## Algorytm: Wyszukiwanie liniowe (Linear Search)

**Jak działa?**  
Przeszukuje kolejne elementy tablicy (lub listy) od początku do końca, porównując każdy element z wartością, której szukamy. Jeśli znajdzie element równy szukanej wartości, zwraca jego indeks. Jeśli nie znajdzie, zwraca informację o braku elementu (np. -1). Jest prosty, ale może być wolny przy dużych zbiorach danych.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

**Omówienie kodu:**

* Funkcja przyjmuje tablicę słów (dane) i słowo do znalezienia (szukaneSlowo).
* Zmienna kroki to licznik, który informuje, ile porównań zostało wykonanych.
* Pętla for przechodzi po każdym słowie, porównując je do szukanego.
* Jeśli znaleziono słowo, zwraca jego indeks i kończy działanie.
* Jeśli po przejściu całej tablicy nie znaleziono słowa, zwraca -1.

**Reszta kodu w metodzie Start():**

* Wczytuje słowa z pliku (za pomocą funkcji Utils.WczytajSlowaZPliku — prawdopodobnie zwraca tablicę stringów).
* Dla liter a, c, d, m, w, z wyszukuje pierwsze słowo zaczynające się na daną literę.
* Jeśli takie słowo istnieje, używa wyszukiwania liniowego, żeby policzyć, po ilu krokach jest znalezione.
* Wypisuje wyniki na konsolę.
* Następnie wczytuje drugi plik, bierze ostatnie słowo i sprawdza, ile kroków zajmuje jego znalezienie liniowe.

## Algorytm: Wyszukiwanie binarne (Binary Search)

**Jak działa?**  
Działa na posortowanej liście lub tablicy. Na początku sprawdza element środkowy i porównuje go z szukaną wartością. Jeśli to nie jest szukany element, eliminuje połowę tablicy (lewo lub prawo od środka) w zależności od wyniku porównania, po czym powtarza proces na pozostałej połowie. Działa bardzo szybko – w czasie logarytmicznym względem liczby elementów.

**Omówienie kodu:**

**Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.**

**Reszta metody Start():**

* Wczytuje słowa z pliku (znowu Utils.WczytajSlowaZPliku).
* Dla liter a, c, d, m, w, z wyszukuje pierwsze słowo zaczynające się na tę literę.
* Jeśli takie słowo jest znalezione, wywołuje wyszukiwanie binarne, by sprawdzić, ile kroków zajmuje znalezienie słowa.
* Wypisuje wyniki na konsolę.
* Na końcu wczytuje inny plik, bierze ostatnie słowo i sprawdza, ile kroków zajmie jego znalezienie binarne.

## Algorytm: Sortowanie przez wybieranie (Selection Sort)

**Jak działa?**  
Działa iteracyjnie — dla każdej pozycji w tablicy szuka najmniejszego elementu spośród nieposortowanych i zamienia go z elementem na tej pozycji. W ten sposób krok po kroku buduje posortowaną część tablicy od początku. Ma złożoność czasową O(n²), więc jest powolny przy dużych danych, ale prosty do zrozumienia i implementacji.

**Omówienie kodu:**

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

**Metoda Start():**

* Wczytuje słowa z pliku (plik1 i plik2).
* Wywołuje funkcję SelectionSort na tych tablicach.
* Wypisuje ile słów zostało posortowanych i ile kroków (porównań) zajęło sortowanie.

## Algorytm: QuickSort (Sortowanie szybkie)

**Jak działa?**  
Dzieli tablicę na dwie części względem tzw. pivota (elementu wzorcowego) tak, że elementy mniejsze od pivota znajdują się po lewej, a większe po prawej. Następnie rekurencyjnie sortuje każdą z tych części. Działa bardzo szybko w praktyce — średnia złożoność to O(n log n).

**Omówienie kodu:**

**Funkcja QuickSort (rekurencyjna):**

**Obraz zawierający tekst, Czcionka, oprogramowanie, zrzut ekranu

Zawartość wygenerowana przez AI może być niepoprawna.**

* lewy i prawy to indeksy graniczne aktualnie sortowanego fragmentu tablicy.
* Rekurencja działa, dopóki lewy < prawy.

