МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Ахо-Корасик

Студентка гр. 7383 _____ Чемова К.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы.

Ознакомиться с алгоритмом Ахо-Корасика для эффективного поиска всех вхождений всех строк-образцов в заданную строку.

Выполнение работы.

Для решения поставленной задачи был написан класс `Aho_Karasik` и структура `bohr_vertex`. Структура используется для реализации бора. Бор – это дерево, в котором каждая вершина обозначает какую-то строку (корень обозначает нулевую строку – ε). На ребрах между вершинами записана буква, таким образом, добираясь по ребрам из корня в какую-нибудь вершину и контангенируя буквы из ребер в порядке обхода, мы получим строку, соответствующую этой вершине.

В этой структуре мы используем поля:

- `int num` – номер строки-образца;
- `bool flag` – показыватn является ли вершина исходной строкой;
- `map <char, int> edge` – вершины, в которые мы можем пойти из данной;
- `map <char, int> auto_move` – запоминает переходы автомата.

Переход выполняется по двум параметрам – текущей вершине `m_par` и символу `m_symb`, по которому нам надо сдвинуться из этой вершины. Необходимо найти вершину `u`, которая обозначает наидлиннейшую строку, состоящую из суффикса строки `m_par`(возможно нулевого) + символа `m_symb`. Если такого в боре нет, то идем в корень, `int s_link` – суффиксная ссылка, `int par` – индекс вершины родителя, `char symb` – символ на ребре от родителя к этой вершине. В классе `Aho_Karasik` хранятся `vector<bohr_vertex> bohr` – вектор, для хранения вершин, `vector <string> patterns` – вектор для хранения строк-шаблонов, `int counter` – счетчик вершин. Также реализованы некоторые методы: `void add_string ()` – метод, создающий бор и добавляющий строки в `patterns`, `void find_all_pos()` – метод, который выполняет автоматные переходы и выводит ответ на экран, `int suff_link()` – возвращает суффиксальную ссылку для данной вершины,

`int get_auto_move()` – метод, который выполняет автоматные переходы, `void check()` – метод, который осуществляет хождение по хорошим суффиксальным ссылкам из текущей позиции, учитывая, что эта позиция оканчивается на символ i.

Тестирование.

Программа собрана в операционной системе Ubuntu 16.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось.

Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении А.

Исследование алгоритма.

Структура данных map из STL реализована красно-черным деревом, а время обращения к его элементам пропорционально логарифму числа элементов. Следовательно вычислительная сложность O((H + n)logk + c), где H длина текста, в котором производится поиск, n – общая длина всех слов в словаре, k – размер алфавита, c – общая длина всех совпадений.

Сложность по памяти – O(H + n), так как. память выделяется для вершин шаблонов и для хранения текста.

Выводы.

В ходе данной лабораторной работы был реализован алгоритм Ахо-Корасика на языке C++. Данный алгоритм производит точный поиск набора образцов в строке. Были изучены новые структуры данных и понятия, такие как бор, суффиксальные ссылки. Код программы представлен в приложении Б.

ПРИЛОЖЕНИЕ А

ТЕСТОВЫЕ СЛУЧАИ

Входные данные	Результат
ACT A$ $	1
ACATC A$ $	1 3
ACATC C$$ $	2

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПОИСКА ПОДСТРОКИ

1.cpp

```
#include <iostream>
#include <map>
#include <vector>
using namespace std;

struct bohr_vertex {
  bool flag;
  int num;//номер образца
  int s_link;//суффиксная ссылка
  int par;//вершина-отец в дереве
  char symb; //символ на ребре от par к этой вершине
  map <char, int> edge;//номер вершины, в которую мы придем по символу с
номером i в алфавите
  map <char, int> auto_move;//auto_move - запоминание перехода автомата
  bohr_vertex(int m_par, char m_symb): par(m_par), symb(m_symb) {
    flag = false;
    num = 0;
    s_link = -1;//изначально суф. ссылки нет
  }
};

class Aho_Karasik{
  vector <bohr_vertex> bohr;
  vector <string> patterns;
  int counter;//счетчик узлов бора
public:
  Aho_Karasik(){//создание корня бора
    counter = 1;
    bohr.push_back(bohr_vertex(0, 0));
  }

  void add_string_to_bohr(string& s, int str){//добавление строки-образца в
бор
    int n = 0;//начинаем с корня
    patterns.push_back(s);
    for (int i = 0; i < s.length(); i++){
      if (bohr[n].edge.find(s[i]) == bohr[n].edge.end()){//нет путей в
искомую вершину
        bohr.push_back(bohr_vertex(n, s[i]));
        bohr[n].edge[s[i]] = counter++;
      }
      n = bohr[n].edge[s[i]];
    }
    bohr[n].flag = true;//n - конечная вершина
    bohr[n].num = patterns.size();
  }
```

```
  int suff_link(int v){//возвращает индекс суффиксной ссылки
    if (bohr[v].s_link == -1){//если еще не считали
      if (v == 0 || bohr[v].par == 0) //если v - корень или предок v - корень
        bohr[v].s_link = 0;
      else
        bohr[v].s_link = get_auto_move(suff_link(bohr[v].par), bohr[v].symb);
    }
    return bohr[v].s_link;
  }

  int get_auto_move(int v, char symbol){//переходы автомата
    if (bohr[v].auto_move.find(symbol) == bohr[v].auto_move.end())
      if (bohr[v].edge.find(symbol) != bohr[v].edge.end())
        bohr[v].auto_move[symbol] = bohr[v].edge[symbol];
      else if (v == 0)//если v - корень
        bohr[v].auto_move[symbol] = 0;
      else
        bohr[v].auto_move[symbol] = get_auto_move(suff_link(v), symbol);
    return bohr[v].auto_move[symbol];
  }

  void check(int v, int i) {  //хождение по хорошим суффиксальнм ссылкам из
текущей позиции, учитывая, что эта позиция оканчивается на символ i
    for (int u = v; u != 0; u = suff_link(u))
      if (bohr[u].flag)
        cout<< i - patterns[bohr[u].num - 1].size() + 2 << " " << bohr[u].num
<< endl;
  }

  void find_all_pos(string s) {
    int u = 0;
    for (int i = 0; i < s.length(); i++) {
      u = get_auto_move(u, s[i]);
      check(u, i);
    }
  }
};

int main(){
  Aho_Karasik a_k;
  string text, pattern;
  int num;
  cin >> text >> num;
  for (int i = 0; i < num; i++){
      cin >> pattern;
      a_k.add_string_to_bohr(pattern, i + 1);
  }
  a_k.find_all_pos(text);
  return 0;
}
```

