МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА» - Российский технологический университет»

РТУ МИРЭА

Отчёт по выполнению практического задания № 6.2
Тема:
«Поиск образца в тексте»

Выполнил студент: Данов Арсений Иванович

Группа: ИКБО-10-23

Москва 2024

Цель:

Освоить приёмы реализации алгоритмов поиска образца в тексте.

Задание:

Разработайте приложения в соответствии с заданиями в индивидуальном варианте.

Вариант:

Номер: *6*

Задачи варианта:

1. Дан произвольный текст, состоящий из слов, разделенных знаками препинания. Отредактировать его, оставив между словами по одному пробелу, а между предложениями по два.
2. Дана непустая строка S, длина которой N не превышает 106 . Считать, что элементы строки нумеруются от 1 до N. Требуется для всех i от 1 до N вычислить π[i] – префикс функцию.

1.1 Формулировка задачи

Дан произвольный текст, состоящий из слов, разделенных знаками препинания. Отредактировать его, оставив между словами по одному пробелу, а между предложениями по два.

1.2 Ход решения

Для выполнения первой задачи варианта необходимо отредактировать текст, исключив из него все знаки препинания и замених их нужным количеством пробелов.

Всего в письменной речи используется не очень большой набор знаков препинания. Для разграничения слов используют запятые, точки с запятой, двоеточия, тире, слешы, кавычки, скобки, а также пробелы. Для разграничения предложений используются точки, вопросительный и восклицательный знаки, а также символы переноса строки.

Поэтому для выполнения задания достаточно линейно проходиться по строке, из которой все символы не знаки препинания переписывать в новую строку, знаки препинания разделения слов замнять одним пробелом, а разделения предложений — двумя. Также при вставке пробелов необходимо учитывать их количество, чтобы не вставить два пробела между словами из-за двух или трёх знаков препинания подряд.

1.3 Исходный код программ

На рисунке 1 представлен исходный коды программы.

```
6   void add_space(string &text, int count)
7   {
8       int ln = text.length();
9       if (ln == 0)
10      {
11          return;
12      }
13      if (ln == 1)
14      {
15          text += string(count, ' ');
16      }
17      if (text[ln - 1] != ' ')
18      {
19          text += string(count, ' ');
20      }
21  }
22
23  string replace(string text)
24  {
25      string new_str = "";
26      string end = "!?.\n";
27      string between = ",;:-\"\'()/\\ «»—[]";
28      for (int i = 0; i < text.length(); ++i)
29      {
30          if (end.find(text[i]) != string::npos)
31          {
32              add_space(new_str, 2);
33          }
34          else if (between.find(text[i]) != string::npos)
35          {
36              add_space(new_str, 1);
37          }
38          else
39          {
40              new_str += text[i];
41          }
42      }
43      return new_str;
44  }
```

Рисунок 1 — Функции замены символов и вставки пробелов

1.4 Результаты тестирования

На рисунке 2 представлены результаты тестирования.

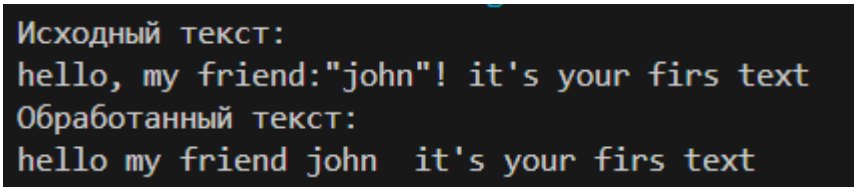

Рисунок 2 — Результат тестирования задания 1

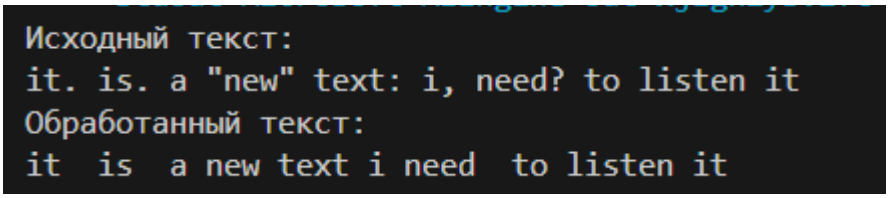

Рисунок 3 — Результат тестирования задания 1

2.1 Формулировка задачи

Дана непустая строка S, длина которой N не превышает 10^6. Считать, что элементы строки нумеруются от 1 до N. Требуется для всех i от 1 до N вычислить π[i] – префикс функцию.

