**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**отчет**

**по лабораторной работе** №**5**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: Алгоритм Ахо-Корасик**

Студент гр.8304 \_\_\_\_\_\_\_\_\_\_\_\_\_ Холковский К.В.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ Размочаева Н.В.

Санкт-Петербург

2020

**Задание.**

Вариант 1

На месте джокера может быть любой символ, за исключением заданного.

**Цель работы.**

Используя реализацию точного множественного поиска, решите задачу точного поиска для одного образца с *джокером*.

В шаблоне встречается специальный символ, именуемый джокером (wild card), который "совпадает" с любым символом. По заданному содержащему шаблоны образцу *P* необходимо найти все вхождения Р в текст Т.

Например, образец а*b*??с? с джокером ? встречается дважды в тексте *xabvccbababcax*.

Символ джокер не входит в алфавит, символы которого используются в *T*. Каждый джокер соответствует одному символу, а не подстроке неопределённой длины. В шаблон входит хотя бы один символ не джокер, т.е. шаблоны вида ??? недопустимы. Все строки содержат символы из алфавита {*A*, *C*, *G*, *T*, *N*}

**Описание алгоритма.**

Алгоритм строит конечный автомат, которому затем передаёт строку поиска. Автомат получает по очереди все символы строки и переходит по соответствующим рёбрам. Если автомат пришёл в конечное состояние, соответствующая строка словаря присутствует в строке поиска.

Для того чтобы найти все вхождения в текст заданного шаблона с масками Q, необходимо обнаружить вхождения в текст всех его безмасочных кусков.  
Пусть {Q1,…,Qk} — набор подстрок Q, разделенных масками, и пусть {l1,…,lk} — их стартовые позиции в Q. Например, шаблон abφφcφ содержит две подстроки без масок ab и cc и их стартовые позиции соответственно 1 и 5.

Для алгоритма понадобится массив C. C[i] — количество встретившихся в тексте безмасочных подстрок шаблона, который начинается в тексте на позиции i. Тогда появление подстроки Qi в тексте на позиции j будет означать возможное появление шаблона на позиции j−li+1.

1. Используя алгоритм Ахо-Корасик, находим безмасочные подстроки шаблона Q: когда находим Qi в тексте T на позиции j, увеличиваем на единицу C[j−li+1].
2. Каждое i, для которого C[i] = k, является стартовой позицией появления шаблона Q в тексте.

Вычислительная сложность алгоритма: O(2m + n + a), где n – длинна шаблона, m – длинна текста, a – кол-во появлений подстрок шаблона.

**Описание функций и структур данных.**

Структура для хранения вершины бора, а сам бор хранится в векторе таких вершин:

**struct** bohr\_vrtx {  
 bohr\_vrtx(**int** p = -1, **char** c = 0) : symb(c), par(p) {}  
 **int** next\_vrtx[K] = { -1,-1,-1,-1,-1 };  
 **int** auto\_move[K] = { -1,-1,-1,-1,-1 };  
 std::vector<**int**> pat\_num;  
 **int** par = -1;  
 **int** suff\_link = -1;  
 **int** suff\_flink = -1;  
 **bool** flag = **false**;  
 **char** symb = 0;  
};

Функция добавления строки в бор:

**void** add\_string\_to\_bohr(**const** std::string& s, std::vector<bohr\_vrtx>& bohr, std::vector<std::string>& pattern)

Функция проверки на наличие строки в боре:

**bool** is\_string\_in\_bohr(**const** std::string& s, std::vector<bohr\_vrtx> **const**& bohr)

Функция выявления суффиксной ссылки:

**int** get\_suff\_link(**int** v, std::vector<bohr\_vrtx>& bohr)

Функция для перехода из вершины v:

**int** get\_auto\_move(**int** v, **char** ch, std::vector<bohr\_vrtx>& bohr)

Функция выявления хорошей суффиксной ссылки:

**int** get\_suff\_flink(**int** v, std::vector<bohr\_vrtx>& bohr)

Функция хождения по хорошим суффиксным сслыкам:

**void** check(**int** v, **int** i, std::vector<bohr\_vrtx>& bohr, std::vector<std::string>& pattern, std::vector<**int**>& C, std::ostream & out)

Функция поиска:

**void** find\_all\_pos(std::istream & in, std::ostream & out)

**Выводы.**

В ходе выполнения работы, была написана программа, находящая точное вхождение образца с джокером и получены знания о такое структуре данных как бор.