**Funkcja Partition (podział tablicy względem pivota):**

**Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.**

* Przesuwamy elementy tak, by te mniejsze lub równe pivotowi znalazły się przed nim.
* Po zakończeniu pętli umieszczamy pivot na jego ostatecznej pozycji.

**Funkcja Zamien (zamiana elementów):**

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

* Zamienia miejscami dwa elementy w tablicy.

**Metoda Start():**

* Wczytuje słowa z plików.
* Wywołuje QuickSort na każdej tablicy, licząc kroki (porównania).
* Wyświetla, ile słów posortowano i ile kroków to zajęło.

## Operacje na macierzach — dodawanie, mnożenie, transpozycja

### Dodawanie macierzy

**Jak działa?**  
Dodajemy do siebie odpowiadające elementy dwóch macierzy o tych samych wymiarach, tworząc nową macierz wynikową o tych samych wymiarach.

### Mnożenie macierzy

**Jak działa?**  
Mnożymy macierz A (rozmiar r1 x c1) przez B (r2 x c2) tylko wtedy, gdy liczba kolumn A (c1) jest równa liczbie wierszy B (r2). Element na pozycji (i, j) macierzy wynikowej to suma iloczynów elementów i-tego wiersza A i j-tej kolumny B.

### Transpozycja macierzy

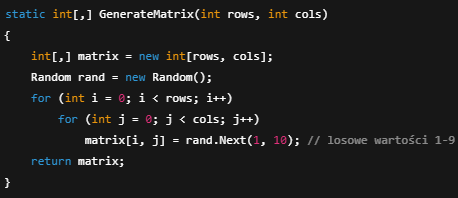
**Jak działa?**  
Zamienia wiersze na kolumny — element (i, j) oryginalnej macierzy staje się elementem (j, i) macierzy transponowanej.

### Omówienie kodu:

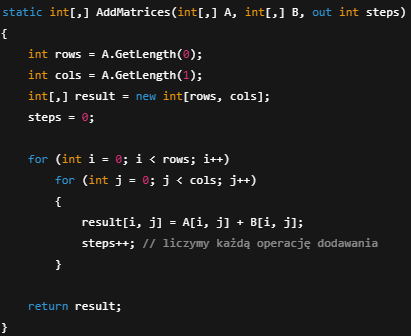
#### Metoda Start():

* Testuje kolejno dodawanie macierzy o rozmiarach 2x2, 3x3, 4x4, 5x5.
* Testuje mnożenie macierzy z różnymi wymiarami, w tym przypadek błędnych wymiarów.
* Tworzy macierz 3x2, wypisuje ją, transponuje i wypisuje wynik oraz liczbę kroków (operacji).

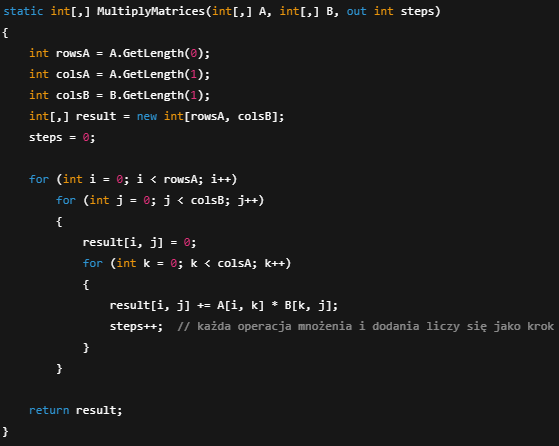
#### Generowanie macierzy:



#### Dodawanie macierzy:

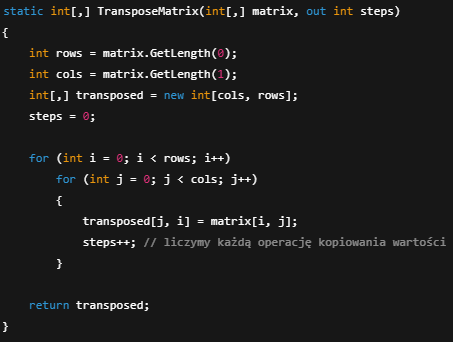


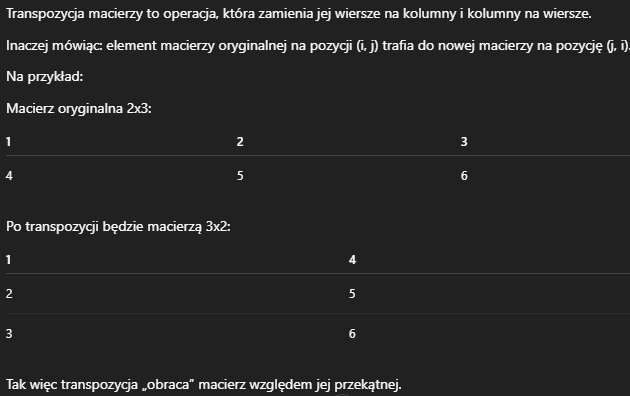
#### Mnożenie macierzy:



Sprawdzenie wymiarów jest w TestMultiplication — mnożenie odbywa się tylko, jeśli kolumny A = wiersze B.

#### Transpozycja macierzy:





## Obliczanie n-tej liczby Fibonacciego

#### 1. Rekurencja

**Jak działa?**  
Funkcja wywołuje sama siebie, obliczając Fib(n) = Fib(n-1) + Fib(n-2) aż do wartości bazowych (n=0 lub n=1). Proste, ale bardzo nieefektywne, bo dużo powtarzających się obliczeń.

#### 2. Memoizacja (zapamiętywanie wyników)

**Jak działa?**  
Rekurencja z zapamiętywaniem wcześniej obliczonych wartości w słowniku (cache). Dzięki temu każda wartość jest liczona tylko raz, co znacznie przyspiesza obliczenia.

#### 3. Bottom-up (programowanie dynamiczne)

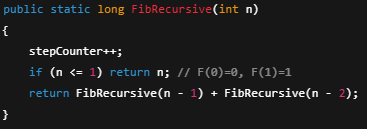
**Jak działa?**  
Najpierw oblicza i zapisuje w tablicy wartości od 0 do n w kolejności rosnącej, wykorzystując wcześniej obliczone liczby, a potem zwraca wynik. To najbardziej efektywna metoda z trzech.

### Omówienie kodu:

#### Zmienna globalna licząca kroki:

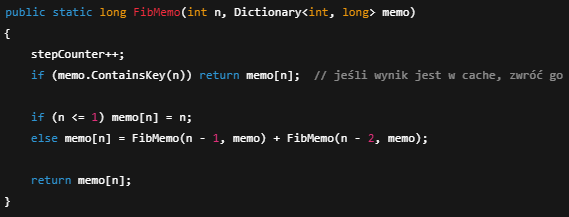
static long stepCounter = 0;

#### Rekurencja:



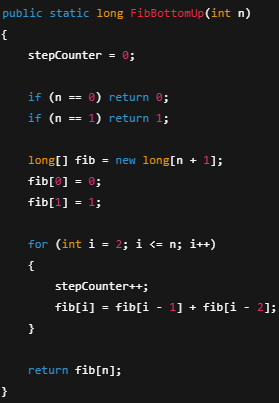
* Wywołuje się rekurencyjnie dwa razy dla każdego n>1.
* Bardzo wolne dla dużych n, bo liczba wywołań rośnie wykładniczo.

#### Memoizacja:



* Każdy wynik jest zapisywany w memo.
* Unika powtarzających się obliczeń, działa dużo szybciej.

#### Bottom-up:



* Buduje tablicę od dołu.
* Bardzo szybkie i efektywne.

### W metodzie Start():

* Pokazuje wyniki dla n=30 rekurencyjnie (ostrzega, że powyżej 40 to wolne).
* Pokazuje wynik dla n=357 memoizacją i bottom-up.
* Wypisuje liczbę kroków (wywołań lub iteracji).

