Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

Programowanie Komputerów

Ekstrakcja słów kluczowych

autor	Bartłomiej Pogodziński
prowadzący	mgr inż. Grzegorz Kwiatkowski
rok akademicki	2019/2020
kierunek	informatyka
rodzaj studiów	SSI
semestr	2
termin laboratorium	piątek, 12:00 – 13:30
sekcja	22
termin oddania sprawozdania	2020-07-07

1 Treść zadania

Napisać własną implementację dwóch algorytmów ekstrakcji słów kluczowych dotyczących dokumentu (tekstu) w zbiorze dokumentów (tekstów).

2 Analiza zadania

Zagadnienie przedstawia problem prawidłowego przygotowania danych oraz wydajnej (czasowo oraz pamięciowo) implementacji algorytmu ekstrakcji słów kluczowych z dokumentów (tekstów) zapisanych w pliku tekstowym.

2.1 Struktury danych

Program wykorzystuje *tablicę haszującą* z rozwiązywaniem konfliktów poprzez *listę jednokierunkową* do zapisywania liczby dokumentów zawierających dany lemat (formę słownikową). Składa się ona z tablicy wskaźników na głowę listy jednokierunkowej zawierającej w swoich węzłach liczbę dokumentów z lematem oraz wskaźnik na kolejny element w liście.

W programie wykorzystywane są również *listy jednokierunkowe* zawierające wskaźniki na kolejne elementy oraz *tokeny* - słowa będące potencjalnymi słowami kluczowymi sprowadzone do postaci słownikowej zapisane jako tablica znaków.

Algorytm TextRank posiada kilka wariantów implementacyjnych. Ten zastosowany w programie jest całkiem dosłowną realizacją *grafu współwystępowania słów*. Każdy węzeł grafu składa się z ciągu znaków tworzącego opisywane słowo, aktualnego wyniku, wyniku w następnej iteracji, liczby połączeń z innymi elementami oraz wskaźnika na głowę *listy jednokierunkowej* przechowującej wskaźniki na węzły połączone z opisywanym. Wszystkie węzły znajdują się również w nadrzędnej *liście jednokierunkowej*.

Program korzysta również z różnych *struktur pomocniczych* w celu optymalizacji pamięciowej lub zwiększenia przejrzystości kodu. Przykładem takiej struktury jest token algorytmu TF-IDF zawierający tablicę znaków ze słowem oraz wynik działania algorytmu dla tego słowa. Struktury takie posiadają zazwyczaj jedną instancję i nie zawierają wskaźników na elementy o takim samym typie.

3 Algorytmy

Z uwagi na optymalizację pamięciową program unika przechowywania w pamięci podręcznej całej zawartości pliku wejściowego. Pozwala to na operowanie na znacznie większych zbiorach tekstów, ale też powoduje większe skomplikowanie działania programu.

W implementacji wykorzystano algorytmy *TF-IDF* oraz *TextRank* - bazujący na algorytmie indeksowania stron internetowych PageRank wykorzystywanym przez wyszukiwarkę Google. Obydwa z nich opierają się na wyliczaniu współczynnika "kluczowości" dla danego tokenu w zbiorze, ale metody jego wyliczania są już zgoła różne.

3.1 TF-IDF

Współczynnik TF-IDF jest iloczynem dwóch członów *Term Frequency* oznaczanym jako *TF* oraz *Inverse Document Frequency* oznaczanym jako *IDF*. TF wyliczane jest jako liczba wystąpień danego tokenu podzielona przez liczbę tokenów w analizowanym dokumencie. natomiast IDF to logarytm naturalny z liczby dokumentów w zbiorze podzielonej przez liczbę dokumentów zawierających dany token.

Algorytm ten dla każdego tokenu w zbiorze wymaga znalezienia liczby dokumentów, w których występuje. Jest to realizowane poprzez dwukrotne czytanie pliku wejściowego.

Pierwsze czytanie pozwala na zbudowanie tablicy haszującej, której kluczami są poszczególne tokeny. Pod danym tokenem-kluczem w tablicy znajduje się liczba dokumentów zawierających dany token-klucz. Zastosowanie idealnej funkcji mieszającej pozwoliłoby na stworzenie tablicy w czasie $O(n)$. W pesymistycznym przypadku - zdegenerowaniu do listy jednokierunkowej - złożoność czasowa opisuje funkcja $O(n^2)$.

Drugie przejście przez plik polega na kolejnym czytaniu dokumentów i przepisywaniu do pliku wyjściowego ze znalezionymi słowami kluczowymi. Taka forma (wraz z tablicą hashującą) pozwala na utrzymanie niewielkiej zależności czasu obsługi jednego dokumentu od liczby wszystkich dokumentów.

W algorytmie TF-IDF z listy tokenów generowana jest lista wystąpień tokenów (lista unikatowych tokenów z dokumentu wraz z liczbą wystąpień w dokumencie), na podstawie której później generowana jest tablica - już z uwzględnieniem liczby elementów i faktora IDF. Transformacja na tablicę pozwala na zastosowanie szybszych algorytmów sortujących - w tym przypadku *qsort* z biblioteki standardowej.

3.2 TextRank

Początkowo generowany jest graf współwystępowania słów. Program korzysta z ważonych połączeń między węzłami - czyli tokeny występujące wielokrotnie w swoim otoczeniu tworzą silniejsze połączenie.

Podczas tworzenia grafu dla każdego nowego tokena zostaje utworzony węzeł w grafie - reprezentowany jako element listy jednokierunkowej węzłów grafu. Jeśli token już wystąpił nowy węzeł nie jest tworzony. Dla każdego węzła tworzona jest lista wskaźników na *węzły sąsiednie*.

W węzłach sąsiednich znajdują się takie tokeny, które w wejściowej liście tokenów nie są odległe o więcej o *N* tokenów, gdzie N to *rozmiar okna sąsiedniości*. Wartością domyślną rozmiaru okna jest 2. Po zbudowaniu struktury połączeń każdy węzeł grafu inicjowany jest wartością 1.

Kolejny jest etap iteracji algorytmu. W każdym przejściu węzły "oddają" swoją wartość węzłom połączonym - proporcjonalnie do liczby współwystąpień. Liczba przejść jest stała.

Wynikowa lista jednokierunkowa węzłów z finalnymi wynikami jest ostatecznie transformowana w tablicę i sortowana względem wyników za pomocą funkcji *qsort* z biblioteki standardowej.

3.3 TextRank-IDF

Jest to połączenie dwóch poprzednich algorytmów. Dla każdego tokenu wynik *TextRank* mnożony jest przez jego współczynnik *IDF*, dzięki czemu uwzględniona jest powtarzalność niektórych tokenów w zbiorze dokumentów. Słowa kluczowe stają się więc bardziej unikatowe względem innych w zbiorze.

4 Specyfikacja zewnętrzna

Program obsługuje pliki wejściowe CSV z wymuszonymi cudzysłowami i bez linii nagłówków o zadanej formie:

```
\NAGŁÓWEK", \TREŚĆ ARTYKUŁU", \PRZYGOTOWANE TOKENY"
```

Przygotowane tokeny rozdzielone są spacjami i składają się z liter łacińskich.

