Садржај

Глава 1

1. круг квалификација

Задатак: Средња брзина

Аутор: Филип Марић

Тело је први део пута прешло крећући се равномерно, једном брзином, а други део пута такође равномерно, крећући се неком другом брзином напиши програм који одређује средњу брзину овог кретања.

Напомена: средња брзина се израчунава тако што се укупан пређени пут подели са укупним протеклим временом.

Опис улаза

Са стандардног улаза се учитава цео број s_1, затим цео број t_1, затим цео број s_2 и цео број t_2 (сваки број је у посебном реду). Вредности s_1 и s_2 представљају број метара које је возило прешло у првом и у другом делу пута. Бројеви t_1 и t_2 представљају број секунди које је тело провело крећући се у првом и у другом делу пута.

Опис излаза

На стандардни излаз испсиати средњу брзину у метрима у секунди.

Пример 1

Улаз	*Излаз*	*Објашњење*
4 2 8 4	2	$$v_{sr} = \frac{s_1 + s_2}{t_1 + t_2} = \frac{4+8}{2+4} = \frac{12}{6} = 2$$

Пример 2

Улаз	*Излаз*	*Објашњење*
50 8 35 2	8.5	$$v_{sr} = \frac{s_1 + s_2}{t_1 + t_2} = \frac{50+35}{8+2} = \frac{85}{10} = 8.5$$

Решење

Улазне податке ћемо учитати у 4 променљиве (`s1`, `t1`, `s2` и `t2`). Затим ћемо применом формуле израчунати средњу брзину. На крају ћемо ту средњу брзину и исписати.

```
s1 = int(input())
t1 = int(input())
s2 = int(input())
t2 = int(input())

vsr = (s1 + s2) / (t1 + t2)

print(vsr)
```

Задатак: Атлетичари

Аутор: Филип Марић

У зависности од трка које трче, атлетичаре делимо на краткопругаше, средњепругаше и дугопругаше. Краткопругаши (или спринтери) трче трке дужине највише 400 метара, средњепругаши трче дуже трке од њих, али највише дужине једне миље, док дугопругаши трче трке дуже од тога. Написати програм којим се на основу дужине трке у којој атлетичар учествује одређује категорију којој он припада.

Опис улаза

Дужина трке у метрима - цео број од 60 до 42195.

Опис излаза

На стандардни излаз исписати једну од следећих речи: `kratko`, `srednje`, `dugo`.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
`400`	`kratko`	`1500`	`srednje`

Решење

Из формулације задатка произилази да имамо три категорије, које су одређене следећим интервалима:

- ако дужина у метрима припадају интервалу $[60, 400]$, тркач је краткопругаш;
- ако дужина у метрима припада интервалу $(400, 1609]$, тркач је средњепругаш (једна миља има 1609 метара);
- ако дужина у метрима припада интервалу $(1609, 42195]$, тркач је дугопругаш.

Одавде произилази кôд са три међусобно независна услова, којима се у произвољном редоследу проверава припадност температуре једном од три интервала $[60, 400]$, $(400, 1609]$ и $(1609, 42195]$. Пошто знамо да је унети број метара увек у интервалу $[60, 42195]$, довољно је испитивати припадност интервалима $(-\infty, 400]$, $(400, 1609]$ и $(1609, +\infty)$.

```
t = int(input()) # temperatura
if t <= 0:
    print("cvrsto")
```

```
if t > 0 and t < 100:
    print("tecno")
if t >= 100:
    print("gasovito")
```

Међутим, до решења се може доћи и уз коришћење такозване *конструкције* `else-if`, следећим поступком:

- ако број метара није већи од 400, тркач је краткопругаш;
- у супротном (број метара јесте већи од 400): ако је број метара није већи од 1609 (припада другом интервалу), тркач је средњепругач;
- у супротном (број метара је већи од 1609), тркач је дугопругаш.

Овим се избегавају провере које нису заиста неопходне.

```
t = int(input()) # temperatura
if t <= 0:
    print("cvrsto")
elif t < 100:
    print("tecno")
else:
    print("gasovito")
```

Аналогно претходном решењу, можемо проверавати припадност броја метара идући здесна од интервала $(1609, \infty)$ наниже.

```
t = int(input()) # temperatura
if t >= 100:
    print("gasovito")
elif t > 0:
    print("tecno")
else:
    print("cvrsto")
```

Задатак: Гпс

Аутор: Владимир Кузмановић

Перица је за рођендан добио паметни спортски сат који омугаћава праћење кретања и пређеног растојања уз помоћ ГПС-а (глобални систем за позиционирање). Перица је отишао са школом на екскурзију са богатим културним програмом и жели да тестира свој нови сат. Током екскурзије ученици треба да посете n локација. Перица жели да премери укупно растојање које ће прећи уз помоћ свог новог сата и замолио нас је да напишемо програм који ће му у томе помоћи.

Опис улаза

У првој линији стандардног улаза се учитава природан број n, $1 \leq n \leq 10^6$, и затим у следећих $n+1$ линија стандардног учитава се по један природан број који редом представљају растојања измећу две узастопне деонице пута. Растојање између две деонице је број у интервалу $[50, 250]$

Опис излаза

На стандардни излаз исписати један природан број који представља укупан пређени пут.

Пример

Улаз	*Излаз*
5	208
17	
25	
48	
37	
14	
67	

Решење

Основна идеја решења задатка може бити да прво учитамо свих $n+1$ реалних бројева у помоћни низ. Након што смо сачували бројеве у помоћном низу, потребно је да их све саберемо и тако одредимо укупни пређени пут.

Иако је идеја једноставна, задатак се на вишим нивоима такмичења не може у потпуности решити исправно на овај начин. Разлог зашто не може, најчешће се крије у постављеним границама за број n и временским и меморијским ограничењима програма. На пример, да је у услову задатка ограничење за број n било $1 \leq n \leq 10^9$, тада би нам за горњу границу вредности броја n, тј. број 1000000000, било потребно приближно 3.8 гигабајта меморије за смештање свих 10^9 бројева, јер сваки цео број заузима 4 бајта у меморији. Поред огромне количине меморије, биће нам потребно и много више времена од дозвољених 2 секунде само за учитавање тих бројева. Јасно је да са доступним ресурсима то не можемо да остваримо.

Са доступним меморијским ограничењем могли бисмо да учитамо 15ак милиона целих бројева, што би био само мали подскуп могућих димензија улаза у проширеној верзији задатка. Дакле, потребно је да смислимо решење које неће користити помоћне низове, ако желимо у потпуности исправно да решимо задатак.

Напомена: На општинском нивоу такмичења се не гледа меморијска сложеност програма, али то што ученици могу да користе додатну количину меморије у својим решењима не значи да је то у потпуности иправно нити да то треба да раде. На нижим нивоима такмичења висока меморијска сложеност програма не доноси негативне поене, али то чини на вишим нивоима. Са тим у вези, ученици од старта треба да уче да је обазрива употреба ресурса у програму заправо срж такмичарског програмирања.

```python
n = int(input())

rastojanja = []
for i in range(1, n+2):
    x = int(input())
    rastojanja.append(x)

ukupno = 0
for x in rastojanja:
    ukupno += x

print(ukupno)
```

Алтернатива првом решењу је да збир свих елемената низа израчунамо уграђеном функцијом. С обзиром да користимо помоћни низ, ни ово решење није у потпуности исправно.

```
n = int(input())

rastojanja = []
for i in range(1, n+2):
    x = int(input())
    rastojanja.append(x)

ukupno = sum(rastojanja)

print(ukupno)
```

Решење које је у потпуности исправно, користи инкременталност да би одредило збир елемената. Потребно је да током учитавања истовремено рачунамо збир елемената. На тај начин постижемо да меморијска сложеност нашег програма буде константа, тј. да не зависи од броја n.

Инкременталност је лако уочити. Ако треба да израчунамо збир k бројева, довољно је да чувамо само збир претходних $k-1$ бројева, али не и саме бројеве. Тада, тражени збир k бројева добијамо тако што познати збир $k-1$ бројева увећамо за вредност k-тог броја.

```
n = int(input())

ukupno = 0
for i in range(1,n+2):
    x = int(input())
    ukupno += x

print(ukupno)
```

Задатак: Најуспешнија партија

Аутор: Владимир Кузмановић

Перица се спрема за међународно такмичење у електронским спортовима и одлучио је да води статистику о броју поена које осваја након сваке партије своје омиљене игрице. Припреме за такмичење трају укупно n дана. С обзиром да Перица има и разне друге обавезе током дана, одлучио је да сваког дана игра тачно једну партију и да на крају сваке партије запише број освојених поена. Потребно је да напишемо програм који ће помоћи Перици да одреди редни број дана када је освојио највише поена. У случају да постоји више дана у којима је освојио максимални број поена, као резултат се узима последњи такав дан. *Напомена: Редни бројеви дана почињу од 1.*

Опис улаза

У првој линији стандардног улаза се учитава природан број n, $1 \leq n \leq 10^6$, и затим у следећих n линија стандардног учитава се по један природан број x_i, $1 \leq x_i \leq 10^9$, који редом представљају број освојених поена тога дана.

Опис излаза

На стандардни излаз исписати редни број последњег дана у којем је освојен највећи број поена.

Пример

Улаз	*Излаз*	*Објашњење*
10 250 300 180 400 260 400 380 400 370 240	8	Перица је освојио највише 400 поена и то 4., 6. и 8. дана. Према услову задатка треба да испишемо редни број последњег дана када је Перица освојио највише поена, па ће резултат бити 8.

Решење

Основна идеја решења задатка може бити да прво учитамо свих $n+1$ реалних бројева у помоћни низ. Након што смо сачували бројеве у помоћном низу, потребно је да одредимо позицију последњег појављивања максимума.

Иако је идеја једноставна, задатак се на вишим нивоима такмичења не може у потпуности решити исправно на овај начин. Разлог зашто не може, најчешће се крије у постављеним границама за број n и временским и меморијским ограничењима програма. На пример, да је у услову задатка ограничење за број n било $1 \leq n \leq 10^9$, тада би нам за горњу границу вредности броја n, тј. број 1000000000, било потребно приближно 3.8 гигабајта меморије за смештање свих 10^9 бројева, јер сваки цео број заузима 4 бајта у меморији. Поред огромне количине меморије, биће нам потребно и много више времена од дозвољених 2 секунде само за учитавање тих бројева. Јасно је да са доступним ресурсима то не можемо да остваримо.

Са доступним меморијским ограничењем могли бисмо да учитамо 15ак милиона целих бројева, што би био само мали подскуп могућих димензија улаза у проширеној верзији задатка. Дакле, потребно је да смислимо решење које неће користити помоћне низове, ако желимо у потпуности исправно да решимо задатак.

Напомена: На општинском нивоу такмичења се не гледа меморијска сложеност програма, али то што ученици могу да користе додатну количину меморије у својим решењима не значи да је то у потпуности иправно нити да то треба да раде. На нижим нивоима такмичења висока меморијска сложеност програма не доноси негативне поене, али то чини на вишим нивоима. Са тим у вези, ученици од старта треба да уче да је обазрива употреба ресурса у програму заправо срж такмичарског програмирања.

```
n = int(input())

poeni=[]
for i in range(0, n):
    x = int(input())
    poeni.append(x)

maxInd = 0
for i in range(0, len(poeni)):
    if poeni[i] >= poeni[maxInd]:
```

```
        maxInd = i

print(maxInd + 1)
```

Алтернатива првом решењу је да позицију последњег појављивања максимума израчунамо уграђеним функцијама. Иако су решења са уграђеним функцијама углавном кратка и елегантна, често се иза уграђених функција крије скривена временска сложеност. Наиме, уграђене функције су често ефикасније од онога што ми можемо да напишемо ако се ради о уобичајеним алгоритмима, али нису магично решење свих проблема, јер нису тако лако прилагодљиве различитим случајевима. Сложеност се крије управо у том прилагођавању програма ономе што нам нуде уграђене функције.