## BFS (Breadth-First Search) i DFS (Depth-First Search)

### 1. BFS — Przeszukiwanie wszerz

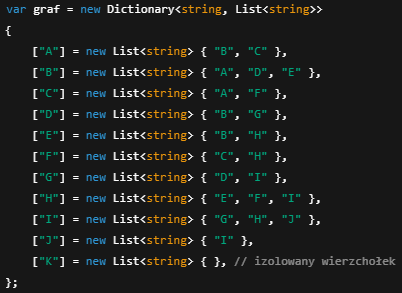
**Jak działa?**  
Przeszukuje graf poziomami, zaczynając od wierzchołka startowego, odwiedzając najpierw wszystkie jego bezpośrednie sąsiednie wierzchołki, potem sąsiednie tych wierzchołków itd. Wykorzystuje kolejkę (FIFO).

### 2. DFS — Przeszukiwanie w głąb

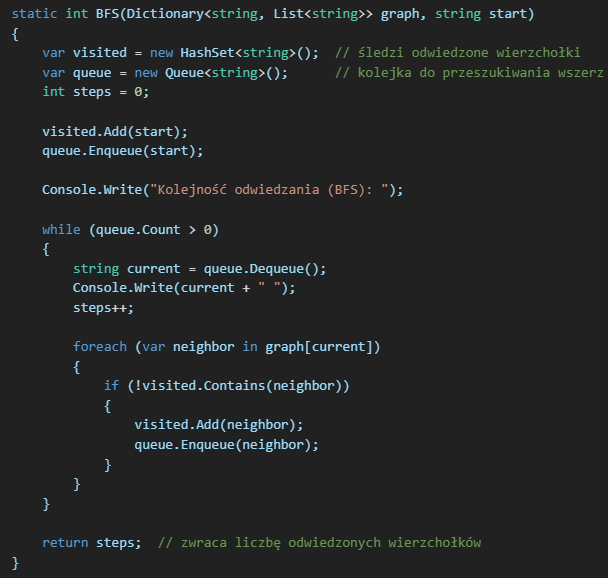
**Jak działa?**  
Przeszukuje graf wgłąb, idąc wzdłuż jednej ścieżki tak daleko, jak to możliwe, zanim się cofnie i przejdzie do kolejnej ścieżki. Wykorzystuje rekurencję lub stos.

### Omówienie kodu:

#### Definicja grafu:

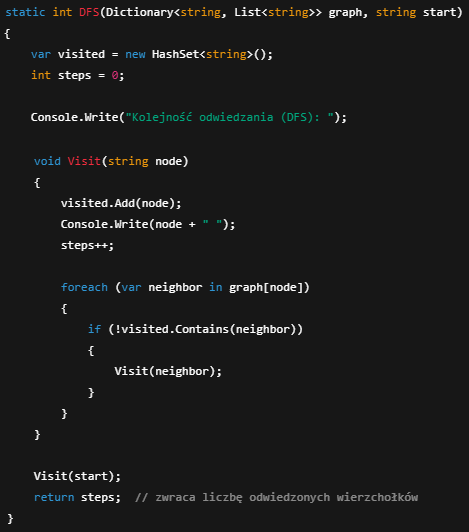


#### BFS:



* queue przechowuje wierzchołki do odwiedzenia.
* Zaczynamy od start.
* Dla każdego wierzchołka dodajemy nieodwiedzonych sąsiadów do kolejki.
* Wypisuje kolejność odwiedzin i liczy kroki.

#### DFS:



* Rekurencyjnie odwiedzamy wierzchołki i ich sąsiadów, idąc w głąb.
* Odwiedzamy każdego sąsiada, jeśli nie był jeszcze odwiedzony.
* Wypisuje kolejność odwiedzin i liczy kroki.

## Wykrywanie cyklu w grafie nieskierowanym

### 1. Wykrywanie cyklu metodą BFS

**Jak działa?**  
Przeszukuje graf wszerz, pamiętając dla każdego wierzchołka jego rodzica (skąd został odwiedzony). Jeśli podczas przeszukiwania napotka się na już odwiedzony wierzchołek, który nie jest rodzicem obecnego wierzchołka, oznacza to wykrycie cyklu.

