

Programowanie 3

dokumentacja projektu Tataraki i balonik

Kamil Ptak, grupa 4/8

27 stycznia 2023

Temat projektu

Projekt wykonany na podstawie zadania 3. *Tataraki i balonik* z konkursu Algorytmion 2018

W pliku słownik.txt znajduje się słownik, w którym słowa (każde w nowej linii) posortowane są alfabetycznie.

Napisz program, który dla zadanej początkowej litery słowa, poszukiwał będzie w tym słowniku wyrazów, które da się podzielić na pewnym miejscu w ten sposób, że zarówno do miejsca podziału jak i od miejsca podziału (tę część czytamy wspak), tak powstałe słowa również znajdują się w tym słowniku.

Przykładowo, jeśli podalibyśmy jako argument literę t, to program mógłby znaleźć słowo tataraki, bo dzieląc je po czwartej literze, otrzymamy słowa tata i ikar, a dla litery b, program mógłby zwrócić słowo balonik (podział po trzeciej literze na słowa bal i kino).

Zakładamy dodatkowo, że zarówno poszukiwane słowo, jak i jego składowe, są co najmniej dwuliterowe.

Program ma zwracać wszystkie wyrazy spełniające warunki zadania (na zadaną literę początkową).

Opis pobieranych danych przez program

Program pobiera od użytkownika pojedynczą literę polskiego alfabetu. W przypadku wprowadzenia kilku liter, program pobiera tylko pierwszą z nich, natomiast jeśli użytkownik poda symbol nie będący literą, program zwróci komunikat o błędnym rodzaju danych i zakończy się.

Ponadto program korzysta z pliku danych "słownik.txt" zawierający słownik języka polskiego, który na potrzeby projektu został uszczuplony w celu szybszego wykonywania się programu. Jeśli owy plik nie istnieje, program zakończy się wraz z odpowiednim komunikatem.

Opis otrzymywanych rezultatów

Zamierzonym wynikiem działania programu są słowa, znajdujące się w pliku "słownik.txt" składające się z dwóch innych słów również znajdujących się w słowniku. Pierwsze słowo zaczyna się od litery podanej przez użytkownika, drugie natomiast jest pisane wspak.

Znalezione słowa wypisane są w ekranie konsoli w następującej formie:

$$znalezioneSłowo = pierwszeSłowo \quad drugieSłowo \quad (1)$$

Zastosowany algorytm do rozwiązania zadania

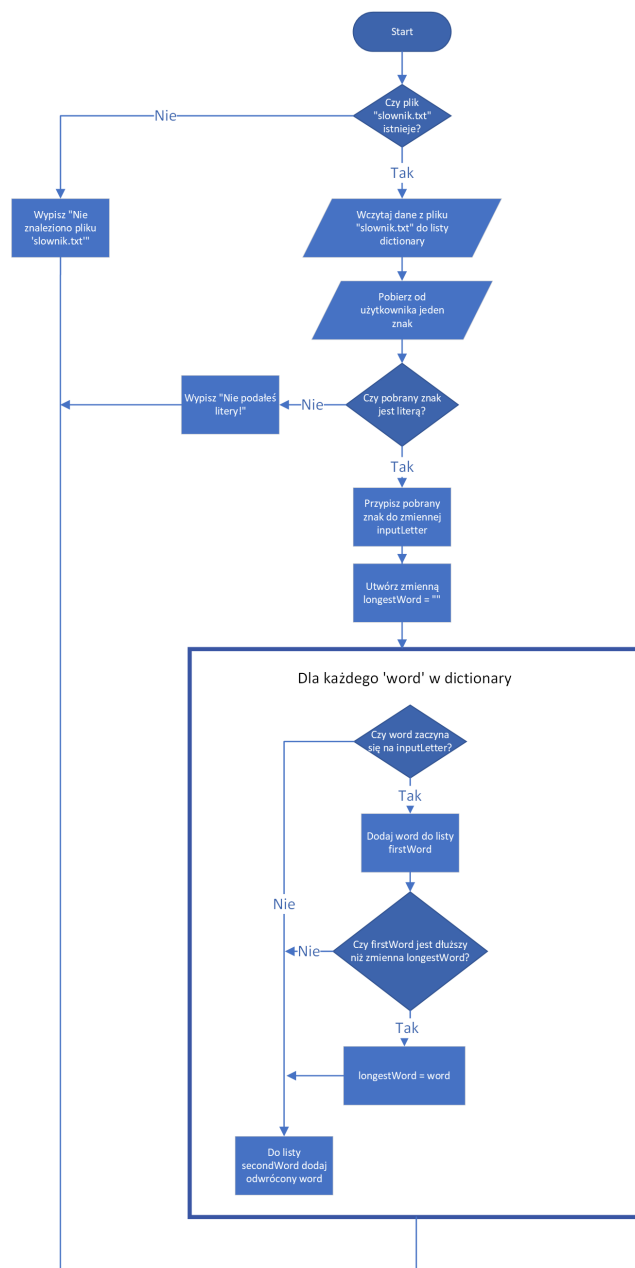
Słowny opis algorytmu

Program pobiera od użytkownika pojedynczą literę i przypisuje jej wartość do zmiennej *inputLetter*, oraz pobiera słowa z pliku 'słownik.txt' i dopisuje je do listy ciągów znaków *dictionary*. W przypadku braku pliku 'słownik.txt' albo podania przez użytkownika znaku innego niż litera, program zwraca odpowiedni komunikat i zamyka się.

