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: Алгоритм Ахо-Корасик

Студент гр. 8383 _____ Мирсков А.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы

Изучить алгоритм Ахо-Корасик. Решить с его помощью задачу точного поиска набора образцов и задачу точного поиска для одного образца с джокером.

Постановка задачи

Вар. 5. Вычислить максимальное количество дуг, исходящих из одной вершины в боре; вы́резать из строки поиска все найденные образцы и вывести остаток строки поиска.

Задача 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
```

Задача 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
```

Реализация

На вход программе подается текст, количество шаблонов и сами шаблоны. По шаблонам строится бор, который хранится в классе Vertex. Класс содержит ссылку на предка и ссылки на сыновей. Вершины, отвечающие за конец каждой из строк отмечаются терминальными.

Далее начинается обработка текста и одновременно с этим построение префиксного автомата на основе получившегося бора. Каждый символ текста производит переход в новое состояние автомата и вызывает функцию вычисления суффиксной ссылки для этого состояния. Вычисление происходит следующим образом: осуществляется переход по суффиксной ссылке предка и далее обычный переход из вершины, в которую пришел алгоритм. После каждого перехода вызывается функция, которая формирует ответ с помощью быстрых ссылок. Она проверяет является ли вершина терминальной и если является производит спуск в стартовую вершину по быстрым ссылкам, добавляя к ответу все терминальные вершины, в которые попадет.

После завершения алгоритма начинается выполнение индивидуализации. Вызывается поиск в глубину по бору, который запоминает максимальное количество исходящих дуг. Далее происходит вырезание из текста найденных подстрок и вывод остатка на экран.

Для поиска вхождений одного образца с джокером образец делится на подстроки, разделяясь джокерами. Далее на основе полученных строк строится автомат по тому же алгоритму, что и в предыдущей задаче, но при проходе по быстрым ссылкам к стартовому состоянию при попадании в терминальную вершину, увеличивается на единицу изначально заведенный и заполненный нулями массив, по индексу, который вычисляется как индекс начала вхождения подстроки образца в текст минус стартовая позиция подстроки образца в образец плюс один.

Сложность алгоритма

Сложность по времени построения автомата O(n), где n — сумма длин образцов. Сложность обработки символов текста O(m), где m — длина строки.
Итоговая сложность O(n+m).

Сложность по памяти — O(k*n), где k — размер алфавита, т. к. в автомате не больше n вершин и для каждой вершины нужно хранить k переходов.

Для поиска одного образца с джокером сложность по времени построения автомата O(p), где p — сумма длин вырезанных подстрок. Сложность обработки символов текста O(m), где m — длина строки.
Итоговая сложность O(p+m).

Сложность по памяти — O(k*p + m), где k — размер алфавита, m — длина текста, т. к. в автомате не больше p вершин и для каждой вершины нужно хранить k переходов и требуется хранить массив C длины m.

Описание функций и структур данных

Множественный поиск

Код программы представлен в приложении А. Он содержит следующие функции и структуры данных.

void addString(Vertex* suffAuto, std::string& s, int ind)
Добавляет строку s в автомат suffAuto.

void read(std::string& text, Vertex* suffAuto, std::vector<int>* arr)
Считывает данные с консоли.

void getAnswer(std::string& text, Vertex* suffAuto, std::vector<std::pair<int,int>>* answer)
Вычисляет индексы вхождений образцов и сохраняет их в векторе answer.

void writeAnswer(Vertex* suffAuto, std::vector<int>* arr,
std::vector<std::pair<int,int>>* answer, std::string& text)
Выводит ответ в консоль.

Класс Vertex:

Vertex(Vertex *par, char symb = 'A')

Конструктор для добавления новых вершин

Vertex (std::vector<std::pair<int,int>>* mainAnswer, std::vector<int>* mainArr)

Конструктор для создания стартовой вершины.

Vertex* getLink()

Вычисляет суффиксную ссылку или возвращает, если уже вычислена.

Vertex* go(char symb)

Осуществляет переход в автомате по символу symb.

Vertex* next(char symb)

Возвращает указатель на вершину, в которую можно перейти из текущей по символу symb.

