range(n):
        print(v[i], end=" ")
    print()

def permutacije(i, n, perm, m):
    global n1, n2
    if i == n:
        p = [0] * n
        r = 0
        if m < 0:
            for k in range(n):
                p[k] = perm[k]
                if k > 0:
                    r += rastojanje(xsl[p[k]], ysl[p[k]], xsl[p[k - 1]], ysl[p[k - 1]])
            perm1.append(p)
            rast1.append(r)
            n1 += 1
        else:
            for k in range(n):
                p[k] = perm[k]
                if k > 0:
                    r += rastojanje(xsd[p[k]], ysd[p[k]], xsd[p[k - 1]], ysd[p[k - 1]])
            perm2.append(p)
            rast2.append(r)
            n2 += 1
    else:
        for j in range(i, n):
            p = perm[i]
            perm[i] = perm[j]

```

---

```

perm[j] = p
permutacije(i + 1, n, perm, m)
p = perm[i]
perm[i] = perm[j]
perm[j] = p

if __name__ == "__main__":
    n = int(input())

    for i in range(n):
        y, m = map(float, input().split())
        if m < 0:
            xsl.append(126 / m)
            ysl.append(y)
            il.append(i + 1)
            nl += 1
        else:
            xsd.append(126 / m)
            ysd.append(y)
            id.append(i + 1)
            nd += 1

    perm = list(range(max(nl, nd)))
    for i in range(nl):
        perm[i] = i
    permutacije(0, nl, perm, -1)
    for i in range(nd):
        perm[i] = i
    permutacije(0, nd, perm, 1)

    min_distance = 1000000000
    im, jm = 0, 0

    for i in range(n1):
        for j in range(n2):
            r = rast1[i] + rast2[j] + rastojanje1(perm1[i][nl - 1], perm2[j][0])
            if r < min_distance:
                min_distance = r
                im, jm = i, j

    for i in range(nl):
        print(il[perm1[im][i]], end=" ")
    for i in range(nd):
        print(id[perm2[jm][i]], end=" ")

```

Максималан број поена је могуће добити и коришћењем принципа динамичког програмирања за решавање пороблема путујућег трговца, али познавање тог решења се не очекује од ученика на овом нивоу такмичења.

## Глава 3

# Општинско такмичење

### Задатак: Дани у недељи

Аутор: Ођиен Тешић

На једној далекој планети, једна недеља траје  $n$  дана и дани су нумерисани бројевима 1, 2, 3, ...,  $n$ . Дане бројимо од првог дана прве недеље. Познато је да је данас дан са укупним редним бројем  $a$ .

Напиши програм који одређује који ће дан у недељи по реду бити за  $x$  дана (та недеља не мора бити прва недеља).

#### Опис улаза

Прве три линије стандардног улаза садрже по један природан број. То су редом бројеви  $n$ ,  $a$  и  $x$ , такви да је  $1 \leq n \leq 1000$ ,  $1 \leq a \leq 100000$ ,  $1 \leq x \leq 100000$ .

#### Опис излаза

На стандардни излаз исписати тражени редни број дана.

#### Пример 1

Улаз	Излаз
15	5
2	
18	

Објашњење

Данас је 2. дан, а за 18 дана ће бити 20. дан. Дани у другој недељи су 16, 17, 18, 19, 20, 21, ..., 29. и 30. Према томе, то ће бити пети дан те недеље.

#### Пример 2

Улаз
7
3
20

Излаз

2

Објашњење

Данас је 3. дан, а за 20 дана ће бити 23. дан. Дани у четвртој недељи су 22, 23, 24, 25, 26, 27. и 28. Према томе, то ће бити други дан те недеље.

#### Решење

---

Приметимо, дани прве недеље су нумерисани бројевима од 1 до  $n$ , дани друге недеље бројевима од  $n + 1$  до  $2n$ , дани треће недеље бројевима од  $2n + 1$  до  $3n$  итд. Једно решење је да израчунамо којој недељи припада дан са редним бројем  $a + x$ , па да на основу тога одредимо који је то дан у тој недељи. Други начин је да приметимо да је ово управо остатак при дељењу броја  $a + x$  са  $n$ . Једини случај када треба бити пажљив је када је  $a + x$  делјив са  $n$ . Тада је остатак при дељењу једнак 0, али је одговор  $n$ .

