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 _____ Ларин А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить принцип работы алгоритма Ахо-Корасик для нахождения подстрок в строке. Решить с его помощью задачи

Основные теоретические положения.

Алгоритм Ахо — Корасик — алгоритм поиска подстроки, разработанный Альфредом Ахо и Маргарет Корасик, реализует поиск множества подстрок из словаря в данной строке.

Алгоритм строит конечный автомат, которому затем передаёт строку поиска. Автомат получает по очереди все символы строки и переходит по соответствующим рёбрам. Если автомат пришёл в конечное состояние, соответствующая строка словаря присутствует в строке поиска.

Несколько строк поиска можно объединить в дерево поиска, так называемый бор (префиксное дерево). Бор является конечным автоматом, распознающим одну строку из m — но при условии, что начало строки известно.

Первая задача в алгоритме — научить автомат «самовосстанавливаться», если подстрока не совпала. При этом перевод автомата в начальное состояние при любой неподходящей букве не подходит, так как это может привести к пропуску подстроки (например, при поиске строки aabab, попадается aabaabab, после считывания пятого символа перевод автомата в исходное состояние приведёт к пропуску подстроки — верно было бы перейти в состояние a, а потом снова обработать пятый символ). Чтобы автомат самовосстанавливался, к нему добавляются суффиксные ссылки (так что детерминированный автомат превращается в недетерминированный). Например, если разобрана строка aaba, то бору предлагаются суффиксы aba, ba, a. Суффиксная ссылка — это ссылка на узел, соответствующий самому длинному суффиксу, который не заводит бор в тупик.

Для корневого узла суффиксная ссылка — петля. Для остальных правило таково: если последний распознанный символ — c, то осуществляется обход по суффиксной ссылке родителя, если оттуда есть дуга, нагруженная символом c

суффиксная ссылка направляется в тот узел, куда эта дуга ведёт. Иначе — алгоритм проходит по суффиксной ссылке ещё и ещё раз, пока либо не пройдёт по корневой ссылке-петле, либо не найдёт дугу, нагруженную символом c.

Этот автомат недетерминированный. Преобразование недетерминированного конечного автомата в детерминированный в общем случае приводит к значительному увеличению количества вершин. Но этот автомат можно превратить в детерминированный, не создавая новых вершин: если для вершины v некуда идти по символу c, проходимся по суффиксной ссылке ещё и ещё раз — пока либо не попадём в корень, либо будет куда идти по символу c.

Детерминизация увеличивает количество конечных вершин: если суффиксные ссылки из вершины v ведут в конечную u, сама v тоже объявляется конечной. Для этого создаются так называемые конечные ссылки

Конечные ссылки - это суффиксные ссылки, ведущие на ближайшие конечные вершины, т. е. вершины, путь от корня до которых соответсвует переходу по символам паттерна. Обход по конечным ссылкам даёт все совпавшие строки.

Задание

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:
Первая строка содержит текст (T,$1 \le |T| \le 100000$).
Вторая - число n ($1 \le n \le 3000$), каждая следующая из n строк содержит шаблон из набора $P=\{p1,\ldots,pn\}1 \le |pi| \le 75$
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:

```
CCCA
```

```
1
CC
```

