**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №6**

**по дисциплине «Проектирование и анализ алгоритмов»**

Тема: Алгоритм Ахо-Корасик

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Корытов П.В. |
| Преподаватель |  | Балтрашевич В.Э. |

Санкт-Петербург

2018

**Цель работы**

Изучить реализацию алгоритма Ахо-Корасик

**Основные теоретические положения**

Есть множество шаблонов и один текст .

Суммарная длина всех образцов -

Нужно найти, какие образцы из данного множества встречаются в этом тексте. Применение алгоритма Кнута-Морриса-Пратта неэффективно, т.к. алгоритм нужно будет запустить для каждого текста.

**Построение бора**

Бор - дерево с корнем в некоторой вершине, каждое ребро которого подписано некоторой буквой.

Пусть дан набор строк -

У дерева каждая дуга имеет пометку в виде одного символа, и не может быть двух дуг из одной вершины, которые имеют одинаковую пометку.

*Элемент бора на псевдокоде:*

**struct** Node:  
 **Node** son[k] // массив сыновей  
 **Node** go[k] // массив переходов (запоминаем переходы в ленивой рекурсии), используемый для вычисления суффиксных ссылок  
 **Node** parent // вершина родитель  
 **Node** suffLink // суффиксная ссылка (вычисляем в ленивой рекурсии)  
 **Node** up // сжатая суффиксная ссылка  
 **char** charToParent // символ, ведущий к родителю  
 **bool** isLeaf // флаг, является ли вершина терминалом  
 **vector<int>** leafPatternNumber // номера строк, за которые отвечает терминал

*Добавление слова в бор на псевдокоде:*

**fun** addString(**string** s, **int** patternNumber):  
 **Node** cur = root  
 **for** i = 0 **to** s.length - 1  
 **char** c = s[i]  
 **if** cur.son[c] == 0  
 cur.son[c] = Node  
 /\* здесь также нужно обнулить указатели на переходы и сыновей \*/  
 cur.son[c].suffLink = 0  
 cur.son[c].up = 0  
 cur.son[c].parent = cur  
 cur.son[c].charToParent = c  
 cur.son[c].isLeaf = *false*  
 cur = cur.son[c]  
 cur.isLeaf = *true*  
 cur.leafPatternNumber.pushBack(patternNumber)

Пусть слово, приводящие в вершину в боре. Узлы бора можно понимать как состояния автомата, корень - как начальное состояние. Соответственно, узлы бора, в которых заканчиваются строки, становятся терминальными.

**Переходы по бору**

Заведем несколько функций для перехода по бору:

Во-первых, нужно определить функцию перехода "наверх":

возвращает родителя вершины

**Суффиксные ссылки**

суффиксная ссылка, и существует переход из в по символу

Суффиксная ссылка для вершины вершина, в которой оканчивается наидлиннейший собственный суффикс строки, соответствующей вершине Суффикс может быть и нулевой длины - т.е. суффиксная ссылка может вести в корень.

Другими словами, если максимальный суффикс Суффиксные ссылки для вышеуказанного примера:

Если ссылка не обозначена, значит, она ведёт в корень.

*Вычисление суффиксной ссылки на псевдокоде*

**Node** getSuffLink(**Node** v):  
 **if** v.suffLink == *null* // если суффиксная ссылка ещё не вычислена  
 **if** v == root **or** v.parent == root  
 v.suffLink = root  
 **else**  
 v.suffLink = getLink(getSuffLink(v.parent), v.charToParent)  
 **return** v.suffLink

В данном случае и далее используется ленивая рекурсия - ссылки не вычисляются, если не нужны, но вычисленные ссылки запоминаются. Это позволяет экономить ресурсы.

**Переход по бору**

Переход осуществляется по текущей вершине и символу

*Функция перехода на псевдокоде*

**Node** getLink(**Node** v, **char** c):   
 **if** v.go[c] == *null* // если переход по символу c ещё не вычислен  
 **if** v.son[c]  
 v.go[c] = v.son[c]  
 **else** **if** v == root   
 v.go[c] = root   
 **else**   
 v.go[c] = getLink(getSuffLink(v), c)  
 **return** v.go[c]

**Сжатые суффиксные ссылки**

При построении автомата может возникнуть ситуация, что ветвление есть не на каждом символе. Тогда можно использовать **сжатые суффиксные ссылки:**

ближайшее допускающее состояние (терминал) перехода по суффиксным ссылкам.

*Вычисление сжатых суффиксных ссылок на псевдокоде:*

**Node** getUp(**Node** v):  
 **if** v.up == *null* // если сжатая суффиксная ссылка ещё не вычислена  
 **if** getSuffLink(v).isLeaf  
 v.up = getSuffLink(v)  
 **else** **if** getSuffLink(v) == root  
 v.up = root  
 **else**   
 v.up = getUp(getSuffLink(v))  
 **return** v.up

В результате вышеописанных действий построен конечный детерминированный автомат.

