Лабораторная работа № 4 по курсу дискретного анализа: строковые алгоритмы

Выполнил студент группы М8О-207Б-21 Меркулов Фёдор.

Условие

Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.
Вариант алгоритма: Поиск одного образца при помощи алгоритма Апостолико-Джанкарло.
Вариант алфавита: Числа в диапазоне от 0 до $2^{32} - 1$.

Метод решения

Я познакомился с алгоритмом Апостолико-Джанкарло, используя материалы лекции и книгу Дэн Гасфилда “Строки, деревья и последовательности в алгоритмах”.

Моё решение поставленной задачи состоит из частей:

1. Подсчёт функции для расширенного правила плохого символа
2. Подсчёт функции для правила хорошего суффикса
3. Применение алгоритма Апостолико-Джанкарло с использованием уже известных функций

Расширенное правило плохого символа:

Когда несовпадение случилось в позиции i образца и x - несовпадающий символ в тексте, нужно сдвинуть образец вправо, совместив с этим x ближайшее вхождение x в образце слева от позиции i

Препроцессинг должен найти для каждой позиции i в образце и каждого символа алфавита x позицию ближайшего появления x в образце слева от i. Очевидный подход состоит в создании для этой информации двумерного массива размера $x \times |\Sigma|$. Списки позиций, где x входит в образец, для всякого x из алфавита Σ накапливаются за время $O(n)$ и занимают память только $O(n)$.

Правило хорошего суффикса:

Пусть строка образца приложена к тексту, и подстрока t из текста совпадает с суффиксом образца, но следующий левый символ уже не совпадает. Найдём, если она существует правую копию t' строки t в образце, такую что t' не является суффиксом образца. Сдвинем образец вправо, приложив подстроку t' в образце к подстроке t в тексте. Если t' не существует, то сдвинем левый конец образца за левый конец t в тексте на наименьший сдвиг, при котором префикс сдвинутого образца совпал бы с суффиксом t в тексте. Если такого сдвига не существует, то сдвинем образец на наименьший сдвиг, при котором собственный префикс сдвинутого образца совпадает с суффиксом вхождения образца в текст. Если такой сдвиг невозможен, нужно сдвинуть образец на n мест, т. е. сдвинуть образец за t в тексте.

Препроцессинг для правила хорошего суффикса:

Пусть для каждого i: $L(i)$ - наибольшая позиция, меньшая n и такая, что строка $P[i..n]$(Суффикс образца начиная с позиции i) совпадает с суффиксом строки $P[1..L(i)]$. Если такой позиции нет, $L(i)$ считается равным 0.

$L(i)$ определяет позицию правого конца крайней правой копии $P[i..n]$, которая сама не является суффиксом образца.

Препроцессинг вычисляет $L(i)$ для каждой позиции i в образце. Это делается за время $O(n)$.

Алгоритм Апостолико-Джанкарло:

Апостолико и Джанкарло предложили вариант алгоритма Бойера-Мура, который допускает замечательное простое доказательство линейной оценка наихудшего времени счёта. В этом варианте никакой символ из текста не участвует в сравнениях после его первого совпадения с каким-нибудь символом из образца. Отсюда немедленно следует, что число сравнений не превзойдёт $2m$ (m - длина текста). Каждое сравнение даёт либо совпадение, либо несовпадение; последних может быть только m, так как при каждом несовпадении происходит ненулевой сдвиг образца, а совпадений - не больше m, так как никакой символ текста не сравнится после совпадения с символом из образца.

В алгоритме Бойера-Мура нужно сделать две модификации:

- При поиске образца в тексте (уже после препроцессинга) мы будем работать с вектором M длины m и на каждой фазе сравнения/сдвига будем модифицировать не более одного элемента.
- Используя векторы N ($N_j(P)$ - длина наибольшего суффикса подстроки $P[1..j]$, который является также суффиксом полной строки P) и M, ускоряет алгоритм Бойера-Мура за счёт пропуска некоторых проверок.

Чтобы прояснить саму идею, предположим, что алгоритм собирается сравнить символы $P(i)$ и $T(h)$ (T - текст), и допустим, что $M(h) > N_i$. Это означает, что подстрока P длины N_i кончается в позиции i и совпадает с суффиксом P. Значит, суффиксы этих подстрок длины N_i должны совпадать, и мы можем заключить, что следующие N_i сравнений (от $P(i)$ и $T(h)$ влево) алгоритма Бойера-Мура дадут совпадение. Далее, если $N_i = i$, то нашлось вхождение P в T, а если $Ni < i$, то мы можем быть уверены, что следующее сравнение (после N_i совпадений) даст несовпадение. Следовательно, в имитации Бойера-Мура, если $M(h) > N_i$, мы можем избежать по меньшей мере N_i явных сравнений.

Алгоритм Апостолико-Джанкарло выполняет не более $2m$ сравнений символов и не более $O(m)$ дополнительных действий.

Описание программы

Считывание происходит почисленное, а не посимвольное, что позволяет самостоятельно не убирать незначащие нули и не углубляться какой способом реализовано разделение между числами (табуляция или некоторое количество пробелов, всё равно перед таким подходом)

Для решения поставленной задачи используются 2 вектора. Первый: $pattern$ - вектор, хранящий тип $unsigned\ long\ long\ int$ (чтобы числа от 0 до $2^{32}-1$ смогли корректно считаться) нужен для считывания паттерна. Второй: $text$ – вектор, векторов, хранящий тип $unsigned\ long\ long\ int$ нужен для считывания текста и отслеживанием за индексом каждого числа в строке и за номером строки, на которой располагается число в тексте.
Далее применяется алгоритм Апостолико-Джанкарло в который передаются оба вектора и таким образом находятся все вхождения паттерна в текст, а также выводятся строки в которых они находятся и соответствующие позиции в этих строках.

