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

Выполнил студент группы 08-208 МАИ *Скворцов Александр.*

Условие

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

2. Поиск одного образца при помощи алгоритма Апостолико-Джанкарло.

3. Алфавит: числа в диапазоне от 1 до $2^{32} - 1$.

Метод решения

Алгоритм Апостолико-Джанкарло — один из вариантов поиска Бойера-Мура, в котором совпадение ищется прикладыванием правого конца образца к тексту и сравнением слева-направо. Сдвиги всё также осуществляется по правилу хорошего суффикса и плохого символа. Основное отличие заключается в хранении дополнительного массива, длинной равной длине текста, который хранит информацию о количестве совпавших ранее символов. С его помощью легче доказывается линейность алгоритма, а также он позволяет пропустить часть сравнений, что делает его ещё и быстрее Боера-Мура.
Таким образом, для реализации алгоритма необходимо:

1. Массив R, равный мощности алфавита, где $R(i)$ — максимальная правая позиция символа i в P.

2. Массив N, где $N(i)$ — длина наибольшего суффикса $P[1..i]$, который является также суффиксом P.

3. Массив L', где $L'(i)$ — наибольшая позиция, такая что $P[i..n]$ совпадает с суффиксом $P[1..L'(i)]$, а символ, предшествующий ему не равен $P(i-1)$.

4. Массив l', где $l'(i)$ — длина наибольшего суффикса $P[i..n]$, который является также префиксом P.

5. Массив M, равный длине текста, где $M(i)$ <= количества символов совпавших с суффиксом P.

Стоит отметить, что массивы L' и l' строятся на основе N, который в свою очередь является Z-функцией для реверсированного образца, а значит они строятся за линейное время. Также можно заметить, что алгоритм двигается только в одну сторону, поэтому для экономии памяти можно обрабатывать текст по мере поступления и хранить его, как и M, в массиве с длиной равной образцу.

Описание программы

Вся программа поместилась в одном файле и разделена на несколько функций:

- int main() — Вводит образец, а затем находит его вхождения в текст по мере ввода.
- void EnterPattern(std::vector<unsigned int> &pattern) — Вводит образец в pattern из стандартного ввода.
- void RealtimeSearchString(std::vector<unsigned int> &p) — Функция поиска подстроки в тексте, получаемом из стандартного потока.
- int max(int a, int b), int max(int a, int b, int c) — Нахоит максимальный из двух/трех элеметов соответственно и возвращает его значение.
- void BuildBCTable(int *bmBC, std::vector<unsigned int> &s) — Строит для s таблицу плохих символов R и записывает её в bmBC.
- void BuildSLTable(int *bmSL, std::vector<unsigned int> s) — Строит для s таблицу длин суффиксов N и записывает её в bmSL.
- void BuildGSTable(int *bmGS, int *bmSL, int veclen) — Строит для s таблицу хороших суффиксов L' с помощью с bmSL и записывает её в bmGS длиной veclen.
- void BuildPSTable(int *bmPS, int *bmSL, int veclen) — Строит для s таблицу префиксов, совпавших с суффиксами, l' с помощью с bmSL и записывает её в bmGS длиной veclen.

Дневник отладки

№	Ответ чекера	Причина ошибки
1,2	Compilation error	Компилиция без нужных ключей, вследствие чего некоторые предупреждения игнорирвались
3,4	Wrong answer at test 01.t	Программа выводила некоторые дополнительные данные
5,6	Wrong answer at test 04.t	Программа не пропускала некоторые пробельные символы, поэтому ввод был некорректным
7,8	Wrong answer at test 04.t	Программа не учитывала пустые строки
9-18	Runtime error at test 14.t, got signal 6	Причиной ошибки являлся недостаток памяти, не было учтено, что длина строки неограниченна. Для устранения проблемы было решено хранить текст в массиве с длиной образца, каждый раз сдвигая не только паттерн, но и текст.

Тест производительности

Проверим заявленную сложность алгоритма Апостолико-Джанкарло. Пусть каждый раз длина строки и образца увеличиваются в C раз. Тогда его время работы увеличится в C раз в соответствии со сложностью $O(n + m)$.

$\|pattern\|$	$\|text\|$	Время работы в мс
2000	10000	6
10000	50000	34
50000	250000	151
250000	1250000	521

Как видно из таблицы, при $C = 5$ время также увеличиватся в 5 или даже меньше раз. Также было произведено сравнение с наивным алгоритмом, но результаты не сильно отличаются. Это можно объяснить случайной генерацией чисел. При большой длине образца вероятность совпадения даже половины паттерна крайне мала.

Выводы

Как уже отмечалось ранее, алгоритм Апостолико-Джанкарло при поиске двигается только вправо, находит совпадения в порядке их расположения и никогда не возвращается назад. Поэтому его можно использовать для обнаружения подстроки в режиме реального времени. К тому же алгоритм можно оптимизировать, чтобы кол-во памяти было пропорционально длине образца. Наиболее эффективно его использовать на алфавитах большой мощности(тогда время поиска стремится к $O(m/n)$)) или в естественно-языковых текстах, где нет бессмысленных повторений, но эвристические правила все также позволяют пропустить ненужные сравнения, производящиеся в наивном алгоритме.