**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: **АлгоритмАхо-Корасик**

|  |  |  |
| --- | --- | --- |
| Студентгр. 8304 |  | Чешуин Д.И. |
| Преподаватель |  | Размочаева Н.В. |

Санкт-Петербург

2020

**Цель работы.**

Изучить алгоритм Ахо-Корасик и алгоритм поиска вхождений шаблонов с “джокерами” в строку. Написать программу, реализующую эти алгоритмы работы со строками.

**Постановка задачи.**

**Вариант 3.** Вычислить длину самой длинной цепочки из суффиксных ссылок и самой длинной цепочки из конечных ссылок в автомате.

**Алгоритм Ахо-Корасик**

**Задание.**

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.

**Описание алгоритма.**

В начале алгоритма бор заполняется символами шаблонов. Для этого поочередно обрабатывается каждый символ шаблона. Если перехода в боре ля текущей вершины нет, то вершина создается, добавляется в бор и в нее совершается переход по текущему символу. Если вершина с переходом по текущему символу уже существует, то в нее совершается переход.

Далее осуществляется поиск шаблонов в текстовой строке. Для этого обрабатывается автомат, полученный из созданного бора путем добавления суффиксных ссылок.

Обрабатывается текущий символ текстовой строки. Если в автомате уже существует ребро-переход по символу в вершину, то осуществляется переход в эту вершину. Если ребра-перехода в автомате еще нет, но существует переход по текущему символу в вершину-сына, то этот переход осуществляется и добавляется в ребра автомата. Если такого перехода также не существует, то переход осуществляется по суффиксной ссылке и также заносится в ребра автомата.

Для нахождения суффиксной ссылки для вершины, осуществляется переход в предка вершины, затем переход по суффиксной ссылке предка и переход по текущему символу. Если предок не имеет суффиксной ссылки, то для него она определяется аналогичным образом рекурсивно.

Если во время перехода в автомате встречается терминальная вершина, это означает, что шаблон в подстроке найден. Вычисляется индекс его в строке и заносится в вектор результата.

**Анализ алгоритма.**

Таблица переходов автомата хранится в структуре std::unordered\_map, которая реализована как хэш-таблица. Тогда сложность алгоритма по операциям будет равна O(M+N) M–длина всех символов слов шаблонов, N – длина текста, в котором осуществляется поиск.

Сложность алгоритма по памяти: O (M+N), M – длина всех символов слов шаблонов, N – длина текста, в котором осуществляется поиск.

**Алгоритм поиска шаблона с “джокером”.**

**Задание.**

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 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 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.

**Описание алгоритма.**

В начале работы алгоритма считывается шаблон, поиск которого будет осуществляться. Этот шаблон разделяется функцией на подшаблоны, которые были разделены друг от друга символом джокера в строке-шаблоне. Также запоминаются индексы этих подшаблонов в строке-шаблоне для дальнейшей работы алгоритма.

Далее с помощью алгоритма Ахо-Корасик подшаблоны заносятся в бор и осуществляется их поиск в строке. Когда подшаблон находится в строке поиска, то инкрементируется значение, находящееся в ключе хэш-таблицы совпадений подшаблонов. Этот ключ определяется как индекс вхождения подшаблона в строку минус индекс подшаблона в строке-шаблоне.

После того, как вся строка поиска будет обработана и все подшаблоны найдены, то проверяются значения вектора вхождения подшаблонов. Если в каком-либо ключе этой хэш-таблицы хранится число, равное количеству всех подшаблонов шаблона, значит строка-шаблон входит в строку поиска на этом индексе полностью. Индекс вхождения этого шаблона запоминается и заносится в вектор результата.

**Анализ алгоритма.**

Сложность по операциям аналогична алгоритму Ахо-Корасик – O(M + N).

Сложность по памяти - O (2\*M+N+W), M –длина всех символов слов шаблона, N – длина текста, в котором осуществляется поиск, W – количество подшаблонов.

**Описание функций и СД.**

Для решения задачи был реализован класс Automat, класс JokerFinder и структура отдельной вершины бора – Node. UML диаграмму использованных структур данных смотри на рисунке 1.

