**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: А**хо-Корасик**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1384 |  | Галенко А.С. |
| Преподаватель |  | Шевелева А.М. |

Санкт-Петербург

2023

**Задание 1.**

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.

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 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 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≤∣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.

Sample Input:

ACTANCA

A$$A$

$

Sample Output:

1

**Выполнение работы.**

Для решения первого задания был описан класс Aho\_Corasick\_tree, предоставляющий следующий интерфейс, позволяющий решить первую задачу:

* *void addStringtoBor(string line)*: Позволяет добавить строку в бор
* *Node getSuffixLink(Node \*cur\_node)*: Возвращает указатель элемента в векторе вершин бора, являющейся суффиксной ссылкой элемента *cur\_node*.
* *Node getTerminalSuffixLink(Node \*cur\_node)*: Возвращает индекс «сжатой» (терминальной) суффиксной ссылки, т.е. такой, на которой заканчивается какое-либо слово.
* *Node\* transit\_across\_sym(Node\* cur\_node, char sym)* Возвращает указатель элемента, соответствующий вершине, связанной с вершиной *node* ребром со значением *symb*. Если таковой нет, возвращается указатель элемента, связанного с вершиной суффиксной ссылкой *node* ребром со значением *symb*.
* *void find\_samples(std::string &text, std::map<std::string, int> &patterns)*: Находит и сохраняет индексы вхождения сохранённых ранее строк в *text*. Map нужен для сохранения в очередь с приоритетом, которая в выводе даст отсортированный по индексу каждого паттерна список.
* void printRes(): Выводит список (ответ), для этого в цикле извлекается максимальный элемент структуры Text\_Entry, и выводятся его поля на экран.

После ввода данных в main все строки-паттерны добавляются в бор, после чего вызывается *findSamples* для поданного текста. Список с ответом сортируется согласно условиям задачи и выводится его содержимое с помощью printRes().

Для решения второй задачи были добавлены следующие функции в класс:

* void convertPatternToBor(std::string &pattern, const char &wildcard) : Находит подстроки каждого паттерна, разделенные символом джокера (wildcard), и добавляет их в бор.
* void findAnswer(const std::string &text, const std::string &pattern) : функция итогового поиска шаблона в строке text

Был модифицирован метод *addStringtoBor* класса – теперь в классе есть массив bor\_substrings, который сохраняет пары: указатель на вершину и индекс начала подстроки в шаблоне. При создании терминальной вершины в данный массив заносится соответствующая информация.

**Выводы.**

В процессе выполнения лабораторной работы был реализован алгоритм Ахо-Корасик. Были разработаны программы для точного поиска набора образцов в строке, а также для поиска всех вхождений строки, содержащей символ-джокер в тексте.

Приложение 1

КОД ДЛЯ ПЕРВОЙ ЗАДАЧИ

task1.cpp:

#include <iostream>

#include <map>

#include <vector>

#include <queue>

/\*\*

\* Структура Bor\_Node определяет вершину в Боре,

\* is\_terminal - флаг, является ли вершина терминалом

\* terminal\_line - вся строка, которая сохраняется в конечной вершине,

\* parent\_node - вершина-родитель,

\* suffix\_pointer - суффиксная ссылка,

\* terminal\_suffix\_pointer - сжатая суффиксная ссылка,

\* parent\_symbol - символ строки в данной вершине,

\* sons\_nodes - вектор дочерних вершины,

\* transitions\_array - вектор переходов (запоминаем переходы в ленивой рекурсии),

\* используемый для вычисления суффиксных ссылок.

\*/

struct Node

{

bool is\_terminal = false;

std::string terminal\_line;

Node \*parent\_node = nullptr;

Node \*suffix\_pointer = nullptr;

Node \*terminal\_suffix\_pointer = nullptr;

char parent\_symb;

std::vector<Node\*> sons\_nodes;

std::vector<Node\*> transitions\_array;

};

struct Text\_Entry

{

Text\_Entry(int pos, int num)

: pos\_index(pos) , list\_number(num) {}

int pos\_index;

int list\_number;

};

struct less\_Text\_Entry

{

bool operator() (const Text\_Entry \*first, const Text\_Entry \*second) const

{

if(first->pos\_index != second->pos\_index)

return first->pos\_index > second->pos\_index;

else

return first->list\_number > second->list\_number;

