Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Дискретный анализ»

Студент: Т. Д. Голубев
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22
Дата:
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №4

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

Вариант алгоритма: Поиск одного образца при помощи алгоритма Кнута-Морриса-Пратта.

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

Запрещается реализовывать алгоритмы на алфавитах меньшей размерности, чем указано в задании.

1 Описание

Требуется написать реализацию алгоритма поиска подстроки в строке Кнута-Морриса-Пратта.

Как сказано в [1]: Предположим, что при некотором выравнивании P около T наивный алгоритм обнаружил совпадение первых i символов из P с их парами из T, а при следующем сравнении было несовпадение. В этом случае наивный алгоритм сдвинет P на одно место и начнет сравнение заново с левого конца P. Но часто можно сдвинуть образец дальше. Например, если P = abcxabcde и при текущем расположении P и T несовпадение нашлось в позиции 8 строки P, то есть возможность сдвинуть P на четыре места без пропуска вхождений P в T. Отметим, что это можно увидеть, ничего не зная о тексте T и расположении P относительно T. Требуется только место несовпадения в P. Алгоритм Кнута-Морриса-Пратта, основываясь на таком способе рассуждений, и делает сдвиг больше, чем наивный алгоритм.

2 Исходный код

Функция *ZFunction* реализует Z-алгоритм (Z[i] – длина наибольшей подстроки, начинающейся в i и совпадающей с префиксом строки).

Функция *SPFunction* реализует префикс-функцию (sp[i] – длина наибольшего собственного суффикса, который совпадает с префиксом строки).

Функция *KMPFunction* реализует алгоритм Кнута-Морисса-Пратта.

Для реализации алфавита в виде слов используется структура *TWord*. Для этой струтуры перегружены операторы «==» и «!=».

kmp.h	
std::vector<int> ZFunction(const TString& s)	Z-алгоритм.
std::vector<int> SPFunction(const TString& s)	Префикс-функция.
std::vector<int> KMPFunction(const TString& pattern, const TString& text)	КМП-алгоритм.

```
1  struct TWord {
2    char word[MAX_WORD_LEN];
3    int lineId, wordId;
4
5    TWord();
6    void Clear();
7    char& operator[](int index);
8    bool operator==(const TWord& rhs) const;
9    bool operator!=(const TWord& rhs) const;
10 };
11
12 using TString = std::vector<TWord>;
```

3 Консоль

```
cat_mood@nuclear-box:~/programming/mai-da-labs/lab04/build$ ./lab04_exe <test.txt
2,1
5,1
12,1
18,1
23,1
30,2
35,2
38,1
44,2
51,1
57,1
62,9
62,16
62,22
62,32
62,39
62,45
62,55
```

4 Тест производительности

Тест производительности представляет из себя следующее: КМП сравнивается с наивным алгоритмом на 3 тестах с паттерном длиной в 10 символов и текстом длиной в 10^5, входные данные из себя представляют случайный набор букв "a"и "b".

```
cat_mood@nuclear-box:~/programming/mai-da-labs/lab04/build$ ./lab04_benchmark
<../tests/e2e/test01.txt
Naive: 6026 ms
KMP: 3682 ms
cat_mood@nuclear-box:~/programming/mai-da-labs/lab04/build$ ./lab04_benchmark
<../tests/e2e/test02.txt
Naive: 4915 ms
KMP: 3953 ms
cat_mood@nuclear-box:~/programming/mai-da-labs/lab04/build$ ./lab04_benchmark
<../tests/e2e/test03.txt
Naive: 5539 ms
KMP: 3542 ms
```

Как видно, КМП в среднем работает быстрее наивный алгоритм.

Это связано с тем, что наивный алгоритм работает в среднем за $O(p \cdot (t - p))$, где p – длина паттерна, а t – длина текста.

Сложность КМП в среднем равна $O(p+t)$, где p – длина паттерна, а t – длина текста.

5 Выводы

Выполнив четвёртую лабораторную работу по курсу «Дискретный анализ», я реализовал алгоритм Кнута-Морриса-Пратта. В ходе работы столкнулся с проблемой хранения текста в памяти, так как в работе наложено жёсткое ограничение на память. Я решил эту проблему выполнением алгоритма в «реальном времени».

Список литературы

[1] Гасфилд Дэн Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология / Пер. с англ. И.В.Романовского. – СПб.: Невский Диалект; БХВ-Петербург, 2003. – 654 с.: ил.