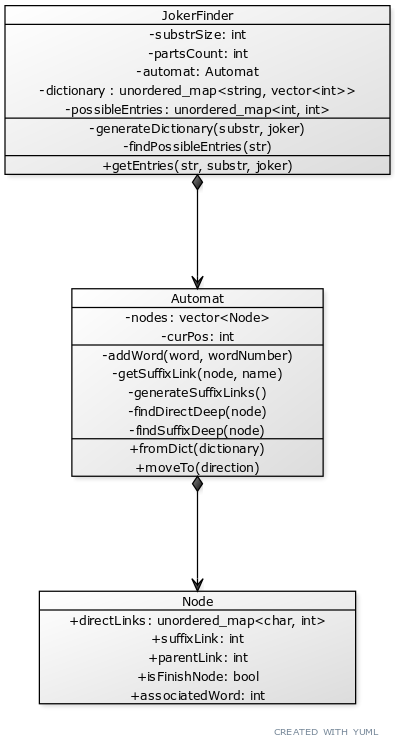


Рисунок 1 – UML диаграмма использованных структур данных

Метод поиска пути:

void fromDict(const vector<string>& dictionary);

Принимает словарь и генерирует соответствующий ему автомат.

vector<int> moveTo(char direction);

Метод принимает символ направления и переводит автомат в соответствующее состояние. Возвращает вектор вхождений строк из переданного словаря.

vector<int> getEntries(const string& str, const string& substr, char joker); Метод принимает текст, шаблон и символ-джокер и возвращает вектор всех вхождений шаблона в текст.

**Тестирование.**

Пример работы вывода программы для поиска вхождений по словарю на входных данных: текст – NTAG, словарь – TAGT, TAG, T, - смотри на рисунке 2.

Пример работы вывода программы для поиска вхождений шаблона на входных данных: текст – ACTANCA, шаблон – A$$A$, джокер - $, - смотри на рисунке 3.

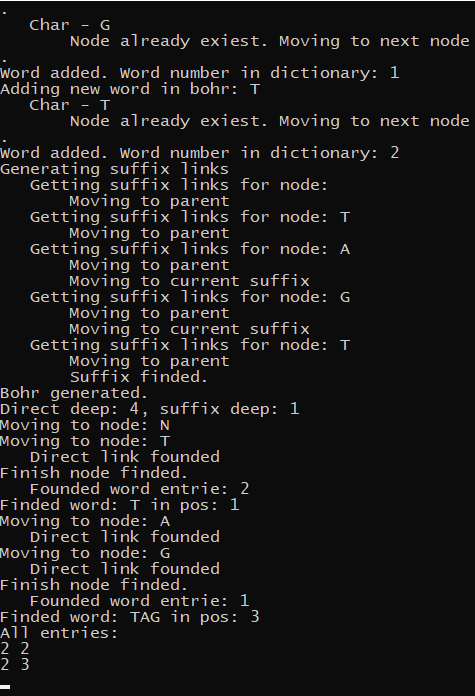


Рисунок 2 – Пример вывода для поиска вхождений по словарю

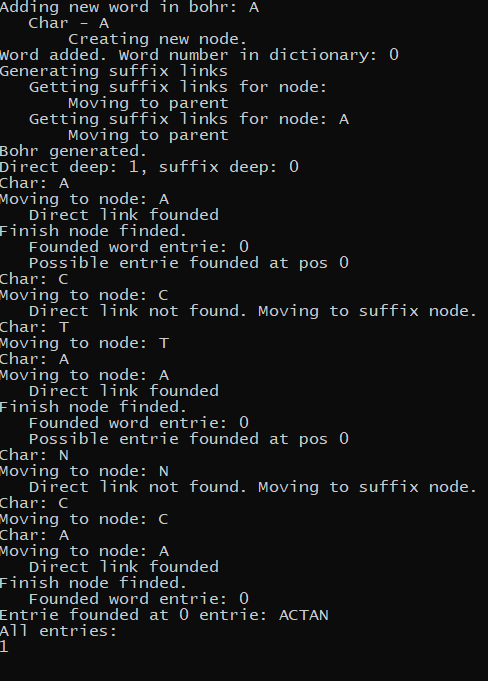


Рисунок 3 – Пример вывода для поиска вхождений по шаблону

**Выводы.**

В ходе выполнения лабораторной работы были получены навыки работы с алгоритмом Ахо-Корасик и алгоритмом поиска подстроки с “джокером”. Были написаны программы, реализующую эти алгоритмы работы со строками.