Plikiem wyjściowym jest również plik CSV o kolejnych kolumnach:

```
Heading,
Content,
Tf_idf_1, ..., Tf_idf_N,
TextRank_1, ..., TextRank_N,
TextRank_idf_1, ..., TextRank_idf_N
```

Gdzie N jest liczbą ekstraktowanych słów kluczowych.

Narzędzie uruchamiane jest z linii poleceń. Obsługuje on parametry opcjonalne ustawiane za pomocą odpowiednich przełączników:

-i	nazwa pliku wejściowego	"data/prepared.csv"
-o	nazwa pliku wyjściowego	"data/sample_out.csv"
-k	liczba słów kluczowych	3
-w	rozmiar okna sąsiedniości TextRank	2

Przykładowe uruchomienie programu:

```
./main -i input.csv -o output.csv -k 2 -w 3
Counting words and documents
Rewriting with keywords
Cleaning memory
Done
```

Uruchomienie programu z nieprawidłowym wykorzystaniem przełączników uruchomieniowych powoduje zgłoszenie błędu - na przykład:

```
./main -i wrong_file.exe
ERR: Wrong input file!

Input file shall be a CSV file with forced quoting consisting of three
columns HEADING, ARTICLE, TOKENS without headers line
```

lub

```
./main -u input.csv
ERR: Invalid input parameters!

To set custom parameters use:
-i     Set input file
-o     Set output file
-k Set number of keywords to extract per algorithm
-w Set TextRank adjacency window size
```

W repozytorium znajduje się również skrypt *cleaning.py*, mający na celu przygotowanie danych wejściowych. Przygotowuje on dane poprzez szereg transformacji mających na celu oczyszczenie danych - w tym usunięcie artykułów niezawierających liter łacińskich, usunięcie słów nieznaczących oraz znaków specjalnych jak i lematyzację. Eksportuje on również dane w formacie akceptowalnym przez program, opisanym wyżej. Zostało zastosowane takie rozwiązanie z uwagi bogaty wybór publicznych narzędzi manipulacji tekstu w środowisku Python.

5 Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem strukturalnym. Funkcjonalności rozdzielone są tematycznie między plikami celem zwiększenia czytelności oraz modularności projektu.

5.1 Ogólna struktura programu

Działanie programu zostało podzielone na etapy.

Pierwszym z nich jest walidacja parametrów wejściowych. Jeśli parametry są nieprawidłowe wyświetlany jest komunikat o błędzie, a program zwalnia zaalokowaną pamięć następnie kończąc działanie. Następnym etapem jest zliczanie dokumentów zawierających poszczególne tokeny. Odbywa się to w pierwszym czytaniu i obsługiwane jest przez moduł *count_import.h*. Opiera się on na generowaniu dla każdego dokumentu listy jednokierunkowej wszystkich unikatowych tokenów w nim zawartych oraz przepisaniu tychże tokenów do tablicy haszującej wystąpień. Dzięki temu każdy unikatowy token z dokumentu jest maksymalnie raz dla dokumentu liczony w tablicy.

Następnym etapem jest przepisywanie z ekstrakcją słów kluczowych. Zarządza nim moduł *rewrite.h*. Dla każdego wiersza pliku wejściowego Nagłówek oraz treść dokumentu jest przepisywana do pliku wyjściowego. Kolumna zawierająca przygotowane tokeny zamieniana jest na listę jednokierunkową i w takiej formie przesyłana do opisanych wyżej algorytmów ekstrakcji. Te zwracają listę jednokierunkową słów kluczowych, które przepisywane są do pliku wyjściowego w odpowiedniej kolejności.

Ostatnim etapem jest zwalnianie pamięci. Implementacja zwalniania pamięci poszczególnych struktur jest zawarta w odpowiadających im modułach.

5.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączniku.

6 Testowanie

Program był testowany na pliku wejściowym przygotowanym przez skrypt cleaning.py zawarty wśród plików źródłowych. Danymi wejściowymi był zbiór 337 artykułów z portalu Medium. Dla części funkcji zostały również napisane testy jednostkowe zawarte w pliku test.c. Program został sprawdzony pod kątem wycieków pamięci.

7 Wnioski

Projekt ten był moim pierwszym o tak złożonej strukturze wewnętrznej. Wymagał dokładnego przemyślenia konstrukcji kodu jeszcze przed samą implementacją. Dodatkowo pozwoliło to na oswojenie się z zarządzaniem pamięcią oraz kontrolą jej wycieków. Sama część przygotowania danych była również całkiem ciekawa, ponieważ zbiór, który wybrałem do testów zawierał wiele problemów - jak na przykład fragmenty kodu HTML, linki czy artykuły zapisane w alfabetach opartych na innym niż łaciński. Pozwoliło to na zapoznanie się z narzędziami do “czyszczenia tekstu” w środowisku Python.

Literatura

[1] Rada Mihalcea and Paul Tarau. TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 404–411, Barcelona, Spain, July 2004. Association for Computational Linguistics.

Dodatek
Szczegółowy opis typów i funkcji

Projekt zaliczeniowy z PPK-SSI

Wygenerowano przez Doxygen 1.8.13

Spis treści

Rozdział 1

Indeks klas

1.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich udokumentowanych plików z ich krótkimi opisami:

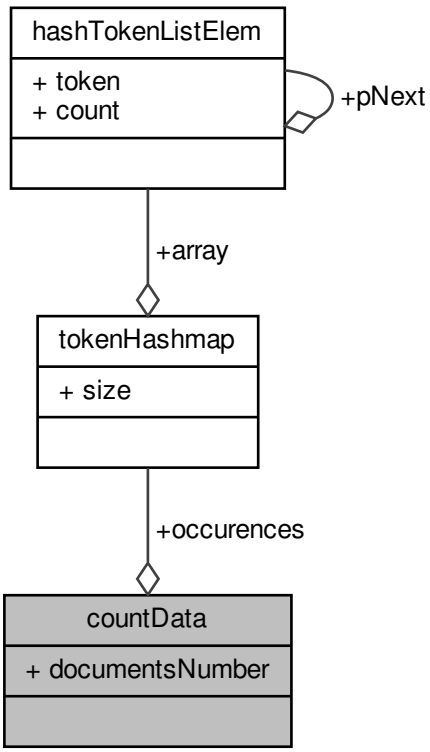
Rozdział 3

Dokumentacja klas

3.1 Dokumentacja struktury countData

```
#include <count_import.h>
```

Diagram współpracy dla countData:



Atrybuty publiczne

- int documentsNumber

  Number of documents in a file.
- struct tokenHashmap * occurences

  Hashmap containing the number of documents with particular token.

3.1.1 Opis szczegółowy

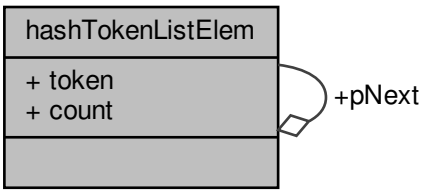
Structure returned after first reading of the document Describes presence of each word in documents

Dokumentacja dla tej struktury została wygenerowana z pliku:

- kod/count_import.h

3.2 Dokumentacja struktury hashTokenListElem

Diagram współpracy dla hashTokenListElem:



Atrybuty publiczne

- char * token

  String containing the token.
- int count

  In how many documents has occured.
- struct hashTokenListElem * pNext

  Pointer to the next element.

