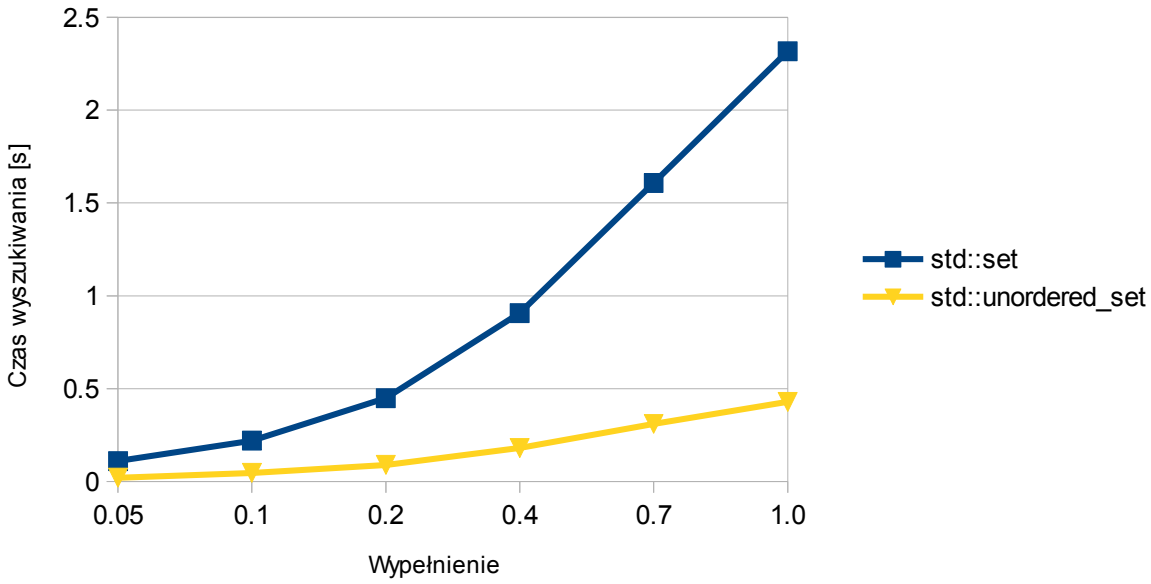
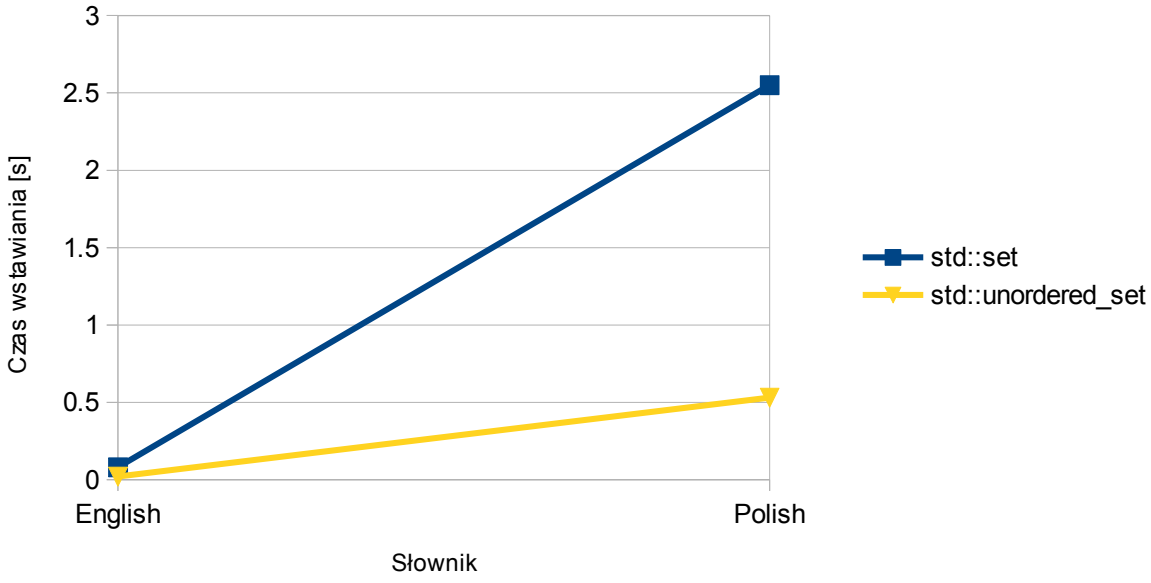
Zadanie 1


2.5
2
1.5
1
0.5
0
Czas wyszukiwania [s]
0.05
0.1
0.2
0.4
0.7
1.0
Wypełnienie
std::set
std::unordered_set


3
2.5
2
1.5
1
0.5
0
Czas wstawiania [s]
English
Polish
Słownik
std::set
std::unordered_set

```
// Marius Rejdak
// Informatyka, mgr, OS1

/*

Running:
#!/bin/fish
g++ -Wall  -std=c++11 -o3 -o zad1 zad1.cpp; and ./zad1

Output:
English to std::set 0.080332s
Polish to std::set 2.55033s
English to std::unordered_set 0.020802s
Polish to std::unordered_set 0.530011s
English searching in std::set 0.067151s
Polish searching in std::set 2.33132s
English searching in std::unordered_set 0.015167s
Polish searching in std::unordered_set 0.42904s
Polish searching in std::set for ratio 0.05 0.112123s
Polish searching in std::unordered_set for ratio 0.05 0.022894s
Polish searching in std::set for ratio 0.1 0.22494s
Polish searching in std::unordered_set for ratio 0.1 0.046557s
Polish searching in std::set for ratio 0.2 0.449548s
Polish searching in std::unordered_set for ratio 0.2 0.089799s
Polish searching in std::set for ratio 0.4 0.907567s
Polish searching in std::unordered_set for ratio 0.4 0.180477s
Polish searching in std::set for ratio 0.7 1.60857s
Polish searching in std::unordered_set for ratio 0.7 0.310871s
Polish searching in std::set for ratio 1 2.31883s
Polish searching in std::unordered_set for ratio 1 0.429663s

*/

#include <iostream>  // std::cout
#include <fstream>         // std::ifstream
#include <iterator>        // std::istream_iterator
#include <algorithm>  // std::sort, std::random_shuffle
#include <vector>   // std::vector
#include <string>          // std::string
#include <ctime>    // time, clock, clock_t, CLOCKS_PER_SEC
#include <cmath>           // std::round
#include <set>                    // std::set
#include <unordered_set> // std::unordered_set

template <class AssocContainer, class ForwardIterator>
float insert_time(AssocContainer &container, ForwardIterator iter_begin, ForwardIterator
iter_end)
{
       clock_t clockStart = clock();
       container.insert(iter_begin, iter_end);
       return ((float)clock()-clockStart)/CLOCKS_PER_SEC;
}

template <class AssocContainer, class Container>
float search_time(AssocContainer &container, Container &search_items)
