**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №6**

**по дисциплине «Построение и Анализ Алгоритмов»**

Тема: **Алгоритм Ахо-Корасик.**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент гр. 6304 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Ковынев М.В. | |
| Преподаватель: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Балтрашевич В.Э. |

Санкт-Петербург

2018

# **Цель работы.**

Освоить и закрепить приемы работы с алгоритмом Ахо-Корасик, практической задачей которого в данной лабораторной работе является поиск каждого шаблона из заданного множества шаблонов в тексте, а также поиск шаблона с символами-джокерами в тексте.

# **Постановка задачи.**

**Задача 1: Алгоритм Ахо-Корасик**

Разработайте программу, решающую задачу точного поиска набора образцов.

**Вход:**

* Первая строка содержит текст **(T, 1≤|T|≤100000)**.
* Вторая - число **n (1≤n≤3000)**, каждая следующая из n строк содержит шаблон из набора **P={p1,…,pn}1≤|pi|≤75**
* Все строки содержат символы из алфавита **{A,C,G,T,N}**

**Выход:**

* Все вхождения образцов из **P** в **T**.
* Каждое вхождение образца в текст представить в виде двух чисел - *i p*, где *i* - позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером *p* (нумерация образцов начинается с 1).
* Строки выхода должны быть отсортированы по возрастанию, сначала номера позиции, затем номера шаблона.

**Задача 2: Поиск с джокером**

Используя реализацию точного множественного поиска, решите задачу точного поиска для одного образца с джокером. В шаблоне встречается специальный символ, именуемого джокером (*wild card*), который "совпадает" с любым символом. По заданному содержащему шаблоны образцу **P** необходимо найти все вхождения **Р** в текст **Т**. Например, образец аb??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 неопределенной длины. В шаблоне входит хотя бы один символ не джокер, те шаблоны вида ???недопустимы. Все строки содержат символы из алфавита **{A,C,G,T,N}**

**Вход:**

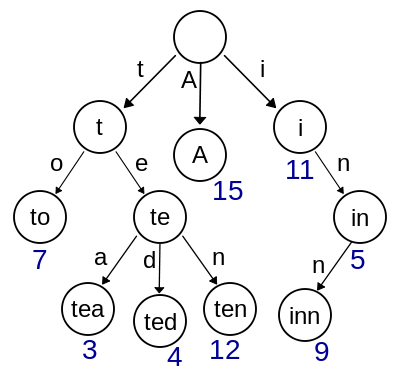
* Текст **(T, 1≤|T|≤100000 )**
* Шаблон **(P, 1≤|P|≤40)**
* Символ джокера

**Выход:**

* Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер).
* Номера должны выводиться в порядке возрастания.

# **Теоретические сведения**

**Бор** – структура данных для хранения набора строк, представляющая из себя [подвешенное дерево](https://neerc.ifmo.ru/wiki/index.php?title=%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE,_%D1%8D%D0%BA%D0%B2%D0%B8%D0%B2%D0%B0%D0%BB%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B5_%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) с символами на [рёбрах](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5_%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2). Строки получаются последовательной записью всех символов, хранящихся на [рёбрах](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5_%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2) между корнем бора и терминальной вершиной. Размер бора линейно зависит от суммы длин всех строк, а поиск в бору занимает время, пропорциональное длине образца. Пример бора для набора образцов {«A», «to», «tea», «ted», «ten», «i», «in», «inn»} представлен на рис. 1.

Рисунок 1 – Пример префиксного дерева

Алгоритм построения префиксного дерева.

**Алгоритм**

Непосредственно построение:

* Начало.
* **Шаг 1.** Создадим дерево из одной вершины (в нашем случае корня).
* **Шаг 2.** Добавление элементов в дерево.
  + Добавляем шаблоны Pi один за другим. Следуем из корня по рёбрам, отмеченным буквами из Pi, пока возможно.
  + Если Pi заканчивается в v, сохраняем идентификатор Pi (например, i) в v и отмечаем вершину v как терминальную.
  + Если ребра, отмеченного очередной буквой Pi нет, то создаем новое ребро и вершину для символа строки Pi.
* Конец.

