Лабораторная работа №7
По курсу: "Анализ алгоритмов"
По теме: "Поиск подстроки в строке"

Студент: Кондрашова О.П.
Группа: ИУ7-55Б
Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Содержание

Введение

Целью данной лабораторной работы является изучение более эффективного способа нахождения поиска подстроки в строке, чем классическое сравнение каждого символа строки с подстрокой, в частности, алгоритма Кнута-Морриса-Пратта и алгоритма Бойера-Мура.

Задачи данной работы:

1. Изучение и описание алгоритмов.;
2. разработка и реализация алгоритмов;
3. тестирование полученного программного обеспечения.

1 Аналитическая часть

В данном разделе приведено описание алгоритмов поиска подстроки.

1.1 Задача поиска подстроки в строке

Пусть дана некоторая строка T (текст) и подстрока S (слово). Задача поиска подстроки сводится к поиску вхождения этой подстроки в указанной строке. Строго задача формулируется следующим образом: пусть задан массив T из N элементов и массив S из M элементов, $0 < M \leq N$. Если алгоритм поиска подстроки обнаруживает вхождение W в T, то возвращается индекс, указывающий на первое совпадение подстроки со строкой.

1.2 Стандартный алгоритм

Простейшим алгоритмом является примитивный алгоритм. Данный алгоритм выглядит следующим образом:

1. I = 1, J = 1
2. Сравнение T[I] с W[J]
3. Совпадение: J = J + 1, I = I +1
4. Несовпадение: J = 1, I = I + 1
5. Если J = M, то подстрока найдена
6. Если I + M > N, то подстрока отсутствует

В худшем случае сложность алгоритма равно $O(T \times S)$, в среднем – $O(T - S + 1)$.

1.3 Алгоритм Кнута-Морриса-Пратта

Алгоритм Кнута-Морриса-Пратта позволяет улучшить показатель количества сравнений: данный алгоритм требует только N сравнений в худшем случае. Идея алгоритма в том, что при каждом несовпадении T[I] и W[J] мы сдвигаемся не на единицу, а на J, так как меньшие сдвиги не приведут к полному совпадению. К сожалению, этот алгоритм поиска дает выигрыш только тогда, когда несовпадению предшествовало некоторое число совпадений, иначе алгоритм работает как примитивный. Так как совпадения встречаются реже, чем несовпадения, выигрыш в большинстве случаев незначителен.

Алгоритм Кнута-Морриса-Пратта основан на принципе конечного автомата. В этом алгоритме состояния помечаются символами, совпадение с которыми должно в данный момент произойти. Из каждого состояния имеется два перехода: один соответствует успешно-му сравнению, другой — несовпадению. Успешное сравнение переводит нас в следующий узел автомата, а в случае несовпадения мы попадаемв предыдущий узел, отвечающий образцу.

При всяком переходе по успешному сравнению в конечном автомате Кнута-Морриса-Пратта происходит выборка нового символа из текста. Переходы, отвечающие неудачному сравнению, не приводят к выборке нового символа; вместо этого они повторно используют последний выбранный символ. Если мы перешли в конечное состояние, то это означает, что искомая подстрока найдена.

Заметим, что при совпадении ничего особенного делать не надо: происходит переход к следующему узлу. Напротив, переходы по несовпадению определяются тем, как искомая подстрока соотносится сама с собой.

Метод КМП использует предобработку искомой строки, а именно: на ее основе создается префикс-функция. Префикс-функция от строки S и позиции i в ней — длина k наибольшего собственного (не равного всей подстроке) префикса подстроки $S[1..i]$, который одновременно является суффиксом этой подстроки. То есть, в начале подстроки $S[1..i]$ длины i нужно найти такой префикс максимальной длины $k < i$, который был бы суффиксом данной подстроки $S[1..k] = S[(i-k+1)..i]$.

Например, для строки "abcdabscabcdabia"префикс-функция будет такой:

[0 ,0 ,0 ,0 ,1 ,2 ,0 ,0 ,1 ,2 ,3 ,4 ,5 ,6 ,0 ,1].

Значения префикс-фукнции для каждого символа шаблона вычисляются перед началом поиска подстроки в строке и затем используются для сдвига.

Особенностью данного алгоритма является то, что он работает на основе автоматов.

