Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Рубежный контроль №1
по курсу "Анализ алгоритмов"

Тема Реализация в параллельном режиме поиска подстроки в строке по алгоритму КМП

Студент Воякин А. Я.

Группа ИУ7-54Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2020 г.

Оглавление

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 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.

1.3 Вывод

Были рассмотрены три различных алгоритма поиска подстроки в строке.

2. Конструкторская часть

В данном разделе была поставлена задача поиска подстроки в строке и описан алгоритм КМП.

2.1 Схемы алгоритмов

На рис. 2.1 представлена схема алгоритма Кнута-Морриса-Пратта:

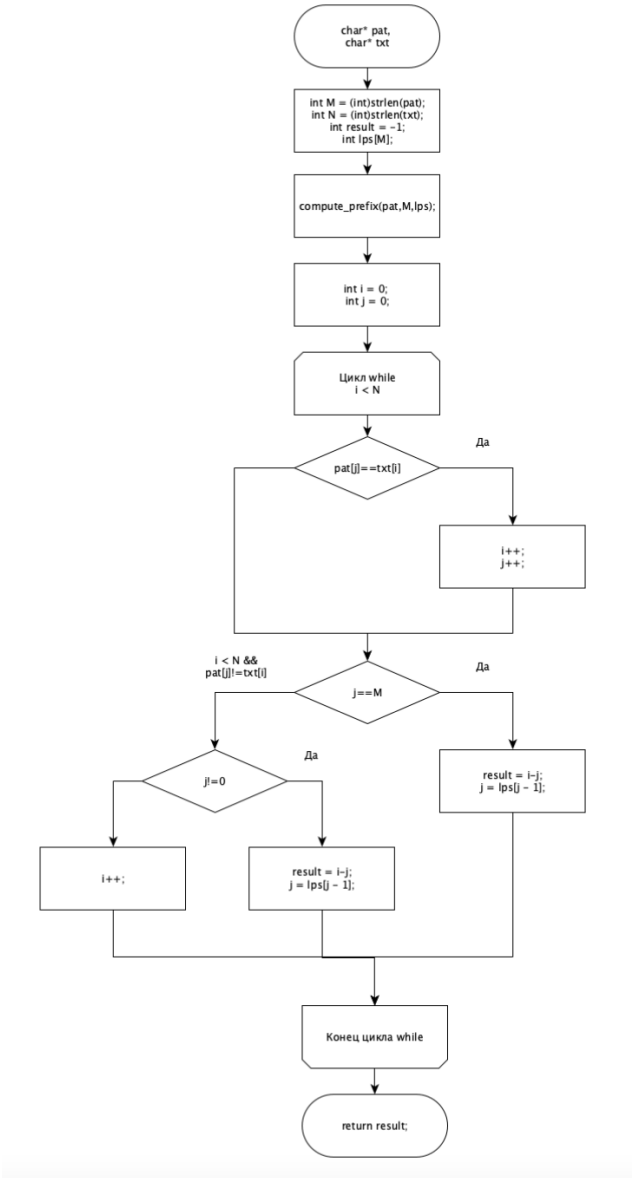

Рис. 2.1: Схема алгоритма Кнута-Морриса-Пратта

На рис. 2.2 представлена схема алгоритма нахождения префикса:

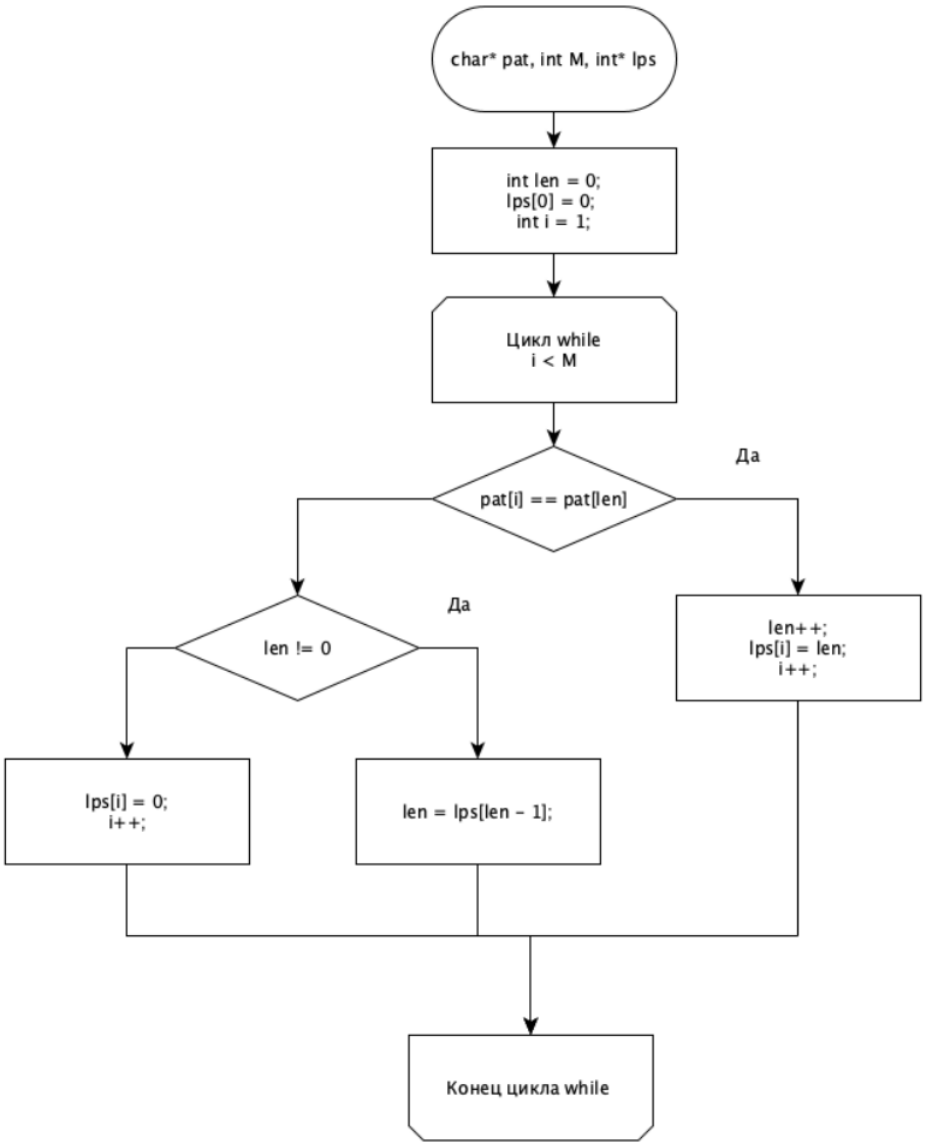

Рис. 2.2: Схема алгоритма нахождения префикса

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 представлен листинг реализованного алгоритма.

В листинге 3.1 представлен распараллеленный алгоритм Кнута-Морриса-Пратта.

