

Matura 2

Zemsta CKE

Do boju!

Zadanie 2.1. (0–3)

Zapisz w pseudokodzie lub w wybranym języku programowania algorytm, który dla danej dodatniej całkowitej liczby n obliczy liczbę bloków w jej zapisie binarnym.

Przykład:

Dla liczby **67** wynikiem jest 3, ponieważ 67 w zapisie binarnym to 1000011 (dwa bloki jedynek i jeden blok zer).

Dla liczby **245** wynikiem jest 5, ponieważ 245 w zapisie binarnym to 11110101 (trzy bloki jedynek i dwa bloki zer).

Uwaga: W zapisie algorytmu możesz korzystać tylko z instrukcji sterujących, operatorów arytmetycznych: dodawania, odejmowania, mnożenia, dzielenia, dzielenia całkowitego i reszty z dzielenia; operatorów logicznych, porównań, instrukcji przypisania lub samodzielnie napisanych funkcji i procedur wykorzystujących powyższe operacje. **Zabronione** jest używanie funkcji wbudowanych oraz operatorów innych niż wymienione, dostępnych w językach programowania, w tym zwłaszcza funkcji zamiany między systemami pozycyjnymi i konwersji między typami danych.

Specyfikacja:

Dane:

n – dodatnia liczba całkowita

Wynik:

b – liczba bloków w zapisie binarnym liczby n



Wszystkie zadania na tej prezentacji pochodzą z matury z Maja 2023

O co chodzi w poleceniu? - Pseudokod

Zapisz w pseudokodzie lub w wybranym języku programowania

- Pseudokod z założenia jest prostszy i bardziej zwięzły od większości prawdziwych języków
- Kartka papieru nie sprawdzi błędów
- My jednak posługujemy się Pythonem, którego składnia jest na tyle łatwa, że pseudokod nie jest konieczny
- Ponadto nie narażamy się na złą interpretację
- Należy bardzo uważać na wcięcia – komputer tego za nas nie dopilnuje (warto używać kratek)
- Warto zaznaczyć, że piszecie w Pythonie

O co chodzi w poleceniu? - Ograniczenia

Uwaga: W zapisie algorytmu możesz korzystać tylko z instrukcji sterujących, operatorów arytmetycznych: dodawania, odejmowania, mnożenia, dzielenia, dzielenia całkowitego i reszty z dzielenia; operatorów logicznych, porównań, instrukcji przypisania lub samodzielnie napisanych funkcji i procedur wykorzystujących powyższe operacje. **Zabronione** jest używanie funkcji wbudowanych oraz operatorów innych niż wymienione, dostępnych w językach programowania, w tym zwłaszcza funkcji zamiany między systemami pozycyjnymi i konwersji między typami danych.

- To zadanie ma na celu sprawdzić zdolność tworzenia algorytmu
- Korzystamy zatem z podstawowych operacji:
 - Przypisanie: `x = 123`
 - Operatory arytmetyczne: `+` `-` `*` `/` `//` `%`
 - Operatory logiczne: `and` `or` `not`
 - Porównania: `>` `<` `>=` `<=` `==` `!=`
 - Instrukcje sterujące: `for` `while` `if` `else` `elif`
- Nie wolno używać niczego więcej
- Lepiej też nie używać ‘pythonicznych’ sztuczek (jak pętla wewnątrz tablicy)

System binarny - przypomnienie

- System dziesiętny i binarny są do siebie zaskakująco podobne
- Oba są systemami pozycyjnymi – to znaczy, że każdej **pozycji** w liczbie **odpowiada** pewna **wartość** (wartości to kolejne potęgi podstawy systemu)
- **Cyfra** na tej pozycji mówi **ile razy** dodajemy do liczby daną wartość
- Przykład:

Wartości:

$$\begin{array}{cccc} 1000 & 100 & 10 & 1 \\ 10^3 & 10^2 & 10^1 & 10^0 \end{array}$$