### 2. Wykrywanie cyklu metodą DFS

**Jak działa?**  
Przeszukuje graf wgłąb, rekurencyjnie odwiedzając sąsiadów i pamiętając rodzica aktualnego wierzchołka. Jeżeli podczas odwiedzania napotka wierzchołek już odwiedzony, który nie jest rodzicem, wykrywa cykl i przerywa dalsze przeszukiwanie.

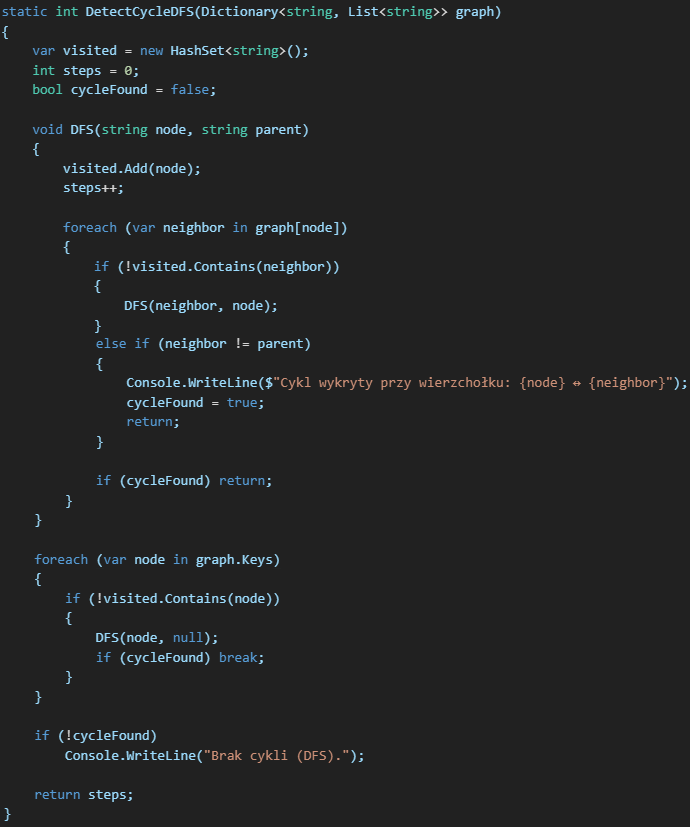
### Omówienie kodu:

#### Funkcja BFS do wykrywania cyklu:



* parent przechowuje rodzica wierzchołka.
* Jeśli znajdziemy sąsiada odwiedzonego, który nie jest rodzicem, oznacza to cykl.

#### Funkcja DFS do wykrywania cyklu:



* DFS rekurencyjnie odwiedza wierzchołki.
* Sprawdza, czy odwiedzony wierzchołek nie jest rodzicem — jeśli tak, to jest cykl.
* Po wykryciu cyklu przerywa dalsze przeszukiwanie.

### Podsumowanie:

* Oba algorytmy sprawdzają podczas przeszukiwania, czy trafiają na odwiedzony wierzchołek, który nie jest rodzicem.
* To pozwala rozróżnić cykle od prostych dróg powrotnych.
* Wypisują informację o wykryciu cyklu i liczą liczbę wykonanych kroków.

## Operacje na kolejce z posortowanymi słowami

### Jak działa kod:

1. **Wczytanie i posortowanie słów**  
   Wczytuje słowa z pliku (Utils.WczytajSlowaZPliku(Utils.plik2)), sortuje je rosnąco (OrderBy(w => w)), a potem konwertuje na tablicę.
2. **Sprawdzenie liczby słów**  
   Jeśli jest ich mniej niż 1000, wyświetla błąd i przerywa działanie.
3. **Utworzenie kolejki (Queue) z posortowanych słów**
4. **Usunięcie elementów z kolejki od indeksu 99 do 199 (czyli 100 elementów)**
   * Konwertuje kolejkę na listę (ToList()),
   * usuwa zakres elementów RemoveRange(99, 101),
   * tworzy nową kolejkę z pozostałych elementów.
5. **Wyświetlenie zawartości kolejki po usunięciu elementów**
   * Wypisuje słowa, co 20 wyrazów przechodzi do nowej linii,
   * na końcu wyświetla liczbę pozostałych elementów.

