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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Построение и анализ алгоритмов»
Тема: Ахо-Корасик

Студентка гр. 9383 _____ Карпекина А.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2021

Цель работы.

Ознакомиться с алгоритмом Ахо-Корасик, научиться использовать его для поиска набора образцов в тексте и шаблонной подстроки.

Задание 1

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст (T, $1 \leq |T| \leq 100000$).

Вторая - число n ($1 \leq n \leq 3000$), каждая следующая из n строк содержит шаблон из набора $P = \{p_1, \ldots, p_n\}$ $1 \leq |p_i| \leq 75$

Все строки содержат символы из алфавита $\{A,C,G,T,N\}$

Выход:

Все вхождения образцов из P

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.

Sample Input:

```
NTAG
3
TAGT
TAG
T
```

Sample Output:

2 2
2 3

Задание 2

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 P в текст T.

Например, образец *ab*??*c*? 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 $\{A,C,G,T,N\}$

Вход:

Текст (T, $1 \le |T| \le 100000$)

Шаблон (P, $1 \le |P| \le 40$)

Символ джокера

Выход:

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер).

Номера должны выводиться в порядке возрастания.

Sample Input:

ACTANCA
A$$A$

$

Sample Output:

1

Описание алгоритма

Бор - дерево, образованное последовательным добавлением всех образцов посимвольно. При добавлении символа создаётся вершина, если соответствующий подобразец ещё не добавлялся; иначе просто осуществляется переход к ранее созданной вершине. Изначально бор состоит из корня.

Автомат - это бор, дополненный суффиксными и конечными ссылками. Это дополнение может происходить заранее или во время обработки текста.

Суффиксная ссылка из вершины A - это ссылка на вершину, соответствующую максимально длинному (под)образцу в автомате, являющемуся несобственным суффиксом (под)образца A. Для корня и его сыновей суффиксная ссылка указывает на корень.

Конечная ссылка из вершины A - это ссылка на вершину, соответствующую максимально длинному образцу, который может быть получен при выполнении нескольких переходов по суфф. ссылкам, начиная с A. Если образцов получить нельзя, то конечная ссылка пуста.

Для построения **суффиксных ссылок** следует применять правило:

- Суффиксная ссылка из корня или из сына корня ведёт в корень.
- Для вычисления суффиксной ссылки вершины x нужно:

1. Перейти к вершине-родителю.

2. Пройти по суффиксным ссылкам минимальное число раз — но не менее 1 раза — чтобы появился путь по ребру x*, или до попадания в корень.

3. Пройти по ребру x. Если мы в корне и ребрах из него нет, то остаться в корне.

Полученная вершина и есть искомая суффиксная ссылка.

* Имя вершины совпадает с именем ребра бора, ведущего в эту вершину.

Для построения **конечной ссылки** (сжатой суффиксной ссылки) нужно переходить по суффиксным ссылкам до тех, пока не попадём в терминальную вершину (т. е. соответствующую образцу). Полученная вершина будет искомой конечной ссылкой. Если попали в корень, то конечная ссылка пуста.

Алгоритм Ахо-Корасик

1. Построить бор из образцов.

2. Построить автомат из бора (можно выполнять по ходу обработки текста).

3. Перейти в корень бора.

4. Посимвольная обработка текста. Для каждого символа:

4.1. Совершить переход в автомате из текущей вершины по рассматриваемому символу:

4.1a. Если есть соответствующее ребро, то перейти по нему;

4.16. Если нет, то:

4.1.6.a. Если находимся в корне, то ничего не делать.

4.1.6.6. Если находимся не в корне, то перейти по суффиксной ссылке и перейти кп. 4.1.

4.2. Добавить в результат вхождение образца, если попали в конечную вершину.

4.3. Обойти цепочку конечных ссылок до конца, сохраняя результаты.

Алгоритм поиска шаблонной подстроки

1. Построить автомат Ахо-Корасик из образцов, полученных выделением максимальных безджокерных подстрок из шаблонной подстроки.

2. Для каждого образца записать смещение (смещения), по которому (по которым) образец находится в шаблонной строке.

