**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 \_\_\_\_\_\_\_\_\_\_\_\_\_ Рыжиков A.В.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ Размочаев Н.В.

Санкт-Петербург

2020

**Цель работы.**

Реализовать режим поиска, при котором все найденные образцы не пересекаются в строке поиска (т.е. некоторые вхождения не будут найдены; решение задачи неоднозначно)..

**Задание.**

Вариант 4  
**Вход:**

Первая строка содержит текст

Вторая – число последующих строк *n*.

Все строки содержат символы из алфавита

**Выход:**

Все вхождения образцов из *P* в *T*.  
Каждое вхождение образца в текст представить в виде двух чисел *i p*

**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.

Несколько строк поиска можно объединить в дерево поиска, так называемый бор префиксное дерево). Бор является конечным автоматом, распознающим одну строку из m но при условии, что начало строки известно.

Первая задача в алгоритме — научить автомат «самовосстанавливаться», если подстрока не совпала. При этом перевод автомата в начальное состояние при любой неподходящей букве не подходит, так как это может привести к пропуску подстроки (например, при поиске строки **aabab**, попадается aab**aabab**, после считывания пятого символа перевод автомата в исходное состояние приведёт к пропуску подстроки — верно было бы перейти в состояние a, а потом снова обработать пятый символ). Чтобы автомат самовосстанавливался, к нему добавляются суффиксные ссылки, нагруженные пустым символом ⌀ (так что детерминированный автомат превращается в недетерминированный). Например, если разобрана строка aaba, то бору предлагаются суффиксы aba, ba, a. Суффиксная ссылка — это ссылка на узел, соответствующий самому длинному суффиксу, который не заводит бор в тупик (в данном случае a).

Для корневого узла суффиксная ссылка — петля. Для остальных правило таково: если последний распознанный символ — c , то осуществляется обход по суффиксной ссылке родителя, если оттуда есть дуга, нагруженная символом c , суффиксная ссылка направляется в тот узел, куда эта дуга ведёт. Иначе — алгоритм проходит по суффиксной ссылке ещё и ещё раз, пока либо не пройдёт по корневой ссылке-петле, либо не найдёт дугу, нагруженную символом c .

.

**Описание функций и структур данных.**

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

**class** Element {  
**public**:  
  
 **explicit** Element(**char** name, Element \*parent) : name(name), parent(parent) {  
 sufLink = **nullptr**;  
 childs.resize(0);  
 numberString = 0;  
 isTerminal = **false**;  
 isFirstElement = **false**;  
 }  
  
 **explicit** Element() {  
 name = **'&'**;  
 parent = **nullptr**;  
 sufLink = **this**;  
 childs.resize(0);  
 numberString = 0;  
 isTerminal = **false**;  
 isFirstElement = **true**;  
 }  
  
  
**private**:  
 **char** name;  
 std::vector<Element \*> childs;  
  
  
**private**:  
 Element \*sufLink;  
 Element \*parent;  
 **bool** isTerminal;  
 **bool** isFirstElement;  
 **int** numberString;  
**public**:  
 **void** addChild(Element \*child) {  
 childs.emplace\_back(child);  
 }  
  
 **void** setTerminal(**int** numberString) {  
 isTerminal = **true**;  
 Element::numberString = numberString;  
 }  
  
 **const** std::vector<Element \*> &getChilds() **const** {  
 **return** childs;  
 }  
  
 **char** getName() **const** {  
 **return** name;  
 }  
  
 Element \*getSufLink() **const** {  
 **return** sufLink;  
 }  
  
 Element \*getParent() **const** {  
 **return** parent;  
 }  
  
 **void** setSufLink(Element \*sufLink) {  
 Element::sufLink = sufLink;  
 }  
  
 **bool** isFirstElement1() **const** {  
 **return** isFirstElement;  
 }  
  
 **bool** isTerminal1() **const** {  
 **return** isTerminal;  
 }  
  
 **int** getNumberString() **const** {  
 **return** numberString;  
 }  
  
  
};

Функция добавления строки в бор:

**void** addString(Element \*top, std::string \*string, **int** numberString)

Функция добавления вершины в бор:

Element \*createElement(Element \*parent, std::string \*string, **int** countElement, **int** numberString)

Функция добавления суффиксных ссылок и обработки текста:

**void** init(Element \*bor, std::vector<std::string> strings, std::string text,  
std::vector<std::pair<**int**, **int**>> \*vectorAnswers, std::ofstream &stream)

**Тестирование**

Таблица 1 – результаты тестирования

|  |  |
| --- | --- |
| **Input** | **Output** |
| sheshehee  3  she  he  e | 1 1  4 1  7 2  9 3 |
| hello\_kitty\_cat  3  hello  kitty  cat | 1 1  7 2  13 3 |
| abababaaabb  3  ab  aaa  bb | 1 1  3 1  5 1  7 2  10 3 |

**Выводы.**

В ходе выполнения работы, была написана программа, находящая точное вхождение образца при котором все найденные образцы не пересекаются в строке поиска.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД**

#include **<iostream>**#include **<vector>**#include **<queue>**#include **<algorithm>**#include **<fstream>  
  
class** Element {  
**public**:  
  
 **explicit** Element(**char** name, Element \*parent) : name(name), parent(parent) {  
 sufLink = **nullptr**;  
 childs.resize(0);  
 numberString = 0;  
 isTerminal = **false**;  
 isFirstElement = **false**;  
 }  
  
 **explicit** Element() {  
 name = **'&'**;  
 parent = **nullptr**;  
 sufLink = **this**;  
 childs.resize(0);  
 numberString = 0;  
 isTerminal = **false**;  
 isFirstElement = **true**;  
 }  
  
  
**private**:  
 **char** name;  
 std::vector<Element \*> childs;  
  
  
**private**:  
 Element \*sufLink;  
 Element \*parent;  
 **bool** isTerminal;  
 **bool** isFirstElement;  
 **int** numberString;  
**public**:  
 **void** addChild(Element \*child) {  
 childs.emplace\_back(child);  
 }  
  
 **void** setTerminal(**int** numberString) {  
 isTerminal = **true**;  
 Element::numberString = numberString;  
 }  
  
 **const** std::vector<Element \*> &getChilds() **const** {  
 **return** childs;  
 }  
  
 **char** getName() **const** {  
 **return** name;  
 }  
  
 Element \*getSufLink() **const** {  
 **return** sufLink;  
 }  
  
 Element \*getParent() **const** {  
 **return** parent;  
 }  
  
 **void** setSufLink(Element \*sufLink) {  
 Element::sufLink = sufLink;  
 }  
  
 **bool** isFirstElement1() **const** {  
 **return** isFirstElement;  
 }  
  
 **bool** isTerminal1() **const** {  
 **return** isTerminal;  
 }  
  
 **int** getNumberString() **const** {  
 **return** numberString;  
 }  
  
  
};  
  
  
Element \*createElement(Element \*parent, std::string \*string, **int** countElement, **int** numberString) {  
 **auto** \*element = **new** Element(string->at(countElement), parent);  
 **if** (countElement == string->length() - 1) {  
 element->setTerminal(numberString);  
 }  
 parent->addChild(element);  
 **return** element;  
}  
  
**void** addString(Element \*top, std::string \*string, **int** numberString) {  
 **int** count = 0;  
 Element \*cur = top;  
 **while** (count < string->length()) {  
 **char** name = cur->getName();  
 **bool** isFind = **false**;  
 std::vector<Element \*> childs = cur->getChilds();  
 **for** (**auto** child : childs) {  
 **if** (child->getName() == string->at(count)) {  
 cur = child;  
 count++;  
 isFind = **true**;  
 **if** (count == string->length()) {  
 cur->setTerminal(numberString);  
 }  
 **break**;  
 }  
 }  
 **if** (!isFind) {  
 cur = createElement(cur, string, count, numberString);  
 count++;  
 }  
 }  
}  
  
**bool** comp(std::pair<**int**, **int**> a, std::pair<**int**, **int**> b) {  
 **if** (a.first == b.first) {  
 **return** a.second < b.second;  
 } **else** {  
 **return** a.first < b.first;  
 }  
}  
  
**void** init(Element \*bor, std::vector<std::string> strings, std::string text,  
 std::vector<std::pair<**int**, **int**>> \*vectorAnswers, std::ofstream &stream) {  
 std::queue<Element \*> queue;  
 **for** (**auto** local : bor->getChilds()) {  
 local->setSufLink(bor);  
 **for** (**auto** local2 : local->getChilds()) {  
 queue.push(local2);  
 }  
 stream << local->getName() << **" in queue"** << std::endl;  
 }  
  
