Лабораторная работа № 4 по курсу дискретного анализа: поиск образца в тексте

Выполнил студент группы М80-208Б-18 МАИ *Синявский Андрей.*

Условие

1. Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.

2. Вариант 3-1.

   Апостолико-Джанкарло.

   Вариант алфавита: Слова не более 16 знаков латинского алфавита (регистронезависимые).

Метод решения

1. Препроцессинг "по учебнику где N-функция вычисляется через Z-функцию, а позиции слов-символов для правила плохого символа будут храниться в unordered_map, где ключ - слово, а значение - вектор со вхождениями этого слова в паттерн. Таким образом поиск слова будет работать за константу.

2. Фазы поиска/сдвига. В начале каждой фазы просматривается элемент вектора M под номером текущей позиции в тексте при поиске. Путём сравнения значений вектора M со значениями N-функции, а также изменением элементов из M при частичном совпадении паттерна с текстом, некоторые сравнения пропускаются. В случае обнаружения (не)совпадения паттерн сдвигается по правилам из алгоритма Бойера-Мура.

Описание программы

Каждое слово содержится в объекте класса TLetter, который по сути является обёрткой над std::string. Слова из паттерна и текста хранятся в векторах из TLetter. Весь препроцессинг выведен в отдельный класс TPreprocessor, векторы, получающиеся при обработке паттерна (для правила хорошего суффикса, N-функции и т.д.) доступны как публичные поля препроцессора. Поиск разбит на 3 функции: AD_search, которая отвечает за пропуск лишних сравнений по алгоритму Апостолико-Джанкарло, в случае несовпадения вызывает функцию boyer_move, сдвигающую паттерн по правилам Бойера-Мура, а в случае совпадения кроме boyer_move ещё и функцию entry, отвечающую за корректный вывод информации о вхождении паттерна в текст.

Класс	Значение
class TLetter	Обёртка над std::string для красивого представления текста
class TPreprocessor	Что-то вроде очень своеобразного функтора, который принимает паттерн, производит с ним все необходимые для препроцессинга операции, и хранит у себя их результат
Функция	Значение
void entry(...)	Функция для вывода вхождения
void boyer_move(...)	Служит для сдвига паттерна вдоль текста
void AD_search(...)	Для поиска паттерна с использованием функции boyer_move
int main()	Главная функция, в которой происходит чтение данных и вызов функции поиска.

Дневник отладки

При создании этой таблицы была использована история посылок.

Время	Проблема	Описание
2020/03/02 21:26:07	Ошибка выполнения	Ошибка,связанная с выходом за границу вектора PrefSuff, которую долго не удавалось выявить на тестах.
2020/03/11 22:46:35	Превышено реальное время работы	Забыл прописать ключ оптимизации в мейкфайле.
2020/03/14 13:50:33	Работает	Через три дня бессмысленных попыток оптимизировать код самому додумался спросить у преподавателя, в чём может быть проблема, и он указал на проблему выше. Теперь всё работает.

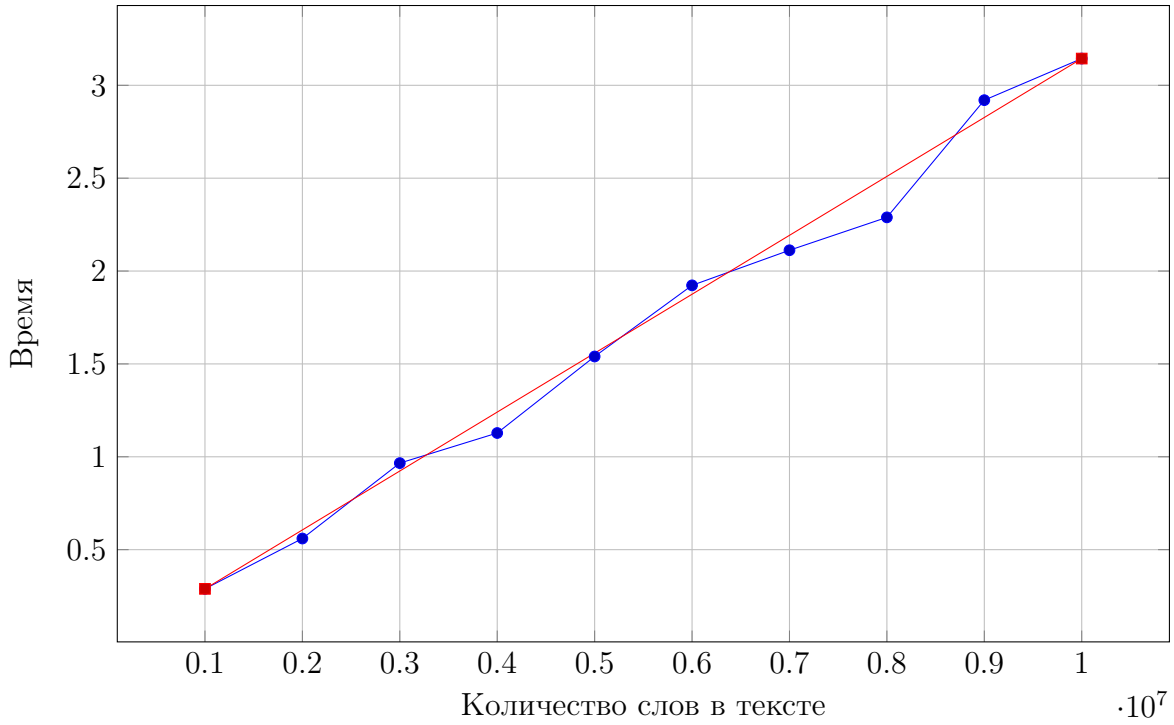
Тест производительности

Для генерации тестов использовалась следующая программа:

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>

int main(int argc, char **argv) {
```

```
    size_t numWords = std::atoi(argv[1]);
    std::string name = argv[2];
    std::ofstream os(name);
    srand(time(0));
    for (size_t i = 0; i < 5; i++) {
        for (size_t symb = 0; symb < 1; symb++) {
            os << static_cast<char>(rand() % 5 + 97);
        }
        if (i != 4) {
            os << ' ';
        }
    }
    os << "\n";
    for (size_t cnt = 0; cnt < numWords; cnt++) {
        for (size_t symb = 0; symb < 1; symb++) {
            os << static_cast<char>(rand() % 5 + 97);
        }
        if ((cnt + 1) % 10 == 0) {
            os << "\n";
        } else {
            os << " ";
        }
    }
    return 0;
}
```



Недочёты

Я вижу 2 недочёта в своей программе: 1) ужасно написанный метод нахождения N-функции, который по 10 раз переворачивает вектора. 2) реализация препроцессинга через класс, которая не делает особого смысла, может запутать читающего код, и наверняка нарушает какие-то принципы ООП.

Выводы

Проделав работу, я реализовал не только заданный в варианте алгоритм Апостолико-Джанкарло, но и алгоритм Бойера-Мура, так как первый основан на втором, а отладить сначала более простой алгоритм я посчитал наиболее логичным. Вообще при выполнении работы, кроме улучшения навыков написания алгоритмов, я научился лучше отлаживать свой код. Например, открыл для себя возможность сравнить скорость и результаты своего алгоритма с наивным его вариантом, обычно работающим значительно медленнее, но в реализации которого сложнее налажать.