Друштво математичара Србије
Фондација Петља

Такмичења из програмирања за ученике основних школа

2019. година

Садржај

Глава 1

Сезона 2019/2020.

1.1 Први круг квалификација

Задатак: Књига

Поставка: Књига има N поглавља. Никола је првог дана прочитао A поглавља, а другог дана два поглавља више него првог. Написати програм који учитава целе позитивне бројеве N, A, и исписује колико поглавља је остало Николи да прочита.

Улаз: Са стандардног улаза се у првом реду уноси број N ($1 \leq N \leq 99$), а у другом реду број A ($1 \leq A \leq 99$). Улазни подаци су такви да преостали број поглавља никад није негативан.

Излаз: На стандардни излаз исписати један број, број поглавља која Никола још није прочитао.

Пример 1

Улаз	*Излаз*
15	9
2	

Пример 2

Улаз	*Излаз*
20	0
9	

Решење: Решење је сасвим једноставно. Ако је Никола првог дана прочитао A поглавља а другог дана $A+2$ поглавља, то значи да му остаје да прочита још $N-A-(A+2)$ поглавља.

```
n = int(input())
a = int(input())
print(n - a - (a+2))
```

Задатак: Сијалица

Поставка: Стубови уличне расвете су нумерисани редом по улицама. У свакој улици има по N стубова, тако да су стубови у првој улици нумерисани 1, 2, … N, у другој су бројеви стубова $N+1$, $N+2$, … $2N$ итд. Мирко је добио задатак да у свакој другој улици замени сијалицу на сваком трећем стубу. Када је дошао до стуба са бројем A,

Мирко се забројао. Написати програм који учитава целе позитивне бројеве N и A и одговара на питање да ли на том стубу треба заменити сијалицу.

Улаз: Са стандардног улаза се у првом реду уноси број N ($1 \leq N \leq 1000$), а у другом реду број A ($1 \leq A \leq 1000$).

Излаз: На стандардни излаз исписати само реч da ili ne.

Пример 1

Улаз	*Излаз*
20	da
75	

Пример 2

Улаз	*Излаз*
100	ne
300	

Решење: Када би се стубови и улице бројали од 0, тада би редни број улице могао да се добије као $A \operatorname{div} N$, а редни број стуба у улици као $A \bmod N$ (где су са div и $\bmod$ означени целобројни количник, тј. остатак при дељењу два цела броја). Међутим, пошто се у задатку броји од 1, учитану ознаку на стубу треба смањити за 1, а добијене резултате за редни број улице и стуба у улици повећати за 1. То значи да редни број улице треба рачунати као $(A-1) \operatorname{div} N + 1$, а редни број стуба у улици као $(A-1) \bmod N + 1$.

Након овог израчунавања потребно је још само проверити да ли је редни борј улице дељив са 2 и редни број стуба у улици дељив са 3.

```
n = int(input())
a = int(input())
ulica = (a-1) // n + 1
br_u_ulici = (a-1) % n + 1
if (ulica % 2 == 0) and (br_u_ulici % 3 == 0):
    print('da')
else:
    print('ne')
```

Задатак: Запета

Поставка: Дат је низ ASCII карактера, међу којима се појављује тачно један знак "," (запета) и на самом крају тачно један знак "." (тачка). Сви остали знаци, ако их има, су слова енглеске абецеде, цифре, заграде и размаци и они могу да се понављају. Написати програм који учитава дати низ карактера, а на стандардни излаз исписује текст у коме су део до запете и део од запете разменили места, док тачка остаје на крају текста.

Улаз: На стандардном улазу се у једном реду налази низ од највише 100 ASCII карактера.

Излаз: На стандардни излаз исписати измењени низ карактера.

Пример 1

Улаз	*Излаз*
Da si hteo, mogao si.	mogao si,Da si hteo.

Пример 2

Улаз	*Излаз*
,a b c.	a b c,.

Пример 3

Улаз	*Излаз*
`x,.`	`,x.`

Решење: У овом задатку је најпогодније да се прочита цео ред улаза као један стринг (без обзира на то да ли стринг садржи размаке). Након учитавања можемо да издвојимо део A од почетка стринга до запете (не укључујући запету) а затим и део B од запете до тачке (не укључујући запету и тачку).

У језику Python поделу стринга на делове који су раздвојени сепаратором можемо постићи методом `split` (сепаратор наводимо као аргумент). Последњи карактер стринга `s` можемо обрисати помоћу `s[:-1]` (издвајамо карактере од почетка па све до последњег, не укључујући тај последњи).

После оваквог издвајања остаје још само да испишемо делове у траженом редоследу (део B, запета, део A и на крају тачка).

```
s = input()
s = s[:-1] # odbacujemo tacku
a, b = s.split(',')
print(b, ',', a,'.', sep='')
```

Задатак: Непливачи

Поставка: Деда Раде жели да упише својих четворо унука непливача у школу пливања. Инструктор му је рекао да је остао само један слободан термин, за који могу да се пријаве само деца виша од 110 сантиметара. Деда Раде не жели да раздваја унуке, па ће их уписати у школу пливања само ако сви испуњавају услов. Написати програм који одређује да ли деда Раде већ сада може да упише свих четворо унука у школу пливања.

Улаз: Са стандардног улаза се уносе четири цела позитивна броја не већа од 180, сваки у посебном реду – висине деда Радетових унука.

Излаз: На стандардни излаз исписати реч SVI ако је свако од четворо деце више од 110cm, а реч NIKO ако бар једно дете није више од 110 cm.

Пример 1

Улаз	*Излаз*
`131`	`SVI`
`111`	
`128`	
`117`	

Пример 2

Улаз	*Излаз*
`131`	`NIKO`
`110`	
`128`	
`117`	

Решење: Након учитавања потребно је само проверити да ли су сва 4 учитана броја већа од 110. Најједноставнији начин је употреба сложеног логичког услова.

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())
if (a > 110 and b > 110 and c > 110 and d > 110):
    print('SVI')
```

```
else:
    print('NIKO')
```

Задатак: Сусрет

Поставка: Новак и Јагош су стари другари и они се често без договарања сачекују на свом омиљеном месту у неко уобичајено време, па ако први дочека другог онда проведу неко време дружећи се. Када Новак стигне први, он чека Јагоша 10 минута, а Јагош (ако он стигне први) чека Новака 15 минута.

Написати програм који за дата времена данашњег појављивања Новака и Јагоша одговара на питање да ли су се састали.

Улаз: Учитавају се четири цела броја, NS, NM, JS, JM редом, сваки у посебном реду стандардног улаза. Бројеви NS, NM представљају сат и минут Новаковог доласка, а JS, JM представљају сат и минут Јагошевог доласка. Подаци ће представљати исправно записана времена (то јест, важиће $0 \leq NS \leq 23$, $0 \leq NM \leq 59$, $0 \leq JS \leq 23$, $0 \leq JM \leq 59$).

Излаз: На стандардни излаз исписати само реч da ili ne.

Пример 1		**Пример 2**		**Пример 3**		**Пример 4**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
10	da	21	ne	0	da	0	ne
15		15		15		0	
10		21		0		23	
25		26		0		59	

Решење: Задатак се једноставније решава ако се после учитавања прво израчуна број минута од почетка дана до Николиног и Јагошевог доласка: $N = N_{sat} \cdot 60 + N_{min}$, и $J = J_{sat} \cdot 60 + J_{min}$.

Уīнежђена īранања

Решавање задатка сада можемо да довршимо тако што најпре проверимо ко је први стигао на место састанка. Ако је то Новак, онда проверавамо да је Јагош дошао највише 10 минута након Новака, а ако је Јагош први стигао, онда проверавамо да ли је Новак дошао највише 15 минута након Јагоша.

```
nsat = int(input())
nmin = int(input())
jsat = int(input())
jmin = int(input())
n = nsat*60 + nmin
j = jsat*60 + jmin
if (n<j): # ako je Novak dosao pre
    if (j <= n+10):
        print('da')
    else:
        print('ne')
else: # ako je Jagos dosao pre (ili su dosli istovremeno)
    if (n <= j+15):
```

```
        print('da')
    else:
        print('ne')
```

Сложени услов

Можемо и да формирамо сложени услов, који директно одговара на питање да ли је дошло до сусрета. Да би одговор био потврдан, треба да важи $N \leq J \leq N + 10$ или $J \leq N \leq j + 15$, у противном одговор је одречан.

```
nsat = int(input())
nmin = int(input())
jsat = int(input())
jmin = int(input())
n = nsat*60 + nmin
j = jsat*60 + jmin
if (n <= j and j <= n+10) or (j <= n and n <= j+15):
    print('da')
else:
    print('ne')
```

Задатак: Јаја

Поставка: У сваку кутију за јаја може да стане K јаја. Када Јоца пакује јаја, он користи најмањи број кутија у које јаја могу да стану, али пошто је сујеверан, он никада не оставља у кутији (позитиван) број јаја дељив са 3.

Написати програм који одређује колико кутија треба Јоци да би спаковао J јаја.

Улаз: у првом реду стандардног улаза је број K, који је увек 10 или 15. У другом реду је цео број J, такав да је $0 \leq J \leq 100$.

Излаз: На стандардни излаз исписати један цео број, број кутија које ће Јоца употребити.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
10	2	15	1
18		10	

Решење: Прво што можемо да приметимо је да у кутије од 15 јаја можемо да ставимо највише 14 јаја. Према томе, оваквим кутијама можемо одмах након учитавања да смањимо величину за 1 и поједноставимо даље разматрање.

Погледајмо шта би се догодило када би се кутије пуниле редом до краја. Тада би број потребних кутија био $\lceil \frac{J}{K} \rceil$, где је J број јаја која треба спаковати, а K број јаја која стају у кутију.

Пошто након почетне исправке величина кутије K више није дељива са 3, то услов дељивости може да утиче само на последњу употребљену кутију. Зато још треба посебно испитати ситуацију када је број јаја у последњој употребљеној кутији дељив са 3. Таква сиутација може бити разрешена или узимањем нове кутије (само ако је

неопходно), или пребацивањем једног или више јаја из претходно попуњених кутија у последњу. Додавање јаја у последњу кутију није могуће само у следећа два случаја:

(1) када нема ниједне претходно попуњене кутије из које бисмо "позајмили" јаја (то јест када је $J \leq K$).

(2) када у последњој кутији има места само за још једно јаје (то јест када је $J \bmod K = K - 1$). У овом случају додавање није могуће зато што би пребацивањем једног јајета из било које попуњене кутије у тој претходно попуњеној кутији остао број јаја дељив са 3.

У свим осталим случајевима се из претпоследње у последњу кутију могу пребацити бар једно или два јаја. На тај начин, број јаја у последњој кутији у сваком случају више неће бити дељив са 3, а од два могућа пребацивања (једно или два јаја) једно од тих пребацивања мора одговарати и претпоследњој кутији, тако да ни у њој број јаја не буде дељив са 3. Према томе, у свим случајевима осим два издвојена раније, број кутија се не мора повећавати.

```
vel_kutije = int(input())
br_jaja = int(input())
if vel_kutije % 3 == 0:
    vel_kutije -= 1

br_kutija = (br_jaja  + vel_kutije - 1) // vel_kutije
u_poslednjoj = br_jaja % vel_kutije

# ako je u poslednjoj kutiji nezgodan broj
if u_poslednjoj > 0 and u_poslednjoj % 3 == 0:
    # ako nemamo odakle u nju da dodamo (poslednja kutija je jedina)
    # ili ako bi se dodavanjem pokvarila prethodna kutija
    if br_kutija == 1 or u_poslednjoj + 1 == vel_kutije:
        br_kutija += 1  # onda ce trebati kutija vise

print(br_kutija)
```

Задатак: Огрлица

Поставка: Сара чува перле у N кутијица нумерисаних бројевима од 1 до N. Она увек прави огрлицу тако што из сваке од N кутијица редом по бројевима узме по једну перлу и у истом редоследу наниже перле на конац.

Написати програм који одређује на колико начина Сара може да изабере перле за следећу огрлицу.

Улаз: у првом реду стандардног улаза је цео позитиван број N, број кутијица, не већи од 10. У сваком од следећих N редова је по један цео једноцифрен број, број перли у одговарајућој кутијици по реду.

Излаз: На стандардни излаз исписати један цео број, број начина на које Сара може да направи следећу огрлицу.

Пример 1

Улаз	*Излаз*
5 3 3 2 5 1	90

Пример 2

Улаз	*Излаз*
3 7 0 2	0

Решење: Пошто се перле могу комбиновати свака са сваком, број начина да се наниже огрлица је једнак производу броја перли из појединих кутија. Дакле, потребно је само помножити дате бројеве перли и исписати производ тих бројева.

```
n = int(input())
p = 1
for i in range(n):
    a = int(input())
    p = p * a
print(p)
```

Задатак: Секција

Поставка: У школи је велико интересовање за програмерску секцију, на којој ће се правити рачунарске игре. Наставник је поставио услов да на секцију могу да иду само они ученици који имају 5 из информатике и бар 4 из математике.

Написати програм који одређује колико ученика може да се пријави за секцију.

Улаз: у првом реду број N, број ученика заинтересованих за секцију, природан број не већи од 200. У сваком од следећих N редова низ оцена једног од заинтересованих ученика из свих 11 предмета редом, без размака. Оцена из математике је шеста, а из информатике девета у низу од 11 цифара.

Излаз: На стандардни излаз исписати један цео број, број ученика који испуњавају услов.

Пример 1

Улаз	*Излаз*
4 55555355455 55555455455 55555455555 22222522522	2

Пример 2

Улаз	*Излаз*
2 55555555455 55555355555	0

Решење: Најједноставније је да учитавамо сваки ред као стринг од 11 карактера (цифара). Потребно је да пребројимо оне стрингове код којих је шеста цифра слева (бројећи од 1) већа од 3, а девета слева једнака 5.

Након пребројавања исписујемо вредност бројача.

```
n = int(input())
br = 0
for i in range(n):
```

```
    s = input()
    if int(s[5]) > 3 and int(s[8]) == 5:
        br += 1
print(br)
```

Задатак: Статистике33

Поставка: Чувени кошаркаш Дабло Трипловић има толико добре статистике, да се већ помало сумња да су његови успеси добро пребројани.

Написати програм који одређује колико пута је Трипловић постигао такозвани трипл–дабл.

Улаз: На стандардноим улазу се у првом реду налази цео број N, $(1 \leq N \leq 100)$, број утакмица које је Трипловић одиграо у сезони. У сваком од следећих N редова по 3 цела броја раздвојена по једним размаком: број поена, скокова и асистенција редом (ови бројеви нису већи од 1000).

Излаз: На стандардни излаз исписати један број, број утакмица на којима је Трипловић постигао трипл–дабл.

Напомена

За потребе овог задатка трипл–дабл дефинишемо као најмање двоцифрен учинак (10 или више) у све три категорије које се прате. Другим речима: сва три броја већа од 9 у једном реду улаза.

Пример

Улаз	*Излаз*
3 20 9 7 127 12 11 11 14 12	2

Решење: Потребно је пребројати редове улаза у којима су сва три броја већа од 9. Увешћемо бројач који пре петље постављамо на 0, а затим у петљи учитавамо по три броја, прверавамо да ли су сва три већа од 9, и ако јесу - увећавамо бројач.

По завршетку петље исписујемо вредност бројача.

```
n = int(input())
br_tripl_dabl = 0
for utak in range(n):
    s = input().split()
    a, b, c = int(s[0]), int(s[1]), int(s[2])
    if a > 9 and b > 9 and c > 9:
        br_tripl_dabl += 1
print(br_tripl_dabl)
```

Задатак: Статистике35

Поставка: Чувени кошаркаш Дабло Трипловић има толико добре статистике, да се већ помало сумња да су његови успеси добро пребројани.

Написати програм који одређује колико пута је Трипловић постигао такозвани трипл–дабл.

Улаз: Са стандардног улаза се у првом реду уноси цео број N $(1 \leq N \leq 100)$, број утакмица које је Трипловић одиграо у сезони. У сваком од следећих N редова је по 5 целих бројева раздвојених по једним размаком: број поена, скокова, асистенција, блокада и украдених лопти редом (ови бројеви нису већи од 1000).

Излаз: На стандардни излаз исписати један цео број, број утакмица на којима је Трипловић постигао трипл–дабл.

Напомена

За потребе овог задатка трипл–дабл дефинишемо као најмање двоцифрен учинак (10 или више) у бар три од пет категорија које се прате. Другим речима: бар три броја већа од 9 у једном реду улаза.

Пример

Улаз	*Излаз*

`3`
`20 9 7 2 1`
`127 12 11 3 0`
`0 10 11 14 12`
	`2`

Решење: У овом задатку треба пребројати редове улаза у којима су бар три од пет бројева већи од 9. Да бисмо установили да ли неки ред улаза треба бројати, треба у сваком реду пребројати елементе који су већи од 9. То значи да ћемо у овом задатку применити технику пребројавања "на два нивоа": глобално бројимо редове који испуњавају услов, а у сваком реду бројимо елементе веће од 9.

Формираћемо двоструку петљу: спољна петља ће ићи по утакмицама (тј. редовима улаза), а унутрашња по категоријама које се прате на свакој утакмици. Бројач трипл-даблова иницијализујемо пре спољне петље, а бројач двоцифрених учинака пре унутрашње.

```
n = int(input())
br_tripl_dabl = 0
for utak in range(n):
    s = input().split()
    br_dvoc = 0
    for kateg in range(5):
        if int(s[kateg]) > 9:
            br_dvoc += 1
    if br_dvoc >= 3:
        br_tripl_dabl += 1
print(br_tripl_dabl)
```

Додатна решења овог задатка су доступна на страни 19.

Задатак: Близанци

Поставка: Марија и Петар су близанци и желимо да свакоме од њих двоје купимо по једно одело као поклон за рођендан, али тако да се цене та два поклона што мање разликују (при томе није битно чији поклон ће бити скупљи).

Написати програм који учитава цене свих женских одела и свих мушких одела, а одређује и исписује најмању разлику између цена женског и мушког одела.

Улаз: Опис улаза: са стандардног улаза се учитава:

- у првом реду број мушких одела m ($1 \leq m \leq 50000$),
- у другом реду m целих бројева (цели бројеви између 1 и $2 \cdot 10^9$ раздвојени по једним размаком) - цене мушких одела
- у трећем реду број женских одела z ($1 \leq z \leq 50000$)
- и у четвртом реду z целих бројева (цели бројеви између 1 и $2 \cdot 10^9$ раздвојени по једним размаком) - цене женских одела.

Излаз: На стандардни излаз исписати најмању вредност разлике цена мушког и женског одела.

Пример

Улаз	*Излаз*
5 4680 2120 7940 11530 17820 4 850 13420 5770 6300	1090

Објашњење

Најмања разлика се постиже када се купе одела чије су цене 4680 и 5770 динара.

Решење:

Анализа

Наивно решење

Један могући приступ је да одредимо разлику (прецизније, апсолутну вредност разлике) у цени између сваког мушког и сваког женског одела, па од тих разлика нађемо најмању. Овакво решење ће дати тачан резултат у мањим примерима, али на већим примерима се неће извршити на време.

