Calculate sentence score

1.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Chapter 2

File Documentation

2.1 src/functions.cpp File Reference

```
#include "functions.hpp"
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <time.h>
#include <algorithm>
#include <functional>
#include <map>
#include <set>
#include <random>
#include <string.h>
#include <chrono>
#include <boost/log/core.hpp>
#include <boost/log/trivial.hpp>
#include <boost/log/expressions.hpp>
#include <boost/log/utility/setup/file.hpp>
#include <boost/log/utility/setup/common_attributes.hpp>
```

Include dependency graph for functions.cpp:

2.2 src/functions.hpp File Reference

```
#include <vector>
#include <string>
#include <bitset>
#include <map>
```

Include dependency graph for functions.hpp: This graph shows which files directly or indirectly include this file:

Functions

- void init_logging ()

  This function helps in logging level.
- void init_random_matrices (std::vector< std::bitset< 100 >> &matrices)

  initializing random matrices

- void read_dict_from_file (const std::string &dictionary_filename, std::vector< std::string > &dictionary)

    reading dictionary from file
- void read_text_from_file (const std::string &text_filename, std::vector< std::string > &text)

    reading input text, stop words from file
- void init_hashes (std::vector< size_t > &hashes, const std::hash< std::string > &hasher, const std::vector< std::string > &words)

    calculating hashes of words stored in words
- void correct_dict_hashes_and_extract_duplicates (std::vector< std::vector< size_t >> &duplicates, std↩ ::vector< size_t > &hashes)

    finding duplicates and correcting dictionary hashes
- void correct_text_hashes (std::vector< size_t > &text_hashes, const std::vector< std::string > &text, const std::vector< std::vector< size_t >> &duplicates, const std::vector< std::string > &dictionary)

    correcting input text hashes, maybe there are duplicates
- void hash_function (std::vector< std::pair< size_t, size_t >> &hash_table, const std::vector< size_t > &hashes)

    hashing open adressing with linear probbing algorithm
- void search_and_calculate_matrices (std::bitset< 100 > &output, std::map< std::string, size_t > &words↩ _count, const std::vector< size_t > &text_single_term_hashes, const std::vector< size_t > &text_double↩ _term_hashes, const std::vector< size_t > &text_triple_term_hashes, const std::vector< size_t > &stop_↩ words_hashes, const std::vector< std::pair< size_t, size_t >> &hash_table, const std::vector< std::bitset< 100 >> &matrices, const std::vector< std::string > &dictionary)

    correcting input text hashes, maybe there are duplicates
- void extract_words_count (const std::map< std::string, size_t > &m)
- void write_in_file ()
- void print_hash_table (const std::vector< std::pair< size_t, size_t >> &hash_table, size_t begin=0, size_t end=466548)
- void print_duplicates (const std::vector< std::vector< size_t >> &duplicates, const std::vector< std::string > &dictionary)
- void print_matrix (const std::vector< size_t > &matrix)
- void print_words (const std::vector< std::string > &dictionary)
- void print_vector (const std::vector< size_t > &vector)
- void print_bitset (const std::bitset< 100 > &bitset)
- void print_bitset_vector (const std::bitset< 100 > &bitset)
- bool contains_duplicates (std::vector< size_t > a)
- bool contains_duplicates_h (std::vector< size_t > a)
- void init_random_matrices (std::vector< std::vector< size_t >> &matrices)

2.2.1 Function Documentation

2.2.1.1 contains_duplicates()

```
bool contains_duplicates (
            std::vector< size_t > a )
```

Definition at line 426 of file functions.cpp.

```
{
    if (a.size() < 2)
    {
        return false;
    }
    sort(a.begin(), a.end());
    //std::cout << a[a.size() - 1] << std::endl;
    //std::cout << a[0] << std::endl;
    for (int i = 0; i < a.size() - 1; i++)
    {
        if (a[i] == a[i + 1])
        {
            std::cout << a[i] << std::endl;
        }
    }
    return false;
}
```

2.2.1.2 contains_duplicates_h()

```
bool contains_duplicates_h (
              std::vector< size_t > a )
```

Definition at line 445 of file functions.cpp.

```
{
    if (a.size() < 2)
    {
        return false;
    }

    for (int i = 0; i < a.size() - 1; i++)
    {
        a[i] = a[i] % a.size();
    }

    sort(a.begin(), a.end());
    //std::cout << a[a.size() - 1] << std::endl;
    //std::cout << a[0] << std::endl;
    for (int i = 0; i < a.size() - 1; i++)
    {
        if (a[i] == a[i + 1])
        {
            std::cout << a[i] << std::endl;
        }
    }
    return false;
}
```

2.2.1.3 correct_dict_hashes_and_extract_duplicates()

```
void correct_dict_hashes_and_extract_duplicates (
              std::vector< std::vector< size_t >> & duplicates,
              std::vector< size_t > & hashes )
```

finding duplicates and correcting dictionary hashes

Parameters

duplicates	- contain words old hashes, indexes to the words with that hashes, and new hashes
hashes	- hash of dictionary, contain hashed strings of each word in dictionary

Definition at line 171 of file functions.cpp.

References duplicates.