Так, например, для нахождения в строке "abababcb"подстроки "ababcb мы построим следующий автомат 1, где состояния маркируются ожидаемыми символами:

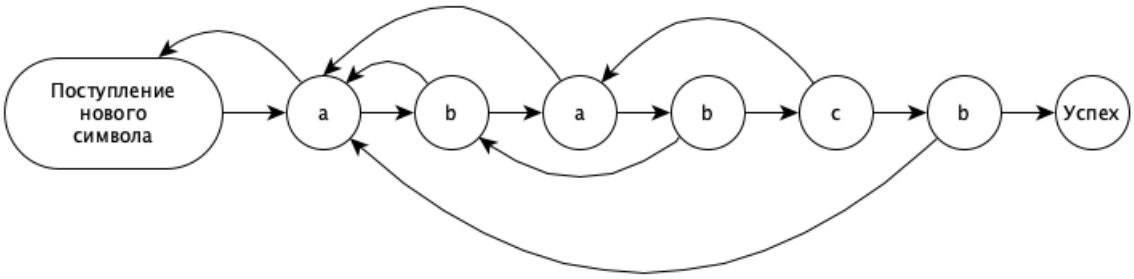

Рис. 1: Автомат

Также существует оптимизация алгоритма:

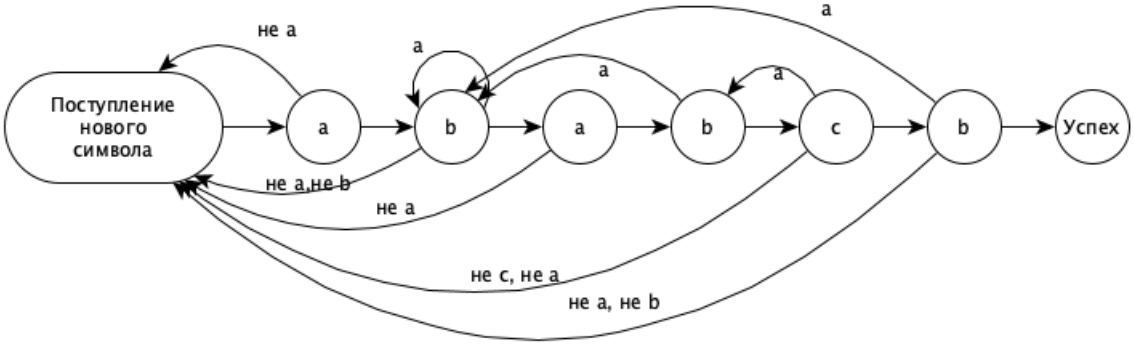

Рис. 2: Оптимизированный автомат

1.4 Алгоритм Бойера-Мура

Алгоритм поиска строки Бойера — Мура считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке.

Преимущество этого алгоритма в том, что ценой некоторого количества предварительных вычислений над шаблоном (но не над строкой, в которой ведётся поиск) шаблон сравнивается с исходным текстом не во всех позициях — часть проверок пропускаются как заведомо не дающие результата.

Идея БМ-поиска – сравнение символов начинается с конца образца, а не с начала, то есть сравнение отдельных символов происходит справа налево. Затем с помощью некоторой эвристической процедуры вычисляется величина сдвига вправо s. И снова производится сравнение символов, начиная с конца образца.

Простейший вариант алгоритма Бойера-Мура состоит из следующих шагов. На первом шаге мы строим таблицу смещений для искомого образца. Процесс построения таблицы будет описан ниже. Далее мы совмещаем начало строки и образца и начинаем проверку с последнего символа образца. Если последний символ образца и соответствующий ему при наложении символ строки не совпадают, образец сдвигается относительно строки на величину, полученную из таблицы смещений, и снова проводится сравнение, начиная с последнего символа образца. Если же символы совпадают, производится сравнение предпоследнего символа образца и т. д. Если все символы образца совпали с наложенными символами строки, значит мы нашли подстроку и поиск окончен. Если же какой-то (не последний) символ образца не совпадает с соответствующим символом строки, мы сдвигаем образец на один символ вправо и снова начинаем проверку с последнего символа. Весь алгоритм выполняется до тех пор, пока либо не будет найдено вхождение искомого образца, либо не будет достигнут конец строки.

Таблица смещений строится следующим образом. Каждому символу ставится в соответствие величина, равная разности длины шаблона и поряд-

кового номера символа (если символ повторяется, то берется самое правое вхождение).

Величина смещения для каждого символа образца зависит только от порядка символов в образце, поэтому смещения удобно вычислить заранее и хранить в виде одномерного массива, где каждому символу алфавита соответствует смещение относительно последнего символа образца.

1.5 Вывод

Были рассмотрены три различных алгоритма поиска подстроки в строке.

2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов.