```
n1 = int(input())
a1 = list(map(int, input().split()))
n2 = int(input())
a2 = list(map(int, input().split()))
min_razlika = abs(a2[0] - a1[0])

for i1 in range(n1):
    for i2 in range(n2):
        min_razlika = min(min_razlika, abs(a1[i1] - a2[i2]))
```

```
print(min_razlika)
```

Упоредни пролаз кроз уређене низове

Ефикаснији приступ је да се низови цена најпре сортирају, а да се затим истовремено пролази кроз оба низа, рачунајући разлику текућих елемената и напредујући у оном низу у којем је цена тренутно мања. Успут се, наравно, по потреби ажурира најмања забележена разлика. Када се стигне до краја било којег низа, поступак је завршен и најмања забележена разлика је тада и укупно најмања.

Заиста, пошто су низови сортирани, када се упореде почетни елементи из оба низа, онај који је мањи од њих нема потребе упоређивати са осталим елементима низа коме он не припада, јер ће разлика моћи бити само већа (јер је тај низ сортиран). Тај елемент онда можемо избацити из даљег разматрања тако што ћемо у низу у ком се он налази прећи на следећи елемент. У специјалном случају када су почетни елементи оба низа једнаки, разлика је једнака нули, што је најмања могућа разлика, па нема потребе вршити даљу анализу.

На пример, нека су након сортирања вредности једнаке следећим.

```
1 14 28 33 45
8 21 22 41 56 68
```

- Прво поредимо елементе 1 и 8. Разлика је 7. Разлика између броја 1 и свих даљих бројева у доњем низу је већа од 7, па број 1 не морамо више анализирати.
- Након тога поредимо бројеве 14 и 8 и добијамо разлику 6. Разлика између броја 8 и свих бројева иза 14 је већа, па сада ни 8 не морамо више анализирати.
- Поредимо сада бројеве 14 и 21, разлика је 7, а 14 не морамо више да анализирамо.
- И разлика између 28 и 21 је 7, а број 21 не морамо више да анализирамо.
- Разлика између 28 и 22 је 6, а 22 не морамо да анализирамо даље.
- Разлика између 28 и 41 је 13, а 28 не морамо да анализирамо даље.
- Разлика између 33 и 41 је 8, а 33 не морамо да анализирамо даље.
- Разлика између 45 и 41 је 4, а 41 не морамо да анализирамо даље.
- Разлика између 45 и 56 је 11, а 45 не морамо да анализирамо даље. Пошто нема више елемената у горњем низу, поступак се завршава.

Можемо закључити да је најмања могућа разлика једнака 4 (за бројеве 41 и 45).

```
n1 = int(input())
a1 = sorted(map(int, input().split()))
n2 = int(input())
a2 = sorted(map(int, input().split()))
i1, i2 = 0, 0
min_razlika = abs(a2[0] - a1[0])
while i1 < n1 and i2 < n2:
```

```
    if a1[i1] <= a2[i2]:
        min_razlika = min(min_razlika, a2[i2] - a1[i1])
        i1 += 1
    else:
        min_razlika = min(min_razlika, a1[i1] - a2[i2])
        i2 += 1

print(min_razlika)
```

Задатак: Поклони за родитеље

Поставка: Ненад жели да са путовања донесе поклоне својим родитељима. Пошто путује нискотарифном авио-компанијом и не жели да плаћа додатне трошкове, ограничена је маса коју може да понесе у свом ранцу. Оцу ће купити поклон у једној, а мајци у другој продавници. Написати програм који на основу познатих маса и цена свих поклона у обе продавнице помаже Ненаду да сваком од родитеља купи по један поклон, тако да поклони збирно буду што вреднији и да заједно не прелазе дато ограничење масе.

Улаз: са стандардног улаза се уносе следећи подаци: У првом реду број N_1 ($1 \leq N_1 \leq 10^5$), број производа у првој продавници. У сваком од наредних N_1 редова по два цела броја, сваки између 1 и 10^9, који представљају масу и затим цену једног производа из прве продавнице. У следећем реду број N_2 ($1 \leq N_2 \leq 10^5$), број производа у другој продавници. У сваком од наредних N_2 редова по два цела броја, сваки између 1 и 10^9, који представљају масу и затим цену једног производа из друге продавнице.

У следећем и последњем реду цео број између 1 и $2 \cdot 10^9$, највећа дозвољена маса за пар поклона заједно.

Излаз: На стандардни излаз исписати највећи могући збир цена првог и другог поклона које је могуће понети.

Пример

Улаз	*Излаз*
3 2 4 3 6 4 5 3 2 3 3 2 4 5 6	9

Решење:

Наивно решење

Можемо да проверимо сваки пар предмета (у коме је један предмет из једне, а други из друге продавнице). Ово решење је квадратне сложености, па може да донесе поене само за мале примере.

```
# ucitamo podatke i probamo
# svaki proizvod iz prve sa svakim iz druge prodavnice

n1 = int(input())
a1 = []
for i in range(n1):
    a1.append(list(map(int, input().split()))) # [masa, cena]

n2 = int(input())
a2 = []
for i in range(n2):
    a2.append(list(map(int, input().split()))) # [masa, cena]

maks_masa = int(input()) # najveca masa koja moze da stane u ranac

maks_cena = 0
for masa1, cena1 in a1:
    if masa1 >= maks_masa: continue
    for masa2, cena2 in a2:
        if masa1 + masa2 <= maks_masa and cena1 + cena2 > maks_cena:
            maks_cena = cena1 + cena2

print(maks_cena)
```

Нагађање

Такмичар који не уме да реши задатак, може да покуша да "упеца" неки број поена тако што напише једноставнији програм, који у суштини нагађа резултат. Нагађање може бити мање или више успешно.

Један могући покушај је да се нађе највећа цена из сваке продавнице и да се испише збир тих највећих цена.

Овај покушај нагађања се може комбиновати са наивним (неефикасним) решењем, тако да за довољно мале дужине низова задатак решавамо егзактно квадратним алгоритмом, а за велике примере покушамо да погодимо решење (што обично не даје добре резултате).

```
maks_cena1 = 0
maks_cena2 = 0

n1 = int(input())
for i in range(n1):
    masa, cena = input().split() # stringovi
    maks_cena1 = max(maks_cena1, int(cena))

n2 = int(input())
for i in range(n2):
    masa, cena = input().split() # stringovi
    maks_cena2 = max(maks_cena2, int(cena))
```

```
maks_masa = int(input()) # najveca masa koja moze da stane u ranac
print(maks_cena1 + maks_cena2)
```

Сортирање и максимуми префикса

Решење које претендује на максималан број поена треба да буде ефикасније од квадратног. У ту сврху увек је корисно да подаци буду на неки начин сортирани. Питање је: како сортирати?

Ако оба низа уредимо тако да масе расту, можемо у једном низу да кренемо од почетка (од најлакшег предмета) а у другом од краја. На тај начин брзо долазимо до парова предмета који су у збиру испод лимита масе, а при томе најближи том лимиту. Ипак, ово није довољно да се задатак реши ефикасно, јер масе и цене не расту заједно, па предмет мање масе може бити вреднији. То значи да при прегледању свих парова који су кандидати за најбољи пар не бисмо избегли квадратно време.

Уређивање по цени није боље, јер бисмо тада за фиксиран предмет из једне продавнице морали међу онима из друге (који у збиру са првим) поправљју укупну вредност, да тражимо оне који се уклапају по маси и опет спадамо на квадратно време.

Оно што би нам овде помогло (након сортирања оба низа предмета по маси) је да можемо за сваки предмет P из једног низа да брзо одговоримо колико вреди највреднији предмет тог низа, чија маса није већа од масе предмета P. Такве одговоре можемо да припремимо након сортирања а пре испитивања парова, тако што фромирамо још један низ у коме ће се ти одговори наћи. Конкретније, можемо у једном пролазу кроз низ сортиран по маси да за сваки префикс тог низа израчунамо цену највреднијег предмета у префиксу. Након тога првобитна идеја о истовременом проласку кроз један низ од почетка а кроз други од краја доводи до ефикасног решења.

Прикажимо рад овог алгоритма на једном примеру. Нека су цене и тежине мушких поклона дате у левој, а женских у десној колони, при чему смо поклоне сортирали по тежинама и нека је капацитет ранца једнак 20.

```
4   3                5   6          6
7   9                9   4          6
12  4                10  12         12
16  11               14  9          12
19  2                18  7          12
```

Покушавамо да за први мушки поклон (то је (4, 3)) пронађемо скуп поклона који се могу са њим упарити. Пошто је други низ сортиран, то ће бити неки поклони који се налазе на почетку тог низа. Крећемо од краја, елиминишемо последњи женски поклон јер је збир 4 + 18 већи од 20 и проналазимо да је последњи женски поклон који се може упарити са првим мушким онај који има масу 14. Потребно је пронаћи најскупљи женски поклон који има масу мању или једнаку 14. У томе нам помаже последња колона у којој су записани максимуми префикса. За поклон чија је маса 14 можемо очитати вредност 12, што значи да постоји неки женски поклон чија је маса мања или једнака 14 који има вредност 12 (то је поклон (10, 12)). Дакле, ако купимо први мушки поклон, тада је највећа цена коју можемо постићи једнака 3 + 12 = 15.

Прелазимо на други мушки поклон (то је (7, 9)) и одређујемо женске поклоне са којима

се он може комбиновати. Веома важна напомена је то да се поклони који се нису могли укомбиновати са претходним мушким поклонима, не могу укомбиновати ни са текућим (јер је он још тежи од претходних). Дакле, довољно је само међу оним поклонима који су се могли укомбиновати са претходним поклоном наћи оне који се могу укомбиновати са текућим. Пошто је низ женских поклона сортиран по тежини, анализирамо и евентуално елиминишемо елементе са његовог краја (тј. краја оног дела низа где смо се претходно зауставили). Поклон масе 14 се не може комбиновати са поклоном масе 7 (јер им је укупна маса 21 већа од носивости ранца 20). Најтежи женски поклон који се може упарити са оним мушким масе 7 је онај чија је маса 10. Поново на основу низа максимума префикса очитавамо да је највреднији женски поклон чија маса не прелази 10 онај чија је вредност 12 (то је опет (10, 12)), па се његовим комбиновањем са мушким поклоном (7, 9) добија вредност 12 + 9 = 21, што је боље од претходне.

Прелазимо на следећи мушки поклон (то је (12, 4)), елиминишемо женске поклоне (10, 12) и (9, 4) јер се они не могу комбиновати са мушким поклоном масе 12 и закључујемо да је једини женски поклон који се може комбиновати са (12, 4) поклон (5, 6). Тиме добијамо вредност 4+6 = 10, што је лошије од претходне.

Преласком на следећи мушки поклон (то је (16, 11)), елиминишемо и женски поклон (5, 6) и закључујемо да се ни један мушки поклон који је тежак 16 или више не може укомбиновати ни са једним женским поклоном.

Дакле, оптимално упаривање је упаривање поклона (7, 9) и (10, 12).

```
# ucitamo podatke, sortiramo,
# izracunamo maksimalne cene prefiksa druge grupe proizvoda
# idemo kroz prvu grupu sleva a kroz drugu zdesna, tako da
# za svaki iz prve grupe nadjemo najvredniji iz druge koji moze da stane
# u ranac sa njim


n1 = int(input())
a1 = []
for i in range(n1):
    a1.append(list(map(int, input().split()))) # [masa, cena]

n2 = int(input())
a2 = []
for i in range(n2):
    a2.append(list(map(int, input().split()))) # [masa, cena]

maks_masa = int(input()) # najveca masa koja moze da stane u ranac

a1.sort()
a2.sort()

maks_cena_do2 = [0]
for i in range(n2):
```

```
    maks_cena_do2.append(max(maks_cena_do2[i], a2[i][1]))

maks_cena = 0 # maksimalni zbir cena dva proizvoda
i, j = 0, n2-1
while i < n1 and a1[i][0] < maks_masa:
    while j >= 0 and a1[i][0] + a2[j][0] > maks_masa:
        j -= 1
    if j >= 0:
        maks_cena = max(maks_cena, a1[i][1] + maks_cena_do2[j+1])
    i += 1

print(maks_cena)
```

Сортирање, максимуми префикса и бинарна претрага

Претходна идеја се може мало и изменити. Довољно је да сортирамо само други низ и израчунамо максимуме његових префикса, као у претходном решењу. Након тога можемо за сваки предмет неуређеног првог низа да бинарном претрагом нађемо предмет највеће масе из другог низа, који се може понети заједно са првим, а онда (користећи максимуме префикса другог низа) да нађемо и вредност највреднијег предмета у другом низу, који се може понети заједно са текућим предметом из првог низа.

```
import bisect

n1 = int(input())
a1 = []
for i in range(n1):
    a1.append(list(map(int, input().split()))) # [masa, cena]

n2 = int(input())
a2 = []
for i in range(n2):
    a2.append(list(map(int, input().split()))) # [masa, cena]

maks_masa = int(input()) # najveca masa koja moze da stane u ranac
a2.sort() # sortiramo proizvode iz druge prodavnice po masi

# za svaku poziciju u sortiranom nizu proizvoda iz druge prodavnice
# odredjujemo maksimalnu cenu proizvoda striktno pre te pozicije
maks_cena_do = [0]
for i in range(n2):
    maks_cena_do.append(max(maks_cena_do[i], a2[i][1]))

beskonacna_cena =  maks_cena_do[-1] + 1

maks_cena = 0 # maksimalni zbir cena dva proizvoda
for masa1, cena1 in a1: # za svaki proizvod iz prve prodavnice
  # odredjujemo poziciju prvog preteskog predmeta iz druge prodavnice
  pj = bisect.bisect_right(a2, [maks_masa - masa1, beskonacna_cena])
```

```
  if pj > 0: # ako postoji dovoljno lak predmet (pre pj su dovoljno laki)
    # biramo najvredniji od dovoljno lakih
    maks_cena = max(maks_cena, cena1 + maks_cena_do[pj])

print(maks_cena)
```

1.2 Други круг квалификација

Задатак: Кусур

Поставка: Андрија и Бојан су у кафићу попили по један (исти) сок. Андрија је дао a, а Бојан b динара, а добили су кусур од k динара. Како треба да поделе кусур да би једнако платили?

Улаз: Са стандардног улаза се учитавају три цела позитивна броја, a, b и k редом, сваки у посебном реду. Сва три броја су мања или једнака 1000 и таква да постоји целобројно решење.

Излаз: На стандардни излаз исписати два броја, сваки у посебном реду. Први број је део кусура који треба да добије Андрија, а други број је Бојанов део кусура.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
80	0	500	390
100	20	150	40
20		430	

Решење: Два сока су коштала $a + b - k$, а један сок $s = \frac{a+b-k}{2}$ динара. Према томе, Андрија треба да добије $a - s$ а Бојан $b - s$ динара.

```
a = int(input())
b = int(input())
kusur = int(input())
sok = (a + b - kusur) // 2
print(a - sok)
print(b - sok)
```

Задатак: Деца

Поставка: Сања има тачно онолико година колико и њено троје деце заједно. Дате су године Сање и њено двоје деце у произвољном редоследу. Одредити године трећег Сањиног детета.

Улаз: Са стандардног улаза се учитавају три цела позитивна броја који нису већи од 100, сваки у посебном реду. Један од бројева представља Сањине године, а остала два године двоје од њене деце.

Излаз: На стандардни излаз исписати један број, број година трећег Сањиног детета.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
17	14	15	15
43		20	
12		50	

Решење: Од три учитана броја прво треба одредити највећи - то је број година маме Сање. Збир година два детета можемо да одредимо тако што од збира сва три броја одузмемо мамин број година. Коначно, године трећег детета добијамо када од мамиког броја година одузмемо збир година два детета.

```
a = int(input())
b = int(input())
c = int(input())
mama = max(a, b, c)
prva_dva = a + b + c - mama
trece = mama - prva_dva
print(trece)
```

Задатак: Кутије

Поставка: Дате су две кутије за ципеле облика квадра. Свака кутија је дата својом дужином, ширином и висином. Испитати да ли се нека од ових кутија може убацити у другу. Једна кутија се може убацити у другу ако се може окренути тако да јој одговарајуће димензије буду строго мање од одговарајућих димензија друге кутије.

Улаз: У првом реду стандардног улаза налазе се три цела броја d_1, s_1 и v_1 раздвојени по једним размаком, дужина, ширина и висина прве кутије. У другом реду стандардног улаза налазе се три цела броја d_2, s_2 и v_2 такође раздвојени по једним размаком, дужина, ширина и висина друге кутије. Сви бројеви су позитивни и мањи од 1000.

Излаз: На стандардни излаз исписати `DA` ако се било која од кутија може убацити у другу, а `NE` иначе.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
5 9 2	DA	3 4 5	NE	3 4 5	DA
3 6 10		3 4 5		2 3 4	

Решење: Нека су $a_1 \leq b_1 \leq c_1$ димензије прве, а $a_2 \leq b_2 \leq c_2$ димензије друге кутије. Да не бисмо проверавали све могуће оријентације кутија, доказаћемо да прва кутија може да стане у другу ако и само ако је $a_1 \leq a_2, b_1 \leq b_2, c_1 \leq c_2$. Смер $\Leftarrow$ овог тврђења је очигледан, па је довољно доказати смер $\Rightarrow$. Претпоставимо да нису испуњена сва три услова $a_1 \leq a_2, b_1 \leq b_2, c_1 \leq c_2$.

Ако није $a_1 \leq a_2$, онда $c_1 \geq b_1 \geq a_1 \geq a_2$, па ни једна димензија прве кутије није мања од a_2 и прва кутија не може да стане у другу.

Ако није $b_1 \leq b_2$, онда $c_1 \geq b_1 \geq b_2 \geq a_2$, па пошто не могу обе ивице c_1 и b_1 да се поставе паралелно са c_2, то прва кутија ни у овом случају не може да стане у другу.

Ако није $c_1 \leq c_2$, онда $c_1 \geq c_2 \geq b_2 \geq a_2$, па c_1 није мање ни од једне стране друге кутије и поново прва кутија не може да стане у другу. Овим је доказ завршен.

Према томе, да бисмо проверили да ли прва кутија може да стане у другу, довољно је да након сортирања ивица сваке кутије проверимо да важе сва три услова $a_1 \leq a_2, b_1 \leq b_2, c_1 \leq c_2$ (ако прва кутија не може да стане у другу када су овако окренуте, онда не може да стане ни на који начин).

Наравно, из истог разлога важи да друга кутија може да стане у прву само ако је $a_2 \leq a_1, b_2 \leq b_1, c_2 \leq c_1$.

На питање из задатка дајемо потврдан одговор било да је $a_1 \leq a_2, b_1 \leq b_2, c_1 \leq c_2$ или $a_2 \leq a_1, b_2 \leq b_1, c_2 \leq c_1$, а ако не једна од од ових тројки услова није испуњена, одговор ће бити одречан.

