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

ОТЧЕТ

по практической работе №5
по дисциплине «Построение и Анализ Алгоритмов»
Тема: Алгоритм Ахо-Корасик

Студент гр. 8382 _____ Вербин К.М.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Научиться реализовывать алгоритм Ахо-Корасик и реализовать с его помощью программу для поиска вхождений шаблонов в текст и поиска вхождений в текст шаблона с джокером .

Задание 1

Разработайте программу, решающую задачу точного поиска набора образцов

Входные данные

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 $P = \{p1, \ldots, pn\} 1 \leq |pi| \leq 75$

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

Выходные данные

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.

Пример входных данных

NTAG

3

TAGT

TAG

T

Соответствующие выходные данные

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 PP необходимо найти все вхождения PP в текст TT. 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}

Входные данные

Текст (T, $1 \leq |T| \leq 100000$)

Шаблон (P, $1 \leq |P| \leq 40$)

Символ джокера

Выходные данные

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

Пример входных данных

ACTANCA

A$$A$

$

Соответствующие выходные данные

1

Вариант 3

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

Описание алгоритма

алгоритма Ахо-Корасика

На вход принимаются текст, количество шаблонов и сами шаблоны. По всем шаблонам строится бор, состоящий из экземпляров класса TNode, хранящих ссылку на родителя и потомков. Вершина бора для последнего символа каждого шаблона отмечается как терминальная. Сама структура хранится в классе Trie. Затем строится автомат, где состояния - вершины бора. Строятся суффиксные ссылки, для каждой вершины v - это ссылка на вершину в боре, в которой оканчивается наидлиннейший собственный суффикс строки, соответствующей вершине v. Суффиксная ссылка находится следующим образом: происходит переход в вершину по суффиксной ссылке родителя, а затем из нее совершается переход по заданному символу. После построения автомата начитается обработка теста на поиск вхождений шаблонов. Для каждого символа вызывается функция поиска потомка: если из текущей вершины есть потомок с рассматриваемым символом, переходим в него, иначе переходим по суффикс-ссылке к другой вершине и ищем потомка от нее. Если символ в боре отсутствует, текущей вершиной становится корень. После проверки символа, функция возвращает вектор с номерами шаблонов, входящими в найденный шаблон, который добавляется в вектор со всеми найденными индексами вхождения. Далее они выводятся на экран. Для индивидуализации в функции нахождения потомка в боре в момент поиска шаблонов в найденной подстроке, при каждом переходе по суффиксной

ссылке увеличивается счетчик количества суффиксных ссылок. Для подсчета максимального количества конечных ссылок счетчик увеличивается только в том случае, если встреченная вершина является терминальной.

Описание структур данных

Класс TNode - содержит элементы бора и функции для работы с ними

- char symbol - символ, которому соответствует вершина
- unordered_map sons - ассоциативный массив для хранения потомков вершины.
- TNode* parent - указатель на предка вершины.
- TNode* suffLink - суффиксная ссылка.
- string str - подстрока, которой соответствует
- int terminated - номер шаблона, концу которого соответствует вершина или 0, если вершина не является терминальной.
- explicit TNode(char c): symbol(c), terminated(0){} - конструктор класса.
- void insert(string temp, int numPattern) - функция для вставки шаблона в бор.
- vector getChain(char c, int *maxSuffLen, int *maxEndLen) - функция для поиска в боре потомка по символу.
- void makeSuffixLinks() - функция для построения суффиксных ссылок в боре. void printTrie(TNode* root) - функция для печати информации об элементах бора.

Класс Trie - обертка над классом TNode для работы с бором.

- TNode node - корень бора.
- int maxSuffLen - максимальная длина цепочки суффиксных ссылок(для индивидуализации)

- int maxEndLen - максимальная длина цепочки конечных ссылок(для индивидуализации).
- void printMaxLenghts() - функция для вывода максимальных длин цепочек суффиксных и конечных ссылок.
- TNode* getRoot() - функция получения указателя на корень бора для вывода элементов бора. о Остальные методы аналогичны методам класса TNode.

Сложность алгоритма по памяти

Каждый символ шаблона представляет собой вершину бора, поэтому сложность по памяти составляет O(n). Т.к. на каждой позиции в тексте могут встретиться все p шаблонов, полная сложность по памяти составляет O(n + m * p).