Vertex* fastLink()

Вычисляет быструю суффиксную ссылку или возвращает, если уже вычислена.

void check(int ind)

Проходит по быстрым ссылкам до стартовой вершины и добавляет нужные индексы к ответу.

void setLeaf()

Меняет состояние флага терминальной вершины.

int countVert()

Вычисляет максимальное количество исходящих дуг.

Поиск одного шаблона с джокерами.

void addString(Vertex* suffAuto, std::string& s, int ind)

Добавляет строку s в автомат suffAuto.

void read(std::string& text, Vertex* suffAuto, std::vector<int>* arr)

Считывает данные с консоли.

void getAnswer(std::string& text, Vertex* suffAuto, std::vector<std::pair<int,int>>* answer)

Вычисляет индексы вхождений образцов и сохраняет их в векторе answer.

void writeAnswer(Vertex* suffAuto, std::vector<int>* arr,
std::vector<std::pair<int,int>>* answer, std::string& text)

Выводит ответ в консоль.

Класс Vertex:

Vertex(Vertex *par, char symb = 'A')

Конструктор для добавления новых вершин

Vertex (std::vector<std::pair<int,int>>* mainAnswer, std::vector<int>* mainArr)

Конструктор для создания стартовой вершины.

Vertex* getLink()

Вычисляет суффиксную ссылку или возвращает, если уже вычислена.

Vertex* go(char symb)

Осуществляет переход в автомате по символу symb.

Vertex* next(char symb)

Возвращает указатель на вершину, в которую можно перейти из текущей по символу symb.

Vertex* fastLink()

Вычисляет быструю суффиксную ссылку или возвращает, если уже вычислена.

void check(int ind)

Проходит по быстрым ссылкам до стартовой вершины и прибавляет единицу к нужным элементам массива C.

void setLeaf()

Меняет состояние флага терминальной вершины.

Тестирование

Тесты находятся в папке Tests.

Для тестирования был написан скрипт на python3, который запускает все тесты из папки вместе.

Множественный поиск.

№	Ввод	Вывод
1	NTAG 3 TAGT TAG T	maximum edge = 1 cut string: N 2 2 2 3
2	ACGACG 2 ACG TT	maximum edge = 2 cut string: 1 1 4 1
3	TNT 4 T N NT TN	maximum edge = 2 cut string: 1 1 1 4 2 2 2 3 3 1
4	AACGAA 3 AA	maximum edge = 3 cut string: C 1 1

	CC GA	4 3 5 1
5	AAAA 3 A AA AAA	maximum edge = 1 cut string: 1 1 1 2 1 3 2 1 2 2 2 3 3 1 3 2 4 1

Полный вывод программы для теста 2.

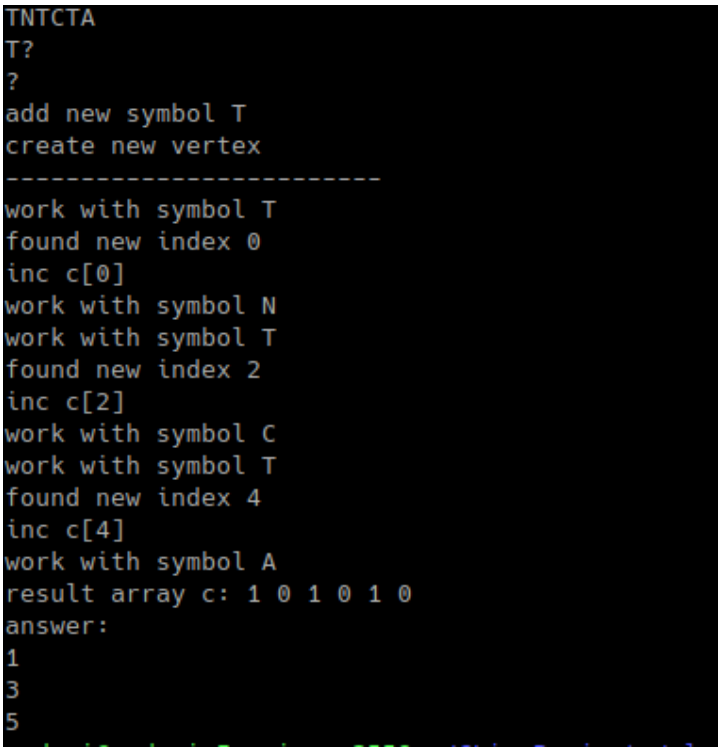
```
NTAG
3
TAGT
TAG
T
add string: TAGT
add string: TAG
add string: T
WORK:
current symbol: N
current symbol: T
found string with index 2
current symbol: A
current symbol: G
found string with index 1

SUFFIX AUTO:
this is root
path:
is termenated = false
suff link to root

path: T
is termenated = true
suff link to root

path: TA
is termenated = false
suff link to root

path: TAG
is termenated = true
suff link to root

path: TAGT
is termenated = true
```

```
ANSWER:
maximum edge = 1
cut string: N
2 2
2 3
```