```
n = int(input())
a = int(input())
x = int(input())
redniBrojDana = (a + x) % n
if redniBrojDana == 0:
    print(n)
else:
    print(redniBrojDana)
```

## Задатак: Топлотни појас

Аутор: Филип Марић

Планета Земља се дели на следећих 5 топлотних појасева:

- *жарки појас* се простире између северног и јужног повратника
- *северни умерени појас* се простире између северног повратника и северног поларника
- *северни хладни појас* се простире изнад северног поларника
- *јужни умерени појас* се простире између јужног повратника и јужног поларника
- *јужни хладни појас* се простире испод јужног поларника

Ако се зна да се повратници налазе на  $23^{\circ}30'$  северне односно јужне географске ширине, а поларници на  $66^{\circ}30'$  северне односно јужне географске ширине, напиши програм који на основу унете географске ширине неког града одређује топлотни појас коме припада.

### Опис улаза

Са стандарданог улаза се уноси географска ширина. Уноси се прво број степени, затим број минута и онда слово **s** односно **j** које означава да ли је у питању северна или јужна географска ширина. Претпоставити да унети град није ни на једном повратнику ни на једном поларнику.

### Опис излаза

На стандардни излаз исписати ознаку **z** (жарки), **su** (северни умерени), **sh** (северни хладни), **ju** (јужни умерени) или **jh** (јужни хладни), у зависности од појаса коме унета географска ширина припада.

### Пример 1

Улаз	Излаз
15	<b>z</b>
28	
<b>j</b>	

*Објашњење*

Пошто се град налази на јужној полулопти, али изнад јужног повратника, у питању је жарки појас.

### Пример 2

Улаз

40  
15  
**s**

Излаз

**su**

*Објашњење*

Град се налази на северној полулопти, између северног повратника и северног поларника, па се налази у северном умереном појасу.

### Пример 3

*Улаз*

```
66
42
j
```

*Излаз*

```
jh
```

*Објашњење*

Град се налази на јужној полулопти, испод јужног поларника, па се налази у јужном хладном појасу.

### Решење

Најједноставнији начин да поредимо углове задате у степенима и минутима је да их пре поређења претворимо у minute. Угао који има  $s$  степени и  $m$  минута има укупно  $60s + m$  минута. Након тога се до решења долази анализом свих могућих случајева.

```
def uMinute(s, m):
    return s * 60 + m

steponi = int(input())
minuti = int(input())
sj = input()

if uMinute(steponi, minuti) < uMinute(23, 30):
    print("z")
elif uMinute(steponi, minuti) < uMinute(66, 30):
    print(sj + "u")
else:
    print(sj + "h")
```

## Задатак: Робот достављач

*Аутор:* Милан Вујделић

У једној фирмци која се бави онлајн продајом, за доставу купљене робе користе се роботи. Купци које ће Робот Миле сутра послуживати станују сви на истој страни исте улице. Миле за сваку адресу зна њену координату, тј. растојање од почетка улице до те зграде. Миле је програмиран тако да адресе обилази редом којим су му задате.

Напиши програм који за дати број достава и дате координате зграда (растојања од зграда од почетка улице) израчунава и исписује укупно растојање које ће робот Миле прећи од прве до последње доставе.

### Опис улаза

У првом реду стандардног улаза налази се природан број  $n$ , не већи од 30. У сваком од следећих  $n$  редова налази се по један неозначен цео број, растојање од почетка улице до следеће зграде (тј. координата зграде) у коју треба доставити наруџбину. Ови бројеви нису већи од 10000.

### Опис излаза

На стандардни излаз исписати само један цео број, укупну дужину пута који пређе робот Миле.

### Пример 1

---

Улаз	Излаз
5	13
2	
7	
7	
1	
3	

#### Објашњење

Позиције зграда у редоследу достављања су 2, 7, 7, 1, 3. Миле прво иде од позиције 2 до позиције 7, затим од 7 до 7 (достава у истој згради), па од 7 до 1 и на крају од 1 до 3. Растојања пређена од доставе до доставе су редом 5, 0, 6, 2, а збир тих растојања је 13.

#### Пример 2

Улаз

1
7

Излаз

0
---

#### Објашњење

Прва и последња достава су иста достава. То значи да се Миле уопште не креће између прве и последње доставе и да је пређено растојање од прве до последње доставе једнако 0.

#### Решење

Зграде можемо да посматрамо као тачке на бројевној полуправој. Координате зграда су дати цели бројеви. Растојање између две зграде једнако је апсолутној вредности разлике њихових координата.

За датих  $n$  бројева треба да се израчуна збир растојања између узастопних зграда, а то је збир апсолутних вредности разлика суседних бројева.

Да би решење било једноставније, након читања броја  $n$  може и прва од датих  $n$  координата да се учита пре петље, а да се кроз петљу прође само  $n - 1$  пута (а не  $n$  пута).

У петљи се учитава координата следеће зграде, растојање од претходне зграде до ње се додаје на збир и на крају се нова зграда запамти као претходна да би се рачунање исправно наставило у следећем пролазу кроз петљу.

На крају само треба да се испише израчунати збир.