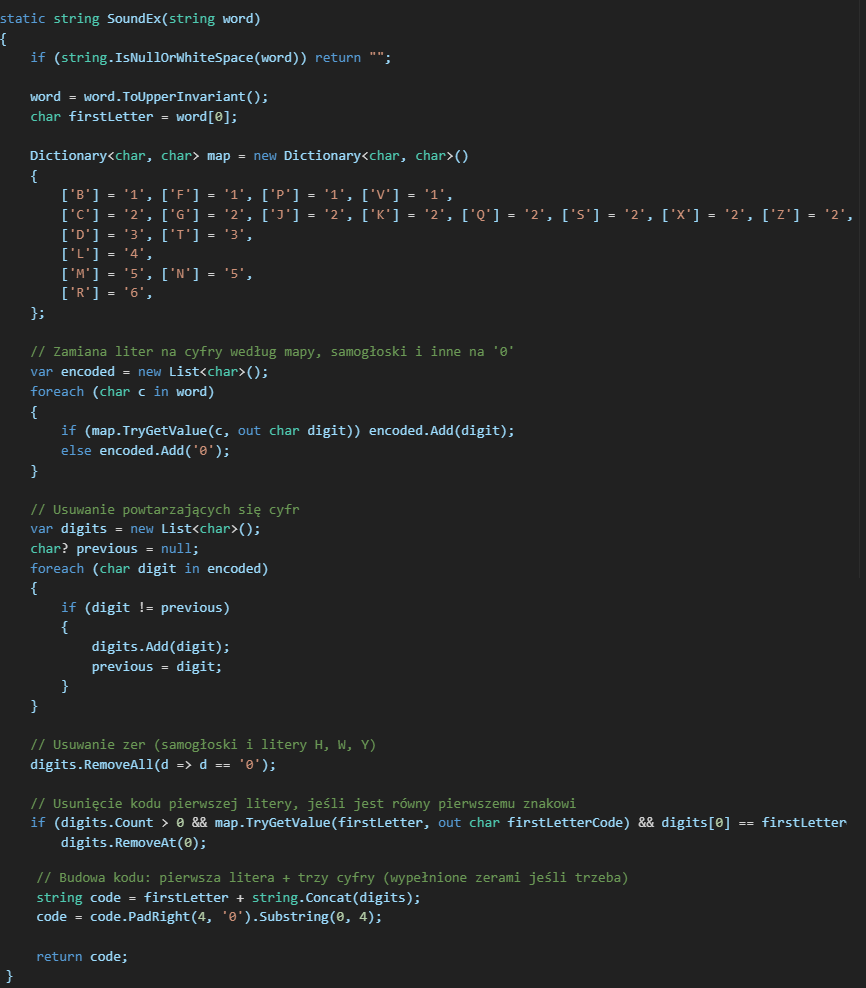
### Szczegóły ważne:

* Indeksy w RemoveRange są zero-based, więc indeks 99 to 100. element w kolejce.
* Usuwane jest 101 elementów — czyli elementy od 100 do 200 (włącznie).
* Queue nie ma metody usuwania elementów po indeksie, dlatego jest zamieniana na listę do modyfikacji.

## Algorytm SoundEx

**Jak działa?**  
Przekształca słowo w kod fonetyczny, który grupuje podobnie brzmiące słowa pod tym samym kodem. Bazuje na przypisaniu literom cyfr według grup dźwięków, usuwa powtarzające się cyfry i samogłoski (oraz litery H, W, Y), a wynik kodu to 4-znakowy ciąg zaczynający się od pierwszej litery słowa.