Sample Output:

```
1 1
2 1
```

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
```

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

Реализация

Описание алгоритма

Создается бор и инициализируется одной пустой вершиной.

Далее в него заносятся паттерны, которые в дальнейшем будем искать. Для этого в цикле пробегается каждый паттерн, и по каждому символу либо происходит переход по дуге, если она уже есть, либо создается новая. Вершина, соответсвующая последнему символу паттерна помечается как конечная.

Для просчета очередной суффиксной ссылки происходит переход по суффиксной ссылке родителя и далее по одному из символов из исходной вершины. Сооветствующая вершина инциндентная исходной будет ссылаться на вершину, являющуюся потомком вершины, на которую ведет суффиксная ссылка родителя. Суффиксные ссылки высчитываются по необходимости т. е. Совершается проход по строке для поиска, и при переходе по очередной вершине, в случае если для нее отсутствует соответствующий переход к потомку происходит переход по высчитанной тут же суффиксной ссылке.

Обработка текста представляет собой последовательные переходы по состояниям автомата по символам текста. При этом переход может происходить по ребру, соответствующему пришедшему символу, если таковое присутствует, либо по суффиксной ссылке, если нужного ребра нет. Переход возможен всегда, так если на вход будут приходить символы, не соответствующие паттернам, то суффиксные ссылки приведут в корень бора, соответствующему постой строке.

После перехода происходит следование по конечным ссылкам и финальные вершины запоминаются как вхождение паттерна в строку поиска.

Переходы(пары — тек.вершина, пришедший символ) высчитываются лишь однажды и кешируются.

Задача с джокером решается похожим образом, но для каждого символа, при невозможности перейти по дуге соответсвующей входному символу делается попытка перейти по символу джокера. Переход по символу джокера точно так же кешируется.

Сложность по памяти —O(m*k), где m — сумма длина шаблонов, k — длин алфавита.

Сложность по времени — линейная от суммы длинн шаблона и текста O(m*k+n+l), где n - длина строки поиска, l – общая длина всех совпадений

Описание функций и структур данных

Вершины хранятся в виде структуры Vertex

```
struct Vertex {
    int children[k];       // Массив потомков
    int ix;                // Индекс в массиве
    int suffixIx;          //Суффикс-ссылка
    int advanceVertexes[k];//Кешированные переходы
    int par;               //Ссылка на родителя
    int goodSuffixIx;      //Конечная ссылка
    bool flag;             //Является ли вершина конечной
    char symbol;           //Символ соответсвующей сходной дуги
};
```

Бор представлени в виде вектора вершин

```
std::vector<Vertex> bohr;
```

Паттерны хранятся в виде вектора строк
vector<string> patterns;

`Vertex createVertex(int p, char c)` – Создание вершины. P – ссылка на родителя, c – символ входной дуги

void bohrInit() - инициальзация бора одной вершиной

void addString(const string &s) – добавление строки s к бору

`int getAdvanceVertex(int v, char ch)` - получение следующей вершины из v по символу ch

int getSuffixIx(int v) – получение суффикс ссылки для вершины v

int getGoodSuffixIx(int v) – получение конечной ссылки для вершины v

void follow(int v, int i, vector<pair<int, int>> &occ) - следование по конечным ссылкам начиная с вершины v, соответствующей позиции в тексте i. Встреченые вхождения записываются в вектор вхождений occ

vector<pair<int, int>> findAllOccurrences(const string &s) — проход по строке в поисках совпадений. Возвращает позиции вхождения паттерна в строку s в виде вектора пар <Позиция, индекс паттерна>.

Тесты.

Задание 1

1.

```
abac
2
aba
ac

/-----------------\
Adding sample: aba
current char: a
Adding new edge: 0{a} -> 0{a}
Current vertex: 1{a}
current char: b
Adding new edge: 1{a} -> 1{b}
Current vertex: 2{b}
current char: a
Adding new edge: 2{b} -> 2{a}
Current vertex: 3{a}
Vertex 3{a} marked final
Sample added aba
\-----------------/
c

/-----------------\
Adding sample: ac
current char: a
Current vertex: 1{a}
current char: c
Adding new edge: 1{a} -> 3{c}
Current vertex: 4{c}
Vertex 4{c} marked final
Sample added ac
\-----------------/
Input string: abac
STATE 0 INPUT a AT POSITION 0

/>=>=>=>=>=>=>=>=>\
Getting advance from vertex 0{a} by char a
Advance from vertex 0{a} by char a is 1{a}
\>=>=>=>=>=>=>=>=>/
NEW STATE 1

Follow vertex 1{a} for matches

/->->->->->->->->->->\
```

```
Getting follow suffix link for vertex 1{a}
Follow suffix link for vertex 1{a} is 0{a}
\->->->->->->->->->->/
STATE 1 INPUT b AT POSITION 1

/>=>=>=>=>=>=>=>=>=>\
Getting advance from vertex 1{a} by char b
Advance from vertex 1{a} by char b is 2{b}
\>=>=>=>=>=>=>=>=>=>/
NEW STATE 2