```
n = int(input())
gde_sam_bio = int(input())
predjeni_put = 0
for i in range(1, n):
    gde_sam_sad = int(input())
    predjeni_put = predjeni_put + abs(gde_sam_sad - gde_sam_bio)
    gde_sam_bio = gde_sam_sad

print(predjeni_put)
```

## Задатак: Минимакс

Аутор: Владимира Кузмановић

Наставник је Перици задао домаћи задатак у којем на основу 4 задате декадне цифре треба да одреди разлику највећег и најмањег четвороцифреног броја који се могу формирати помоћу тих цифара. Написати програм који ће на основу 4 унете декадне цифре помоћи Перици да провери да ли је тачно урадио домаћи.

#### Опис улаза

У сваком од 4 реда стандардног улаза налази се по једна декадна цифра, од којих бар једна није нула.

### Опис излаза

На стандардни излаз исписати само један природан број који представља разлику највећег и најмањег четвороцифрених броја који се помоћу датих цифара могу формирати.

### Пример 1

Улаз	Излаз
3	6174
7	
5	
9	

#### Објашњење

Највећи четвороцифрени број који се може формирати помоћу датих цифара је 9753, а најмањи је 3579. На стандардни излаз треба исписати њихову разлику  $9753 - 3579 = 6174$ .

### Пример 2

#### Улаз

2  
0  
5  
6

#### Излаз

4464

#### Објашњење

Највећи четвороцифрени број који се може формирати помоћу датих цифара је 6520, а најмањи је 2056. На стандардни излаз треба исписати њихову разлику  $6520 - 2056 = 4464$ .

**Напомена:** У задатку се тражи најмањи четвороцифрени број, па треба водити рачуна да број не сме починјати нулом. Управо због тога, најмањи четвороцифрени број је 2056.

### Пример 3

#### Улаз

0  
0  
5  
0

#### Излаз

0

#### Објашњење

Највећи четвороцифрени број који се може формирати помоћу датих цифара је 5000, а најмањи је такође 5000. На стандардни излаз треба исписати њихову разлику  $5000 - 5000 = 0$ .

**Напомена:** У задатку се тражи најмањи четвороцифрени број, па треба водити рачуна да број не сме починјати нулом. Управо због тога, најмањи четвороцифрени број је 5000, што ће уједно бити и највећи четвороцифрени број.

### Решење

Цифре можемо сортирати употребом библиотечке функције. Због водећих нула потребно је обратити пажњу на формирање најмањег броја.

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

---

```
d = int(input())
m1, m2, m3, m4 = sorted((a, b, c, d))

max_num = m4*1000 + m3*100 + m2*10 + m1
if m1 != 0:
    min_num = m1*1000 + m2*100 + m3*10 + m4
elif m2 != 0:
    min_num = m2*1000 + m3*100 + m4
elif m3 != 0:
    min_num = m3*1000 + m4
else:
    min_num = m4*1000

print(max_num - min_num)
```

Исти ефекат се може постићи и на мало другачији начин.

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

m1, m2, m3, m4 = sorted((a, b, c, d))
max_num = m4*1000 + m3*100 + m2*10 + m1

if m1 == 0:
    m1, m2 = m2, m1
if m1 == 0:
    m1, m3 = m3, m1
if m1 == 0:
    m1, m4 = m4, m1

min_num = m1*1000 + m2*100 + m3*10 + m4
print(max_num - min_num)
```

