

MATURA

Von Neumann Kontratakuje

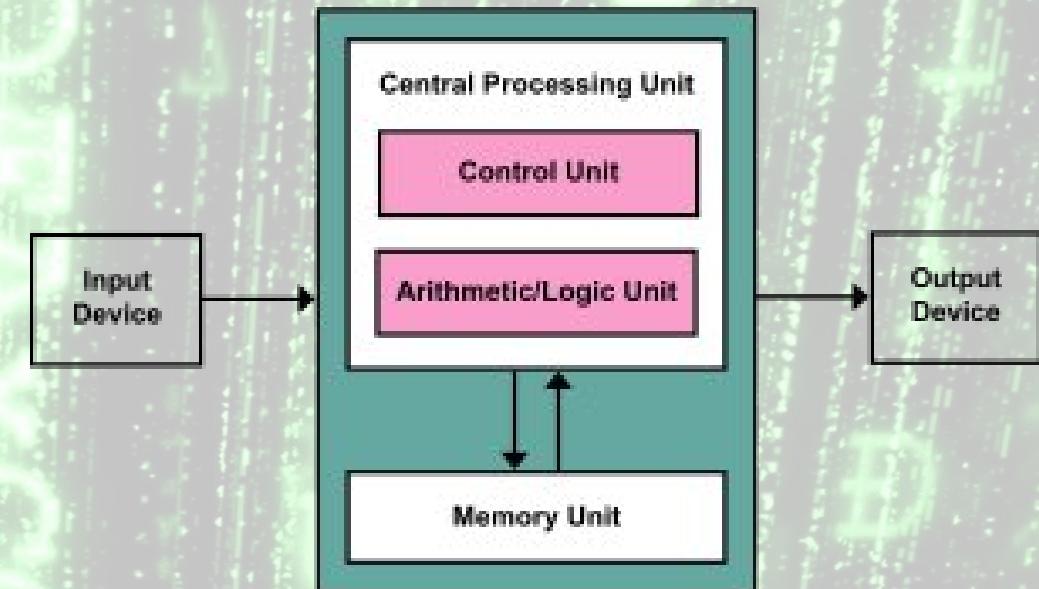
John von Neumann

- Węgiersko-amerykański uczony pochodzenia żydowskiego; matematyk, informatyk, fizyk i inżynier chemik.
- Von Neumann był głównym twórcą teorii gier i teorii automatów komórkowych, stworzył formalizm matematyczny mechaniki kwantowej, uczestniczył w projekcie Manhattan i przyczynił się do rozwoju numerycznych prognoz pogody.



Architektura von Neumanna

- W architekturze tej komputer składa się z czterech głównych komponentów:
- Pamięci komputerowej przechowującej dane programu oraz instrukcje programu; każda komórka pamięci ma unikatowy identyfikator nazywany jej adresem
- Jednostki sterującej odpowiedzialnej za pobieranie danych i instrukcji z pamięci oraz ich sekwencyjne przetwarzanie
- Jednostki arytmetyczno-logicznej odpowiedzialnej za wykonywanie podstawowych operacji arytmetycznych.
- Urządzeń wejścia/wyjścia służących do interakcji z operatorem
- Jednostka sterująca wraz z jednostką arytmetyczno-logiczną tworzą procesor.