Follow vertex 2{b} for matches

/->->->->->->->->->->\
Getting follow suffix link for vertex 2{b}

/>=>=>=>=>=>=>=>=>=>\
Getting advance from vertex 0{a} by char b
Advance from vertex 0{a} by char b is 0{a}
\>=>=>=>=>=>=>=>=>=>/
Follow suffix link for vertex 2{b} is 0{a}
\->->->->->->->->->->/
STATE 2 INPUT a AT POSITION 2

/>=>=>=>=>=>=>=>=>=>\
Getting advance from vertex 2{b} by char a
Advance from vertex 2{b} by char a is 3{a}
\>=>=>=>=>=>=>=>=>=>/
NEW STATE 3

Follow vertex 3{a} for matches
<!> Found match at position 1 of pattern aba

/->->->->->->->->->->\
Getting follow suffix link for vertex 3{a}

/>=>=>=>=>=>=>=>=>=>\
Getting advance from vertex 0{a} by char a
Advance from vertex 0{a} by char a is 1{a}
\>=>=>=>=>=>=>=>=>=>/

/->->->->->->->->->->\
Getting follow suffix link for vertex 1{a}
Follow suffix link for vertex 1{a} is 0{a}
\->->->->->->->->->->/
Follow suffix link for vertex 3{a} is 0{a}
\->->->->->->->->->->/
```

```
STATE 3 INPUT c AT POSITION 3

/>=>=>=>=>=>=>=>=>\
Getting advance from vertex 3{a} by char c
jump to suffix 1{a} by char c

/>=>=>=>=>=>=>=>=>\
Getting advance from vertex 1{a} by char c
Advance from vertex 1{a} by char c is 4{c}
\>=>=>=>=>=>=>=>=>/
Advance from vertex 3{a} by char c is 4{c}
\>=>=>=>=>=>=>=>=>/
NEW STATE 4

Follow vertex 4{c} for matches
<!> Found match at position 3 of pattern ac

/->->->->->->->->->\
Getting follow suffix link for vertex 4{c}

/>=>=>=>=>=>=>=>=>\
Getting advance from vertex 0{a} by char c
Advance from vertex 0{a} by char c is 0{a}
\>=>=>=>=>=>=>=>=>/
Follow suffix link for vertex 4{c} is 0{a}
\->->->->->->->->->/
1 1
3 2

Var 5 individualisation
Max out-edges from a vertex: 2
String with patterns removed:
```

2.

```
CCCA
1
CC

1 1
2 1
Max out-edges from a vertex: 1
String with patterns removed:
A
```

3.

```
qabcbadabq
```

```
4
abc
abq
bc
ba

2 1
3 3
5 4
8 2
Max out-edges from a vertex: 2
String with patterns removed:
qd
```

4.

```
abcdqqdcba
4
a
b
c
d
1 1
2 2
3 3
4 4
7 4
8 3
9 2
10 1
Max out-edges from a vertex: 4
String with patterns removed:
qq
```

5.

```
abcdqqdcba
1
abcdqqdcba
1 1
Max out-edges from a vertex: 1
String with patterns removed:
```

Задание 2

6.

```
ACTANCA
A$$A$
$
```

```
1
Max out-edges from a vertex: 1
String with patterns removed:
CA
```

7.

```
ACTANCA
A$$A
$
1
4
Max out-edges from a vertex: 1
String with patterns removed:
```

8.

