Praca domowa 1 – hamming

Szlachetka Miłosz
Nr albumu. 114014

1.Cel zadania

Celem zadania było zaimplementowanie programu w języku Java, który w pliku znajdowałby numer linii, której zawartość jest najbardziej zbliżona do zadanego łańcucha tekstowego. Ustalanie najbardziej podobnego zapisu odbywa się przy użyciu algorytmu Hamminga, który wyznacza ilość pozycji, na których dwa ciągi znaków różnią się od siebie.

2.Realizacja zadania

Program składa się z dwóch plików : Hamming.java oraz WebClient.java.

Hamming.java
Metody **klasy Hamming**:

- **public static int calculateDistance(String s1, String s2)** - zawiera implementację algorytmu, obliczającego odległość Hamminga między dwoma ciągami znaków. Jeśli chociaż jeden z argumentów metody ma wartość null to metoda zwraca wartość -1. Jeśli długości argumentów(ciągów znaków) są różne, metoda zwraca wartość -1. Jeśli ciągi znaków są równej długości to w pętli następuje ich porównywanie znak po znaku w taki sposób, że porównywane są znaki na takich samych pozycjach. Metoda zwraca ilość miejsc, na których wystąpiła niezgodność między łańcuchami znaków.

WebClient.java
Metody **klasy WebClient:**

- **public static void main(String[] args)** - zawiera wywołanie metody **findBestFitLineNumber** z parametrami pobranymi z konsoli(args)**.** W wyniku jej wywołania zostaje otrzymany numer linii o najlepszym dopasowaniu do zadanego łańcucha znakowego. Jeśli metoda **findBestFitLineNumber** zwraca wartość różną od -1 następuje wypisanie wyniku.
- **public static List<String> tokenizeData(String data)** - powoduje usunięcie spacji z początku i końca parametru “data”. Następnie zamienia wszystkie litery na małe, a na końcu rozdziela napis w miejscach wystąpienia znaku spacji i zwraca. Uzyskane w ten sposób fragmenty napisu są umieszczane w liście **ArrayList** i zwracane.
- **public static String fillTokenEnd(String token, int n)** - dodaje znaki “!” na **końcu** argumentu “token”. Ilość doklejonych znaków “!” jest określona argumentem “n”. Zwraca nowo utworzony napis.
- **public static String fillTokenBeginning(String token, int n)** - dodaje znaki “!” na **początku** argumentu “token”. Ilość doklejonych znaków “!” jest określona argumentem “n”. Zwraca nowo utworzony napis.

- **public static int compareTwoTokens(String s1, String s2)** - odpowiada za znajdowanie odległości Hamminga między dwoma łańcuchami znaków o dowolnej dłguości (łańcuchy mogą być różnej długości) z użyciem metody z **klasy Hamming** o nazwie **calculateDistance**. Jeśli łańcuchy są różnej długości następuje uzupełnienie krótszego z nich z użyciem metody **fillTokenEnd** i użycie metody **calculateDistance**, a następnie użycie metody **fillTokenBeginning** i użycie metody **calculateDistance** w celu wyznaczenia lepszego dopasowania. Jeśli łańcuchy są równej długości to wykonywana jest tylko metoda **calculateDistance.**
- **public static int compareSameLengthLists(List<String> list1, List<String> list2)** - wyznacza minimalną odległość Hamminga między dwoma listami łańcuchów. Listy muszą mieć tą samą ilość elementów. Elementy list porównywane są w różnych konfiguracjach (np. pierwszy element list1 z ostatnim elementem list2, drugi z list1 z pierwszym list2, ostatni element z list1 z drugim z list2). Z każdej takiej konfiguracji zbierana jest suma odległości Hamminga. Zwrócona zostaje najmniejsza z otrzymanych sum.
- **public static int compareDifferentLengthLists(List<String> list1, List<String> list2)** - wyznacza minimalną odległość Hamminga między dwoma listami łańcuchów z założeniem, że list1 ma mniejszą długość niż list2. Elementy list porównywane są w różnych konfiguracjach (np. pierwszy element list1 z ostatnim elementem list2, drugi z list1 z pierwszym list2). Dodatkowo obliczana jest długość łańcucha znaków, który w danej konfiguracji nie był porównywany. Jego długość oraz wyniki otrzymane z porównywania elementów list algorytmem Hamminga są sumowane. Takie postępowanie występuje dla każdej konfiguracji. Zwrócona zostaje wartość najmniejsza z wyznaczonych.
- **public static int findMinDistance(List<String> list1, List<String> list2)** - oblicza minimalną odległość Hamminga dla dwóch list zarówno o tej samej długości jak i o różnych długościach. Metoda sprawdza rozmiary poszczególnych list i na ich podstawie podejmuje decyzje o wywołaniu wyżej opisanych metod - **compareDifferentLengthLists, compareSameLengthLists.**
- **public static long findBestFitLineNumber(String filePath, String pattern)** - metoda tworzy listę łańcuchów znakowych **L1** z danych podanych w argumencie “pattern”. Lista ta jest tworzona z wykorzystaniem wyżej wymienionej metody **tokenizeData.** Następnie w metodzie jest otwierany plik, którego ścieżka została określona w parametrze “filePath”. Później odczytywane są z niego dane linia po linii. Z każdej linii wyodrębniane są tokeny **L2** za pomocą metody **tokenizeData.** Wywołana zostaje metoda w.w. **findMinDistance,** która zwraca najmniejszą odległość Hamminga między listami tokenów(łańcuchów znaków)**.** Jeśli otrzymana odległość jest mniejsza niż dotychczas posiadana(początkowo odległość jest wartością maksymalną dla liczb typu int) następuje zapisanie numeru linii pliku, który zawierał sprawdzaną frazę. Zwracana jest pierwsza linia, która zawiera najmniejszą odległość Hamminga między listami składającymi się z łańcuchów znaków.

3.Opis działania programu

Z linii komend pobierane są argumenty będące ścieżką do pliku oraz analizowany łańcuch tekstowy. Z analizowanego łańcucha wyodrębniana jest lista tokenów(łańcuchów znaków)

Otwierany jest plik o podanej uprzednio ścieżce dostępu. Następnie pobierane są z niego dane linia po linii. Z każdej linii wyodrębniana jest lista tokenów (łańcuchów znaków). Następnie listy te porównywane są między sobą w różnych konfiguracjach, zależnych od tego czy listy są tej samej długości czy nie. Jeśli poszczególne elementy składowe list są różnych długości wtedy krótszy element zostaje uzupełniony do długości dłuższego elementu po to by móc porównać je algorytmem Hamminga. Dopełnianie elementów realizowane jest poprzez dodanie na koniec łańcucha znaków “!”, oraz dodane tych samych znaków na początek łańcucha. W obu przypadkach liczona jest odległość Hamminga i brana jest mniejsza z uzyskanych wartości. Z porównywania poszczególnych elementów list zostają otrzymane wartości, które następnie są sumowane i dają odległość Hamminga między listami łańcuchów znaków. Jeśli aktualnie sprawdzana linia posiada niższą wartość odległości Hamminga to jej numer zostaje zachowany i sprawdzane są dalsze linie do momentu aż zostanie napotkana linia o niższej wartości odległości Hamminga. W ten sposób po sprawdzeniu wszystkich linii z pliku otrzymany zostaje numer linii o najmniejszej odległości Hamminga.