Referenced by BOOST_AUTO_TEST_CASE(), and main().

```
{
    auto start = std::chrono::steady_clock::now();
    size_t size = hashes.size();
    std::map<size_t, size_t> m;

    if (size < 2)
    {
        return;
    }

    for (size_t i = 0; i < size; ++i)
    {
        m[hashes[i]] = 0;
    }
    for (size_t i = 0; i < size; ++i)
    {
        if (m[hashes[i]])
        {
            size_t temp = hashes[i];
            while (m[temp])
            {
                temp++;
            }
            duplicates.push_back({ hashes[i], i, temp });
            hashes[i] = temp;
        }
        else
        {
            m[hashes[i]] = 1;
        }
    }
    auto end = std::chrono::steady_clock::now();
    BOOST_LOG_TRIVIAL(debug) << "correct_dict_hashes_and_extract_duplicates()  "
        << std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << " ms";
}
```

Here is the caller graph for this function:

2.2.1.4 correct_text_hashes()

```
void correct_text_hashes (
            std::vector< size_t > & text_hashes,
            const std::vector< std::string > & text,
            const std::vector< std::vector< size_t >> & duplicates,
            const std::vector< std::string > & dictionary )
```

correcting input text hashes, maybe there are duplicates

Parameters

text_hashes	- hash of text, contain hashed strings of each word in text
text	- contain all words in input text(maximum 500terms)
duplicates	- contain words old hashes, indexes to the words with that hashes, and new hashes
dictionary	- conatin all words in dictionary.txt

Definition at line 210 of file functions.cpp.

References duplicates, and text_size.

Referenced by main().

```
{
    auto start = std::chrono::steady_clock::now();
    size_t duplicate_size = duplicates.size();
    size_t text_size = text_hashes.size();
    for (size_t i = 0; i < duplicates.size(); ++i)
    {
        for (size_t j = 0; j < text_size; ++j)
        {
            if (text_hashes[j] == duplicates[i][0])
            {
                if (text[j] == dictionary[duplicates[i][1]])
                {
                    text_hashes[j] = duplicates[i][2];
                }
            }
        }
    }
    auto end = std::chrono::steady_clock::now();
    BOOST_LOG_TRIVIAL(debug) << "correct_text_hashes()  "
        << std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << " ms";
}
```

Here is the caller graph for this function:

2.2.1.5 extract_words_count()

```
void extract_words_count (
            const std::map< std::string, size_t > & m )
```

Definition at line 327 of file functions.cpp.

Referenced by main().

```
{
    typedef std::function<bool(std::pair<std::string, int>, std::pair<std::string, int>)> Comparator;

    Comparator compFunctor = [](std::pair<std::string, size_t> elem1 ,std::pair<std::string, size_t> elem2)
    {
        return elem1.second > elem2.second;
    };

    std::set<std::pair<std::string, int>, Comparator> setOfWords(
            m.begin(), m.end(), compFunctor);

    // Iterate over a set using range base for loop
    // It will display the items in sorted order of values
    for (std::pair<std::string, int> element : setOfWords)
        BOOST_LOG_TRIVIAL(info) << element.first << " :: " << element.second;
}
```

Here is the caller graph for this function:

2.2.1.6 hash_function()

```
void hash_function (
            std::vector< std::pair< size_t, size_t >> & hash_table,
            const std::vector< size_t > & hashes )
```

hashing open adressing with linear probbing algorithm

Parameters

hash_table	- hash table contain hashes and indexes to the words in dictionary
hashes	- hash of dictionary, contain hashed strings of each word in dictionary

Definition at line 236 of file functions.cpp.

References hash_table(), and words_count.

Referenced by BOOST_AUTO_TEST_CASE(), and main().

```
{
    auto start = std::chrono::steady_clock::now();
    size_t words_count = hashes.size();
    for (size_t i = 0; i < words_count; ++i)
    {
        size_t index = hashes[i] % words_count;
        while (hash_table[index].first)
        {
            index = (index + 1) % words_count;
        }
        hash_table[index] = std::make_pair(hashes[i], i);
    }
    auto end = std::chrono::steady_clock::now();
    BOOST_LOG_TRIVIAL(debug) << "hash_function()  "
        << std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << " ms";
}
```

Here is the call graph for this function: Here is the caller graph for this function:

2.2.1.7 init_hashes()

```
void init_hashes (
            std::vector< size_t > & hashes,
            const std::hash< std::string > & hasher,
            const std::vector< std::string > & words )
```

calculating hashes of words stored in words

Parameters

hasher	- object that hashes words
words	- all words that need to be hashed

Definition at line 136 of file functions.cpp.

References hasher.

Referenced by BOOST_AUTO_TEST_CASE(), and main().

```
{
    auto start = std::chrono::steady_clock::now();
    size_t size = hashes.size();
    if (size == words.size() - 2)
    {
        for (size_t i = 0; i < size; ++i)
        {
            hashes[i] = hasher(words[i] + "-" +
                                words[i + 1] + "-" +
                                words[i + 2]);
        }
    }
    else if (size == words.size() - 1)
    {
        for (size_t i = 0; i < size; ++i)
        {
            hashes[i] = hasher(words[i] + "-" +
                                words[i + 1]);
        }
    }
    else
    {
        for (size_t i = 0; i < size; ++i)
        {
            hashes[i] = hasher(words[i]);
        }
    }

    auto end = std::chrono::steady_clock::now();
    BOOST_LOG_TRIVIAL(debug) << "init_hashes()  "
        << std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << " ms";
}
```

Here is the caller graph for this function:

2.2.1.8 init_logging()

`void init_logging ( )`

This function helps in logging level.

Definition at line 22 of file functions.cpp.