У програмском језику Пајтон постоји уграђена функција која може да одреди максимални елемент набројиве колекције (низ, вектор, ...) и постоји уграђена функција која може да одреди прво појављивање неког елемента x у набројивој колекцији. Функције се редом зову *max* и *index*. Међутим, не постоји уграђена функција која може истовремено да одреди максимални елемент и његову позицију у низу.

Због тога, морамо прво да одредимо максимални елемент колекције уз помоћ функције *max*. Након тога, треба да одредимо последње појављивање максимума. Уз помоћ функције *index* можемо да одредимо позицију првог појављивања максимума, што нам не треба. Нама је потребна последња позиција појављивања максимума Да бисмо је одредили, можемо прво да окренемо низ као у огледалу уз помоћ функције *reverse*. Јасно је да ће након обртања низа, последње појављивање максимума у оригиналном низу, заправо бити прво појављивање у обрнутом низу. Дакле, сада можемо да одредимо прво појављивање максимума уз помоћ функције *index* у обрнутом низу. Тражену позицију максимума у оригиналном низу ћемо добити тако што од укупне дужине низа одузмемо вредност добијену применом функције *index* на обрнутом низу.

Ово је место на којем се крије сложеност, јер ћемо описаним поступком три пута проћи кроз цео низ. У првом пролазу ћемо одредити максимум, у другом пролазу ћемо окренути низ као у огледалу и на крају у трећем пролазу ћемо одредити индекс првог појављивања максимума у обрнутом низу.

С обзиром да користимо помоћни низ, ни ово решење није у потпуности исправно. Поред тога, скривена сложеност је још једна замка коју треба да избегнемо.

```
n = int(input())

poeni=[]
for i in range(0, n):
    x = int(input())
    poeni.append(x)

maxEl = max(poeni)
poeni.reverse()
maxInd = poeni.index(maxEl)
print(len(poeni) - maxInd)
```

Решење које је у потпуности исправно, користи инкременталност да би одредило последњу позицију максимума. Потребно је да током учитавања истовремено рачунамо позицију максимума. На тај начин постижемо да меморијска сложеност нашег програма буде константа, тј. да не зависи од броја n, као и одређивање резултата у тачно једном пролазу кроз учитане бројеве.

Инкременталност је лако уочити. Ако треба да израчунамо максимум k бројева, довољно је да чувамо само максималну вредност претходних $k-1$ бројева, али не и саме бројеве. Тада, тражену позицију максимума k бројева добијамо тако што упоредимо познати максимум $k-1$ бројева са вредности k-тог броја и уколико је потребно ажурирамо раније упамћену позицију максимума.

```
n = int(input())

max = -1
redniBroj = -1
for i in range(1,n+1):
    x = int(input())
    if x >= max:
        max = x
        redniBroj = i

print(redniBroj)
```

Задатак: Имена презимена

Аутор: Душан Попадић

У одељењу има n дечака и m девојчица. За потребе позоришне представе коју спремају у одељењу, потребно је изабрати једну девојчицу и једног дечака за главне улоге. Исписати све могуће парове које наставник може изабрати за главне улоге.

Опис улаза

У првом реду стандардног улаза се уносе бројеви n и m ($1 \leq n, m \leq 30$) раздвојени размаком. У n редова се налазе се имена дечака. У наредних m редова налазе се имена девојчица. Свако име је једна реч која има највише 10 слова.

Опис излаза

На излаз исписати све могуће парове, сваки пар у посебном реду. У оквиру једног пара треба да се испише прво име дечака, па девојчице, а имена треба да буду раздвојена размаком. Парове исписати у произвољном редоследу.

Пример 1

Улаз	*Излаз*
`2 3`	`nikola katarina`
`nikola`	`nikola saska`
`marko`	`nikola nevena`
`katarina`	`marko katarina`
`saska`	`marko saska`
`nevena`	`marko nevena`

Пример 2

Улаз	*Излаз*
`3 2`	`bosko dunja`
`bosko`	`bosko marija`
`boris`	`boris dunja`
`bosko`	`boris marija`
`dunja`	`bosko dunja`
`marija`	`bosko marija`

Решење

Опис главног решења

Потребно је унети све дечаке и девојчице у два низа, а затим итерирати по њиховим елементима са две угњеждене петље: једна петља итерира по дечацима, док ће друга по девојчицама. Овако састављен пар дечака и девојчице је потребно исписати.

```
n, m = map(int, input().split())

decaci = []
devojcice = []

for _ in range(n):
    decaci.append(input())

for _ in range(m):
    devojcice.append(input())

for decak in decaci:
    for devojcica in devojcice:
        print(f"{decak} {devojcica}")
```

Задатак: Куповина скија

Аутор: Душан Попадић

Дуња је наградној игри освојила ваучер од k динара за куповину у једној познатој радњи спортске опреме. Она је одлучила да га искористи како би себи купила нове скије и нове панцерице (ципеле за скијање). У понуди је n врста скија и m врста панцерица и за сваку врсту се зна цена за један пар. Дуња не жели да плаћа додатно у односу на оно што има на ваучеру, али такође не би волела да јој пропадне превише новца са ваучера. Зато је замолила вас да за њу изаберете један пар скија и један пар панцерица тако да укупна цена не пређе k, али да искористи што више новца са ваучера (тј. да цена буде што приближнија k).

Опис улаза

У првом реду стандардног улаза се уносе три броја $k(1000 \leq k \leq 10^9)$, n и $m(1 \leq n, m \leq 10^5)$. У наредних n редова се уноси по један природан број $s_i(1 \leq s_i \leq 10^9)$ који представља цену i-тог пара скија. У наредних m редова се уноси по један природан број $p_i(1 \leq p_i \leq 10^9)$ који представља цену i-тог пара панцерица.

Гарантује се да ће Дуња моћи да купи барем један пар скија и један пар панцерица.

Додатна ограничења

Тест примери су подељени у групе, тј. подзадатке:

- у тест примерима вредним 10 поена додатно важи $s_i, p_i \leq 100$
- у тест примерима вредним 40 поена додатно важи $n, m \leq 10^3$
- у тест примерима вредним 50 поена нема додатних ограничења

Опис излаза

У јединому реду стандардног излаза исписати један број који представља колико ће Дуњи остати новца на ваучеру после куповине.

Пример 1

Улаз	*Излаз*	*Објашњење*

`35000 3 2`
`22500`
`17300`
`5000`
`14000`
`7000`
	`3700`	Могуће комбинације цена за пар скија и панцерица су 22500+14000=36500, 22500+7000=29500, 17300+14000=31300, 17300+7000=24300, 5000+14000=19000 и 5000+7000=12000 динара. Дакле, Дуњи се највише исплати да купи скије и панцерице за 31300 динара јер је то највећи број за који не мора да додаје новац (тј. довољан јој је само ваучер). Дуњи пропада 35000-31300=3700 динара.

Пример 2

Улаз	*Излаз*	*Објашњење*

`20000 3 3`
`16000`
`12000`
`3800`
`8000`
`3000`
`5000`
	`0`	Дуњи се највише исплати да купи скије за 12000 и панцерице за 8000. Дакле укупно ће платити 20000 динара што је и вредност ваучера, па пропада 0 динара.

Решење

Решење бирањем најскупљих скија и панцерица

У првом подзадатку који доноси 10 поена су све цене мање од 100 одакле се може закључити да ће цена скија и панцерица бити највише 200 динара, што је сигурно мање од буџета (буџет је по тексту задатка сигурно макар 1000 динара). Дакле бирамо најскупље панцерице и најскупље скије.

```
k, n, m = map(int, input().split())

max_skije = 0
max_pancerice = 0

for _ in range(n):
    cena = int(input())
    max_skije = max(max_skije, cena)

for _ in range(m):
    cena = int(input())
    max_pancerice = max(max_pancerice, cena)

naj_cena = max_skije + max_pancerice

print(k - naj_cena)
```

Решење генерисањем свих парова

Задатак је могуће решити тако што се за сваки пар скија провери збир цена са сваким паром панцерица и онда изаберемо највећи збир цена који је мањи или једнак буџету.

Анализа сложености

Како је потребно проћи кроз све парове цена скија и панцерица којих има n односно m, сложеност је $O(n \cdot m)$. Ово решење доноси 40 поена.

Могуће је комбиновати ово решење са решењем првог подзадатка тако да се добије 50 поена. Потребно је одредити максимуме оба низа и уколико су у буџету изабрати најскупље скије и панцерице, иначе проверити све парове.

```
k, n, m = map(int, input().split())

skije = [int(input()) for _ in range(n)]
pancerice = [int(input()) for _ in range(m)]

naj_cena = 0

for i in range(n):
    for j in range(m):
        cena = skije[i] + pancerice[j]
        if cena <= k:
            naj_cena = max(cena, naj_cena)

print(k - naj_cena)
```

Решење коришћењем два показивача

Задатак је могуће решити применом технике два показивача. Прво сортирамо оба низа и затим крећемо од најјефтинијих скија и најскупљих панцерица. Уколико им је цена већа од буџета тада бирамо јефтиније панцерице (померамо показивач у низу панцерица улево). Уколико им је цена у оквиру буџета онда је памтимо као потенцијално решење и бирамо скупље скије (померамо показивач у низу панцерица удесно).

Анализа сложености

Сортирање низова је сложености $O(n \cdot \log(n))$, односно $O(m \cdot \log(m))$. Након што се низови сортирају потребно је још по једном проћи кроз сваки ($O(n)$ односно $O(m)$) што не утиче на укупну сложеност. Дакле, укупна сложеност је $O(n \cdot \log(n) + m \cdot \log(m))$. Ово решење доноси 100 поена.

```
k, n, m = map(int, input().split())

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

pancerice = [int(input()) for _ in range(m)]

# soritranje oba niza neopadajuce
skije.sort()
pancerice.sort()

# najbolja cena koju smo nasli do sada
najboljaCena = 0
```

```
i, j = 0, m - 1

while i < n and j >= 0:
    cena = skije[i] + pancerice[j]

    if cena > k:
        j -= 1
    else:
        najboljaCena = max(najboljaCena, cena)
        i += 1

print(k - najboljaCena)
```

Решење коришћењем бинарне претраге

Задатак је могуће у потпуности решити и применом бинарне претраге. Потребно је сортирати низ панцерица и затим проћи кроз све цене скија, одредити колико остаје пара у буџету (зваћемо то буџет за панцерице) након што се те скије купе и онда у сортираном низу цена панцерица бинарном претрагом тражити највећу вредност која је мања или једнака од буџета за панцерице. Наравно, током ове претраге памтимо највећу укупну цену коју смо успели да постигнемо.

Алтернативно, може се радити са сортираним низом скија, а онда пролазити кроз све цене панцерица.

Анализа сложености

Сортирање низа је сложености $O(m \cdot \log(m))$, а бинарне претраге кроз тај низ је $O(\log(m)$. Како се бинарна претрага у низу панцерица по једном позива за сваку цену скија, сложеност претраживања је $O(n \log(m))$, па је укупна сложеност $O(m \log(m) + n \log(m))$. Ово решење доноси 100 поена.

```
from bisect import bisect_right

# Unos k, n, m u jednom redu
k, n, m = map(int, input().split())

# Unos cena za skije
skije = [int(input()) for _ in range(n)]

# Unos cena za pancerice i sortiranje
pancerice = sorted(int(input()) for _ in range(m))

najCena = 0

for cena_skije in skije:
    budzetPancerice = k - cenaSkije

    # koristeci binarnu pretragu trazimo najvecu cenu pancerica,
    # koja je u okviru budzeta. Ako ne postoji cena ce biti -1
```

```
    indexPancerice = bisect_right(pancerice, budzetPancerice) - 1

    if indexPancerice >= 0:
        cenaPancerice = pancerice[indexPancerice]
        najCena = max(najCena, cenaSkije + cenaPancerice)

print(k - naj_cena)
```

Задатак: Такси промо-кôд

Аутор: Филип Марић

Корисник је добио промо-кôд са којим плаћа вожњу таксијем 500 динара мање, али цена коју плаћа не сме бити нижа од цене поласка која је 300 динара. Написати програм који за унету цену вожње без попуста одређује колико корисник треба да плати након урачунавања овог попуста.

