

**WYPEŁNIA ZDAJĄCY**

**KOD**

--	--	--

**PESEL**

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

**Miejsce na naklejkę.**

*Sprawdź, czy kod na naklejce to*

**E-100.**

*Jeżeli tak – przyklej naklejkę.*

*Jeżeli nie – zgłoś to nauczycielowi.*

**EGZAMIN MATURALNY Z INFORMATYKI**  
**POZIOM ROZSZERZONY**  
**CZĘŚĆ I**

TERMIN: **marzec 2022 r.**

CZAS PRACY: **60 minut**

LICZBA PUNKTÓW DO UZYSKANIA: **15**

**WYPEŁNIA ZDAJĄCY**

**WYBRANE:**

.....  
(system operacyjny)

.....  
(program użytkowy)

.....  
(środowisko programistyczne)

**Instrukcja dla zdającego**

1. Sprawdź, czy arkusz egzaminacyjny zawiera 8 stron (zadania 1–3).  
Ewentualny brak zgłoś przewodniczącemu zespołu nadzorującego egzamin.
2. Odpowiedzi zapisz w miejscu na to przeznaczonym przy każdym zadaniu.
3. Pisz czytelnie. Używaj długopisu/pióra tylko z czarnym tuszem/atramentem.
4. Nie używaj korektora, a błędne zapisy wyraźnie przekreśl.
5. Pamiętaj, że zapisy w brudnopisie nie będą oceniane.
6. Wpisz zadeklarowane (wybrane) przez Ciebie na egzamin system operacyjny, program użytkowy oraz środowisko programistyczne.
7. Na tej stronie oraz na karcie odpowiedzi wpisz swój numer PESEL i przyklej naklejkę z kodem.
8. Nie wpisuj żadnych znaków w części przeznaczonej dla egzaminatora.



EINP-R1-**100**-2203

## Zadanie 1. Algorytm Levenshteina

Algorytm Levenshteina, służy do obliczania odległości edycyjnej (odległości Levenshteina). Jest to najmniejsza liczba działań prostych, przeprowadzająca jeden napis w drugi.

Za działania proste uznajemy:

- wstawienie nowego znaku,
- usunięcie znaku,
- zamiana na inny znak.

W ramach algorytmu zostaje utworzona tablica o wymiarach  $n + 1$  na  $m + 1$  gdzie  $n$  i  $m$  to długości porównywanych słów. Pierwszy wiersz i kolumnę uzupełniamy wartościami od 0 do, odpowiednio  $n$  i  $m$ .

Następnie porównujemy kolejne litery pierwszego słowa (indeksowane od  $i = 1$  do  $n$ ) ze wszystkimi literami drugiego słowa (indeksowane od  $j = 1$  do  $m$ ) stosując poniższą procedurę.

Jeśli literki są identyczne, ustawiamy *koszt* na 0, jeśli nie, na 1. Teraz musimy komórkę o indeksach  $[i, j]$  wypełnić wartością, którą będzie minimum z:

- wartości komórki  $[i, j - 1]$  zwiększonej o 1,
- wartości komórki  $[i - 1, j]$  zwiększonej o 1,
- wartości komórki  $[i - 1, j - 1]$  powiększonej o *koszt*.

Po wykonaniu wszystkich porównań, naszą odległością edycyjną będzie wartość w komórce  $[n + 1, m + 1]$ .

Poniżej przedstawiono tabelę sporządzoną w celu obliczenia odległości edycyjnej dla słów *foka* i *kotka*.

0	1	2	3	4	5
1	1	2	3	4	5
2	2	1	2	3	4
3	2	2	2	2	3
4	3	3	3	3	2

Liczba 2 umieszczona w prawym dolnym rogu oznacza odległość edycyjną obliczoną dla słów *foka* i *kotka*. Zatem od jednego do drugiego wyrazu możemy przejść w dwóch prostych działaniach:

- *foka* → *kotka*: 1) zamieniając *f* na *k*, 2) dodając literę *t*
- *kotka* → *foka*: 1) zamieniając *k* na *f*, 2) usuwając literę *t*.

### Zadanie 1.1. (0-1)

Dla podanych par słów ustal odległość edycyjną i ustal w jaki sposób przeprowadzić *słowo1* w *słowo2*.

<i>słowo1</i>	<i>słowo2</i>	Odległość edycyjna	Sposób przeprowadzenia
<i>foka</i>	<i>kotka</i>	2	1) zamień <i>f</i> na <i>k</i> 2) dodaj literę <i>t</i>
<i>pole</i>	<i>kot</i>		
<i>marka</i>	<i>ariada</i>		

Miejsce na obliczenia:

[illegible]

### Zadanie 1.2. (0-2)

Dla słów *lampa* i *fam* uzupełnij poniższą tabelkę w celu obliczenia odległości edycyjnej.

0	1	2	3	4	5
1					
2					
3					

### Zadanie 1.3. (0-3)

W wybranej przez siebie notacji (schemat blokowy, lista kroków, pseudokod, język programowania) zapisz algorytm obliczania odległości Levenshteina.

**Uwaga:** W zapisie algorytmu możesz wykorzystać tylko operacje arytmetyczne (dodawanie, odejmowanie, mnożenie, dzielenie, dzielenie całkowite, reszta z dzielenia), instrukcje porównania, instrukcje sterujące i przypisania do zmiennych lub samodzielnie napisane funkcje, wykorzystujące wyżej wymienione operacje. Ponadto masz do dyspozycji funkcję *min()*, która dla dowolnej liczby parametrów wejściowych (liczb całkowitych) zwraca liczbę najmniejszą spośród nich.