Временная сложность алгоритма

Построение бора имеет линейную сложность O(n), где n - сумма длин паттернов. Построение суффиксных ссылок реализуется через обход в ширину со сложностью O(V + E), где E – кол-во ребер, V - кол-во вершин, эту сложность можно адаптировать для бора: O(n + n) = O(2n) = O(n). Перебор символов текста в боре занимает O(m), где m - длина текста. В конечном итоге сложность алгоритма составляет O(2n + m) = O(n + m).

Алгоритм поиска подстрок с джокером

Построение бора и суффиксных ссылок реализуется так же, как в первом алгоритме, за исключением того, что бор состоит не из шаблонов, а из подстрок данного шаблона без джокеров {P1 , P2, …, Pk}. При вставке их в бор, в терминальной вершине в векторе сохраняется позиция начала шаблона и его размер. Затем запускается поиск для каждого символа. Появление подстроки Pi означает возможное появление шаблона на позиции j – i + 1, где

j - текущая позиция в тексте, i - позиция начала подстроки в маске. Затем в дополнительном векторе подсчитываются такие позиции, и если в ячейке количество вхождений равно k, значит там было вхождение шаблона.

Сложность алгоритма по памяти

Сложность по памяти, как и в первом алгоритме, составляет O(n+m*p).

Временная сложность алгоритма

В случае с поиском шаблона с джокером построение бора будет иметь сложность O(h), где h - сумма длин подстрок. Построение суффиксных ссылок так же, как и в первом алгоритме имеет сложность O(h). Прохождение текста по бору также составляет O(n), где n - длина текста. Не учитывается прохождение дополнительно вектора, т.к. его длина равна длине исходного текста. В конечном итоге сложность алгоритма составляет O(2h + n) = O(2h + n)

Тестирование

Input	Output
asdfasfaslkas 4 as fas fasf a	Max suffix link chain lenght: 2 Max end link chain lenght: 2 Index Pattern 1 1 1 4 4 2 4 3 5 1 5 4 7 2 8 1 8 4

	12 1 12 4
asdfasdased 3 asd asdf as	Max suffix link chain lenght: 1 Max end link chain lenght: 1 Index Pattern 1 1 1 2 1 3 5 1 5 3 8 3
actataaha ax x	Max suffix link chain lenght: 1 Max end link chain lenght: 1 Index Pattern 1 4 6 7
xabvccbababcax ab??c? ?	Max suffix link chain lenght: 1 Max end link chain lenght: 1 Index Pattern 2 8

Вывод

Было получено теоретическое представление об алгоритме Кнута-МоррисаПратта и на основе него были реализованы программы для поиска вхождений подстроки и поиска циклического сдвига строки.