2.1 Схемы алгоритмов

На рис. 3 представлена схема алгоритма Кнута-Морриса-Пратта:

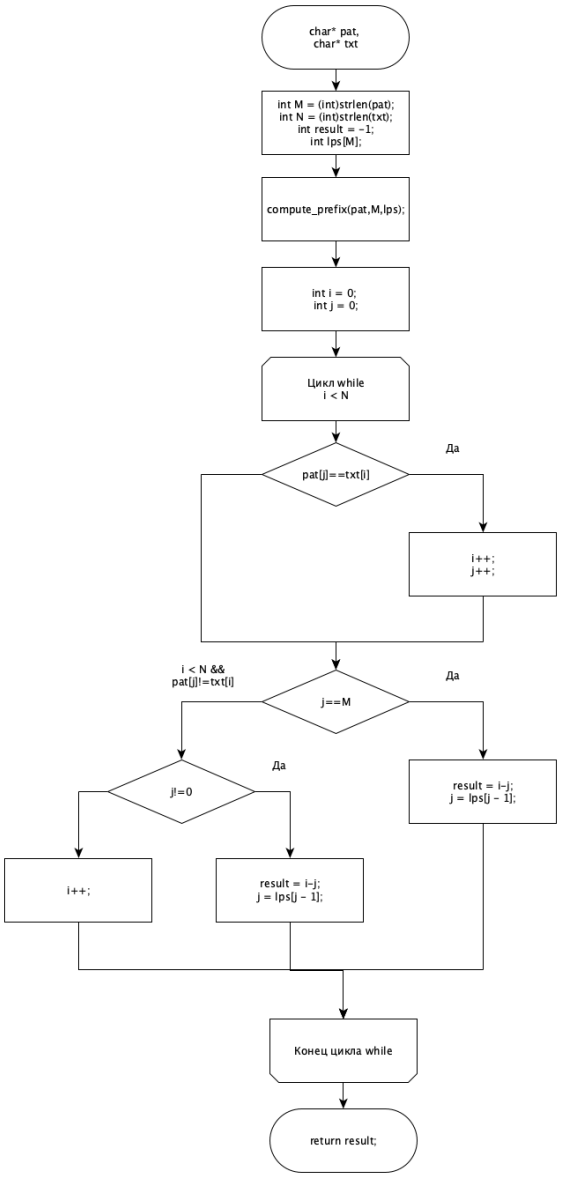

Рис. 3: Схема алгоритма Кнута-Морриса-Пратта

На рис. 4 представлена схема алгоритма нахождения префикса:

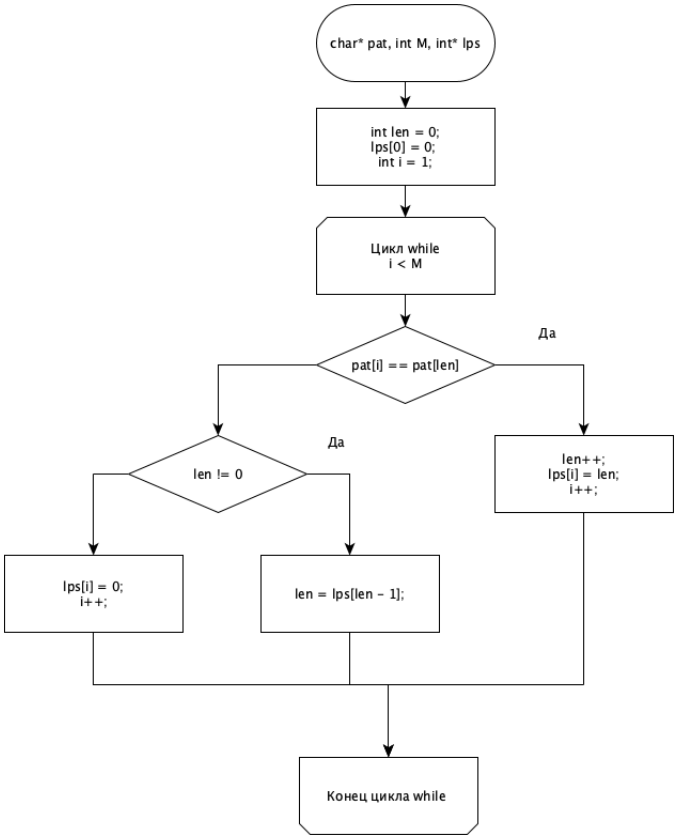

Рис. 4: Схема алгоритма нахождения префикса

На рис. 5 представлена схема алгоритма Бойера-Мура:

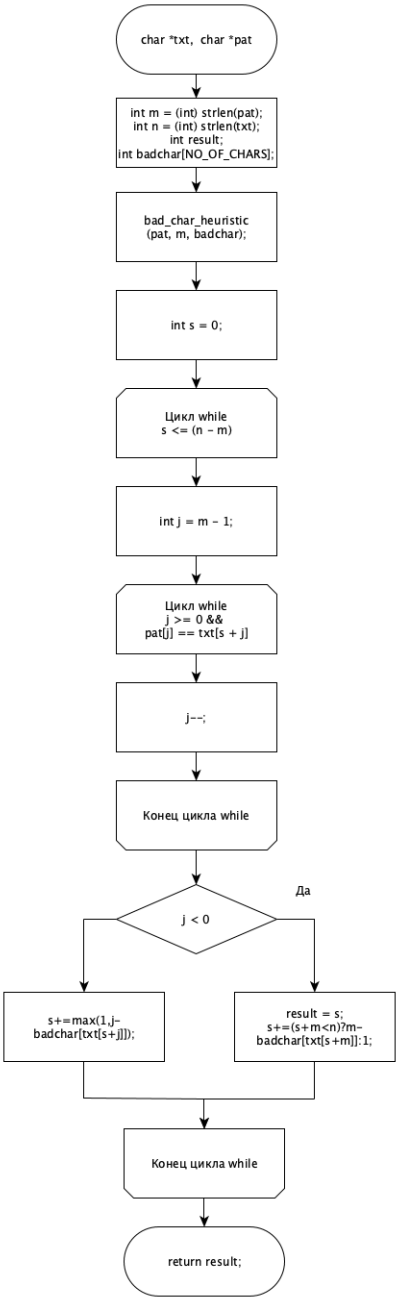

Рис. 5: Схема алгоритма Бойера-Мура

На рис. 6 представлена схема функции обработки массива смещений:

Рис. 6: Схема обработки массива смещений

2.2 Вывод

В данном разделе были рассмотрены схемы алгоритмов.

3 Технологическая часть

В этом разделе будут изложены требования к программному обеспечению и листинги алгоритмов.

3.1 Средства реализации

Данная программа разработана на языке C++, поддерживаемом многими операционными системами. Проект выполнен в среде Xcode.

3.2 Листинги функций

В данном разделе представлены листинги реализованных алгоритмов.

В листинге ?? представлен алгоритм Кнута-Морриса-Пратта.

Листинг 1: Алгоритм Кнута-Морриса-Пратта

```
int KMPsearch(char* pat, char* txt)
{
    int M = (int)strlen(pat);
    int N = (int)strlen(txt);

    int result = -1;

    int lps[M];

    compute_prefix(pat, M, lps);

    int i = 0;
    int j = 0;
    while (i < N)
    {
        if (pat[j] == txt[i])
        {
            j++;
            i++;
        }

        if (j == M)
        {
            result = i-j;
            j = lps[j - 1];
        }

        else if (i < N && pat[j] != txt[i])
        {
            if (j != 0)
                j = lps[j - 1];
            else
                i++;
```

```
34          }
35      }
36      return result;
37  }
38
39  void compute_prefix(char* pat, int M, int* lps)
40  {
41      int len = 0;
42
43      lps[0] = 0;
44
45      int i = 1;
46      while (i < M)
47      {
48          if (pat[i] == pat[len])
49          {
50              len++;
51              lps[i] = len;
52              i++;
53          }
54
55          else
56          {
57              if (len != 0)
58              {
59                  len = lps[len - 1];
60              }
61              else
62              {
63                  lps[i] = 0;
64                  i++;
65              }
66          }
67      }
68  }
```

В листинге 2 представлен алгоритм Бойера-Мура.

Листинг 2: Алгоритм Бойера-Мура

```
int BMsearch(char *txt,  char *pat)
{
    int m = (int) strlen(pat);
    int n = (int) strlen(txt);

    int result;

    int badchar[NO_OF_CHARS];

    bad_char_heuristic(pat, m, badchar);

    int s = 0;
    while (s <= (n - m))
    {
        int j = m - 1;

        while (j >= 0 && pat[j] == txt[s + j])
            j--;

        if (j < 0)
        {
            result = s;

            s += (s + m < n) ? m - badchar[txt[s + m]] : 1;

        }
        else
            s += max(1, j - badchar[txt[s + j]]);
    }

    return result;
}

void bad_char_heuristic(char *str, int size,
                        int badchar[NO_OF_CHARS])
{
    int i;

    for (i = 0; i < NO_OF_CHARS; i++)
        badchar[i] = -1;

    for (i = 0; i < size; i++)
        badchar[(int) str[i]] = i;
}
```

3.3 Вывод

В данном разделе были представлены листинги реализованных алгоритмов.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы.

4.1 Примеры работы

Далее приведены примеры работы программы:

```
abababcb
ababcb

KMP:
Pattern found at index: 2

BM:
Pattern found at index: 2
```

Рис. 7: Пример №1

```
theretheyare
they

KMP:
Pattern found at index: 5

BM:
Pattern found at index: 5
```

Рис. 8: Пример №2

4.2 Вывод

В данном разделе были приведены примеры работы программы.

Заключение

В ходе выполнения данной лабораторной работы были изучены два алгоритма для поиск подстроки в строке: Кнута-Морриса-Пратта и Бойера-Мура. Во время разработки программного обеспечения были получены практические навыки реализации указанных алгоритмов.

Список литературы

[1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.-М.:Техносфера, 2009.