**Тестирование**

|  |  |
| --- | --- |
| **Input** | **Output** |
| NACGNTTACGGTCACNN  AC$$T$AC$$  $  C | 2 |
| NACGNTTACGGTCACNN  AC$$T$AC$$  $  A | 2  8 |
| ACTANCA  A$$A$  $  G | 1 |

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД**

#include **<iostream>**#include **<map>**#include **<vector>**#include **<fstream>  
  
constexpr const char**\* PATH\_IN = **"D:/test.txt"**;  
**constexpr const char**\* PATH\_OUT = **"D:/result.txt"**;  
  
**const int** K = 5;  
**const** std::map<**char**, **int**> alphabet = { {**'A'**,0},{**'C'**,1},{**'G'**,2},{**'T'**,3},{**'N'**,4} };  
std::vector<size\_t> l;  
  
**struct** bohr\_vrtx {  
 bohr\_vrtx(**int** p = -1, **char** c = 0) : symb(c), par(p) {}  
 **int** next\_vrtx[K] = { -1,-1,-1,-1,-1 };  
 **int** auto\_move[K] = { -1,-1,-1,-1,-1 };  
 std::vector<**int**> pat\_num;  
 **int** par = -1;  
 **int** suff\_link = -1;  
 **int** suff\_flink = -1;  
 **bool** flag = **false**;  
 **char** symb = 0;  
};  
  
  
**void** add\_string\_to\_bohr(**const** std::string& s, std::vector<bohr\_vrtx>& bohr, std::vector<std::string>& pattern) {  
 **int** num = 0; *//начинаем с корня* **for** (**char** i : s) {  
 **char** ch = alphabet.at(i); *//получаем номер в алфавите* **if** (bohr[num].next\_vrtx[ch] == -1) { *//-1 - признак отсутствия ребра* bohr.emplace\_back(num, ch);  
 bohr[num].next\_vrtx[ch] = bohr.size() - 1;  
 }  
 num = bohr[num].next\_vrtx[ch];  
 }  
 bohr[num].flag = **true**;  
 pattern.push\_back(s);  
 bohr[num].pat\_num.push\_back(pattern.size() - 1);  
}  
  
**bool** is\_string\_in\_bohr(**const** std::string& s, std::vector<bohr\_vrtx> **const**& bohr) {  
 **int** num = 0;  
 **for** (**char** i : s) {  
 **char** ch = alphabet.at(i);  
 **if** (bohr[num].next\_vrtx[ch] == -1) {  
 **return false**;  
 }  
 num = bohr[num].next\_vrtx[ch];  
 }  
 **return true**;  
}  
  
**int** get\_auto\_move(**int** v, **char** ch, std::vector<bohr\_vrtx>& bohr);  
  
**int** get\_suff\_link(**int** v, std::vector<bohr\_vrtx>& bohr) {  
 **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(get\_suff\_link(bohr[v].par, bohr), bohr[v].symb, bohr);  
 }  
 **return** bohr[v].suff\_link;  
}  
  
**int** get\_auto\_move(**int** v, **char** ch, std::vector<bohr\_vrtx>& bohr) {  
 **if** (bohr[v].auto\_move[ch] == -1) {  
 **if** (bohr[v].next\_vrtx[ch] != -1)  
 bohr[v].auto\_move[ch] = bohr[v].next\_vrtx[ch];  
 **else if** (v == 0)  
 bohr[v].auto\_move[ch] = 0;  
 **else** bohr[v].auto\_move[ch] = get\_auto\_move(get\_suff\_link(v, bohr), ch, bohr);  
 }  
 **return** bohr[v].auto\_move[ch];  
}  
  
**int** get\_suff\_flink(**int** v, std::vector<bohr\_vrtx>& bohr) {  
 **if** (bohr[v].suff\_flink == -1) {  
 **int** u = get\_suff\_link(v, bohr);  
 **if** (u == 0) *//либо v - корень, либо суф. ссылка v указывает на корень* bohr[v].suff\_flink = 0;  
 **else** bohr[v].suff\_flink = (bohr[u].flag) ? u : get\_suff\_flink(u, bohr);  
 }  
 **return** bohr[v].suff\_flink;  
}  
  