ПРИЛОЖЕНИЕ

КОД ПРОГРАММЫ

алгоритма Ахо-Корасика

```
#include <vector>
#include <fstream>
#include <map>
#include <unordered_map>
#include <queue>
#include <algorithm>

using namespace std;
void inputConsole(string *word, int *n ,vector<string> *templates);

class TNode{
private:
    char symbol;
    unordered_map<char, TNode*> sons;
    TNode* parent = nullptr;
    TNode* suffLink = nullptr;
    string str = "";
    int terminated;

public:
    explicit TNode(char c): symbol(c), terminated(0){}

    void insert(string temp, int numPattern){
        TNode* curr = this;

        for (char symbol: temp) {
            if (curr->sons[symbol] == nullptr) {
```

```
                curr->sons[symbol] = new TNode(symbol);   // создаем нового ребенка
                curr->sons[symbol]->parent = curr;
                curr->sons[symbol]->str = curr->str + symbol;
            }
            curr = curr->sons[symbol];
        }
        cout << "Insert pattern: " << temp << endl;
        curr->terminated = numPattern + 1;
    }


    //  поиск символа префикса в боре и всех вхождений шаблонов в его путь
    vector<int> getChain(char c, int *maxSuffLen, int *maxEndLen){
        vector<int> templatesInside;
        int currSuffLen = 0;
        int currEndLen = 0;
        static const TNode* curr = this;

        for (; curr != nullptr ; curr = curr->suffLink) {
            for (auto son: curr->sons) {
                cout << "Child: " << son.first << endl;
                if(son.first == c) {
                    cout << "This is the child they were looking for! \n";
                    curr = son.second;
                    for (auto node = curr; node->suffLink != nullptr; node = node->suffLink, currSuffLen++)
                        if(node->terminated > 0) {
                            currEndLen++;
```

```
                        cout << "A terminal vertex '"<<node->symbol <<"' was found for the template " << node->terminated << endl;
                        templatesInside.push_back(node->terminated);
                    }
                *maxSuffLen = (*maxSuffLen < currSuffLen) ? currSuffLen : *maxSuffLen;
                *maxEndLen = (*maxEndLen < currEndLen) ? currEndLen : *maxEndLen;
                cout << "Current max suffix link chain lenght: " << *maxSuffLen << endl;
                cout << "Current max end link chain lenght: " << *maxEndLen << endl;
                return templatesInside;
            }
        }
    }
    curr = this;
    cout << "The end of the template for this position was not found.\n";
    return templatesInside;
}


// функция для построения суффиксных ссылок
void makeSuffixLinks(){


    queue<TNode*> q;
    for (auto son: sons) {     // можно внести это в цикл
        q.push(son.second);
    }
```

```
            while(!q.empty()) {
                TNode* curr = q.front();  //  берем вершину из очереди для обработки
                cout << "Considered vertex: " << curr->symbol << " Substring: " << curr->str << endl;
                for(pair<const char, TNode *> son: curr->sons) {
                   q.push(son.second);
                }
                q.pop();

                TNode* par = curr->parent;
                if(par != nullptr)    // переходим по суфф. ссылке предыдущей вершины
                   par = par->suffLink;


                while(par && par->sons.find(curr->symbol) == par->sons.end()) //проверка, есть ли нужный символ
                {

                   par = par->suffLink;                  // в потомках рассматриваемой вершины,
                }                                        // если нет, то переходим по суфф ссылке

                if(par)  {curr->suffLink = par->sons[curr->symbol]; cout << " Suffix link: " << curr->suffLink->symbol << " Substring: " << curr->suffLink->str << endl;}
                   //  присваиваем суффиксную ссылку, если она найдена
                else curr->suffLink = this;      //  иначе присваиваем ссылку в себя
```

```
        cout << endl;
    }

}


void printTrie(TNode* root){
    TNode* curr = root;

    cout <<"\nString: " << curr->str << endl;
    if(curr->terminated > 0)
        cout <<"--->Terminated!" << "\n";

    if(curr->parent) {
        if (curr->parent->symbol != '\0') {
            cout << "    Symbol: " << curr->symbol << "    ";
            cout << "    Parent: " << curr->parent->symbol << endl;
        }
        if (curr->parent->symbol == '\0') {
            cout << "    Symbol: " << "'\\0' ";
            cout << "    Parent: root" << endl;
        }
    }
    else
        cout << "    Root" << endl;
    if(curr->suffLink)
        cout << "    Suffix link: " << curr->suffLink->str << endl;

    cout << "    Children: ";
    if(curr->sons.size() > 0) {
```

```
            for (auto c:curr->sons) {
                cout << c.first << " ";
            }
            cout << endl;
        }
        else cout << " none \n";
        for(auto tmp:curr->sons) {
            if (tmp.second) {
                printTrie(tmp.second);
            }
        }
    }
};

class Trie{
private:
    TNode node;
    int maxSuffLen;
    int maxEndLen;
public:
    Trie(): node('\0'), maxSuffLen(0), maxEndLen(0) {}

    void printMaxLenghts(){
        cout << "\nMax suffix link chain lenght: " << maxSuffLen << endl;
        cout << "Max end link chain lenght: " << maxEndLen << endl;
    }
    TNode* getRoot(){
        return &node;
    }
    vector<int> getChain(char c){
```

```
        return node.getChain(c, &maxSuffLen, &maxEndLen);
    }
    void makeSuffixLinks(){
        node.makeSuffixLinks();
    }
    void insert(string temp, int numPattern){
        node.insert(temp, numPattern);
    }
};


int main() {
    string str;
    int n;
    map<int, vector<int>> res;
    vector<string> templates(10);
//    inputConsole(&str, &n, &templates);
//----------------------------------------
    cin >> str >> n;
    templates.resize(n);
    for (int i = 0; i < n; ++i) { cin >> templates[i]; }
    //----------------------------------------

    Trie root;
    // построение бора
    cout << "\nStarted the construction of the Trie... \n";
    for (int j = 0; j < n; ++j) {
        root.insert(templates[j], j);
    }
    cout << "------------------------- Suffix links ----------------------------\n";
```

```
    cout << "\nThe process of creating suffix links...\n";
    root.makeSuffixLinks();
    cout << "------------------------Built the trie ----------------------------\n";
    root.getRoot()->printTrie(root.getRoot());

    cout << "----------------------- Substring search ----------------------------\n";
    for (int i = 0; i < str.length(); ++i) {
        cout << "\nSymbol: " << str[i] << "  Index: " << i << endl;
        vector<int> tmp = root.getChain(str[i]);
        for (auto index: tmp) {
            res[i - templates[index - 1].size() + 2].push_back(index);
            sort(res[i - templates[index - 1].size() + 2].begin(),
                 res[i - templates[index - 1].size() + 2].end());
        }
    }
    root.printMaxLenghts();

    cout << "Index Pattern\n";
    for (auto it: res) {
        for (auto k: it.second) {
            cout << "" << it.first << "      " <<  k << endl;


        }
    }
    return 0;
}

void inputConsole(string *word,int *n ,vector<string> *templates){
    ifstream file;
    file.open("input.txt");
```

```
    if (file.is_open()) {
        file >> *word >> *n;
        templates->resize(*n);
        for (int i = 0; i < *n; ++i) {
            file >> (*templates)[i];
        }
        file.close();
    } else {
        cout << "File isn't open!";
    }
}
```