Уместо библиотечке функције, могуће је извршити сортирање и ручно.

```
x = int(input())
m4 = x
```

```
x = int(input())
if (x > m4):
    m3 = m4
    m4 = x
else:
    m3 = x

x = int(input())
if (x > m4):
    m2 = m3
    m3 = m4
    m4 = x
elif (x > m3):
    m2 = m3
    m3 = x
else:
    m2 = x

x = int(input())
if (x > m4):
```

```

m1 = m2
m2 = m3
m3 = m4
m4 = x
elif (x > m3):
    m1 = m2
    m2 = m3
    m3 = x
elif (x > m2):
    m1 = m2
    m2 = x
else:
    m1 = x

maxNum = m4*1000 + m3*100 + m2*10 + m1
minNum = 0
if (m1 != 0):
    minNum = m1*1000 + m2*100 + m3*10 + m4
elif (m2 != 0):
    minNum = m2*1000 + m3*10 + m4
elif (m3 != 0):
    minNum = m3*1000 + m4
else:
    minNum = m4*1000

print(maxNum - minNum)

```

## Задатак: Обим паралелограма

*Автор: Филип Марић*

Ако је познат обим паралелограма  $O$  и то да је једна страница за  $d$  дужа од друге, напиши програм који одређује дужине страница овог паралелограма.

### Опис улаза

Са стандардног улаза се уноси цео број  $O$  (између 1 и 200), а затим цео број  $d$  (између 1 и 100). Бројеви  $O$  и  $d$  су у сваком тесту такви да су тражене дужине страница цели бројеви.

### Опис излаза

На стандардни излаз исписати у истом реду два цела броја раздвојена једним размаком. Ти бројеви треба да буду редом целобројне дужине страница  $a$  и  $b$ , тако да је  $a \leq b$ .

### Пример 1

Улаз	<i>Излаз</i>
46	10 13
3	

*Објашњење*

Страница  $b = 13$  је заиста за 3 дужа од странице  $a = 10$ , а обим је  $2(a + b) = 2(10 + 13) = 2 \cdot 23 = 46$ .

### Пример 2

*Улаз*

102  
1

*Излаз*

25 26

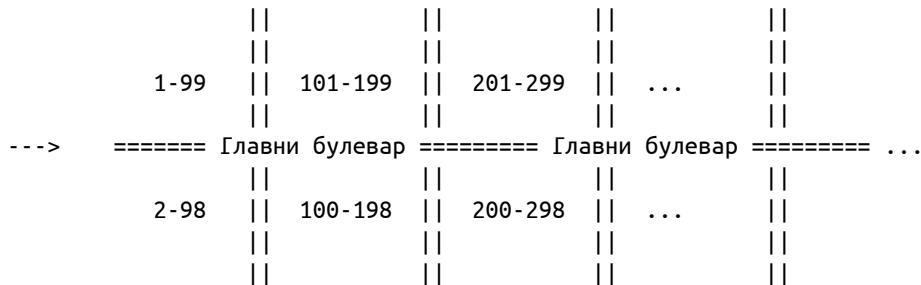
## Решење

Ако је краћа страница  $a$ , дужа  $b = a + d$ , важи да је  $O = 2(a + b) = 2(a + a + d) = 4a + 2d$ . Зато је  $a = (O - 2d)/4$ .

```
O = int(input())
d = int(input())
a = (O - 2*d) // 4
b = a + d
print(a, b)
```

## Задатак: Адресе

Са обе стране Главног булевара налазе се блокови зграда. Сваки блок је са једне стране булевара. Зграде са парним бројевима налазе се са десне, а зграде са непарним бројевима са леве стране булевара. Бројеви зграда које се налазе у истом блоку су исте парности и почињу истом цифром стотина (претходне цифре су им такође једнаке, ако постоје). Бројеви зграда у узастопним блоковима припадају узастопним стотинама (изузетак су бројеви дељиви са 100, види интервале у објашњењима).



Ако је потребно стићи са једне дате адресе на другу адресу у Главном булевару, одредити да ли се циљна адреса у односу на полазну налази у истој згради, у истом блоку, блоку преко пута, претходном блоку, следећем блоку, претходном блоку преко пута, следећем блоку преко пута, или неком другом блоку.

### Опис улаза

У сваком од два реда стандардног улаза налази се по један један природан број, не већи од 100000. Први број је број зграде из које се полази, а други је број циљне зграде.

### Опис излаза

На стандардни излаз исписати један од следећих текстова: *ista zgrada*, *isti blok*, *sledeci blok*, *prethodni blok*, *preko puta*, *sledeci blok preko puta*, *prethodni blok preko puta* или *daleko*.

*Савет:* да би се избегле грешке у куцању, најбоље је да се ови текстови копирају у програм.