2137

$$\begin{array}{r} 2000 + 100 + 30 + 7 \\ 1000 \times 2 \quad 100 \times 1 \quad 10 \times 3 \quad 1 \times 7 \end{array}$$

System binarny - przypomnienie

- Analogicznie działa to w systemie binarnym, z tym, że:
 - Są tylko dwie cyfry: 0 i 1
 - Kolejne pozycje odpowiadają potęgom dwójką ($1, 2, 3, 8, 16 \dots$) zamiast potęgom dziesiątki ($1, 10, 100 \dots$)
- Przykład:

$$\begin{array}{r} & 8 & 4 & 2 & 1 \\ & 2^3 & 2^2 & 2^1 & 2^0 \\ 1101 & & & & \\ 8 \times 1 & + & 4 \times 1 & + & 0 \times 0 & + & 1 \times 1 \end{array}$$

... a zatem po
przetłumaczeniu na
dziesiętny, $1101 = 13$

Algorytm zamiany DEC-BIN

- Aby zamienić liczbę dziesiętną na binarną, musimy wiedzieć co wpisać na każdej pozycji
- Weźmy za przykład liczbę 25
- Będziemy dzielić ją przez dwa, aż nie zredukujemy jej do 0
- Reszty zapisane po kolej ułożą się w liczbę binarną
- Zwróćcie uwagę, że podobnie wygląda rozbijanie na cyfry liczby w systemie dziesiętnym (z tą różnicą, że dzielimy przez potęgi dziesiątki, nie dwójkę)

Algorytm zamiany DEC-BIN

$25 \% 2 = 1$	#cyfra nr 1:1	1
$25 // 2 = 12$	# $25//2$	
$12 \% 2 = 0$	#cyfra nr 2:0	01
$12 // 2 = 6$	# $25//4$	
$6 \% 2 = 0$	#cyfra nr 3:0	001
$6 // 2 = 3$	# $25//8$	
$3 \% 2 = 1$	#cyfra nr 4:1	1001
$3 // 2 = 1$	# $25//16$	
$1 \% 2 = 1$	#cyfra nr 5:1	11001
$1 // 2 = 0$	# $25 // 32$	koniec

A jak to się ma do zadania?

- Powyższy algorytm ma zastosowanie w tym i w wielu innych zadaniach maturalnych
- Musimy go sprytnie wykorzystać, aby policzyć „bloki”
- Pomyśły?
- Podpowiedź: tworzenie reprezentacji liczby binarnej w pamięci będzie dość trudne

Zapisz w pseudokodzie lub w wybranym języku programowania algorytm, który dla danej dodatniej całkowitej liczby n obliczy liczbę *bloków* w jej zapisie binarnym.

Przykład:

Dla liczby **67** wynikiem jest 3, ponieważ 67 w zapisie binarnym to 1000011 (dwa *bloki* jedynek i jeden *blok* zer).

Dla liczby **245** wynikiem jest 5, ponieważ 245 w zapisie binarnym to 11110101 (trzy *bloki* jedynek i dwa *bloki* zer).

Rozwiązanie

- Nie musimy liczyć bloków jako takich
- Wystarczy, że policzymy ile razy nowy blok się rozpoczął
- W jaki sposób można rozpoznać rozpoczęcie bloku?

...

Rozwiązanie

- Nie musimy liczyć bloków jako takich
- Wystarczy, że policzymy ile razy nowy blok się rozpoczął
- W jaki sposób można rozpoznać rozpoczęcie bloku?
- **Każdy blok (oprócz pierwszego) zaczyna się tam, gdzie sąsiadują ze sobą dwie różne cyfry**
- Jak można wykryć taką sytuację, skoro umiemy wyznaczyć po kolej wszystkie cyfry liczby binarnej?

...