```
a1, b1, c1 = sorted(map(int, input().split()))
a2, b2, c2 = sorted(map(int, input().split()))
if a1 < a2 and b1 < b2 and c1 < c2:
    print('DA')
elif a1 > a2 and b1 > b2 and c1 > c2:
    print('DA')
else:
    print('NE')
```

Задатак: Статистике35

Поставка: *Овај задатак је поновљен.* *Види текст задатка на страни 8.*

Решење: У овом задатку треба пребројати редове улаза у којима су бар три од пет бројева већи од 9. Да бисмо установили да ли неки ред улаза треба бројати, треба у сваком реду пребројати елементе који су већи од 9. То значи да ћемо у овом задатку применити технику пребројавања "на два нивоа": глобално бројимо редове који испуњавају услов, а у сваком реду бројимо елементе веће од 9.

Формираћемо двоструку петљу: спољна петља ће ићи по утакмицама (тј. редовима улаза), а унутрашња по категоријама које се прате на свакој утакмици. Бројач трипл-даблова иницијализујемо пре спољне петље, а бројач двоцифрених учинака пре унутрашње.

```
n = int(input())
br_tripl_dabl = 0
for utak in range(n):
    s = input().split()
    br_dvoc = 0
    for kateg in range(5):
        if int(s[kateg]) > 9:
            br_dvoc += 1
    if br_dvoc >= 3:
        br_tripl_dabl += 1
print(br_tripl_dabl)
```

Задатак: Трансформације

Поставка: Трансформацијом броја x зовемо замену броја x бројем $x \cdot x + 1$. Одредити троцифрени завршетак броја који се добија после n трансформација броја a.

Улаз: Са стандардног улаза се у првом реду уноси број a $(2 \leq a \leq 9)$, а у другом реду број n $(10 \leq n \leq 1000)$.

Излаз: На стандардни излаз исписати три цифре без размака, цифре којима се завршава број a^n.

Пример 1

Улаз	*Излаз*
5 11	802

Пример 2

Улаз	*Излаз*
2 403	026

Решење:

Анализа

Број a је трансформисањем веома брзо увећава, тако да већ после неколико трансформација даље рачунање или није тачно (због прекорачења) или је врло споро (због величине броја).

Пошто су нам потребне само последње три цифре трансформисаног броја, користићемо модуларну аритметику. Уместо са $a \cdot a + 1$, замењиваћемо a са $(a \cdot a + 1) mod 1000$. Нови резултат ће бити конгруентан старом по модулу 1000, па се зато при овој измени последње три цифре резулатата неће променити.

Након израчунавања трансформисаног броја по модулу 1000 треба још повести рачуна да се испишу и водеће нуле ако их има.

```
a = int(input())
n = int(input())
for i in range(n):
    a = (a*a + 1) % 1000

s = str(a)
while len(s) < 3:
    s = '0' + s

print(s)
```

Задатак: Рутер

Поставка: Дуж једне улице су равномерно распоређене зграде (растојање између сваке две суседне је једнако). За сваку зграду је познат број корисника које нови добављач интернета треба да повеже. Одредити у коју од зграда треба поставити рутер тако да би укупна дужина оптичких каблова којим се сваки од корисника повезује са рутером била минимална (рачунати само дужину каблова од зграде до зграде и занемарити дужине унутар зграда).

Улаз: У првом реду стандардног улаза налази се број n ($0 \leq n \leq 50000$), а у наредном n природних бројева раздвојених размацима који представљају број корисника у свакој од n зграда.

Излаз: На стандардни излаз исписати минималну дужину каблова.

Пример

Улаз	*Излаз*
`6` `3 5 1 6 2 4`	`30`

Решење: Наивно решење би подразумевало да се израчуна дужина каблова за сваку могућу позицију рутера и да се одабере најмањи. Да бисмо израчунали дужину каблова, ако је рутер у згради на позицији k, рачунамо заправо збир

$$\sum_{i=0}^{n-1} |k-i| \cdot a_i,$$

где је a_i број корисника у згради i. Тај тежински збир можемо израчунати у времену $O(n)$, па пошто се испитује n позиција, алгоритам би био сложености $O(n^2)$.

Много боље решење и линеарни алгоритам можемо добити ако применимо принцип инкременталности и избегнемо рачунање у сваком кораку из почетка. Размотримо како се дужина каблова мења када се рутер помера са зграде k на зграду $k+1$.

Дужину каблова за рутер у згради $k+1$ добијамо од дужине каблова за рутер у згради k тако што ту дужину увећамо за укупан број станара закључно са зградом k и умањимо је за укупан број станара почевши од зграде $k+1$. То је заправо интуитивно прилично јасно и без компликованог математичког извођења. Померањем рутера за дужину једне зграде надесно, сваком станару који живи закључно до зграде k дужина кабла се повећала за једно растојање између зграда, а свим станарима од зграде $k+1$ надесно се та дужина смањује за једно растојање између зграда.

Укупне бројеве станара пре и после дате зграде можемо такође рачунати инкрементално (при преласку на наредну зграду, први број се увећава, а други умањује за број станара текуће зграде).

Дакле, у програму можемо да памтимо три ствари: дужину каблова d_k ако је рутер на позицији k, укупан број станара pre_k пре зграде k и укупан број станара $posle_k$ од зграде k до краја. На почетку, када је $k = 0$, први број d_0 морамо експлицитно израчунати као $\sum_{i=1}^{n-1} i \cdot a_i$ (за то нам је потребно време $O(n)$), други број треба иницијализовати на нулу $pre_0 = 0$, а трећи на укупан број свих станара $posle_k = \sum_{i=0}^{n-1} a_i$ (и за то нам је потребно време $O(n)$). Затим за свако k од 1 до $n-1$ рачунамо $pre_k = pre_{k-1} + a_{k-1}$, $posle_k = posle_{k-1} + a_{k-1}$ и $d_k = d_{k-1} + pre_k + posle_k$.

Интересантно, укупну почетну дужину каблова можемо израчунати у линеарној сложености и без множења, тако што на збир додамо број станара у последњој згради, затим број станара у последње две зграде, затим број станара у последње три зграде и тако даље, све док не додамо број станара у свим зградама осим прве.

Пошто је и за једну и за другу фазу потребно време $O(n)$, то је уједно сложеност овог алгоритма.

```
n = int(input())
stanara = list(map(int, input().split()))

# krećemo od zgrade 0
# ukupna dužina kablova ako je ruter u tekućoj zgradi
duzina_kablova = 0
for i in range(n):
    duzina_kablova += stanara[i] * i

# broj stanara pre tekuće zgrade
stanara_pre = 0
# broj stanara od tekuće zgrade do kraja
stanara_posle = sum(stanara)


# minimalna duzina kablova
min_duzina_kablova = duzina_kablova

# obrađujemo sve zgrade od 1 do n-1
for k in range(1, n):
    # ažuriramo brojeve stanara
    stanara_pre += stanara[k-1]
    stanara_posle -= stanara[k-1]
    # ažuriramo duz
    duzina_kablova += stanara_pre - stanara_posle
    if duzina_kablova < min_duzina_kablova:
        min_duzina_kablova = duzina_kablova

print(min_duzina_kablova)
```

Задатак: Одбојка

Поставка: Данас се игра важан квалификациони меч између две локалне одбојкашке екипе, A и B. Бранко, који је велики љубитељ одбојке, због обавеза није могао да прати меч, па је видео само како се играчи поздрављају по завршетку утакмице. Тада се на екрану појавио последњи од оних досадних статистичких података који каже да је екипа A освојила укупно a поена, а екипа B укупно b поена током целе утакмице.

На основу ове информације Бранко сада покушава да закључи која екипа је победила. Наравно, Бранко зна да се меч играо на три добијена сета, а СВАКИ сет на 25 освојених поена (ни пети сет није скраћен), с тим да сет не може да се заврши са $25 : 24$. У случају резултата $24 : 24$ сет се наставља док једна од екипа не стекне предност од 2 поена.

Напишите програм који Бранку даје одговор на питање које га мучи.

Улаз: Са стандардног улаза се у првом реду уноси цео број a $(0 \leq a \leq 100)$, а у другом реду цео број b $(0 \leq b \leq 100)$, бројеви поена које су освојиле екипе током целе утакмице.

Излаз: На стандардни излаз исписати само једно слово.

- Ако је на основу укупног броја освојених поена извесно да је меч добила екипа A, исписати слово A.
- Ако је на основу укупног броја освојених поена извесно да је меч добила екипа B, исписати слово B.
- Ако није извесно која екипа је добила меч (то јест, ако је меч могла добити било која екипа), исписати слово N.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
`72`	`B`	`89`	`N`
`90`		`82`	

Решење: Размотримо најпре колико најмање поена треба да освоји екипа да би добила меч. Да би екипа добила меч, треба да добије три сета, а у сваком добијеном сету треба да освоји бар 25 поена. То значи да екипа која на крају меча има мање од 75 поена никако није могла да добије меч. Са друге стране, могуће је да екипа која освоји 75 поена добије меч, али то зависи и од броја поена противничке екипе.

Размотримо зато и колико екипа која изгуби три сета (а тиме и меч) може да надмаши победничку екипу у простом збиру поена. Екипа која је изгубила може да добије два сета са највише по 25 поена предности (резултатом 25:0), а да изгуби три сета са по најмање два поена заостатка. То значи да поражена екипа може да сакупи чак $2 \cdot 25 - 3 \cdot 2 = 50 - 6 = 44$ поена више од противника а да ипак изгуби меч. Екипа која има 45 или више поена предности у збиру, није могла да изгуби меч.

Из ове анализе произилази следећи закључак:

- ако је $a \geq 75$ и $b - a \leq 44$, екипа A је могла да добије меч.
- ако је $b \geq 75$ и $a - b \leq 44$, екипа B је могла да добије меч.

Пошто је меч завршен победом једне екипе, бар један од ових услова ће бити испуњен. Јасно, ако је испуњен само један од ових услова онда знамо победника, а ако су испуњена оба услова, онда је победник могла бити било која екипа.

```
a = int(input())
b = int(input())
mogla_a = (a > 74) and (a + 44 >= b)
mogla_b = (b > 74) and (b + 44 >= a)
if mogla_a and mogla_b:
    print('N')
elif mogla_a:
    print('A')
else: # mogla_b
    print('B')
```

Задатак: Хороскоп

Поставка: Јединствени матични број грађана (скраћено ЈМБГ) је низ од 13 цифара, који у нашој земљи свако добија још као сасвим мали. Прве две цифре овог низа представљају дан, а следеће две месец рођења. На пример, ако нечији ЈМБГ почиње

са 0904, он је рођен деветог априла, а ако почиње са 3112 он је рођен тридесет првог децембра.

Перица воли да се забавља читајући разне хороскопе, а у једном тренутку се запитао колико има људи рођених у сваком од хороскопских знакова, за које би требало да важи оно што пише у хороскопу. Перица је некако успео да дође до повеће листе матичних бројева, а зна и почетне и завршне датуме за сваки знак:

- ОВАН – од 21. марта до 20. априла
- БИК – од 21. априла до 21. маја
- БЛИЗАНЦИ – од 22. маја до 21. јуна
- РАК – од 22. јуна до 22. јула
- ЛАВ – од 23. јула до 22. августа
- ДЕВИЦА – од 23. августа до 22. септембра
- ВАГА – од 23. септембра до 23.октобра
- ШКОРПИЈА – од 24. октобра до 22. новембра
- СТРЕЛАЦ – од 23. новембра до 21. децембра
- ЈАРАЦ – од 22. децембра до 20. јануара
- ВОДОЛИЈА – од 21. јануара до 19. фебруара
- РИБЕ – од 20. фебруара до 20. марта

Перица сада жели да преброји рођења у сваком од хороскопских знакова. Напишите програм који помаже Перици у бројању.

Улаз: На стандардном улазу се у првом реду налази број N $(1 \leq N \leq 200)$, број матичних бројева које је Перица набавио. У сваком од N наредних редова је по један матични број (13 цифара без размака).

Излаз: На стандардни излаз исписати 12 целих бројева, сваки у посебном реду. Први број је број особа рођени у знаку овна, други - у знаку бика, итд. Последњи, дванаести број је број особа рођених у знаку рибе.

Пример 1

Улаз	*Излаз*
5	2
1504279718463	0
0412622712918	0
1903326718649	0
0710262713307	0
0104646719372	0
	1
	0
	1
	0
	0
	1

Решење: Потребно је за сваки прочитани матични број одредити датум рођења особе, а затим за тај датум рођења одредити одговарајући хороскопски знак. Ово можемо да урадимо на различите начине. Један начин је да прво направимо списак уређених тројки, које се састоје од индекса хороскопског знака, и дана и месеца када се тај знак

завршава. Тада је довољно да у петљи за текући ЈМБГ издвојимо месец и дан рођења особе и поредимо га редом са завршним датумима хороскопских знакова, док датум рођења не буде мањи од краја неког знака (пошто је број знакова веома мали, можемо слободно употребљавати линеарну претрагу). Када се то догоди, знамо којем знаку припада текући датум рођења. Датуме можемо представити било уређеним паровима (месец, дан), било их кодирати целим бројевима по принципу 100 пута месец + дан.

Да бисмо довршили задатак, можемо да уведемо листу од 12 бројача, тако да сви почињу од 0, а када за текућу особу установимо хороскопски знак, увећавамо за 1 бројач који одговара том знаку.

```
OVAN, BIK, BLIZANCI, RAK, LAV, DEVICA, VAGA, \
SKORPIJA, STRELAC, JARAC, VODOLIJA, RIBE = range(12)

poslednji_dan_znaka = (
    (JARAC, 120),       #  до 20. јануара
    (VODOLIJA, 219),    #  до 19. фебруара
    (RIBE, 320),        #  до 20. марта
    (OVAN, 420),        #  до 20. априла
    (BIK, 521),         #  до 21. маја
    (BLIZANCI, 621),    #  до 21. јуна
    (RAK, 722),         #  до 22. јула
    (LAV, 822),         #  до 22. августа
    (DEVICA, 922),      #  до 22. септембра
    (VAGA, 1023),       #  до 23.октобра
    (SKORPIJA, 1122),   #  до 22. новембра
    (STRELAC, 1221),    #  до 21. децембра
    (JARAC, 1231)       #  до 31. децембра
)

n = int(input())
br = [0] * 12
for i in range(n):
    jmbg = input()
    mesec_dan = int(jmbg[2:4] + jmbg[0:2])
    for znak, kraj in poslednji_dan_znaka:
        if mesec_dan <= kraj:
            br[znak] += 1
            break

for x in br:
    print(x)
```

Задатак: Аритмогриф

Поставка: При решавању једне познате врсте аритметичких ребуса потребно је иста слова заменити истим цифрама, а различита слова различитим цифрама, тако да се добије тачна једнакост. На пример, за ребус $SNEG + KRUG = SPORT$ једно

од решења је $1937 + 8627 == 10564$. Можемо се уверити да је наведена једнакост заиста тачна, али то није тема овог задатка. Овде нас интересује други услов, а то је да ли је замена слова цифрама правилно изведена. На пример, понуђено "решење" $4832 + 5962 == 10794$ није исправно (иако је, узгред речено, једнакост тачна) зато што је слово S на једном месту замењено цифтом 4, а на другом цифром 1. Такође, цифра 4 одговара двама различитим словима (S и T).

Написати програм који проверава да ли је у овом типу ребуса замена слова цифрама правилно изведена.

Улаз: Са стандардног улаза се у првом реду уноси стринг, који представља описани аритметички ребус. Стринг може да садржи велика слова енглеске абецеде, размаке, знаке математичких операција и и знак једнакости. Укупна дужина стринга није већа од 100 а број различитих слова која се појављују у стрингу није већи од 10.

У другом реду се уноси стринг који представља понуђено решење ребуса из првог реда. Други стринг се од првог разликује искључиво по томе што је свако слово замењено неком цифром.

Излаз: На стандардни излаз исписати "da" ако је замена правилна (без обзира на то да ли је једнакост тачна), а "ne" ако замена није правилна.

Пример 1

Улаз	*Излаз*
`SNEG + KRUG = SPORT`	`da`
`2341 + 8751 = 20679`	

Пример 2

Улаз	*Излаз*
`SNEG + KRUG = SPORT`	`ne`
`4832 + 5962 = 10794`	

Решење: Задатак ћемо свакако решавати пролазећи истовремено кроз оба дата стринга. Пошто пролазећи кроз дате стрингове словима ребуса придружујемо цифре понуђеног решења, природно је да формирамо речник у коме памтимо та придруживања.

Када у петљи дохватимо текуће слово и текућу цифру, интересује нас да ли је ово прво појављивање тог слова.

Ако текуће слово није у речнику, ово је његово прво појављивање. Сада је важно да ли се и текућа цифра први пут појављује. Да бисмо то установили, користићемо скуп искоришћених цифара. Ако цифра није у скупу, то је њено прво појављивање и можемо у речнику да текућем слову придружимо текућу цифру, а у скуп искоришћених цифара да додамо текућу цифру. Ако је цифра већ била у скупу, то значи да понуђено решење не ваља.

Ако је текуће слово већ у речнику, онда само проверавамо да је њему придружена вредност из речника једнака текућој цифри. Ако није једнака, онда то поново значи да понуђено решење не ваља.

На крају, ако нисмо констатовали да понуђено решење не ваља, онда је замена слова цифрама исправна.

```
rebus = input()
resenje = input()

zamene = {}
iskoriscene_cifre = set()
```

```
dobra_zamena = True

for i in range(len(rebus)):
    cifra = zamene.get(rebus[i], None)
    if cifra == None:
        if resenje[i] in iskoriscene_cifre:
            dobra_zamena = False
            break
        zamene[rebus[i]] = resenje[i]
        iskoriscene_cifre.add(resenje[i])
    elif cifra != resenje[i]:
        dobra_zamena = False
        break

if dobra_zamena:
    print('da')
else:
    print('ne')
```

Друга могућност је да у речнику за свако слово памтимо све цифре које су том слову придружене. Дакле кључеви тог речника су слова, а вредности су скупови цифара. Након формирања речника проверавамо да су сви скупови цифара једночлани (ако нису, понуђено решење не ваља). Овом провером смо потврдили да је сваком слову придружена тачно једна цифра, али не снамо да ли су различитим словима придружене различите цифре. Да бисмо то проверили, можемо да претходну проверу ставимо у функцију, а затим да функцију позовемо још једном, али са датим стринговима (ребус и понуђено решење) у замењеним улогама. Тако за сваку цифру добијамо скуп слова којима је та цифра придружена. Ако су и сви ови скупови једночлани, тиме потврђујемо и да је свака цифра придружена тачно једном слову, што значи да је замена слова цифрама исправна.