Zaczynamy

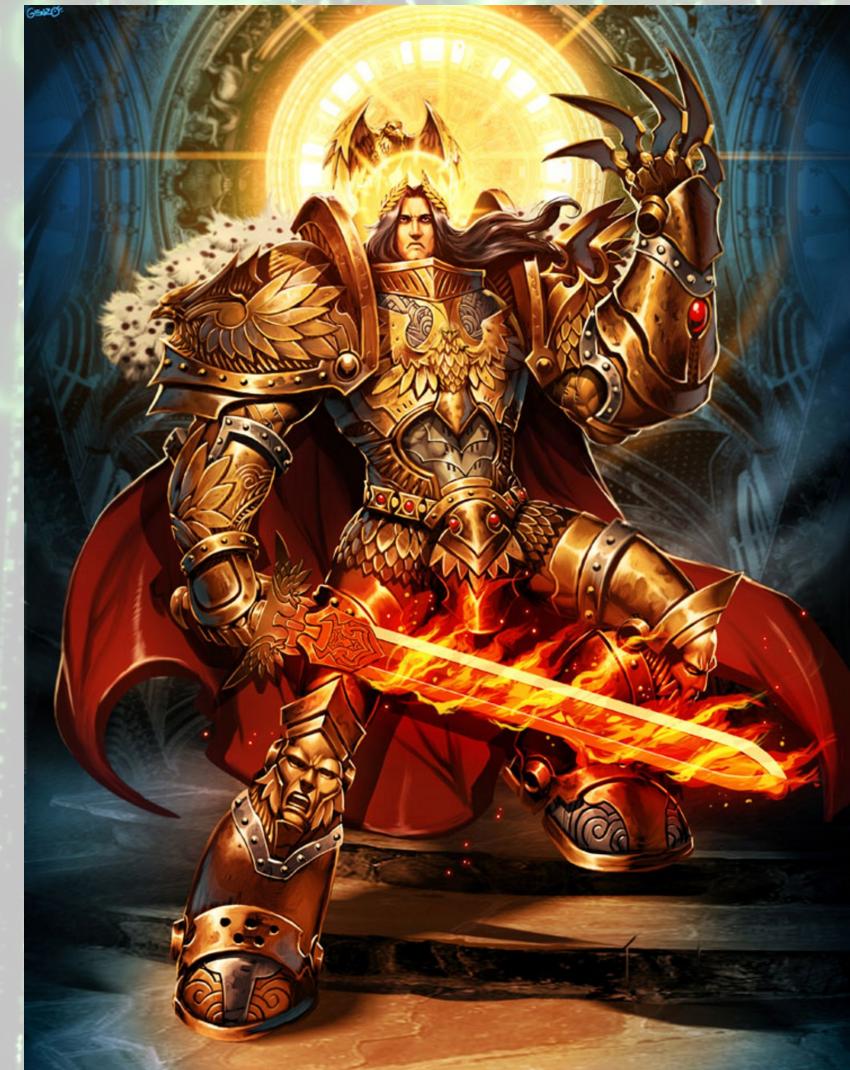
Zadanie 2.2. (0–2)

W plikach `slowa1.txt`, `slowa2.txt` i `slowa3.txt` znajdują się po trzy wiersze:

- w pierwszym wierszu każdego pliku zapisana jest liczba całkowita dodatnia n , oznaczająca długość słowa
- w drugim wierszu zapisane jest n -literowe słowo s , składające się z małych liter alfabetu angielskiego a-z
- w третьim wierszu zapisane są dwie liczby k_1 i k_2 , oddzielone spacją.

Napisz program z zaimplementowaną funkcją `czy_mniejszy`. Jako wynik Twój program powinien wypisywać TAK lub NIE, w zależności od wyniku funkcji `czy_mniejszy`. Odpowiedzi dla poszczególnych plików zapisz w pliku `wyniki2_2.txt`.

Dla przykładowego pliku `sufiks_1.txt`, Twój program powinien dać odpowiedź: TAK, a dla przykładowego pliku `sufiks_2.txt` – odpowiedź: NIE.



Kontekst

Zadanie 2. Sufiksy

Słowo definiujemy jako ciąg złożony z małych liter alfabetu angielskiego.

Niech $s[1..n]$ będzie słowem o długości $n > 0$.

Sufiksem słowa s nazywamy każde jego podsłowo kończące na ostatniej pozycji słowa s . Sufiks $s[k..n]$ nazywamy k -tym sufiksem.

Przykład 1.

słowo $s[1..10] = \text{mascarpone}$ ma następujące sufiksy:

k	$s[k..n]$
1	mascarpone
2	ascarpone
3	scarpone
4	carpone
5	arpone
6	rpone
7	pone
8	one
9	ne
10	e

Uporządkowanie alfabetyczne wszystkich sufiksów słowa *mascarpone* daje następującą kolejność ich numerów (od najmniejszego): 5, 2, 4, 10, 1, 9, 8, 7, 6, 3:

k	$s[k..n]$
5	arpone
2	ascarpone
4	carpone
10	e
1	mascarpone
9	ne
8	one
7	pone
6	rpone
3	scarpone

Poniżej zapisano funkcję **czy_mniejszy($n, s, k1, k2$)**. Wynikiem funkcji jest wartość PRAWDA, gdy sufiks $s[k1..n]$ jest mniejszy w porządku alfabetycznym od sufiksu $s[k2..n]$ oraz FAŁSZ w przeciwnym przypadku.