{
       clock_t clockStart = clock();
       for(auto item : search_items)
       {
              container.find(item);
       }
```

```
	return ((float)clock()-clockStart)/CLOCKS_PER_SEC;
}

int main(int argc, char *argv[]) {
	std::ifstream in_english("english.dic", std::ifstream::in);
	std::ifstream in_polish("polish.dic", std::ifstream::in);
	std::istream_iterator<std::string> in_iter_english(in_english),
in_iter_polish(in_polish), eos;
	std::vector<std::string> lines_english(in_iter_english, eos);
	std::vector<std::string> lines_polish(in_iter_polish, eos);

	srand(unsigned(time(0)));
	std::random_shuffle(lines_english.begin(), lines_english.end());
	std::random_shuffle(lines_polish.begin(), lines_polish.end());

	std::set<std::string> set_polish, set_english;
	std::unordered_set<std::string> uset_polish, uset_english;

	std::cout << "English to std::set " << insert_time(set_english, lines_english.begin(),
lines_english.end()) << "s\n";
	std::cout << "Polish to std::set " << insert_time(set_polish, lines_polish.begin(),
lines_polish.end()) << "s\n";
	std::cout << "English to std::unordered_set " << insert_time(uset_english,
lines_english.begin(), lines_english.end()) << "s\n";
	std::cout << "Polish to std::unordered_set " << insert_time(uset_polish,
lines_polish.begin(), lines_polish.end()) << "s\n";

	std::cout << "English searching in std::set " << search_time(set_english, lines_english)
<< "s\n";
	std::cout << "Polish searching in std::set " << search_time(set_polish, lines_polish) <<
"s\n";
	std::cout << "English searching in std::unordered_set " << search_time(uset_english,
lines_english) << "s\n";
	std::cout << "Polish searching in std::unordered_set " << search_time(uset_polish,
lines_polish) << "s\n";

	float slices[] = {0.05, 0.1, 0.2, 0.4, 0.7, 1};
	std::vector<std::string> test_lines;

	for(float ratio : slices)
	{
		test_lines.insert(test_lines.end(), lines_polish.begin(), lines_polish.begin()+
(int)round(lines_polish.size()*ratio));
		std::cout << "Polish searching in std::set for ratio " << ratio << " " <<
search_time(set_polish, test_lines) << "s\n";
		std::cout << "Polish searching in std::unordered_set for ratio " << ratio << " " <<
search_time(uset_polish, test_lines) << "s\n";
		test_lines.clear();
	}

	return 0; //Huge success!