Referenced by main().

```
{
    logging::register_simple_formatter_factory<logging::trivial::severity_level, char>("Severity");

    logging::add_file_log(
        keywords::file_name = "logfile.log",
        keywords::format = "[%TimeStamp%] [%ThreadID%] [%Severity%] [%ProcessID%] [%LineID%] %Message%"
    );

    logging::core::get()->set_filter
    (
        logging::trivial::severity >= logging::trivial::trace
    );

    logging::add_common_attributes();
}
```

Here is the caller graph for this function:

2.2.1.9 init_random_matrices() [1/2]

```
void init_random_matrices (
            std::vector< std::bitset< 100 >> & matrices )
```

initializing random matrices

Parameters

matrices	- vector of bitsets, contain all initialized matrices of length 100

Definition at line 39 of file functions.cpp.

References dict_size, and matrices().

Referenced by BOOST_AUTO_TEST_CASE(), and main().

```
{
    auto start = std::chrono::steady_clock::now();
    size_t dict_size = matrices.size();
    std::default_random_engine dre(std::random_device{}());
    std::uniform_int_distribution<long long> dist(0, (1ll << 50) - 1);
    for (size_t i = 0; i < dict_size; ++i)
    {
        matrices[i] = dist(dre);
        matrices[i] <<= 50;
        matrices[i] |= dist(dre);
    }
    auto end = std::chrono::steady_clock::now();
    BOOST_LOG_TRIVIAL(debug) << "init_random_matrices()  "
        << std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << " ms";
}
```

Here is the call graph for this function: Here is the caller graph for this function:

2.2.1.10 init_random_matrices() [2/2]

```
void init_random_matrices (
            std::vector< std::vector< size_t >> & matrices )
```

Definition at line 470 of file functions.cpp.

References dict_size, matrices(), and matrix_size.

```
{
    size_t dict_size = matrices.size();
    size_t matrix_size = matrices[0].size();

    for (size_t i = 0; i < dict_size; ++i)
    {
        for (size_t j = 0; j < matrix_size; ++j)
        {
            matrices[i][j] = rand() % 2;
        }
    }
}
```

Here is the call graph for this function:

2.2.1.11 print_bitset()

```
void print_bitset (
            const std::bitset< 100 > & bitset )
```

Definition at line 405 of file functions.cpp.

```
{
    for (size_t j = 0; j < 10; ++j)
    {
        for (size_t i = j * 10; i < j * 10 + 10; ++i)
        {
            std::cout << bitset[i] << " ";
        }
        std::cout << std::endl;
    }
}
```

2.2.1.12 print_bitset_vector()

```
void print_bitset_vector (
            const std::bitset< 100 > & bitset )
```

Definition at line 417 of file functions.cpp.

```
{
    for (size_t j = 0; j < bitset.size(); ++j)
    {
        std::cout << bitset[j];
    }
    std::cout << std::endl;
}
```

2.2.1.13 print_duplicates()

```
void print_duplicates (
            const std::vector< std::vector< size_t >> & duplicates,
            const std::vector< std::string > & dictionary )
```

Definition at line 354 of file functions.cpp.

References duplicates.

```
{
    for (int i = 0; i < duplicates.size(); ++i)
    {
        for (int j = 0; j < duplicates[0].size(); ++j)
        {
            if (j == 1)
            {
                std::cout << "dictionary[" << duplicates[i][j] << "] = " << dictionary[duplicates[i][j]] <<
      " ";
            }
            else
            {
                std::cout << duplicates[i][j] << " ";
            }

        }
        std::cout << std::endl;
    }
}
```

2.2.1.14 print_hash_table()

```
void print_hash_table (
            const std::vector< std::pair< size_t, size_t >> & hash_table,
            size_t begin = 0,
            size_t end = 466548 )
```

Definition at line 345 of file functions.cpp.

References hash_table().

```
{
    for (size_t i = begin; i < end; ++i)
    {
        //std::cout << "hash = " << hash_table[i].first << "  index = " << hash_table[i].second <<
      std::endl;
        std::cout << hash_table[i].first << " " << hash_table[i].second << std::endl;
    }
}
```

Here is the call graph for this function:

2.2.1.15 print_matrix()

```
void print_matrix (
            const std::vector< size_t > & matrix )
```

Definition at line 375 of file functions.cpp.

```
{
    for (size_t j = 0; j < 10; ++j)
    {
        for (size_t i = j * 10; i < j * 10 + 10; ++i)
        {
            std::cout << matrix[i] << " ";
        }
        std::cout << std::endl;
    }
}
```

2.2.1.16 print_vector()

```
void print_vector (
            const std::vector< size_t > & vector )
```

Definition at line 396 of file functions.cpp.

```
{
    for (size_t i = 0; i < vector.size(); ++i)
    {
        std::cout << vector[i];
    }
    std::cout << std::endl;
}
```

2.2.1.17 print_words()

```
void print_words (
            const std::vector< std::string > & dictionary )
```

Definition at line 387 of file functions.cpp.

```
{
    for (size_t i = 0; i < dictionary.size(); ++i)
    {
        std::cout << dictionary[i] << std::endl;
    }
    std::cout << std::endl;
}
```

2.2.1.18 read_dict_from_file()

```
void read_dict_from_file (
            const std::string & dictionary_filename,
            std::vector< std::string > & dictionary )
```

reading dictionary from file

Parameters

dictionary_filename	- the filename that need to be read
dictionary	- contain all words in the file

Definition at line 56 of file functions.cpp.