**ПРИЛОЖЕНИЕ A.  
ИСХОДНЫЙ КОД ПРОГРАММЫ**

main.cpp.

#include <iostream>

#include <vector>

#include <queue>

#include <fstream>

#include <cstring>

#include <algorithm>

#include <unordered\_map>

using namespace std;

class **IOManager**

{

private:

static istream\* input;

static ostream\* output;

public:

static void **setStreamsFromArgs**(int argc, char\*\* argv)

{

if(argc > 1)

{

for(int i = 1; i < argc; i++)

{

if(strcmp(argv[i], "-infile") == 0)

{

if(i + 1 < argc)

{

input = new ifstream(argv[i + 1]);

i += 1;

}

}

if(strcmp(argv[i], "-outfile") == 0)

{

if(i + 1 < argc)

{

output = new ofstream(argv[i + 1]);

i += 1;

}

}

}

}

}

static istream& **getIS**()

{

return \*input;

}

static ostream& **getOS**()

{

return \*output;

}

static void **resetStreams**()

{

if(input != & IOManager::getIS())

{

delete input;

input = &IOManager::getIS();

}

if(output != & IOManager::getOS())

{

delete output;

output = &IOManager::getOS();

}

}

};

class **Automat**

{

private:

struct **Node**

{

unordered\_map<char, int> directLinks;

int suffixLink = 0;

int parentLink = 0;

bool isFinishNode = false;

int associatedWord = -1;

};

private:

vector<Node> nodes;

int currPos = 0;

private:

void **addWord**(const string& word, int wordNumber);

int **getSuffixLink**(int node, char name);

void **generateSuffixLinks**();

unsigned **findDirectDeep**(int node);

unsigned **findSuffixDeep**(int node);

public:

**Automat**()

{

//добавление корня

nodes.push\_back(Node());

}

void **fromDict**(const vector<string>& dictionary);

vector<int> **moveTo**(char direction);

};

class **JokerFinder**

{

private:

int substrSize = 0;

int partsCount = 0;

Automat automat;

unordered\_map<string, vector<int>> dictionary;

unordered\_map<int, int> possibleEntries;

private:

void **generateDictionary**(const string& substr, char joker);

void **findPossibleEntries**(const string& str);

public:

vector<int> **getEntries**(const string& str, const string& substr, char joker);

};

void **dictionary**();

void **joker**();

int **main**(int argc, char\*\* argv)

{

IOManager::setStreamsFromArgs(argc, argv);

string buf;

int choice = 0;

IOManager::getOS() << "Enter 1 if you want to find any word from dictionary." << endl;

IOManager::getOS() << "Enter 2 if you want to find joker" << endl;

IOManager::getIS() >> choice;

getline(IOManager*::getIS()*, *buf*);

if(choice == 1)

{

dictionary();

}

else if(choice == 2)

{

joker();

}

return 0;

}

void **dictionary**()

{

IOManager::getOS() << "Search from dictionary." << endl;

Automat bohr;

string str, buf;

unsigned count = 0;

vector<string> words;

vector<pair<int, int>> entries;

IOManager::getOS() << "Enter main string.." << endl;

getline(IOManager*::getIS()*, *str*);

IOManager::getOS() << "Enter word count." << endl;

IOManager::getIS() >> count;

getline(IOManager*::getIS()*, *buf*);

IOManager::getOS() << "Enter words." << endl;

for(unsigned i = 0; i < count; i++)

{

string word;

getline(IOManager*::getIS()*, *word*);

words.push\_back(word);

}

bohr.fromDict(words);

for(unsigned i = 0; i < str.size(); i++)

{

auto entrie = bohr.moveTo(str[i]);

if(!entrie.empty())

{

for(unsigned j = 0; j < entrie.size(); j++)

{

IOManager::getOS() << "Finded word: " << words[entrie[j]] << " in pos: " << i << endl;

entries.push\_back(make\_pair(i - words[entrie[j]].size() + 1, *entrie[j]*));

