Лабораторная работа № 4 по курсу дискретного анализа: Строковые алгоритмы

Условие

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

Вариант алгоритма:
Поиск одного образца при помощи алгоритма Апостолико-Джанкарло.
Вариант алфавита:
Слова не более 16 знаков латинского алфавита (регистронезависимые).

Метод решения

Выполнение работы можно разделить на 2 основных этапа:

- Реализация алгоритма Бойера-Мура, на его основе будет работать программа
- Надстройка над алгоритмом для избавления от лишних сравнений

Суть алгоритма заключается в том, что когда мы сдвигаемся по правилу хорошего суффикса, то мы гарантируем себе, что эта часть текста совпадает с текстом, но при этом далее всё равно будем это проверять. Чтобы избавиться от лишних проверок, создадим вспомогательный массив M, который и будет сообщать нам о том, что какой-то кусок текста можно не проверять повторно.

Описание программ

К сожалению, тестирующая система не позволила использовать модульный подход к решению поставленной задачи, поэтому весь рабочий код размещён в единственном файле *main.c*. Можно выделить осовные функции программы:

- FillBadCharTable — заполняет таблицу плохих символов [1]
- ZFunction — считает Z-функцию строки
- CreateGoodSuffTable — заполняет массив сдвигов по правилу хорошего суффикса

Далее на основе заполненных таблиц вычисляется сдвиг паттерна по тексту, при очередном несовпадении (функция *GetShiftBM*). Далее выполняется алгоритм Апостолико-Джанкарло с заполнением массива M, который и помогает сократить количество проверок.

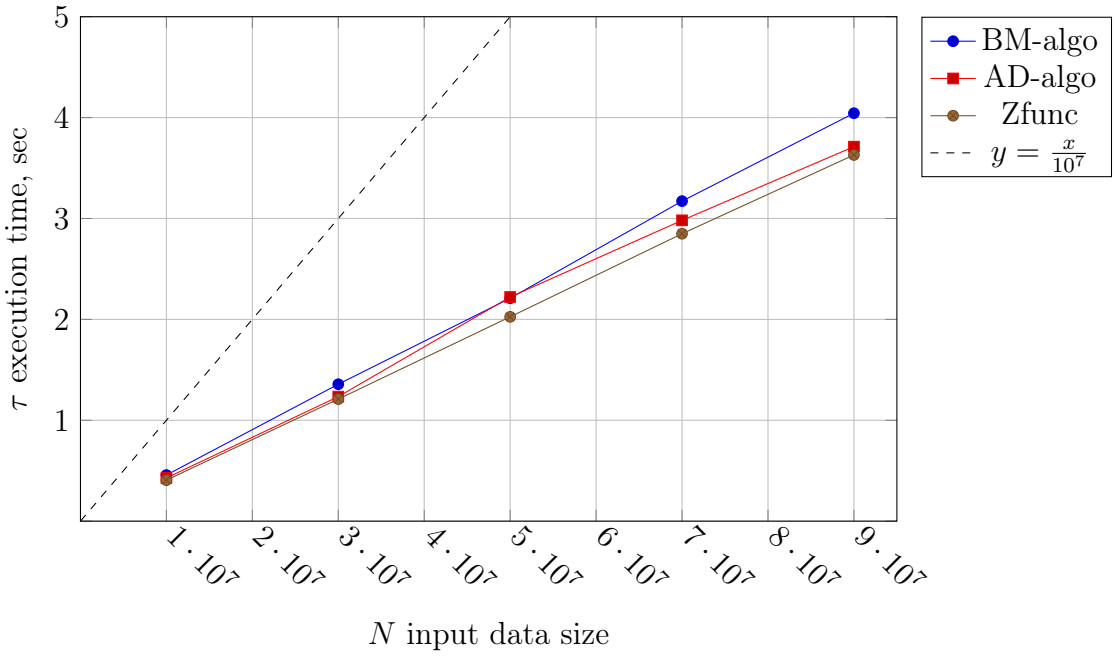
[1] Для реализации таблицы плохих символом используется B-дерево из 2-ой лабораторной работы