3. Инициализировать массив, заполненный нулями, длиной, совпадающей с текстом.

4. Выполнить поиск по тексту с использованием автомата. При обнаружении образца инкрементировать ячейку массива по адресу, образованному разностью номера начального символа образца в тексте и смещения образца. Если у образца несколько смещений, то инкрементировать все соответствующие ячейки массива.

В результате шаблонная подстрока будет начинаться в тех местах текста, для которых соответствующая ячейка массива содержит количество образцов с учётом кратности.

Функции и структуры данных

class TrieNode:

- int parent - вершина родитель
- char key - символ на ребре от parent к этой вершине
- int next_node[] - массив с номерами вершин по алфавиту
- int templates_number - номер строки-шаблона для вершины next_ver[i]
- bool flag - проверка на шаблон
- int suffix_link - суффиксная ссылка от вершины
- int movement[] - запоминание перехода автомата
- TrieNode(int, char) — конструктор класса, принимает вершину-родителя и значения ребра при переходе из родителя
- TrieNode() - конструктор по умолчанию
- ~TrieNode() - деструктор класса

class Trie:

std::vector <BohrVertex> bohr — вектор, хранящий вершины бора

int GetMovement(int node, int edge) — возвращает вершину, в которую совершен переход

int GetSuffixLink(int node); — возвращает вершину, в которую ведёт суффиксная ссылка из node

void Result(std::vector <std::string>& templates, int node, int symbol_number) — формирует результат поиска шаблонов в тексте

Trie() - конструктор

~Trie() - деструктор

void TrieAddNode(std::map<char, int> alphabet, int index, std::string sample) — добавляет строку - образец в бор

void MatchesSearch(std::vector <std::string> templates, std::string sample, std::map<char, int> alphabet) — осуществляет поиск совпадений строке

Для поиска в текст строки с джокером в классе Trie добавлены методы:

std::vector <int> Templates(std::map<char, int> alphabet, std::stringstream& template_string, char joker) — сопоставляет символы алфавита и его значения, строку и символ джокера, для разделения строки с джокером на подстроки для бора

void ResultPrint(const std::vector<int>& additional_array, int text_size, int length) — проверяет совпадения и выводит результат

Оценка сложности по памяти и времени

Пусть T - длина текста, в которой выполняется поиск, n - суммарная длина всех образцов, s - размер алфавита, k - общая длина всех вхождений образцов в текст.

Автомат хранится как индексный массив, сложность по времени - O(ns+T+k), по памяти - O(ns).

Тестирование

Результаты тестирования программ в таблице 1.

Таблица 1 - Тестирование программы

Входные данные	Выходные данные
NTAG 3 TAGT TAG T	2 2 2 3
ACACAC 2 AC CA	2 1 3 2 4 1 5 2 6 1
ACACAC 2 AC C	1 1 2 2 3 1 4 2 5 1 6 2
ACTANCA A$$A$ $	1
ACATN CT $	1
CAT A $	1

Выводы.

