****

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МИРЭА – Российский технологический университет»**

# РТУ МИРЭА

Отчет по выполнению практического задания №2.3

**Тема:** Поиск образца в тесте

Дисциплина: **«**СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ**»**

Выполнил студент Кузнецов А. А.

группа ИКБО-01-21

**Москва 2022**

**Цель работы:** получить знания и навыки применения алгоритмов поиска в тексте подстроки (образца).

**Задание:**

1. Выполнить разработку программы, выполняя все этапы разработки.

2. Включить в этап «Описание модели (подход к решению)» описание алгоритма рассматриваемого метода. Разобрать алгоритм на примере. Подсчитать количество сравнений для успешного поиска первого вхождения образца в текст и безуспешного поиска. Определить функцию (или несколько функций) для реализации алгоритма. Определить предусловие и постусловие.

3. Сформировать таблицу тестов с указанием успешного и неуспешного поиска, используя большие и небольшие по объему текст и образец и включить ее в этап тестирование.

4. Разработать программу

5. Оценить практическую сложность алгоритма в зависимости от длины текста и длины образца и отобразить результаты в таблицу (для отчета).

6. Сравнить время поиска различных методов на одних и тех же наборах данных.

**Индивидуальный вариант:**

№1) Дано предложение, состоящее из слов, разделенных знаками препинания. Определить количество слов равных последнему слову, больших последнего слова.

№2) Строка S была записана много раз подряд, после чего из получившейся строки взяли произвольную часть строки - подстроку и передали как входные данные. Необходимо определить минимально возможную длину исходной строки S.

Реализация алгоритмом Кнута-Мориса-Пратта.

**Математическая модель решения**

Для выполнения задания индивидуального варианта выл выбран алгоритм Кнута-Мориса-Пратта. Схема алгоритма:

* Считать значения префикс-функции \pi[i] будем по очереди: от i=1 к i=n-1 (значение \pi[0] просто присвоим равным нулю).
* Для подсчёта текущего значения \pi[i] мы заводим переменную j, обозначающую длину текущего рассматриваемого образца. Изначально j = \pi[i-1].
* Тестируем образец длины j, для чего сравниваем символы s[j] и s[i]. Если они совпадают — то полагаем \pi[i] = j+1 и переходим к следующему индексу i+1. Если же символы отличаются, то уменьшаем длину j, полагая её равной \pi[j-1], и повторяем этот шаг алгоритма с начала.
* Если мы дошли до длины j=0 и так и не нашли совпадения, то останавливаем процесс перебора образцов и полагаем \pi[i] = 0 и переходим к следующему индексу i+1.

Пример нахождения префикс-функции для строки "abcabcd".

Префикс-функция равна: [0, 0, 0, 1, 2, 3, 0], что означает:

* у строки "a" нет нетривиального префикса, совпадающего с суффиксом;
* у строки "ab" нет нетривиального префикса, совпадающего с суффиксом;
* у строки "abc" нет нетривиального префикса, совпадающего с суффиксом;
* у строки "abca" префикс длины 1 совпадает с суффиксом;
* у строки "abcab" префикс длины 2 совпадает с суффиксом;
* у строки "abcabc" префикс длины 3 совпадает с суффиксом;
* у строки "abcabcd" нет нетривиального префикса, совпадающего с суффиксом.

Другой пример — для строки "aabaaab" она равна: [0, 1, 0, 1, 2, 2, 3].

