Московский авиационный институт
(национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Дискретный анализ»

Студентка: А. Довженко
Преподаватель: Д. Е. Ильвохин
Группа: 08-207
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №4

Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.

Вариант алгоритма: Поиск одного образца-маски: в образце может встречаться "джокер равный любому другому символу. При реализации следует разбить образец на несколько, не содержащих "джокеров найти все вхождения при помощи алгоритма Ахо-Корасик и проверить их относительное месторасположение.

Вариант алфавита: Числа в диапазоне от 0 до $2^{32} - 1$.

1 Описание

Поиск одного образца с джокерами сводится к множественному поиску с помощью алгоритма Ахо-Корасик с некоторой модификацией. В дерево ключей нужно поместить не один образец, а все подстроки образца, не содержащие джокеры. Затем построить для дерева связи неудач для каждой вершины, обойдя дерево в ширину. Связь неудач связывает в дереве две вершины, первая из которых это вершина, полученная после прохода по тексту и совпадении символов текста с каким-то образцом до следующей вершины, в которой найдено несовпадение, а вторая вершина – это вершина соответствующая символу другого образца, перфикс которого является максимальным суффиксом уже пройденного пути. Это существенно ускоряет поиск. Препроцессинг для построения связей неудач выполняется за $O(n)$, где n – суммарная длина всех подстрок образца без джокеров. Помимо всего прочего при построении дерева нужно найти длины подстрок образца, не содержащих джокеров, и их позиции начала в образце. Это понадобится в дальнейшем при поиске образца в тексте, для проверки относительного месторасположения подстрок.

Непосредственно при поиске нужно пройти по тексту и найти для каждого подобразца начальные позиции вхождений подобразцов в текст. Для каждого такого начала j подобразца P_i в T увеличиваем счетчик в ячейке $j - l_i + 1$ вектора C на единицу (изначально равен длине текста и инициализирован нулями). После окончания прохода текста, просматриваем вектор C в поисках ячеек со значением k (число подобразцов без джокеров). Вхождение P в T, начинающееся с позиции p, имеется только в том случае, если $C(p) = k$.

Это обеспечивает корректность данного метода. В самом деле, если существует вхождение подобразца P_i в текст T в позиции j из T, а подобразец P_i начинается в позиции l_i в P, то это дает одно "свидетельство что P входит в T, начиная с позиции $p = j - l_i + 1$. Следовательно, P входит в T, начиная с p в том и только в том случае, если такие показания для позици p получены от всех k подобразцов.
Задача сводится к тому, чтобы заполнить 2 вектора – образец и текст, на основе образца построить дерево ключей, создать связи неудач, а затем совершить поиск единственным проходом по тексту.

2 Исходный код

```
class TTrieNode
{
public:
    friend class TTrie;
    TTrieNode();
    virtual ~TTrieNode();
private:
    std::map<TUInt32, TTrieNode *> to;
    TTrieNode *linkFail;
    std::vector<int> out;
};

class TTrie
{
public:
    TTrie();
    void Create(const std::vector<std::string> &);
    void Search(const std::vector<TUInt32>&, const int&, std::vector<std::pair<int, int
        > >&);
    virtual ~TTrie();
private:
    TTrieNode *root;
    std::vector<int> lensPatterns;
    int withoutJoker;

    void CreateLinks();
};
```

3 Консоль

```
karma@karma:~/mai_study/DA/da4$ cat ./tests/01.t
213 ? 8 5 ?
49869 213 5 5 767 564 6969 5 8 8 5 981 8 69 213 8 8 86 5 938
314 8 65 11781 892 22114 9248 4823 44913 63632 8 5 65 5 74 213 13373 8 5 19
8 213 99642 8 5 5 3748 7734 56653 8 8494 5 58 4835 1 56771 213 8 213 62921
9 8152 6 213 213 23791 213 5 213 91167 5 61 8 5 8 5 213 213 64 5
5 8 8 8726 29669 1163 8 213 64987 93 213 8 9996 1 5 8 7156 8 213 7
56711 8 167 135 96 4289 5 58142 286 8 51827 14 213 3388 8341 8 213 24 5 213
8 14 3222 4833 9 213 8 213 5 12 7362 73 963 7 5 213 634 213 5 17678
8 9136 816 27 5 92947 213 89833 278 54646 5 8 87815 8 11729 213 8 93667 8 7186
5 8 35 5 213 213 51278 5 213 45 213 8 8 6 449 8 92 5 5 8
642 213 3345 83 8 3 6682 5 31332 213 213 8 5 7195 8 7159 145 8 213 8
karma@karma:~/mai_study/DA/da4$ cat ./tests/01.t | ./da4
2,16
3,2
10,10
karma@karma:~/mai_study/DA/da4$ cat ./tests/02.t
1 2 3 4 5 ? ? ? ?
1 2 3 4 5 6 9 12
1 2 3 4 5 7 10 13
1 2 3 4 5 8 11 14
karma@karma:~/mai_study/DA/da4$ cat ./tests/02.t | ./da4
1,1
2,1
```