### Пример 1

Улаз	Излаз
123	<i>isti blok</i>
147	

*Објашњење*

Оба броја су непарна, па су зграде са исте стране булевара. При томе оба броја припадају интервалу [100, 199], па се зграде налазе у истом блоку.

### Пример 2

Улаз

158  
249

Излаз

*sledeci blok preko puta*

*Објашњење*

Бројеви су различите парности, па се зграде налазе са различитих страна булевара, тј. циљна зграда је у неком блоку преко пута. Пошто број полазне зграде припада интервалу [100, 199], а број циљне зграде интервалу [200, 299], циљна зграда је у следећем блоку преко пута.

**Решење**

Циљна зграда може да буде “далеко” у односу на претходну било да је на истој или супротној страни булевара, неколико блокова пре или после блока у коме је полазна зграда. Да не бисмо морали да о сваком од тих случајева посебно водимо рачуна, најбоље је да одговор *daleko* буде подразумевани одговор, односно почетна вредност променљиве у којој држимо одговор. То је вредност која ће остати у променљивој ако не буде испуњен ниједан од услова под којима бисмо циљну зграду описали другачије.

Услови које треба да испитамо могу да се организују хијерархијски:

- ако су бројеви  $a$  и  $b$  једнаки, то је иста зграда
- иначе, ако су  $a$  и  $b$  исте парности (ако су зграде са исте стране булевара)
  - ако им је исти количник при дељењу са 100, зграда  $b$  је у истом блоку
  - иначе, ако је количник зграде  $b$  већи за један, зграда  $b$  је у следећем блоку
  - иначе, ако је количник зграде  $b$  мањи за један, зграда  $b$  је у претходном блоку
- иначе (бројеви различите парности, зграде са различитих страна булевара)
  - ако им је исти количник при дељењу са 100, зграда  $b$  је преко пута
  - иначе, ако је количник зграде  $b$  већи за један, зграда  $b$  је у следећем блоку преко пута
  - иначе, ако је количник зграде  $b$  мањи за један, зграда  $b$  је у претходном блоку преко пута

У свим осталим случајевима, зграда је далеко.

```
a = int(input())
b = int(input())

ista_strana = (a%2 == b%2)
a_stotina = a//100
b_stotina = b//100
gde = 'daleko'

if a==b:
    gde = 'ista zgrada'
elif ista_strana:
    if a_stotina == b_stotina:
        gde = 'isti blok'
    elif b_stotina == a_stotina + 1:
        gde = 'sledeci blok'
    elif b_stotina == a_stotina - 1:
        gde = 'prethodni blok'
else:
    if a_stotina == b_stotina:
        gde = 'preko puta'
    elif b_stotina == a_stotina + 1:
        gde = 'sledeci blok preko puta'
    elif b_stotina == a_stotina - 1:
        gde = 'prethodni blok preko puta'

print(gde)
```

**Задатак: Слепо куцање речи**

*Аутор: Филип Марин*

Наставница информатике учи децу слепо куцање. За зада су обрадили само слова у основном реду (a, s, d, f, g, h, j, k, l). Напиши програм који помаже наставници да са списка речи одабере оне речи, које може да уврсти у вежбање тј. да одабере речи које се састоје само од слова из основног реда.

## Опис улаза

Са стандардног улаза се учитава број  $n$  ( $1 \leq n \leq 100$ ), а затим  $n$  речи, од којих свака садржи највише 10 малих слова енглеске абеџеде (свака реч је у посебном реду).

## Опис излаза

На стандардни излаз у једном реду исписати укупан број речи које се састоје од слова из основног реда, као и њихов укупан број слова (то је број слова које ћаци треба да откуцају у целом вежбању). Бројеве раздвојити једним размаком.

### Пример 1

Улаз            Излаз  
4                2 9

alfa  
lisa  
kajak  
beta

*Објашњење*

Речи које се састоје само од слова из основног реда су **alfa** и **kajak** и њихов укупан број слова је  $4+5=9$ .

### Пример 2

Улаз

3  
grad  
gama  
delta

Излаз

0 0

*Објашњење*

Ниједна од ових речи не може да се уврсти у вежбање, па је и број речи и број слова у њима једнак 0.

## Решење

Проверу да ли су сва слова речи из основног реда можемо извршити класичним алгоритмом линеарне претраге и имплементирати је у помоћној функцији. Анализирамо један по један карактер дате речи и ако нађемо на карактер који није у првом реду враћамо вредност нетачно. Ако прођемо све карактере речи и при том не нађемо карактер ван основног реда, враћамо тачно.