}

};

class Aho\_Corasick\_tree

{

std::vector<char> alphabet;

std::priority\_queue<Text\_Entry\*, std::vector<Text\_Entry\*>, less\_Text\_Entry> answer\_priority\_queue;

Node \*root;

int getAlphabetIndex(char sym)

{

for(int index = 0; index < alphabet.size(); index++)

{

if(alphabet[index] == sym)

return index;

}

return -1;

}

public:

Aho\_Corasick\_tree(std::vector<char> alphabet)

: alphabet(alphabet)

{

root = new Node;

root->transitions\_array.resize(alphabet.size());

root->sons\_nodes.resize(alphabet.size());

}

Node \*getSuffixLink(Node \*cur\_node)

{

if(!cur\_node->suffix\_pointer) {

if (cur\_node == root or cur\_node->parent\_node == root)

cur\_node->suffix\_pointer = root;

else

cur\_node->suffix\_pointer = transit\_across\_sym(getSuffixLink(cur\_node->parent\_node),

cur\_node->parent\_symb);

}

return cur\_node->suffix\_pointer;

}

Node \*getTerminalSuffixLink(Node \*cur\_node)

{

if(!cur\_node->terminal\_suffix\_pointer)

{

Node \*cur\_suffix = getSuffixLink(cur\_node);

if(cur\_suffix == root)

cur\_node->terminal\_suffix\_pointer = root;

else

cur\_node->terminal\_suffix\_pointer = (cur\_suffix->is\_terminal)

? cur\_suffix : getTerminalSuffixLink(cur\_suffix);

}

return cur\_node->terminal\_suffix\_pointer;

}

Node\* transit\_across\_sym(Node\* cur\_node, char sym)

{

auto &cur\_transitions = cur\_node->transitions\_array;

int sym\_index = getAlphabetIndex(sym);

if(!cur\_transitions[sym\_index])

{

if(cur\_node->sons\_nodes[sym\_index])

cur\_transitions[sym\_index] = cur\_node->sons\_nodes[sym\_index];

else if(cur\_node == root)

cur\_transitions[sym\_index] = root;

else

cur\_transitions[sym\_index] = transit\_across\_sym(getSuffixLink(cur\_node), sym);

}

return cur\_transitions[sym\_index];

}

void addStringtoTree(std::string line)

{

Node \*cur\_node = root;

for(auto symb : line)

{

int symb\_index = getAlphabetIndex(symb);

auto &cur\_sons = cur\_node->sons\_nodes;

if(!cur\_sons[symb\_index])

{

cur\_sons[symb\_index] = new Node;

cur\_sons[symb\_index]->parent\_node = cur\_node;

cur\_sons[symb\_index]->parent\_symb = symb;

cur\_sons[symb\_index]->transitions\_array.resize(alphabet.size());

cur\_sons[symb\_index]->sons\_nodes.resize(alphabet.size());

}

cur\_node = cur\_sons[symb\_index];

}

cur\_node->is\_terminal = true;

cur\_node->terminal\_line = line;

}

void find\_samples(std::string &text, std::map<std::string, int> &patterns)

{

Node \*cur\_node = root;

for(int index = 0; index < text.size(); index++)

{

cur\_node = transit\_across\_sym(cur\_node, text[index]);

auto i\_node = cur\_node;

while (i\_node != root)

{

if(i\_node->is\_terminal)

answer\_priority\_queue.push(new Text\_Entry(index + 2 - i\_node->terminal\_line.size(),

patterns[i\_node->terminal\_line]));

i\_node = getTerminalSuffixLink(i\_node);

}

}

}

void printRes()

{

while(!answer\_priority\_queue.empty())

{

Text\_Entry \*cur\_entry = answer\_priority\_queue.top();

std::cout << cur\_entry->pos\_index << ' '

<< cur\_entry->list\_number << '\n';

answer\_priority\_queue.pop();

}

}

};

int main()

{

Aho\_Corasick\_tree solver\_tree({'A', 'C', 'G', 'T', 'N'});

std::string text, temp\_line;

std::map<std::string, int> patterns;

int pattern\_num;

std::cin >> text;

std::cin >> pattern\_num;

for(int i = 0; i < pattern\_num; i++)

{

std::cin >> temp\_line;

solver\_tree.addStringtoTree(temp\_line);

patterns[temp\_line] = i + 1;

}

solver\_tree.find\_samples(text, patterns);

solver\_tree.printRes();

return 0;