Rozwiązanie

- Nie musimy liczyć bloków jako takich
- Wystarczy, że policzymy ile razy nowy blok się rozpoczął
- W jaki sposób można rozpoznać rozpoczęcie bloku?
- **Każdy blok (oprócz pierwszego) zaczyna się tam, gdzie sąsiadują ze sobą dwie różne cyfry**
- Jak można wykryć taką sytuację, skoro umiemy wyznaczyć po kolei wszystkie cyfry liczby binarnej?
- **Przy wyznaczaniu nowej, musimy pamiętać poprzednią cyfrę. Wtedy możemy je porównać. Jeśli są różne, zaczął się blok**

Rozwiążanie

Loading . . .

Rozwiązanie

```
blocks = 0
last = -1 #kazda cyfra bedzie rozna od -1
while number > 0:
    digit = number % 2
    if(last != digit):
        blocks += 1
    last = digit
    number //= 2
```

Jedziemy dalej

- Kolejne zadanie jest banalne, jeśli rozwiązaliśmy poprzednie – pomóżmy je:

Zadanie 2.2. (0–2)

Podaj, ile liczb w pliku bin.txt składa się z **co najwyżej dwóch bloków** (zgodnie z definicją *bloku* podaną wcześniej).

Dla danych z pliku bin_przyklad.txt poprawna odpowiedź to 3.

- To jest dużo ciekawsze:

Zadanie 2.3. (0–2)

Wypisz największą z liczb zapisanych w pliku bin.txt.

Dla danych z pliku bin_przyklad.txt poprawna odpowiedź to 1000111110111100000.

Arkusze i pliki znajdziecie tutaj:

<https://arkusze.pl/matura-stara-informatyka-2023-maj-poziom-rozszerzony/>



Porównywanie liczb binarnych

- Treść pliku zostanie odczytana jako zbiór liczb dziesiętnych złożonych z zer i jedynek
- Wartości będą zatem niepoprawne
- Moglibyśmy tłumaczyć liczby binarne na dziesiętne – jest to pierwsze przychodzące na myśl rozwiązańe
- Zależy nam jednak na czasie
- Czy da się to zrobić szybciej, niż implementując algorytm konwersji dec-bin?

Porównywanie liczb binarnych

- Jeśli zamierzamy tylko porównywać liczby, to nie musimy ich wcale konwertować
- Porównywanie liczb zawsze przebiega identycznie, niezależnie od systemu liczbowego
- Przykładowo 111 będzie zawsze większe od 110



Rozwiążanie

Loading . . .

Rozwiązanie

```
file = open("informatyka-2023-maj-matura-rozszerzona-zalaczniki/Dane_2305/bin_przyklad.txt")
numbers = file.readlines()
largest = -1
for x in numbers:
    number = int(x) #inaczej będziemy sortować alfabetycznie
    if number > largest:
        largest = number
print(largest)
```

Kolejne zadanie

Zadanie 2.5. (0–3)