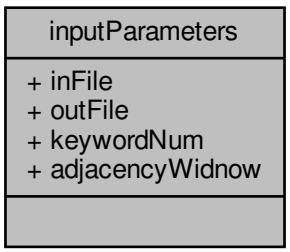
Dokumentacja dla tej struktury została wygenerowana z pliku:

- kod/hashmap.h

3.3 Dokumentacja struktury inputParameters

`#include <handle_input.h>`

Diagram współpracy dla inputParameters:



Atrybuty publiczne

- char * inFile

  Input file name.
- char * outFile

  Output file name.
- int keywordNum

  Number of keywords to extract.
- int adjacencyWidnow

  Textrank tokens adjacency window.

3.3.1 Opis szczegółowy

Program input parameters

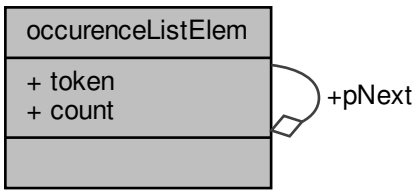
Dokumentacja dla tej struktury została wygenerowana z pliku:

- kod/handle_input.h

3.4 Dokumentacja struktury occurenceListElem

`#include <token.h>`

Diagram współpracy dla occurenceListElem:



Atrybuty publiczne

- char * token

  String containing the token.
- int count

  Occurence counter.
- struct occurenceListElem * pNext

  Pointer to the next element.

3.4.1 Opis szczegółowy

List of the tokens that were present at least once in the document

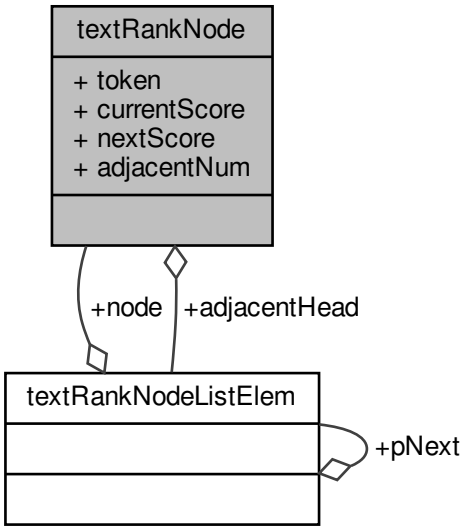
Dokumentacja dla tej struktury została wygenerowana z pliku:

- kod/token.h

3.5 Dokumentacja struktury textRankNode

`#include <textrank_graph.h>`

Diagram współpracy dla textRankNode:



Atrybuty publiczne

- char * token

  Word stored in a node.
- double currentScore

  Current score of the node.
- double nextScore

  Score at the end of iteration.
- int adjacentNum

  Number of connected nodes.
- struct textRankNodeListElem * adjacentHead

  Head of the connected nodes list.

3.5.1 Opis szczegółowy

Stores one node data

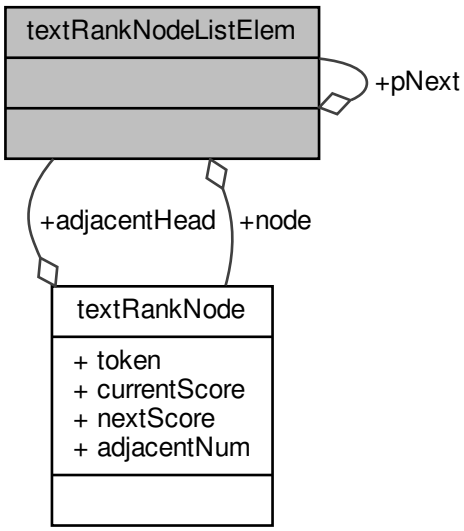
Dokumentacja dla tej struktury została wygenerowana z pliku:

- kod/textrank_graph.h

3.6 Dokumentacja struktury textRankNodeListElem

`#include <textrank_graph.h>`

Diagram współpracy dla textRankNodeListElem:



Atrybuty publiczne

- struct textRankNode ∗ node

  Node stored in the list.
- struct textRankNodeListElem ∗ pNext

  Pointer to the next node.

3.6.1 Opis szczegółowy

element of nodes list

Dokumentacja dla tej struktury została wygenerowana z pliku:

- kod/textrank_graph.h

3.7 Dokumentacja struktury tfIdfToken

```
#include <tf_idf.h>
```

Diagram współpracy dla tfIdfToken:

Atrybuty publiczne

- char * token

  Potential keyword.
- double score

  Tf-Idf score.

3.7.1 Opis szczegółowy

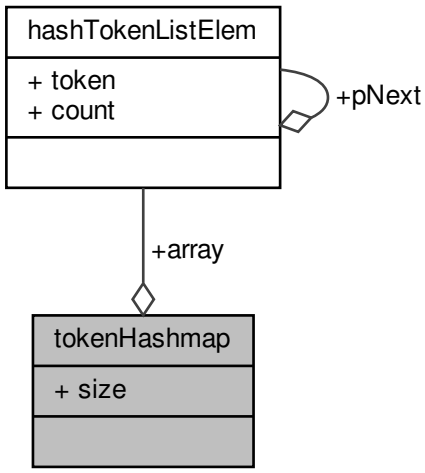
Structure contains token with its Tf-Idf score

Dokumentacja dla tej struktury została wygenerowana z pliku:

- kod/tf_idf.h

3.8 Dokumentacja struktury tokenHashmap

Diagram współpracy dla tokenHashmap:



Atrybuty publiczne

- struct hashTokenListElem ∗ array [HASHMAP_SIZE]

  Array of hashmap elements.
- int size

  Size of the hashmap array.

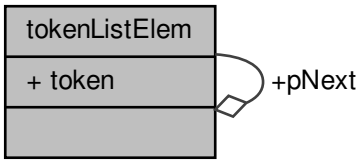
Dokumentacja dla tej struktury została wygenerowana z pliku:

- kod/hashmap.h

3.9 Dokumentacja struktury tokenListElem

`#include <token.h>`

Diagram współpracy dla tokenListElem:



Atrybuty publiczne

- char * token

  Word token.
- struct tokenListElem * pNext

  Pointer to the next element.