Specyfikacja

Dane:

n – długość słowa,

$s[1..n]$ – słowo zapisane jako tablica znaków (numerowanych od 1),

$k1$ – numer pierwszego sufiksu ($1 \leq k1 \leq n$),

$k2$ – numer drugiego sufiksu ($1 \leq k2 \leq n$, $k1 \neq k2$).

Wynik:

PRAWDA jeśli sufiks $s[k1..n]$ jest mniejszy w porządku alfabetycznym od $s[k2..n]$, albo FAŁSZ – w przeciwnym wypadku.

czy_mniejszy ($n, s, k1, k2$)

$i \leftarrow k1$

$j \leftarrow k2$

dopóki ($i \leq n$ oraz $j \leq n$) wykonuj

jeżeli ($s[i] == s[j]$)

$i \leftarrow i + 1$

$j \leftarrow j + 1$

w przeciwnym razie

jeżeli ($s[i] < s[j]$)

zakończ z wynikiem PRAWDA

w przeciwnym razie

zakończ z wynikiem FAŁSZ

jeżeli ($j \leq n$)

zakończ z wynikiem PRAWDA

w przeciwnym razie

zakończ z wynikiem FAŁSZ

Pułapki maturalnej notacji

- Indeksowanie tablicy:
• Pseudokod w arkuszach zakłada, że liczymy od 1 (powinno być to wskazane)
- „Cięcie” tablicy:
• Zapis podobny jak w Pythonie, ale dwukropek leży ..
• Jak widzimy, obowiązuje indeksowanie włączne (elementy 1 i 10 wchodzą w skład „plastra”)
- Do tych reguł nie należy się przywiązywać – za każdym razem trzeba zwrócić uwagę jak należy odczytywać dany zapis

Potęga Pythona

- Czy wszystkie dane są nam potrzebne?
 - Przypomnienie: w każdym pliku dostajemy trzy wiersze. Jeden zawiera długość słowa, drugi samo słowo, a trzeci dwie liczby k1 i k2.

Potęga Pythona

- Czy wszystkie dane są nam potrzebne?
 - Przypomnienie: w każdym pliku dostajemy trzy wiersze. Jeden zawiera długość słowa, drugi samo słowo, a trzeci dwie liczby k1 i k2.
- Linię z długościami słów możemy zignorować – Python sam może sprawdzić długość słów.
- Tak samo jest z dołączonym opisem funkcji – u nas wystarczy porównanie (zwróćcie jednak uwagę na przypadki brzegowe – co, jeśli oba sufiksy są identyczne?)
- Takie informacje są potrzebne użytkownikom języka C++, który działa na dużo niższym poziomie i pewne operacje trzeba realizować ręcznie

Czy porównanie stringów wystarczy?

- Czy zawsze dobrze wiemy, jak działa dana funkcja języka programowania?
- Nietrudno byłoby sobie wyobrazić sytuację, gdzie porównanie np. zawsze uznaje krótszy napis za mniejszy
- Jak sobie radzić takimi sytuacjami?

Czy porównanie stringów wystarczy?

- Czy zawsze dobrze wiemy, jak działa dana funkcja języka programowania?
- Nietrudno byłoby sobie wyobrazić sytuację, gdzie porównanie np. zawsze uznaje krótszy napis za mniejszy
- Jak sobie radzić takimi sytuacjami?
- Zazwyczaj da się napisać prosty test. Najlepiej jest wyszukać przypadek skrajny, aby mieć pewność co do wyniku.
- Czasem test jest szybszy niż szukanie w internecie

```
>>> "abc" < "abcabc"  
True
```

```
>>> "abc" > "z"  
False  
>>> "abc" < "z"  
True  
>>> "aaa" < "aaa"  
False
```

Rozwiążanie

Loading . . .