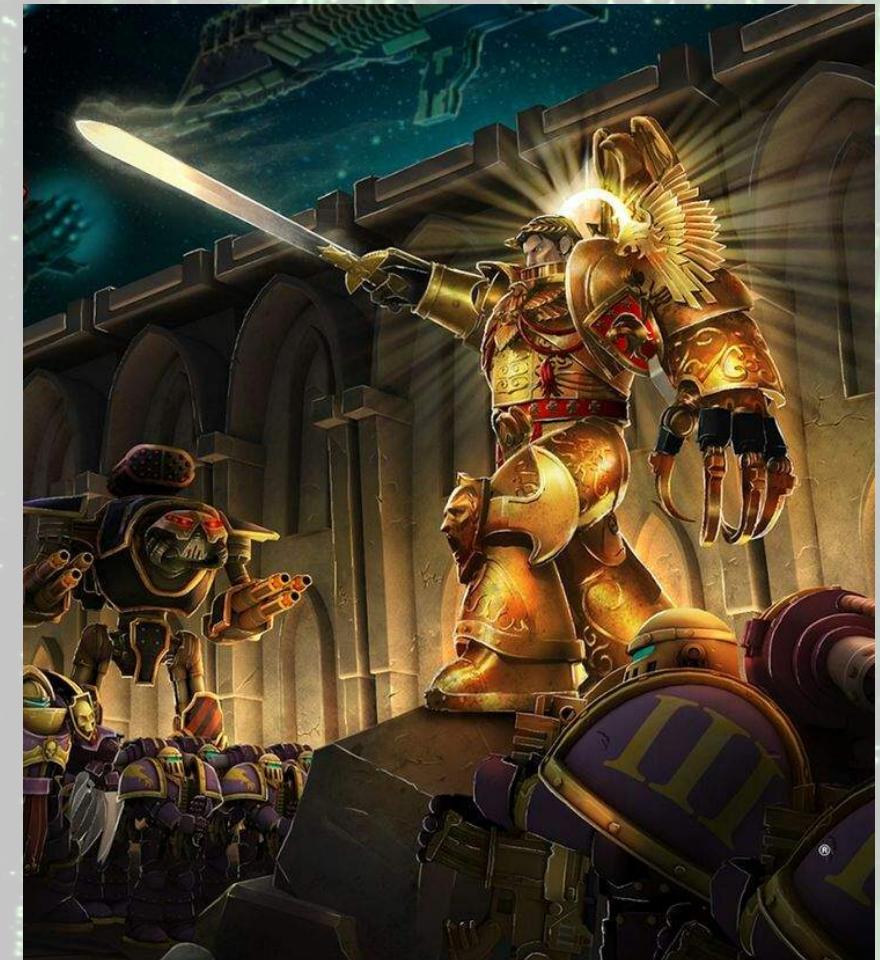
Napisz program, który dla każdej binarnej liczby p zapisanej w pliku bin.txt obliczy wynik działania

$$p \text{ XOR } (p \text{ div } 2)$$

gdzie XOR to operacja bitowa opisana wcześniej, a $p \text{ div } 2$ oznacza połowę liczby p , zaokrągloną w dół do liczby całkowitej.

Otrzymane wyniki podaj w systemie binarnym. Zapisz je do pliku wyniki2_5.txt w kolejności występowania liczb w pliku bin.txt, każdy wynik w oddzielnym wierszu.

Odpowiedź dla danych z pliku bin_przyklad.txt znajduje się w pliku odp_bin_przyklad.txt.



XOR – Exclusive OR

- XOR to operacja przeprowadzana na 2^* bitach
- Operacja zwraca 1, kiedy bity są różne i 0, gdy są takie same
- Jeśli poddajemy tej operacji dwie liczby, to porównujemy parami bity na tych samych pozycjach
- Jeśli jedna z liczb jest mniejsza, to na jej początku stawiamy odpowiednio wiele zer (nie wpływa to na jej wartość)

p	q	$p \text{ XOR } q$
1	1	0
1	0	1
0	1	1
0	0	0

*Zdarzają się przypadki wykonywania XOR na więcej niż dwóch bitach - rabini są jednak niezgodni co do oczekiwanej wyniku takiej operacji

XOR – Exclusive OR

- Operacja XOR jest dość powszechna w programowaniu
- Wiele języków programowania udostępnia operator xor
- W pythonie robi to operator \wedge (który jednak jest operatorem logicznym i zwraca True/False zamiast 0/1)
- XOR można także zrealizować jako złożenie operatorów dodawania i modulo:

$$A \text{ xor } B = (A + B) \% 2$$

- Oczywiście zakładamy tu, że A i B to wartości 0 lub 1

Dzielenie

- Jeśli w treści zadania pojawia się takie sformułowanie:
zaokrągloną w dół do liczby całkowitej.

To zawsze chodzi o wynik operacji dzielenia całkowitego (w Pythonie operator `//`)

- Większość języków programowania domyślnie stosuje takie dzielenie podczas pracy na liczbach całkowitych
- Python jednak stara się nam ułatwić życie i przy dzieleniu automatycznie zamienia liczby całkowite na zmienoprzecinkowe – należy na to uważać

Dzielenie

- Jak wiecie, aby wykonać dzielenie całkowite przez 10 na liczbie dziesiętnej, wystarczy „przesunąć przecinek” o jedno miejsce w lewo i zignorować wszystko, co za nim

$$12345 \text{ // } 10 = 1234$$

- Podobnie wygląda sytuacja z dzieleniem liczby binarnej przez 2
- W jaki sposób zrealizować to w programie?