Листинг 3.1: Распараллеленный алгоритм Кнута-Морриса-Пратта

```
1  static vector<string> mainFunc(vector<string> patterns, vector<string> text, const int
       threads) {
2    vector<string> result;
3    size_t numberOfStrings = text.size();
4
5    omp_set_num_threads(threads);
6    #pragma omp parallel for num_threads(threads)
7    for (int i = 0; i < numberOfStrings; i++) {
8      for (int j = 0; j < patterns.size(); j++) {
9        string resultOfLine = "";
10       if (text[j] == "") continue;
11       KMP(patterns[j], text[i], resultOfLine);
12       if (!resultOfLine.empty()) {
13         string numberLine = "\n\t Pattern \"" + patterns[j] + "\", number of line: " +
               to_string(i + 1);
14         #pragma omp critical
15         {
16           result.push_back(numberLine);
17           result.push_back(resultOfLine);
18           result.push_back(text[i]);
19         }
20       }
21     }
22   }
23   return result;
24 }
25
26 static vector<int> ComputePrefixFunction(string P) {
27   size_t len = P.length();
28   vector<int> s;
29   for (int i = 0; i < len; i++) s.push_back(0);
30   int border = 0;
31   for (int i = 1; i < len; i++) {
```

```
32      while ((border > 0) && (P[i] != P[border])) {
33        int index = border - 1;
34        border = s[index];
35      }
36      if (P[i] == P[border]) border = border + 1;
37      else border = 0;
38      s[i] = border;
39    }
40    return s;
41  }
42
43  static vector<int> FindAllOccurrences(string P, string T) {
44    string S = P + '#' + T;
45    vector<int> s = ComputePrefixFunction(S);
46    vector<int> result;
47    int len_pattern = P.length();
48    for (int i = (len_pattern + 1); i < S.length(); i++) {
49      if (s[i] == len_pattern) {
50        int find = 1 + i - 2 * len_pattern;
51        result.push_back(find);
52      }
53    }
54    return result;
55  }
56
57  void KMP(string Pattern, string Text, string &result) {
58    vector<int> res = FindAllOccurrences(Pattern, Text);
59    int len = Pattern.length();
60    for (int i = 0; i < res.size(); i++) {
61      int start = res[i];
62      int end = start + len - 1;
63      string buffer = "    " + to_string(start) + "-" + to_string(end);
64      result += buffer;
65    }
66  }
```

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

На рисунке 4.1 представлена демонстрация работы программы:

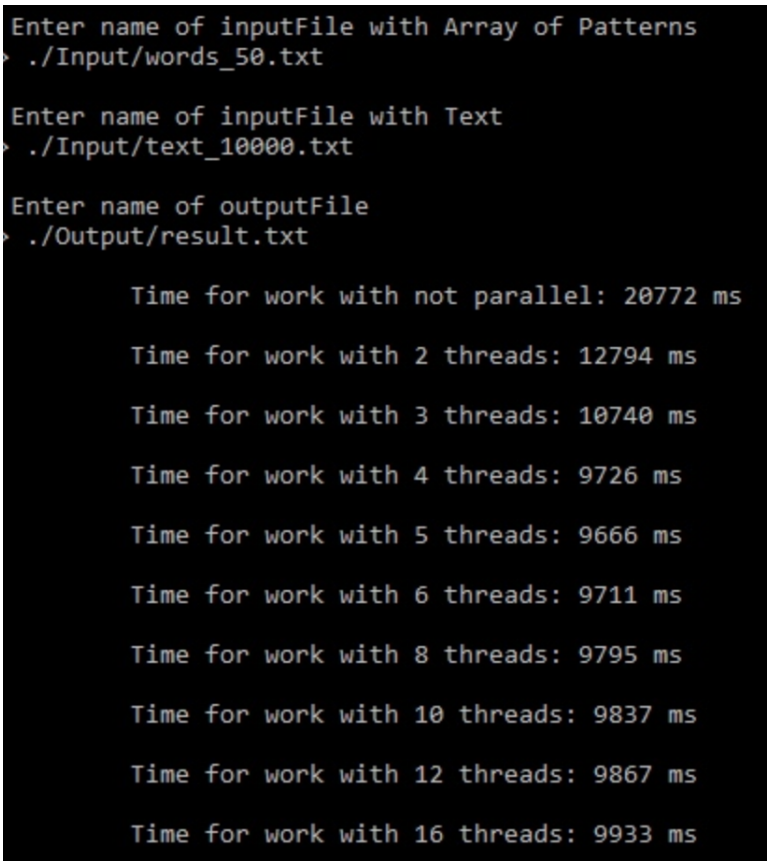

Рис. 4.1: Демонстрация работы программы

4.2 Вывод

В данном разделе были приведены примеры работы программы.

Заключение

В ходе выполнения был реализован в параллельном режиме поиск подстроки в строке по алгоритму КМП.

Литература

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

Литература