Поиск одного шаблона с джокером

№	Ввод	Вывод
1	ACTANCA A$$A$ $	1
2	ACG A*G *	1
3	ACGTN * *	1 2 3 4 5
4	TNTCTA T? ?	1 3 5
5	ABCABC ABC *	1 4

Полный вывод программы для теста 4



Выводы

В ходе лабораторной работы был изучен принцип работы алгоритма Ахо-Корасик и было решено две задачи, с использованием этого алгоритма.

Приложение А

Задача 1

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>

class Vertex {
public:
    Vertex(Vertex *par, char symb = 'A'): symbol(symb), parent(par), start(par-
>start) {
        par->nextBor[symb] = this;
        answer = par->answer;
        arr = par->arr;
    }

       Vertex  (std::vector<std::pair<int,int>>*  mainAnswer,  std::vector<int>*
mainArr):
    answer(mainAnswer), arr(mainArr) {
        start = this;
        link = this;
        parent = this;
    }

    Vertex* getLink() {
        if (link == nullptr)
            if (this == start || parent == start)
                link = start;
            else
                link = parent->getLink()->go(symbol);
        return link;
    }

    Vertex* go(char symb) {
        if (nextAuto[symb] == nullptr) {
            if (nextBor[symb] != nullptr)
                nextAuto[symb] = nextBor[symb];
            else
                    nextAuto[symb] = this == start ? this : this->getLink()-
>go(symb);
        }
```

```
    return nextAuto[symb];
}

Vertex* next(char symb) {
    Vertex* newVertex =  nextBor[symb];
    if (newVertex == nullptr) {
        newVertex = new Vertex(this, symb);
    }
    return newVertex;
}

Vertex* fastLink() {
    if (f_link == nullptr) {
        Vertex* u = getLink();
        if (u == start)
            f_link = start;
        else
            f_link = (u->leaf) ? u : u->fastLink();
    }
    return f_link;
}

void check(int ind) {
    if (this == start) return;
    if (leaf) {
        answer->emplace_back(ind - (*arr)[leafInd] + 2, leafInd+1);
        std::cout << "found string with index " << leafInd << '\n';
    }
    fastLink()->check(ind);
}

void setLeaf() {
    leaf = true;
}

void setInd(int ind) {
    leafInd = ind;
}

int countVert() {
    int maxEdge = 0;
    for (auto i: nextBor) maxEdge += (i!=nullptr);
```

```
            for (auto i: nextBor) if (i) maxEdge = std::max(maxEdge, i->countVert());
        return maxEdge;
    }

    char getSymbol() {
        return symbol + 'A';
    }

    void printVertex(std::string path) {
        if (this == start) std::cout << "this is root\n";
        if (this != start) path += symbol + 'A';
        std::cout << "path: " << path << '\n';
        std::cout << "is termenated = " << (leaf ? "true\n" : "false\n");
        if (link == start) std::cout << "suff link to root\n";
         else if (link) std::cout << "suff link to " << link->getSymbol() << '\n';
        std::cout << '\n';

        for (auto i: nextBor) if (i) i->printVertex(path);
    }

private:
    char symbol;
    Vertex* parent = this;
    Vertex* link = nullptr;
    Vertex* f_link = nullptr;
    Vertex* start = nullptr;
    std::vector<Vertex*> nextBor = std::vector<Vertex*>(26, nullptr);
    std::vector<Vertex*> nextAuto = std::vector<Vertex*>(26, nullptr);
    bool leaf = false;
    int leafInd;
    std::vector<std::pair<int,int>>* answer;
    std::vector<int>* arr;
};

void addString(Vertex* suffAuto, std::string& s, int ind) {
    std::cout << "add string: " << s << '\n';
    Vertex* vertex = suffAuto;
    for (char c:s) {
        c -= 'A';
        if (vertex->next(c) == nullptr) {
```