}

Приложение 2

КОД ДЛЯ ВТОРОЙ ЗАДАЧИ

task2.cpp:

#include <iostream>

#include <vector>

#include <algorithm>

/\*\*

\* Структура Bor\_Node определяет вершину в Боре,

\* is\_terninal - флаг, является ли вершина терминалом

\* terminal\_line - вся строка, которая сохраняется в конечной вершине,

\* parent\_node - вершина-родитель,

\* suffix\_pointer - суффиксная ссылка,

\* terminal\_suffix\_pointer - сжатая суффиксная ссылка,

\* symbol - символ строки в данной вершине,

\* text\_position - вектор для хранения найденных индексов

\* вхождения строки (терминальной вершины) в тексте,

\* sons\_nodes - вектор дочерних вершины,

\* transitions\_array - вектор переходов (запоминаем переходы в ленивой рекурсии),

\* используемый для вычисления суффиксных ссылок.

\*/

struct Bor\_Node

{

bool is\_terminal = false;

std::string terminal\_line;

Bor\_Node \*parent\_node = nullptr;

Bor\_Node \*suffix\_pointer = nullptr;

Bor\_Node \*terminal\_suffix\_pointer = nullptr;

char symbol;

std::vector<int> text\_positions;

std::vector<Bor\_Node\*> sons\_nodes;

std::vector<Bor\_Node\*> transitions\_array;

};

/\*\*

\*

\*/

class Aho\_Corasick\_tree

{

// алфавит текста и шаблона

std::vector<char> alphabet;

// вектор пар <подстрока шаблона в Боре> - <стартовая позиция в шаблоне>

std::vector<std::pair<Bor\_Node\*, int>> bor\_substrings;

// вектор для определения стартовых позиций шаблона в тексте

std::vector<int> search\_array;

// корневая вершина Бора

Bor\_Node \*root;

/\*\*

\* Функция, рассчитывающая индекс в массиве sonds\_nodes или transitions\_array

\* для символа из алфавита

\* @param sym - символ, который нужно перевести в индекс

\* @return -1, если символа нет в алфавите, в ином случае - найденную позицию в алфавите

\*/

int getAlphabetIndex(char sym)

{

for(int index = 0; index < alphabet.size(); index++)

{

if(alphabet[index] == sym)

return index;

}

return -1;

}

public:

/\*\*

\* Конструктор класса, решающего задачу поиска шаблонов с масками

\* @param alphabet - принимает в аргументы алфавит текста и паттернов

\*/

explicit Aho\_Corasick\_tree(const std::vector<char>& alphabet)

: alphabet(alphabet)

{

root = new Bor\_Node;

root->transitions\_array.resize(alphabet.size());

root->sons\_nodes.resize(alphabet.size());

}

/\*\*

\* Функция для поиска суффиксной ссылки

\* @param cur\_node - верщина бора, для которой нужно найти

\* суффиксную ссылку

\* @return указатель на найденную вершину

\*/

Bor\_Node \*getSuffixLink(Bor\_Node \*cur\_node)

{

if(!cur\_node->suffix\_pointer) {

if (cur\_node == root or cur\_node->parent\_node == root)

cur\_node->suffix\_pointer = root;

else

cur\_node->suffix\_pointer = transit\_across\_sym(getSuffixLink(cur\_node->parent\_node),

cur\_node->symbol);

}

return cur\_node->suffix\_pointer;

}

/\*\*

\* Рекурсивная функция для поиска сжатой суффиксной ссылки

\* @param cur\_node верщина бора, для которой нужно найти

\* такую суффиксную ссылку

\* @return указатель на найденную вершину

\*/

Bor\_Node \*getTerminalSuffixLink(Bor\_Node \*cur\_node)

{

if(!cur\_node->terminal\_suffix\_pointer)

{

Bor\_Node \*cur\_suffix = getSuffixLink(cur\_node);

if(cur\_suffix == root)

cur\_node->terminal\_suffix\_pointer = root;

else

cur\_node->terminal\_suffix\_pointer = (cur\_suffix->is\_terminal)

? cur\_suffix : getTerminalSuffixLink(cur\_suffix);

}

return cur\_node->terminal\_suffix\_pointer;

}

/\*\*

\* Функция для вычисления перехода между вершинами Бора

\* @param cur\_node - вершина, для которой нужно найти переход по символу

\* @param sym - символ перехода

\* @return указатель на найденную вершину

\*/

Bor\_Node\* transit\_across\_sym(Bor\_Node\* cur\_node, const char sym)