Rozwiązanie

```
plik1 = open("informatyka-2023-czerwiec-matura-rozszerzona-zalaczniki/slowa1.txt")
plik2 = open("informatyka-2023-czerwiec-matura-rozszerzona-zalaczniki/slowa2.txt")
plik3 = open("informatyka-2023-czerwiec-matura-rozszerzona-zalaczniki/slowa3.txt")

aktualny_plik = plik1

aktualny_plik = aktualny_plik.readlines()
slowo = aktualny_plik[1].strip()
k1 = int(aktualny_plik[2].split(" ")[0]) - 1
k2 = int(aktualny_plik[2].split(" ")[1]) - 1

sufiks1 = slowo[k1:]
sufiks2 = slowo[k2:]
print("s1: " + sufiks1 + " s2: " + sufiks2)
print(sufiks1 < sufiks2)
```

Jedziemy dalej

Zadanie 2.3. (0–3)

Dana jest dodatnia liczba całkowita n oraz słowo $s[1..n]$. Naszym celem jest obliczenie wartości elementów tablicy $T[1..n]$ zawierającej numery sufiksów słowa $s[1..n]$ uporządkowanych w porządku alfabetycznym.

Przykład:

dla słowa *mascarpone* wynikowa tablica T to [5, 2, 4, 10, 1, 9, 8, 7, 6, 3],

dla słowa *kalafiorowa* wynikowa tablica T to [11, 4, 2, 5, 6, 1, 3, 7, 9, 8, 10].

Z wykorzystaniem funkcji `czy_mniejszy(n, s, k1, k2)` zapisz w wybranej przez siebie notacji (w postaci pseudokodu lub w wybranym języku programowania) algorytm, który obliczy wartości elementów tablicy T zawierającej numery sufiksów zgodnie z porządkiem alfabetycznym sufiksów słowa s .

Uwaga: w zapisie możesz wykorzystać tylko operacje arytmetyczne (dodawanie, odejmowanie, mnożenie, dzielenie, dzielenie całkowite, reszta z dzielenia), odwoływanie się do pojedynczych elementów tablicy, porównywanie liczb lub znaków, instrukcje sterujące i przypisania lub samodzielnie napisane funkcje zawierające wyżej wymienione operacje.



Już to widzieliśmy

- Znowu obowiązują nas ograniczenia odnośnie elementów języka, które możemy wykorzystać
- Przypomnijmy, są to:
 - Operacje arytmetyczne
 - Porównania
 - Zmienne (pojedyncze oraz tablice)
 - if, else, while, for (instrukcje sterujące)
 - Samodzielnie napisane funkcje
- Tutaj dodatkowo możemy użyć funkcji `czy_mniejszy()`. Zakładamy, że jest ona już napisana i nie musimy jej ręcznie definiować.

Wskazówki

- Zadanie maturalne nie być napisane przesadnie efektywnie
- Priorytetem jest dobry wynik i krótki czas pisania kodu
- Często istnieje „sprintniejsze” rozwiązanie o mniejszej złożoności obliczeniowej – to jest jednak pułapka. Szukanie tego rozwiązania potrwa zbyt długo.
- Szukając odpowiedzi, należy zwrócić uwagę na to, czego dokładnie dotyczy pytanie – tutaj mamy znaleźć numery sufiksów uporządkowanych alfabetycznie

Wskazówki II

- Szukając odpowiedzi, należy zwrócić uwagę na to, czego dokładnie dotyczy pytanie – tutaj mamy znaleźć pozycję każdego sufiksu, jeśli uporządkowalibyśmy je alfabetycznie
- We wcześniejszej części arkusza jest podpowiedź!
- Czyli mówiąc prościej, indeks to liczba wyrażająca...

Uporządkowanie alfabetyczne wszystkich sufiksów słowa *mascarpone* daje następującą kolejność ich numerów (od najmniejszego): 5, 2, 4, 10, 1, 9, 8, 7, 6, 3:

k	$s[k..n]$
5	arpone
2	ascarpone
4	carpone
10	e
1	mascarpone
9	ne
8	one
7	pone
6	rpone
3	scarpone

Wskazówki II

- Szukając odpowiedzi, należy zwrócić uwagę na to, czego dokładnie dotyczy pytanie – tutaj mamy znaleźć pozycję każdego sufiksu, jeśli uporządkowalibyśmy je alfabetycznie
- We wcześniejszej części arkusza jest podpowiedź!
- Czyli mówiąc prościej, indeks sufiksu w tablicy to liczba wyrażająca...
- ...liczbę sufiksów w słowie, które **są mniejsze** od zadanego.