Опис улаза

Са стандардног улаза се уноси цена без попуста. Цена је природан број мањи од 10000.

Опис излаза

На стандардни излаз се исписује цена са попустом.

Пример 1

Улаз	*Излаз*
1300	800

Пример 2

Улаз	*Излаз*
600	300

Решење

Нека је цена вожње без попуста c. Ако је вредност $c - 500$ мања од 300 динара, цена после попуста је 300, а у супротном је цена после попуста $c - 500$. Дакле, цена после попуста је максимум вредности 300 и $c - 500$.

```
cena_bez_popusta = int(input())
print(max(300, cena_bez_popusta - 500))
```

Задатак се може решити и експлицитним гранањем.

```
cena_bez_popusta = int(input())
if (cena_bez_popusta > 800):
    print(cena_bez_popusta - 500)
else:
    print(300)
```

Задатак: Број сугласника

Аутор: Огњен Тешић

На улазу је дата реч s дужине n (дужина речи је не више од 1000 карактера) која садржи мала слова **енглеске абецеде**. Избројати колико сугласника постоји у датој речи (бројати и понављања). *Напомена:* Ради једноставности сматрамо да је `r` увек сугласник.

Опис улаза

У првом реду стандардног улаза је дата дужина речи n ($1 \leq n \leq 1000$). У другом реду стандардног улаза је дата речи s.

Опис излаза

У јединим реду стандардног излаза исписати колико сугласника постоји у датој речи.

Пример 1

Улаз	*Излаз*	*Објашњење*
10 ponedeljak	6	У речи се јављају сугласници `p`, `n`, `d`, `l`, `j` и `k` (приметите да се `lj` не рачуна као један сугласник, већ као два - `l` и `j`).

Пример 2

Улаз	*Излаз*	*Објашњење*
9 radijator	5	У речи се јављају сугласници `r`, `d`, `j`, `t` и `r`.

Решење

Проћи ћемо кроз целу реч карактер по карактер и проверити да ли је одређени карактер самогласник. Сваки пут када нађемо карактер који није самогласник, повећаћемо бројач сугласника. На крају, исписаћемо укупан број сугласника.

```python
def je_samoglasnik(c):
    return c == 'a' or c == 'e' or c == 'i' or c == 'o' or c == 'u'

n = int(input())
s = input()

broj_suglasnika = 0

# Prolazak kroz svaki karakter u reci.
for c in s:
    if not je_samoglasnik(c):
        broj_suglasnika += 1

print(broj_suglasnika)
```

Задатак: Пресек интервала

Аутор: Огњен Тешић

Дат је природан број n ($2 \leq n \leq 1000$). Колико целих бројева се налази у пресеку затворених интервала $[a_1, b_1], [a_2, b_2], \dots, [a_n, b_n]$?

Опис улаза

На стандардном улазу се уноси природан број n.

У наредних n редова се уносе по два цела броја a_i и b_i ($a_i \leq b_i$). Сви бројеви који се уносе су по апсолутној вредности не већи од 1000.

Опис излаза

На стандардни излаз исписати тачно један број - број целих бројева који се налазе у пресеку.

Пример 1

Улаз	*Излаз*	*Објашњење*

`4`
`-32 70`
`5 8`
`4 9`
`6 75`
	`3`	Бројеви 6, 7 и 8 се налазе у сва четири затворена интервала.

Пример 2

Улаз	*Излаз*	*Објашњење*

`2`
`1 5`
`5 2024`
	`1`	Једини број који припада у оба затворена интервала је број 5.

Пример 3

Улаз	*Излаз*	*Објашњење*

`2`
`5 9`
`10 15`
	`0`	Не постоји број који се налази у оба интервала.

Решење

Решење задатка се своди на проналажење пресека затворених интервала. Да бисмо то одредили, потребно је да нађемо највећи број који је леви крај свих интервала, у ознаци max_a, и најмањи број који је десни крај свих интервала, у ознаци min_b.

Када добијемо те две вредности, можемо закључити следеће о пресеку: ако је $max\_a \leqslant min\_b$, онда је број целих бројева у том интервалу $min\_b - max\_a + 1$. Међутим, ако је $max\_a > min\_b$, то значи да интервали имају празан пресек, те је одговор 0.

```python
n = int(input())

# Највећи почетак интервала постављамо на минималну вредност (-1000)
max_a = -1000
# Најмањи крај интервала постављамо на максималну вредност (1000)
min_b = 1000

for _ in range(n):
    a, b = map(int, input().split())
    max_a = max(max_a, a)
    min_b = min(min_b, b)

# Ако постоји пресек (почетна тачка <= крајња тачка)
if max_a <= min_b:
    # Број целих бројева у пресеку је (min_b - max_a + 1)
    print(min_b - max_a + 1)
else:
    # Ако нема пресека, исписујемо 0
    print(0)
```

Задатак: Најдужи подниз са 3 парна броја

Аутор: Душан Попадић

Дат је низ природних бројева дужине n. Одредити дужину најдужег подниза (узастопних елемената) у коме се налази највише три парна броја.

Опис улаза

У првом реду стандардног улаза уноси се цео број n $(1 \leq n \leq 5 \cdot 10^5)$, дужина низа. У другом реду уноси се n целих бројева a_i $(1 \leq a_i \leq 10^9)$, елементи низа. Елементи низа су раздвојени размацима.

Опис излаза

Исписати на стандардни излаз дужину најдужег подниза у коме се налази највише три парна броја.

Подзадаци

- $n \leq 5 \cdot 10^3$ (40 поена)
- без додатних ограничења (60 поена)

Пример 1

Улаз	*Излаз*	*Објашњење*
`11` `1 2 4 6 1 3 8 10 5 7 2`	`7`	Најдужи подниз са највише три парна броја је $[1, 3, 8, 10, 5, 7, 2]$

Пример 2

Улаз	*Излаз*	*Објашњење*
`10` `2 7 6 8 10 1 3 5 3 12`	`8`	Најдужи подниз је $[7, 6, 8, 10, 1, 3, 5, 3]$.

Пример 3

Улаз	*Излаз*
`6` `2 1 4 3 13 8`	`6`

Решење

Идеја решења је да применимо технику покретног прозора (енгл. *sliding window*) да бисмо пронашли најдужи узастопни подниз који садржи највише три парна броја.

Ова техника функционише тако што се користе два показивача, леви (`l`) и десни (`d`), који иницијално показују на почетак низа. Показивач `d` се помера надесно, додајући нове елементе у прозор. При сваком кораку, ако је нови елемент парни број, бројимо га у променљивој `broj_parnih`.

Док је `broj_parnih` мањи или једнак 3, настављамо са ширењем прозора увећавањем показивача `d`. Међутим, када `broj_parnih` пређе 3, што значи да тренутни подниз садржи више парних бројева него што је дозвољено, померамо леви показивач `l` надесно све док поново не задовољимо услов са највише три парна броја у прозору. При сваком померању `l`, ако из прозора избацимо парни број, умањујемо `broj_parnih` за 1.

Током проласка кроз низ пратимо највећу дужину прозора који задовољава услове и чувамо ту вредност као `max_duzina`. Сложеност овог приступа је $\mathcal{O}(n)$, јер се сваки показивач кроз низ помера највише једном.

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

max_duzina = 0
l = 0
d = 0
broj_parnih = 0

while d < n:
    if elementi[d] % 2 == 0:
        broj_parnih += 1
    while broj_parnih > 3:
        if elementi[l] % 2 == 0:
          broj_parnih -= 1;
        l += 1
    max_duzina = max(max_duzina, d - l + 1)
    d += 1

print(max_duzina)
```

Задатак: Програмски језици

Аутор: Владимир Кузмановић

Перица се запослио као млади статистичар и као први задатак је добио да одреди резултате анкете о популарности програмских језика. У анкети је учествовало n испитаника и од сваког испитаника је захтевано да наведе програмски језик који најчешће користи, као и оцену коју би дао том језику.

Број програмских језика које испитаници могу навести у анкети није унапред познат, али је познато да се језици у анкети могу јављати више од једанпут. Након обраде свих одговора испитаника, Перица треба да одреди просечну оцену заокружену на две децимале за сваки програмски језик који су испитаници навели и да направи извештај у којем ће резултати бити сортирани азбучно по називу програмског језика.

Опис улаза

У првој линији стандардног улаза учитава се број испитаника n, $1 \leq n \leq 10^6$. Затим се у следећих n линија учитавају одговори испитаника. Сваки одговор садржи назив програмског језика и оцену раздвојене једним размаком. Назив програмског језика је реч дужине највише 7 слова, а оцена је цео број у интервалу $[0, 100]$.

Опис излаза

На стандардни излаз исписати резултате анкете сортиране азбучно по програмским језицима. У сваком реду стандардног излаза штампају се информације о једном програмском језику. За сваки програмски језик у истом реду исписати његов назив и просечну оцену заокружену на две децимале раздвојене једним размаком.

Пример

Улаз	*Излаз*
`10`	`C 85.00`
`Python 87`	`C# 87.00`
`C++ 89`	`C++ 92.00`
`Java 83`	`Java 86.00`
`C++ 95`	`Python 93.00`
`Python 100`	
`C 85`	
`Python 92`	
`Java 89`	
`C++ 92`	
`C# 87`	

Решење

Наивно, у програму бисмо могли да чувамо низ програмских језика и да сваком језику придружимо низ оцена које су му испитаници доделили. Имајући у виду да је сваки одговор облика *језик оцена*, да се језици могу понављати и да их корисник не мора уносити у сортираном поретку, лако се уочава да за сваки одговор морамо да проверимо да ли је неки претходни испитаник већ навео тај програмски језик или не. У случају да јесте, тада у низ оцена тог програмског језика треба да додамо нову оцену. У случају да није, тада у низ програмских језика треба да додамо тај нови језик и да му придружимо једноелементни низ са унетом оценом.

Иако једноставан, овакав начин попуњавања низа је врло неефикасан. Наиме, са сваким уносом морамо да претражујемо постојећи низ. Претраживање морамо да вршимо обичном линеарном претрагом, јер немамо гаранцију да ће наш низ бити сортиран. Попуњавање низа на овај начин је квадратне сложености и није погодно за велике димензије улаза.

С обзиром да наш низ није сортиран, пре исписивања резултата морамо да га сортирамо лексикографски по називима програмских језика и тек након тога да испишемо садржај низа. Сортирање можемо да извршимо неким ефикасним алгоритмом у логлинеарном времену, па сложеношћу програма доминира попуњавање низа. Након сортирања, просечне оцене можемо да рачунамо добро познатим алгоритмом пре самог штампања података о конкретном језику. Дакле, ако желимо ефикасно решење, морамо ефикасније да попуњавамо наш низ приликом учитавања резултата.