Referenced by BOOST_AUTO_TEST_CASE(), and main().

```
{
    auto start = std::chrono::steady_clock::now();
    std::ifstream file(dictionary_filename);
    size_t counter = 0;
    if (file.is_open())
    {
        std::string line;
        size_t i = 0;
        while (std::getline(file, line))
        {
            dictionary[i++] = line;
            counter++;
        }
        dictionary.resize(counter);
        file.close();
    }
    else
    {
        BOOST_LOG_TRIVIAL(error) << "Couldn't open " << dictionary_filename << " for reading";
        std::cerr << "Couldn't open " << dictionary_filename << " for reading\n";
    }
    auto end = std::chrono::steady_clock::now();
    BOOST_LOG_TRIVIAL(debug) << "read_dict_from_file()  "
        << std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << " ms";
}
```

Here is the caller graph for this function:

2.2.1.19 read_text_from_file()

```
void read_text_from_file (
            const std::string & text_filename,
            std::vector< std::string > & text )
```

reading input text, stop words from file

Parameters

text_filename	- the filename that need to be read
text	- contain all words in the file

Definition at line 83 of file functions.cpp.

Referenced by BOOST_AUTO_TEST_CASE(), and main().

```
{
    auto start = std::chrono::steady_clock::now();
    std::ifstream file(text_filename);
    if (file.is_open())
    {
        std::string line;
        size_t i = 0, counter = 0;
```

```
       while (std::getline(file, line))
       {
           //std::cout << line << " " << line.size() << std::endl;
           size_t size = line.size();;
           if(line[size - 1] < 'A' || line[size - 1] > 'Z' &&
               line[size - 1] < 'a' || line[size - 1] > 'z')
           {
               line.pop_back();
           }

            char* str = &line[0];
            char* pch;
            pch = strtok(str, " ,.-?");

            //std::cout << line << " " << line.size() << std::endl;

            while (pch != NULL)
            {
                for (size_t j = 0; j < strlen(pch); j++)
                {
                    if (pch[j] >= 65 && pch[j] <= 92)
                    {
                        pch[j] = pch[j] + 32;
                    }
                }
                //std::cout << pch << " " << strlen(pch) << std::endl;
                text[i++] = pch;
                pch = strtok(NULL, " ,.-?");
                counter++;
            }
        }
        text.resize(counter);
        file.close();
    }
    else
    {
        BOOST_LOG_TRIVIAL(error) << "Couldn't open " << text_filename << " for reading";
        std::cerr << "Couldn't open " << text_filename << " for reading\n";
    }

    auto end = std::chrono::steady_clock::now();
    BOOST_LOG_TRIVIAL(debug) << "read_from_text_file()  "
        << std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << " ms";
}
```

Here is the caller graph for this function:

2.2.1.20 search_and_calculate_matrices()

```
void search_and_calculate_matrices (
            std::bitset< 100 > & output,
            std::map< std::string, size_t > & words_count,
            const std::vector< size_t > & text_single_term_hashes,
            const std::vector< size_t > & text_double_term_hashes,
            const std::vector< size_t > & text_triple_term_hashes,
            const std::vector< size_t > & stop_words_hashes,
            const std::vector< std::pair< size_t, size_t >> & hash_table,
            const std::vector< std::bitset< 100 >> & matrices,
            const std::vector< std::string > & dictionary )
```

correcting input text hashes, maybe there are duplicates

Parameters

output	- the result of the program, score of the sentence
text_single_term_hashes	- hash of text, contain hashed strings of each word in text
text_double_term_hashes	- hash of text, contain hashed strings of each double combination of words in text
stop_words_hashes	- contain all hashes of stop words
hash_table	- hash table contain hashes and indexes to the words in dictionary
matrices	- vector of bitsets, contain all initialized matrices of length 100
dictionary	- conatin all words in dictionary.txt

Definition at line 255 of file functions.cpp.

References dict_size, hash_table(), matrices(), matrix_size, text_double_term_hashes(), text_single_term_↩
hashes(), text_size, and text_triple_term_hashes().

Referenced by BOOST_AUTO_TEST_CASE(), and main().

```
{
    auto start = std::chrono::steady_clock::now();
    size_t text_size = text_single_term_hashes.size();
    size_t dict_size = dictionary.size();
    size_t matrix_size = output.size();
    std::vector<size_t> indexes(text_size);
    std::vector<const std::vector<size_t>*> terms(3);
    terms[0] = &text_single_term_hashes;
    terms[1] = &text_double_term_hashes;
    terms[2] = &text_triple_term_hashes;
    for(size_t k = 3; k > 0; --k)
    {
        for (size_t i = 0; i < text_size - k + 1; ++i)
        {
            bool b = true;
            for(size_t l = 0; l < k; ++l)
            {
                if (indexes[i + l])
                {
                    b = false;
                }
            }
            if(k == 1)
            {
                for (size_t l = 0; l < stop_words_hashes.size(); ++l)
                {
                    if (stop_words_hashes[l] == (*terms[0])[i])
                    {
                        indexes[i] = 1;
                        b = false;
                    }
                }
            }
            if (b)
            {
                size_t index = ((*terms[k - 1])[i]) % dict_size;
                for (size_t j = 0; j < dict_size; ++j)
                {
                    if (hash_table[index].first == ((*terms[k - 1])[i]))
                    {
                        for(size_t l = 0; l < k; ++l)
                        {
                            indexes[i + l] = 1;
                        }
                        output |= matrices[hash_table[index].second];
                        size_t& value = words_count[dictionary[hash_table[index].second]];
                        value? value++ : value = 1;
                        break;
                    }
                    else
                    {
                        index = (index + 1) % dict_size;
                    }
                }
            }
        }
    }

    auto end = std::chrono::steady_clock::now();
    BOOST_LOG_TRIVIAL(debug) << "search_and_calculate_matrices()  "
        << std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << " ms";
}
```