Алгоритм поиска подстрок с джокером

```
#include <iostream>
#include <vector>
#include <fstream>
#include <map>
#include <unordered_map>
#include <queue>
#include <algorithm>

using namespace std;
char inputConsole(string *str, string *templates);

class TNode{
private:
    char symbol;
    unordered_map<char, TNode*> sons;
    TNode* parent = nullptr;
    TNode* suffLink = nullptr;
    string str = "";
```

```
        int terminated;
        vector<pair<int, int>> arrayPatterns;

    public:
        explicit TNode(char c): symbol(c), terminated(0){}

        void insert(const string &temp, int pos, int size) {
            TNode* curr = this;
            for (char symbol: temp) {
                if (curr->sons[symbol] == nullptr) {
                    curr->sons[symbol] = new TNode(symbol);   // создаем нового ребенка
                    curr->sons[symbol]->parent = curr;
                    curr->sons[symbol]->str = curr->str + symbol;
                }
                curr = curr->sons[symbol];
            }
            cout << "Insert substring: " << temp << endl;
            curr->terminated = 1;
            curr->arrayPatterns.emplace_back(pos, size); // сохраняем все вставленные маски в вектор
        }                                       // pos - позиция начала в шаблоне
                                                // size - длина маски


        //  поиск символа префикса в боре и всех вхождений шаблонов в его путь
        vector<pair<int, int>> getChain(char c, int *maxSuffLen, int *maxEndLen) {
            vector <pair<int, int>> templatesInside;
```

```
            int currSuffLen = 0;
            int currEndLen = 0;
            static const TNode* curr = this;

            for (; curr != nullptr ; curr = curr->suffLink) {
               for (auto son: curr->sons) {
                  cout << "Child: " << son.first << endl;
                  if(son.first == c) {
                     cout << "This is the child they were looking for! \n";
                     curr = son.second;
                     for (auto node = curr; node->suffLink != nullptr; node = node->suffLink, currSuffLen++) {
                        if(node->terminated > 0)
                           currEndLen++;
                        for (auto it: node->arrayPatterns) {
                           templatesInside.push_back(it);
                        }
                     }
                     *maxSuffLen = (*maxSuffLen < currSuffLen) ? currSuffLen : *maxSuffLen;
                     *maxEndLen = (*maxEndLen < currEndLen) ? currEndLen : *maxEndLen;
                     cout << "Current max suffix link chain lenght: " << *maxSuffLen << endl;
                     cout << "Current max end link chain lenght: " << *maxEndLen << endl;
                     return templatesInside;
                  }
               }
            }
```

```
            curr = this;
            return {};
        }


        // функция для построения суффиксных ссылок
        void makeSuffixLinks(){

            queue<TNode*> q;
            for (auto son: sons) {     // можно внести это в цикл
                q.push(son.second);
            }
            while(!q.empty()) {
                TNode* curr = q.front();   //  берем вершину из очереди для обработки
                cout << "Considered vertex: " << curr->symbol << " Substring: " << curr->str << endl;
                for(pair<const char, TNode *> son: curr->sons)
                    q.push(son.second);
                q.pop();

                TNode* par = curr->parent;
                if(par != nullptr)    // переходим по суфф. ссылке предыдущей вершины
                    par = par->suffLink;


                while(par && par->sons.find(curr->symbol) == par->sons.end()) //проверка, есть ли нужный символ
```

```
                par = par->suffLink;                    // в потомках рассматриваемой
вершины,
                                                        // если нет, то переходим по суфф
ссылке
                if(par)  {curr->suffLink = par->sons[curr->symbol]; cout << " Suffix
link: " << curr->suffLink->symbol << " Substring: " << curr->suffLink->str <<
endl;}
                                                        //  присваиваем суффиксную ссылку,
если она найдена
                else curr->suffLink = this;      //  иначе присваиваем ссылку в себя
            }

        }

        void printTrie(TNode* root){
            TNode* curr = root;