```
def dobra_zamena(a, b):
    zamene = {}
    for i in range(len(a)):
        s = zamene.get(a[i], set())
        s = s | {b[i]}
        zamene[a[i]] = s

    dobra = True
    for i in range(len(a)):
        if len(zamene[a[i]]) != 1:
            dobra = False
    return dobra

rebus = input()
resenje = input()
if dobra_zamena(rebus, resenje) and dobra_zamena(resenje, rebus):
    print('da')
```

```
else:
    print('ne')
```

Задатак: Две полице

Поставка: У продавници се продају две врсте полица чије су цене и дужине познате. Желимо да поставимо полице дуж једног зида тако да попунимо што већи део дужине зида. При том, ако се иста дужина зида може попунити на више начина, бирамо јефтинију варијанту. Напиши програм који одређује највећу могућу дужину дела зида која се може попунити полицама и најмању цену по којој се то може урадити.

Улаз: У првом реду стандардног улаза се налази број z ($1 \leq z \leq 10^6$) који представља дужину зида. У два наредна реда се налазе описи полица: дужина (природан број између 1 и 10^6) и цена (природан број између 1 и 1000).

Излаз: На стандардни излаз исписати највећу могућу дужину дела зида који се може попунити полицама и најмању могућу цену, раздвојене једним размаком.

Пример

Улаз	*Излаз*
24 3 10 5 8	24 54

Објашњење

Цео зид се може попунити са 3 полице дужине 3 и 3 полице дужине 5 или са 8 полица дужине 3. Цена у првом случају је 54, а у другом случају је 80.

Решење: Задатак решавамо грубом силом тј. исцрпном претрагом свих могућности. Крећемо од варијанте у којој немамо полица типа 1 и затим додајемо једну по једну полицу типа 1, све док је то могуће (док је дужина дела зида попуњеног полицама типа 1 мања или једнака од укупне дужине зида). За сваки број полица типа 1 који тренутно разматрамо, израчунавамо највећи могући број полица типа 2 који се може поставити (њега одређујемо целобројним дељењем дужине дела зида иза постављених првих полица дужином друге полице). Након тога израчунавамо преосталу дужину зида иза свих полица и цену и ако је потребно ажурирамо минимум.

```
zid = int(input())
s = input().split()
polica1, cena1 = int(s[0]), int(s[1])
s = input().split()
polica2, cena2 = int(s[0]), int(s[1])

min_ostalo = zid
min_cena = 0

# isprobavamo sve mogucnosti
# dodajemo jednu po jednu policu tipa 1 dok god je to moguce
broj1 = 0
while broj1 * polica1 <= zid:
```

```
    # racunamo najveci moguci broj preostalih polica tipa 2
    broj2 = (zid - broj1*polica1) // polica2
    ostalo = zid - broj1*polica1 - broj2*polica2
    # cena svih polica
    cena = broj1*cena1 + broj2*cena2
    # azuriramo minimum ako je potrebno
    if (ostalo < min_ostalo or
            (ostalo == min_ostalo and cena < min_cena)):
        min_ostalo = ostalo
        min_cena = cena
    broj1 += 1

print(zid - min_ostalo, min_cena)
```

Задатак: Пуно фигурица

Поставка: Друштвену игру игра k играча и сваки играч игру почиње са по k фигура, при чему све фигуре једног играча морају бити исте јачине, док су јачине фигура различитих играча различите. Фигуре су доступне у неограниченим количинама, при чему је познат низ јачина доступних фигура. Напиши програм који одређује највећи број играча k који могу играти игру тако да разлика између укупне јачине свих фигура било која два играча не пређе задату границу.

Улаз: Са стандардног улаза се учитава број n ($1 \leq n \leq 10^5$), а затим у наредном реду n различитих расположивих јачина фигура (природни бројеви између 1 и 10^5). Наредни ред садржи границу (природни број између 1 и 10^{12}).

Излаз: На стандардни излаз исписати два броја раздвојена размаком – број k и најмању разлику укупних јачина фигура када игра k играча.

Пример

Улаз	*Излаз*
5 5 4 2 7 3 15	4 12

Објашњење

Ако играчи узму по четири фигуре јачине 5, 4, 2 и 3 највећа разлика јачина фигура играча биће $4 \cdot 5 - 4 \cdot 2 = 12$. Ако би играло 5 играча, морали би да узму и фигуре јачине 7 и највећа разлика би била $5 \cdot 7 - 5 \cdot 2 = 25$, што је веће од дозвољене границе.

Решење:

Наивно решење

Задатак се може решити тако што се за свако k између 2 и n провери да ли је могуће игру одиграти са k играча.

Централно питање је како за дато k изабрати k играча, тако да разлика између укупне јачине фигура најјачег и најслабијег међу њима буде што мања. Ефикасан алгоритам се може направити ако се низ јачина фигура претходно сортира. Играчи тада треба да

узимају фигуре чије су јачине узастопних k елемената низа (ако би неки играч заменио фигуре, добила би се већа разлика између најјачег и најслабијег играча). Зато је довољно проверити све узастопне k-точлане поднизове низа (којих има $n-k$), одузети последњи од првог члана и помножити резултат са k (јер сваки играч узима по k фигура).

Пошто се са порастом броја играча разлика може само повећати, претрагу можемо прекинути када први пут нађемо вредност k такву да игру не могу играти k играча. Сложеност таквог алгоритма, суштински заснованог на линеарног претрази је $O(n^2)$.

```
def najmanja_razlika_k(a, k):
    min = float('inf')
    for i in range(len(a) + 1 - k):
        if a[i+k-1] - a[i] < min:
            min = a[i+k-1] - a[i]
    return min


n = int(input())
a = sorted(map(int, input().split()))
granica = int(input())
k = 2
while k <= n and k * najmanja_razlika_k(a, k) <= granica:
    k += 1

print(k-1, (k-1) * najmanja_razlika_k(a, (k-1)))
```

Бинарна претрага

Брже решење можемо добити ако оптималну вредност k тражимо бинарном претрагом (то је могуће, јер вредности разлика расту са порастом k). Сложеност таквог алгоритма је $O(n \log n)$.

```
def najmanja_razlika_k(a, k):
    min = float('inf')
    for i in range(len(a) + 1 - k):
        if a[i+k-1] - a[i] < min:
            min = a[i+k-1] - a[i]
    return min


n = int(input())
a = sorted(map(int, input().split()))
granica = int(input())
lK, dK = 2, n
while lK <= dK:
    k = lK + (dK - lK) // 2
    if k * najmanja_razlika_k(a, k) <= granica:
      lK = k+1
    else:
```

```
        dK = k-1

print(dK, dK * najmanja_razlika_k(a, dK))
```

Два показивача

Још једно брзо решење можемо да добијемо техником два показивача. За сваки леви крај сегмента одређујемо највећу могућу вредност десног краја која не прелази границу. Након сортирања користимо сегмент $[poc, kraj]$ који представља изабране фигуре. Овај сегмент је на почетку $[0, 0]$. Ако је разлика између укупне јачине најјачих и најслабијих фигура из сегмента довољно мала, продужавамо сегмент надесно (повећавамо $kraj$), а ако је већа од дозвољене, онда скраћујемо сегмент слева (повећавамо poc). Сигурни смо да након повећања левог краја, десни крај не морамо смањивати (размисли зашто). Успут памтимо највећу дужину сегмента и оптималну разлику при тој дужини сегмента. Када прођемо цео низ јачина фигура, исписујемо коначне вредности највеће дужине сегмента и оптималне разлике. Сложеност оваквог решења је $O(n \log n)$ у почетној фази сортирања, а након тога, у другој фази је линеарна $O(n)$.

```
n = int(input())
a = sorted(map(int, input().split()))
granica = int(input())

poc, kraj, opt_k, opt_razlika = 0, 0, 1, 0
while kraj < n:
    k = kraj-poc+1
    razlika = k*(a[kraj]-a[poc])
    if razlika > granica:
        poc += 1
    else:
        if k > opt_k:
            opt_k = k
            opt_razlika = razlika
        elif k == opt_k:
            opt_razlika = min(opt_razlika, razlika)
        kraj += 1
print(opt_k, opt_razlika)
```

1.3 Трећи круг квалификација

Задатак: Клацкалица

Поставка: Ђаци прваци су после школе свратили у парк да се клацкају. Током игре покушавали су и да претегну једно друго. Испоставило се да је Милица са школском торбом тешка отприлике као Радоје без торбе, а Радоје са Милицином торбом као Бојан без торбе. Можете ли да одредите приближно Милицину тежину ако су познате тежине Радоја и Бојана?

Улаз: Са стандардног улаза се у првом реду уноси цео број R $(25 \leq R \leq 50)$, а у

другом реду цео број B $(25 \leq B \leq 50)$, Радојева и Бојанова тежина редом.

Излаз: На стандардни излаз исписати један цео број, Милицину приближну тежину.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
35 42	28	39 41	37

Решење: Тежину торбе можемо да одредимо као $T = B - R$. Након тога лако добијамо Милицину тежину као $B - T$.

```
radoje = int(input())
bojan = int(input())
torba = bojan - radoje
milica = radoje - torba
print(milica)
```

Задатак: Приземље

Поставка: У једној згради лифт се веома ретко користи. Кад год уђете у приземље те зграде, лифт је слободан и стоји на неком спрату, а често управо у приземљу.

Пензионер Божур се из хобија бави електроником, па га је интересовало да ли би се исплатило да се лифт подеси тако да чим је слободан, да се врати у приземље. Зато је током прошлог месеца аутоматски прикупио податке, а сада хоће да упосли унука Уроша, програмера, да одреди колико се у приземљу просечно чека на лифт. Урошу би помоћ добро дошла.

Улаз: Са стандардног улаза се у првом реду уноси цео број N $(1 \leq N \leq 1000)$, број позива лифта из приземља. У наредних N редова је по један цео број S $(0 \leq S \leq 15)$, број спратова које је празан лифт прешао по позиву из приземља (да би стигао у приземље).

Излаз: На стандардни излаз исписати један цео број, просечан број спратова које лифт пређе да би празан дошао у приземље на позив, заокружен навише.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
5 4 0 0 2 0	2	2 0 0	0

Решење: Да се добије средња вредност, потребно је сабрати свих N вредности S, добијени збир поделити са N. На крају треба још заокружити добијену вредност навише, што се може постићи стандардном функцијом из библиотеке или коришћењем везе да је $\lceil \frac{a}{b} \rceil = \frac{a+b-1}{b}$.

```
import math
n = int(input())
```

```
zbir = 0
for i in range(n):
    s = int(input())
    zbir += s

print(math.ceil(zbir / n))
```

Задатак: Брана

Поставка: Даброви граде брану да би подигли ниво воде изнад улаза у њихова склоништа. Дабра Животу боли зуб, па он данас уместо да ради, процењује колико још има посла. Живота је прво измерио висину сваког улаза у склониште, а затим за неколико околних стабала оценио колико би она подигла ниво воде ако би била уграђена у брану.

Живота жели да за сваки могући ниво воде одреди колико би улаза у склоништа остало на сувом (те улазе би требало затрпати). Улаз је на сувом ако је његова висина строго већа од нивоа воде.

Улаз: Са стандардног улаза учитава се број улаза n ($0 \leq n \leq 50000$), а затим и висине улаза (цели бројеви), задати у сортираном редоследу од највишег до најнижег. Након тога се учитава број m ($1 \leq m \leq 50000$) који представља број могућих нивоа воде, а затим и m бројева који представљају нивое за које треба одговорити колико би улаза у склоништа остало на сувом, када би вода достигла тај ниво. Сваки број се налази у посебном реду.

Излаз: На стандардни излаз за сваки ниво воде исписати тражени број непотопљених улаза, сваки у посебном реду.

Пример 1

Улаз	*Излаз*
5	0
79	4
63	3
63	5
46	
13	
4	
85	
40	
60	
0	

Пример 2

Улаз	*Излаз*
4	3
30	0
29	1
29	0
28	4
5	
28	
30	
29	
31	
27	

Решење: У задатку је потребно ефикасно одредити број елемената сортираног низа који су већи од датог броја. Ако нађемо позицију првог елемента који је већи од датог броја, тада број таквих елемената можемо одредити тако што израчунамо разлику између укупног броја чланова низа и те позиције.

Најједноставнији начин да нађемо позицију првог елемента који је већи од датог броја је да применимо линеарну претрагу и да редом проверавамо један по један елемент све док не дођемо или до краја низа или до тражене позиције. Сложеност овакве претраге

је $O(n)$, па би укупна сложеност решења била $O(m \cdot n)$, што је превише имајући у виду ограничења дата у задатку.

```
n = int(input())
visine_ulaza = [int(input()) for i in range(n)]

m = int(input())
for i in range(m):
    nivo_vode = int(input())
    broj = 0
    for visina in visine_ulaza:
        if visina > nivo_vode:
        broj++
    print(broj)
```

Позиција се ефикасно може пронаћи применом алгоритма бинарне претраге. Најједноставније је употребити библиотечку имплементацију.

```
n = int(input())
visine_ulaza = [int(input()) for i in range(n)]
visine_ulaza.reverse()

rez = [] # rezultate smeštamo u niz zbog kasnijeg bržeg ispisa
m = int(input())
for i in range(m):
    nivo_vode = int(input())
    rez.append(n - bisect.bisect_right(visine_ulaza, nivo_vode))
print(*rez, sep='\n')
```

Наравно, позицију првог елемента који је већи од датог можемо пронаћи и ручно имплементираном бинарном претрагом.

```
def prvi_veci(a, x):
    l = 0; d = len(a) - 1;
    while l <= d:
        s = l + (d - l) // 2
        if a[s] <= x:
            l = s + 1
        else:
            d = s - 1
    return d + 1

n = int(input())
visine_ulaza = [int(input()) for i in range(n)]

rez = [] # rezultate smeštamo u niz zbog kasnijeg bržeg ispisa
m = int(input())
for i in range(m):
    nivo_vode = int(input())
    rez.append(n - prvi_veci(visine_ulaza, nivo_vode))
```

```
print(*rez, "\n")
```

Задатак: Највећи број од датих цифара

Поставка: За дати број исписати највећи број који се пише истим цифрама.

Улаз: Са стандардног улаза се у првом реду уноси број N $(1 \leq N \leq 10^{30})$.

Излаз: На стандардни излаз исписати тражени број.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
90123	93210	777	777

Решење: Задатак једноставно решавамо тако што сортирамо цифре опадајуће. С обзиром на велики број цифара, најбоље је резултат представити ниском карактера.

```
a = input()
a = sorted(a, reverse = True)
for c in a:
    print(c, end='')
print()
```

Резултат је могуће представити и низом цифара.

```
a = list(input())
a.sort(reverse = True)
print(*a, sep ='')
```

Интересантно, задатак се може решити у једном реду (мада то решење не препоручујемо, јер није разумљиво колико и претходна).

```
print("".join(sorted(input(), reverse=True)))
```

Задатак: Ризико

Поставка: Током игре ризико нападачки и одбрамени играч улазе у битке бацајући коцкице. Сваки од њих има 3 коцкице а на основу тренутног стања игре одређује се колико од њих сме да баци. Када се баце коцкице, упоређују се најјача коцкица нападача (она са највећим бројем) са најачом коцкицом одбрамбеног играча, затим друга по јачини са другом по јачини и најмања са најмањом. Ако је неки играч бацио мање коцкица, тада се најслабије коцкице оног другог играча занемарују (онај ко је бацао више коцкица је у предности). Када се две коцкице упоређују, ако нападач има строго јачу коцкицу (коцикцу на којој је број строго већи) одбрана губи један поен, док у супротном напад губи један поен (тиме је одбрамбени играч у благој предности). Напиши програм који на основу резултата бацања коцкица одређује број поена које губе један и други играч.

Улаз: Први ред стандардног улаза садржи број бачених коцкица нападача (1, 2 или 3), а наредни ред садржи бројеве (од 1 до 6) на коцкицама које је нападач бацио. Наредни ред стандардног улаза садржи број бачених коцкица одбрамбеног играча (1, 2 или 3), а наредни ред садржи бројеве (од 1 до 6) на коцкицама које је одбрамбени играч бацио.

Излаз: На стандардни излаз исписати број поена које губи нападач и број поена које губи одбрамбени играч, раздвојене једним размаком.

Пример 1

Улаз	*Излаз*
3 5 6 3 3 5 3 4	1 2

Објашњење

Нападач 6 се пореди са одбраном 5, нападач 5 са одбраном 4 и нападач 3 са одбраном 3. У прве две борбе побеђује нападач, па одбрана губи два поена, док у трећој побеђује одбрана (јер нападач није добио строго већи број), па нападач губи 1 поен.

Пример 2

Улаз

```
3
1 4 6
2
5 5
```

Излаз

```
1 1
```

Објашњење

Нападач 6 се пореди са одбраном 5, нападач 4 са одбраном 5, док се нападач 1 занемарује. У првој борби побеђује нападач, па одбрана губи поен, док у другој побеђује одбрана, па нападач губи поен.

Решење: Задатак решавамо тако што сортирамо низ коцкица нападача и низ коцкица одбрамбеног играча опадајуће и онда редом поредимо једну по једну коцкицу од почетка низа све док се неки од низова не исцрпи и на одговарајући начин ажурирамо бројаче изгубљених фигура за играча који напада и играча који се брани.