**void** check(**int** v, **int** i, std::vector<bohr\_vrtx>& bohr, std::vector<std::string>& pattern, std::vector<**int**>& C, std::ostream & out) {  
 **bool** f = **false**;  
 **int** vert\_num;  
 **for** (**int** u = v; u != 0; u = get\_suff\_flink(u, bohr)) {  
 out << u << **" -> "**;  
 **if** (bohr[u].flag) {  
 f = **true**;  
 vert\_num = u;  
 **for** (**int** k : bohr[u].pat\_num) {  
  
 **int** j = i - pattern[k].size() + 1;  
 **int** adsad = j - l[k] + 1;  
 **if** (adsad > 0 && adsad < C.size()) *//проверка выхода за рамки* ++C[adsad];  
 }  
  
 }  
 }  
 out << **'0'**;  
 **if** (f)  
 out << **"\nWas finded leaf - "** << vert\_num << **" vertex"**;  
 out << **"\n\n"**;  
}  
  
**void** find\_all\_pos(std::istream & in, std::ostream & out){  
  
 std::vector<std::string> pattern;  
 std::vector<bohr\_vrtx> bohr = { bohr\_vrtx() };  
  
 std::string curr, text, tamplate;  
 **char** J, no;  
 in >> text >> tamplate >> J >> no;  
 tamplate += J;  
 **for** (**int** i = 0; i < tamplate.size(); ++i) {  
 **if** (tamplate[i] == J) {  
 **if** (curr.empty())  
 **continue**;  
 **else** {  
 add\_string\_to\_bohr(curr, bohr, pattern);  
 l.push\_back(i - curr.size());  
 curr = **""**;  
 **continue**;  
 }  
 }  
 curr += tamplate[i];  
 }  
  
 **int** u = 0;  
 std::vector<**int**> C(text.size());  
 **for** (**int** i = 0; i < text.size(); ++i) {  
 out << **"Go from "** << u << **" vertex"**;  
 u = get\_auto\_move(u, alphabet.at(text[i]), bohr);  
 out << **" to "** << u << **" vertex\nLink way for this vertex to start - "**;  
 check(u, i, bohr, pattern, C, out);  
 }  
 **for** (**int** k = 0; k < C.size(); ++k)  
 **if** (C[k] == l.size()) {  
 **bool** is\_correct = **true**;  
 **for** (size\_t i = k; i < k + tamplate.size() - 1; ++i) {  
 **if** (tamplate[i - k] == J && text[i-1] == no){ *//проверка на запрещенный символ* is\_correct = **false**;  
 **break**;  
 }  
 }  
 **if** (is\_correct && k + tamplate.size() - 2 <= text.size())  
 out << k << **"\n"**;  
 }  
}  
  
**int** main() {  
 **int** choseIn, choseOut;  
 std::cout << **"Input: 1 - console, 0 - file"** << std::endl;  
 std::cin >> choseIn;  
 **if**(choseIn!=0 && choseIn!=1) {  
 std::cout << **"Wrong chose Input"**;  
 **return** 0;  
 }  
 std::cout << **"Output: 1 - console, 0 - file"** << std::endl;  
 std::cin >> choseOut;  
 **if**(choseOut!=0 && choseOut!=1) {  
 std::cout << **"Wrong chose Output"**;  
 **return** 0;  
 }  
  
 **if**(choseIn==1 && choseOut==1)  
 find\_all\_pos(std::cin, std::cout);  
 **if**(choseIn==1 && choseOut==0){  
 std::ofstream file;  
 file.open(PATH\_OUT);  
  
 **if** (!file.is\_open()) {  
 std::cout << **"Can't open file!\n"**;  
 **return** 0;  
 }  
 find\_all\_pos(std::cin, file);  
 }  
 **if**(choseIn==0 && choseOut==1){  
 std::ifstream file;  
 file.open(PATH\_IN);  
  
 **if** (!file.is\_open()) {  
 std::cout << **"Can't open file!\n"**;  
 **return** 0;  
 }  
 find\_all\_pos(file, std::cout);  
 }  
 **if**(choseIn==0 && choseOut==0) {  
 std::ofstream file1;  
 file1.open(PATH\_OUT);  
  
 **if** (!file1.is\_open()) {  
 std::cout << **"Can't open file!\n"**;  
 **return** 0;  
 }  
 std::ifstream file2;  
 file2.open(PATH\_IN);  
  
 **if** (!file2.is\_open()) {  
 std::cout << **"Can't open file!\n"**;  
 **return** 0;  
 }  
  
 find\_all\_pos(file2, file1);  
 }  
 **return** 0;  
}