```
            auto *newVertex = new Vertex(vertex, c);
        }
        vertex = vertex->next(c);
    }
    vertex->setLeaf();
    vertex->setInd(ind);
}

void read(std::string& text, Vertex* suffAuto, std::vector<int>* arr) {
    std::cin >> text;

    int n; std::cin >> n;
    for (int i = 0; i < n; i++) {
        std::string p; std::cin >> p;
        addString(suffAuto, p, i);
        arr->push_back(p.length());
    }
}

void        getAnswer(std::string&        text,        Vertex*        suffAuto,
std::vector<std::pair<int,int>>* answer) {
    int ind = 0;
    std::cout << "WORK:\n";
    Vertex* curVertex = suffAuto;
    for (char c: text) {
        std::cout << "current symbol: " << c << '\n';
        curVertex = curVertex->go(c-'A');
        curVertex->check(ind);
        ind++;
    }
    std::sort(answer->begin(), answer->end());
}

void      writeAnswer(Vertex*      suffAuto,      std::vector<int>*      arr,
std::vector<std::pair<int,int>>* answer, std::string& text) {
    std::cout << '\n';
    std::cout << "SUFFIX AUTO:\n";
    suffAuto->printVertex("");
    std::cout << "ANSWER:\n";
    std::cout << "maximum edge = " << suffAuto->countVert() << '\n';
    for (auto i: (*answer)) {
        for (int j = i.first-1; j < i.first-1 + (*arr)[i.second-1]; j++) {
```

```
            text[j] = '$';
        }
    }
    std::cout << "cut string: ";
    for (auto i: text)
        if (i != '$')
            std::cout << i;
    std::cout << "\n";
    for (auto j:(*answer))
        std::cout << j.first << ' ' << j.second << '\n';
}

int main() {
    auto* answer = new std::vector<std::pair<int,int>>;
    auto* arr = new std::vector<int>;
    auto* suffAuto = new Vertex(answer, arr);
    std::string text;

    read(text, suffAuto, arr);

    getAnswer(text, suffAuto, answer);