Dzielenie

- Bardzo łatwo – znów skorzystamy z faktu, że Python domyślnie stosuje system dziesiętny
- Aby wykonać dzielenie $//2$, możemy na liczbach wczytanych z pliku wykonać operację $//10$

$$11010 \text{ (26)} // 10 = 1101 \text{ (13)}$$

- W wielu językach programowania można się posłużyć operacją przesunięcia bitowego

Dzielenie liczby na cyfry

- W omawianym zadaniu musimy przeprowadzić operacje na kolejnych cyfrach liczby
- Aby tego dokonać, musimy ją sobie podzielić
- W przypadku liczby dziesiętnej możemy wydobyć ostatnią cyfrę liczby operacją $\% 10$
- Następnie operacją $// 10$ sprawiamy, że na miejsce ostatniej wchodzi przedostatnia
- Możemy powtarzać powyższe kroki, aż nie odczytamy wszystkich cyfr
- W pozostałych systemach liczbowych działa to identycznie – wystarczy zamienić 10 na podstawę danego systemu (np. 2 w binarnym)

Rozwiążanie

Loading . . .

Rozwiązanie

```
1 usage  ▲ MikoStoro
def div2(num):
    return num//10

1 usage  ▲ MikoStoro
def xor(b1,b2):
    return (b1+b2)%2

file = open("informatyka-2023-maj-matura-rozszerzona-zalaczniki/Dane_2305/bin_przyklad.txt")
numbers = file.readlines()

for number in numbers:
    number = int(number)
    divided = div2(number)
    result = ""
    while(divided > 0 or number > 0):
        b1 = divided % 10
        b2 = number % 10
        number //= 10
        divided //= 10
        bit = xor(b1,b2)
        result = str(bit) + result
    print(result)
```

Jeszcze jedno!

Zadanie 3.1. (0-2)

Fragmentem 2-cyfrowym nazywamy dwie następujące po sobie cyfry w pliku `pi.txt`. Wszystkich fragmentów 2-cyfrowych zapisanych w tym pliku jest 9 999. Ostatni rozpoczyna się w wierszu nr 9 999.

Przykładowe fragmenty 2-cyfrowe podano w poniższej tabeli.

i	Fragment 2-cyfrowy złożony z cyfr na pozycjach $i, i+1$
1	14
2	41
3	15
9	35

Znajdź liczbę wszystkich fragmentów 2-cyfrowych, które są zapisami dziesiętnymi liczb o wartościach **większych** od 90.



Wzorce

- Jeśli pamiętacie jedno z poprzednich zadań, to powinniście zauważyc coś znajomego
- Znajdowanie „fragmentów dwucyfrowych” przypomina nieco szukanie granicy między blokami z zadania 2
- Można zatem zastosować użytą tam metodę, jedynie nieznacznie ją zmieniając
- Takich powtarzających się motywów jest w zadaniach maturalnych wiele – dlatego warto przerobić ich jak najwięcej

Cięcie list (slicing)

- W tym zadaniu nie będą bardzo istotne, ale ułatwiają nam życie i w praktyce przydają się bardzo często
- Indeksując listę, możemy użyć znaku : aby zastosować tzw. slicing
- Dzięki niemu możemy pracować na podzbiorze elementów listy bez konieczności modyfikowania jej
 - `lista[:5]` – wszystkie elementy aż do 4 (0,1,2,3,4)
 - `lista[5:]` - wszystkie elementy zaczynając od 5 (5,6,7...)
 - `lista[3:7]` – elementy od 3 do 6 (3,4,5,6)

Rozwiążanie

Loading . . .