### Algorytm:

**Dane wejściowe:**

*słowo1*, *słowo2* – słowa, dla których obliczamy odległość edycyjną

$n, m$  – długości słów, odpowiednio: *słowo1* i *słowo2*

**Dane wyjściowe:**

*odleglosc* – odległość edycyjna dla słów: *slovo1* i *slovo2*

This image shows a full page of blank graph paper. The grid consists of thin, light gray horizontal and vertical lines that intersect to form small squares across the entire surface. There are no margins, text, or other markings on the paper.

## Zadanie 2. Test

Oceń, czy poniższe zdania są prawdziwe. Zaznacz P, jeśli zdanie jest prawdziwe, albo F, jeśli zdanie jest fałszywe. W każdym zadaniu uzyskasz punkt, jeśli poprawnie odpowiesz na wszystkie jego części.

### Zadanie 2.1. (0-1)

Na licencji ADWARE jest rozpowszechniane oprogramowanie, które

1.	jest rozpowszechniane za darmo, ale zawiera funkcje wyświetlające reklamy.	P	F
2.	ma otwarty kod źródłowy.	P	F
3.	jest opłacane przez użytkownika.	P	F
4.	może być używane tylko przez z góry ustalony czas.	P	F

### Zadanie 2.2. (0-1)

Do jednoznacznego zakodowania znaków pięcioelementowego alfabetu wystarczą/y:

1.	2 bity.	P	F
2.	3 bity.	P	F
3.	5 bitów.	P	F
4.	8 bitów.	P	F

### Zadanie 2.3. (0-1)

Oceń prawdziwość podanych zdań.

1.	Plakat do druku lepiej przygotować w modelu barw RGB niż CMYK.	P	F
2.	Kolor żółty jest kolorem podstawowym w modelu RGB.	P	F
3.	W wyniku nałożenia się składowych Yellow i Magenta w modelu CMYK otrzymamy kolor czerwony.	P	F
4.	W modelu barw CMYK litera C pochodzi od angielskiego słowa <i>contrast</i> .	P	F

### Zadanie 3. Ciągi rekurencyjne

Dana jest następująca funkcja rekurencyjna:

funkcja **wynik(*i*)**

jeżeli  $i < 3$

zwróć 1 i zakończ

w przeciwnym razie

jeżeli  $i \bmod 2 = 0$

zwróć **wynik( $i - 3$ )** + **wynik( $i - 1$ )** + 1

w przeciwnym razie

zwróć **wynik( $i - 1$ )** mod 7

**Uwaga:** Operator mod oznacza resztę z dzielenia.

#### Zadanie 3.1. (0-2)

Uzupełnij poniższą tabelkę wpisując w drugiej kolumnie wynik wywołania funkcji dla argumentów z pierwszej kolumny.

<i>i</i>	<b>wynik(<i>i</i>)</b>
2	1
3	
4	
5	
6	
7	
8	

#### Zadanie 3.2. (0-2)

Wykonaniem elementarnym nazywać będziemy wykonanie **wynik(0)**, **wynik(1)** lub **wynik(2)**. Natomiast złożonością elementarną **wynik(*i*)** nazywamy liczbę wykonań elementarnych będących efektem uruchomienia **wynik(*i*)**. Złożoność elementarną **wynik(*i*)** oznaczamy przez ***E*(*i*)**. Na przykład złożoność elementarna **wynik(4)** wynosi ***E*(4) = 2**, ponieważ wykonując **wynik(4)**, wywołamy **wynik(3)** i **wynik(1)** (wykonanie elementarne), a z kolei przy wykonaniu **wynik(3)** wywołamy **wynik(2)** (drugie wykonanie elementarne).

Uzupełnij tabelkę znajdującą się na następnej stronie.

$i$	$E(i)$
0	1
3	1
5	
7	
9	
10	

Okazuje się, że  $E(i)$  można opisać rekurencyjnym wyrażeniem, którego niekompletną postać podano poniżej. Uzupełnij brakujące miejsca tak, aby  $E(i)$  dawało poprawną złożoność elementarną **wynik**( $i$ ) dla każdego całkowitego nieujemnego  $i$ .

$$E(0) = E(1) = E(2) = 1$$

$$E(i) = E(\dots\dots\dots) + E(\dots\dots\dots) \quad \text{dla parzystego } i > 2$$

$$E(i) = E(\dots\dots\dots) \quad \text{dla nieparzystego } i > 2$$

### Zadanie 3.3. (0-2)

Naszym celem jest wyznaczenie największej liczby spośród wartości funkcji **wynik**(0), **wynik**(1), ..., **wynik**(1000) bez konieczności rekurencyjnego wyznaczania kolejnych wartości. Poniżej prezentujemy niekompletny algorytm realizujący to zadanie.

$W[0] \leftarrow 1$

$W[1] \leftarrow 1$

$W[2] \leftarrow 1$

$max\_wart \leftarrow 1$

**dla**  $i = 3, 4, \dots, 1000$  **wykonuj**

**jeżeli**  $i \bmod 2 = 0$

$W[i] \leftarrow \dots\dots\dots$

**w przeciwnym razie**

$W[i] \leftarrow \dots\dots\dots$

**jeżeli**  $W[i] > max\_wart$

$\dots\dots\dots$

**zwróć**  $max\_wart$

Uzupełnij brakujące miejsca w algorytmie tak, aby zwracał on największą liczbę spośród **wynik**(0), **wynik**(1), ..., **wynik**(1000).

**BRUDNOPIS (nie podlega ocenie)**