}
```

Zadanie 2

```
// Marius Rejdak
// Informatyka, mgr, OS1

/*

Running:
#!/bin/fish
g++ -Wall  -std=c++11 -o3 -o zad2 zad2.cpp
./zad2 polish.dic << cat text.txt

*/

#include <iostream>  // std::cout, std::cin
#include <fstream>          // std::ifstream
#include <iterator>         // std::istream_iterator
#include <string>           // std::string
#include <sstream>          // std::ostringstream
#include <vector>           // std::vector
#include <unordered_set> // std::unordered_set
#include <ctype.h>          // isupper, tolower

enum STRING_CASE {STRING_CASE_Lower, STRING_CASE_First_Upper, STRING_CASE_All_Upper,
STRING_CASE_Mixed};

enum STRING_CASE test_string_case(std::string &s)
{
        size_t num_upper_case = 0, num_lower_case = 0, i = 0;
        bool first_upper_case = false;

        for(auto c = s.begin(); c < s.end(); ++c, ++i)
        {
                if(isupper(*c))
                {
                        ++num_upper_case;
                        if(i == 0)
                        {
                                first_upper_case = true;
                        }
                }
                else
                {
                        ++num_lower_case;
                }
        }

        if(num_lower_case == s.size())
                return STRING_CASE_Lower;
        else if(first_upper_case && num_upper_case == 1)
                return STRING_CASE_First_Upper;
        else if(num_upper_case == s.size())
                return STRING_CASE_All_Upper;
        else
                return STRING_CASE_Mixed;
}

void string_to_lower(std::string &s, size_t first = 0)
{
        for(auto c = s.begin()+first; c < s.end(); ++c)
        {
```

```
                *c = tolower(*c);
        }
}

bool spellchecker(std::string &word, std::unordered_set<std::string> &dictionary)
{
        enum STRING_CASE word_case = test_string_case(word);
        std::string word_temp(word);
        std::unordered_set<std::string> possible_words;

        switch(word_case)
        {
        case STRING_CASE_All_Upper:
                possible_words.insert(word);
                string_to_lower(word_temp, 1);
        case STRING_CASE_First_Upper:
                possible_words.insert(word_temp);
                string_to_lower(word_temp);
        case STRING_CASE_Lower:
                possible_words.insert(word_temp);
                break;
        case STRING_CASE_Mixed:
        default:
                possible_words.insert(word);
        }

        for(auto test : possible_words)
        {
                if(dictionary.find(test) != dictionary.end())
                        return true;
        }
        return false;
}

std::vector<std::string> get_words()
{
        std::vector<std::string> words;
        std::string word;
        while(std::cin >> word)
        {
                words.push_back(word);
        }
        return words;
}

std::string set_color(int color)
{
        std::ostringstream stream;
        stream << "\033[" << color << "m";
        return stream.str();
}

std::string print_with_context(std::vector<std::string>::iterator error,
std::vector<std::string>::iterator begin, std::vector<std::string>::iterator end)
{
        std::ostringstream stream;
        for(auto i = error-2; i >= begin && i < error; ++i)
        {
                stream << *i << " ";
        }
        stream << set_color(31) << *error << set_color(0) << " ";
```

```
        for(auto i = error+1; i <= error+2 && i < end; ++i)
        {
                stream << *i << " ";
        }
        stream << std::endl;
        return stream.str();
}

int main(int argc, char *argv[])
{
        if(argc < 2)
                return 1;
        std::ifstream in_file(argv[1], std::ifstream::in);
        std::istream_iterator<std::string> in_iter_file(in_file), in_iter_eof;
        std::unordered_set<std::string> uset_words(in_iter_file, in_iter_eof);
        std::vector<std::string> text = get_words();

        std::cout << "Found errors:" << std::endl;
        for(auto word_iter = text.begin(); word_iter < text.end(); ++word_iter)
        {
                if(!spellchecker(*word_iter, uset_words))
                {
                        std::cout << print_with_context(word_iter, text.begin(), text.end());
                }
        }

        return 0; //Huge success!
}
```