```
ACACGGG
ACXXG
X
1
3
Max out-edges from a vertex: 1
String with patterns removed:
```

Выводы.

В результате работы была написана полностью рабочая программа решающая поставленную задачу при использовании изученных теоретических материалов. Программа было протестирована, результаты тестов удовлетворительны.

ПРИЛОЖЕНИЕ А(ЛИСТИНГ ПРОГРАММЫ)

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

using namespace std;

#define JOKER
#define TASK

//#define DEBUG

#ifdef DEBUG
#define BDEBUG
#define CDEBUG
#endif

#define AL_TO_NUM(AL) ( (AL)==J?(K-1):( (AL)>='a'&&(AL)<='z'?(AL)-'a':
((AL)>='A'&&(AL)<='Z'?(AL)-'A':(AL)) ) )  //Symbol to index
#define NUM_TO_AL(NUM) ((char)( (NUM)==(K-1)?J:(((NUM)>=0 && (NUM)< K)?((NUM)
+'a'):(NUM)) ))                      //Index To symbol

const int K = 27;//Alphabet length(with joker)
struct Vertex {
    int children[K], ix, suffixIx, advanceVertexes[K], par, goodSuffixIx;
    bool flag;
    char symbol;
};
vector<Vertex> bohr;
vector<string> patterns;
char J = -1;//Joker symbol

Vertex createVertex(int p, char c) {//Create and init fields
    Vertex v;
    for (int i = 0; i < K; i++) {
        v.children[i] = v.advanceVertexes[i] = -1;
    }
    v.flag = false;
    v.suffixIx = -1;
    v.par = p;
    v.symbol = c;
    v.goodSuffixIx = -1;
    return v;
```

```
}

VOID BOHRINIT() {//INIT WITH ONE VERTEX
    BOHR.PUSH_BACK(CREATEVERTEX(0, '\0'));
}

VOID ADDSTRING(CONST STRING &S) {
    INT CURLAST = 0;
#IFDEF BDEBUG
    COUT<<"\N/-----------------\\"<<ENDL;
    COUT<<"ADDING SAMPLE: "<<S<<ENDL;
#ENDIF
    FOR (INT I = 0; I < S.LENGTH(); I++) {//RUN THROUGH CHARS
#IFDEF BDEBUG
        COUT<<"CURRENT CHAR: "<<S[I]<<ENDL;
#ENDIF
        CHAR CH = AL_TO_NUM(S[I]);



        IF (BOHR[CURLAST].CHILDREN[CH] == -1) {//CHECK IF ALREADY EXIST
#IFDEF BDEBUG
            COUT<<"ADDING NEW EDGE:
"<<CURLAST<<"{"<<NUM_TO_AL(BOHR[CURLAST].SYMBOL)<<"} -> "<<BOHR.SIZE() -
1<<"{"<<NUM_TO_AL(CH)<<"}"<<ENDL;
#ENDIF
            BOHR.PUSH_BACK(CREATEVERTEX(CURLAST, CH));//MAKE NEW VERTEX IF NOT
            BOHR[CURLAST].CHILDREN[CH] = BOHR.SIZE() - 1;
        }
        CURLAST = BOHR[CURLAST].CHILDREN[CH];
#IFDEF BDEBUG
        COUT<<"CURRENT VERTEX:
"<<CURLAST<<"{"<<NUM_TO_AL(BOHR[CURLAST].SYMBOL)<<"}"<<ENDL;
#ENDIF
    }
    BOHR[CURLAST].FLAG = TRUE;
#IFDEF BDEBUG
    COUT<<"VERTEX "<<CURLAST<<"{"<<NUM_TO_AL(BOHR[CURLAST].SYMBOL)<<"}"<<" MARKED
FINAL"<<ENDL;
#ENDIF
    PATTERNS.PUSH_BACK(S);
    BOHR[CURLAST].IX = PATTERNS.SIZE() - 1;

#IFDEF BDEBUG
    COUT<<"SAMPLE ADDED "<<S<<ENDL;
```

```
    COUT<<"\\-----------------/"<<ENDL;
#ENDIF

}



INT GETADVANCEVERTEX(INT V, CHAR CH);