3.9.1 Opis szczegółowy

Element of the text token linked list

Dokumentacja dla tej struktury została wygenerowana z pliku:

- kod/token.h

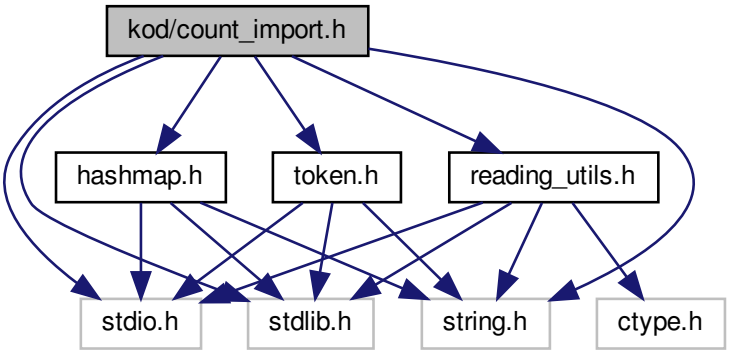
Rozdział 4

Dokumentacja plików

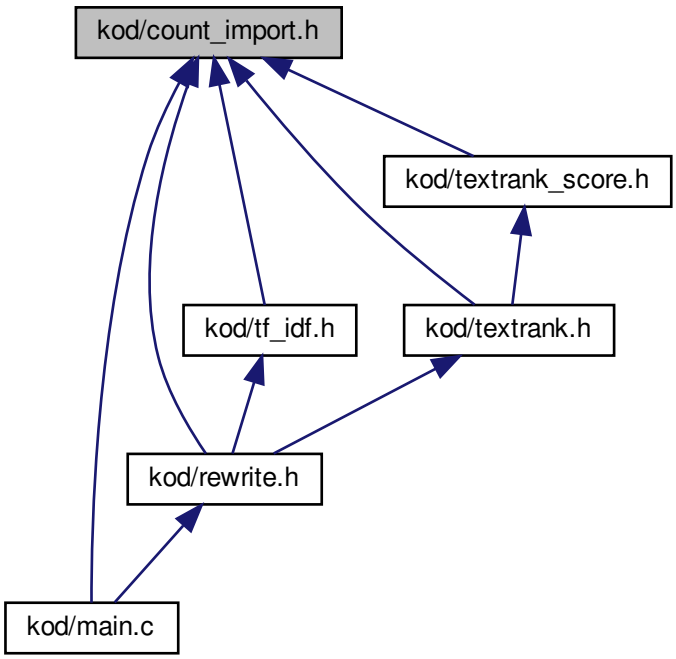
4.1 Dokumentacja pliku kod/count_import.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "hashmap.h"
#include "token.h"
#include "reading_utils.h"
```

Wykres zależności załączania dla count_import.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- struct countData

Funkcje

- struct countData ∗ getCountData (FILE ∗file)

  Get the countData object pointer, Does not change the file pointer.
- void freeCountData (struct countData ∗∗data)

  Frees the memory taken by countData.

4.1.1 Dokumentacja funkcji

4.1.1.1 freeCountData()

```
void freeCountData (
            struct countData ** data )
```

Frees the memory taken by countData.

Parametry

data	pointer to the data pointer

4.1.1.2 getCountData()

```
struct countData* getCountData (
            FILE * file )
```

Get the countData object pointer, Does not change the file pointer.

Parametry

file	pointer to the file

Zwraca

struct countData* pointer to newly created object containing basic info about document

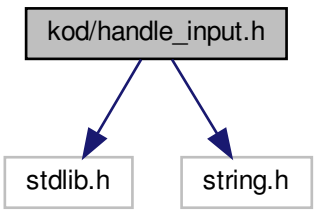
Reading the tokens

Resaving the tokens to the hashmap

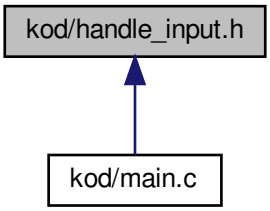
4.2 Dokumentacja pliku kod/handle_input.h

```
#include <stdlib.h>
#include <string.h>
```

Wykres zależności załączania dla handle_input.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- struct inputParameters

Definicje

- #define **DEFAULT_IN_FILE** "../data/prepared.csv"
- #define **DEFAULT_OUT_FILE** "../data/sample_out.csv"
- #define **DEFAULT_KEYWORD_NUM** 3
- #define **DEFAULT_ADJACENCY_WINDOW** 2

Funkcje

- struct inputParameters ∗ handleInput (int argc, char ∗argv[ ])

  Handle input parameters.
- void freeInputParameters (struct inputParameters ∗params)

  Frees memory taken by the input parameters instance.

4.2.1 Dokumentacja funkcji

4.2.1.1 freeInputParameters()

```
void freeInputParameters (
            struct inputParameters * params )
```

Frees memory taken by the input parameters instance.

Parametry

params	pointer to the parameters structure

4.2.1.2 handleInput()

```
struct inputParameters* handleInput (
            int argc,
            char * argv[] )
```

Handle input parameters.

Parametry

argc	Number of input arguments
argv	Input arguments array

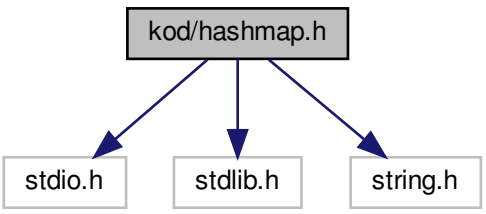
Zwraca

struct inputParameters* Pointer to input parameters

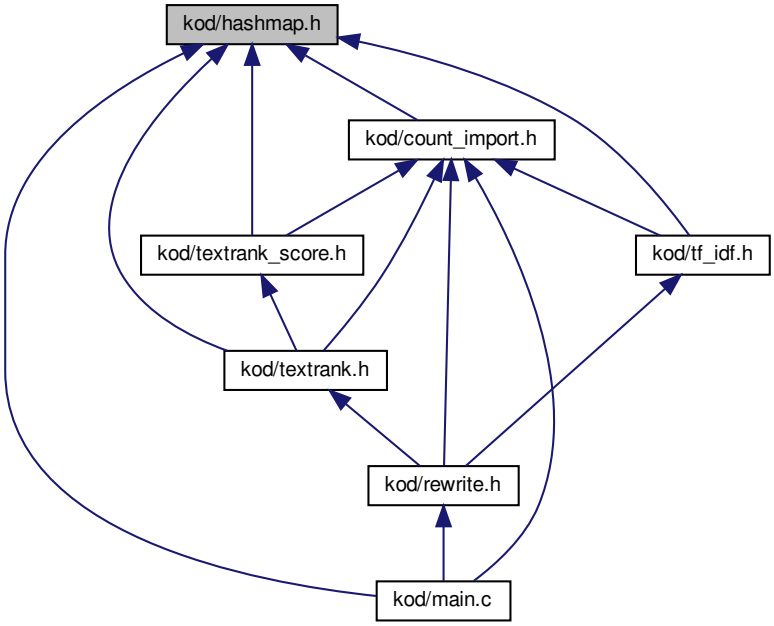
4.3 Dokumentacja pliku kod/hashmap.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Wykres zależności załączania dla hashmap.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- struct hashTokenListElem
- struct tokenHashmap

Definicje

- #define **HASHMAP_SIZE** 10000

Funkcje

- struct tokenHashmap ∗ initTokenHashmap ()

  Initializes a hashmap of tokens.
- int hash (char ∗token)

  Gets the index in hashmap of the element.
- void addOccurenceToMap (struct tokenHashmap ∗hashmap, char ∗token)

  Adds token occurence to the hashmap.
- int getOccurenceFromMap (struct tokenHashmap ∗hashmap, char ∗token)

  Get the occurence count of particular token from hashmap.
- void freeHashmap (struct tokenHashmap ∗hashmap)

  Frees the memory taken by the hashmap.
- void freeHashTokenList (struct hashTokenListElem ∗∗pHead)

  Frees the memory taken by the list and nullifies pointer.

4.3.1 Dokumentacja funkcji

4.3.1.1 addOccurenceToMap()