```
n_napad = int(input())
napad = sorted(map(int, input().split()), reverse = True)
n_odbrana = int(input())
odbrana = sorted(map(int, input().split()), reverse = True)
br_kockica = min(n_napad, n_odbrana)
gubi_napad, gubi_odbrana = 0, 0
for i in range(br_kockica):
    if napad[i] > odbrana[i]:
        gubi_odbrana += 1
    else:
        gubi_napad += 1
print(gubi_napad, gubi_odbrana)
```

Задатак: Атп победник

Поставка: Током године играју се многи тениски турнири на којима тенисери освајају поене. Поени се сабирају и на крају године објављује се завршна листа на којој су тенисери рангирани на основу укупног броја поена током те године. Напиши програм који на основу резултата свих турнира одређује најбољег тенисера и његове поене (претпоставити да ће победник имати строго већи број поена од свих осталих).

Улаз: Са стандардног улаза се уноси број n, а затим у наредних n редова подаци о освојеним поенима тенисера: презиме тенисера и затим број освојених поена.

Излаз: На стандардни излаз исписати презиме и укупан број поена победника.

Пример 1

Улаз
```
9
Djokovic 1000
Nadal 800
Federer 600
Nadal 1000
Federer 800
Djokovic 600
Djokovic 1000
Cicipas 800
Nadal 600
```

Излаз
```
Djokovic 2600
```

Пример 2

Улаз
```
6
Zverev 1000
Cicipas 800
Medvedev 700
Thiem 1000
Cicipas 800
Medvedev 600
```

Излаз
```
Cicipas 1600
```

Решење: У првој фази програма је потребно да израчунамо укупан број поена за сваког играча. Податке можемо чувати у мапи која пресликава име играча у његов број поена. Након тога одређујемо највећу вредност у мапи (што можемо урадити било применом библиотечке функције, било проласком кроз све елементе мапе и ажурирањем максимума).

```
n = int(input())
tabela = dict()
for i in range(n):
    ime, poeni = input().split()
    tabela[ime] = tabela.get(ime, 0) + int(poeni)

pobednik, max_poena = '', -1
for ime, poeni in tabela.items():
    if max_poena < poeni:
        pobednik = ime
        max_poena = poeni

print(pobednik, max_poena)
```

Задатак: Атп листа

Поставка: Током године играју се многи тениски турнири на којима тенисери освајају поене. Поени се сабирају и на крају године објављује се завршна листа на којој су тенисери рангирани на основу укупног броја поена током те године. Напиши програм

који на основу резултата свих турнира одређује k најбољих тенисера и њихове поене (претпоставити да ће сви тенисери имати различит број поена).

Улаз: Са стандардног улаза се учитава број турнира n, а затим подаци о освојеним поенима на тим турнирима. За сваки турнир се учитава број m који представља број тенисера који су освајали поене, и након тога m линија које садрже податке о освојеним поенима тенисера на том турниру (презиме тенисера које има највише 50 карактера и број поена између 10 и 2000). На крају се уноси број k који одређује колико најбољих тенисера треба одредити.

Излаз: На стандардни излаз исписати презимена и укупан освојени број поена за k најбољих тенисера у опадајућем редоследу укупног освојеног броја поена.

Пример

Улаз	*Излаз*
3 3 Djokovic 1000 Nadal 800 Federer 600 3 Nadal 1000 Federer 800 Djokovic 600 3 Djokovic 1000 Cicipas 800 Nadal 600 3	Djokovic 2600 Nadal 2400 Federer 1400

Решење: У првој фази програма је потребно да израчунамо укупан број поена за сваког играча. Податке можемо чувати у мапи која пресликава име играча у његов број поена. Након тога треба одредити k највећих вредности у мапи. Један начин је да се сви елементи из мапе прекопирају у низ, да се низ сортира опадајуће и да се прочита првих k елемената тако сортираног низа.

```
br_turnira = int(input())
tabela = dict()
for t in range(br_turnira):
    br_osvajaca = int(input())
    for osv in range(br_osvajaca):
        ime, poeni = input().split()
        tabela[ime] = tabela.get(ime, 0) + int(poeni)
br_najboljih = int(input())

tabela_lista = [(poeni, ime) for ime, poeni in tabela.items()]
tabela_lista.sort(reverse = True)
for i in range(br_najboljih):
    print(tabela_lista[i][1], tabela_lista[i][0])
```

Задатак: Слаткиши за сав новац

Поставка: Дуж једне улице деца продају слаткише. Јована има G динара и жели да их потроши тако што ће кренути да се шета дуж улице и од сваког детета, редом, ће купити тачно један слаткиш (ни једно дете неће прескочити). Ако су познате цене свих слаткиша и ако је позната позиција са које Јована креће у куповину, одредити број слаткиша које ће Јована на тај начин купити.

Улаз: Са стандардног улаза се уноси број деце који продају слаткише n ($1 \leq n \leq 5 \cdot 10^4$), а затим и низ који садржи цене слаткиша (позитивни природни бројеви мањи од 100). Након тога се уноси број упита k ($1 \leq k \leq 5 \cdot 10^4$), а затим у k наредних редова упити који садрже позицију првог детета које ће Јована обићи (позиције се броје од нуле), а затим број динара које Јована има.

Излаз: На стандардни излаз за сваки упит у посебном реду исписати број слаткиша које ће Јована купити.

Пример

Улаз	*Излаз*
7	0
3 5 1 2 3 1 4	2
4	5
3 1	3
2 5	
2 13	
0 10	

Објашњење

У првом упиту Јована има један динар, а креће од детета које продаје слаткиш за 2 динара, па не може да купи ни један слаткиш.

У другом упиту Јована има пет динара и купује слаткише који коштају 1 и 2 динара.

У трећем упиту Јована има 13 динара и купује слаткише који коштају 1, 2, 3, 1 и 4 динара.

У четвртом упиту Јована има 10 динара и купује слаткие који коштају 3, 5 и 1 динар.

Решење: Решење грубом силом подразумева да се за сваку почетну позицију са које Јована креће редом сабирају цене колача, све док је збир мањи од новца које Јована има. Сложеност таквог решења је $O(n \cdot q)$, што је с обзиром на ограничења недопустиво споро.

```
n = int(input())
cene = list(map(int, input().split()))
Q = int(input())
for q in range(Q):
     p, novac = map(int, input().split())
     kupljeno = 0
     while p < n and novac >= cene[p]:
         novac -= cene[p]
```

```
        kupljeno += 1
        p += 1
    print(kupljeno)
```

Ефикасније решење можемо постићи ако уместо низа цена колача одржавамо низ збирова префикса низа цена (чувамо низ z такав да је $z_k = 0$ и $z_k = \sum_{i=0}^{k-1} c_k$). Пошто су цене позитивне, низ збирова префикса је сортиран растуће и може се претраживати бинарном претрагом. Ако Јована креће са позиције p а последњи колач купује на позицији p', тада се цена коју ће платити може израчунати као $z_{p'+1} - zp$. Дакле, бинарном претрагом ћемо наћи највећу вредност p' такву да је $z_{p'+1} - z_p \leq N$ (где је N износ новца који Јована има).

Низ префиксних збирова можемо израчунати у сложености $O(n)$ приликом учитавања низа цена, након чега вршимо q бинарних претрага, свака је сложености $O(\log n)$. Дакле, укупна сложеност је $O(n + q \log n)$.

Нагласимо да је ово један од задатака у ком се одговара на упите тј. у ком се наизменично читају подаци са стандардног улаза и исписују на стандардни излаз. Ако се не обрати посебна пажња, пуно времена може отићи на синхронизацију између улаза и излаза.

У језику Python најбоље решење је да све резултате сместимо у низ и да на крају програма испишемо тај низ, али не у петљи, већ само једним позивом функције `print`.

```
import bisect

n = int(input())
cene = list(map(int, input().split()))
zbir_prefiksa_cena = [0] * (n+1)
for i in range(n):
    zbir_prefiksa_cena[i+1] = zbir_prefiksa_cena[i] + cene[i]

# da bismo ubrzali ispis, smeštamo rezultate u niz
rez = []
q = int(input())
for i in range(q):
    p, novac = map(int, input().split())
    k = bisect.bisect_right(zbir_prefiksa_cena, zbir_prefiksa_cena[p] + novac, p)
    rez.append(k - p - 1)
# ispisujemo niz jednom naredbom
print(*rez, sep="\n")
```

Задатак: Мали поштар

Поставка: Јовица зарађује џепарац тако што доноси пакете својим комшијама. Поделу креће од своје куће и потребно је да пакете разнесе у друге куће распоређене дуж улице и да се врати назад у своју кућу. За сваку кућу познато је растојање од почетка улице. Најкраћи пут би прешао ако би пакете сложио тако да их редом дели комшијама дуж улице. Пошто Јовица жели да буде у доброј физичкој форми, он током поделе

пакета трчи и жели да пакете уреди тако да пређе што већи пут. Напиши програм који одређује највећи пут који може да пређе.

Улаз: Са стандардног улаза се уноси број кућа у које треба донети пакете (међу њима се налази и Јовицина кућа), а затим и растојања тих кућа од почетка улице.

Излаз: На стандардни излаз исписати највеће растојање које Јовица може прећи током поделе пакета.

Пример 1

Улаз	*Излаз*
5 7 3 6 10 2	24

Објашњење

Постоји више начина да Јовица претрчи 24 дужне јединице. На пример, ако је његова кућа на позицији 3, он може да редом обилази куће 3, 7, 2, 10, 6, 3.

Пример 2

Улаз

```
7
3 5 11 4 2 17 9
```

Излаз

```
56
```

Решење: Решење грубом силом подразумева да се провере сви могући редоследи обиласка n кућа, тј. свих $n!$ пермутација датих бројева и да се утврди која од њих даје највећу могућу вредност пређеног пута. Овај алгоритам је изразито неефикасан и може се применити само на веома, веома мале улазе (практично, само за $n \leq 10$ програм може да реши задатак у датом временском ограничењу).

```
import itertools

def zbirApsRazlika(a, n):
    zbir = 0
    for k in range(1, n):
        zbir += abs(a[k] - a[k-1])
    zbir += abs(a[n-1] - a[0])
    return zbir;

n = int(input())
a = map(int, input().split())

maks = 0
for p in itertools.permutations(a):
    z = zbirApsRazlika(p, n)
    maks = max(maks, z)
```

```
print(maks)
```

Интуитивно нам је јасно да ће се пуно претрчати ако се стално трчи са једног на други крај улице. Један грамзиви приступ је да се прво обиђе крајња лева кућа, па затим крајња десна, па друга слева, па друга здесна и тако "цик-цак". Докажимо да је ово грамзиво решење исправно.

За дати распоред $x_0, \dots x_{n-1}$ одређујемо суму апсолутних вредности разлика елемената тј. покушавамо да максимизујемо суму

$$|x_0 - x_1| + |x_1 - x_2| + ... + |x_{n-2} - x_{n-1}| + |x_{n-1} - x_0|.$$

Сваки елемент се у суми јавља тачно два пута. У зависности од међусобног односа бројева неки елементи ће бити узети са знаком $+$, а неки са знаком $-$, и то тако да се тачно елемената узима са знаком $+$ и тачно елемената узима са знаком $-$. Циљ нам је да елементи који се узимају са знаком $+$ буду што већи, а да ови са знаком $-$ буду што мањи. Распоред који иде цик-цак постиже да се за знаком $+$ узме $\frac{n}{2}$ већих елемената низа, а са знаком $-$ узме $\frac{n}{2}$ мањих елемената низа (ако их је непаран број, тада се средњи узима једном са знаком $-$, а једном са знаком $+$). Заиста, ако имамо 6 елемената $a_0 \leq a_1 \leq a_2 \leq a_3 \leq a_4 \leq a_5$, цик-цак распоред даје вредност

$$|a_0 - a_5| + |a_5 - a_1| + |a_1 - a_4| + |a_4 - a_2| + |a_2 - a_3| + |a_3 - a_0|,$$

што је једнако

$$(a_5 - a_0) + (a_5 - a_1) + (a_4 - a_1) + (a_4 - a_2) + (a_3 - a_2) + (a_3 - a_0),$$

тј.

$$2 \cdot (a_5 + a_4 + a_3) - 2 \cdot (a_2 + a_1 + a_0)$$

Јасно је да се не може добити више од овога, а ово се, видели смо, увек може експлицитно достићи баш цик-цак распоредом.

Елементе низа можемо експлицитно сортирати елементе низа, па затим распоредити елементе у нови низ по цик-цак редоследу у нови низ и за тај нови низ израчунати збир апсолутних вредности разлика суседних елемената.

```
n = int(input())
a = sorted(map(int, input().split()))
b = n*[0]
i = 0; j = n-1;
for k in range(n):
    if k % 2 != 0:
        b[k] = a[i]; i += 1
    else:
        b[k] = a[j]; j -= 1

zbir = 0
for k in range(1, n):
```

```
    zbir += abs(b[k] - b[k-1])
zbir += abs(b[n-1] - b[0]);

print(zbir)
```

Помоћни низ се може веома једноставно избећи у имплементацији.

```
n = int(input())
a = sorted(map(int, input().split()))
s = 0
pomeri_sleva = True
poc, kraj = 0, n-1
while poc < kraj:
    s += a[kraj]-a[poc]
    if pomeri_sleva:
        poc += 1
    else:
        kraj -= 1
    pomeri_sleva = not pomeri_sleva
s += a[kraj] - a[0]
print(s)
```

Ако пажљиво размотримо доказ коректности, примећујемо да распоред у коме идемо од прве до последње, па до друге, затим претпоследње куће итд., није једини који даје максимални пређени пут. Довољно је само да наизменично узимамо елементе из прве и друге половине низа (у било ком редоследу). Ово инспирише још једноставнији алгоритам за израчунавање траженог максимума (саберемо $\frac{n}{2}$ бројева са почетка, одузмемо $\frac{n}{2}$ бројева са краја низа и помножимо са 2).

```
n = int(input())
a = sorted(map(int, input().split()))

zbir = 0
for i in range(n // 2):
    zbir += a[n-1-i]
    zbir -= a[i]

print(zbir * 2)
```

Задатак: Инвестициони фонд

Поставка: Инвестициони фонд који се бави продајом криптовалута је објавио очекивани профит за сваки могући уложени износ у биткоинима. Никола је студент-програмер који је био веома вредан током прошле године, зарадио је N биткоина и жели да распореди улоге тако да добије што је већи могући профит. Пошто тренутно мора да спрема испите и не може да програмира, ти му помози тако што ћеш написати програм који одређује максимални профит.

Улаз: Са стандардног улаза се уноси број биткоина N ($1 \leq N \leq 5000$), које Никола има на располагању а затим N позитивних природних бројева раздвојених са по јед-

ним размаком који представљају очекивану добит за сваки уложени износ од 1 до N у биткоинима.

Излаз: На стандардни излаз исписати највећи могући износ биткоина који Никола може добити након улагања.

Пример 1

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

Пример 2

Улаз
```
6
10 21 32 41 51 61
```
Излаз
```
64
```

Објашњење

Ако Јовица два пута уложи по 3 биткоина имаће профит 32+32 = 64.

Пример 3

Улаз
```
10
1 5 8 9 10 17 17 20 24 26
```

Излаз
```
27
```

Објашњење

Ако Јовица уложи 6 биткоина и два пута по 2 биткоина, имаће профит 17 + 5 + 5 = 27.

Решење: Задатак решавамо тако што за сваки могући новчани улог n између 1 и N израчунавамо зараду која се може добити ако се уложи баш тај улог. Добит од улога n је дата на улазу, и након тога на преостаје $N - n$ новца. Тиме смо за сваки улаз n полазни проблем свели на потпроблем истог облика, али мање димензије, који се може решити рекурзијом. Ако обележимо са $f(n)$ највећи профит који се може добити улагањем n динара, важе следеће рекурентне релације:

$$\begin{aligned} f(0) &= 0 \\ f(N) &= \max_{1 \leq n \leq N} (d(n) + f(N-n)) \end{aligned}$$

На основу овога је веома једноставно дефинисати рекурзивну функцију.

```
def zarada(N, dobit):
    if N == 0:
        return 0
    maks = 0
    for n in range(1, N+1):
        maks = max(maks, dobit[n] + zarada(N-n, dobit))
    return maks
```

```
N = int(input())
dobit = [0] + list(map(int, input().split()));
print(zarada(N, dobit))
```

Пошто у претходном решењу долази до понављања рекурзивних позива, потребно је применити динамичко програмирање. Један начин је да се изврши мемоизација.

```
def zarada_(N, dobit, memo):
    if memo[N] != -1:
        return memo[N]
    if N == 0:
        return 0
    maks = 0
    for n in range(1, N+1):
        maks = max(maks, dobit[n] + zarada_(N-n, dobit, memo))
    memo[N] = maks
    return maks

def zarada(N, dobit):
    memo = (N+1) * [-1]
    return zarada_(N, dobit, memo)

N = int(input())
dobit = [0] + list(map(int, input().split()));
print(zarada(N, dobit))
```

Рекурзије се можемо ослободити и задатак можемо решити динамичким програмирањем навише.

```
N = int(input())
dobit = [0] + list(map(int, input().split()))
dp_zarada = (N+1) * [0]
for n in range(1, N+1):
    dp_zarada[n] = 0
    for ulog in range(1, n+1):
        c = dobit[ulog] + dp_zarada[n-ulog]
        if c >= dp_zarada[n]:
            dp_zarada[n] = c
print(dp_zarada[N])
```

1.4 Ревијално такмичење из програмирања, 21. 4. 2020.

Задатак: Дан у месецу

Поставка: Ако се зна који је дан у недељи био први дан текућег месецу, напиши програм који одређује који дан је данас.

Улаз: Прва линија стандардног улаза садржи ознаку дана у недељи првог дана текућег месеца (1 - понедељак, 2 - уторак, 3 - среда, 4 - четвртак, 5 - петак, 6 - субота, 7 - недеља), а друга линија садржи редни број текућег дана у текућем месецу.

Излаз: На стандардни излаз исписати ознаку данашњег дана.

Пример 1

Улаз	*Излаз*
1 4	4

Објашњење

Први дан је био понедељак, па је четврти дан четвртак.

Пример 2

Улаз

```
1
8
```

Излаз

```
1
```

Објашњење

Први дан је био понедељак, па је и осми дан понедељак.

Пример 3

Улаз

```
3
17
```

Излаз

```
5
```

Објашњење

Први дан је била среда, па је седамнаести дан петак.

Решење: На ознаку првог дана у месецу додаћемо колико је дана прошло до данашњег. Тај број протеклих дана се добија тако што се од редног броја данашњег дана одузме 1. Пошто у обзир треба узети и периодично понављање дана, треба пронаћи остатак при дељењу са 7. Пошто се дани броје од 1 до 7, а остаци при дељењу са 7 су од 0 до 6, примењујемо "трик" да пре израчунавања остатка одузмемо 1, а након израчунавања додамо 1.