```python
# učitavamo broj ispitanika
n = int(input())

# lista odgovora je na početku prazna
# u listi čuvamo parove oblika
# (naziv jezika, lista ocena)
prog_jezici = []
# redom učitavamo odgovore
for i in range(0,n):
    linija = input().split(" ")
    jezik = linija[0]
    ocena = float(linija[1])
    # proveravamo da li smo jezik već ranije učitali
    indeks = -1
```

```
    for j in range(0, len(prog_jezici)):
        if prog_jezici[j][0] == jezik:
            indeks = j
            break
    # ako nismo, dodajemo novi jezik u listu odgovora
    if (indeks == -1):
        prog_jezici.append([jezik, [ocena]])
    # inače, dodajemo ocenu u listu ocena
    else:
        prog_jezici[j][1].append(ocena)

# sortiramo jezike prema nazivu
prog_jezici.sort(key = lambda x : x[0])

# i na kraju štampamo rezultat
for x in prog_jezici:
    # određujemo prosečnu ocenu
    s = sum(x[1])/len(x[1])
    # prikazujemo rezultat
    print(x[0], format(s, ".2f"))
```

Из анализе неефикасног решења, јасно је да попуњавање низа треба да учинимо ефикаснијим. Поред тога, рачунање просечне оцене можемо такође да учинимо меморијски ефикаснијим. Уместо да чувамо читав низ оцена, довољно је да за сваки програмски језик одржавамо текући збир оцена које су доделили корисници, као и укупан број оцена тог језика. На крају ћемо просечну оцену лако добити као količник та два броја.

Да бисмо попуњавање низа учинили ефикаснијим потребно је да пре свега учинимо претрагу раније унетих језика ефикаснијом и да по могућству избацимо накнадно сортирање. У програмском језику Пајтон можемо да користимо речник, тј. структуру података *dict* која упарује кључ са жељеном вредности. Као кључ у речнику ћемо користити назив програмског језика, а као вредност ћемо да користимо уређени пар чија ће прва координата бити текући збир оцена, а друга координата ће бити број оцена додељених програмском језику.

Интерно, колекција *dict* је хеш табела у којој се приликом одреживања хеш вредности користи управо кључ речника. Употребом хеш табеле гарантују се константна или скоро констатна времена за све уобичајене операције: додавање, ажурирање и брисање, што је значајно ефикасније од линеарног времена у наивном решењу. Идејно, попуњавање речника се уопште не разликује од попуњавања у наивном случају. Правилним избором структуре података којом представљамо информације у програму добили смо неупоредиво ефикасније решење. Укупна временска сложеност изградње речника у овом случају биће линеарна.

Речник у Пајтону није сортирана колекција, па пре сортирања прво морамо да сортирамо кључеве речника, тј. називе програмских језика. Сортирање можемо да извршимо ефикасним алгоритмом у логлинеарном времену. Након сортирања кључева, остаје нам само да испишемо резултате. Временска сложеност сортирања ће доминирати сложеношћу програма, па ће укупна временска сложеност решења бити логлинеарна.

```
# učitavamo broj ispitanika
n = int(input())
# recnik u kojem čuvamo programske jezike i rezultate anketa
```

```
# naziv jezika je ključ, a kao vrednost čuvaćemo uređeni par
# oblika (zbir ocena, broj ocena)
prog_jezici = {}
# učitavamo odgovore ispitanika
for i in range(0, n):
    linija = input().split(" ")
    jezik = linija[0]
    ocena = float(linija[1])
    # proveravamo da li se programski jezik već nalazi u rečniku
    unos = prog_jezici.get(jezik)
    # ako nije u rečniku, dodajemo ga
    if unos is None:
        prog_jezici[jezik] = (ocena, 1)
    # ako jeste, ažuriramo zbir i broj cena
    else:
        prog_jezici[jezik] = (unos[0] + ocena, unos[1] + 1)

# rečnik nije sortirana kolekcija, pa prvo moramo da sortiramo kljuceve
# i zatim prikažemo rezultate
for k in sorted(prog_jezici.keys()):
    t = prog_jezici[k]
    print(k, format(t[0]/t[1], ".2f"))
```

Задатак: Наелектрисање

Аутор: Љубомир Бановић

Сва материја у универзуму (па самим тим и у Периној соби) има свој коефицијент наелектрисања. Пера је недавно купио електрометар (уређај за мерење наелектрисања) којим је измерио наелектрисање сваког предмета у својој соби. Наелектрисање може бити позитивно, негативно или неутрално. Пера је одлучио да укупно наелектрисање његове собе треба да буде неутрално јер се то слаже уз неутралне боје зидова собе. Занима га на колико различитих начина може да одабере предмете које ће избацити из собе тако да укупно наелектрисање собе буде неутрално. Напиши програм који за дате коефицијенте наелектрисања сваког предмета одређује тај број начина.

Опис улаза

Са стандардног улаза се учитава број предмета у Периној соби n ($1 \leqslant n \leqslant 20$). Затим се у наредном реду учитава n целих бројева одвојених размаком који представљају коефицијенте наелектрисања предмета c_i ($-10^6 \leqslant c_i \leqslant 10^6$).

Опис излаза

На стандардни излаз исписат један број који представља на колико начина је могуће одабрати предмете које треба избацити из собе тако да наелектрисање у соби буде неутрално.

Подзадаци

- $n \leqslant 5$ (20 поена)
- без додатних ограничења (80 поена)

Пример 1

Улаз	*Излаз*	*Објашњење*
`5`	`6`	Могућа избацивања су:
`1 -2 3 0 -1`		`Избачено [1, -2, 3, 0, -1], остаје {}`
		`Избачено [1, -2, 3, _, -1], остаје {0}`
		`Избачено [_, -2, 3, 0,  _], остаје {1, -1}`
		`Избачено [_, -2, 3, _,  _], остаје {1, 0, -1}`
		`Избачено [1,  _, _, 0,  _], остаје {-2, 3, -1}`
		`Избачено [1,  _, _, _,  _], остаје {-2, 3, 0, -1}`

Пример 2

Улаз	*Излаз*	*Објашњење*
`4`	`4`	Могућа избацивања су:
`-2 4 -2 -2`		`Избачено [-2, 4, -2, -2], остаје {}`
		`Избачено [-2, _,  _,  _], остаје {4, -2, -2}`
		`Избачено [ _, _, -2,  _], остаје {-2, 4, -2}`
		`Избачено [ _, _,  _, -2], остаје {-2, 4, -2}`

Решење

Задатак се своди на то да се одреди број подскупова предмета у Периној соби чији збир наелектрисања износи 0. Решење користи рекурзивно набрајање да би се одредио број могућих подскупова чији збир наелектрисања износи 0. Функција `brojPodskupovaSaZbirom0` рекурзивно пролази кроз све предмете у соби и проверава сваки могући подскуп. За сваки предмет постоје две опције: додати га у подскуп (што утиче на тренутни збир) или га изоставити. Када се достигне крај списка предмета (услов `indeks == naelektrisanja.size()`), функција проверава да ли је збир тренутног подскупа једнак нули и враћа 1 ако јесте, односно 0 ако није. На овај начин, функција враћа укупан број подскупова са неутралним наелектрисањем.

```
# rekurzivna funkcija kojom se nabrajaju svi podskupovi niza naelektrisanja
# i broje oni ciji je zbir jednak 0
def brojPodskupovaSaZbirom0_(naelektrisanja, indeks, zbir):
    if indeks == len(naelektrisanja):
        if zbir == 0:
            return 1
        else:
            return 0
    return brojPodskupovaSaZbirom0_(naelektrisanja, indeks + 1, \
                                    zbir + naelektrisanja[indeks]) + \
           brojPodskupovaSaZbirom0_(naelektrisanja, indeks + 1, zbir)

def brojPodskupovaSaZbirom0(naelektrisanja):
    indeks = 0
    zbir = 0
    return brojPodskupovaSaZbirom0_(naelektrisanja, indeks, zbir)

n = int(input())
naelektrisanja = [int(x) for x in input().split()]
print(brojPodskupovaSaZbirom0(naelektrisanja))
```

Задатак: ПИН апликације од ПИН-а телефона

Аутор: Филип Марић

Студент треба да одреди четвороцифрени ПИН за нову апликацију, али није сигуран да ће га добро запамтити. Зато је смислио да тај ПИН израчуна на основу њему добро познатог ПИН-а телефона. Сабраће ПИН свог телефона, са бројем који добије када том ПИН-у замени прву и последњу цифру и са бројем који добије када том ПИН-у замени две средишње цифре. Нови ПИН ће бити последње 4 цифре тако добијеног збира. На пример, ако је ПИН телефона 1234, сабраће бројеве 1234 + 4231 + 1324 и добиће збир 6789, који је уједно нови ПИН.

Опис улаза

Са стандардног улаза се уноси четвороцифрени ПИН телефона (може да има и водеће нуле).

Опис излаза

На стандардни излаз исписати четвороцифрени ПИН апликације (укључујући и водеће нуле ако их има).

Напомена: приказ водећих нула у броју `p` се у језику C++ може постићи наредбом

```
cout << setw(4) << setfill('0') << p << endl;
```

а у језику Python наредбом

```
print(str(p).zfill(4))
```

Пример 1

Улаз	*Излаз*
`1234`	`6789`

Пример 2

Улаз	*Излаз*	*Објашњење*
`9999`	`9997`	Ако је ПИН телефона 9999, тада ће добити збир 9999 + 9999 + 9999 = 29997, па је нови ПИН 9997. Написати програм који на основу унетог четвороцифреног ПИН-а телефона одређује нови четвороцифрени ПИН који ће се користити за апликацију.

Решење

Опис главног решења

Потребно је да из унетог четвороцифреног ПИНа телефона извучемо све четири цифре. Након тога рачунамо два нова броја: један добијен заменом прве и последње цифре, а други заменом две средишње цифре. Збир оригиналног ПИН-а, првог и другог броја се рачуна, а нови ПИН за апликацију представља последње четири цифре тог збира (водећи рачуна о водећим нулама).

```
pinTelefona = int(input())

#odredjujemo cifre PIN-a telefona
c0 = pinTelefona % 10
c1 = (pinTelefona // 10) % 10
c2 = (pinTelefona // 100) % 10
c3 = (pinTelefona // 1000) % 10
```

```
# broj dobijen zamenom cifre jedinica i hiljada
zamenaSpoljasnjih = 1000*c0 + 100*c2 + 10*c1 + c3
# broj dobijen zamenom cifre desetica i stotica
zamenaUnutrasnjih = 1000*c3 + 100*c1 + 10*c2 + c0

# PIN aplikacije su poslednje 4 cifre zbira
pinAplikacije = (pinTelefona + zamenaSpoljasnjih + zamenaUnutrasnjih) % 10000

# ispisujemo cetvorocifreni PIN aplikacije sa eventualnim vodecim nulama
print(str(pinAplikacije).zfill(4))
```

Задатак: Број јаких лозинки

Аутор: Филип Марић

Лозинка је јака ако има бар 8 карактера, бар једно мало, бар једно велико слово, бар једну цифру и бар један специјални карактер (карактер који није ни слово ни цифра). Напиши програм који одређује колико унетих лозинки је јако.

Опис улаза

Са стандардног улаза се учитава број лозинки n ($1 \leq n \leq 100$), а затим n лозинки (свака у посебном реду).

Опис излаза

На стандардни излаз исписати број јаких лозинки.

Пример

Улаз	*Излаз*	*Објашњење*

`5`
`Zdravo123!`
`3@abC`
`petlja17`
`GEJMERI_2024`
`music_Shop-85`
	`2`	Јаке су само прва и последња лозинка. Друга је прекратка, У трећој недостају велико слово и специјални карактер, а у четвртој недостаје мало слово.

Решење

Можемо дефинисати посебну функцију којом се проверава да ли је лозинка јака. Пролазимо кроз карактере дате ниске и за сваки одређујемо да ли је мало слово, велико слово, цифра или ништа од тога. Користимо четири логичке променљиве којима региструјемо да ли се појавио неки крактер одговарајуће врсте. Њих на почетку постављамо на вредност нетачно, а ако се појави одговарајући карактер, мењамо им вредност на тачно. Лозинка је јака ако и само ако након проласка кроз све карактере све променљиве имају вредност тачно и ако је лозинка довољно дугачка (проверу дужине је могуће урадити и на почетку функције).

У главном програму учитавамо n лозинки, за сваку проверавамо да ли је јака и ако јесте увећавамо вредност бројача (који је на почетку иницијализован на нулу). Након обраде свих лозинки исписујемо вредност бројача.