 **while** (!queue.empty()) {  
 **char** name = queue.front()->getName();  
 stream << name << **" in queue"** << std::endl;  
  
 Element \*element = queue.front()->getParent();  
  
 **bool** isExistX = **false**;  
  
 **while** (!isExistX) {  
 element = element->getSufLink();  
 **for** (**auto** child : element->getChilds()) {  
 **if** (child->getName() == name) {  
 isExistX = **true**;  
 element = child;  
 **break**;  
 }  
 }  
 **if** (element->isFirstElement1() && !isExistX) {  
 isExistX = **true**;  
 }  
  
 }  
 queue.front()->setSufLink(element);  
 **for** (**auto** child : queue.front()->getChilds()) {  
 queue.push(child);  
 }  
 queue.pop();  
 }  
  
 **int** count = 0;  
 Element \*cur = bor;  
 **while** (count < text.length()) {  
 **bool** isExistX = **false**;  
 **for** (**auto** child : cur->getChilds()) {  
 **if** (child->getName() == text[count]) {  
 cur = child;  
 isExistX = **true**;  
 count++;  
 **if** (cur->isTerminal1()) {  
 *//goSuffixLinkTerminal(cur->getSufLink(), &strings, count, &vectorAnswers);* **if** (cur->getChilds().empty()) {  
 vectorAnswers->emplace\_back(  
 std::make\_pair(count - strings.at(cur->getNumberString()).length() + 1,  
 cur->getNumberString()));  
 } **else** {  
  
 **bool** isNotFindNextChild = **true**;  
  
 **for** (**auto** child2 : cur->getChilds()) {  
 **if** (child2->getName() == text[count]) {  
 isNotFindNextChild = **false**;  
 **break**;  
 }  
 }  
  
 **if** (isNotFindNextChild) {  
 vectorAnswers->emplace\_back(  
 std::make\_pair(count - strings.at(cur->getNumberString()).length() + 1,  
 cur->getNumberString()));  
 }  
 }  
  
 }  
 **break**;  
 }  
 }  
  
 **if** (!isExistX) {  
 **if** (cur->isFirstElement1()) {  
 count++;  
 }  
 cur = cur->getSufLink();  
 }  
  
 }  
}  
  
**int** main() {  
 std::string text;  
 std::vector<std::string> strings;  
 strings.resize(0);  
 **int** countStrings = 0;  
  
 std::ifstream fin;  
 fin.open(**"C:\\Users\\Alex\\Desktop\\in.txt"**);  
  
 **int** your\_choose = 0;  
 std::cout << **"If you want to enter data from a file, enter \'1\'\n"**;  
 std::cin >> your\_choose;  
  
 **if** (your\_choose == 1) {  
 fin >> text;  
 fin >> countStrings;  
 } **else** {  
 std::cin >> text;  
 std::cin >> countStrings;  
 }  
  
 **for** (**int** i = 0; i < countStrings; ++i) {  
 std::string string;  
 **if** (your\_choose == 1) {  
 fin >> string;  
 } **else** {  
 std::cin >> string;  
 }  
 strings.emplace\_back(string);  
 }  
  
 fin.close();  
  
 **auto** \*bor = **new** Element;  
  
 **for** (**int** j = 0; j < countStrings; ++j) {  
 addString(bor, &strings[j], j);  
 }  
  
 std::vector<std::pair<**int**, **int**>> vectorAnswers;  
 vectorAnswers.resize(0);  
  
  
 std::ofstream ofstream(**"C:\\Users\\Alex\\Desktop\\out.txt"**);  
 **if** (your\_choose != 1) {  
 ofstream.basic\_ios<**char**>::rdbuf(std::cout.rdbuf());  
 }  
  
 init(bor, strings, text, &vectorAnswers, ofstream);  
  
 std::sort(vectorAnswers.begin(), vectorAnswers.end(), comp);  
  
 **for** (**auto** answer: vectorAnswers) {  
 std::cout << answer.first << **" "** << answer.second + 1 << std::endl;  
 }  
  
 **if**(your\_choose==1){  
 **for** (**auto** answer: vectorAnswers) {  
 ofstream<< answer.first << **" "** << answer.second + 1 << std::endl;  
 }  
 }  
  
 ofstream.close();  
  
 **return** 0;  
}