У главном програму анализирамо једну по једну реч. За све оне за које помоћном функцијом уврдимо да се састоје искључиво из карактера основног реда увећавамо бројач речи за један и бројач слова за дужину те речи.

```
def OK_rec(rec):
    for c in rec:
        if c not in "asdfghjkl":
            return False
    return True
```

```
n = int(input())
broj_reci = 0
broj_slova = 0
for i in range(n):
    rec = input()
    if OK_rec(rec):
        broj_reci += 1
        broj_slova += len(rec)
```

```
print(broj_reci, broj_slova)
```

## Задатак: Одбојкашки резултат

*Аутар: Душан Пойадић*

Напиши програм који на основу списка добијених поена од почетка одбојкашког меча одређује тренутни резултат.

Одбојкашки меч се игра док један тим не освоји три сета. Сет осваја први тим који освоји 25 или више поена, са разликом најмање 2. Дакле, тим може да освоји сет резултатом  $25 : 0$ ,  $25 : 1$ , ... или резултатом  $25 : 23$ . Резултат  $25 : 24$  не означава крај сета јер разлика у поенима није барем 2. У том случају се игра наставља док један од тимова не направи разлику у поенима од најмање 2 (дакле  $26 : 24$ ,  $27 : 25$ ,  $28 : 26$  ...). Уколико дође до  $2 : 2$  у сетовима, последњи сет се не игра до 25 него до 15 уз остала правила која остају иста (мора да се направи најмање 2 разлике). Када један тим освоји сет, од следећег поена се почиње нови сет у коме оба тима имају 0 поена на почетку.

**Како се може доћи до резултата 25 : 24?** Ово се може десити уколико у једном тренутку оба тима имају по 24 поена, тада ће наредни освојен поен довести до 25 : 24 или 24 : 25, али то неће завршити сет.

## Опис улаза

Са стандардног улаза се читава ниска карактера А или В који редом означавају да ли је поен добио тим А или тим В. Гарантовано је да су улазни подаци могући, односно да ако неко слово означава поен којим се завршава меч, онда иза њега нема више слова на улазу. Улазна ниска садржи најмање један, а највише 500 карактера.

## Опис излаза

У првом реду стандардног улаза исписати резултат у сетовима, у другом резултат у поенима у текућем сету (у сваком реду прво исписати резултат тима А, затим двотачку и на крају резултат тима В). Уколико је **меч завршен** исписати само резултат у сетовима, без резултата у поенима.

### Пример 1

*Узлаз* ААВАВААБВААААБАВВААБАВААААБВАААБАВААБАВААБАВВАБАВА *Излаз* 1:0  
2:4

## *Објашњење*

## *Објашњење*

Тим А осваја први сет резултатом 25 : 17 (сет се завршио у тренутку када је тим А освојио свој 25. поен), а у другом сету води тим В резултатом 2 : 4.

## Пример 2

Указ

BABABABABABABABBBBBBABAABAAAABABABABABABABABABABABABAABBBB  
ABBAAAABBBBABAABBAABABABAABABABABBBBAAABBBBBBAAABBABAA

Излаз

0:1  
27:26

## *Објашњење*

## *Објашњење*

Први сет осваја тим В резултатом 29 : 31. Сет се није завршио раније јер ниједан тим није успео да направи разлику од најмање 2 поена, а да је притом освојио бар 25 поена. У другом сету је тренутно резултат 27 : 26 и сет није готов јер још нико није направио разлику од 2 поена, а да притом има бар 25 поена (иако су тимови током самог сета водили за више од 2, нису тада имали барем 25 поена).

---

### Пример 3

Улаз

```
BABABABABABABABBBBABABABAABABABABABABABAABBABABBAAAAA  
BBABBBBAABBABABAABABABBBBAABAAABBBBBBAAABABBAAAAAAAAAAAAAA  
AAAAAAAAABBBBBBBBBBBBBBBBBBABAABABAABABAABBABBABB
```

Излаз

2:3

Објашњење

Објашњење

Први сет осваја тим В резултатом 29 : 31. Други сет осваја тим А резултатом 28 : 26. Трећи сет осваја тим А резултатом 25 : 0, а четврти тим В резултатом 0 : 25. Пети сет осваја тим В резултатом 11 : 15. Пошто је меч готов, не исписујемо поене већ само сетове.