```
void addOccurenceToMap (
            struct tokenHashmap * hashmap,
            char * token )
```

Adds token occurence to the hashmap.

Parametry

hashmap	Hashmap of the tokens
token	Token whose occurence will be incremented

4.3.1.2 freeHashmap()

```
void freeHashmap (
            struct tokenHashmap * hashmap )
```

Frees the memory taken by the hashmap.

Parametry

hashmap	pointer to the hashmap instance

4.3.1.3 freeHashTokenList()

```
void freeHashTokenList (
            struct hashTokenListElem ** pHead )
```

Frees the memory taken by the list and nullifies pointer.

Parametry

pHead	

4.3.1.4 getOccurenceFromMap()

```
int getOccurenceFromMap (
            struct tokenHashmap * hashmap,
            char * token )
```

Get the occurence count of particular token from hashmap.

Parametry

hashmap	Hashmap of the tokens
token	Token whose occurence will be returned

Zwraca

int Occurence count of the token

4.3.1.5 hash()

```
int hash (
            char * token )
```

Gets the index in hashmap of the element.

Parametry

token	Element of which the index will be found

Zwraca

int Index in hashmap

4.3.1.6 initTokenHashmap()

```
struct tokenHashmap* initTokenHashmap ( )
```

Initializes a hashmap of tokens.

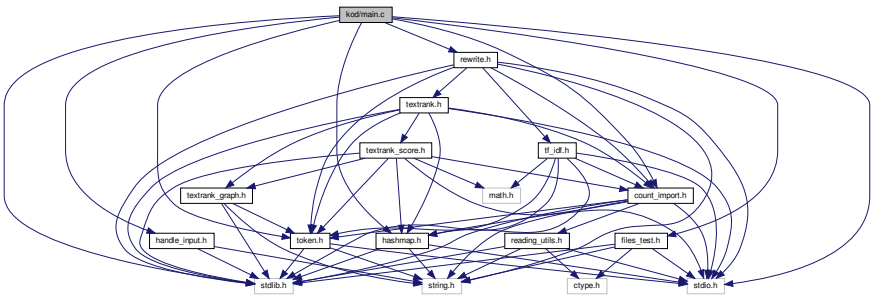
Zwraca

struct tokenHashmap* pointer to a fresh token hashmap instance

4.4 Dokumentacja pliku kod/main.c

```
#include <stdlib.h>
#include <stdio.h>
#include "handle_input.h"
#include "hashmap.h"
#include "token.h"
#include "count_import.h"
#include "rewrite.h"
#include "files_test.h"
```

Wykres zależności załączania dla main.c:



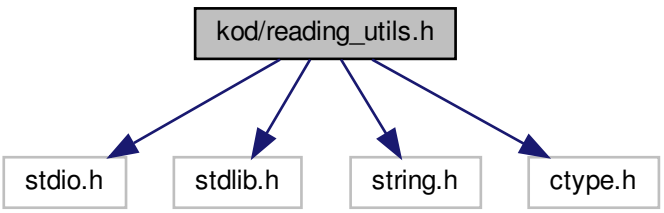
Funkcje

- int **main** (int argc, char ∗argv[ ])

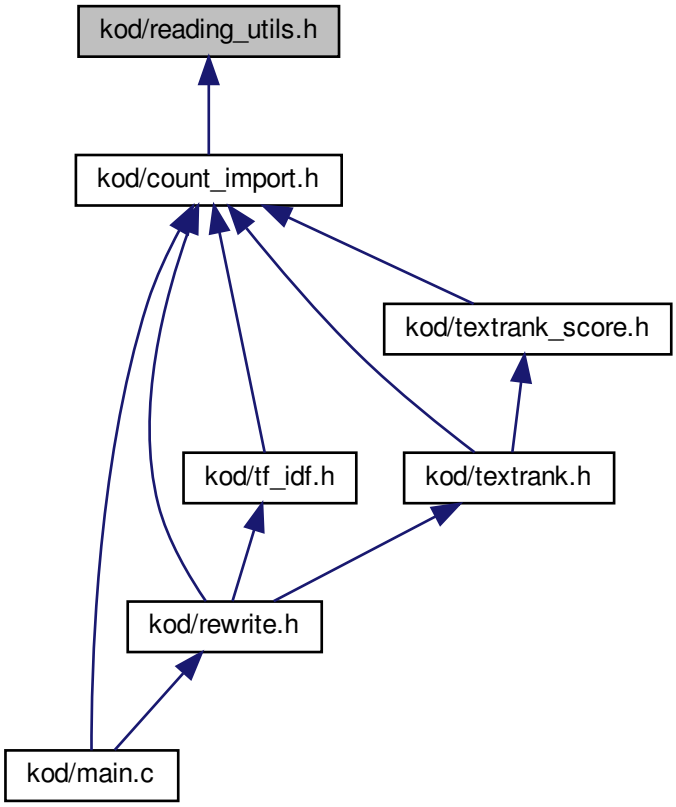
4.5 Dokumentacja pliku kod/reading_utils.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
```

Wykres zależności załączania dla reading_utils.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Funkcje

- char ∗ readOneWord (FILE ∗file)
  Reads one word from file.

4.5.1 Dokumentacja funkcji

4.5.1.1 readOneWord()

```
char* readOneWord (
            FILE * file )
```

Reads one word from file.

Parametry

file	pointer to the file

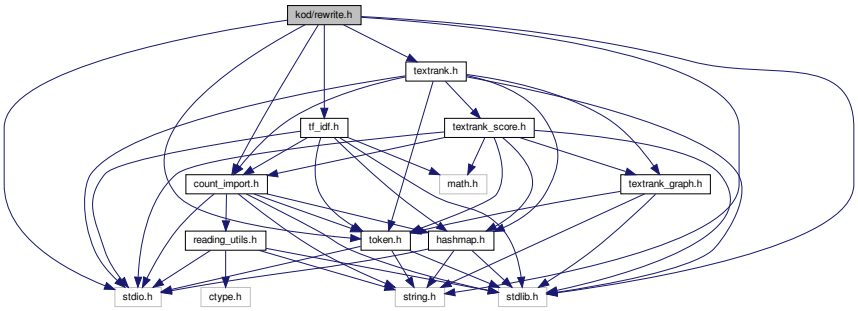
Zwraca

char∗∗ pointer to the first letter of the string

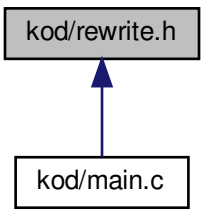
4.6 Dokumentacja pliku kod/rewrite.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "count_import.h"
#include "token.h"
#include "tf_idf.h"
#include "textrank.h"
```

Wykres zależności załączania dla rewrite.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Funkcje

- int rewriteWithKeywords (FILE ∗inFile, FILE ∗outFile, struct countData ∗cntData, int keywordNum, int adjacencyWindow)

  Writes the HEADING, ARTICLE and set of found keywords to the outputFile.
- int rewriteChar (FILE ∗inFile, FILE ∗outFile)

  reqrites one character and returns the rewritten character
- int rewriteField (FILE ∗inFile, FILE ∗outFile)

  Rewrtes one field to output file. One symbol after quotation marh will be rewritten as well.
- struct tokenListElem ∗ tokenizeField (FILE ∗inFile)

  Creates a token list based on a TOKENS field.
- int writeList (FILE ∗outFile, struct tokenListElem ∗tokenHead)

  Writes given token list to a output stream separated by comma.
- int writeHeaders (FILE ∗outFile, int keywordsNum)

  Writes file headers to the output file.

4.6.1 Dokumentacja funkcji

4.6.1.1 rewriteChar()