Here is the call graph for this function: Here is the caller graph for this function:

2.2.1.21 write_in_file()

```
void write_in_file ( )
```

2.3 src/main.cpp File Reference

```
#include "functions.hpp"
#include <iostream>
#include <time.h>
#include <string.h>
#include <chrono>
#include <map>
#include <set>
#include <algorithm>
#include <functional>
#include <boost/log/trivial.hpp>
```

Include dependency graph for main.cpp:

Macros

- #define max_dict_size 500000
- #define max_term_size 20000
- #define max_stop_words_size 200
- #define matrix_size 100

Functions

- int main ()

2.3.1 Macro Definition Documentation

2.3.1.1 matrix_size

```
#define matrix_size 100
```

Definition at line 15 of file main.cpp.

Referenced by init_random_matrices(), main(), and search_and_calculate_matrices().

2.3.1.2 max_dict_size

```
#define max_dict_size 500000
```

Definition at line 12 of file main.cpp.

Referenced by main().

2.3.1.3 max_stop_words_size

```
#define max_stop_words_size 200
```

Definition at line 14 of file main.cpp.

Referenced by main().

2.3.1.4 max_term_size

```
#define max_term_size 20000
```

Definition at line 13 of file main.cpp.

Referenced by main().

2.3.2 Function Documentation

2.3.2.1 main()

```
int main ( )
```

Definition at line 17 of file main.cpp.

References correct_dict_hashes_and_extract_duplicates(), correct_text_hashes(), dict_hashes(), dict_size, dictionary(), dictionary_filename, duplicates, duplicates_size, extract_words_count(), hash_function(), hash_table(), hasher, init_hashes(), init_logging(), init_random_matrices(), matrices(), matrix_size, max_dict_size, max_stop_words_size, max_term_size, output(), read_dict_from_file(), read_text_from_file(), search_and_calculate_matrices(), stop_words(), stop_words_filename, stop_words_hashes(), stop_words_size, text(), text_double_term_hashes(), text_filename, text_single_term_hashes(), text_size, text_triple_term_hashes(), and words_count.

```
{
    init_logging();

    auto start = std::chrono::steady_clock::now();
    const std::string dictionary_filename = "dictionary.txt";
    std::vector<std::string> dictionary(max_dict_size);
    read_dict_from_file(dictionary_filename, dictionary);
    size_t dict_size = dictionary.size();

    const std::string text_filename = "text2.txt";
    std::vector<std::string> text(max_term_size);
    read_text_from_file(text_filename, text);
    size_t text_size = text.size();

    const std::string stop_words_filename = "stop_words.txt";
    std::vector<std::string> stop_words(max_stop_words_size);
    read_text_from_file(stop_words_filename, stop_words);
    size_t stop_words_size = stop_words.size();

    std::hash<std::string> hasher;
    std::vector<std::vector<size_t>> duplicates;

    std::vector<size_t> dict_hashes(dict_size);
    init_hashes(dict_hashes, hasher, dictionary);
    correct_dict_hashes_and_extract_duplicates(duplicates,
   dict_hashes);
```

```
    size_t duplicates_size = duplicates.size();

    std::vector<size_t> text_single_term_hashes(text_size);
    init_hashes(text_single_term_hashes, hasher,
   text);
    if(duplicates_size)
    {
        correct_text_hashes(text_single_term_hashes,
   text, duplicates, dictionary);
    }

    std::vector<size_t> text_double_term_hashes(text_size - 1);
    init_hashes(text_double_term_hashes, hasher,
   text);
    if(duplicates_size)
    {
        correct_text_hashes(text_double_term_hashes,
   text, duplicates, dictionary);
    }

    std::vector<size_t> text_triple_term_hashes(text_size - 2);
    init_hashes(text_triple_term_hashes, hasher,
   text);
    if(duplicates_size)
    {
        correct_text_hashes(text_triple_term_hashes,
   text, duplicates, dictionary);
    }

    std::vector<size_t> stop_words_hashes(stop_words_size);
    init_hashes(stop_words_hashes, hasher,
   stop_words);
    if(duplicates_size)
    {
        correct_text_hashes(stop_words_hashes,
   stop_words, duplicates, dictionary);
    }

    std::vector<std::bitset<100>> matrices(dict_size);
    init_random_matrices(matrices);

    std::vector<std::pair<size_t,size_t>> hash_table(dict_size);
    hash_function(hash_table, dict_hashes);

    std::map<std::string, size_t> words_count;
    std::bitset<100> output(matrix_size);
    search_and_calculate_matrices(output, words_count,
   text_single_term_hashes, text_double_term_hashes,
        text_triple_term_hashes, stop_words_hashes,
   hash_table, matrices, dictionary);

    extract_words_count(words_count);
    BOOST_LOG_TRIVIAL(info) << "Dictionary size = " << dict_size;
    BOOST_LOG_TRIVIAL(debug) << "Duplicates count = " << duplicates.size();
    BOOST_LOG_TRIVIAL(info) << "Stop words count = " << stop_words.size();
    BOOST_LOG_TRIVIAL(info) << "Input words count = " << text_single_term_hashes.
   size();
    BOOST_LOG_TRIVIAL(info) << "The result is = " << output;

    auto end = std::chrono::steady_clock::now();
    BOOST_LOG_TRIVIAL(debug) << "Elapsed time in milliseconds : "
        << std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << " ms";
    return 0;
}
```