```
def jakaLozinka(lozinka):
    if len(lozinka) < 8:
```

```
        return False
    malo = False
    veliko = False
    cifra = False
    specijalni = False
    for c in lozinka:
        if c.islower():
            malo = True
        elif c.isupper():
            veliko = True
        elif c.isdigit():
            cifra = True
        else:
            specijalni = True
    return malo and veliko and cifra and specijalni

n = int(input())
broj_jakih = 0
for i in range(n):
    lozinka = input()
    if jakaLozinka(lozinka):
        broj_jakih += 1

print(broj_jakih)
```

Задатак: Чудна осмосмерка

Аутор: Душан Попадић, Филип Марић

Осмосмерка је игра у којој је дата табела димензија $n \times n$ попуњена великим словима енглеске абецеде и још k речи са стране. Циљ игре је да се задате речи пронађу у табели. Реч у табели може бити у био ком од осам смерова: нагоре, надоле, улево, удесно или дијагонално на све 4 стране (ка горе лево, ка горе десно, ка доле лево и ка доле десно). Реч може почети од било ког слова у табели и не мора попунити цео ред, колону или дијагоналу.

Деда Васа је купио часопис у коме се налази чудна осмосмерка у којој је могуће да постоје речи које се не налазе у табели, чак и да се нека реч налази на више места у табели. Помозите деда Васи тако што ћете пребројати колико се задатих речи може наћи у табели поштујући правила осмосмерке.

Опис улаза

У првом реду стандардног улаза се налазе два природна броја, не већа од 100, раздвојена размацима: n и k. У наредних n редова се налази по једна реч од n великих латиничних слова енглеске абецеде. Свака реч представља по један ред табеле. У наредних k редова се налази по једна реч састављена од великих слова енглеске абецеде. За ове речи је потребно проверити да ли се налазе у табели.

Ограничења

У тест примерима вредним 50 поена важи да ако се реч појављује у табели, сигурно се појављује водоравно или усправно - односно у овим тест примерима није потребно проверавати дијагонале.

Опис излаза

У јединoм реду стандардног излаза исписати један природан број који представља колико се речи (од k задатих) налази у табели.

Пример 1

Улаз	*Излаз*	*Објашњење*
5 6 SMERZ QUXZX XNPXQ UJKAX XAWMR UJKA PERA MUNJA MIRA SMER SUPA	4	У табели се налази 4 речи: UJKA (4. ред од 1. слова удесно), MUNJA (2. колона, од 1. слова надоле), SMER (1. ред од 1. слова улево) и SUPA (1. ред од 1. слова дијагонално доле десно).

Пример 2

Улаз	*Излаз*	*Објашњење*
4 7 AMQE MROO XTOP VSAZ STRM KOSA AT TOP KRV ZORA TRUBADUR	4	У табели се налазе 4 речи: STRM (4. ред, од 2. слова нагоре), TOP (3. ред од 2. слова удесно), ZORA (4. ред, од 4. слова дијагонално горе лево) и AT (4. ред, од 3. слова дијагонално горе лево).

Решење

С обзиром на то да су улазни подаци релативно мали, у овом задатку није потребно посебно водити рачуна о сложености, већ можемо имплементирати једноставно решење. За сваку реч пролазимо кроз целу матрицу и из сваког поља крећемо у сваком од 8 смерова и проверавамо да ли постоји поклапање са нашом речи.

```
smerR = [0, 0, 1, -1, 1, -1, 1, -1]
smerK = [1, -1, 0, 0, 1, -1, -1, 1]

def granice(i, j, n, m):
    return 0 <= i < n and 0 <= j < m
```

```
def provera(tabela, rec, n):
    for i in range(n):
        for j in range(n):
            for s in range(8):
                x, y, r = i, j, 0
                while granice(x, y, n, n) and tabela[x][y] == rec[r]:
                    x += smerR[s]
                    y += smerK[s]
                    r += 1
                    if r == len(rec):
                        return 1
    return 0


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

tabela = [input().strip() for _ in range(n)]

brReci = 0

for _ in range(k):
    rec = input().strip()
    brReci += provera(tabela, rec, n)

print(brReci)
```

Задатак: Астроном

Аутори: Немања Мајски, Огњен Тешић

Лука је астроном аматер и страствено прати кретање небеских тела на свом телескопу. Недавно је приметио низ чудних промена у сјају једне звезде. Наиме, забележио је низ вредности који представљају промене у сјају звезде током ноћи. Свака вредност може бити позитивна (повећање сјаја), негативна (смањење сјаја) или нула (нема промене).

Међутим, Лука је приметио да се у неким деловима ноћи дешава нешто занимљиво. Током одређених периода, све промене у сјају се међусобно пониште, што значи да се сјај звезде врати на исти ниво као и на почетку тог периода. Он верује да управо ти периоди крију посебне тајне о понашању звезда.

Ваш задатак је да помогнете Луки да преброји све могуће интервале у којима се промене сјаја међусобно поништавају, тј. враћају на исту вредност као на почетку тог периода.

Опис улаза

Први ред стандардног улаза садржи природан број $N \leqslant 2 \cdot 10^5$ - број забележених промена у сјају звезде током ноћи.

Други ред стандардног улаза садржи N размаком раздвојених целих бројева који представљају промене у сјају звезде (сви бројеви су по апсолутној вредности не већи од 10^9).

Опис излаза

На стандардни излаз треба исписати тачно један цео број - број тражених интервала.

Подзадаци

- $N \leqslant 100$ - 20 поена;
- $N \leqslant 5000$ - 30 поена;
- без додатних ограничења - 50 поена.

Пример 1

Улаз	*Излаз*	*Објашњење*
`7` `3 4 -7 1 2 -1 6`	`2`	Интервали за које ово важи су $[3, 4, -7]$ и $[4, -7, 1, 2]$.

Пример 2

Улаз	*Излаз*	*Објашњење*
`3` `1 0 1`	`1`	Једини интервал за који ово важи је $[0]$.

Решење

Задатак је да пронађемо број интервала у низу у којима је збир елемената једнак нули. Задатак ћемо решити користећи префиксне суме. Префиксна сума на индексу i представља збир свих елемената у низу од почетка до тог индекса. Ако два елемента у низу имају исту префиксну суму, онда збир елемената између њих мора бити нула. Префиксне суме се лако рачунају на следећи начин: за сваки елемент у низу, додајемо његову вредност на тренутну префиксну суму `current_sum`.

Чување броја појављивања префиксних сума у мапи/речнику: користимо мапу `prefix_sume` која прати колико пута је нека префиксна сума већ виђена. На почетку, иницијализујемо `prefix_sume[0] = 1` јер се префиксна сума 0 подразумевано појављује једном (пре почетка низа). То нам омогућава да рачунамо интервале који почињу од првог елемента и имају збир 0.

Бројање интервала са збиром 0: ако се `current_sum` већ налази у мапи/речнику `prefix_sume`, то значи да постоје поднизеви који имају збир 0 и завршавају се на тренутној позицији. Тиме повећавамо `broj_intervala` за број појављивања те префиксне суме (што је у мапи `prefix_sume[current_sum]`).

Ажурирање мапе/речника са новом појавом префиксне суме: на крају сваке итерације, повећавамо број појављивања `current_sum` у мапи `prefix_sume` за 1.

```python
def broj_intervala(n, niz):
    prefix_sume = {0: 1}  # Prefiks suma 0 se pojavljuje jednom na početku
    trenutna_suma = 0
    broj_intervala = 0

    for x in niz:
        trenutna_suma += x

        if trenutna_suma in prefix_sume:
            broj_intervala += prefix_sume[trenutna_suma]

        prefix_sume[trenutna_suma] = prefix_sume.get(trenutna_suma, 0) + 1
```

```
    return broj_intervala

n = int(input())
niz = list(map(int, input().split()))

print(broj_intervala(n, niz))
```

Глава 2

2. круг квалификација

Задатак: Куповина прибора

Аутор: Филип Марић

Миљана жели да купи школски прибор. Од родитеља је добила D динара. Купила је x оловака по цени од X динара и y гумица по цени од Y динара (за то је сигурно имала довољно новца). Ако свеска кошта Z динара, напиши програм који одређује колико највише свесака Миљана може да купи од преосталог новца.

Опис улаза

Са стандардног улаза се учитавају природни бројеви D, x, X, y, Y и Z (сваки у посебном реду).

Опис излаза

На стандрадни излаз исписати тражени највећи број свески које Миљана може да купи.

Пример 1

Улаз	*Излаз*	*Објашњење*

`1000`
`3`
`45`
`7`
`30`
`100`
	`6`	Након куповине оловака и гумица, Миљани је остало 655 динара (оловке су коштале 135 динара, а гумице 210 динара). За 655 динара може да купи највише 6 свески које коштају 100 динара.

Пример 2

Улаз	*Излаз*

`800`
`10`
`10`
`10`
`20`
`100`
	`5`

Решење

Након куповине оловака и гумица Миљани преостаје $D - (x \cdot X + y \cdot Y)$ динара. Највећи број свесака које може купити се добија целобројним дељењем тог износа са ценом свеске Z.

```
D = int(input())
x = int(input())
X = int(input())
y = int(input())
Y = int(input())
Z = int(input())
z = (D - (x*X + y*Y)) // Z
print(z)
```

Задатак: Померање часовника

Аутор: Алекса Милисављевић

У складу са преласком на зимско рачунање времена, 27. октобра се у 03:00 сат померио на 02:00. Перица је тог дана погледао на сат и видео да је тренутно h:m, тј. h сати и m минута. Одредити колико је сати је било пре x минута. Гарантује се да је решење једнозначно. Такође се гарантује и да је пре x минута био 27. октобар.

Гарантује се да померање часовника не утиче на резултат у 60% примера.

Опис улаза

У првој линији стандардног улаза дато је тренутно време, у формату hh:mm и позитиван цео број x.

Опис излаза

У првој и јединој линији стандардног излаза исписати време пре x минута у формату hh:mm.

Пример 1

Улаз	*Излаз*
14:47 43	14:04

Пример 2

Улаз	*Излаз*
05:47 253	02:34

Решење

Опис главног решења

У овом блоку се описује главно решење задатка.

```
[t, tx] = input().split(' ')
x = int(tx)
[th, tm] = t.split(':')
h = int(th)
m = int(tm)
m += h * 60
if h >= 3:
   m += 60
m -= x
h = m // 60
```

```
m = m % 60
if h >= 3:
    h-=1

print('{:02}:{:02}'.format(h,m))
```

Задатак: Три на три

Аутор: Милан Вугделија

Поклопац може да поклопи лонац ако је величина поклопца иста или већа од величине лонца. Дате су величине три поклопца и три лонца. Колико највише лонаца може да се поклопи?

Опис улаза

У првом реду стандардног улаза су целобројне величине поклопаца у опадајућем поретку (с тим да два поклопца могу имати исту величину).

У другом реду су целобројне величине лонаца у опадајућем поретку (с тим да два лонца могу имати исту величину).

Сви бројеви су природни и мањи од 1000. Бројеви у истом реду су раздвојени по једним размаком.

Опис излаза

На стандардни излаз исписати један природан цео број, број могућих поклапања.

Пример 1

Улаз	*Излаз*	*Објашњење*
`5 4 2` `4 4 2`	`3`	Први поклопац поклапа први лонац, други поклопац поклапа други лонац и трећи поклопац поклапа трећи лонац.

Пример 2

Улаз	*Излаз*	*Објашњење*
`200 150 100` `300 200 100`	`2`	Пошто је први лонац већи од свих поклопаца, њега не можемо да поклопимо. Први поклопац стављамо на други лонац, а други или трећи поклопац стављамо на трећи лонац. Дакле успели смо да поклопимо 2 лонца: други и трећи.

Пример 3

Улаз	*Излаз*	*Објашњење*
`300 200 100` `600 500 400`	`0`	Сви лонци су већи од свих поклопаца тако да ниједан не може бити поклопљен.

Решење