INT GETSUFFIXIX(INT V) {
#IFDEF DDEBUG
    COUT<<"GETTING SUFFIX LINK FOR"<<V<<"{"<<NUM_TO_AL(BOHR[V].SYMBOL)<<"}"<<ENDL;
#ENDIF
    IF (BOHR[V].SUFFIXIX == -1)//YET HAVE NONE
        IF (V == 0 || BOHR[V].PAR == 0)//TRIVIAL
            BOHR[V].SUFFIXIX = 0;
        ELSE
#IFDEF DDEBUG
            COUT<<"ADVANCING FROM PARENT"<<ENDL;
#ENDIF
            BOHR[V].SUFFIXIX = GETADVANCEVERTEX(GETSUFFIXIX(BOHR[V].PAR),
BOHR[V].SYMBOL);//CALC FROM PARENT'S SUFFIX
#IFDEF DDEBUG
    COUT<<"SUFFIX LINK FOR"<<V<<"{"<<NUM_TO_AL(BOHR[V].SYMBOL)<<"}"<<"
IS"<<BOHR[V].SUFFIXIX<<"{"<<NUM_TO_AL(BOHR[BOHR[V].SUFFIXIX].SYMBOL)<<"}"<<ENDL;
#ENDIF
    RETURN BOHR[V].SUFFIXIX;
}

INT GETADVANCEVERTEX(INT V, CHAR CH) {
    //ADVANCE BY CHAR, CHECKING J IN BOHR
#IFDEF CDEBUG
    COUT<<"\N/>=>=>=>=>=>=>=>=>\\"<<ENDL;
    COUT<<"GETTING ADVANCE FROM VERTEX "<<V<<"{"<<NUM_TO_AL(BOHR[V].SYMBOL)<<"} "<<"BY
CHAR "<<NUM_TO_AL(CH)<<ENDL;
#ENDIF
    IF (BOHR[V].ADVANCEVERTEXES[CH] == -1)//DONT HAVE REGULAR ADVANCE VERTEX
        IF (BOHR[V].ADVANCEVERTEXES[AL_TO_NUM(J)] == -1) {//DONT HAVE J ADVANCE VERTEX
            IF (BOHR[V].CHILDREN[CH] != -1) {//HAVE REGULAR CHILD
                BOHR[V].ADVANCEVERTEXES[CH] = BOHR[V].CHILDREN[CH];
            } ELSE IF (BOHR[V].CHILDREN[AL_TO_NUM(J)] != -1) {//HAVE J CHILD
                BOHR[V].ADVANCEVERTEXES[CH] = BOHR[V].CHILDREN[AL_TO_NUM(J)];
            } ELSE IF (V == 0)//IN INIT VERTEX
                BOHR[V].ADVANCEVERTEXES[CH] = 0;
```

```
            ELSE {
                AUTO SF = GETSUFFIXIX(V);
#IFDEF CDEBUG
                COUT<<"JUMP TO SUFFIX "<<SF<<"{"<<NUM_TO_AL(BOHR[SF].SYMBOL)<<"}
"<<"BY CHAR "<<NUM_TO_AL(CH)<<ENDL;
#ENDIF
                BOHR[V].ADVANCEVERTEXES[CH] = GETADVANCEVERTEX(SF, CH);//JUMP TO SUFFIX'S
ADVANCE VERTEX
            }
        } ELSE {//HAVE J ADVANCE VERTEX
            BOHR[V].ADVANCEVERTEXES[CH] = BOHR[V].ADVANCEVERTEXES[AL_TO_NUM(J)];
        }
#IFDEF CDEBUG
    COUT<<"ADVANCE FROM VERTEX "<<V<<"{"<<NUM_TO_AL(BOHR[V].SYMBOL)<<"} "<<"BY CHAR
"<<NUM_TO_AL(CH)<<" IS
"<<BOHR[V].ADVANCEVERTEXES[CH]<<"{"<<NUM_TO_AL(BOHR[BOHR[V].ADVANCEVERTEXES[CH]].SYMBOL)<<
"} "<<ENDL;
    COUT<<"\\>=>=>=>=>=>=>=>=>/"<<ENDL;
#ENDIF

    RETURN BOHR[V].ADVANCEVERTEXES[CH];

}

INT GETGOODSUFFIXIX(INT V) {
#IFDEF CDEBUG
    COUT<<"\N/->->->->->->->->->->\\"<<ENDL;
    COUT<<"GETTING FOLLOW SUFFIX LINK FOR VERTEX
"<<V<<"{"<<NUM_TO_AL(BOHR[V].SYMBOL)<<"}"<<ENDL;
#ENDIF
    IF (BOHR[V].GOODSUFFIXIX == -1) {//SELF ONES SUFFIXES LEADING TO FINALS TO REDUCE
ASYMPTOTIC
        INT U = GETSUFFIXIX(V);
        IF (U == 0)
            BOHR[V].GOODSUFFIXIX = 0;
        ELSE