Here is the call graph for this function:

2.4 src/unit_tests.cpp File Reference

```
#include "functions.hpp"
#include <unordered_set>
#include <bitset>
#include <boost/test/unit_test.hpp>
#include <boost/log/trivial.hpp>
#include <boost/log/core.hpp>
#include <boost/log/expressions.hpp>
```

```
#include <boost/log/utility/setup/file.hpp>
#include <boost/log/utility/setup/common_attributes.hpp>
```
Include dependency graph for unit_tests.cpp:

Macros

- #define BOOST_TEST_MODULE functions
- #define max_dict_size 500000
- #define max_term_size 20000
- #define max_stop_words_size 200
- #define matrix_size 100

Functions

- std::vector< std::string > dictionary (500000)
- std::vector< size_t > dict_hashes (500000)
- std::vector< std::string > text (20000)
- std::vector< size_t > text_single_term_hashes (20000)
- std::vector< size_t > text_double_term_hashes (20000)
- std::vector< size_t > text_triple_term_hashes (20000)
- std::vector< std::string > stop_words (200)
- std::vector< size_t > stop_words_hashes (200)
- std::vector< std::pair< size_t, size_t > > hash_table (500000)
- std::vector< std::bitset< 100 > > matrices (500000)
- std::bitset< 100 > output (100)
- BOOST_AUTO_TEST_CASE (read_dict_from_file_test)
- BOOST_AUTO_TEST_CASE (read_text_from_file_test)
- BOOST_AUTO_TEST_CASE (init_random_matrices_test)
- BOOST_AUTO_TEST_CASE (init_hashes_test)
- BOOST_AUTO_TEST_CASE (correct_dict_hashes_and_extract_duplicates_test)
- BOOST_AUTO_TEST_CASE (hash_function_test)
- BOOST_AUTO_TEST_CASE (search_and_calculate_matrices_test)

Variables

- std::hash< std::string > hasher
- const std::string dictionary_filename = "dictionary.txt"
- size_t dict_size
- const std::string text_filename = "text1.txt"
- size_t text_size
- const std::string stop_words_filename = "stop_words.txt"
- size_t stop_words_size
- std::vector< std::vector< size_t > > duplicates
- size_t duplicates_size
- std::map< std::string, size_t > words_count

2.4.1 Macro Definition Documentation

2.4.1.1 BOOST_TEST_MODULE

```
#define BOOST_TEST_MODULE functions
```

Definition at line 1 of file unit_tests.cpp.

2.4.1.2 matrix_size

```
#define matrix_size 100
```

Definition at line 17 of file unit_tests.cpp.

2.4.1.3 max_dict_size

```
#define max_dict_size 500000
```

Definition at line 14 of file unit_tests.cpp.

Referenced by BOOST_AUTO_TEST_CASE().

2.4.1.4 max_stop_words_size

```
#define max_stop_words_size 200
```

Definition at line 16 of file unit_tests.cpp.

2.4.1.5 max_term_size

```
#define max_term_size 20000
```

Definition at line 15 of file unit_tests.cpp.

Referenced by BOOST_AUTO_TEST_CASE().

2.4.2 Function Documentation

2.4.2.1 BOOST_AUTO_TEST_CASE() [1/7]

```
BOOST_AUTO_TEST_CASE (
              read_dict_from_file_test  )
```

Definition at line 47 of file unit_tests.cpp.

References dict_size, dictionary(), dictionary_filename, max_dict_size, and read_dict_from_file().

```
{
    read_dict_from_file(dictionary_filename,
   dictionary);
    dict_size = dictionary.size();
    BOOST_REQUIRE_LE( dict_size, max_dict_size);
    for(size_t i = 0; i < dict_size; ++i)
    {
        BOOST_CHECK_NE(dictionary[i].size(), 0);
        for(size_t j = 0; j < dictionary[i].size(); ++j)
        {
            // Decimal 32 = [Space] character
            // Decimal 127 = [DEL] character
            // (32, 127) in this area characters are visible
            BOOST_WARN(dictionary[i][j] > 32 &&
                        dictionary[i][j] < 127);
        }
    }
}
```

Here is the call graph for this function:

2.4.2.2 BOOST_AUTO_TEST_CASE() [2/7]

```
BOOST_AUTO_TEST_CASE (
              read_text_from_file_test  )
```

Definition at line 66 of file unit_tests.cpp.

References max_term_size, read_text_from_file(), stop_words(), stop_words_filename, stop_words_size, text(), text_filename, and text_size.

```
{
    read_text_from_file(text_filename, text);
    text_size = text.size();
    BOOST_REQUIRE_LE( text_size, max_term_size);
    for(size_t i = 0; i < text_size; ++i)
    {
        BOOST_CHECK_NE(text[i].size(), 0);
        for(size_t j = 0; j < text[i].size(); ++j)
        {
            // Decimal 32 = [Space] character
            // Decimal 127 = [DEL] character
            // (32, 127) in this area characters are visible
            BOOST_WARN(text[i][j] > 32 &&
                    text[i][j] < 127);
        }
    }

    read_text_from_file(stop_words_filename,
   stop_words);
    stop_words_size = stop_words.size();
    BOOST_REQUIRE_LE( stop_words_size, max_term_size);
    for(size_t i = 0; i < stop_words_size; ++i)
    {
        BOOST_CHECK_NE(stop_words[i].size(), 0);
        for(size_t j = 0; j < stop_words[i].size(); ++j)
        {
            // Decimal 32 = [Space] character
            // Decimal 127 = [DEL] character
            // (32, 127) in this area characters are visible
            BOOST_WARN(stop_words[i][j] > 32 &&
                    stop_words[i][j] < 127);
        }
    }
}
```

Here is the call graph for this function:

2.4.2.3 BOOST_AUTO_TEST_CASE() [3/7]

```
BOOST_AUTO_TEST_CASE (
              init_random_matrices_test   )
```

Definition at line 101 of file unit_tests.cpp.