```
prvi = int(input())
danas = int(input())
print((prvi + (danas - 1) - 1) % 7 + 1)
```

Задатак: Ветрење

Поставка: Кућа има по један прозор на источној, јужној и западној страни. У кући је ваздух устајао ако није отворен ни један прозор, кућа се проветрава ако је отворен један или два прозора, а промаја је ако су отворена сва три прозора. Одредити шта се тренутно дешава у кући на основу тога који прозори су отворени.

Улаз: Улаз се састоји од 3 реда, а у сваком реду је реч `DA` или `NE`, према томе да ли је отворен прозор на једној страни куће. Дате речи се редом односе на прозоре на источној, јужној и западној страни.

Излаз: Један од текстова `promaja`, `vetrenje`, или `ustajao vazduh` (без наводника), који описује стање у кући.

Пример 1

Улаз	*Излаз*
`DA`	`vetrenje`
`NE`	
`DA`	

Пример 2

Улаз	*Излаз*
`NE`	`ustajao vazduh`
`NE`	
`NE`	

Пример 3

Улаз	*Излаз*
`DA`	`promaja`
`DA`	
`DA`	

Решење: Задатак се може решити гранањем, тј. конструкцијом `else-if`. Проверу да ли су сва три прозора отворена можемо извршити оператором "логичко и", проверу да ли је бар један отворен можемо извршити оператором "логичко или".

```
istok = input()
jug = input()
zapad = input()
if istok == 'DA' and jug == 'DA' and zapad == 'DA':
    print('promaja')
elif istok == 'DA' or jug == 'DA' or zapad == 'DA':
    print('vetrenje')
else:
    print('ustajao vazduh')
```

Задатак: Продужавање титлова

Поставка: Станислав је ђак првог разреда и тек учи да чита. Покушава да прочита титлове на цртаном филму који гледа током карантина, али му често недостаје неколико секунди да би их прочитао до краја. Његова сестра жели да му помогне тако што ће продужити трајање сваког титла за пет секунди. Међутим, некада је пауза до наредног титла краћа од 5 секунди, па да се титлови не би преклапали, она титл продужава тачно до почетка наредног. Последњи титл се сигурно може продужити за 5 секунди. Пошто је филм дугачак, ово је напоран посао. Напиши програм који би Станислављевој сестри помогао да овај посао брже заврши.

Улаз: Са стандардног улаза се уноси укупан број титлова n ($1 \leq n \leq 1000$), и након тога у наредних n линија подаци о титловима. За сваки титл се уноси време почетка и време краја (у односу на почетак филма), раздвојени размаком. Свако време задато је у облику `hh:mm:ss`. Филм не траје дуже од 3 сата.

Излаз: На стандардни излаз исписати податке о продуженим титловима (у истом формату у ком су унети).

Пример

Улаз	*Излаз*
5	5
00:01:43 00:01:48	00:01:43 00:01:53
00:01:56 00:01:59	00:01:56 00:02:04
00:02:17 00:02:23	00:02:17 00:02:24
00:02:24 00:02:26	00:02:24 00:02:30
00:02:30 00:02:38	00:02:30 00:02:43

Решење: Подацима о времену најједноставније можемо баратати ако их претворимо у секунде (протекле од претходне поноћи), слично као у задатку Поноћ. Једноставности ради, издвојићемо функције за учитавање и испис времена и података о титлу (време почетка и краја). Након учитавања података претворићемо их у секунде, а пре исписивања ћемо их претворити у сате, минуте и секунде.

Титлове ћемо обрађивати у петљи и у сваком тренутку ћемо одржавати податке о текућем и наредном титлу. Текући титл ћемо учитати пре петље, а затим ћемо у петљи $n-1$ пута учитавати наредни титл, поправљати текући (знајући време почетка наредног) и након тога наредни проглашавати текућим за следећу итерацију. Сваки текући титл у петљи обрађујемо тако што препишемо време почетка и мање од увећаног времена завршетка и времена почетка наредног титла. Последњи титл ћемо обрадити након петље, тако што ћемо преписати време почетка и увећано време завршетка.

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

def odSekundi(S):
    s = S % 60; S = S // 60
    m = S % 60; S = S // 60
    h = S
    return (h, m, s)

# ucitavamo vreme sa standardnog ulaza u formatu hh:mm:ss
def ucitajVreme(vreme):
    str = vreme.split(":")
    return uSekunde(int(str[0]), int(str[1]), int(str[2]))

# ucitavamo podatke o titlu (pocetak i kraj u formatu hh:mm:ss)
def ucitajTitl():
    str = input().split()
    start = ucitajVreme(str[0])
    end = ucitajVreme(str[1])
    return (start, end)


# ispisujemo podatke o vremenu u formatu hh:mm:ss
def ispisiVreme(vreme):
    (h, m, s) = odSekundi(vreme)
    print(format(h, "02"), format(m, "02"), format(s, "02"), sep=":", end="")
```

```
# ispisiujemo podatke o titlu (pocetak i kraj u formatu hh:mm:ss)
def ispisiTitl(start, end):
    ispisiVreme(start)
    print(" ", end="")
    ispisiVreme(end)
    print()

# ukupan broj titlova
n = int(input())
print(n)
# ucitavamo podatke o tekucem titlu
(start_t, end_t) = ucitajTitl()
for i in range(1, n):
    # ucitavamo podatke o narednom titlu
    (start_n, end_n) = ucitajTitl()

    # obradjujemo podatke o tekucem titlu, uzevsi u obzir pocetak narednog
    ispisiTitl(start_t, min(end_t + 5, start_n))
    # naredni postaje tekuci za sledecu iteraciju
    start_t = start_n; end_t = end_n

# poslednji titl se sigurno produzava
ispisiTitl(start_t, end_t + 5)
```

Задатак: Абацаба

Поставка: Низ слова $ABACABADABACABAEABACABADABACABAFABACA...$ се може формирати на следећи начин:

1. Низ је на почетку празан.
2. На низ се допише прво велико слово енглеског алфабета које се не појављује у формираном делу низа, а иза тог слова се понове сва слова која су се појавила пре њега.
3. Корак 2 се понови потребан број пута

Тако после прве примене корака 2 добијамо низ A, после друге низ ABA, после треће низ $ABACABA$ итд.

Одредити слово које се појављује на n-том месту у низу, бројећи места од 1. Редослед слова у енглеском алфабету је ABCDEFGHIJKLMNOPQRSTUVWXYZ.

Улаз: Један природан број мањи од 67 108 864.

Излаз: Једно велико слово енглеског алфабета.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
8	D	65	A	100	C

Решење: Решење грубом силом је да се у меморији креира цео низ карактера и да

се затим прочита карактер са одговарајуће позиције. Ово решење троши превише меморије, а и времена док се дугачак низ карактера не изгради.

```
n = int(input())
s = ""
slovo = 0
while len(s) <= n-1:
    s = s + chr(slovo + ord('A')) + s
    slovo += 1
print(s[n-1])
```

Задатак се може решити без креирања дугачке ниске карактера, ако пажљиво проучимо правилност по којој се слова појављују. Прво слово `A` се налази на месту 1, прво слово `B` на месту 2, прво слово `C` на месту 4, прво слово `D` на месту 8 итд., па се може наслутити да се прва појављивања слова налазе на местима која су степени броја 2.

Заиста, ако дужину низа карактера пре уметања новог слова абецеде обележимо са d_k, важи да је $d_0 = 0$ (пре уметања слова `A` не налази се ниједан карактер) и да је $d_{k+1} = 2d_k + 1$ (пре уметања новог слова налази се ниска која је добијена тако што је претходно слово спојено са два појављивања ниске која се појављивала пре тог претходног слова). Зато је $d_k = 2^k - 1$ (важи да је $2^0 - 1 = 0$ и да је $2^{k+1} - 1 = 2 \cdot (2^k - 1) + 1$), док је прва позиција слова k (ако се слова броје од нуле) једнака 2^k.

Дакле, ако је унети број n неки степен двојке $n = 2^k$, онда је у питању место на ком се слово први пут појављује и важи да је $k = \log_2 n$ ($\log_2 n$ означава број k такав да је $2^k = n$, нпр. $\log_2 32 = 5$, јер је $2^5 = 32$). Знајући k слово лако можемо одредити сабирајући k са ASCII/UNICODE кодом слова `A`.

У супротном проблем можемо свести на проблем мање димензије. Нека је $2^k < n < 2^{k+1}$, тј. нека је k највећи степен двојке мањи од n. Карактер који тражимо налази се, дакле, иза позиције 2^k у делу низа који је добијен копирањем дела низа испред позиције 2^k. Зато је n-ти карактер у целом низу једнак $(n - 2^k)$-том карактеру у копираном делу, а пошто део који се копира почиње на почетку низа, једнак је $(n-2^k)$-том карактеру у целом низу.

Овим смо описали рекурзивни поступак којим ефикасно можемо доћи до решења.

$$f(n) = \begin{cases} \log_2 n, \text{за } n = 2^k \\ f(n - 2^k), \text{за } 2^k < n < 2^{k+1} \end{cases} .$$

На пример, $f(23) = f(23 - 16) = f(7) = f(7 - 4) = f(3) = f(3 - 2) = f(1) = 0$, па се на месту 23 налази слово `A`. Слично, на пример, важи да је $f(40) = f(40 - 32) = f(8) = 3$, па се на месту 40 налази слово `D`.

На основу претходне дефиниције би се могла имплементирати рекурзивна функција (за то би било потребно испитати да ли је дати број степен броја 2, наћи логаритам таквог броја, и наћи највећи степен броја 2 од ког је дати број већи или једнак). Ипак, за тим нема потребе. Анализирајући рад такве рекурзивне функције можемо унапред закључити шта ће њен резултат бити, без потребе за њеним извршавањем. Претпоставимо да знамо бинарни запис броја n тј. да знамо да се број n представља као збир

неких степенова двојке $2^{s_m} + 2^{s_{m-1}} + \ldots + 2^{s_0}$. Током рекурзије од броја ће се одузимати један по један степен двојке, све док не остане само 2^{s_0} и тада ћемо знати да је $k = s_0$. Дакле важи да је резултат једнак најмањем степену двојке који учествује разлагању броја на збир степенова двојке тј. да је тражени број k једнак позицији најдешње јединице у бинарном запису броја (претпостављамо да се позиције броје од 0, здесно). До траженог резултата је могуће доћи дељењем броја са 2_0^s, тј. узастопним дељењем броја са 2 све док последњи сабирак не постане 1, тј. све док је број паран (то ће се догодити тачно s_0 пута).

```
n = int(input())
k = 0
while n % 2 == 0:
    n //= 2
    k += 1
print(chr(ord('A') + k))
```

Сличан резултат можемо добити и баратајући директно са интерним записом броја у рачунару, коришћењем битовских оператора. Позицију крајње десне јединице једноставно можемо израчунати шифтовањем (померањем) бинарног записа броја удесно за једно место све док последња цифра не постане једнака 1 (последњу цифру можемо испитати битовском конјункцијом са 1).

```
from string import ascii_uppercase
n = int(input())
k = 0
while n & 1 == 0:
    n = n >> 1
    k += 1
print(ascii_uppercase[k])
```

Задатак: Минимална сума два броја формирана од датих цифара

Поставка: Написати програм који одређује минималну могућу суму два броја формирана од цифара датог низа (од 0 до 9). Сваки елемент низа мора бити употребљен било у једном, било у другом броју, а бројеви могу да почну и нулом. На пример, за цифре 5, 3, 0, 7, 5 добија се најмања сума 92 (јер је $35 + 057 = 92$).

Улаз: Прва линија стандардног улаза садржи број цифара n ($2 \leq n \leq 30$), а следећих n линија по једну цифру.

Излаз: Исписати у једној линији стандардног излаза тражену минималну суму.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
5	92	6	381	5	3
5		1		0	
3		2		1	
0		3		0	
7		4		2	
5		5		0	
		6			

Решење: Решење се може раздвојити на две фазе. У првој фази се скуп свих цифара може разбити на подскуп цифара од којих ћемо направити први сабирак и на подскуп цифара од којих ћемо направити други сабирак. У другој фази ћемо од сваког подскупа цифара формирати сабирке.

Друга фаза се једноставно решава, јер је јасно да је збир мањи што су сабирци мањи, а да се најмањи број од фиксираног скупа цифара добија ако се цифре надовежу слева надесно у неопадајућем поретку. Заиста, ако бисмо у броју $(c_{m-1}c_{m-2}\dots c_1c_0)_{10}$ где за неко $m > i > j \geq 0$ важи $c_i > c_j$, разменили те две цифре, добио би се мањи број (јер је $(c_{m-1}10^{m-1}+\dots c_i10^i+\dots+c_j10^j+\dots c_0)-(c_{m-1}10^{m-1}+\dots c_j10^i+\dots+c_i10^j+\dots c_0) = (c_i-c_j)10^i+(c_j-c_i)10^j = (c_i-c_j)(10^i-10^j) > 0$ (јер су оба чиниоца позитивна). Дакле, ако цифре нису сортиране неопадајући, тада број није минималан. Последица овога је и то да се све нуле стављају на почетак сабирака, па самим тим не утичу на њихове вредности (могу се потпуно изоставити већ приликом учитавања).

Наиван начин да се прва фаза реши је да се пробају све могућности тј. да се испробају сва могућа разбијања полазног скупа на два подскупа. Ово се може имплементирати рекурзивном функцијом, слично као у задатку . Овакав алгоритам је сигурно коректан (јер ефективно испробава све могућности). Ово решење испробава све подскупове којих има експоненцијално много, па је овај алгоритам веома неефикасан.

```
def minZbirPodskupova_(cifre, i, a, b):
    if i == len(cifre):
        return a + b

    return min(minZbirPodskupova_(cifre, i + 1, 10 * a + cifre[i], b),
               minZbirPodskupova_(cifre, i + 1, a, 10 * b + cifre[i]))

def minZbirPodskupova(cifre):
  return minZbirPodskupova_(cifre, 0, 0, 0)


# broj clanova niza
n = int(input())
# ucitavamo elemente niza cifara
cifre = [int(input()) for i in range(n)]

# sortiramo niz cifara
```

```
cifre.sort()

print(minZbirPodskupova(cifre))
```

Докажимо да ће збир бити најмањи ако сабирци имају приближно једнак број цифара (као што смо рекли, водеће нуле можемо занемарити и надаље ћемо претпоставити да су све цифре различите од нуле). Ако би се број цифара сабирака разликовао за више од 1, тада би се мањи збир могао добити ако би се водећа цифра из дужег сабирка пребацила на почетак краћег. Заиста, нека један сабирак има m цифара, а други n цифара, при чему је $m > n + 1$ и нека је водећа цифра првог сабирка c. Први број се након пребацивања водеће цифре смањује за $c \cdot 10^{m-1}$, а други се повећава за $c \cdot 10^n$, па се збир умањује за $c \cdot (10^{m-1} - 10^n)$, што је сигурно позитивно, јер је $m - 1 > n$.

Ако један број има цифру више, онда то треба да буде најмања од свих цифара. Већ смо доказали да цифре оба броја треба да буду сортиране неопадајуће. Ако би краћи број почињао мањом цифром него дужи, тада би се разменом те две цифре збир смањио. Заиста, ако краћи број има n, а дужи $n + 1$ цифру, и ако су водеће цифре $c_n > c_{n-1}$ тада пре размене у збиру учествују сабирци $c_n 10^n$ из првог броја и $c_{n-1}10^{n-1}$, а након размене $c_{n-1}10^n$ и $c_n 10^{n-1}$ (остали сабирци се не мењају). Међутим, $(c_n 10^n + c_{n-1}10^{n-1}) - (c_{n-1}10^n + c_n 10^{n-1}) = (c_n - c_{n-1})(10^n - 10^{n-1}) > 0$, па се збир након размене смањио.

Дакле, најмања од свих цифара започиње дужи број и проблем се свео на то да распоредимо преостале цифра (остао их је паран број). Наиме, дужи број биће облика $c_n 10^n + a$, а краћи облика b, и збир ће бити најмањи могући ако је збир $a + b$ најмањи, тако да надаље водећу цифру дужег броја можемо занемарити, па је проблем који се надаље решава истог облика, без обзира на то да ли је у старту постојао непаран или паран број цифара различитих од нуле.

Распоређивање $2k$ цифара у два k-тоцифрена броја се врши тако што се две најмање од тих цифара узимају за почетне цифре тих k-тоцифрених бројева (њих је могуће распоредити како год желимо). Наиме, ако би се уместо неке од тих цифара узела нека већа цифра, укупан збир би био већи. Заиста, ако су водеће цифре c и c', тада у збировима учествују сабирци $c10^n$ и $c'10^n$. Ако би постојала нека цифра $c'' < c$ која се јавља касније у тим бројевима, тада би се разменом c и c'' збир смањио).

Дакле, задатак се може решити наредним грамзивим алгоритмом. Учитани низ цифара сортирати у монотоно неопадајућем поретку, а затим цифре низа (идући од најмањих) дописивати слева надесно наизменично броју a, па броју b (од позиција веће ка позицији мање тежине). Тражени број је збир два добијена броја.

```
# broj clanova niza
n = int(input())
# ucitavamo elemente niza cifara
cifre = [int(input()) for i in range(n)]

# sortiramo niz cifara
cifre.sort()

a = 0; b = 0
```

```
# naizmenicno smestati sortirane cifre sleva nadesno
for i in range(n):
    if i % 2 == 0:
      a = a * 10 + cifre[i]  # u broj A,
    else:
      b = b * 10 + cifre[i]  # pa u broj B

# ispisujemo dobijeni zbir
print(a + b)
```

Алтернативно, алгоритам се може формулисати тако да се цифре обрађују у неопадајућем поретку и да се у сваком кораку цифра дописује здесна мањем од два броја.

```
# ucitavamo niz cifara
n = int(input())
# ucitavamo elemente niza cifara
cifre = [int(input()) for i in range(n)]

# sortiramo niz cifara
cifre.sort()

# vazi a <= b
a = 0; b = 0
for i in range(n):
    # tekucu cifru dodajemo na kraj manjeg broja
    a = 10 * a + cifre[i]
    # obezbedjujemo da a ostane manji
    if a > b:
        a, b = b, a

# ispisujemo trazeni zbir
print(a + b)
```

Задатак: Максимални принос

Поставка: Фармер поседује њиву димензије $a \times b$ метара. Да би лакше парцелисао њиву, бројеви a и b су цели. На основу субвенције добио је могућности да продужи странице своје њиве укупно за c метара (али тако да њива остане целобројних димензија). Он жели да то уради тако да након продужења површина буде што већа, тако да може да оствари што већи укупан принос. Напиши програм који одређује највећу могућу површину њиве након продужења страница.

Улаз: Са стандардног улаза се учитавају природни бројеви $a, b, c \leq 10^4$, раздвојени са по једним размаком.

Излаз: На стандардни излаз исписати максималну површину након продужења страница.

Пример 1

Улаз *Излаз*
`5 10 3` `80`

Објашњење

Димензија након проширења ће бити 8×10.

Пример 2

Улаз

`9 10 4`

Излаз

`132`

Објашњење

Димензија након проширења ће бити 11×12.

Пример 3

Улаз

`14 17 5`

Излаз

`324`

Објашњење

Димензија након проширења ће бити 18×18.