}

}

}

sort(entries.begin(), entries.end());

IOManager::getOS() << "All entries:" << endl;

for(auto entrie : entries)

{

IOManager::getOS() << entrie.first + 1 << " " << entrie.second + 1 << endl;

}

}

void **joker**()

{

IOManager::getOS() << "Joker search." << endl;

Automat ahoCorasic;

JokerFinder alg;

string str;

string substr;

char joker;

IOManager::getOS() << "Enter main string." << endl;

getline(IOManager*::getIS()*, *str*);

IOManager::getOS() << "Enter substring." << endl;

getline(IOManager*::getIS()*, *substr*);

IOManager::getOS() << "Enter joker." << endl;

IOManager::getIS() >> joker;

auto entries = alg.getEntries(str, substr, joker);

IOManager::getOS() << "All entries:" << endl;

for(auto entrie : entries)

{

IOManager::getOS() << entrie + 1 << endl;

}

}

void Automat::**addWord**(const string& word, int wordNumber)

{

IOManager::getOS() << "Adding new word in bohr: " << word << endl;

if(word.length())

{

int currNode = 0;

for(unsigned i = 0; i < word.length(); i++)

{

IOManager::getOS() << " Char - " << word[i] << endl;

if(nodes[currNode].directLinks.find(word[i]) == nodes[currNode].directLinks.end())

{

IOManager::getOS() << " Creating new node." << endl;

nodes.push\_back(Node());

nodes[currNode].directLinks.emplace(word[i], nodes.size() - 1);

nodes[nodes.size() - 1].parentLink = currNode;

currNode = nodes.size() - 1;

}

else

{

IOManager::getOS() << " Node already exiest. Moving to next node." << endl;

currNode = nodes[currNode].directLinks.find(word[i])->second;

}

}

IOManager::getOS() << "Word added. Word number in dictionary: " << wordNumber << endl;

if(nodes[currNode].associatedWord == -1)

{

nodes[currNode].associatedWord = wordNumber;

nodes[currNode].isFinishNode = true;

}

}

}

void Automat::**generateSuffixLinks**()

{

IOManager::getOS() << "Generating suffix links" << endl;

//очеред из пар: вершина - суффикс

queue<pair<char, int>> open;

open.push(make\_pair('\0', 0));

while(!open.empty())

{

auto current = open.front();

open.pop();

for(auto link : nodes[current.second].directLinks)

{

open.push(link);

}

nodes[current.second].suffixLink = getSuffixLink(current.second, current.first);

//если суффиксная ссылка указывает на конечную вершину автомата, то данная вершина так же будет конечной

if(nodes[nodes[current.second].suffixLink].isFinishNode)

{

nodes[current.second].isFinishNode = true;

}

}

}

int Automat::**getSuffixLink**(int node, char name)

{

IOManager::getOS() << " Getting suffix links for node: " << name << endl;

int currNode = nodes[node].parentLink;

int currSuffix = nodes[currNode].suffixLink;

IOManager::getOS() << " Moving to parent" << endl;

while(currNode != 0)

{

auto findedNode = nodes[currSuffix].directLinks.find(name);

if(findedNode != nodes[currSuffix].directLinks.end())

{

IOManager::getOS() << " Suffix finded." << endl;

return findedNode->second;

}

else

{

IOManager::getOS() << " Moving to current suffix" << endl;

currNode = currSuffix;

currSuffix = nodes[currNode].suffixLink;

}

}

return 0;

}

void Automat::**fromDict**(const vector<string>& dictionary)

{

IOManager::getOS() << "Generating bohr from dictionary." << endl;

for(unsigned i = 0; i < dictionary.size(); i++)

{

addWord(dictionary[i], i);

}

generateSuffixLinks();

IOManager::getOS() << "Bohr generated." << endl;

IOManager::getOS() << "Direct deep: " << findDirectDeep(0) <<", suffix deep: " << findSuffixDeep(0) << endl;

}

vector<int> Automat::**moveTo**(char direction)

{

vector<int> entries;

IOManager::getOS() << "Moving to node: " << direction << endl;

auto findedNode = nodes[currPos].directLinks.find(direction);

while(findedNode == nodes[currPos].directLinks.end() && currPos != 0)