Rozwiązanie

```
numbers = file.readlines() #kazda linia zawiera jedną cyfrę
last = int(numbers[0]) #upewnijmy się, że python odczytał liczbę, nie napis
result = 0
for n in numbers[1:]:
    current = int(n)
    if last*10 + current > 90:
        result +=1
    last = current

print(result)
```

To było za proste

Zadanie 3.2. (0-3)

Wszystkich możliwych różnych fragmentów **2-cyfrowych** jest dokładnie 100. Są nimi fragmenty 00, 01, 02, ..., 99. Można sprawdzić, że np. **2-cyfrowy** fragment równy 27 występuje w pliku pi.txt dokładnie 101 razy.

Znajdź fragmenty **2-cyfrowe**, których liczba wystąpień w pliku pi.txt jest najmniejsza, oraz fragmenty **2-cyfrowe**, których liczba wystąpień w pliku pi.txt jest największa.

W wyniku podaj znalezione fragmenty **2-cyfrowe** oraz liczby ich wystąpień.

W przypadku, gdy więcej niż jeden fragment występuje tyle samo razy, wypisz ten o mniejszej wartości liczbowej.

Dla danych w pliku pi_przyklad.txt poprawna odpowiedź to

00 0

62 4

(minimalna liczba wystąpień: fragment 00, liczba wystąpień 0; maksymalna liczba wystąpień: fragment 62, liczba wystąpień 4)



[HTTP://KAMYUDIGITALARTWORKS.COM](http://KAMYUDIGITALARTWORKS.COM)

Podpowiedzi

- W tym zadaniu można kreatywnie wykorzystać mechanizm indeksowania tablic – indeks to przecież nic innego jak liczba, którą można obliczyć, zamiast podawać jawnie
- Należy użyć mechanizmu wyszukiwania fragmentów dwucyfrowych z poprzedniego zadania – już go napisaliśmy, więc zaoszczędzimy trochę czasu

Rozwiążanie

Loading . . .

Rozwiązanie

```
numbers = file.readlines() #kazda linia zawiera jedną cyfrę
last = int(numbers[0]) #upewnijmy sie, że python odczytał liczbę, nie napis
for n in numbers[1:]:
    current = int(n)
    fragment = last*10 + current
    results[fragment] += 1
    last = current

#znajdzmy skrajne elementy
smallest_index = 0
smallest = results[0]
largest_index = 0
largest = results[0]

for i in range(100):
    #stosując <, a nie <= upewnijmy się, że tylko uwzględnimy tylko pierwszy wynik
    if results[i] < smallest:
        smallest = results[i]
        smallest_index = i
    if results[i] > largest:
        largest = results[i]
        largest_index = i
```

Matura nie ma szans

Informacja do zadań 3.3. i 3.4.

Skończony co najmniej 4-elementowy ciąg liczb (a_1, a_2, \dots, a_n) jest *rosnąco-malejący*, jeśli można podzielić go na dwa ciągi, z których pierwszy jest rosnący, a drugi – malejący, tzn. jeśli istnieje takie $k \in \{2, 3, \dots, n-2\}$, że $a_1 < a_2 < \dots < a_k$ oraz $a_{k+1} > a_{k+2} > \dots > a_n$.

Przykład:

Ciąg $(2, 5, 7, 9, 8, 3, 1)$ jest *rosnąco-malejący*, bo można go podzielić na dwa ciągi: rosnący $(2, 5, 7)$ i malejący $(9, 8, 3, 1)$ lub – odpowiednio – $(2, 5, 7, 9)$ i $(8, 3, 1)$. Ciąg $(5, 9, 9, 4, 1)$ także jest *rosnąco-malejący*.

Przykłady ciągów, które nie są *rosnąco-malejące*, to: $(2, 5, 8, 4, 3, 4, 5)$, $(1, 2, 3, 4)$, $(5, 5, 3, 2, 1)$.

