Лабораторная работа № 4

Алгоритмы поиска подстрок

Цель работы: изучить основные принципы работы алгоритмов поиска подстрок, исследовать их свойства, закрепить навыки структурного программирования.

Общие сведения

Поиск подстрок является частным случаем общей задачи поиска. Тем не менее, именно с этой задачей нередко приходится сталкиваться прикладному программисту. «Стандартные» библиотеки многих языков высокого уровня предлагают те или иные способы решения данной задачи. Однако не секрет, что обобщенные решения не всегда дают требуемый результат в частных случаях. Поэтому программисту необходимо быть готовым к «изобретению велосипеда» для повышения производительности работы своего приложения.

Простейший алгоритм последовательного сравнения текста с образцом дает в худшем случае асимптотическую сложность $O((N - M + 1)M)$, где N – длина текста, M – длина образца (в среднем ситуация выглядит лучше, особенно для «случайного» текста). Существует большое количество алгоритмов, призванных повысить скорость поиска и минимизировать вероятность худшего случая: алгоритм Рабина-Карпа, метод конечных автоматов, алгоритм Кнута-Морриса-Пратта, алгоритм Бойера-Мура и т. д. Рассмотрим некоторые из них.

Алгоритм Рабина-Карпа. Идея этого алгоритма состоит в том, чтобы сократить количество полных проверок идентичности подстрок и образца, путем сопоставления им значений, вычисляемых за постоянное время с помощью некоторой функции (так называемой хеш-функции). Асимптотическая сложность работы такого алгоритма равна $O(N + M)$ в среднем и, к сожалению, $O((N - M + 1)M)$ в худшем. Стоит заметить, что вероятность худшего случая мала. Производительность алгоритма существенным образом зависит от скорости вычисления хеш-функции.

Алгоритм Кнута-Морриса-Пратта. Этот алгоритм основывается на предварительном создании префикс-функции (массива), ассоциированной с образцом и несущей информацию о том, где в образце повторно встречаются его же различные префиксы. Использование этой информации позволяет избежать проверки заведомо недопустимых сдвигов.

Если P – образец длиной M, то префикс-функция f: $\{0, 1, 2, \ldots, M - 1\} \rightarrow \{0, 1, 2, \ldots, M\text{-}1\}$ определяется следующим образом:

$f[q] = \max \{k : k < q$ и префикс P_k является суффиксом $P_q\}$,

т.е. $f[q]$ – длина наибольшего префикса P, являющегося собственным суффиксом P_q. Время предварительной подготовки пропорционально длине образца, время поиска составляет $O(N + M)$ как среднем, так и в худшем случаях.

Алгоритм Бойера-Мура. Для длинных образцов, состоящих из достаточно большого алфавита, возможно одним из лучших алгоритмов является алгоритм Бойера-Мура. Его структура аналогична структуре самого простого алгоритма, но предварительно вычисляются две вспомогательные функции (массива), значения которых используются для вычислений очередных значений индекса при

перемещении по тексту. Кроме того, в отличие от простого алгоритма сопоставление образца с текстом происходит справа налево.

Две вспомогательные функции представляют собой функцию стоп-символа и функцию безопасного суффикса. Их помощь заключается в следующем. Пусть при сравнении на некотором расстоянии образца с текстом выяснилось, что крайняя правая часть совпадает, а затем следует (налево) символ не соответствующий образцу. Функция стоп-символа предлагает сдвинуть образец вправо на такое расстояние, чтобы стоп-символ в тексте оказался напротив крайнего правого вхождения этого символа в образец. Если стоп символа в образце вообще нет, то образец необходимо полностью переместить за стоп-символ. Если стоп символ встречается в образце правее стоп-символа в тексте, то данная функция ничего хорошего не дает, так как, следуя ей, надо было бы сдвинуть образец влево!

С помощью функции безопасного сдвига можно сдвинуть образец вправо настолько, чтобы ближайшее вхождение (правее текущего) безопасного суффикса в образец оказалось напротив безопасного суффикса в тексте.

Алгоритм Боейра-Мура выбирает больший из двух возможных сдвигов. В худшем случае время работы этого алгоритма равно O((*N* – *M* + 1)*M* + *D*) (D – мощность алфавита), однако на практике он оказывается одним из самых эффективных.

Поразрядное «или» со сдвигом. Для образцов небольшой длины (длина не превышает разрядности слова вычислительной системы) достаточно эффективным является именно этот алгоритм. Этот алгоритм использует поразрядные операции. Пусть битовый вектор *R* имеет размер *M*. Вектор R_j – это значение вектора *R* после того, обработан символ *T*[*j*]. Он содержит информацию обо всех вхождениях префикса образца *P*, которые оканчиваются в позиции *j* в тексте *T* для $i \in [0, M-1]$:

$$R_j[i] = \begin{cases} 0, P[0,i] = T[j-i, j] \\ 1, P[0,i] \neq T[j-i, j] \end{cases} \quad (1)$$

Рекурсивно можно вычислить R_{j+1} зная R_j следующим образом. Для каждого $R_j[i] = 0$:

$$R_{j+1}[i] = \begin{cases} 0, P[i+1] = T[j+1] \\ 1, P[i+1] \neq T[j+1] \end{cases} \quad (2)$$

и

$$R_{j+1}[0] = \begin{cases} 0, P[0] = T[j+1] \\ 1, P[0] \neq T[j+1] \end{cases} \quad (3)$$

Если $R_{j+1}[M-1] = 0$, то образец найден. Для быстрого вычисления *R* необходимо для каждого символа алфавит *C* вычислить вспомогательный вектор S_C, (размера M) который будет содержать информацию о позиции данного символа в образце *P*: $S_C[i] = 0$, если $P[i] = C$, при $i \in [0, M-1]$. Зная S_C вычислять *R* можно за две операции: R_{j+1} = СДВИГ_ВЛЕВО (R_j) ИЛИ $SC_{T[j+1]}$

Асимптотическая сложность предварительной подготовки O(*M* + *D*), а собственно поиска – O(*N*).

Задание

1. Разработать функцию, реализующую любой из алгоритмов поиска подстрок.

2. Исследовать алгоритмы поиска реализованной функции и функции стандартной библиотеки `strstr`: построить и сравнить зависимости среднего времени выполнения от размера текста и размера образца. При исследовании алгоритмов обратить особое внимание на формирование текста, в котором осуществляется поиск.
3. Составить отчет, в котором привести графики полученных зависимостей, анализ свойств алгоритмов и выводы по работе.