Решење

```
class Tim:  
    def __init__(self):  
        self.setova = 0  
        self.poena = 0  
  
    def dodajPoenTimu(A, B):  
        A.poena += 1  
        taj_brejk = A.setova == 2 and B.setova == 2  
        cilj = 15 if taj_brejk else 25  
        if A.poena >= cilj and A.poena - B.poena >= 2:  
            A.setova += 1  
            A.poena = 0  
            B.poena = 0  
  
    A = Tim()  
    B = Tim()  
    poeni = input()  
    for poen in poeni:  
        if poen == 'A':  
            dodajPoenTimu(A, B)  
        elif poen == 'B':  
            dodajPoenTimu(B, A)  
  
    print(A.setova, B.setova, sep=":")  
    if A.setova < 3 and B.setova < 3:  
        print(A.poena, B.poena, sep=":")
```

## Задатак: Бејзбол хитац

Аутор: Филип Марић

Бејзбол је веома популаран спорт у САД. Џек је три пута ударио лоптицу и измерено је колико је она одлетала и то у мерама које се користе у САД (јарди, стопе, инчи). Нас занима колики је најдаљи хитац од та три, али у мерама које ми разумемо (метри и центиметри). Напиши програм који нам помаже да то одредимо.

Найомена: Један инч има 2,54 центиметра. Једна стопа има 12 инча, а један јард има 3 стопе.

### Опис улаза

Са стандардног улаза се уносе дужине три хитаца. За сваки хитац је дат цео број јарди, стопе и инча. Бројеви су дати у једном реду, развојени размаком, а ниједна дужина не прелази 50 јарди.

### Опис излаза

На стандардни излаз исписати два цела броја раздвојена једним размаком, број метара и центиметара најдаљег хица, при чему је број центиметара заокругљен на најближи цео број (који ће у свим примерима бити једнозначен).

### Пример 1

<i>Улаз</i>	<i>Излаз</i>
18 2 7	17 25
18 1 7	
14 0 9	

*Објашњење*

Најдужи хитац је онај од 18 јарди, 2 стопе и 7 инча, што је 1724,66 центиметара. Заокругљивањем на најближи цео број добија се 1725 центиметара, што је 17 метара и 25 центиметара.

### Пример 2

<i>Улаз</i>
21 1 7
18 0 3
24 2 11

*Излаз*

22 83

*Објашњење*

Најдужи хитац је онај од 24 јарда, 2 стопе и 11 инча, што је 2283,46 центиметара. Заокругљивањем на најближи цео број добија се 2283 центиметра, што је 22 метра и 83 центиметра.

### Решење

У  $j$  јарди,  $f$  стопе и  $i$  инча има  $12(3j + f) + i$  инча тј.  $(12(3j + f) + i) \cdot 2,54$  центиметара. Дужине сва три хица претварамо у центиметре, одређујемо њихов максимум и заокружујемо добијени број центиметара на најближи цео број. Након тога број метара добијамо као целобржни количник са 100, а број центиметара као остатак при дељењу са 100.

```
j, s, i = map(int, input().split())
d1 = (j*3 + s)*12 + i
j, s, i = map(int, input().split())
d2 = (j*3 + s)*12 + i
j, s, i = map(int, input().split())
d3 = (j*3 + s)*12 + i

cm = int(round(max(d1, d2, d3) * 2.54))
print(cm // 100, cm % 100)
```

## Задатак: Годишње доба

Аутор: Филип Марић

Рећи ћемо да пролеће почиње 20. марта (укључујући и тај дан) и траје до 21. јуна (без тог дана). Лето почиње 21. јуна (укључујући тај дан) и траје до 23. септембра (без тог дана). Јесен почиње 23. септембра (укључујући и тај дан) и траје до 21. децембра (без тог дана). Осталих дана је зима. Напиши програм који на основу унетог датума одређује годишње доба.

### Опис улаза

Са стандардног улаза се уноси дан, а затим и месец (сваки број у посебном реду). Број 1 означава јануар, 2 фебруар, 3 март итд. Сматрати да је унети датум исправан.

### Опис излаза

На стандардни излаз исписати слово p (пролеће), l (лето), j (јесен) или z (зима).

---

### Пример 1

Улаз            Излаз  
1                z  
2

Објашњење

Унет је први фебруар и он је током зиме.

### Пример 2

Улаз

20  
3

Излаз

p

Објашњење

Унет је 20. март и сматрамо да је он први дан пролећа.