```
int rewriteChar (
            FILE * inFile,
            FILE * outFile )
```

reqrites one character and returns the rewritten character

Parametry

inFile	Pointer to the input file stream
outFile	Pointer to the output file stream

Zwraca

int rewritten character

4.6.1.2 rewriteField()

```
int rewriteField (
            FILE * inFile,
            FILE * outFile )
```

Rewrtes one field to output file. One symbol after quotation marh will be rewritten as well.

For example it may rewrite: "lorem ipsum dolor", both with quotes and comma

Parametry

inFile	Pointer to the input file stream
outfile	Pointer to the output file stream

Zwraca

int 0 if no errors, 1 if found EOF, 2 if found Error

4.6.1.3 rewriteWithKeywords()

```
int rewriteWithKeywords (
            FILE * inFile,
            FILE * outFile,
            struct countData * cntData,
            int keywordNum,
            int adjacencyWindow )
```

Writes the HEADING, ARTICLE and set of found keywords to the outputFile.

Parametry

inFile	Pointer to the input file stream set on start position
outFile	Pointer to the output file stream set on start position
cntData	Count data necessary for IDF
keywordNum	Number of keywords to find
adjacencyWindow	TextRank adjacencyWindowSize

Zwraca

int returns 0 if there were no errors

4.6.1.4 tokenizeField()

```
struct tokenListElem* tokenizeField (
            FILE * inFile )
```

Creates a token list based on a TOKENS field.

Parametry

inFile	

Zwraca

struct tokenListElem*

4.6.1.5 writeHeaders()

```
int writeHeaders (
            FILE * outFile,
            int keywordsNum )
```

Writes file headers to the output file.

Parametry

outFile	Output file stream
keywordsNum	Number of keywords per method

Zwraca

int 0 if no errors

4.6.1.6 writeList()

```
int writeList (
            FILE * outFile,
            struct tokenListElem * tokenHead )
```

Writes given token list to a output stream separated by comma.

Parametry

outFile	Pointer to the output file stream
tokenHead	Pointer to the list head

Zwraca

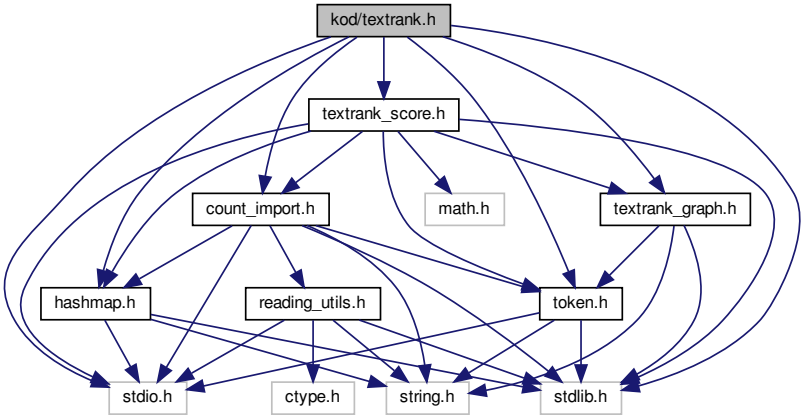
int 0 if no errors

4.7 Dokumentacja pliku kod/textrank.h

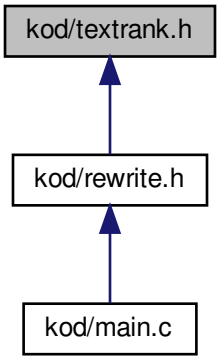
```
#include <stdio.h>
#include <stdlib.h>
#include "token.h"
```

```
#include "hashmap.h"
#include "count_import.h"
#include "textrank_graph.h"
#include "textrank_score.h"
```

Wykres zależności załączania dla textrank.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Funkcje

- struct tokenListElem ∗ getTextRankfKeywords (int expectedKeywords, struct tokenListElem ∗tokenHead, struct countData ∗cntData, int idf, int adjacencyWindow, int iterations)

  Get the Text Rankf Keywords list.

- int textRankSortingComparator (const void ∗a, const void ∗b)

  Compares two TextRank nodes by their score.

4.7.1 Dokumentacja funkcji

4.7.1.1 getTextRankfKeywords()

```
struct tokenListElem* getTextRankfKeywords (
            int expectedKeywords,
            struct tokenListElem * tokenHead,
            struct countData * cntData,
            int idf,
            int adjacencyWindow,
            int iterations )
```

Get the Text Rankf Keywords list.

Parametry

expectedKeywords	number of keywords to extract
tokenHead	list of tokens - tokenized article
cntData	occurence data of all words
idf	1 if scores shall be multiplied by the idf factor, 0 otherwise
adjacencyWindow	How far can a word be from the other to be concidered "close"
iterations	Number of algotithm iterations

Zwraca

struct tokenListElem* list of keywords

4.7.1.2 textRankSortingComparator()

```
int textRankSortingComparator (
            const void * a,
            const void * b )
```

Compares two TextRank nodes by their score.

Parametry

a	first TextRank node
b	second TextRank node

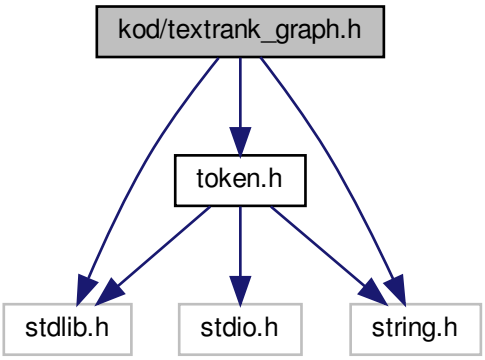
Zwraca

int 1 if first is lower, -1 if bigger and 0 if same score

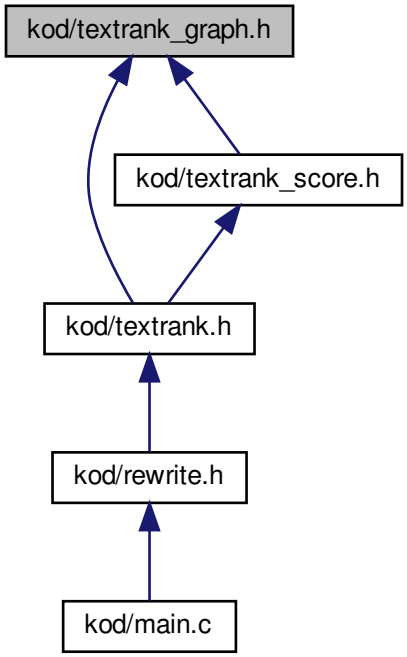
4.8 Dokumentacja pliku kod/textrank_graph.h