Решење: Од свих правоугаоника датог фиксираног обима, највећу површину има квадрат. Заиста, ако је познат обим правоугаоника $O = 2(a + b)$, тада је познат и полуобим $a+b = s$. Површина $a \cdot b = a \cdot (s-a) = a \cdot s - a \cdot a = s^2/4 - (a - s/2)^2$. Пошто је $(a - s/2)^2 \geq 0$, површина не може бити већа од $s^2/4$, а једнака је тој вредности када је $a = b = s/2$. Зато увећање треба направити тако да се добије облик који је што сличнији квадрату.

Нека је $a \leq b$. Ако је $a + c \leq b$, тада се целокупан износ увећања c може додати на мању страницу a. У супротном се прво краћа страница a продужи тако да постане једнака дужој страници b, а затим се преостали износ увећања $(c-(b-a))$ подели што равномерније могуће (ако је то паран број може се добити квадрат, а ако није, тада се добија правоугаоник код којег је једна страница за један дужа од друге). У имплементацији тај ефекат можемо постићи тако што страницу b увећамо за $\left\lfloor \frac{c-(b-a)}{2} \right\rfloor$ и $\left\lceil \frac{c-(b-a)}{2} \right\rceil$.

```
(a, b, c) = map(int, input().split())
if a > b:
    a, b = b, a
if c <= b - a:
    a += c
else:
```

```
    preostalo = c - (b - a)
    a = b + preostalo // 2
    b = b + (preostalo + 1) // 2

print(a * b)
```

Задатак: Највећи поновљени елемент

Поставка: Напиши програм који у низу бројева одређује највећи број који се појављује бар два пута у низу или констатује да такав број не постоји.

Улаз: Са стандардног улаза се учитава број n ($1 \leq n \leq 50000$), а затим у наредних n редова n целих бројева између 1 и 50000.

Излаз: На стандардни излаз исписати тражени број или текст `nema`, ако су сви елементи низа различити.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
6	3	3	nema	6	3
3		1		3	
8		2		3	
2		3		2	
2				2	
3				1	
5				1	

Решење: Решење грубом силом подразумева да се за сваки елемент провери да ли се понавља бар два пута у низу (нпр. тако што се преброје његова појављивања или тако што се изврши претрага за текућим елементом на свим позицијама осим на текућом) и да се пронађе максимум елемената који се појављују више пута. С обзиром на то да се анализира сваки од n елемената и да претрага (или пребројавање појављивања) у најгорем случају захтева обилазак скоро целог низа, сложеност овог алгоритма је $O(n^2)$, па је он недовољно ефикасан.

```
# ucitavamo elemente u niz
n = int(input())
a = [int(input()) for i in range(n)]

# najveci ponovljeni element
max = -1
# analiziramo jedan po jedan element niza
for i in range(n):
    # proveravamo da li se tekuci element ponavlja u nizu
    ponavljaSe = False
    for j in range(n):
        if i != j and a[i] == a[j]:
            ponavljaSe = True
            break
    # azuriramo maksimum ako je potrebno
```

```
    if ponavljaSe and a[i] > max:
        max = a[i]

# prijavljujemo rezultat
if max != -1:
    print(max)
else:
    print("nema")
```

Алгоритам боље сложености се може добити ако се елементи низа претходно сортирају. Након тога могуће је елементе обрађивати са краја низа и за сваки елемент проверавати да ли је једнак претходном (ако се елемент понавља, након сортирања сва његова појављивања постају узастопна унутар низа). Пошто елемент обрађујемо у опадајућем редоследу, први елемент који се понавља уједно је и највећи такав елемент. Сложеношћу доминира сортирање, које, ако се ефикасно изврши сложености $O(n \log n)$.

```
# ucitavamo element u niz
n = int(input())
a = [int(input()) for i in range(n)]

# sortiramo ih
a.sort()

# obilazimo elemente u opadajucem redosledu
k = n-1
while k > 0:
    # element je ponovoljen akko je jednak prethodnom
    if a[k] == a[k-1]:
        # prvi ponovljeni je ujedno i najveci takav
        print(a[k])
        break
    k -= 1

# dosli smo do kraja i nismo nasli ponovljeni
if k == 0:
    print("nema")
```

Задатак можемо решити и употребом структура података. Ако учитане бројеве сместимо у скуп, тада ефикасно за сваки нови елемент можемо проверити да ли је поновљен или нови. Поновљене елементе можемо сместити у посебан низ и на крају максимум тог низа можемо одредити у линеарној сложености. Алтернативно, поновљене елементе бисмо могли скадиштити у сортираном скупу и тада бисмо максимум могли очитати у логаритамском времену (али по цену скупљег додавања елемената у колекцију поновљених). Ако претпоставимо да су операције над скупом логаритамске сложености, сложеност овог алгоритма је $O(n \log n)$.

```
n = int(input())
brojevi = set()   # skup svih ucitanih elemenata
```

```
ponovljeni = []   # lista svih elemenata koji se ponavljaju
for i in range(n):
    x = int(input())
    if x in brojevi:          # element je vec bio ucitan
        ponovljeni.append(x)  #    dodajemo ga medju ponovljene
    else:                     # element nije ranije bio ucitan
        brojevi.add(x)        #    dodajemo ga u skup ucitanih

if len(ponovljeni) > 0:
    print(max(ponovljeni))    # ispisujemo najveci ponovljeni element
else:
    print("nema")
```

Задатак: Суме трапеза

Поставка: На основу квадратне матрице A_{nxn} формирати матрицу B_{nxn} такву да је $b_{i,j}$ једнак суми оних $a_{p,q}$ за које је $p \leq i$, а $p + q \leq i + j$. Ти елементи су смештени у правоугаоном трапезу, како је приказано на слици.

$a_{0,0}$	...	...	...	...	...	$a_{0,i+j}$	...
...	...	...	...	...	$a_{1,i+j-1}$	...	...
...	...	...	...	...	...	...	...
...	...	...	$a_{i-1,j+1}$	...	...	...	...
$a_{i,0}$	...	$a_{i,j}$	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

Улаз: Са стандардног улаза се учитава број n ($3 \leq n \leq 750$), а затим и квадратна матрица димензије nxn која садржи бројеве између 0 и 9.

Излаз: На стандардни излаз исписује се матрица сума трапезних области.

Пример

Улаз *Излаз*

```
4          1 1 6 7
1 0 5 1    4 10 13 14
3 1 2 1    14 18 19 21
4 1 0 2    18 20 25 29
0 1 3 4
```

Решење: Формулација задатка сугерише једноставно, али неефикасно, решење које обично прво пада на памет: да за свако v и k израчунамо $B_{v,k}$ као суму трапеза коме је (v, k) теме кроз које пролази крак трапеза. За то израчунавање се користе два помоћна циклуса, и укупна сложеност је $O(n^4)$.

```
def saberiTrapeze(A, B, n):
  for v in range(n):
    for k in range(n):
      B[v][k] = 0
      for p in range(v + 1):
```

```
            kraj = min(n-1, v + k - p)
            for q in range(kraj + 1):
                B[v][k] += A[p][q]

def ispisi(A, n):
  for v in range(n):
    print(*A[v])
  print()

def ucitaj():
    n = int(input())
    A = [ [0]*n for _ in range(n) ]
    A = [ list(map(int, input().split())) for _ in range(n) ]
    B = [ [0]*n for _ in range(n) ]
    return n, A, B

n, A, B = ucitaj()
saberiTrapeze(A, B, n)
ispisi(B, n)
```

У следећем алгоритму избегавамо понављање истих сабирања тако што користимо већ добијене суме (слично као у задатку). Размотримо трапез чију суму треба уписати на позицију (v, k). Претпоставимо, за почетак да се (v, k) не налази на левој, горњој, нити десној ивици матрице (тј. да је $v \neq 0$ и $k \neq 0$ и $k \neq n-1$). Сви елементи који су на наредној слици означени овако и овако, део су збира $B_{v,k-1}$. Слично, сви елементи означени овако и овако део су збира $B_{v-1,k+1}$. Сви елементи који су означени овако део су збира $B_{v-1,k}$.

$a_{0,0}$	...	...	...	...	...	...	$a_{0,v+k}$	...
...	...	...	...	...	...	$a_{1,v+k-1}$	...	...
...	...	...	...	...	...	...	...	...
...	...	...	$a_{v-1,k}$	$a_{v-1,k+1}$	...	...	...	...
$a_{v,0}$	...	$a_{v,k-1}$	$a_{v,k}$	...	...	...	...	...
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Пошто се у збиру $B_{v,k-1} + B_{v-1,k+1} + A_{v,k}$ два пута појављују елементи њиховог пресека чији је збир $B_{v-1,k}$, важи да је

$$B_{v,k} = B_{v,k-1} + B_{v-1,k+1} - B_{v-1,k} + A_{v,k}$$

За остале елементе слично се добија:

$$\begin{aligned} B_{0,0} &= A_{0,0}, \\ B_{0,k} &= B_{0,k-1} + A_{0,k}, \quad \text{за } k > 0, \\ B_{v,0} &= B_{v-1,1} + A_{v,0}, \quad \text{за } 0 < v < n, \\ B_{v,n-1} &= B_{v,n-2} + A_{v,n-1}, \quad \text{за } 0 < v < n. \end{aligned}$$

Коришћењем ових формула, ако је израчунавање по врстама слева надесно, проблем се може решити у две петље и без коришћења помоћне матрице B, у сложености $O(n^2)$.

```
def saberiTrapeze(A, B, n):
  for v in range(n):
    for k in range(n):
      B[v][k] = A[v][k];
      if (v == 0 and k == 0):
        continue

      if (v == 0):
        B[0][k] += B[0][k - 1]
        continue

      if (k == 0):
        B[v][0] += B[v - 1][1]
        continue

      if (k == n - 1):
        B[v][k] += B[v][k - 1]
        continue

      B[v][k] += B[v][k-1] + B[v-1][k+1] - B[v-1][k]

def ispisi(A, n):
  for v in range(n):
    print(*A[v])
  print()

def ucitaj():
    n = int(input())
    A = [ [0]*n for _ in range(n) ]
    A = [ list(map(int, input().split())) for _ in range(n) ]
    B = [ [0]*n for _ in range(n) ]
    return n, A, B

n, A, B = ucitaj()
saberiTrapeze(A, B, n)
ispisi(B, n)
```

Задатак: Број сортираних тројки

Поставка: Написати програм којим се у датом низу a целих бројева одређује колико постоји тројки $i < j < k$ таквих да је $a_i < a_j < a_k$.

Улаз: У првој линији стандардног улаза налази се број елемената низа n ($1 \leq n \leq 50000$), а у следећих n линија налази се редом елементи низа a (цели бројеви између

0 и 50000).

Излаз: На стандардном излазу у једној линији приказати тражени број тројки - ако је тај број већи од 10^9 приказати му последњих 9 цифара (без почетних нула).

Пример 1

Улаз *Излаз*

```
5
4
1
5
3
8
```

```
3
```

Објашњење

Trojke su 4, 5, 8, zatim 1, 5, 8 i na kraju 1, 3, 8.

Пример 2

Улаз

```
5
1
2
3
4
5
```

Излаз

```
10
```

Пример 3

Улаз

```
5
5
4
3
2
1
```

Излаз

```
0
```

Решење:

Решење грубом силом

Задатак можемо решити анализирањем свих тројки a_i, a_j, a_k таквих да је $i < j < k$. То постижемо коришћењем 3 бројачка циклуса, и то редом, први циклус којим бројач i узима вредности од 0 до $n-3$, други циклус којим бројач j узима вредности од $i+1$

до $n-2$ и трећи циклус којим бројач k узима вредности од $j+1$ до $n-1$ (угнежђене петље овог типа смо видели, на пример, у задацима Сви суфикси низа бројева од 1 до n и Серије 123).

Пребројавамо све тројке које задовољавају дати услов. Ако је $a_i < a_j < a_k$ бројач уређених тројки увећамо за 1. Пошто број тројки може бити превелики тражи се остатак при дељењу тог броја са 10^9. Овај остатак је могуће израчунати на крају тела спољашње петље. Наиме, израчунавање остатка у сваком кораку би непотребно значајно успорило програм, а број елемената низа је такав да увећање бројача током једног извршавања две унутрашње петље не може проузроковати прекорачење опсега целих бројева.

Приметимо да је број поређења веома велики - он кубно зависи од броја елемената низа.

Једна могућа оптимизација наста је када се примети да нам трећи циклус није потребан у случају када је $a_i \geq a_j$. Дакле, у телу другог циклуса се проверава да ли је $a_i < a_j$ и трећи циклус се извршава само ако је овај услов испуњен. Ипак, овим се и даље задржава кубна сложеност у најгорем случају (то је случај сортираног низа).

Анализа сваког средишњег елемента тројке

Анализирајмо нешто ефикаснији приступ решењу задатка. Основна идеја је да за сваки елемент a_j где је $0 < j < n-1$, одредимо број уређених тројки код којих је a_j средњи елемент (први и последњи елемент низа не могу бити средњи елементи неке тројке). Елемент a_j је средњи елемент уређене тројке $a_i < a_j < a_k$, за сваки елемент a_i лево од њега који је мањи од њега, и за сваки елемент a_k десно од њега који је већи од њега. Да би одредили број уређених тројки код којих је a_j средњи елемент потребно је одредити:

- $brojManjihLevo$ - број елемената лево од елемента a_j (елемената a_i за $0 \leq < i$) који су мањи од њега ($a_i < a_j$),
- $brojVecihDesno$ - број елемената десно од елемента a_j (a_k за $j < k < n$) који су већи од њега ($a_k > a_j$).

Тада број уређених тројки у којима је a_j средњи елемент добијамо као прозвод $brojManjihLevo \cdot brojVecihDesno$. Тада укупан број тројки увећавамо за овај број.

При том, потребно је примењивати аритметику по модулу. О њеним општим правилима било је речи у задатку Операције по модулу, али ипак можемо извршити неке ситне оптимизације (које, заправо и нису критичне за брзину коначног програма). Пошто можемо претпоставити да је претходна вредност укупног броја већ мања од 10^9 пре његовог увећавања нема потребе израчунавати његов остатак по модулу m тј. уместо правла $br = (br \bmod m + p \bmod m) \bmod m$ можемо само применити $br = (br + p \bmod m) \bmod m$.

Приметимо да је временска сложеност овог алгоритма квадратна, а не кубна, што је веома значајна уштеда, али, могуће је конструисати и ефикаснији алгоритам од тога.

```
# ucitavamo elemente u niz
n = int(input())
```

```
a = [int(input()) for _ in range(n)]

# ukupan broj sortiranih trojki
sortiranih_trojki = 0
# obradjujemo jedan po jedan element niza
for j in range(1, n-1):
    # odredjujemo broj elemenata levo od tekuceg manjih od njega
    manjih_levo = 0
    x = a[j]
    for i in range(j):
      if a[i] <x:
        manjih_levo += 1

    # odredjujemo broj elemenata desno od tekuceg vecih od njega
    vecih_desno = 0
    for k in range(j+1, n):
      if x < a[k]:
        vecih_desno += 1

    # uvecavamo brojac za ukupan broj trojki u kojem je tekuci
    # element srednji
    mod = 1_000_000_000
    sortiranih_trojki = \
        (sortiranih_trojki + manjih_levo * vecih_desno) % mod

# ispisujemo ukupan broj sortiranih trojki
print(sortiranih_trojki)
```

Пребројавање мањих и већих елемената испред и иза датог се може извршити и библиотечким функцијама.

Сортирање обједињавањем

Кључни део решења је за сваки елемент низа одредити број елемената који се у низу налазе лево од њега и строго су мањи од њега и број елемената који се у низу налазе десно од њега и строго су већи од њега. То је могуће ефикасно урадити техником "подели-па-владај", модификацијом алгоритма сортирања обједињавањем (слично као у задатку Број инверзија). Једноставности ради, претпоставимо прво да су сви елементи у низу различити.

Ако је низ једночлан, тада је он сортиран и не постоје ни мањи ни већи елементи од тог датог.

У супротном, претпоставимо да је низ подељен на две (непразне) половине приближно једнаке дужине. Претпоставимо да је након рекурзивног позива свака половина сортирана и да за сваки елемент знамо колико у тој половини низа има мањих елемената лево и већих елемената десно од њега. На пример, ако је полазни низ `4 1 5 3 8`, добијамо

```
niz:            1 4 5    3 8
```

```
manjih_levo:     0 0 2    0 1
vecih_desno:     1 1 0    1 0
```

Након тога прелазимо на обједињавање два сортирана низа и ажурирање тражених података за цео низ. Уобичајено, користимо два показивача (који указују на текући елемент у свакој половини) и помоћни низ у који смештамо резултат. Користимо и два помоћна низа у које смештамо резултујуће податке о броју мањих и већих елемената. Поредимо два текућа елемента.

- Ако је текући елемент леве половине мањи од текућег елемента десне половине, преписујемо га у резултујући сортирани низ. Елементи лево од њега су само они у левој половини, па се број елемената који су лево од њега и мањи од њега само преписује из леве половине. Сви елементи у десној половини су десно од њега у полазном низу. Од њих, они који су мањи од текућег елемента леве половине су већ преписани у резултујући низ, док су сви преостали елементи десне половине већи од њега. Зато се укупан број елемената који су десно и од текућег и већи су од њега може добити сабирањем таквих елемената из леве половине низа (који знамо на основу рекурзивног позива) и броја преосталих елемената у десној половини низа (који лако израчунавамо одузимањем позиције текућег елемента у десној половини, од њеног укупног броја елемената).
- Ако је текући елемент десне половине већи од текућег елемента леве половине, онда њега преписујемо у резултујући сортирани низ. Укупан број елемената који су лево од њега и мањи су од њега се добија сабирањем броја елемената у десној половини који су лево од текућег и мањи су од њега (а тај број знамо на основу резултата рекурзивног позива) и броја преписаних елемената леве половине (јер су сви елементи у левој половини лево од текућег елемента десне половине, док су од њега мањи тачно они који су већ преписани). Што се тиче броја елемената који су десно од текућег елемента десне половине и већи су од њега, ту само преписујемо податак о елементима из десне половине (јер у левој половини нема елемената који су десно од текућег).

У текућем примеру, добијамо следећи резултат:

```
niz:             1   3   4    5    8
manjih_levo:     0   0+1 0    2    1+3
vecih_desno:     1+2 1   1+1  0+1  0
```

тј.

```
niz:             1  3  4  5  8
manjih_levo:     0  1  0  2  4
vecih_desno:     3  1  2  1  0
```

Ако елементи нису различити, алгоритам је потребно још мало дорадити.