### Пример 3

Улаз

19  
3

Излаз

z

### Пример 4

Улаз

21  
6

Излаз

l

### Решење

Задатак је решити угнежђеним гранањем. Прво испитујемо месец. Ако откријемо да се ради о неком месецу који се простира кроз два годишња доба, посебно испитујемо дан.

```
dan = int(input())
mesec = int(input())

if mesec < 3:
    print('z')
elif mesec == 3:
    if dan < 20:
        print('z')
    else:
        print('p')
elif mesec < 6:
    print('p')
elif mesec == 6:
    if dan < 21:
        print('p')
    else:
        print('l')
elif mesec < 9:
```

```

print('l')
elif mesec == 9:
    if dan < 23:
        print('l')
    else:
        print('j')
elif mesec < 12:
    print('j')
elif mesec == 12:
    if dan < 21:
        print('j')
    else:
        print('z')

```

Задатак можемо решити и тако што датум представимо у облику четвороцифреног броја (који добијамо тако што месец помножмо са 100 и додамо дан).

```

dan = int(input())
mesec = int(input())
mes_dan = 100 * mesec + dan

if mes_dan < 320:
    print('z')
elif mes_dan < 621:
    print('p')
elif mes_dan < 923:
    print('l')
elif mes_dan < 1221:
    print('j')
else:
    print('z')

```

## Задатак: Пар-непар у круг

*Автор: Филип Марић*

За дати природан број  $n$  исписати све низове бројева дужине  $2n$  који испуњавају следећа три услова:

- садрже сваки број од 1 до  $2n$  тачно једном;
- у низу се прво налазе непарни, а затим парни бројеви;
- део са непарним бројевима се добија ротирањем низа  $1, 3, \dots, 2n - 1$ , а део са парним бројевима се добија ротирањем низа  $2, 4, \dots, 2n$ . Ротирање низа подразумева пребацивање првог елемента на крај низа (и то може бити поновљено 0 или више пута).

На пример, за  $n = 3$  један такав низ је  $3, 5, 1, 2, 4, 6$ . Заиста, у њему се сваки број од 1 до  $2n = 6$  јавља тачно једном, прво иду непарни бројеви  $3, 5, 1$ , а затим парни  $2, 4, 6$ , део  $3, 5, 1$  се добија ротирањем низа  $1, 3, 5$  за једну позицију (први елемент низа  $1, 3, 5$  је пређачен на крај), а део  $2, 4, 6$  ротирањем низа  $2, 4, 6$  за нула позиција.

### Опис улаза

Са стандардног улаза се учитава број  $n$  ( $1 \leq n \leq 50$ ).

### Опис излаза

На стандардни излаз исписати све тражене низове. Низови треба да буду поређани лексикографски (како је приказано у примерима): када се упореде било која два исписана низа, вредност првог елемента који им је различит мора бити мања у оном низу који је раније исписан.

---

**Пример 1**

Улаз	Излаз
2	1 3 2 4
	1 3 4 2
	3 1 2 4
	3 1 4 2

**Пример 2**

Улаз	Излаз
3	1 3 5 2 4 6
	1 3 5 4 6 2
	1 3 5 6 2 4
	3 5 1 2 4 6
	3 5 1 4 6 2
	3 5 1 6 2 4
	5 1 3 2 4 6
	5 1 3 4 6 2
	5 1 3 6 2 4

**Решење**

Задатак решавамо коришћењем уgnежђених петљи. Две спољне петље набрајају за колико места се ротирају непарни и за колико места се ротирају парни бројеви. Када су одређени ти бројеви (назовимо их  $i$  и  $j$ ) исписујемо низове непарних тј. парних бројева. Сви непарни бројеви су облика  $2k + 1$ , а парни облика  $2k + 2$ . Ефекат ротације постижемо тако што  $k$  увећамо за  $i$  тј.  $j$ , а затим пронађемо остатак при дељењу са  $n$ . На пример, ако је  $n = 4$  и  $i = 2$  врши се следећи испис низа непарних бројева:

```
k:          0   1   2   3
2k+1:       1   3   5   7
(k+i)%n    2   3   0   1
2((k+i)%n)+1  5   7   1   3

n = int(input())
for i in range(n):
    for j in range(n):
        for k in range(n):
            print(2*((k+i)%n)+1, end=" ")
        for k in range(n):
            print(2*((k+j)%n) + 2, end=" ")
    print()
```