Дневник отладки

Самая первая ошибка, что я по привычке в самом конце вывел “\n”, что привело к $WA1$

Далее я понял, что у меня неправильно совершалась работа с массивом $skip$, $skip[i+j]$ менял значение, хотя при совпадении длин у паттерна и текста получалось, что $i+j=-1$ и происходило обращение к несуществующей позиции, в связи с чем я получил $RE5$

Я считал, что нужно вывести индекс текста, а не строки в которой находилось найденное число (начало встречи паттерна) из-за чего получил ошибку $WA5$

Я решил просто обойти ситуацию, которая приводила к $RE5$ и прописал условие, чтобы $i+j$ были не меньше 0, но вследствие чего выяснил, что переопределение $skip[i+j]$ в принципе было неверной идеей, так как паттерн просто пропускал не все, но множество случаев, когда он встречался в тексте из-за неправильных сдвигов. Из-за чего был получен $WA9$

Тест производительности

Время работы алгоритма я решил проверить с помощью утилиты time. Входные данные: “1.txt”, “2.txt”, “3.txt”, “4.txt”, “5.txt”, “6.txt”, “7.txt”, “8.txt”, “9.txt”. Данные файлы представляют собой случайно сгенерированные числа с переносами строк, цифра N в названии которых говорит о количестве строк во входном файле, соответственно равном N*10^6.

```
papik@papik-VirtualBox:~/DA4$ time ./main < 1.txt

real    0m2,155s

user    0m1,684s

sys     0m0,426s

papik@papik-VirtualBox:~/DA4$ time ./main < 2.txt

real    0m4,242s

user    0m3,490s

sys     0m0,740s

papik@papik-VirtualBox:~/DA4$ time ./main < 3.txt

real    0m5,005s

user    0m4,199s

sys     0m0,736s

papik@papik-VirtualBox:~/DA4$ time ./main < 4.txt
```

real 0m6,882s

user 0m5,738s

sys 0m1,126s

papik@papik-VirtualBox:~/DA4$ time ./main < 5.txt

real 0m7,660s

user 0m6,459s

sys 0m1,187s

papik@papik-VirtualBox:~/DA4$ time ./main < 6.txt

real 0m7,885s

user 0m6,626s

sys 0m1,242s

papik@papik-VirtualBox:~/DA4$ time ./main < 7.txt

real 0m10,364s

user 0m8,938s

sys 0m1,370s

papik@papik-VirtualBox:~/DA4$ time ./main < 8.txt

real 0m9,848s

user 0m8,144s

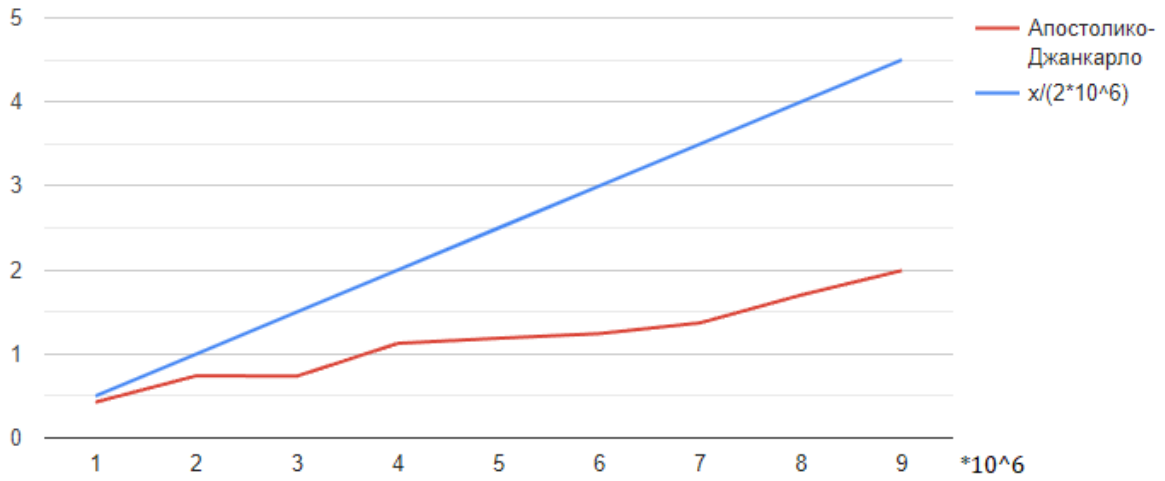
sys 0m1,701s

papik@papik-VirtualBox:~/DA4$ time ./main < 9.txt

real 0m14,675s

user 0m12,614s

sys 0m1,993s



Согласно данным измерениям, можно сказать, что с ростом количества входных данных время работы согласуется с заявленной сложностью, то есть растёт линейно.

Выводы

Был рассмотрен алгоритм Апостолико-Джанкарло. Он позволяет находить все вхождения заданной подстроки в строку за линейное время. Когда уже знаешь историю очень трудного и неполного анализа алгоритма Бойера-Мура, поражаешься тому, что близкий вариант этого алгоритма имеет простую линейную оценку времени.

В процессе выполнения лабораторной работы была изучена соответствующая теория, а также была разработана программа на C++, которая реализует данный алгоритм.

Результаты лабораторной работы свидетельствуют о высокой эффективности алгоритма Апостолико-Джанкарло при поиске подстроки в строке.