Дневник отладки

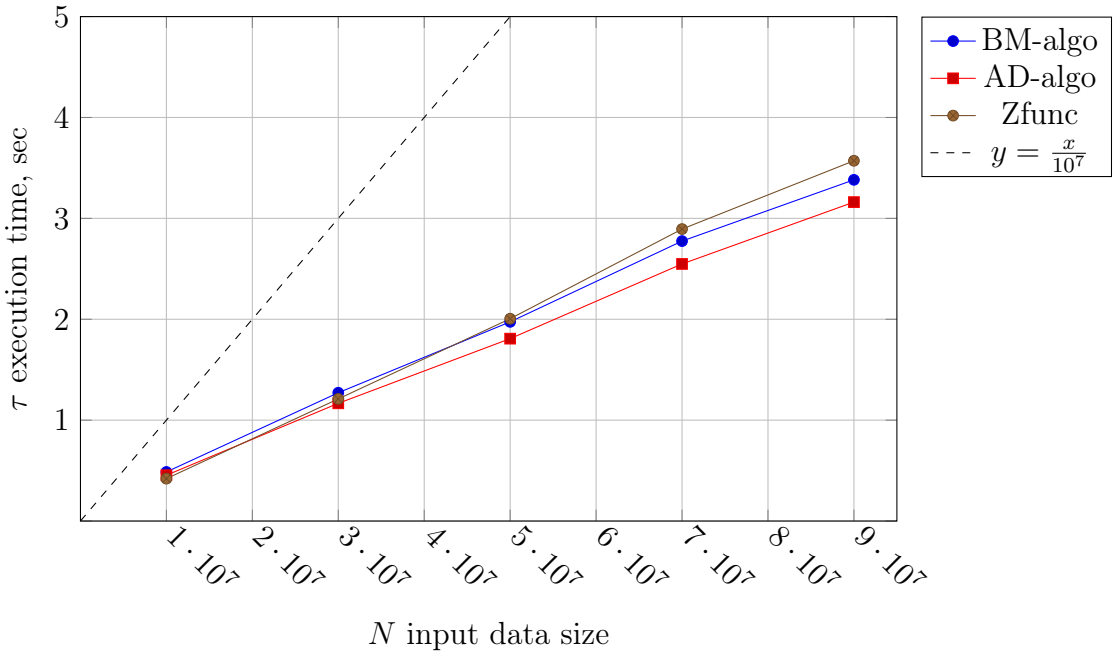
№	Описание ошибки	Способ устранения
1	WA-9 не находились все вхождения паттерна, если текст состоял из одинаковых букв	Эта ошибка была скорее из-за невнимательности, в месте, где выбирался сдвиг по алгоритму, нужно было проверять не $\neq$, а именно $<$
2	WA-5 (вар 4 – 1) длина паттерна вычислялась некорректно	Нужно было присвоить в переменную длинны паттрена соответствующее поле вектора, а не значение, которое возвращает функция чтения строки (т.к. она возвращает длину последнего слова.)

Тест производительности

Длина паттерна во много раз меньше текста:



Длина паттерна соизмерима с текстом:



Выводы

Во время выполнения лабораторной работы вспомнил основные понятия, связанные со строками, изучил такие алгоритмы как: Бойера—Мура, Апостолико—Джанкарло, Z-блоки, Ахо-Корасик. Сравнил производительность некоторых из них. Видно, что эти алгоритмы (кроме последнего, он не проверялся на скорость работы) имеют линейную сложность. Ещё из графиков можно сделать вывод, что время алгоритмов практически одинаково на случайных данных. То есть с точки зрение времени выполнения кода их можно считать идентичными, что нельзя сказать про время препроцессинга и расхода памяти. Касательно алгоритма Ахо-Корасик, он более подходит для поиска множества образцов, в отличие от перечисленных алгоритмов. Данный алгоритм можно сравнить с суффиксными деревьями, но только здесь в боре хранятся паттерны, а в суффиксном дереве весь текст целиком. И то, и то имеет свои плюсы, поэтому существуют оба этих алгоритма.