{

auto &cur\_transitions = cur\_node->transitions\_array;

int sym\_index = getAlphabetIndex(sym);

if(!cur\_transitions[sym\_index])

{

if(cur\_node->sons\_nodes[sym\_index])

cur\_transitions[sym\_index] = cur\_node->sons\_nodes[sym\_index];

else if(cur\_node == root)

cur\_transitions[sym\_index] = root;

else

cur\_transitions[sym\_index] = transit\_across\_sym(getSuffixLink(cur\_node), sym);

}

return cur\_transitions[sym\_index];

}

/\*\*

\* Функция для поиска подстрок, которые разделены

\* символом wildcard в pattern. Найденные строки добавляются в Бор

\* @param pattern - шаблон поиска

\* @param wildcard - символ джокера

\*/

void convertPatternToBor(std::string &pattern, const char &wildcard)

{

std::string substring;

int index;

auto position = [&] () -> int { return int(index - substring.length() + 1); };

for(index = 0; index < pattern.size(); index++)

{

if(pattern[index] != wildcard)

substring.push\_back(pattern[index]);

else if(!substring.empty())

{

addStringToTree(substring, position());

substring.clear();

}

}

if(!substring.empty())

addStringToTree(substring, position());

}

/\*\*

\* Функция для добавления строки line в Бор

\* @param line - строка, которую нужно добавить

\* @param pattern\_position - индекс начала line в строке шаблона,

\* который необходимо сохранить в конечной вершине бора

\*/

void addStringToTree(std::string &line, int pattern\_position)

{

Bor\_Node \*cur\_node = root;

for(int index = 0; index < line.size(); index++)

{

int symb\_index = getAlphabetIndex(line[index]);

auto &cur\_sons = cur\_node->sons\_nodes;

if(!cur\_sons[symb\_index])

{

cur\_sons[symb\_index] = new Bor\_Node;

cur\_sons[symb\_index]->parent\_node = cur\_node;

cur\_sons[symb\_index]->symbol = line[index];

cur\_sons[symb\_index]->transitions\_array.resize(alphabet.size());

cur\_sons[symb\_index]->sons\_nodes.resize(alphabet.size());

}

cur\_node = cur\_sons[symb\_index];

}

cur\_node->is\_terminal = true;

cur\_node->terminal\_line = line;

bor\_substrings.push\_back(std::make\_pair(cur\_node, pattern\_position));

}

/\*\*

\* Функция для поиска подстрок из Бора в

\* строке text

\* @param text - строка, по которой производится поиск

\*/

void find\_samples(const std::string &text)

{

Bor\_Node \*cur\_node = root;

int finded\_index;

for(int index = 0; index < text.size(); index++)

{

cur\_node = transit\_across\_sym(cur\_node, text[index]);

auto i\_node = cur\_node;

while (i\_node != root)

{

if(i\_node->is\_terminal)

{

finded\_index = index - i\_node->terminal\_line.length() + 2;

i\_node->text\_positions.push\_back(finded\_index);

}

i\_node = getTerminalSuffixLink(i\_node);

}

}

}

/\*\*

\* Функция итогового поиска шаблона в text

\* @param text - строка, по которой производится поиск

\* @param pattern - шаблон

\*/

void findAnswer(const std::string &text, const std::string &pattern)

{

find\_samples(text);

search\_array.resize(text.length());

int substrings\_count = (int) bor\_substrings.size();

std::fill(search\_array.begin(), search\_array.end(), 0);

for(auto cur\_subsring\_node : bor\_substrings)

{

for(auto cur\_text\_position : cur\_subsring\_node.first->text\_positions)

{

int index = cur\_text\_position - cur\_subsring\_node.second + 1;

if(index >= 0 and text.length() >= pattern.length() + index - 1)

search\_array[index]++;

}

}

for(int index = 0; index < search\_array.size(); index++)

{

if(search\_array[index] == substrings\_count)

std::cout << index << "\n";

}

}

};

int main()

{

Aho\_Corasick\_tree solver\_tree({'A', 'C', 'G', 'T', 'N'});

std::string text, pattern;

char joker;

std::cin >> text;

std::cin >> pattern;

std::cin >> joker;

solver\_tree.convertPatternToBor(pattern, joker);

solver\_tree.findAnswer(text, pattern);

return 0;

}