**Код программы**

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include <iostream>  #include <vector>  #include <string>  using namespace std;  // пример функции обработки, которая выводит индекс начала найденного образца  int f(size\_t i)  {  //printf("%d\n", i);  return 1;  }  // str строка поиска.  // obr образец, который ищем.  // pi массив длин префиксов для образца (минимум сколько символов в образце).  // int f(size\_t i) когда образец найден, вызывается эта функция,  // ей передается индекс начала найденного в str образца.  // функция возвращает 0, если надо прекратить поиск и 1, если надо продолжить.  void prefix\_find(char\* str, char\* obr, size\_t\* pi, int (\*f)(size\_t), int& counter)  {  pi[0] = 0; // в i-м элементе (его индекс i-1) количество совпавших  // символов в начале образца и в конце подстроки длины i.  // p[0]=0 всегда, p[1]=1, если начинается с двух одинаковых  size\_t l; // будет длина образца  // заполняем массив длин префиксов для образца  for (l = 1; obr[l]; ++l)  {  size\_t j = pi[l - 1];  while ((j > 0) && (obr[l] != obr[j])) // не равны  j = pi[j - 1]; // берем ранее рассчитанное значение (начиная с максимально возможных)  if (obr[l] == obr[j]) // равны  ++j;  pi[l] = j;  }  size\_t j = 0; // количество совпавших символов, оно же индекс сравниваемого  // символа в образце. В строке сравниваемый символ будет иметь индекс i  for (size\_t i = 0; str[i]; ++i)  {  while ((j > 0) && (str[i] != obr[j]))  // Очередной символ строки не совпал с символом в образце. Сдвигаем образец,  // причем точно знаем, что первые j символов образца совпали с символами строки  // и надо сравнить j+1й символ образца (его индекс j) с i+1м символом строки.  j = pi[j - 1]; // если j=0, то достигли начала образца и цикл следует прервать  if (str[i] == obr[j]) // есть совпадение очередного символа  ++j; // увеличиваем длину совпавшего фрагмента на 1  if (j == l) {  if (!f(i - l + 1)) // образец найден, вызовем функцию обработки  return; // и выйдем из процедуры, если она вернет 0.  counter++;  }  }  }  vector<char>\* get\_last\_word(string str) {  vector<char> \* last\_word = new vector<char>();  reverse(str.begin(), str.end());  for (auto& i : str) {  if (isalpha(i)) {  last\_word->push\_back(i);  }  else if (last\_word->size() > 0)  break;  }  return last\_word;  }  void task1()  {  cout << "Enter text: ";  string text;  cin.get();  getline(cin, text);  char\* str = new char[65546](); //Aba bala ba, da ba la bada la ba! Bala bla bada ba.  string string\_str = str;  strcpy(str, text.c\_str());  vector<char>\* last\_word = get\_last\_word(string\_str);  char \* obr = new char[65546]();  for (int i = 0; i < last\_word->size(); i++)  obr[i] = last\_word->at(last\_word->size()-i-1);  cout << "Last word finded: " << obr << endl;  size\_t \* pi = new size\_t();  int counter = 0;  prefix\_find(str, obr, pi, f, counter);  cout << "Word (" << obr << ") was founded in text for " << counter << " times." << endl;  system("pause");  }  void task2()  {  cout << "Enter text: ";  string text\_str;  cin.get();  getline(cin, text\_str);  cout << "Enter sub: ";  string text\_sub;  cin.get();  getline(cin, text\_sub);  char \* str = new char[65546](); //I love SIAOD, it likes me!    char \* sub = new char[65546]();  strcpy(str, text\_str.c\_str());  strcpy(sub, text\_sub.c\_str());  size\_t\* pi = new size\_t();  int counter = 0;  prefix\_find(str, sub, pi, f, counter);  cout << "The sentence has been copied at least " << counter << " times." << endl;  system("pause");  }  int main() {  cout << "Enter the task number: ";  int task;  cin >> task;  switch (task) {  case 1:  task1();  break;  case 2:  task2();  break;  default:  break;  }  return 0;  } |

**Результаты тестирования**

**Задание 1**

Программа предлагает выбрать номер задания (см. Рис 1). Введем “1” и текст со знаками препинания и разными регистрами букв для проверки задания (см. Рис. 2).

****

Рисунок 1. Ввод номера задания.

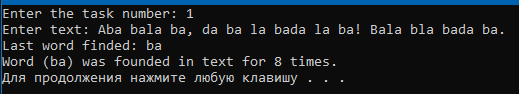


Рисунок 2. Ввод текста и проверка выполнения.

По результатам тестирования подстрока (последнее слово – ba) было найдено верное количество раз (8 раз).

**Задание 2**

Теперь выберем второе задание и скопируем 7 раз предложение. В качестве подстроки введем часть этого предложения. (см. Рис. 3).

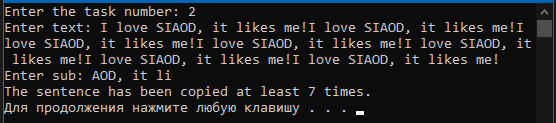
****

Рисунок 3. Проверка второго задания.

По результатам тестирования программа определила, что предложение было скопировано как минимум 7 раз, что является корректным значением.

**ВЫВОДЫ**

В результате выполнения данной работы мной были освоены навыки применения алгоритмов для поиска подстроки в тексте. Мной был изучен алгоритм Кнута-Мориса-Пратта.

Он был применён для решения заданий практической работы. Таким образом, мной был получен практический опыт использования алгоритмов для поиска подстроки.