Zadanie 3.3. (0–3)

Podaj, ile jest wszystkich *rosnąco-malejących* ciągów złożonych z dokładnie sześciu kolejnych cyfr zapisanych w pliku pi.txt.

Dla pliku pi_przyklad.txt poprawna odpowiedź to 3.

(w pliku pi_przyklad.txt są trzy ciągi *rosnąco-malejące* złożone z dokładnie sześciu cyfr: 028841, 089986, 899862)



Wskazówki

- Tutaj przyda się cięcie listy
- Może warto pomóc sobie definiując funkcję która określi czy ciąg jest rosnąco-malejący
- Znów stosujemy mechanizm podobny do wyszukiwania fragmentów 2-cyfrowych
- Zwróćcie uwagę, że istnieje dokładnie jeden punkt, w którym ciąg rosnący przechodzi w malejący
- Punkt ten jednak może znajdować się na różnych pozycjach (z wyjątkiem pierwszej i ostatniej)
- Metoda `file.readlines()` czyta treść razem ze znakami końca linii!

Rozwiążanie

Loading . . .

Rozwiązanie

```
numbers = file.readlines()
numbers = [int(x) for x in numbers]

num_of_series = 0
for i in range(len(numbers)-6):
    slc = numbers[i:i + 6]
    res = czy_ros_mal(slc)
    if res:
        num_of_series += 1
        print(slc)
        print(res)

print(num_of_series)
```

```
def czy_ros_mal(series):
    mal = False
    if not(series[0] < series[1]):
        return False

    for i in range(1, 5):
        if mal:
            if not(series[i] > series[i+1]):
                return False
        else:
            if not(series[i] < series[i+1]):
                mal = True
    return True
```

Ostatnie zadanie

Zadanie 3.4. (0–2)

Znajdź najdłuższy ciąg kolejnych cyfr z pliku `pi.txt`, który jest *rosnąco-malejący*, oraz pozycję, na której on się rozpoczyna. W pliku `pi.txt` jest tylko jeden taki ciąg o największej długości.

Wynik zapisz w dwóch wierszach: w pierwszym wierszu zapisz pozycję, od której zaczyna się znaleziony ciąg, a w drugim wypisz znaleziony ciąg. Cyfry ciągu zapisz jedną po drugiej, bez znaku odstępu.

Dla danych w pliku `pi_przyklad.txt` poprawna odpowiedź to

77

0899862

(najdłuższy ciąg *rosnąco-malejący* w pliku `pi_przyklad.txt` to ciąg 0899862 o długości 7 rozpoczynający się w 77 wierszu pliku).



Wskazówki

- Zadanie mamy praktycznie rozwiązane
- Problem: ciągi mogą być dowolnej długości
- Co charakteryzuje ciąg?

Wskazówki

- Zadanie mamy praktycznie rozwiązane
- Problem: ciągi mogą być dowolnej długości
- Co charakteryzuje ciąg?
 - Miejsce rozpoczęcia
 - Długość

Wskazówki

- Zadanie mamy praktycznie rozwiązane
- Problem: ciągi mogą być dowolnej długości
- Co charakteryzuje ciąg?
 - Miejsce rozpoczęcia
 - Długość

Rozwiążanie

Loading . . .

Rozwiązanie

```
def czy_ros_mal(series):
    mal = False
    if not (series[0] < series[1]):
        return False

    for i in range(1, len(series) - 1):
        if mal:
            if not (series[i] > series[i + 1]):
                return False
        else:
            if not (series[i] < series[i + 1]):
                mal = True
    return True
```

```
#minimalna dlugosc ciagu to 3
curr_index = 0
curr_len = 3

max_index = -1
max_len = -1

while (curr_index + curr_len < len(numbers)):
    slc = numbers[curr_index:curr_index + curr_len]
    if czy_ros_mal(slc):
        if curr_len > max_len:
            max_index = curr_index
            max_len = curr_len
        curr_len += 1
    else:
        curr_index += 1
        curr_len = 3
print(numbers[max_index:max_index + max_len])
```