В ходе выполнения работы был изучен алгоритм Ахо-Корасик, а также реализована программа, осуществляющая поиск набора образцов в указанном тексте.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb5_1.cpp

```
#include <iostream>
#include <vector>
#include <cstring>
#include <map>
#include <algorithm>

std::vector <std::pair<int, int>> output;

class TrieNode
{
public:
	int parent;
	char key;
	int next_node[5];
	int templates_number = 0;
	bool flag = false;
	int suffix_link = -1;
	int movement[5];
	TrieNode(int, char);
	TrieNode() = default;
	~TrieNode() = default;
};

class Trie
{
	std::vector <TrieNode> trie;
	int GetMovement(int node, int edge);
	int GetSuffixLink(int node);
	void Result(std::vector <std::string>& templates, int node, int
symbol_number);

public:
	Trie();
	~Trie() = default;
	void TrieAddNode(std::map<char, int> alphabet, int index,
std::string sample);
```

```
	void MatchesSearch(std::vector <std::string> templates,
std::string sample, std::map<char, int> alphabet);

};

Trie::Trie()
{
	trie.push_back(TrieNode(0,0));
}

TrieNode::TrieNode(int parent, char symbol)
{
	for (int i = 0; i < 5; i++)
	{
		next_node[i] = -1;
		movement[i] = -1;
	}
	this->parent = parent;
	this->key = symbol;
}

void Trie::TrieAddNode(std::map<char, int> alphabet, int index,
std::string sample)
{
	int n = 0;
	int edge = 0;
	for (int i = 0; i < sample.length(); i++){
		char symbol = sample[i];
		edge = alphabet[symbol];
		if (trie[n].next_node[edge] == -1)
		{
			trie.push_back(TrieNode(n, edge));
			trie[n].next_node[edge] = trie.size() - 1;
		}
		n = trie[n].next_node[edge];
	}
	trie[n].flag = true;
```

```
      trie[n].templates_number = index;
}

int Trie::GetSuffixLink(int node)
{
      if (trie[node].suffix_link == -1)
            if (node == 0 || trie[node].parent == 0)
                  trie[node].suffix_link = 0;
      else
            trie[node].suffix_link =
GetMovement(GetSuffixLink(trie[node].parent), trie[node].key);
      return trie[node].suffix_link;
}

int Trie::GetMovement(int node, int edge)
{
      if (trie[node].movement[edge] == -1)
            if (trie[node].next_node[edge] != -1)
                  trie[node].movement[edge] =
trie[node].next_node[edge];
            else
                  if (node == 0)
                        trie[node].movement[edge] = 0;
                  else
                        trie[node].movement[edge] =
GetMovement(GetSuffixLink(node), edge);
      return trie[node].movement[edge];
}

void Trie::Result(std::vector <std::string>& templates, int node, int
symbol_number)
{
      for(int i = node; i != 0; i = GetSuffixLink(i))
            if (trie[i].flag)
                  output.push_back(std::make_pair(symbol_number -
templates[trie[i].templates_number].length() + 1,
trie[i].templates_number + 1));
```

```
}

void Trie::MatchesSearch(std::vector <std::string> templates,
std::string text, std::map<char, int> alphabet)
{
    int symbol_number = 0;
     int edge;
     for (int i = 0; i < text.length(); i++)
     {
          char symbol = text[i];
          edge = alphabet[symbol];
          symbol_number = GetMovement(symbol_number, edge);
          Result(templates, symbol_number, i + 1);
     }
}

int main()
{
     std::map<char, int> alphabet { {'A', 0}, {'C', 1}, {'G', 2},
{'T', 3}, {'N', 4}};
     Trie trie;
     std::vector <std::string> templates;
     std::string text;
     int templates_quantity;
     std::string template_string;
     std::cin >> text >> templates_quantity;
     for (int templates_number = 0; templates_number <
templates_quantity; templates_number++)
     {
          std::cin >> template_string;
          trie.TrieAddNode(alphabet, templates_number,
template_string);
          templates.push_back(template_string);
     }
     trie.MatchesSearch(templates, text, alphabet);
     sort(output.begin(), output.end());
```

```
	for (std::vector <std::pair<int, int>>::iterator iter =
output.begin(); iter!=output.end(); ++iter)
		if (true)
			std::cout<< (*iter).first << ' ' << (*iter).second <<
"\n";
	return 0;
}
```