Построение занимает, очевидно, *O(|P1|+…+|Pk|) =O(n)* времени, так как поиск буквы, по которой нужно переходить, происходит за  *O(1)*.

Поскольку на каждую вершину приходится O(|Σ|) памяти, то использование памяти есть O(n|Σ|).

# **Ход работы.**

1. Написана структура для вершины бора:

class Node

{

public:

// Конструктор класса Node

Node(int \_parent, wchar\_t \_symbol, size\_t size);

int parent; // Родитель (вершина)

wchar\_t symb; // Символ на ребре от parent к этой вершине

bool flag; // Проверка вершины на строку

int suffixLink; // Суффиксная ссылка

// Массив номеров вершин переходов по символу алфавита

std::vector<int> nextNode;

std::vector<int> indexPattern; // Номера подстрок в данной вершине

std::vector<int> autoMove; // Запоминание перехода автомата

};

1. Создан класс Bor в котором будет храниться бор, а также все методы и данные, необходимые для реализации алгоритма Ахо-Корасик:

class Bor

{

public:

// Конструктор класса

Bor();

// Поиск подстроки

void searchPatterns(const std::wstring& text);

// Добавление подстроки

void addPattern(const std::wstring& pattern);

// Вывод резлуьтат

void printResult() const;

std::vector<std::pair<int, int>> result; // Индексы вхождений найденных подстрок в тексте

std::vector<Node> nodesBor; // Вершины бора

std::vector<std::wstring> patterns; // Массив введенных подстрок

// Массив начaльных позиции подстрок в шаблоне с wildcards

std::vector<int> positionsPatternWildcard;

std::vector<int> vectorWildcards; // Массив для поиска строки с wildcards

std::vector<int> wildcardResult; // Результат поиска

alphabetMap alphabet;

private:

Node createNode(int parent, wchar\_t symbol);

int getSuffixLink(int currentNode);

int getAutoMove(int node, int character);

void checkIndex(int startNode, int index);

uint16\_t parseAlphabet(wchar\_t character);

};

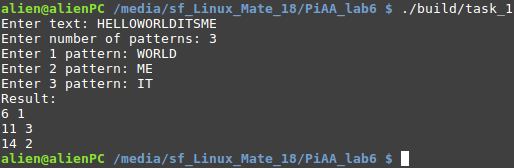
1. Задача нахождения шаблонов из заданного множества шаблонов в тексте предполагает заполнение бора этими шаблонами, а затем поиск вхождений этих шаблонов в текст. Результат работы представлен на рис. 2.

Рисунок 2 — Результат работы алгоритма Ахо-Корасик.

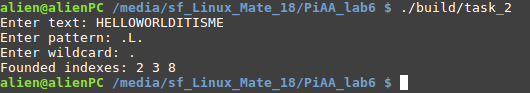
1. Задача нахождения шаблона с символами-джокерами в тексте предполагает разбиение шаблона по джокерам. Затем получившимися шаблонами заполняется бор, выполняется поиск вхождений этих шаблонов в текст. Далее происходит работа с массивом С, заполненным нулями в количестве text.size(): для каждого вхождения Pi в j-й позиции текста увеличивается счетчик C[j - li + 1] на единицу, где li – начальные позиции подстрок в исходном шаблоне. Вхождение исходного шаблона в текст, начинающееся в позиции p, имеется только в том случае, если С[p] = k, где k – количество подстрок, полученных парсингом исходного шаблона по джокеру. Результат работы представлен на рис. 3

Рисунок 3 — Результат работы поиска циклического сдвига.

1. Для сборки проекта был создан CMakeLists и скрипт для запуска сборки.
2. Реализованы параметризованные Gtest для проверки корректности результатов работы программ, решающих поставленные задачи.

# **Вывод.**

В ходе выполнения данной лабораторной работы изучен и реализован алгоритм Ахо-Корасик, посредством выполнения задач поиска шаблонов из заданного множества шаблонов в тексте, а также поиск шаблона с символами-джокерами.