            cout <<"\nString:" << curr->str << endl;
            if(curr->terminated > 0)
                cout <<"--->Terminated!" << "\n";

            if(curr->parent && curr->parent->symbol != '\0') {
                cout <<"    Symbol:" <<curr->symbol << "\n";
                cout <<"    Parent:" << curr->parent->symbol << endl;
            }
            else if(curr->parent && curr->parent->symbol == '\0')
                cout <<"    Parent: root" << endl;
            else
                cout <<"    Root" << endl;

            if(curr->suffLink)
```

```
            cout << "    Suffix link: " << curr->suffLink->str << endl;

        cout << "    Children:";
        if(curr->sons.size() > 0) {
            for (auto c:curr->sons) {
                cout << c.first << " ";
            }
            cout << endl;
        }
        else cout << " none \n";
        for(auto tmp:curr->sons) {
            if (tmp.second) {
                printTrie(tmp.second);
            }
        }
    }
};

class Trie{
private:
    TNode node;
    int maxSuffLen;
    int maxEndLen;
public:
    Trie(): node('\0'), maxSuffLen(0), maxEndLen(0){}

    void printMaxLenghts(){
        cout << "\nMax suffix link chain lenght: " << maxSuffLen << endl;
        cout << "Max end link chain lenght: " << maxEndLen << endl;
    }
```

```
        TNode* getRoot(){ return &node; }

        vector<pair<int, int>> getChain(char c){ return node.getChain(c, &maxSuffLen, &maxEndLen); }

        void makeSuffixLinks(){  node.makeSuffixLinks(); }

        void insert(const string &temp, int pos, int size){ node.insert(temp, pos, size); }
    };

    int main() {
        string str;
        char joker;
        string pattern;
    //    joker = inputConsole(&str, &pattern);
    //----------------------------------------
        cin >> str >> pattern >> joker;
        //----------------------------------------
        int freePart = 0;  // счетчик подстрок в шаблоне без джокера
        Trie root;
        string mask;
        vector <size_t> maskEnter(str.size());   // массив флагов попадания подстроки в текст
        char c;
        cout << "\nStarted the construction of the Trie... \n";
        // построение бора
        for (size_t i = 0; i <= pattern.size(); i++)
        {
```

```
            if(i == pattern.size())  // если встречен конец строки, присваиваем с значение джокера,
                                     // чтобы вставить оставшуюся маску в бор
                c = joker;
            else c = pattern[i];

            if (c != joker) {       // накапливаем маску без джокеров
                mask += c;
            }
            else if (!mask.empty()) { // если встречен джокер и подстрока непустая, добавляем маску в бор
                freePart++;
                root.insert(mask, i - mask.size(), mask.size());
                mask.clear();
            }
        }
        cout << "-------------------------- Suffix links ----------------------------\n";
        cout << "\nThe process of creating suffix links...\n";
        root.makeSuffixLinks();

        cout << "------------------------Built the trie ----------------------------\n";
        root.getRoot()->printTrie(root.getRoot());
        cout << "------------------------ Substring search ----------------------------\n";
        for (int j = 0; j < str.size(); j++) {
            cout << "\nSymbol: " << str[j] << "  Index: " << j << endl;
            vector<pair<int, int>> tmp = root.getChain(str[j]);
            for (auto pos: tmp) {                   // добавление всех вхождений масок
                int i = j - pos.first - pos.second + 1;
                if (i >= 0 && i + pattern.size() <= str.size())  // отмечаем вхождение, если границы шаблона
```

```
                maskEnter[i]++;                    // входят в границы текста
            }
        }

        root.printMaxLenghts();

        cout << "Index Pattern\n";
        for (int i = 0; i < maskEnter.size(); i++)
            if (maskEnter[i] == freePart){
                cout << i+1 << endl;     // печать индексов вхождения шаблона
            }
        return 0;
    }

    char inputConsole(string *str, string *templates) {
        ifstream file;
        char joker;
        file.open("input.txt");
        if (file.is_open()) {
            file >> *str >> templates[0] >> joker;
            file.close();
        } else {
            cout << "File isn't open!";
        }
        return joker;
    }
```