**Использование автомата**

В общих чертах, получившийся автомат нужно использовать следующим образом: по очереди просматривать символы текста, для каждого символа осуществляя переход по где текущее состояние. Оказавшись в новом состоянии, отметить по сжатым суффиксным ссылкам строки, которые встретились, и, если требуется, позицию.

*Использование автомата на псевдокоде:*

**fun** processText(**string** t):   
 **Node** cur = root  
 **for** i = 0 **to** t.length - 1   
 **char** c = t[i] - 'a'  
 cur = getLink(cur, c)  
 /\* В этом месте кода должен выполняться переход по **сжатой** суффиксной ссылке getUp(cur). Для вершины,  
 обнаруженной по ней тоже ставим, что она найдена, затем повторяем для её сжатой суффиксной ссылки  
 и так до корня. \*/

**Ход работы**

1. На основе теоретических положений описана структура для хранения бора:

typedef struct Node{

vector<Node\*> son; //Children

vector<Node\*> go; //Transitions

Node\* parent;

char charToParent;

Node\* suffLink;

Node\* up; //Compressed suffix link

bool isLeaf;

vector<int> leafPatternNumber;

Node();

}Node;

1. Для ускорения работы ссылки хранятся в векторе из 27 элементов (26 символов английского алфавита и один джокер). Для перевода char в символ от 0 до 26 написана функция:

char SYM(const char ch, const char jok){

static char joker = jok;

if ((joker!=jok) && (jok!=0) && (jok!=32))

joker = jok;

if (((ch == joker) || (jok == 32)) && (joker!=0))

return 26;

return tolower(ch) - 'a';

}

1. Описана процедура добавления строки в бор:

void addString(const string & s, int patternNumber, Node\* root, const char jok){

Node\* cur = root;

for (size\_t i = 0; i < s.length(); i++){

char ch = s[i];

char c = SYM(ch, jok);

if (cur->son[c] == nullptr){

cur->son[c] = new Node;

cur->son[c]->parent = cur;

cur->son[c]->charToParent = ch;

}

cur->isLeaf = false;

cur = cur->son[c];

}

cur->isLeaf = true;

cur->leafPatternNumber.push\_back(patternNumber);

}

1. Описаны функции для переходов по автомату. В процедурах используется метод ленивой рекурсии

Node\* getSuffLink(Node\* v, Node\* root){

if (v->suffLink == nullptr){

if ((v == root) || (v->parent == root))

v->suffLink = root;

else

v->suffLink = getLink(getSuffLink(v->parent, root), v->charToParent, root);

}

return v->suffLink;

}

Node\* getLink(Node\* v, char c, Node\* root){

char nc = SYM(c);

char jok = SYM(0, 32);

if (v->go[nc] == nullptr){

if (v->son[nc])

v->go[nc] = v->son[nc];

else if (v->son[jok]){

v->go[jok] = v->son[jok];

return v->go[jok];

}

else if (v == root)

v->go[nc] = root;

else

v->go[nc] = getLink(getSuffLink(v, root), c, root);

}

return v->go[nc];

}

1. Описана функция перехода по сжатым суффиксным ссылкам:

Node\* getUp(Node\* v, Node\* root){

if (v->up == nullptr){

if (getSuffLink(v, root)->leafPatternNumber.size())

v->up = getSuffLink(v, root);

else if (getSuffLink(v, root) == root)

v->up = root;

else

v->up = getUp(getSuffLink(v, root), root);

}

return v->up;

}

1. Описан тип для хранения результата работы алгоритма:

typedef struct ACresEl{

size\_t index;

int patternNumber;

ACresEl(size\_t ind, int pN);

}ACresEl;

typedef vector<ACresEl> ACres;

И сама процедура, выполняющая алгоритм Ахо-Корасик:

void processText(const string & T, Node\* root, ACres & res){

Node\* cur = root;

for (size\_t i = 0; i < T.length(); i++){

char c = T[i];

cur = getLink(cur, c, root);

for (Node\* u = cur; u != root; u = getUp(u, root)){

if (u->leafPatternNumber.size()!=0)

for (auto it : u->leafPatternNumber){

res.push\_back(ACresEl(i, it));

}

}

}

}

1. Реализованы процедуры, конвертирующие результат в вид, требуемый заданием:

void convertStepik(ACres & res, vector<size\_t> & len){

ACres::iterator it;

for (it = res.begin(); it < res.end(); it++){

it->index = it->index + 2 - len[it->patternNumber];

it->patternNumber++;

}

}

ACres doAhoCorasick(const string & T, const vector<string> & P, const char joker){

ACres res;

Node\* bohr = new Node;

size\_t size = P.size();

vector <size\_t> len(size);

int i = 0;

for (auto it : P){

addString(it, i, bohr, joker);

len[i++] = it.length();

}

processText(T, bohr, res);

convertStepik(res, len);

return res;

}

1. Для проведения тестирования написана функция сравнения результатов:

bool CMP (const ACres & a, const ACres & b){

if (a.size()!=b.size())

return 0;

ACres::const\_iterator ait;