Rozwiążanie

Loading . . .

Rzwiązanie

```
tablica = []
slowo = "mascarpone"
for i in range(len(slowo)):
    tablica.append(0)

for i in range(len(slowo)):
    sufiks = slowo[i:]
    pozycja = 0
    for j in range(len(slowo)):
        if czy_mniejszy(len(slowo), slowo, j, i):
            pozycja += 1
    #print(sufiks + ":" + str(pozycja))
    tablica[pozycja] = (i+1) # liczymy od 1

print(tablica)
```

Kolejne zadanie

Zadanie 2.4. (0–3)

W pliku `slowa4.txt` znajduje się 10 wierszy. Każdy wiersz zawiera liczbę n ($1 \leq n \leq 100$) oraz n -literowe słowo s składające się z małych liter alfabetu angielskiego. Dane w wierszu są oddzielone znakiem odstępu.

Napisz program, który dla każdego słowa s z pliku wypisze jego sufiks najmniejszy w porządku alfabetycznym.

Przykład:

Sufiksem najmniejszym w porządku alfabetycznym dla słowa *mascarpone* jest *arpone*, a dla słowa *truskawki* sufiksem najmniejszym w porządku alfabetycznym jest *awki*.

Dla przykładowego pliku `sufiks_4.txt`, zawierającego tylko 4 wiersze (ze słowami: *banan*, *mascarpone*, *abcaabbaabbccba*, *maturazinformatyki*), Twój program powinien dać odpowiedź:

an
arpone
a
aturazinformatyki

Wyniki zapisz w pliku `wyniki2_4.txt`, każdy sufiks w oddzielnym wierszu, zgodnie z kolejnością danych w pliku `slowa4.txt`.



Pomysły?

+ + + + +
+ + + + +
+ + + + +
+ + + + +
+ + + + +
+ + + + +
+ + + + +
+ + + + +
+ + + + +
+ + + + +

WYDZIAŁ
EDUKACJI
SPOŁECZNO-
ROZWOJOWEJ
i
SPOŁECZNOSTY
MEDIÓW

Pomysły?

- Chyba już widzieliśmy podobne zadania
- To jest *de facto* wyszukiwanie najmniejszego elementu tablicy
- Nie ma zatem potrzeby porównywać każdego sufiku ze wszystkimi innymi (tutaj byłoby to rozwiązanie dość skomplikowane).
- **Pytanie bonusowe:** W tym zadaniu słowa są długie (100 znaków). Ile porównań musielibyśmy wykonać, gdybyśmy porównywali każdy sufiks z każdym?

Rozwiążanie

Loading . . .

Rozwiązanie

```
for s in lines:
    slowo = s.split(" ")[1]
    najmniejszy_sufiks = slowo
    for i in range(len(slowo)-1):
        sufiks = slowo[i:]
        if sufiks < najmniejszy_sufiks:
            najmniejszy_sufiks = sufiks
print(najmniejszy_sufiks)
```

To jest względnie proste

Zadanie 3.1. (0–2)

Liczbę binarną nazywamy **zrównoważoną**, gdy zawiera tyle samo zer i jedynek, natomiast **prawie zrównoważoną**, gdy liczba jedynek różni się od liczby zer o 1.

Przykład:

Liczba 101010 jest liczbą **zrównoważoną**.

Liczba 1011010 jest liczbą **prawie zrównoważoną**.

Podaj, ile jest liczb binarnych **zrównoważonych** oraz ile jest liczb binarnych **prawie zrównoważonych** w pliku anagram.txt.

Dla danych z pliku przyklad.txt prawidłową odpowiedzią jest:

21

15



Rozwiazanie

Loading . . .

Rozwiązanie

```
zrownowazonych = 0
prawie_zrownowazonych = 0

for liczba in liczby:
    zer = 0
    jedynek = 0
    for d in liczba:
        print(d)
        if d == '1':
            jedynek += 1
        if d == '0':
            zer += 1
    if zer == jedynek:
        zrownowazonych += 1
    if abs(zer-jedynek) == 1:
        prawie_zrownowazonych += 1
```

Następne zadanie

Zadanie 3.2. (0–3)

Anagramy cyfrowe to liczby utworzone z tego samego zestawu cyfr ustawionych w różnych kolejnościach. Przy tym pierwsza cyfra liczby nie może być równa zero.