            BOHR[V].GOODSUFFIXIX = (BOHR[U].FLAG) ? U : GETGOODSUFFIXIX(U);
    }
#IFDEF CDEBUG
    COUT<<"FOLLOW SUFFIX LINK FOR VERTEX "<<V<<"{"<<NUM_TO_AL(BOHR[V].SYMBOL)<<"} IS
"<<BOHR[V].GOODSUFFIXIX<<"{"<<NUM_TO_AL(BOHR[BOHR[V].GOODSUFFIXIX].SYMBOL)<<"}"<<ENDL;
    COUT<<"\\->->->->->->->->->->/"<<ENDL;
#ENDIF

    RETURN BOHR[V].GOODSUFFIXIX;
```

```
}

VOID FOLLOW(INT V, INT I, VECTOR<PAIR<INT, INT>> &OCC) {
#IFDEF CDEBUG
    COUT<<"\NFOLLOW VERTEX "<<V<<"{"<<NUM_TO_AL(BOHR[V].SYMBOL)<<"} FOR MATCHES"<<ENDL;
#ENDIF
    FOR (INT U = V; U != 0; U = GETGOODSUFFIXIX(U)) {
        IF (BOHR[U].FLAG) {//FINAL => FOUND
            OCC.PUSH_BACK({I - PATTERNS[BOHR[U].IX].LENGTH() + 1, BOHR[U].IX + 1});
#IFDEF CDEBUG
            COUT<<"<!> FOUND MATCH AT POSITION "<<I-PATTERNS[BOHR[U].IX].LENGTH()+1<<" OF
PATTERN "<<PATTERNS[BOHR[U].IX]<<ENDL;
#ENDIF
        }
    }
}

VECTOR<PAIR<INT, INT>> FINDALLOCCURRENCES(CONST STRING &S) {
    VECTOR<PAIR<INT, INT>> OCC;
    INT U = 0;
    FOR (INT I = 0; I < S.LENGTH(); I++) {
#IFDEF CDEBUG
        COUT<<"STATE "<<U<<" INPUT "<<S[I]<<ENDL;
#ENDIF
        U = GETADVANCEVERTEX(U, AL_TO_NUM(S[I]));
#IFDEF CDEBUG
        COUT<<"NEW STATE "<<U<<ENDL;
#ENDIF
        FOLLOW(U, I + 1, OCC);
    }
    RETURN OCC;
}

VOID OUTVECPAIRS(CONST VECTOR<PAIR<INT, INT>> &OCC) {
    FOR (CONST PAIR<INT, INT> &IT:OCC) {
#IFDEF JOKER
        COUT << IT.FIRST << ENDL;
#ELSE
        COUT << IT.FIRST << " " << IT.SECOND << ENDL;
#ENDIF
    }
}
```

```
VOID TASK(STRING S, VECTOR<PAIR<INT, INT>> VEC) {

#IFDEF DEBUG
    COUT<<"\NVAR 5 INDIVIDUALISATION"<<ENDL;
#ENDIF

    INT MAX = 0;
    FOR (VERTEX I:BOHR) {
        INT S = 0;
        FOR (INT C:I.CHILDREN)IF (C != -1)S++;
        IF (S > MAX)MAX = S;
    }
    COUT << "MAX OUT-EDGES FROM A VERTEX: " << MAX << ENDL;

    COUT << "STRING WITH PATTERNS REMOVED: " << ENDL;

    FOR (INT I = 0; I < S.LENGTH(); I++) {
        BOOL B = TRUE;
        FOR (PAIR<INT, INT> P:VEC) {
            IF (I >= P.FIRST - 1)
                IF (I <= P.FIRST + PATTERNS[P.SECOND - 1].LENGTH() - 2)
                    B = FALSE;
        }
        IF (B)
            COUT << S[I];
    }
}

INT MAIN() {
    BOHRINIT();
    STRING S = "";
    STRING TMP = "";

#IFDEF JOKER
    CIN>>S;
    CIN>>TMP;
    CIN>>J;
    ADDSTRING(TMP);
#ELSE
    INT N;
    CIN >> S;
    CIN >> N;
```

```
    WHILE (N--) {
        CIN >> TMP;
        ADDSTRING(TMP);
    }
#ENDIF

#IFDEF DEBUG
    COUT<<"INPUT STRING: "<<S<<ENDL;
#IFDEF JOKER
    COUT<<"JOKER SYMBOL: "<<J<<ENDL;
#ENDIF
#ENDIF


    AUTO VEC = FINDALLOCCURRENCES(S);
    OUTVECPAIRS(VEC);

#IFDEF TASK
    TASK(S, VEC);
#ENDIF
}
```