References dict_size, init_random_matrices(), and matrices().

```
102 {
103     matrices.resize(dict_size);
104     init_random_matrices(matrices);
105     size_t size = matrices.size();
106     std::unordered_set<std::bitset<matrix_size>> un_set(matrices.begin(),
      matrices.end());
107     BOOST_WARN_EQUAL(un_set.size(), size);
108 }
```

Here is the call graph for this function:

2.4.2.4 BOOST_AUTO_TEST_CASE() [4/7]

```
BOOST_AUTO_TEST_CASE (
              init_hashes_test   )
```

Definition at line 110 of file unit_tests.cpp.

References dict_hashes(), dict_size, dictionary(), hasher, init_hashes(), stop_words(), stop_words_hashes(), stop_words_size, text(), text_double_term_hashes(), text_single_term_hashes(), text_size, and text_triple_term_↩ hashes().

```
111 {
112     dict_hashes.resize(dict_size);
113     init_hashes(dict_hashes, hasher, dictionary);
114     BOOST_CHECK_EQUAL(dict_hashes.size(), dict_size);
115     size_t size = dict_hashes.size();
116     std::unordered_set<std::bitset<matrix_size>> un_set(dict_hashes.begin(),
      dict_hashes.end());
117     BOOST_WARN_EQUAL( un_set.size(), size );
118     un_set.clear();
119 
120     text_single_term_hashes.resize(text_size);
121     init_hashes(text_single_term_hashes,
      hasher, text);
122     BOOST_WARN_EQUAL(text_single_term_hashes.size(),
      text_size);
123     size = text_single_term_hashes.size();
124     un_set = std::unordered_set<std::bitset<matrix_size>>(
      text_single_term_hashes.begin(),text_single_term_hashes.end()
      );
125     BOOST_WARN_LE( un_set.size(), size );
126     un_set.clear();
127 
128 
129     text_double_term_hashes.resize(text_size - 1);
130     init_hashes(text_double_term_hashes,
      hasher, text);
131     BOOST_CHECK_EQUAL(text_double_term_hashes.size(),
      text_size - 1);
132     size = text_double_term_hashes.size();
133     un_set = std::unordered_set<std::bitset<matrix_size>>(
      text_double_term_hashes.begin(),text_double_term_hashes.end()
      );
134     BOOST_WARN_LE( un_set.size(), size );
135     un_set.clear();
136 
137     text_triple_term_hashes.resize(text_size - 2);
138     init_hashes(text_triple_term_hashes,
      hasher, text);
139     BOOST_CHECK_EQUAL(text_triple_term_hashes.size(),
      text_size - 2);
        size = text_triple_term_hashes.size();
        un_set = std::unordered_set<std::bitset<matrix_size>>(
      text_triple_term_hashes.begin(),text_triple_term_hashes.end()
      );
        BOOST_WARN_LE( un_set.size(), size );
        un_set.clear();

        stop_words_hashes.resize(stop_words_size);
        init_hashes(stop_words_hashes, hasher,
      stop_words);
        BOOST_CHECK_EQUAL(stop_words_hashes.size(),stop_words_size);
        size = stop_words_hashes.size();
        un_set = std::unordered_set<std::bitset<matrix_size>>(stop_words_hashes.begin(),
      stop_words_hashes.end());
        BOOST_WARN_EQUAL( un_set.size(), size );
    }
```

Here is the call graph for this function:

2.4.2.5 BOOST_AUTO_TEST_CASE() [5/7]

```
BOOST_AUTO_TEST_CASE (
             correct_dict_hashes_and_extract_duplicates_test   )
```

Definition at line 153 of file unit_tests.cpp.

References correct_dict_hashes_and_extract_duplicates(), dict_hashes(), dict_size, and duplicates_size.

```
    {
        correct_dict_hashes_and_extract_duplicates(
      duplicates,dict_hashes);
        size_t duplicates_size = duplicates.size();
        BOOST_CHECK_EQUAL(duplicates_size, 0);
        BOOST_CHECK_EQUAL(dict_hashes.size(), dict_size);
        size_t size = dict_hashes.size();
        std::unordered_set<std::bitset<matrix_size>> un_set(dict_hashes.begin(),
      dict_hashes.end());
        BOOST_WARN_EQUAL( un_set.size(), size );
    }
```

Here is the call graph for this function:

2.4.2.6 BOOST_AUTO_TEST_CASE() [6/7]

```
BOOST_AUTO_TEST_CASE (
             hash_function_test   )
```

Definition at line 164 of file unit_tests.cpp.

References dict_hashes(), dict_size, hash_function(), and hash_table().

```
    {
        hash_table.resize(dict_size);
        hash_function(hash_table, dict_hashes);
        BOOST_CHECK_EQUAL(hash_table.size(), dict_hashes.size());
        for(size_t i = 0; i < dict_size; ++i)
        {
            BOOST_CHECK_NE(hash_table[i].first,0);
        }
    }
```

Here is the call graph for this function:

2.4.2.7 BOOST_AUTO_TEST_CASE() [7/7]

```
BOOST_AUTO_TEST_CASE (
            search_and_calculate_matrices_test  )
```

Definition at line 175 of file unit_tests.cpp.