2.2 Ход решения

Для нахождения префикс функции строки сначала необходимо создать массив из нулевых значений, которые мы будем динамически заполнять по ходу выполнения. Изначально префикс функция является препроцессингом для алгоритмов поиска вхождения подстроки в строку. В ней каждое значение является длиной наибольшего собственного префикса части образца, одновременно являющееся его суффиксом.

На рисунке 4 представлен пример рассчитанной префикс функции для образца текста.

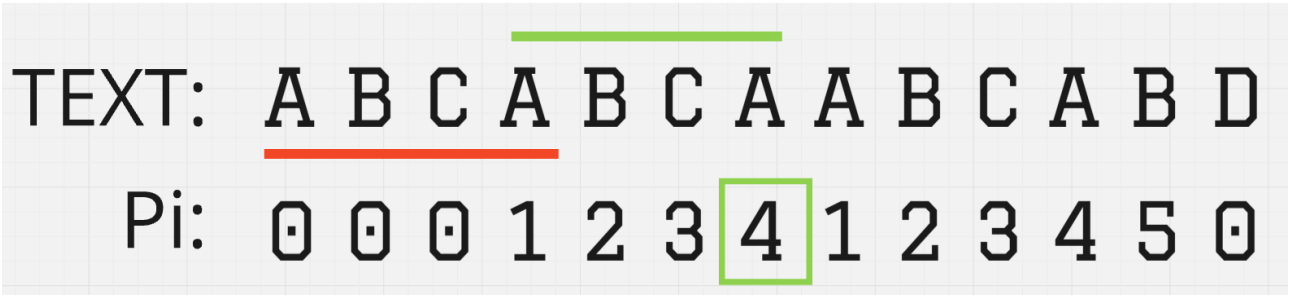

Рисунок 4 — Пример рассчёта Pi функции

Выделенный зеленым сверху текст имеет длину 4 и совпадает с первыми четыремя символами всего образца, из-за чего у последнего символа зеленой подстроки префиксная функция принимает значение 4.

Расчёт функции происходит динамически для каждого элемента образца. Значение функции в точке 0 полагается равным 0, а далее для каждого символа происходит сравнение: если он равен символу, чей индекс определяется значением префикс функции предыдущего элемента, значит он также является продолжением префикса строки и его значение префиксной функции будет на единицу больше, чем у предыдущего элемента. Иначе же проверяется значение на единицу меньше прошлого значения префиксной функции до тех пор, пока

не будет найдено совпадение или значение не обратится в ноль, после чего результат сохраняется.

2.3 Исходный код программ

На рисунке 5 представлен исходный коды программы.

```
54  vector<int> prefix_function(string s)
55  {
56      int n = s.length();
57      vector<int> p(n, 0);
58      for (int i = 1; i < n; ++i)
59      {
60          int cur = p[i - 1];
61          while (s[i] != s[cur] && cur > 0)
62          {
63              cur = p[cur - 1];
64          }
65          if (s[i] == s[cur])
66          {
67              p[i] = cur + 1;
68          }
69      }
70      return p;
71  }
```

Рисунок 5 — Функция расчёта префиксной функции

2.4 Результаты тестирования

На рисунке 6-7 представлены результаты тестирования. Также в таблицах 1-2 представлены результаты практической оценки сложности алгоритма префиксной функции и алгоритма поиска, на ней основанного.

```
Исходный текст:
good morning, i want to say google
Префиксная функция:
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 1, 0, 0 ]
```

Рисунок 6 — Результат тестирования задания 2

```
Исходный текст:
have a nice day, sir, i have to go with my hands
Префиксная функция:
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 2, 0, 0, 0 ]
```

Рисунок 7 — Результат тестирования задания 1

Таблица 1 — Оценка практической сложности префикс функции

Длина строки	**Количество сравнений**
10	40
100	420
1 000	5 194
10 000	56 608

Таблица 2 — Оценка практической сложности КМП поиска

Длина строки	**Количество сравнений**	
	Удачный поиск	**Неудачный поиск**
10	60	77
100	117	445
1 000	2 021	4 137
10 000	9 297	40 487

3.1 Выводы

В ходе решения поставленных задач были освоены приёмы реализации алгоритмов поиска образца в тексте.

В случае применения линейного поиска сложность алгоритма довольно высока, но благодаря применению более эффективных алгоритмов можно снизить сложность до окололинейной. В частности использование префикс функции и КМП поиска позволяет значительно снизить количество выполняемых операций.