2.cpp

```
#include <iostream>
#include <map>
#include <vector>
using namespace std;

struct bohr_vertex {
  bool flag;
  int num;//номер образца
  int s_link;//суффиксная ссылка
  int par;//вершина-отец в дереве
  char symb; //символ на ребре от par к этой вершине
  map <char, int> edge;//номер вершины, в которую мы придем по символу с
номером i в алфавите
  map <char, int> auto_move;//auto_move - запоминание перехода автомата
  bohr_vertex(int m_par, char m_symb): par(m_par), symb(m_symb) {
    flag = false;
    num = 0;
    s_link = -1;//изначально - суф. ссылки нет
  }
};

class Aho_Karasik{
  vector <bohr_vertex> bohr;
  vector <string> patterns;
  int counter;//счетчик узлов бора
  char joker;
public:
  Aho_Karasik(char _joker, string& s): joker(_joker){//создание корня бора
    counter = 1;
    bohr.push_back(bohr_vertex(0, 0));
    int n = 0;//начинаем с корня
    patterns.push_back(s);
    for (int i = 0; i < s.length(); i++){//добавление строки-образца в бор
      if (bohr[n].edge.find(s[i]) == bohr[n].edge.end()){//если от вершины
нет путей в искомую
        bohr.push_back(bohr_vertex(n, s[i]));
        bohr[n].edge[s[i]] = counter++;
      }
      n = bohr[n].edge[s[i]];
    }
    bohr[n].flag = true;//n - конечная вершина
    bohr[n].num = patterns.size();
  }

  int suff_link(int v){//возвращает индекс суффиксной ссылки
    if (bohr[v].suff_link == -1){//если еще не считали
      if (v == 0 || bohr[v].par == 0) //если v - корень или предок v - корень
        bohr[v].suff_link = 0;
      else
        bohr[v].suff_link = get_auto_move(suff_link(bohr[v].par),
bohr[v].symb);
    }
    return bohr[v].suff_link;
```

```
  }

  int get_auto_move(int v, char symbol){//переходы автомата
    if (bohr[v].auto_move.find(symbol) == bohr[v].auto_move.end())
      if (bohr[v].edge.find(symbol) != bohr[v].edge.end())
        bohr[v].auto_move[symbol] = bohr[v].edge[symbol];
      else if (bohr[v].edge.find(joker) != bohr[v].edge.end())
        bohr[v].auto_move[symbol] = bohr[v].edge[joker];
      else if (v == 0)//если v - корень
        bohr[v].auto_move[symbol] = 0;
      else
        bohr[v].auto_move[symbol] = get_auto_move(suff_link(v), symbol);
    return bohr[v].auto_move[symbol];
  }

  void check(int v, int i) {  //хождение по хорошим суффиксальнм ссылкам из
текущей позиции, учитывая, что эта позиция оканчивается на символ i
    for (int u = v; u != 0; u = suff_link(u))
      if (bohr[u].flag)
        cout << i - patterns[bohr[u].num - 1].size() + 2 << endl;
  }

  void find_all_pos(string s) {
    int u = 0;
    for (int i = 0; i < s.length(); i++) {
      u = get_auto_move(u, s[i]);
      check(u, i);
    }
  }
};

int main(){
  string text, pattern;
  char joker;
  cin >> text >> pattern >> joker;
  Aho_Karasik a_k(joker, pattern);
  a_k.find_all_pos(text);
  return 0;
}
```