ACres::const\_iterator bit = b.begin();

for (ait = a.begin(); (ait < a.end()) && (bit < b.end()); ait++, bit++){

if ((ait->index!=bit->index) || (ait->patternNumber!=bit->patternNumber))

return 0;

}

return 1;

}

1. Написан ряд юнит-тестов с использованием Google Tests, проверяющих работу процедур. Ошибки, выявляемые тестами, исправлены:

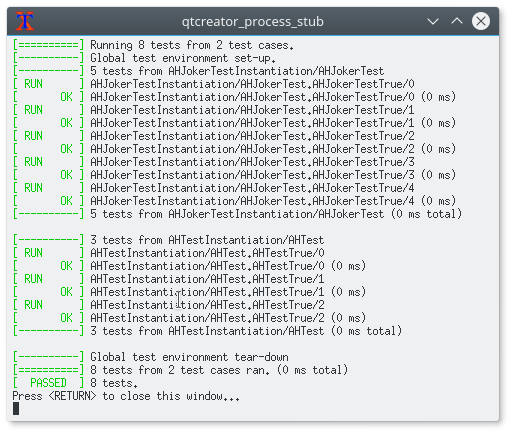


Рисунок 1 – тестирование

Содержимое тестов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название теста | Вход | | | Ожидаемый результат |
| AHTest1 | "abcdabcdabcd" | "a", "ab", "abc", "abcd" | | {{1,1}, {1,2},  {1, 3}, {1,4}, {5,1}, {5,2}, {5,3}, {5,4}, {9,1}, {9,2}, {9,3}, {9,4}} |
| AHTest2 | "CCCA" | "CC" | | {{1,1}, {2,1}} |
| AHTest3 | "AAAA", | {"A","AA"} | | {{1,1}, {1,2}, {2,1}, {2,2}, {3,1}, {3,2}, {4,1}} |
| AHJokerTest1 | "ACT" | "A$" | '$' | {{1,1}} |
| AHJokerTest2 | "xabvccbababcax" | "ab??c?" | '?' | {{2,1}, {8,1}} |
| AHJokerTest3 | "abacadaeaf" | "a?" | '?' | {{1,1}, {3,1}, {5,1}, {7,1}, {9,1}} |
| AHJokerTest4 | "abacadaeaf" | "aqaqaq" | 'q' | {{1,1}, {3,1}, {5,1}} |
| AHJokerTest5 | "qwerqwwrqqqrr" | "###r" | '#' | {{1,1}, {5,1}, {9,1}, {10,1}} |

1. После чего написаны функции для исполняемых модулей, соответствующие заданиям на stepik:

* **lab6** - принимает на вход строку, число образцов, образцы и выводит вхождения всех образцов в строку. Соответствует заданию 1 на stepik
* **lab6joker** - принимает на вход строку, образец и символ джокера. Находит все вхождения образца в строку. Соответствует заданию 2 на stepik

1. Для автоматизации сборки написан bash-скрипт, автоматически выполняющий необходимые операции и запускающий тесты

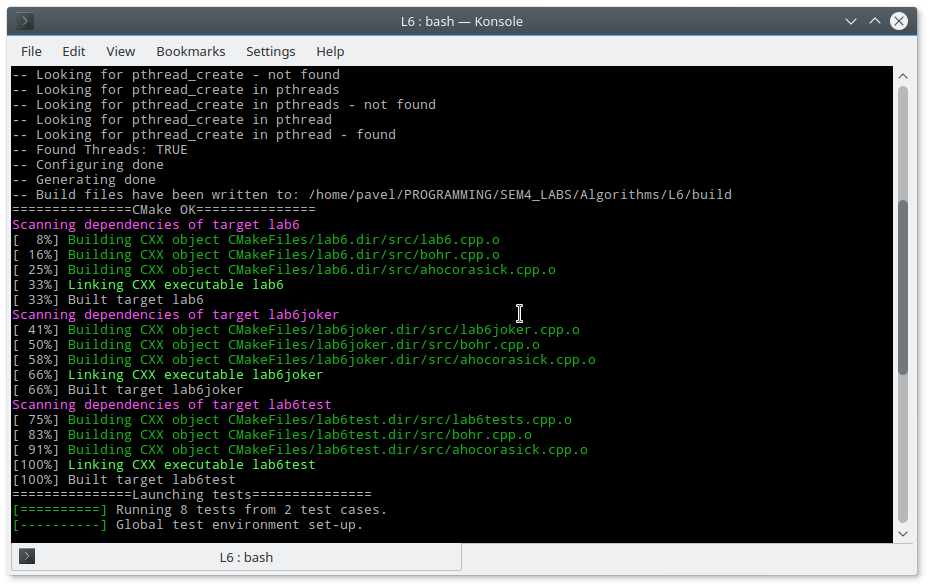


Рисунок 2 – сборка проекта в терминале

1. Выполены соответствующие задания на stepik.

**Вывод**

Изучена реализация алгоритма Ахо-Корасик на языке C++.

Исследованы возможности отладки программ с использованием GNU Debugger. Изучено использование проектов на основе CMake в IDE Qt Creator.