```
#include <stdlib.h>
#include <string.h>
#include "token.h"
```

Wykres zależności załączania dla textrank_graph.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- struct textRankNodeListElem
- struct textRankNode

Funkcje

- struct textRankNodeListElem * generateTextRankGraph (struct tokenListElem *tokenHead, int adjacencyWindow)

  Generates a TextRank graph based on token list.
- void connectTextRankNodes (struct textRankNode *node1, struct textRankNode *node2)

  Connects two nodes - adds each other to lists of adjacent.
- struct textRankNode * getOrCreateTextRankNode (struct textRankNodeListElem **graphHead, char *token)

  Create an empty Text Rank Node.
- int getTextRankNodeCount (struct textRankNodeListElem *graph)

  Get the Text Rank Node Count.
- struct textRankNode ** textRankGraphToArray (struct textRankNodeListElem *graph, int nodeCount)

  Transforms graph to array of node pointers.
- void freeTextRankNodesArray (struct textRankNode **nodeArray, int length)

  Free memory taken by the node arrray.

4.8.1 Dokumentacja funkcji

4.8.1.1 connectTextRankNodes()

```
void connectTextRankNodes (
            struct textRankNode * node1,
            struct textRankNode * node2 )
```

Connects two nodes - adds each other to lists of adjacent.

Parametry

node1	pointer to first node to connect
node2	pointer to the second node to connect

4.8.1.2 freeTextRankNodesArray()

```
void freeTextRankNodesArray (
            struct textRankNode ** nodeArray,
            int length )
```

Free memory taken by the node arrray.

Parametry

nodeArray	Array of nodes
length	length of array

4.8.1.3 generateTextRankGraph()

```
struct textRankNodeListElem* generateTextRankGraph (
            struct tokenListElem * tokenHead,
            int adjacencyWindow )
```

Generates a TextRank graph based on token list.

Parametry

tokenHead	list of tokens
adjacencyWindow	number of tokens on one side concidered as neighbouring

Zwraca

struct textRankNodeListElem* pointer to the head of graph list

4.8.1.4 getOrCreateTextRankNode()

```
struct textRankNode* getOrCreateTextRankNode (
            struct textRankNodeListElem ** graphHead,
            char * token )
```

Create an empty Text Rank Node.

Parametry

graphHead	pointer to the graph head pointer
token	word stored in the new node

Zwraca

struct textRankNode* newly created text rank node

4.8.1.5 getTextRankNodeCount()

```
int getTextRankNodeCount (
            struct textRankNodeListElem * graph )
```

Get the Text Rank Node Count.

Parametry

graph	Pointer to the graph head

Zwraca

int

4.8.1.6 textRankGraphToArray()

```
struct textRankNode** textRankGraphToArray (
            struct textRankNodeListElem * graph,
            int nodeCount )
```

Transforms graph to array of node pointers.

Parametry

graph	Pointer to the graph head
nodeCount	Number of nodes in the graph

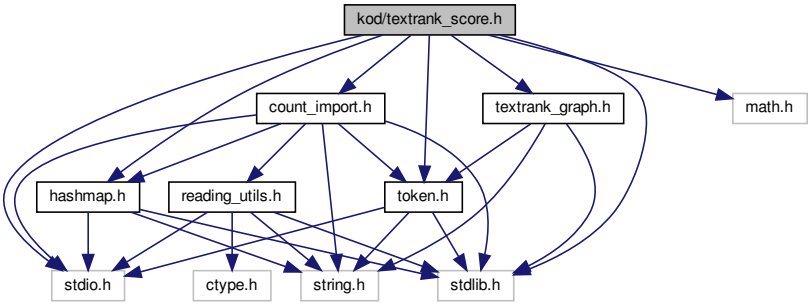
Zwraca

struct extRankNode∗ array Array of graph node pointers

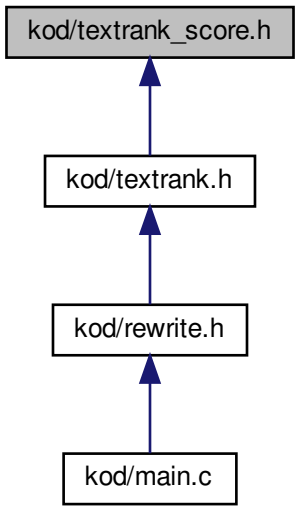
4.9 Dokumentacja pliku kod/textrank_score.h

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "token.h"
#include "hashmap.h"
#include "count_import.h"
#include "textrank_graph.h"
```

Wykres zależności załączania dla textrank_score.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Funkcje

- void processTextRankIterations (struct textRankNodeListElem ∗graph, int iterations)

  Process TextRank scores.
- void donateTextRankScore (struct textRankNode ∗giverNode)

  Gives score of one node to all set as adjacent proportionally to the "weight" of connection.
- void updateTextRankScores (struct textRankNodeListElem ∗graph)

  Iterates over all nodes and updates its scores (nextScore becomes currentScore)
- void multiplyTextRankScoresByIdf (struct textRankNodeListElem ∗graph, struct countData ∗cntData)

  Multiplies each node score by inverse document frequency of a node.

4.9.1 Dokumentacja funkcji

4.9.1.1 donateTextRankScore()

```
void donateTextRankScore (
            struct textRankNode * giverNode )
```

Gives score of one node to all set as adjacent proportionally to the "weight" of connection.

Parametry

giverNode	Pointer to the node that gives its score

4.9.1.2 multiplyTextRankScoresByIdf()

```
void multiplyTextRankScoresByIdf (
            struct textRankNodeListElem * graph,
            struct countData * cntData )
```

Multiplies each node score by inverse document frequency of a node.

Parametry

graph	pointer to graph
cntData	pointer to count data

4.9.1.3 processTextRankIterations()

```
void processTextRankIterations (
            struct textRankNodeListElem * graph,
            int iterations )
```

Process TextRank scores.

Parametry

graph	Pointer to the graph
iterations	Number of algorithm iterations

4.9.1.4 updateTextRankScores()

```
void updateTextRankScores (
            struct textRankNodeListElem * graph )
```

Iterates over all nodes and updates its scores (nextScore becomes currentScore)