References dictionary(), hash_table(), matrices(), output(), search_and_calculate_matrices(), stop_words_↩ hashes(), text_double_term_hashes(), text_single_term_hashes(), and text_triple_term_hashes().

```
{
    std::bitset<matrix_size> temp = output;
    search_and_calculate_matrices(output, words_count, text_single_term_hashes, text_double_term_hashes,
        text_triple_term_hashes, stop_words_hashes, hash_table, matrices, dictionary);
    BOOST_CHECK_NE(temp, output);
}
```

Here is the call graph for this function:

2.4.2.8 dict_hashes()

```
std::vector<size_t> dict_hashes (
            500000  )
```

Referenced by BOOST_AUTO_TEST_CASE(), and main().

Here is the caller graph for this function:

2.4.2.9 dictionary()

```
std::vector<std::string> dictionary (
            500000  )
```

Referenced by BOOST_AUTO_TEST_CASE(), and main().

Here is the caller graph for this function:

2.4.2.10 hash_table()

```
std::vector<std::pair<size_t,size_t> > hash_table (
            500000  )
```

Referenced by BOOST_AUTO_TEST_CASE(), hash_function(), main(), print_hash_table(), and search_and_↩ calculate_matrices().

Here is the caller graph for this function:

2.4.2.11 matrices()

```
std::vector<std::bitset< 100 > > matrices (
             500000 )
```

Referenced by BOOST_AUTO_TEST_CASE(), init_random_matrices(), main(), and search_and_calculate_↩ matrices().

Here is the caller graph for this function:

2.4.2.12 output()

```
std::bitset< 100 > output (
             100 )
```

Referenced by BOOST_AUTO_TEST_CASE(), and main().

Here is the caller graph for this function:

2.4.2.13 stop_words()

```
std::vector<std::string> stop_words (
             200 )
```

Referenced by BOOST_AUTO_TEST_CASE(), and main().

Here is the caller graph for this function:

2.4.2.14 stop_words_hashes()

```
std::vector<size_t> stop_words_hashes (
             200 )
```

Referenced by BOOST_AUTO_TEST_CASE(), and main().

Here is the caller graph for this function:

2.4.2.15 text()

```
std::vector<std::string> text (
             20000 )
```

Referenced by BOOST_AUTO_TEST_CASE(), and main().

Here is the caller graph for this function:

2.4.2.16 text_double_term_hashes()

```
std::vector<size_t> text_double_term_hashes (
            20000  )
```

Referenced by BOOST_AUTO_TEST_CASE(), main(), and search_and_calculate_matrices().

Here is the caller graph for this function:

2.4.2.17 text_single_term_hashes()

```
std::vector<size_t> text_single_term_hashes (
            20000  )
```

Referenced by BOOST_AUTO_TEST_CASE(), main(), and search_and_calculate_matrices().

Here is the caller graph for this function:

2.4.2.18 text_triple_term_hashes()

```
std::vector<size_t> text_triple_term_hashes (
            20000  )
```

Referenced by BOOST_AUTO_TEST_CASE(), main(), and search_and_calculate_matrices().

Here is the caller graph for this function:

2.4.3 Variable Documentation

2.4.3.1 dict_size

```
size_t dict_size
```

Definition at line 24 of file unit_tests.cpp.

Referenced by BOOST_AUTO_TEST_CASE(), init_random_matrices(), main(), and search_and_calculate_↩
matrices().

2.4.3.2 dictionary_filename

```
const std::string dictionary_filename = "dictionary.txt"
```

Definition at line 21 of file unit_tests.cpp.

Referenced by BOOST_AUTO_TEST_CASE(), and main().

2.4.3.3 duplicates

```
std::vector<std::vector<size_t> > duplicates
```

Definition at line 38 of file unit_tests.cpp.

Referenced by correct_dict_hashes_and_extract_duplicates(), correct_text_hashes(), main(), and print_↩
duplicates().

2.4.3.4 duplicates_size

```
size_t duplicates_size
```

Definition at line 39 of file unit_tests.cpp.

Referenced by BOOST_AUTO_TEST_CASE(), and main().

2.4.3.5 hasher

```
std::hash<std::string> hasher
```

Definition at line 19 of file unit_tests.cpp.

Referenced by BOOST_AUTO_TEST_CASE(), init_hashes(), and main().

2.4.3.6 stop_words_filename

```
const std::string stop_words_filename = "stop_words.txt"
```

Definition at line 33 of file unit_tests.cpp.

Referenced by BOOST_AUTO_TEST_CASE(), and main().

2.4.3.7 stop_words_size

```
size_t stop_words_size
```

Definition at line 36 of file unit_tests.cpp.

Referenced by BOOST_AUTO_TEST_CASE(), and main().

2.4.3.8 text_filename

```
const std::string text_filename = "text1.txt"
```

Definition at line 26 of file unit_tests.cpp.

Referenced by BOOST_AUTO_TEST_CASE(), and main().

2.4.3.9 text_size

```
size_t text_size
```

Definition at line 31 of file unit_tests.cpp.

Referenced by BOOST_AUTO_TEST_CASE(), correct_text_hashes(), main(), and search_and_calculate_↩
matrices().

2.4.3.10 words_count

```
std::map<std::string, size_t> words_count
```

Definition at line 43 of file unit_tests.cpp.

Referenced by hash_function(), and main().

Index