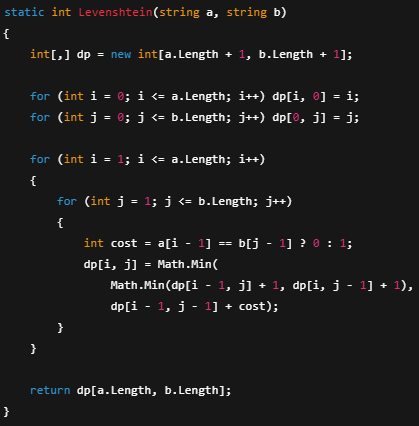
### Omówienie funkcji SoundEx:



### Algorytm Levenshtein (odległość edycyjna)

**Jak działa?**  
Mierzy minimalną liczbę operacji (wstawień, usunięć lub zamian znaków), potrzebnych do zamiany jednego słowa na drugie. Używa programowania dynamicznego, tworząc tablicę wyników dla kolejnych prefiksów słów.

### Omówienie funkcji Levenshtein:



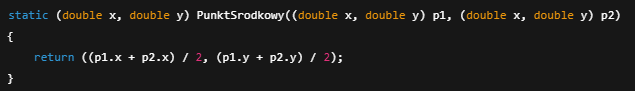
* dp[i, j] to minimalny koszt zamiany pierwszych i znaków a na pierwsze j znaków b.
* Wypełnia tablicę od podstaw, biorąc pod uwagę wstawienia, usunięcia i zamiany.

### Funkcje pomocnicze:

* PorownajSoundEx — porównuje wszystkie pary słów, wypisując te, które mają taki sam kod SoundEx.
* PorownajLevenshtein — porównuje wszystkie pary słów, grupując je według wartości odległości Levenshteina i wypisując je.

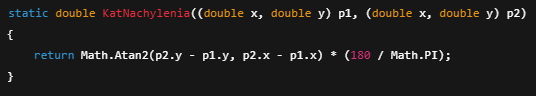
## Punkt środkowy odcinka

**Jak działa?**  
Środek odcinka między punktami p1 i p2 to średnia arytmetyczna ich współrzędnych x i y.



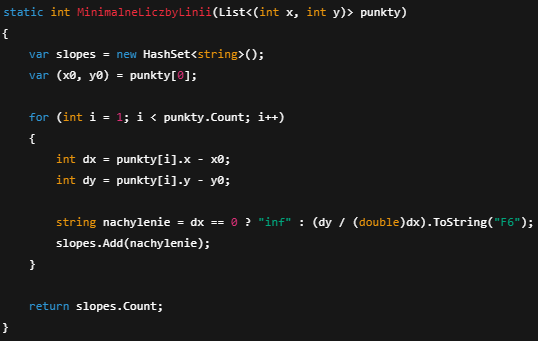
### 2. Kąt nachylenia odcinka względem osi X

**Jak działa?**  
Używa funkcji Math.Atan2 do wyliczenia kąta między odcinkiem a osią X, konwertując radiany na stopnie.



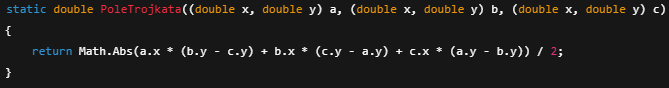
### 3. Minimalna liczba różnych linii przechodzących przez punkty względem punktu odniesienia (0,0)

**Jak działa?**  
Dla punktów w liście wylicza współczynnik kierunkowy (nachylenie) linii przechodzących przez pierwszy punkt i każdy inny, dodając je do zbioru (HashSet) by wyeliminować duplikaty. Wynik to liczba różnych nachyleń — czyli minimalna liczba różnych linii.



### 4. Pole trójkąta na podstawie współrzędnych wierzchołków

**Jak działa?**  
Stosuje wzór na pole trójkąta z współrzędnych (wzór Herona lub wzór determinantowy).



### 5. Relacja linii z okręgiem

**Jak działa?**  
Wylicza odległość punktu środka okręgu od linii (za pomocą wzoru na odległość punktu od prostej). Porównuje tę odległość z promieniem okręgu:

* Jeśli odległość < promień — linia przecina okrąg,
* Jeśli odległość ≈ promień — linia styka się z okręgiem,
* Jeśli odległość > promień — linia nie przecina okręgu.