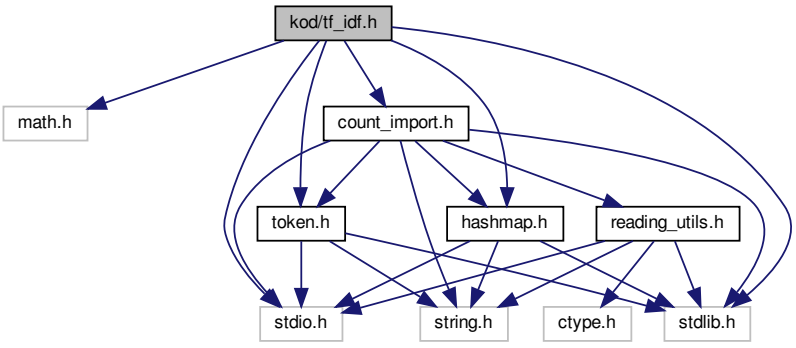
Parametry

graph	Pointer to the graph

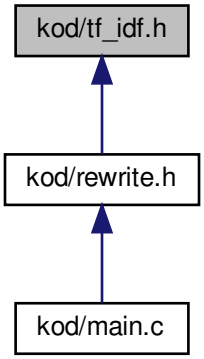
4.10 Dokumentacja pliku kod/tf_idf.h

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "token.h"
#include "hashmap.h"
#include "count_import.h"
```

Wykres zależności załączania dla tf_idf.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- struct tfIdfToken

Funkcje

- struct tokenListElem * getTfIdfKeywords (int expectedKeywords, struct tokenListElem *tokenHead, struct countData *cntData)

  Gets the Tf-Idf keywords list.
- struct occurenceListElem * getOccurencesFromTokens (struct tokenListElem *tokenHead)

  Generates occurence list from token list, makes copy of string.
- double getOneTfIdfScore (char *token, int inDocOccurences, int docLength, struct countData *cntData)

  Calculate tf-idf score for one token.
- struct tfIdfToken * getTfIdfScoresArray (struct occurenceListElem *occurenceHead, int occurenceLength, struct countData *cntData)

  Calculates Tf-Idf scores and saves them in an array with length of occurence list.
- void freeTfIdfScoresArray (struct tfIdfToken **scoresArr, int length)

  Frees the memory taken by the scores array.
- int tfIdfSortingComparator (const void *a, const void *b)

  Compares two tf-idf tokens by their score.

4.10.1 Dokumentacja funkcji

4.10.1.1 freeTfIdfScoresArray()

```
void freeTfIdfScoresArray (
            struct tfIdfToken ** scoresArr,
            int length )
```

Frees the memory taken by the scores array.

Parametry

scoresArr	pointer to the first element pointer
length	length of the array

4.10.1.2 getOccurencesFromTokens()

```
struct occurenceListElem* getOccurencesFromTokens (
            struct tokenListElem * tokenHead )
```

Generates occurence list from token list, makes copy of string.

Parametry

tokenHead	pointer to head of token list

Zwraca

struct occurenceListElem* pointer to new occurence list

4.10.1.3 getOneTfIdfScore()

```
double getOneTfIdfScore (
            char * token,
            int inDocOccurences,
            int docLength,
            struct countData * cntData )
```

Calculate tf-idf score for one token.

Parametry

token	char string containing the token
inDocOccurences	How many times has the word occured in a document
docLength	number of tokens in a document
cntData	pointer to structure with number of documents and token popularity hashmap

Zwraca

double

4.10.1.4 getTfIdfKeywords()

```
struct tokenListElem* getTfIdfKeywords (
            int expectedKeywords,
            struct tokenListElem * tokenHead,
            struct countData * cntData )
```

Gets the Tf-Idf keywords list.

Parametry

expectedKeywords	Number of keywords to extract
tokenHead	list of tokens - tokenized articl0e
cntData	occurence data of all words

Zwraca

struct tokenListElem∗ head of keywords list

4.10.1.5 getTfIdfScoresArray()

```
struct tfIdfToken* getTfIdfScoresArray (
            struct occurenceListElem * occurenceHead,
            int occurenceLength,
            struct countData * cntData )
```

Calculates Tf-Idf scores and saves them in an array with length of occurence list.

Parametry

occurenceHead	Pointer to the head of occurence list
occurenceLength	Length of occurence list
cntData	occurence data of all words

Zwraca

struct tfIdfToken∗ array of tfidf scores with proper tokens

4.10.1.6 tfIdfSortingComparator()

```
int tfIdfSortingComparator (
            const void * a,
            const void * b )
```

Compares two tf-idf tokens by their score.

Parametry

a	first tf-idf token
b	second tf-idf token

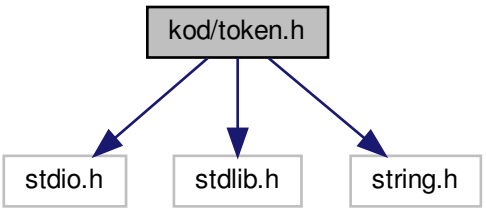
Zwraca

int 1 if first is lower, -1 if bigger and 0 if same score

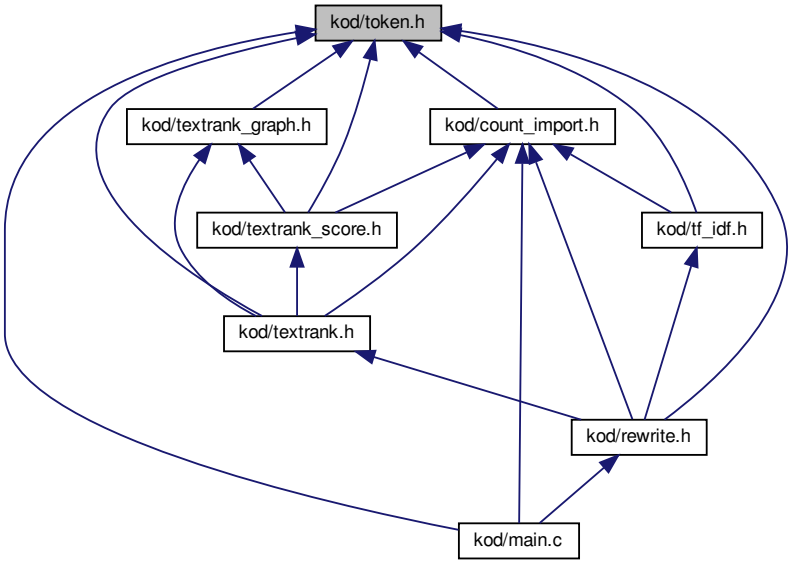
4.11 Dokumentacja pliku kod/token.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Wykres zależności załączania dla token.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- struct tokenListElem
- struct occurenceListElem

Funkcje

- void addToTokenList (struct tokenListElem **pHead, char *token)

  Adds token to token list on head position.
- void freeTokenList (struct tokenListElem **pHead)

  Frees the memory taken by the list and nullifies pointer.
- void addToOccurenceList (struct occurenceListElem **pHead, char *token)

  Increments the word occurence counter or creates new one.
- int getOccurenceListLength (struct occurenceListElem *pHead)

  Gets the Occurence List Length.
- void freeOccurenceList (struct occurenceListElem **pHead)

  Frees the memory taken by the list and nullifies pointer.

4.11.1 Dokumentacja funkcji

4.11.1.1 addToOccurenceList()

```
void addToOccurenceList (
            struct occurenceListElem ** pHead,
            char * token )
```

Increments the word occurence counter or creates new one.

Parametry

pHead	Pointer to the list head pointer
token	Text token

4.11.1.2 addToTokenList()

```
void addToTokenList (
            struct tokenListElem ** pHead,
            char * token )
```

Adds token to token list on head position.

Parametry

pHead	Pointer to the list head pointer
token	Text token

4.11.1.3 freeOccurenceList()

```
void freeOccurenceList (
            struct occurenceListElem ** pHead )
```

Frees the memory taken by the list and nullifies pointer.

Parametry

pHead	Pointer to the list head pointer

4.11.1.4 freeTokenList()

```
void freeTokenList (
            struct tokenListElem ** pHead )
```

Frees the memory taken by the list and nullifies pointer.

Parametry

pHead	Pointer to the list head pointer

4.11.1.5 getOccurenceListLength()

```
int getOccurenceListLength (
            struct occurenceListElem * pHead )
```

Gets the Occurence List Length.

Parametry

pHead	head of the occurence list

Zwraca

int length of the list

Skorowidz