```
a3, a2, a1 = [int(x) for x in input().split()]
b3, b2, b1 = [int(x) for x in input().split()]

br = 0

if a3 >= b3:
    br+=1
```

```
else:
    a1, a2 = a2, a3

if a2 >= b2:
    br+=1
else:
    a1 = a2

if a1 >= b1:
    br+=1

print(br)
```

Задатак: Рођендански поклон

Аутор: Љубомир Бановић

Ивану се ближи рођендан, те је Павле одлучио да га изненади тако што ће му поклонити књигу - дужине a, висине b и ширине c. Како би поклон био занимљивији, Павле је одлучио да књигу упакује у украсну кутију. У књижари је на располагању n различитих украсних кутија, при чему је кутија i дужине x, ширине y и висине z. Павле жели да зна у колико украсних кутија књига може да стане, како би могао да изабере једну од њих за свој поклон. Књига стаје у украсну кутију, ако су дужина, ширина и висина књиге мањи или једнаки дужини, ширини и висини кутије. Такође, кутије није могуће ротирати.

Опис улаза

У првом реду стандардног улаза, уносе се три цела броја: a, b и c $(1 \leq a, b, c \leq 10000)$, димензије књиге.

У другом реду стандардног улаза се уноси број n $(1 \leq n \leq 10000)$, број украсних кутија у књижари.

У наредних n линија се уносе три броја: x, y и z $(1 \leq x, y, z \leq 10000)$, димензије украсне кутије.

Опис излаза

На стандардни излаз исписати један цео број, број украсних кутија у које може да стане књига.

Пример

Улаз	*Излаз*	*Објашњење*

`5 7 3`
`4`
`6 8 4`
`5 7 3`
`10 10 10`
`4 7 3`
	`3`	Књига може да стане у прве три кутије, док књига не може да стане у четврту кутију јер је дужа од ње.

Решење

Одржавамо променљиву `resenje` која ће представљати број кутија у које може да стане књига и иницијализујемо је на вредност 0. У петљи у којој се уносе димензије кутија, проверавамо да ли књига стаје у тренутну кутију. Ако стаје, увећавамо променљиву `resenje` за један. Након уноса димензија свих кутија, променљива `resenje` биће наше решење, те је потребно исписати њену вредност.

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

resenje = 0

for i in range(n):
    x, y, z = map(int, input().split())

    if a <= x and b <= y and c <= z:
        resenje += 1

print(resenje)
```

Задатак: Учешљавање аутомобила

Аутор: Филип Марић

Познати су регистарски бројеви аутомобила у две колоне. По закону се приликом спајања у једну траку аутомобили пропуштају наизменично. Написати програм који одређује редослед аутомобила након спајања у једну траку.

Опис улаза

Са стандардног улаза се учитава број аутомобила у левој колони (природан број између 0 и 100), а затим регистарске таблице свих аутомобила у левој колони (свака је наведена у посебном реду). Затим се на исти начин учитавају подаци о аутомобилима из десне колоне.

Опис излаза

На стандардни излаз исписати регистарске таблице у обједињеној колони (прво пролази први аутомобил из леве колоне).

Пример 1

Улаз

```
4
BG732LM
BG311AD
KG241ED
SO919MN
2
NS111PP
NI829RL
```

Излаз

```
BG732LM
NS111PP
BG311AD
NI829RL
KG241ED
SO919MN
```

Пример 2

Улаз

```
2
PI732ED
SD423AB
2
SU423WJ
VA929JK
```

Излаз

```
PI732ED
SU423WJ
SD423AB
VA929JK
```

Решење

Све регистарске таблице ћемо учитати у два вектора, а затим ћемо штампати по један елемент из оба вектора, све док не стигнемо до краја једног од њих. Након тога ћемо исписати преостале елементе из другог вектора (ако их има).

```
broj_levo = int(input())
levo = []
for i in range(broj_levo):
    levo.append(input())

broj_desno = int(input())
desno = []
for i in range(broj_desno):
    desno.append(input())

l = 0
d = 0
while l < broj_levo and d < broj_desno:
    print(levo[l])
    print(desno[d])
    l += 1
    d += 1

while l < broj_levo:
    print(levo[l])
    l += 1

while d < broj_desno:
    print(desno[d])
    d += 1
```

Задатак: Пласман

Аутор: Филип Марић

На студентском хакатону студентски тимови су освајали поене решавајући програмерске изазове. На крају такмичења познат је списак тимова и њихових освојених поена. Написати програм који сортира тимове по поенима, опадајући и исписује листу, али тако да се испред сваког тима одреди његов пласман на Хакатону. Ако два или више тимова имају исти број поена они деле исти пласман.

Опис улаза

Са стандардног улаза се учитава број тимова n ($1 \leq n \leq 50000$). У наредних n редова наведено је име тима и освојен број поена (цео број између 0 и 100).

Опис излаза

На стандардни излаз исписати сортирани списак. У сваком реду исписати пласман, затим назив тима и на крају освојени број поена.

Пример

Улаз	*Излаз*
`4`	`1. matematicari123 45`
`hakeri 38`	`2. hakeri 38`
`matematicari123 45`	`2. zikini_petlici 38`
`zikini_petlici 38`	`4. pythonovci 21`
`pythonovci 21`	

Решење

Податке о сваком тиму можемо представити структуром (или уређеним паром). Низ структура можемо сортирати опадајући по броју поена, тако што ћемо употребити библиотечку функцију `sort` којој ћемо као трећи аргумент проследити функцију која пореди две структуре тако што пореди број поена сачуван у њима.

Након сортирања ћемо исписати низ, при чему пажљиво треба да бројимо пласман. Први тим у низу сигурно има редни број 1. За наредне тимове постоје две могућности:

- ако им је број поена једнак претходном тиму, тада имају и исти пласман.
- у супротном је пласман једнак редном броју у низу увећаном за 1 (јер се тимови броје од 1, а елементи низа од 0).

```
n = int(input())
timovi = []
for i in range(n):
    str1, str2 = input().split()
    timovi.append((int(str2), str1))

timovi.sort(reverse=True)

plasman = 1
print(plasman, ". ", timovi[0][1], " ", timovi[0][0], sep="")
for i in range(1, n):
    if timovi[i][0] < timovi[i-1][0]:
        plasman = i + 1
    print(plasman, ". ", timovi[i][1], " ", timovi[i][0], sep="")
```

Задатак: Потребан број петица

Аутор: Душан Попадић

Немања има тренутно 4 оцене из математике. Колико је још потребно да добије петица како би имао закључено 5 на крају полугодишта?

Опис улаза

Уносе се четири броја (од 1 до 5), сваки у посебном реду који означавају 4 оцене које Немања тренутно има из математике.

Опис излаза

У једином реду стандардног излаза исписати број потребних петица.

Пример 1

Улаз	*Излаз*	*Објашњење*
3 4 5 5	2	Немања тренутно има просек 4,25 из математике, а ако добије још две петице имаће $(3+4+5+5+5+5)/6 = 4,5$ што је довољно да има закључено 5.

Пример 2

Улаз	*Излаз*	*Објашњење*
1 1 3 4	18	Немања тренутно има просек 2,25 из математике, а ако добије још 18 петица имаће $(1+1+3+4+18*5)/22 = 4,5$ што је довољно да има закључено 5.

Пример 3

Улаз	*Излаз*	*Објашњење*
5 5 5 4	0	Немања тренутно има просек 4,75 из математике што му је већ довољно за закључену петицу, па не мора да добије више ниједну.

Решење

Да би Немања имао закључено 5 из математике мора имати просек који је најмање 4,5. Немања тренутно има 4 оцене, означимо њихов збир са $zbir$. Он треба да добије још неки број петица (који ми треба да одредимо) – тај број ћемо означити са x. Нови збир Немањиних оцена ће бити $z = zbir + 5 \cdot x$, а имаће укупно $n = 4 + x$ оцена (4 које је имао раније и још x петица). Његов нови просек је

$$\frac{z}{n} = \frac{zbir + 5 \cdot x}{4 + x}$$

и он мора бити најмање 4,5. Дакле, потребно је решити неједначину

$$(zbir + 5 \cdot x)(4 + x) \geq 4{,}5$$

по непознатој x ($zbir$ нам је познат – то је збир Немањине прве 4 оцене које учитавамо са улаза). Решење ове једначине је $x \geq 2 \cdot (18 - zbir)$. Приметите да је могуће да десна страна неједнакости буде мања од 0 (ако је $zbir > 18$) што ће се десити ако је Немањин иницијални просек већ довољан за 5 ($18 = 4,5 \cdot 4$). У том случају Немањи више не треба ниједна петица, па је решење 0.

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

zbir = a + b + c + d

brojPetica = max(0, 2*(18-zbir))
```

```
print(brojPetica)
```

Задатак: Зиплајн

Аутори: Милан Вуйгделија, Душан Пойадић

На два стуба чије су висине h_1, h_2, треба окачити сајлу за прелетање. Разлика висина крајева сајле треба да је најмање a, а највише b метара (у тестовима ће бити $a \leqslant b$). Крајеви сајле морају бити на висини од бар c метара. Која је највећа могућа висина вишег краја сајле?

Опис улаза

У пет редова стандардног улаза налази се по један позитиван цео број, мањи или једнак 200. То су редом бројеви h_1, h_2, a, b, c.

Опис излаза

На стандардни излаз исписати један цео број, тражену висину. Ако сајла не може да се постави у складу са условима, исписати 0.

Пример 1

Улаз	*Излаз*	*Објашњење*

`3`
`6`
`1`
`2`
`2`
	`5`	Ако нижи крај сајле буде окачен на самом врху нижег стуба, он ће бити на висини од 3m. Виши крај може да буде највише 2m изнад тога, што је 5m.

Пример 2

Улаз	*Излаз*	*Објашњење*

`7`
`5`
`3`
`100`
`5`
	`0`	Пошто висина сајле мора да буде бар 5m, на стубу од 5m сајла може да се окачи једино на врху стуба. Тада, чак и да други крај окачимо на врх другог стуба, тј. на висини од 7m, разлика висина не би била довољно велика (била би 2m, а потребно је бар 3m). Према томе, није могуће испунити све услове.

Решење

Да би анализа била једноставнија, претпоставићемо да је $h_1 < h_2$ (ако није, можемо да им разменимо вредности). Такође, није потребно да разматрамо случајеве када је виши крај сајле на нижем стубу. Заиста, ако је такво постављање сајле и могуће, сајла може да се постави и обрнуто, са крајевима на истим висинама, али са вишим крајем на вишем стубу. То значи да ништа не пропуштамо ако разматрамо само постављања сајле тако да је виши крај на вишем стубу.

Да би постављање сајле било могуће, потребно је и довољно да нижи стуб буде висок бар c, а да виши стуб буде висок бар $c + a$. Ако ови услови нису испуњени, треба исписати нулу, а ако су испуњени, израчунаћемо највећу могућу висину вишег краја.

Виши крај сајле не можемо да окачимо изнад висине h_2, али ни изнад висине $h_1 + b$. Према томе, највећа висина на коју можемо да окачимо виши крај је $min(h_1 + b, h_2)$.

```
h1 = int(input())
h2 = int(input())
a = int(input())
b = int(input())
c = int(input())

if h1 > h2:
    h1, h2 = h2, h1

if h1 >= c and h2 >= c+a:
    print(min(h2, h1+b))
else:
    print(0)
```

Задатак: Харсхад бројеви 3

Аутор: Душан Попадић

Харсхад бројеви су они бројеви који су дељиви својим збиром цифара. Име долази из Санскрита и на српски бисмо могли да преведемо као бројеви који доносе радост (harṣa (радост) + da (давати)). Дат је низ упита у којима се задају два броја a и b и за сваки треба исписати колико има харсхад бројева у интервалу $[a, b]$.

Опис улаза

У првом реду стандардног улаза се налази број упита q ($1 \leq q \leq 10^5$). У наредних q редова се налазе два по природна броја a и b ($1 \leq a \leq b \leq 10^6$) раздвојена размаком.

Додатна ограничења

- У тест примерима вредним 50 поена важиће $q \leq 1000$ и $b - a \leq 1000$.

Опис излаза

У q редова стандардног излаза редом исписати број харсхад бројева за сваки од упита.

Пример

Улаз	*Излаз*	*Објашњење*

`2`
`7 19`
`57 59`
	`6`
	`0`
		У интервалу [7, 19] има 6 бројева који су дељиви својим збиром цифара: 7, 8, 9, 10, 12 и 18. У интервалу [57, 59] нема бројева који су дељиви својим збиром цифара.

Решење

Наивно решење

Најједноставније решење је да се прође кроз све бројеве од a до b и за сваки провери да ли је харсхад број. Потребно је написати функцију за одређивање збира цифара датог броја и онда је позивати за сваки број између a и b и просто проверити да ли је сваки од тих бројева дељив својим збиром цифара Успут бројимо колико има таквих бројева и на крају испишемо резултат.

Одређивање збира цифара броја се ради тако што се са броја "скидају" цифре са десна на лево и додају у збир. Последњу цифру броја увек можемо добити као остатак тог броја при дељењу са 10 (нпр. остатак при дељењу 1546 са 10 је 6). Након што последњу цифру додамо у збир, треба да је уклонимо како она са њене леве стране била последња. То радимо тако што број целобројно поделимо са 10 (1546 целобројно подељено са 10 је 154). Овај поступак понављамо док наш број не постане 0.

Анализа сложености

За сваки упит ми морамо да прођемо кроз све бројеве између a и b и за сваки да одредимо збир цифара. Одређивање збира цифара је логаритамске сложености, па сложеност једног упита можемо проценити као $O(B \cdot \log(B)$ где B представља највећу вредност коју може имати број b. Оваквих упита има q, па је укупна сложеност $O(q \cdot B \cdot \log(B))$. Ово решење доноси 50 поена.