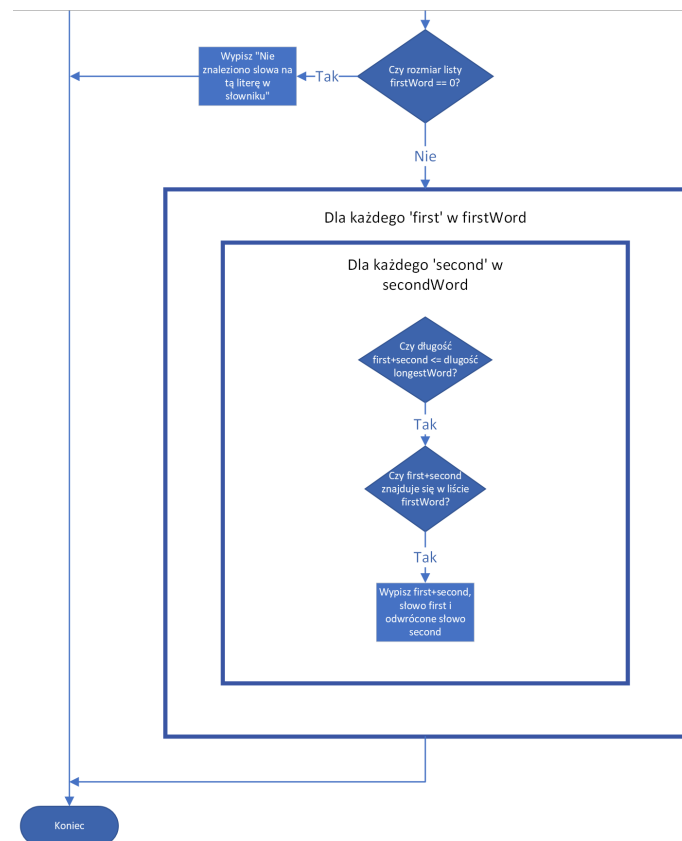
Następnie do listy ciągów znaków *firstWord* dopisujemy wszystkie słowa z *dictionary*, które zaczynają się na *inputLetter*, oraz do listy ciągów znaków *secondWord* dopisujemy wszystkie słowa z *dictionary* pisane wspak. Ponadto do zmiennej *longestWord* przypisujemy najdłuższe słowo znajdujące się w słowniku zaczynające się na *inputLetter*. W przypadku gdy lista *firstWord* jest pusta, program wyświetla odpowiedni komunikat i zamyka się.

Dla każdego słowa w *firstWord* i każdego słowa w *secondWord*, jeśli ich wspólna długość jest równa bądź krótsza niż długość *longestWord* oraz jeśli *firstWord* zawiera słowo będące sumą słów *firstWord* i *secondWord*, program wypisuje sumę tych słów, oraz *firstWord* i odwrócone *secondWord* w konsoli.

Schemat blokowy programu



Rysunek 1: Schemat blokowy programu (część 1)



Rysunek 2: Schemat blokowy programu (część 2)

Zastosowane metody i klasy w programie

Program zawiera pakiet *Handlers* zawierający następujące klasy:

Klasa *ErrorHandlers* zawierającą metodę *errorCodes*(int id), pobierającą numer id błędu, zwracającą przypisany mu komunikat i kończącą program.

Klasa *UserInput* zawierającą następujące dwie metody:

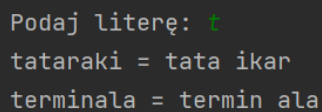
isLetter(char c) - Zwraca wartość true, bądź false w zależności czy podany znak c jest literą.

getInput() - Pobiera od użytkownika pojedynczy znak, następnie wywołuje funkcję *isLetter*(Letter), która jeśli jest prawdziwa wywołuje zewnętrzną metodę *ErrorHandler.errorCodes*(0).

Klasa *FileReader* zawierający metodę *loadDictionary*(String filename), pobierającą słowa z podanego pliku do listy ciągów znaków i zwracającą tablicę utworzoną z tej listy. W przypadku błędu odczytu pliku zostaje wywołana zewnętrzna metoda *ErrorHandler.errorCodes*(2).

Testy na poprawność działania programu

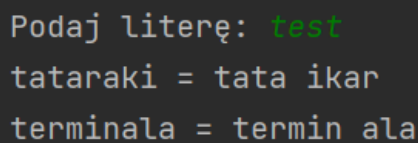
Test 1



```
Podaj literę: t
tataraki = tata ikar
terminala = termin ala
```

Rysunek 3: Test dla poprawnych danych

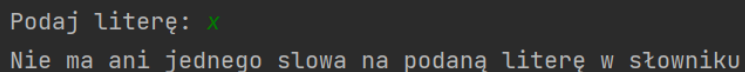
Test 2



```
Podaj literę: test
tataraki = tata ikar
terminala = termin ala
```

Rysunek 4: Test dla nadmiarowej ilości danych

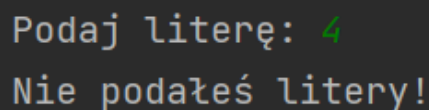
Test 3



```
Podaj literę: x
Nie ma ani jednego słowa na podaną literę w słowniku
```

Rysunek 5: Test dla danych niebędących w pliku słownik.txt

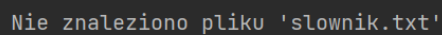
Test 4



```
Podaj literę: 4
Nie podałeś litery!
```

Rysunek 6: Test dla błędnych danych

Test 5



```
Nie znaleziono pliku 'słownik.txt'
```

Rysunek 7: Test dla braku pliku "słownik.txt"

Wnioski

Projekt działa poprawnie zgodnie z poleceniem, jednakże byłem zmuszony usunąć część rekordów z "sownik.txt", gdyż wpływały one na długość wykonywania programu. Mimo wielu prób optymalizacji programu i zastosowania różnych typów do przechowywania danych, nie udało mi się doprowadzić programu do działania na pełnej puli rekordów w optymalnym czasie.