{

IOManager::getOS() << " Direct link not found. Moving to suffix node." << endl;

currPos = nodes[currPos].suffixLink;

findedNode = nodes[currPos].directLinks.find(direction);

}

if(findedNode != nodes[currPos].directLinks.end())

{

IOManager::getOS() << " Direct link founded" << endl;

currPos = nodes[currPos].directLinks.find(direction)->second;

}

int entriePos = currPos;

while(nodes[entriePos].isFinishNode)

{

IOManager::getOS() << "Finish node finded." << endl;

if(nodes[entriePos].associatedWord != - 1)

{

IOManager::getOS() << " Founded word entrie: " << nodes[entriePos].associatedWord << endl;

entries.push\_back(nodes[entriePos].associatedWord);

}

entriePos = nodes[entriePos].suffixLink;

}

return entries;

}

void JokerFinder::**generateDictionary**(const string& substr, char joker)

{

IOManager::getOS() << "Generating dictionary from joker." << endl;

IOManager::getOS() << "Joker: " << substr << " char: " << joker << endl;

string word = "";

for(unsigned offset = 0; offset < substr.size(); offset++)

{

if(substr[offset] != joker)

{

word += substr[offset];

}

if(substr[offset + 1] == joker || offset + 1 >= substr.size())

{

if(!word.empty())

{

auto finded = dictionary.find(word);

if(finded == dictionary.end())

{

vector<int> offsets;

offsets.push\_back(offset - word.size() + 1);

dictionary.emplace(*word*, *offsets*);

partsCount += 1;

}

else

{

finded->second.push\_back(offset - word.size() + 1);

partsCount += 1;

}

IOManager::getOS() << " Part: " << word << " offset: " << offset - word.size() + 1 << endl;

word = "";

}

}

}

}

void JokerFinder::**findPossibleEntries**(const string& str)

{

vector<string> dict;

IOManager::getOS() << "Finding joker entries." << endl;

for(auto pair : dictionary)

{

dict.push\_back(pair.first);

}

automat.fromDict(dict);

for(unsigned i = 0; i < str.size(); i++)

{

IOManager::getOS() << "Char: " << str[i] << endl;

for(auto entrie : automat.moveTo(str[i]))

{

for(auto offset : dictionary.find(dict[entrie])->second)

{

int entriePos = i - dict[entrie].size() - offset + 1;

if(entriePos >= 0 && entriePos + substrSize <= static\_cast<int>(str.size()))

{

IOManager::getOS() << " Possible entrie founded at pos " << entriePos << endl;

auto possibility = possibleEntries.find(entriePos);

if(possibility != possibleEntries.end())

{

possibility->second += 1;

}

else

{

possibleEntries.emplace(*entriePos*, 1);

}

}

}

}

}

}

vector<int> JokerFinder::**getEntries**(const string& str, const string& substr, char joker)

{

vector<int> entries;

substrSize = substr.size();

generateDictionary(substr, joker);

findPossibleEntries(str);

for(auto entrie : possibleEntries)

{

if(entrie.second == partsCount)

{

IOManager::getOS() << "Entrie founded at " << entrie.first << " entrie: " << str.substr(entrie.first, substrSize) << endl;

entries.push\_back(entrie.first);

}

}

sort(entries.begin(), entries.end());

return entries;

}

unsigned Automat::**findDirectDeep**(int node)

{

unsigned deep = 0;

if(nodes[node].directLinks.empty())

{

return 0;

}

else

{

for(auto link : nodes[node].directLinks)

{

unsigned buf = findDirectDeep(link.second);

if(buf > deep)

{

deep = buf;

}

}

return deep + 1;

}

}

unsigned Automat::**findSuffixDeep**(int node)

{ unsigned deep = 0;

if(node == 0)

{

for(unsigned i = 1; i < nodes.size(); i++)

{

unsigned buf = findSuffixDeep(i);

if(buf > deep)

{

deep = buf;

}

}

}

else

{

if(nodes[node].suffixLink == 0)

{

return 0;

}

else

{

return findSuffixDeep(nodes[node].suffixLink) + 1;

}

}

return deep;

}

istream\* IOManager::input = &cin;

ostream\* IOManager::output = &cout;