```python
def zbir_cifara(n):
    zbir = 0
    while n > 0:
        zbir += n % 10  # izdvajanje poslednje cifre
        n //= 10  # uklanjanje poslednje cifre
    return zbir

# Unos broja upita
q = int(input())

# Obrada upita
for _ in range(q):
    a, b = map(int, input().split())
    broj = 0
    for i in range(a, b + 1):
        zbirC = zbir_cifara(i)
        if i % zbirC == 0:
            broj += 1
    print(broj)
```

Решење помоћу префиксних збирова

Можемо приметити да у претходном решењу може да се деси да много пута за исти број проверавамо да ли је харсхад, али и да много пута за исти опсег бројева проверавамо колико има Харсхад бројева у њему. На пример ако је први упит од 10 до 100000, а други од 30 до 200000, ми смо за опсег [30, 200000] два пута проверили колико има харсхад бројева у њему, а то није мало посла. Ово можемо побољшати ако користимо принцип инкременталности и префиксне збирове. Наиме, ако обележимо број харсхад бројева од 1 до 200000 са h_200000, а број харсхад бројева од 1 до 29 са h_29, лако можемо да срачунамо да је број харсхад бројева од 30 до 200000 заправо $h_200000 - h_29$ - то су сви они харсхад бројеви који су мањи или једнаки 200000, али морамо да уклонимо оне који су мањи од 30 (тј. оне који су мањи или једнаки 29).

Када приметимо ово што је горе описано, можемо унапред одредити колико има харсхад бројева од 1 до сваког могућег броја (тј. до 1000000 јер нам је то задато као горње ограничење за вредност b), односно одређујемо све вредности h_1, h_2, ..., h_milion. Након тога, када год нам се појави упит за број харсхад бројева из опсега [a, b], њега рачунамо као $h_b - h_{a-1}$. За одређивање свих вредности h_i такође користимо принцип инкременталности. Ако неки број i није харсхад број, тада је вредност $h_i = h_{i-1}$, тј. број харсхад бројева до i је исти као број харсхад бројева до $i-1$ јер сам број i није ништа променио. Са друге стране, ако број i јесте харсхад, онда је $h_i = h_{i-1} + 1$, односно број харсхад бројева до i је за 1 већи него до $i-1$.

Анализа сложености

Као што смо већ рекли, одговор на упит је само одузимање два број, тј. сложеност му је константна $O(1)$, па је сложеност одговарања на упите једнака броју упита $O(q)$. Наравно, не смемо заборавити сложеност израчунавања вредности h_i. Треба проћи кроз све бројеве од 1 до B (B је максимална вредност коју b може имати) и за сваки проверити да ли је харсхад, што је сложености $O(B \cdot \log(B))$. Укупна сложеност је $O(q + B \cdot log(B)$.

```python
def zbir_cifara(n):
    zbir = 0
    while n > 0:
        zbir += n % 10  # izdvajanje poslednje cifre
        n //= 10  # uklanjanje poslednje cifre
    return zbir

# Kreiranje niza brojHarshad
brojHarshad = [0] * (1000001)

# Popunjavanje niza brojHarshad
for i in range(1, 1000001):
    zbirC = zbir_cifara(i)
    brojHarshad[i] = brojHarshad[i - 1]
    if i % zbirC == 0:
        brojHarshad[i] += 1

# Unos broja upita
q = int(input())

# Obrada upita
for _ in range(q):
    a, b = map(int, input().split())
    print(brojHarshad[b] - brojHarshad[a - 1])
```

Задатак: Паковање кутија

Аутор: Јелена Петровић

Миша је одлучио да ове године направи много новогодишњих пакетића како би обрадовао децу. Поклоне је већ купио и остало је још да их упакује, за шта му је потребно што више кутија. Миша је кренуо бициклом до оближње продавнице како би купио кутије. У продавници је Миша измерио све кутије и закључио да би свака од њих стала у његову корпу од бицикла, али никоје две не би стале у корпу једна поред друге. Миша се досетио генијалне идеје - изабраће кутије тако да може да их стави једну у другу и да сваку затвори поклопцем одозго.

Миша вам је дао списак димензија свих кутија и замолио вас је да му помогнете да одреди колико највише кутија може да понесе. Свака кутија је квадар одређен дужином, ширином и висином. Формалније, треба наћи највећи број m тако да постоји уређена m-торка различитих кутија $k_1, k_2, \ldots, k_m$, тако да кутија k_j може да стане у кутију k_{j+1} за свако $1 \leq j < m$. Једна кутија се може ставити у другу само ако су јој све три димензије мање или једнаке, при чему је дозвољено окренути кутију тако да се ширина и дужина замене, остале ротације нису могуће због поклопца који треба да остане горе.

Могуће је да две кутије имају исте димензије. Оне се сматрају различитим кутијама и може једна да се стави унутар друге.

Опис улаза

У првој линији улаза је дан број n - број кутија у продавници.

У следећих n линија се налазе по три природна броја w_i, d_i, h_i - ширина, дужина и висина i-те кутије.

Опис излаза

У првој и јединој линији исписати један број - највећи број кутија које Миша може да понесе.

Пример 1

Улаз
```
3
5 2 6
3 7 8
10 10 4
```
Излаз
```
2
```
Објашњење

Можемо да ротирамо прву кутију и да је ставимо у другу. Затим их понесемо заједно. Није могуће понети све 3 кутије.

Пример 2

Улаз
```
5
7 1 2
4 4 3
10 10 3
7 7 6
3 8 10
```
Излаз
```
2
```

Пример 3

Улаз
```
10
1 4 3
1 4 4
2 1 2
3 5 4
2 1 2
5 4 2
2 1 3
4 2 4
3 3 2
2 2 4
```
Излаз
```
7
```
Објашњење

Кутије стављамо следећим редом, $3, 5, 7, 1, 2, 8, 4$. Неке од њих морамо ротирати како би стале у следећу кутију. Приметити да кутије 3 и 5 се сматрају различитим, иако имају исте димензије.

Ограничења

- $1 \leq n \leq 2000$
- $1 \leq w_i, d_i, h_i \leq 10^9$, за $1 \leq i \leq n$

Тест примери су подељени у 6 дисјунктних група:

- У тестовима вредним 10 поена: важи $d_i = 1$ и $h_i = 1$ за $1 \leq i \leq n$
- У тестовима вредним 10 поена: важи $n = 2$
- У тестовима вредним 20 поена: важи $n \leq 10$
- У тестовима вредним 25 поена: важи $h_i = 1$ за $1 \leq i \leq n$
- У тестовима вредним 15 поена: важи да су све димензије различите.
- У тестовима вредним 20 поена: нема додатних ограничења.

Решење

Задатак решавамо динамичким прогрмаирањем.

Задатак: Скочко број

Аутор: Филип Марић

Ђаци се играју погађања непознатог троцифреног броја. Када је Жика рекао свој први број, Ђока му је рекао да је погодио тачно једну цифру непознатог броја, али она није на правом месту. Након тога је Жика рекао свој други број. Напиши програм који проверава да ли тај други број може бити непознати број.

Опис улаза

Са стандардног улаза се уносе два троцифрена броја (између 100 и 999), сваки из посебног реда. Цифре сваког броја су међусобно различите.

Опис излаза

На стандардни излаз исписати `da`, ако други број може бити непоснати број тј. `ne` ако не може.

Пример 1

Улаз
```
123
356
```
Излаз
```
da
```

Пример 2

Улаз
```
123
426
```
Излаз
```
ne
```

Пример 3

Улаз
```
123
215
```
Излаз
```
ne
```

Пример 4

Улаз
```
123
312
```
Излаз
```
ne
```

Пример 5

Улаз
```
123
321
```
Излаз
```
ne
```

Пример 6

Улаз
```
789
192
```
Излаз
```
da
```

Решење

Учитавамо оба броја и одређујемо им цифре. Након тога бројимо колико цифара првог броја је једнако цифрама које нису на истом месту у другом броју. Проверавамо и да ли им се на неком месту цифра поклапа. Други број би могао бити решење ако и само ако је први број једнак 1, а не постоји ни једно место на којем им се цифра поклапа.

```
broj1 = int(input())
broj2 = int(input())

c0 = broj1 % 10
c1 = (broj1 // 10) % 10
c2 = (broj1 // 100) % 10

d0 = broj2 % 10
d1 = (broj2 // 10) % 10
d2 = (broj2 // 100) % 10

pogodjenoVanMesta = 0
if c0 == d1 or c0 == d2:
    pogodjenoVanMesta += 1
if c1 == d0 or c1 == d2:
    pogodjenoVanMesta += 1
if c2 == d0 or c2 == d1:
    pogodjenoVanMesta += 1
```

