Лабораторная работа № 4 по курсу Дискретный Анализ. Строковые алгоритмы

Выполнил студент группы 08-207 МАИ *Павлов Иван.*

Условие

Кратко описывается 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: Числа в диапазоне от 0 до $2^{32} - 1$

Метод решения

Я познакомился с алгоритмом Кнута-Морриса-Пратта, используя материалы лекции и статьи в Википедии. Я выбрал реализацию алгоритма через Z-функцию, с помощью которой можно вычислить сильную префикс-функцию.

Алгоритм КМП состоит из 3 частей:

1. Нахождение Z-функции для паттерна.
2. Подсчет сильной префикс-функции для паттерна.
3. Проход по тексту и поиск вхождений, с ускорениями с помощью префикс-функции.

Алгоритм нахождения Z-функции за $O(n)$ основан на использовании уже вычисленной Z-функции для префиксов строки. При обработке каждого символа строки, мы поддерживаем два индекса l и r, которые задают границы самой правой подстроки, совпадающего с её префиксом.

1 Инициализируем $l = r = 0$ и $Z[0] = n$.

2 Для каждой позиции i в подстроке $pattern[2 \dots n]$:

2.1 Если $i > r$, то устанавливаем $l = r = i$ и ищем значение $Z[i]$ по определению.

2.2 Если $i <= r$, то мы можем использовать уже известное значение $z[k]$, где $k = i - l + 1$. Если $z[k] < r - i + 1$, то $z[i] = z[k]$, так как отрезок $pattern[l \dots r]$, совпадающий с префиксом $pattern[i \dots n]$, целиком содержится в отрезке $pattern[0 \dots (k-1)]$. Иначе мы должны пересчитать $Z[i]$ по определению, начиная с позиции $r + 1$.

3 При нахождении значения $Z[i]$, если $i + z[i] - 1 > r$, то мы обновляем значения l и r, так как найденный префикс $pattern[i \dots i + z[i] - 1]$ совпадает с более длинным префиксом $pattern[0 \dots z[i] - 1]$.

Алгоритм подсчета сильной префикс-функции, используя Z-функцию:

1. Инициализируем массив spi размером sz - длиной паттерна.
2. Перебираем элементы Z в обратном порядке.
3. Если значение $Z[i]$ не равно нулю, то мы знаем, что у нас есть подстрока, которая совпадает с началом строки.
4. Обновляем значение элемента $spi[i + Z[i] - 1]$ на $Z[i]$. Это происходит потому, что мы знаем, что подстрока, начинающаяся с позиции i и имеющая длину $Z[i]$, является суффиксом строки, начинающейся с позиции 0. Таким образом, мы можем заключить, что подстрока, начинающаяся с позиции $i+Z[i]-1$, является сильным префиксом строки.

Поиск вхождений:

1. Прикладываем паттерн к строке, сравниваем элементы.
2. Если элементы совпали, сдвигаем паттерн на 1.
3. Если элементы не совпали, сдвигаем паттерн так, чтобы его префикс совпал с суффиксом предыдущей позиции паттерна в строке. В этом нам поможет подсчитанная ранее сильная префикс-функция.
4. Если значение итератора совпало с длиной паттерна, то вхождение найдено.

Сложность алгоритма Кнута-Морриса-Пратта составляет $O(m + n)$, где m - длина строки, n - длина подстроки.

Описание программы

Из-за ограничений по памяти пришлось совместить ввод текста и поиск вхождений.

Для начала я считал паттерн до переноса строки. На каждой итерации цикла считывается очередной символ и проверяется его значение. Если символ является цифрой от 0 до 9, то он добавляется к текущему числу с помощью побитовых сдвигов. Чтобы преобразовать символ в соответствующее ему число, символ $'0'$ вычитается из него (это эквивалентно операции XOR с числом 48). Если же считанный символ не является цифрой, значит было считано целое число, которое нужно добавить в $pattern$. После этого текущее число обнуляется, и следующий считанный символ должен быть цифрой. Это достигается методом конечных автоматов.

Затем, используя приведенные выше алгоритмы, подсчитываем Z-функцию и сильную префикс-функцию.

После этого считаем текст до EOF, совмещая алгоритм Кнута-Морриса-Пратта и конечный автомат. Таким образом были найдены все вхождения паттерна прямо во время ввода текста и сэкономлена память.

Также из-за формата вывода в моем коде присутствует очередь q размера паттерна. Она нужна, чтобы вывести индексы первого совпадения, а не последнего, который находит КМП.

Дневник отладки

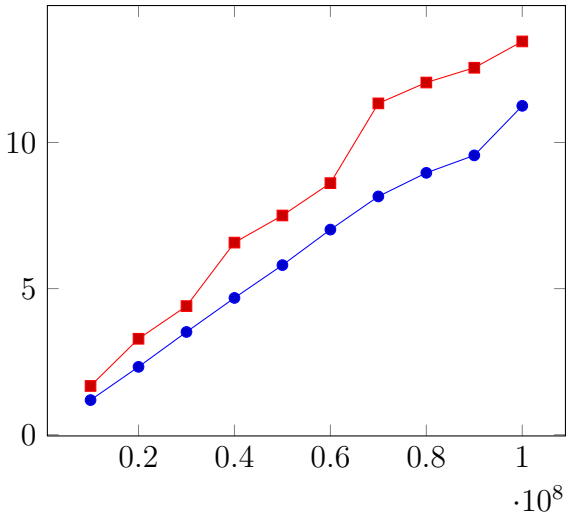
CE1 Случайно запустил код не на том компиляторе.

ML12 Держал весь текст в векторе, расходуя память. Переписал в виде, описанном выше.

WA5 Добавлял и удалял элементы из очереди после применения КМП на каждой итерации, из-за этого пропускал некоторые вхождения. Перенес операции с очередью повыше.

Тест производительности

Я решил замерить время работы КМП (отмечено синим) программы утилитой *time*, сгенерировав тест, в котором случайные числа от 1 до 50 в количестве от 10000000 до 100000000, и каждую 1000 чисел будет вхождение паттерна размером в 50 символов. Сравнивать буду с классическим алгоритмом (отмечено красным), основанным на Z-функции, который работает также за $O(n)$, но делает меньше сдвигов.



Видно, что за счет большего количества сдвигов алгоритм КМП работает быстрее, чем даже алгоритм за $O(n)$, который использует ту же Z-функцию.

Выводы

В ходе выполнения лабораторной работы были изучены различные алгоритмы поиска подстроки в строке и проведено сравнение их эффективности.

Был рассмотрен алгоритм Кнута-Морриса-Пратта, который основывается на использовании Z-функции. Он позволяет находить все вхождения заданной подстроки в строку за линейное время, что делает его одним из наиболее эффективных алгоритмов поиска подстроки.

В процессе выполнения лабораторной работы была изучена теория, лежащая в основе алгоритма Кнута-Морриса-Пратта, а также была разработана программа на C++, которая реализует данный алгоритм.

Результаты, полученные в ходе выполнения лабораторной работы, свидетельствуют о высокой эффективности алгоритма Кнута-Морриса-Пратта при поиске подстроки в строке.