Zadanie: Odległość edycyjna

Etap 1 (0.5 pkt)

Odległością edycyjną nazywamy miarę podobieństwa dwóch wyrazów. Istnieje wiele różnych algorytmów wyznaczających tę miarę, najpopularniejszym z nich jest algorytm Levenshteina.

Algorytm ten pozwala na wyznaczenie ciągu prostych działań przeprowadzających jeden ciąg znaków w drugi. Podstawowa wersja algorytmu zakłada trzy takie operacje: wstawienie nowego znaku, usunięcie znaku, oraz zamianę pojedynczego znaku na inny.

Idea działania algorytmu jest bardzo prosta i opiera się o wykorzystanie programowania dynamicznego. W podstawowej wersji o złożoności pamięciowej $O(mn)$ (gdzie m to długość pierwszego słowa, a n to długość drugiego słowa), polega na wykorzystaniu dwuwymiarowej tablicy o wymiarach $n+1$ na $m+1$ do wyznaczenia odległości dla każdego ze słów prefiksowych. Słowo prefiksowe to podsłowo zawierające $n \geqslant 0$ pierwszych znaków słowa. Każda kolumna i wiersz odpowiada kolejno jednej litrze słowa dodatkowo uwzględniając pusty prefiks.

Pierwszy wiersz reprezentuje odpowiednio odległości dla słowa pustego do danego prefiksu drugiego słowa, co jesteśmy w stanie łatwo wyznaczyć - jest to wykonanie odpowiednio x razy operacji wstawienia nowego znaku, której koszt znamy. Podobnie pierwsza kolumna odpowiada odległościom dla danych prefiksów słowa pierwszego do pustego słowa i odpowiada operacjom usunięcia kolejnych znaków.

	␣	F	O	K	A
␣	0	1	2	3	4
K	1				
O	2				
T	3				

Tablica 1: Pierwszy krok wypełnienia tablicy odległości dla słów KOT i FOKA

Mając tak wypełnioną tablicę można wyznaczyć resztę wartości wykorzystując fakt, że:

1. Krok w prawo, to dodanie znaku
2. Krok w dół, to usunięcie znaku
3. Krok w po skosie, to zmiana znaku z jednego na drugi

a	b
c	

Nowa wartość d to minimum tych trzech wartości (a, b, c) powiększona odpowiednio o koszty poszczególnych operacji. Dla kroku po skosie (zamiany znaków) należy sprawdzić, czy takowa zmiana zaszła. Jeżeli znaki się nie różnią nie należy dodawać kosztu.

$$d = min \begin{cases} \begin{cases} a + changeCost & \text{if letters have changed} \\ a & \text{otherwise} \end{cases} \\ b + removeCost \\ c + addCost \end{cases}$$

	␣	F	O	K	A
␣	0	1	2	3	4
K	1	1	2	2	3
O	2	2	1	2	3
T	3	3	2	2	3

Tablica 2: Wypełniona tablica dla słów KOT i FOKA

Wynik (odległość dla całych słów) można odczytać z prawego dolnego rogu tablicy.
Jako koszty poszczególnych operacji przyjmij wartość 1.0.

Etap 2 (1.0 pkt)

Wykorzystując tablicę dystansów z etapu 1. dodaj zwracanie jako parametr *possibleEditSequences* listy list operacji przeprowadzających jedno słowo w drugie. Należy zwrócić jedynie te o minimalnym koszcie.

W tym celu należy przejść z powrotem z dolnego prawego rogu do górnego lewego znajdując jedynie ścieżki o minimalnym koszcie.

Etap 3 (1.0 pkt)

Dodaj czwartą operację transpozycji, którą funkcja powinna wykorzystywać, jeśli flaga *allowTranspositions* ma wartość *true*. Transpozycja polega na zamianie dwóch sąsiadujących ze sobą liter w słowach. Np. ze słowa ABCD można uzyskać słowo ACBD wykonując jedynie jedną operację transpozycji liter B i C.

Jako koszt transpozycji przyjmij 1.0.

Wskazówka: Zastanów się jak operacja transpozycji jest reprezentowana w tablicy odległości.