Название файла: lb5_2.cpp

```
#include <iostream>
#include <vector>
#include <map>
#include <cstring>
#include <sstream>
#include <algorithm>

class TrieNode{
public:
	int parent;
	char key;
	int next_node[5];
	std::vector <int> templates_number;
	bool flag = false;
	int suffix_link = -1;
	int movement[5];
	TrieNode(int, char);
	TrieNode() = default;
	~TrieNode() = default;
};

class Trie{
```

```
	std::vector <TrieNode> trie;
	std::vector < std::string > template_strings;
	int GetMovement(int node, int edge);
	int GetSuffixLink(int node);

public:
	Trie();
	~Trie() = default;
	void TrieAddNode(std::map<char, int> alphabet, std::string
text);
	void MatchesSearch(std::map<char, int> alphabet, std::string
&s, std::vector <int> & additional_array, const std::vector <int> &
templates_length);
	void Result(int node, int i, std::vector <int>
&additional_array, std::vector <int> templates_length);
	std::vector <int> Templates(std::map<char, int> alphabet,
std::stringstream& template_string, char joker);
	void ResultPrint(const std::vector<int>& additional_array, int
text_size, int length);
};

Trie::Trie(){
	trie.push_back(TrieNode(0,0));
}

TrieNode::TrieNode(int parent, char symbol){
	for (int i = 0; i < 5; i++){
		next_node[i] = -1;
		movement[i] = -1;
	}
	this->parent = parent;
	this->key = symbol;
	templates_number.resize(0);
}

void Trie::TrieAddNode(std::map<char, int> alphabet, std::string
text){
```

```
      int new_node = 0;
      int edge = 0;
      for (int i = 0; i < text.length(); i++){
            char symbol = text[i];
            edge = alphabet[symbol];
            if (trie[new_node].next_node[edge] == -1){
                  trie.push_back(TrieNode(new_node, edge));
                  trie[new_node].next_node[edge] = trie.size() - 1;
            }
            new_node = trie[new_node].next_node[edge];
      }
      trie[new_node].flag = true;
      template_strings.push_back(text);

trie[new_node].templates_number.push_back(template_strings.size() -
1);

}

int Trie::GetSuffixLink(int node){
      if (trie[node].suffix_link == -1)
            if (node == 0 || trie[node].parent == 0)
                  trie[node].suffix_link = 0;
      else
            trie[node].suffix_link =
GetMovement(GetSuffixLink(trie[node].parent), trie[node].key);
      return trie[node].suffix_link;
}

int Trie::GetMovement(int node, int edge){
      if (trie[node].movement[edge] == -1)
            if (trie[node].next_node[edge] != -1)
                  trie[node].movement[edge] =
trie[node].next_node[edge];
            else
                  if (node == 0)
                        trie[node].movement[edge] = 0;
```

```
                else
                        trie[node].movement[edge] =
GetMovement(GetSuffixLink(node), edge);
      return trie[node].movement[edge];
}

void Trie::Result(int node, int i, std::vector <int>&
additional_array, std::vector <int> templates_length){
       for(int u = node; u != 0; u = GetSuffixLink(u))
            if (trie[u].flag){
                  for (const auto &j : trie[u].templates_number)
                        if ((i - templates_length[j] <
additional_array.size()))
                              additional_array[i -
templates_length[j]]++;
            }
}

std::vector <int> Trie::Templates(std::map<char, int> alphabet,
std::stringstream& template_string, char joker){
      std::vector < int > templates_length;
      int length = 0;
      std::string storage;
      while (getline(template_string, storage, joker)){
            if (storage.size() > 0){
                  length += storage.size();
                  templates_length.push_back(length);
                  TrieAddNode(alphabet, storage);
            }
            length++;
      }
      return templates_length;
}

void Trie::ResultPrint(const std::vector<int>& additional_array, int
text_size, int length){
      for (int i = 0; i < text_size; i++)
```

```
            if ((additional_array[i] == template_strings.size()) && (i
+ length <= text_size)){
                  std::cout << i + 1 << "\n";
            }

}

void Trie::MatchesSearch(std::map<char, int> alphabet, std::string
&text, std::vector <int> & additional_array, const std::vector <int>
& templates_length){
      int edge;
      int u = 0;
      int lenght = text.length();
      for (int i = 0; i < lenght; i++){
            char symbol = text[i];
            edge = alphabet[symbol];
            u = GetMovement(u, edge);
            Result(u, i + 1, additional_array, templates_length);
      }
}

int main(){
      std::map<char, int> alphabet { {'A', 0}, {'C', 1}, {'G', 2},
{'T', 3}, {'N', 4}};
      Trie trie;
      std::string text, template_string;
      char joker;
      std::cin >> text >> template_string >> joker;
      std::stringstream str_stream(template_string);
      std::vector <int> templates_length = trie.Templates(alphabet,
str_stream, joker);
      std::vector <int> additional_array(text.length(), 0);
      trie.MatchesSearch(alphabet, text, additional_array,
templates_length);
      trie.ResultPrint(additional_array, text.size(),
template_string.length());
}
```