```
pogodjenoNaMestu = c0 == d0 or c1 == d1 or c2 == d2

if not pogodjenoNaMestu and pogodjenoVanMesta == 1:
    print("da")
else:
    print("ne")
```

Задатак: Робот

Аутор: Михајло Марковић

Робот лута по бесконачном дводимензионалном координатном систему у потрази за пуњачем. Он може у једној секунди из тачке (x, y) у којој се тренутно налази доћи у неку од тачака $(x - 1, y)$, $(x+1, y)$, $(x, y-1)$ или $(x, y+1)$. Другим речима, робот је испрограмиран тако да се може кретати само у правцима који су паралелни осама координатног система и може се зауставити и мењати правац кретања само када се налази у тачкама са целобројним координатама.

На свом путу, робот не сме упасти у језеро које је облика правоугаоника, чије су странице паралелне осама координатног система и чије су координате доњег левог темена (x_1, y_1), а горњег десног темена (x_2, y_2). Сматрати да и тачке које припадају страницама правоугаоника припадају језеру.

Роботу ће се ускоро испразнити батерија. Помозите му тако што ћете наћи за колико најмање секунди он може из тачке (x_p, y_p) у којој се тренутно налази стићи до тачке (x_k, y_k) на којој се налази пуњач, а да на том путу не упадне у језеро.

Опис улаза

У првом реду стандардног улаза налазе се два цела броја x_1 и y_1, координате доњег левог темена правоугаоника који представља језеро.

У другом реду стандардног улаза налазе се два цела броја x_2 и y_2, координате горњег десног темена правоугаоника који представља језеро. $(1 \leq x_1 < x_2 \leq 10^9, 1 \leq y_1 < y_2 \leq 10^9)$

У трећем реду стандардног улаза налазе се два цела броја x_p и y_p, координате почетне тачке на којој се робот налази.

У четвртом реду стандардног улаза налазе се два цела броја x_k и y_k, координате тачке на којој се налази пуњач. $(1 \leq x_p, y_p, x_k, y_k \leq 10^9)$

Гарантује се да тачке (x_p, y_p) и (x_k, y_k) не припадају датом правоугаонику.

Опис излаза

На стандардни излаз исписати један цео број који представља најмањи број секунди потребних роботу да стигне до пуњача, а да испуни све описане услове.

Пример 1

Улаз	*Излаз*	*Објашњење*

`10 10`
`30 20`
`5 15`
`25 25`
	`30`	Робот може за 10 секунди стићи од тачке $(5, 15)$ до тачке $(5, 25)$ крећући се на горе, а затим за 20 секунди може стићи од тачке $(5, 25)$ до тачке $(25, 25)$ крећући се на десно.

Пример 2

Улаз	*Излаз*	*Објашњење*

`10 10`
`30 20`
`25 25`
`20 5`
	`37`	Робот може за 6 секунди стићи од тачке $(25, 25)$ до тачке $(31, 25)$ крећући се на десно, затим за 20 секунди може стићи од тачке $(31, 25)$ до тачке $(31, 5)$ крећући се на доле, а затим за 11 секунди може стићи од тачке $(31, 5)$ до тачке $(20, 5)$ крећући се на лево.

Решење

Поделимо координатни систем на девет квадраната користећи праве добијене продуживањем страница датог правоугаоника.

Када једна тачка (почетна или крајња) припада квадранту изнад датог правоугаоника, а друга квадранту испод датог правоугаоника, најкраћи пут добијамо као мањи од следећа два: обилазак правоугаоника с леве стране (полазећи од тачке (x_p, y_p) долазимо прво до тачке $(x_1 - 1, y_p)$, затим до тачке $(x_1 - 1, y_k)$, па преко ње до тачке (x_k, y_k)) за $x_p - x_1 + 1 + |y_k - y_p| + x_k - x_1 + 1$ секунди или обилазак правоугаоника с десне стране (полазећи од тачке (x_p, y_p) долазимо прво до тачке $(x_2 + 1, y_p)$, затим до тачке $(x_2 + 1, y_k)$, па преко ње до тачке (x_k, y_k)) за $x_2 + 1 - x_p + |y_k - y_p| + x_2 + 1 - x_k$ секунди.

Слично се решава случај када једна тачка припада квадранту лево од датог правоугаоника, а друга квадранту десно од датог правоугаоника. Најкраћи пут добијамо као мањи од следећа два: обилазак правоугаоника с доње стране (полазећи од тачке (x_p, y_p) долазимо прво до тачке $(x_p, y_1 - 1)$, затим до тачке $(x_k, y_1 - 1)$, па преко ње до тачке (x_k, y_k)) за $y_p - y_1 + 1 + |x_k - x_p| + y_k - y_1 + 1$ секунди или обилазак правоугаоника с горње стране (полазећи од тачке (x_p, y_p) долазимо прво до тачке $(x_p, y_2 + 1)$, затим до тачке $(x_k, y_2 + 1)$, па преко ње до тачке (x_k, y_k)) за $y_2 + 1 - y_p + |x_k - x_p| + y_2 + 1 - y_k$ секунди.

У свим осталим случајевима решење је Менхетн растојање које се рачуна по формули $|x_p - x_k| + |y_p - y_k|$.

```python
def ucitaj():
    a, b = input().split()
    a = int(a)
    b = int(b)
    return a, b

def uPravougaoniku(x1,y1,x2,y2,x,y):
    return x1 <= x <= x2 and y1 <= y <= y2

x1,y1=ucitaj()
x2,y2=ucitaj()
xp,yp=ucitaj()
xk,yk=ucitaj()
N=1000000000

if ((uPravougaoniku(x1,y2+1,x2,N,xp,yp) and uPravougaoniku(x1,1,x2,y1-1,xk,yk)) or
    (uPravougaoniku(x1,y2+1,x2,N,xk,yk) and uPravougaoniku(x1,1,x2,y1-1,xp,yp))):
    print(abs(yp-yk)+min(xp+xk-2*x1,2*x2-xp-xk)+2)
elif ((uPravougaoniku(x2+1,y1,N,y2,xp,yp) and uPravougaoniku(1,y1,x1-1,y2,xk,yk)) or
      (uPravougaoniku(x2+1,y1,N,y2,xk,yk) and uPravougaoniku(1,y1,x1-1,y2,xp,yp))):
```

```
    print(abs(xp-xk)+min(yp+yk-2*y1,2*y2-yp-yk)+2)
else:
    print(abs(xp-xk)+abs(yp-yk))
```

Задатак: Авионске карте

Аутор: Душан Попадић

Када је програмер Ђура отишао у пензију одлучио је да мало путује светом. Кренуо је из родног Краљева и авионом путовао по разним градовима света, али је у сваком граду који је посетио био само једном. Од тада је прошло преко 10 година и Ђура препричава своја путовања унуцима, међутим не може да се тачно сети којим редом је обилазио градове, али је схватио да може да реши тај проблем јер је сачувао авионске карте са свог путовања. На свакој карти су написана два града: полазни и крајњи град сваког лета. Помозите Ђури да на основу авионских карата реконструише свој пут.

Опис улаза

У првом реду стандардног улаза се налази број карата n ($1 \leq n \leq 10^5$). У наредних n редова се налазе имена два града раздвојена размаком која представљају полазни и крајњи град на карти. Име сваког града је једна реч састављена од слова енглеске абецеде, краћа од 40 слова.

Додатна ограничења

У тест примерима вредним 50 поена додатно важи $n \leq 1000$.

Опис излаза

На стандардни излаз исписати редом градове које је Ђура обилазио, сваки у посебном реду.

Пример 1

Улаз
```
5
Moskva Atina
Beograd London
Kopenhagen Moskva
Kraljevo Beograd
London Kopenhagen
```

Излаз
```
Kraljevo
Beograd
London
Kopenhagen
Moskva
Atina
```

Објашњење

Ђура креће из Краљева и на основу карте "Краљево -> Београд" закључујемо да је после Краљева био у Београду. Након тога на основу карте "Београд -> Лондон" закључујемо да је наредни град Лондон итд.

Пример 2

Улаз
```
7
Sana AdisAbeba
Kinsasa Vindhuk
Kraljevo Istanbul
AdisAbeba Kinsasa
Istanbul Islamabad
Vindhuk Rabat
Islamabad Sana
```

Излаз
```
Kraljevo
Istanbul
Islamabad
Sana
AdisAbeba
Kinsasa
Vindhuk
Rabat
```

Решење

Наивно решење

Најједноставније решење је да учитамо све карте у низ (податке о свакој карти смештамо у једну структуру или класу података) и затим нађемо почетну карту. Након тога за сваку карту погледамо у који град се иде, испишемо га и међу свим картама тражимо ону којој је тај град почетни. Ово радимо док не испишемо све градове.

Анализа сложености

Како за сваку од карата морамо проћи кроз све остале (да бисмо проверили која иде након ње), сложеност решења је $O(n^2)$. Ово решење доноси 50 поена.

```
class Karta:
    def __init__(self, pocetak, kraj):
        self.pocetak = pocetak
        self.kraj = kraj

n = int(input())
karte = []

for _ in range(n):
    pocetak, kraj = input().split()
    karte.append(Karta(pocetak, kraj))

trenutni = "Kraljevo"
print(trenutni)

for _ in range(n):
    for karta in karte:
        if karta.pocetak == trenutni:
            trenutni = karta.kraj
            print(trenutni)
            break
```

Решење коришћењем бинарне претраге

У претходном решењу смо за сваку карту линеарно тражили ону која иде након ње (то значи да смо пролазили кроз све карте). Ово можемо унапредити тако што ћемо уместо линеарне користити бинарну претрагу. У наивном решењу смо наредну карту тражили тако да се њен почетни град поклапа са крајњим градом тренутне карте - дакле, да бисмо могли да применимо бинарну претрагу потребно нам је да су карте поређане (сортиране) абецедно на основу почетног града. Дакле, у овом решењу ћемо прво сортирати низ карата по почетном граду, а затим за сваку карту тражити ону која иде иза ње користећи бинарну претрагу.

Анализа сложености

Како више не користимо линеарну претрагу него бинарну чија је сложеност $O(\log(n))$, сложеност овог решења је $O(n\log(n))$ што покрива и сортирање и претрагу.

```
import bisect

class Karta:
    def __init__(self, pocetak, kraj):
        self.pocetak = pocetak
        self.kraj = kraj

n = int(input())
karte = []

trenutnaKarta = None

for _ in range(n):
    pocetak, kraj = input().split()
    k = Karta(pocetak, kraj)
    karte.append(k)
    if pocetak == "Kraljevo":
        trenutnaKarta = k

# Sortiramo karte po pocetku
karte.sort(key=lambda k: k.pocetak)

print(trenutnaKarta.pocetak)
print(trenutnaKarta.kraj)

for _ in range(n - 1):
    # Binarna pretraga za pronalazenje sledece karte
    sledeciIndeks = bisect.bisect_left(karte, trenutnaKarta.kraj, key=lambda k: k.pocetak)
    trenutnaKarta = karte[sledeciIndeks]
    print(trenutnaKarta.kraj)
```

Решење коришћењем речника

Овај задатак се може решити и другачијим приступом. Оно што нас заправо занима је да реконструишемо Ђурин пут, а то практично значи да нас за сваки град само занима који град долази након њега. Ово врло лако можемо памтити у речнику користећи почетни град као кључ, а крањи град као вредност. Када учитавамо карте кажемо да је крајњи град вредност у речнику за кључ који је почетни град. Када желимо да реконструишемо пут, проверимо која је вредност за кључ "Краљево" и знамо да нам је то други град. Затим тај други користимо као кључ и вредност у речнику нам даје трећи град. Овај поступак настављамо док не испишемо све градове.

Анализа сложености

Приступ речнику можемо сматрати константним ($O(1)$), тако да је сложеност овог решења $O(n)$.

```
n = int(input())
karte = {}
```

```
for _ in range(n):
    pocetak, kraj = input().split()
    karte[pocetak] = kraj

trenutni = "Kraljevo"
print(trenutni)

for _ in range(n):
    trenutni = karte[trenutni]
    print(trenutni)
```

Задатак: Палиндромски упити

Дата је ниска s са n карактера и q упита. Упити су облика "Променити i-ти карактер у c". После сваког упита, исписати `DA`, уколико ниска јесте палиндром и `NE` у супротном.

Опис улаза

У првом реду стандардног улаза, дати су цели позитивни бројеви n ($1 \leq n \leq 50.000$) и q ($1 \leq q \leq 50.000$), који редом представљају дужину ниске и број упита.

У другом реду стандардног улаза, дата је ниска s са n карактера.

У наредних q редова, дата су по два податка, i и c. Први представља индекс карактера који се мења у нисци. Други представља карактер у који се тај индекс мења.

Гарантује се да је $n, q \leq 1.000$ у 50% примера.

Опис излаза

У i-том реду исписати `DA`, уколико је ниска палиндром након i-тог упита и `NE` у супротном.

Пример

Улаз	*Излаз*	*Објашњење*
`3 4`	`DA`	• Након првог упита, ниска је `aba`, што јесте палиндром
`abc`	`DA`	• Након другог упита, ниска је `aca`, што јесте палиндром
`3 a`	`NE`	• Након трећег упита, ниска је `rca`, што није палиндром
`2 c`	`DA`	• Након четвртог упита, ниска је `rcr`, што јесте палиндром
`1 r`		
`3 r`		

Решење

Опис главног решења

У овом блоку се описује главно решење задатка.

```
[sn,sq] = input().split(' ')
q = int(sq)
n = int(sn)
s = list(input())
cnt = 0
for i in range((n+1)//2):
   if s[i] == s[n-1-i]:
```

```
        cnt+=1

while q:
    [si,c] = input().split(' ')
    i = int(si) - 1
    if s[i] == s[n-1-i]:
        cnt -= 1
    s[i] = list(c.split())[0]
    if s[i] == s[n-1-i]:
        cnt += 1
    if cnt == (n+1)//2:
        print("DA")
    else:
        print("NE")
    q -= 1
```