- Када се преписује текући елемент десне половине, потребно је обратити пажњу да су у левој половини обрађени сви елементи који су мањи или једнаки од њега, па се за број елемената у левој половини који су строго мањи од њега не може узети број преписаних елемената, већ је из тог броја потребно искључити оне преписане елементе који су једнаки текућем. То се лако решава увођењем

једног "заосталог" показивача који ће увек да указује тачно иза елемената леве половине који су строго мањи од текућег елемента десне половине).

- Аналогно, када се преписује елемент леве половине, потребно је да знамо и елементе десне половине који су строго већи од њега, а то нису сви они иза текућег (јер неки од њих могу бити и једнаки текућем). То се лако решава увођењем једног "побеглог" показивача који ће увек да показује тачно иза свих елемената десне половине који су већи или једнаки од текућег елемента леве половине.

Сложеност једног корака обједињавања две половине је линеарна у односу на укупну дужину те две половине, па је сложеност целог поступка сортирања и израчунавања иста као у случају сортирања обједињавањем и износи $O(n \log n)$.

Број тројки се лако добија израчунавањем скаларног производа добијених низова (за шта је довољно $O(n)$ операција).

```
# rekurzivna funkcija koja:
# - popunjava niz manjihLevo koji za svaki element x u delu niza [l, d]
#   izracunava broj elemenata u delu [l, d] koji su levo od x i manji od x
# - popunjava niz vecihDesno koji za svaki element x u delu niza [l, d]
#   izracunava broj elemenata u delu [l, d] koji su desno od x i veci od x
# - kao propratni efekat sortira niz
# Tmp su pomocni nizovi
def  izbroj_(a, l, d, aTmp,
             manjihLevo, manjihLevoTmp, vecihDesno, vecihDesnoTmp):
    # ako niz ima bar dva elementa
    if d - l >= 1:
        # delimo niz u delove [l, s] i [s+1, d]
        s = l + (d - l) // 2

        # rekurzivno obradjujemo obe polovine
        izbroj_(a, l, s, aTmp,
                manjihLevo, manjihLevoTmp, vecihDesno, vecihDesnoTmp)
        izbroj_(a, s+1, d, aTmp,
                manjihLevo, manjihLevoTmp, vecihDesno, vecihDesnoTmp)

        # vrsimo objedinjavanje dva sortirana niza, popunjavajuci
        # pri tom podatke o elementima manjim levo i vecim desno
        # prvo se sve smesta u pomocne nizove
        i = l; i0 = l; j = s+1; j0 = s + 1; k = 0
        while i <= s or j <= d:
            if i <= s and (j > d or a[i] <= a[j]):
                manjihLevoTmp[k] = manjihLevo[i];
                while j0 <= d and a[j0] <= a[i]:
                    j0 += 1
                vecihDesnoTmp[k] = vecihDesno[i] + (d - j0 + 1)
                aTmp[k] = a[i]
                k += 1; i += 1
            else:
```

```
                while i0 < i and a[i0] < a[j]:
                    i0 += 1
                manjihLevoTmp[k] = (i0 - l) + manjihLevo[j]
                vecihDesnoTmp[k] = vecihDesno[j]
                aTmp[k] = a[j]
                k += 1; j += 1

        # kopiramo vrednosti iz pomocnih nizova u glavne
        a[l:d+1] = aTmp[0:d-l+1]
        manjihLevo[l:d+1] = manjihLevoTmp[0:d-l+1]
        vecihDesno[l:d+1] = vecihDesnoTmp[0:d-l+1]

# ulazna tacka u rekurzivnu funkciju
def izbroj(a, manjihLevo, vecihDesno):
    n = len(a)
    aTmp = [0]*n
    manjihLevoTmp = [0]*n
    vecihDesnoTmp = [0]*n
    izbroj_(a, 0, n-1, aTmp,
            manjihLevo, manjihLevoTmp, vecihDesno, vecihDesnoTmp)

# ucitavamo elemente u niz
n = int(input())
a = [int(input()) for i in range(n)]

# za svaki element niza izracunavamo broj elemenata levo od njega
# koji su manji od njega i broj elemenata desno od njega koji su
# veci od njega
manjihLevo = [0] * n; vecihDesno = [0] * n
izbroj(a, manjihLevo, vecihDesno)

# izracunavamo i ispisujemo broj sortiranih trojki
mod = 1_000_000_000
sortiranihTrojki = 0
for i in range(n):
    sortiranihTrojki = \
        (sortiranihTrojki + manjihLevo[i]*vecihDesno[i]) % mod
print(sortiranihTrojki)
```

Фенвиково дрво

Број елемената лево од датог елемента и мањих од њега, као и број елемената десно од датог елемента и већих од њега се може одредити помоћу структуре података која допушта ефикасно постављање вредности елемента на датој позицији низа фиксне величине и израчунавање вредности збира префикса до неке позиције у том низу. То може бити, на пример, сегментно или Фенвиково стабло.

Ако одређујемо број елемената лево од датог који су строго мањи од њега, обилазићемо низ слева надесно и одржаваћемо скуп раније обрађених елемената (ако се елемен-

ти понављају, онда је то заправо мултискуп). Он је иницијално празан и проширује се текућим елементом, чим га обрадимо. Тај мултискуп ће бити представљен низом бројева такав да се на позицији i налази број појављивања елемента i у том мултискупу садржи вредност i, при чему ће над тим низом бити изграђено Фенвиково (или сегментно) дрво. Тада се број елемената строго мањих од датог елемента i може одредити одређивањем збира префикса испред позиције i (што је операција логаритамске сложености ако се користи неко од поменутих дрвета). Пошто величина дрвета зависи од вредности највећег елемента у њему, а нама нису важне апсолутне вредности елемената, него само њихов међусобни однос, пре примене алгоритма можемо сваки елемент заменити са његовим рангом у сортираном низу (исти елементи могу да имају исти ранг). Ово је могуће урадити на било који начин који је описан у задатку Ранг сваког елемента.

Ако анализирамо елементе веће од датог, тада низ треба обилазити здесна налево. Ако нас интересују елементи већи, а не мањи од датог, онда је потребно применити неку функцију која обрће број елемената (на пример, уместо елемента i може се посмтрати $n - i + 1$, чиме интервал елемената од $[1, n]$ остаје непромењен).

```
import bisect

# uvecavanje elementa u Fenvikovom drvetu na poziciji k za vrednost v
def dodaj(drvo, k, v):
    while k < n:
        drvo[k] += v
        k += k & -k

# odredjivanje zbira prefiksa Fenvikovog drveta zakljucno sa pozicijom k
def zbirPrefiksa(drvo, k):
    zbir = 0
    while k > 0:
        zbir += drvo[k]
        k -= k & -k
    return zbir

# ucitavamo elemente u listu
n = int(input())
a = [int(input()) for i in range(n)]

# odredjujemo rang svakog elementa
b = sorted(a)
for i in range(n):
    a[i] = bisect.bisect_left(b, a[i]) + 1;

# pomocu Fenvikovog drveta za svaki element niza odredjujemo
# broj elemenata koji su levo od njega i strogo su manji od njega
drvo = n*[0]
manjihLevo = []
for i in range(n):
```

```
    manjihLevo.append(zbirPrefiksa(drvo, a[i]-1))
    dodaj(drvo, a[i], 1)

# pomocu Fenvikovog drveta za svaki element niza odredjujemo
# broj elemenata koji su desno od njega i strogo su veci od njega
drvo = n*[0]
vecihDesno = []
for i in range(n-1, -1, -1):
    vecihDesno.append(zbirPrefiksa(drvo, n - a[i]))
    dodaj(drvo, n - a[i] + 1, 1)
vecihDesno.reverse()

# odredjujemo broj sortiranih trojki elemenata niza
mod = 1_000_000_000
sortiranihTrojki = 0
for i in range(n):
    sortiranihTrojki = (sortiranihTrojki + manjihLevo[i]*vecihDesno[i]) % mod

print(sortiranihTrojki)
```

Задатак: Не садрже цифру 3

Поставка: Напиши програм који одређује колико природних бројева из интервала $[0, n]$ не садрже цифру 3 у свом декадном запису.

Улаз: Прва линија стандарног улаза садржи природан број n ($n \leq 2 \cdot 10^9$).

Излаз: У првој линији стандардног излаза приказати тражени резултат.

Пример 1

Улаз *Излаз*
15 14

Објашњење

У интервалу $[0, 15]$ постоји 16 бројева, а бројеви 3 и 13 једини садрже цифру 3.

Пример 2

Улаз

999

Излаз

729

Пример 3

Улаз

12345

Излаз

8262

Решење:

Бројање бројева који не садрже цифру 3

Задатак можемо решити тако што за сваки број од 0 до n проверимо да ли садржи цифру 3, и ако не садржи увећамо бројач бројева који не садрже цифру 3 (тај бројач у почетку иницијализујемо на нулу). У том решењу примењује се алгоритам одређивања броја елемената серије који задовољавају дати услов, тј. алгоритам бројања филтриране серије (који смо видели, на пример, у задатку Просек одличних). Проверу да ли број садржи цифру 3 можемо реализовати у посебној функцији, истој какву смо имплементирали у задатку Има ли цифру.

```
def sadrziCifru3(n):
    while True:
        if (n % 10 == 3):
            return True
        n = n // 10
        if n == 0:
            break
    return False

br = 0
n = int(input())
for i in range(0, n+1):
    if not sadrziCifru3(i):
        br = br + 1
print(br)
```

Ефикасније израчунавање броја бројева

Задатак можемо решити и на много ефикаснији начин (али је решење у том случају доста комплексније). Нека $f(a, b)$ означава број бројева из интервала $[a, b]$ који у свом декадном запису не садрже цифру 3, а $f_0(n)$ број таквих бројева из интервала $[0, n]$. Размотримо пример у коме желимо да израчунамо вредност $f_0(4251)$. Све бројеве из интервала $[0, 4251]$ можемо поделити у неколико група тј. подинтервала. Важи да је

$$[0, 4251] = [0, 999] \cup [1000, 1999] \cup [2000, 2999] \cup [3000, 3999] \cup [4000, 4251].$$

Зато је

$$f_0(4251) = f(0, 999) + f(1000, 1999) + f(2000, 2999) + f(3000, 3999) + f(4000, 4251).$$

Важи да је $f(0, 999) = f_0(999)$. Такође, важи и да је $f(1000, 1999)$ једнак броју $f(0, 999)$ тј. $f_0(999)$. Заиста, између интервала $[0, 999]$ и $[1000, 1999]$ може се успоставити бијекција таква да слика у свом декадном запису садржи цифру 3 ако и само ако њен оригинал у свом декадном запису садржи цифру 3. Слично, важи и да је $f(2000, 2999) = f_0(999)$, док је $f(3000, 3999) = 0$ јер сви бројеви у интервалу $[3000, 3999]$ садрже цифру 3. На крају, важи и да је $f(4000, 4251)$ једнако $f(0, 251)$ тј. $f_0(251)$. Зато је

$$f_0(4251) = 3 \cdot f_0(999) + f_0(251).$$

Дакле, ако знамо бројеве $f_0(999)$ и $f_0(251)$ тада можемо израчунати и број $f_0(4251)$.

Иста техника се може применити и на израчунавање бројева $f_0(999)$ и $f_0(251)$.

Важи да је

$$[0, 999] = [0, 99] \cup [100, 199] \cup \ldots \cup [900, 999],$$

па је

$$f_0(999) = f(0, 99) + f(100, 199) + \ldots + f(900, 999).$$

Важи да је $f(0, 99) = f(100, 199) = f(200, 299) = f(400, 499) = \ldots = f(900, 999) = f_0(99)$, а да је $f(300, 399) = 0$. Стога је

$$f_0(999) = 8 \cdot f_0(99) + f_0(99) = 9 \cdot f_0(99).$$

Слично, важи да је

$$f_0(99) = 9 \cdot f_0(9),$$

док је $f_0(9) = 9$ (јер у интервалу $[0, 9]$ који има 10 елемената, једино елемент 3 садржи цифру 3).

Важи и да је

$$[0, 251] = [0, 99] \cup [100, 199] \cup [200, 251],$$

па је

$$f_0(251) = 2 \cdot f_0(99) + f_0(51).$$

Пошто је

$$[0, 51] = [0, 9] \cup [10, 19] \cup [20, 29] \cup [30, 39] \cup [40, 49] \cup [50, 51],$$

важи да је

$$f_0(51) = 4 \cdot f_0(9) + f_0(1).$$

Важи и да је $f_0(1) = 2$ јер су оба елемента интервала $[0, 1]$ такви да не садрже цифру 3.

Дакле, $f_0(4251) = 3 \cdot f_0(999) + f_0(251) = 3 \cdot (9 \cdot f_0(99)) + 2 \cdot f_0(99) + f_0(51) = 3 \cdot (9 \cdot (9 \cdot f_0(9))) + 2 \cdot (9 \cdot f_0(9)) + 4 \cdot f_0(9) + f_0(1) = 3 \cdot 9 \cdot 9 \cdot 9 + 2 \cdot 9 \cdot 9 + 4 \cdot 9 + 2 = 2387$.

Функцију f_0 је могуће рекурзивно дефинисати. Ако се број n разлаже на почетну цифру c и суфикс n' тада се $f_0(n)$ може израчунати на следећи начин, у зависности од цифре c (претпостављамо да број $9 \ldots 9$ има онолико деветки колико цифара има број n').

- Ако је $c < 3$ тада је $f_0(n) = c \cdot f_0(9 \ldots 9) + f_0(n')$,
- Ако је $c = 3$ тада је $f_0(n) = c \cdot f_0(9 \ldots 9)$,
- Ако је $c > 3$ тада је $f_0(n) = (c - 1) \cdot f_0(9 \ldots 9) + f_0(n')$.

Излаз из рекурзије може бити и само случај $f_0(0) = 1$ (0 је једини број у интервалу $[0, 0]$ и он не садржи цифру 3).

Проблем са имплементацијом овакве рекурзивне функције је то што се цифре одвајају слева надесно, што је компликованије него здесна налево, када број n лако разлажемо

на $n \operatorname{div} 10$ и $n \bmod 10$. Стога пре уласка у рекурзију можемо обрнути цифре броја (као у задатку). Такође, за дати број n потребно је одредити одговарајући број који се састоји само од деветки. То једноставно можемо решити тако што конструишемо број који се од броја n добија заменом свих цифара цифром 9 и ако тај број прослеђујемо као други параметар рекурзије (тај број у сваком позиву садржи само деветке и то онолико деветки колико цифара има број n).

Приметимо да се од једног рекурзивног позива за број са k цифара најчешће добијају два рекурзивна позива за бројеве са $k-1$ цифара, што указује на то да је сложеност експоненцијална (за основу 2), што је допустиво, јер је број цифара мали. Ипак, с обзиром на то да се исти рекурзивни позиви преклапају (пре свега они облика $f_0(9\ldots9)$), имплементација се може убрзати динамичким програмирањем или тако што се се примети да је да је $f_0(\underbrace{9\ldots9}_{k}) = 9^k$, па би се ови рекурзивни позиви могли специјализовати.

```
def f_(n, d):
  if n == 0:
      return 1
  c = n % 10
  if c < 3:
      return c * f_(d // 10, d // 10) + f_(n // 10, d // 10)
  elif c == 3:
    return c * f_(d // 10, d // 10)
  else:
    return (c - 1) * f_(d // 10, d // 10) + f_(n // 10, d // 10)

def f(n):
    nObratno = 0
    devetke = 0
    while n > 0:
        nObratno = 10 * nObratno + n % 10
        devetke = 10 * devetke + 9
        n = n // 10
    return f_(nObratno, devetke)

n = int(input())
print(f(n))
```

Уместо рекурзије која ради одозго наниже, решење можемо синтетизовати одоздо навише.

Обрађиваћемо једну по једну цифру броја n, здесна налево и мало по мало ћемо проширивати обрађени суфикс n' broja n. Уведимо променљиву, нпр. br, која током итерације чува вредности $f_0(n')$ за суфиксе n' броја n које током итерације обрађујемо, и променљиву, нпр. t, која чува вредности бројева $f_0(9_0\ldots9) = 9^k$, за бројеве $9\ldots9$ који имају k цифара 9, колико укупно цифара има у тренуном суфиксу n'.

Променљиве t и br иницијализујемо на 1 (претпостављамо да је пре петље обрађен празан суфикс који одговара броју 0 и да је $9^0 = 1$). Затим у петљи обрађујемо цифру

по цифру броја n, кренувши од цифре јединица (алгоритам издвајања цифара описан је у задатку Број и збир цифара броја). Променљиву br ажурирамо на следећи начин, у зависности од текуће цифре c.

- Ако је $c < 3$ тада је $br = c \cdot t + br$,
- Ако је $c = 3$ тада је $br = c \cdot t$,
- Ако је $c > 3$ тада је $br = (c - 1) \cdot t + br$.

Променљиву t ажурирамо на вредност $9 \cdot t$.

Размотримо поново пример израчунавања $f_0(4251)$ и применимо претходно описани алгоритам на број 4251.

Претпоставимо да на почетку променљиве t и br имају вредност 1.

Прво обрађујемо последњу (прву с десна) цифру броја n, а то је цифра 1. Пошто је она мања од 3, ажурирамо br на вредност $c \cdot t + br = 1 \cdot 1 + 1 = 2$, а вредност t на вредност $9 \cdot t = 9$. Након овога, променљива br има вредност $f_0(1)$, а променљива t има вредност $f_0(9)$.

Наредна цифра је 5 и пошто је она већа од 3, ажурирамо вредност br на $(c-1) \cdot t + br = 4 \cdot 9 + 2 = 38$, а вредност t на редност $9 \cdot t = 81$. Након овога, променљива br има вредност $f_0(51)$, а променљива t има вредност $f_0(99)$.

Наредна цифра је 2 и пошто је она мања од 3, ажурирамо вредност br на $c \cdot t + br = 2 \cdot 81 + 38 = 200$, а вредност t на редност $9 \cdot t = 9 \cdot 81 = 729$. Након овога, променљива br има вредност $f_0(251)$, а променљива t има вредност $f_0(999)$.

На крају, почетна цифра је 4 и пошто је она већа од 4, ажурирамо вредност br на $(c - 1) \cdot t + br = 3 \cdot 729 + 200 = 2387$. Вредност t се ажурира на $9 \cdot 729 = 6561$, али се та вредност даље не користи. Након овога, променљива br има вредност $f_0(4251)$, а променљива t има вредност $f_0(9999)$.

```
n = int(input())
t = 1
br = 1
while n > 0:
    c = n % 10
    if c < 3:
        br = c * t + br
    elif c == 3:
        br = c * t
    else:
        br = (c - 1) * t + br
    t = 9 * t
    n = n // 10
print(br)
```