Przykład:

Z liczby 209 zapisanej dziesiętnie można utworzyć 4 anagramy: 209, 902, 290, 920.

Z liczby binarnej 11100 można utworzyć 6 różnych anagramów: 10011, 10101, 10110, 11001, 11010, 11100.

Znajdź wszystkie takie liczby binarne 8-cyfrowe w pliku `anagram.txt`, z których można utworzyć największą liczbę anagramów. Wypisz te liczby w kolejności, w jakiej występują w pliku `anagram.txt`.

Dla danych z pliku `przyklad.txt` prawidłową odpowiedzią jest:

10001011
10111000
10100111
11111000



Sugestia w poprzednim zadaniu

- Zadania maturalne często są skonstruowane w taki sposób, że rozwiązanie poprzedniego zadania daje nam narzędzia do rozwiązania kolejnego
- To nie jest reguła, ale w razie braku pomysłów, można się w ten sposób zainspirować
- Tutaj liczenie cyfr może być dobrym tropem

Ile mamy możliwych kombinacji

- Odpowiedzi na to pytanie udziela dział matematyki zwany kombinatoryką
- Kiedy układamy elementy n-elementowego zbioru, to liczba kombinacji wynosi $n * n-1 * n-2 * \dots * 2 * 1 = n!$
- Dlaczego?

Ille mamy możliwych kombinacji

$$n * n-1 * n-2 * \dots * 2 * 1$$

- Mamy n pustych miejsc i n elementów.
- Każdy element umieszczamy na jednym miejscu w szyku.
- Pierwszy może trafić na dowolne miejsce.
- Drugi element ma już pulę miejsc zmniejszoną o 1.
- Kolejny o 2
- I tak dalej...
- Dla ostatniego elementu zostaje tylko jedno miejsce

Jednakowe elementy

- Należy zwrócić uwagę, że jeśli mamy dwa jednakowe elementy, to nawet jeśli zamienimy je miejscami, wygląd zbioru elementów nie ulega zmianie.
- A zatem im więcej mamy identycznych elementów, tym mniej jest możliwych kombinacji.
- Ostatecznie otrzymujemy następujący wzór: $\frac{n!}{p_1! * p_2! \dots}$
P₁, P₂ itd. to liczba powtórzeń każdego elementu.

A jak to się ma do naszego zadania?

- Otrzymujemy zbiór siedmiu cyfr (pierwszą zawsze musi być 1 i uwzględnienie jej tylko utrudniłoby nam zadanie).
- Liczba kombinacji to zatem: $\frac{7!}{\text{zera!} * \text{jedynki!}}$
- Dla każdej liczby licznik będzie identyczny (rozpatrujemy liczby stałej długości) – możemy go zignorować.
- A zatem liczby mające najwięcej anagramów to takie, dla których wyrażenie $\text{zera!} * \text{jedynki!}$ jest najmniejsze
- **Pytanie bonusowe:** ile możliwych wartości może przyjąć to wyrażenie (dla zbiorów siedmiocyfrowych)?

Rozwiazanie

Loading . . .

Rozwiązanie

```
najmniejszy_mianownik = silnia(8) * silnia(8)
zbior_rozwiazan = []
for liczba in liczby:
    if(len(liczba) != 8):
        continue
    _liczba = liczba[1:]
    print(liczba)
    mianownik = oblicz_mianownik(_liczba)
    if mianownik == najmniejszy_mianownik:
        zbior_rozwiazan.append(liczba)
    if mianownik < najmniejszy_mianownik:
        zbior_rozwiazan = []
        zbior_rozwiazan.append(liczba)
        najmniejszy_mianownik = mianownik

print(zbior_rozwiazan)
```

```
def zer_jedynek(liczba : str):
    zer = 0
    jedynek = 0
    for d in liczba:
        if d == '1':
            jedynek += 1
        if d == '0':
            zer += 1
    return [zer,jedynek]

def silnia(liczba : int):
    wynik = 1
    for i in range(1,liczba+1):
        wynik *= i
    return wynik

def _oblicz_mianownik(z_j : [ int ]):
    return(silnia(z_j[0]) * silnia(z_j[1]))
```