    writeAnswer(suffAuto, arr, answer, text);
}
```

Приложение В

Задача 2

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>

char joker;
int textSize;
int obrSize;

class Vertex {
public:
    Vertex(Vertex *par, char symb = 'A'): symbol(symb), parent(par), start(par-
>start) {
        par->nextBor[symb] = this;
        answer = par->answer;
        arr = par->arr;
        pos = par->pos;
        c_arr = par->c_arr;
    }

    Vertex (std::vector<std::pair<int,int>>* mainAnswer, std::vector<int>*
mainArr,
            std::vector<int>* mainc_arr, std::vector<int>* mainPos):
            answer(mainAnswer), arr(mainArr), pos(mainPos), c_arr(mainc_arr) {
        start = this;
        parent = this;
        link = this;
        f_link = this;
    }

    Vertex* getLink() {
        if (link == nullptr)
            if (this == start || parent == start)
                link = start;
            else
                link = parent->getLink()->go(symbol);
        return link;
    }
```

```
    Vertex* go(char symb) {
        if (nextAuto[symb] == nullptr) {
            if (nextBor[symb] != nullptr)
                nextAuto[symb] = nextBor[symb];
            else
                nextAuto[symb] = this == start ? this : this->getLink()-
>go(symb);
        }
        return nextAuto[symb];
    }

    Vertex* next(char symb, bool flag) {
        Vertex* newVertex =  nextBor[symb];
        if (newVertex == nullptr) {
            if (flag) std::cout << "create new vertex\n";
            newVertex = new Vertex(this, symb);
        }
        else
            if (flag) std::cout << "go to old vertex\n";
        return newVertex;
    }

    Vertex* fastLink() {
        if (f_link == nullptr) {
            Vertex* u = getLink();
            if (u == start)
                f_link = start;
            else
                f_link = (u->leaf) ? u : u->fastLink();
        }
        return f_link;
    }

    void check(int ind) {
        if (this == start) return;
        for (auto leafInd: leafInds) {
       int ind_in_c = ind - (*pos)[leafInd] - (*arr)[leafInd] + 1;
            std::cout << "found new index " << ind_in_c << '\n';
       if (ind_in_c >= 0 && ind_in_c + obrSize <= textSize) {
         (*c_arr)[ind_in_c]++;
                std::cout << "inc c[" << ind_in_c << "]\n";
            }
```

```
            else std::cout << "not in range\n";
     }
        fastLink()->check(ind);
    }

    void setLeaf() {
        leaf = true;
    }

    void setInd(int ind) {
        leafInds.push_back(ind);
    }

private:
    char symbol;
    Vertex* parent = this;
    Vertex* link = nullptr;
    Vertex* f_link = nullptr;
    Vertex* start = nullptr;
    std::vector<Vertex*> nextBor = std::vector<Vertex*>(26, nullptr);
    std::vector<Vertex*> nextAuto = std::vector<Vertex*>(26, nullptr);
    std::vector<int>* pos;
    std::vector<int>* c_arr;
    bool leaf = false;
    std::vector<std::pair<int,int>>* answer;
    std::vector<int>* arr;
    std::vector<int> leafInds;
};

void addString(Vertex* suffAuto, std::string& s, int ind) {
    Vertex* vertex = suffAuto;
    for (char c: s) {
        std::cout << "add new symbol " << c << "\n";
        c -= 'A';
        if (vertex->next(c,true) == nullptr) {
            auto *newVertex = new Vertex(vertex, c);
        }
        vertex = vertex->next(c,false);
    }
    vertex->setLeaf();
    vertex->setInd(ind);
}
```

```
int read(std::string& text, Vertex* suffAuto, std::vector<int>* arr,
std::vector<int>* pos) {
    std::cin >> text;
    textSize = text.length();

    int n = 1;
    int k = 0;
    for (int i = 0; i < n; i++) {
        std::string p; std::cin >> p; obrSize = p.length();
        std::cin >> joker;
        std::string partP;
        for (int j = 0; j < obrSize; j++) {
            if (p[j] == joker) {
                if (!partP.empty()) {
                 (*pos).push_back(j - partP.length());
                    addString(suffAuto, partP, k);
                    arr->push_back(partP.length());
                    partP.clear();
                    k++;
                }
            }
            else {
                partP.push_back(p[j]);
            }
        }
        if (!partP.empty()) {
          (*pos).push_back(obrSize - partP.length());
            addString(suffAuto, partP, k);
            arr->push_back(partP.length());
            k++;
        }
    }
    return k;
}

void getAnswer(std::string& text, Vertex* suffAuto,
std::vector<std::pair<int,int>>* answer) {
    int ind = 0;
    Vertex* curVertex = suffAuto;
    for (char c: text) {
        std::cout << "work with symbol " << c << '\n';
```

```
        curVertex = curVertex->go(c-'A');
        curVertex->check(ind);
        ind++;
    }
    std::sort(answer->begin(), answer->end());
}

void writeAnswer(std::vector<std::pair<int,int>>* answer) {
    for (auto j:(*answer))
        std::cout << j.first << ' ' << j.second << '\n';
}

int main() {
    auto* answer = new std::vector<std::pair<int,int>>;
    auto* arr = new std::vector<int>;
    std::string text;

    std::vector<int>* c_arr = new std::vector<int>;
    std::vector<int>* pos = new std::vector<int>;

    auto* suffAuto = new Vertex(answer, arr, c_arr, pos);

    int k = read(text, suffAuto, arr, pos);
    (*c_arr) = std::vector<int>(textSize, 0);
    std::cout << "-------------------------\n";

    getAnswer(text, suffAuto, answer);

    std::cout << "result array c: ";
    for (auto i: (*c_arr)) std::cout << i << ' ';
    std::cout << '\n';

    std::cout << "answer:\n";
    for (int i = 0; i < textSize; i++) {
        if ((*c_arr)[i] == k) {
            std::cout << i+1 << '\n';
        }
    }
}
```