4 Тест производительности

Я случайным образом генерирую десять файлов, которые представляют из себя следующее:

1 тест – образец состоит только из джокеров. Длина образца небольшая (содержит 10 джокеров). Текст небольшой (10 строк по 100 чисел в каждой строке).
2 тест – образец состоит только из джокеров. Длина образца как в 1 тесте. Текст большой (10000 строк по 100 чисел в каждой).
3 тест – образец состоит только из джокеров. Длина образца – 1000 чисел. Текст – 10000 строк по 100 чисел в каждой.
4 тест – образец не содержит джокеров. Длина образца небольшая (5 чисел). Текст небольшой (10 строк по 100 чисел в каждой).
5 тест – образец не содержит джокеров. Длина образца небольшая (5 чисел). Текст большой, 10000 строк по 100 чисел в каждой.
6 тест – образец не содержит джокеров. Длина образца 100 чисел. Текст – 10000 строк по 1000 чисел.
7 тест – текст состоит исключительно из образца. Образец состоит из 5 чисел. Текст – 10000 строк по 100 чисел в каждой.
8 тест – в тексте нет ни одного вхождения образца. Образец состоит из 5 чисел. Текст – 10000 строк по 100 чисел в каждой.
9 тест – числа и джокеры в образце чередуются. Образец состоит из 5 чисел и джокеров. Текст – 10 строк по 100 чисел в каждой.
10 тест – числа и джокеры в образце чередуются. Образец состоит из 5 чисел и джокеров. Текст – 10000 строк по 100 чисел в каждой.

Сравнение я буду проводить с поиском образца методом find в std::vector, который использует линейный поиск, т.е. проверяет каждое число в тексте.

```
karma@karma:~/mai_study/DA/da4$ cat ./tests/01.t | ./da4
Aho-Corasick search time is: 7e-06 sec.
std::find search time is: 1.1e-05 sec.
karma@karma:~/mai_study/DA/da4$ cat ./tests/02.t | ./da4
Aho-Corasick search time is: 0.002988 sec.
std::find search time is: 0.009999 sec.
karma@karma:~/mai_study/DA/da4$ cat ./tests/03.t | ./da4
Aho-Corasick search time is: 0.004671 sec.
std::find search time is: 0.011328 sec.
karma@karma:~/mai_study/DA/da4$ cat ./tests/04.t | ./da4
Aho-Corasick search time is: 2.1e-05 sec.
```

```
std::find search time is: 8e-06 sec.
karma@karma:~/mai_study/DA/da4$ cat ./tests/05.t | ./da4
Aho-Corasick search time is: 0.007607 sec.
std::find search time is: 0.004663 sec.
karma@karma:~/mai_study/DA/da4$ cat ./tests/06.t | ./da4
Aho-Corasick search time is: 0.115184 sec.
std::find search time is: 0.008494 sec.
karma@karma:~/mai_study/DA/da4$ cat ./tests/07.t | ./da4
Aho-Corasick search time is: 0.007354 sec.
std::find search time is: 0.024181 sec.
karma@karma:~/mai_study/DA/da4$ cat ./tests/08.t | ./da4
Aho-Corasick search time is: 0.080604 sec.
std::find search time is: 0.003783 sec.
karma@karma:~/mai_study/DA/da4$ cat ./tests/09.t | ./da4
Aho-Corasick search time is: 2.2e-05 sec.
std::find search time is: 1.1e-05 sec.
karma@karma:~/mai_study/DA/da4$ cat ./tests/10.t | ./da4
Aho-Corasick search time is: 0.010683 sec.
std::find search time is: 0.006776 sec.
```

По результатам проверки видно, что алгоритм Ахо-Корасик с джокерами показывает достойное время и соответствует заявленной сложности. Явный прогрыш во времени заметен в 6 и 8 тестах. В 6 тесте в результате препроцессинга получим дерево ключей, состоящее из одного пути длины 100 (т.к. джокеров нет), из-за чего, видимо, и получается такое относительно большое время. Кроме того, как оказалось, в тексте нет ни одного вхождения образца. Как и в 8 тесте. Числа в этих тестах генерируются от 1 до 5 знаков. Большая часть переходов по связям неудач будут вести в корень, и сдвиг будет минимален. find должен был показать в этих тестах похожее время, но он не тратит время на препроцессинг. Отсюда такой результат.

5 Отладка

1 посылка на чекер: OK
2 посылка на чекер: OK
Были устранены утечки при работе с памятью.

6 Выводы

Применения алгоритмов поиска подстрок слишком очевидны, чтобы думать о них: задача эта столь частая, что встречается, пожалуй, в любом приложении, занимающемся хоть какой-то работой с текстом.

В лабораторной мне было предложено решить задачу поиска с помощью вариации алгоритма Ахо-Корасик для образца с джокерами. Время требуемое на построение дерева ключей – $O(n)$, где n – суммарная длина всех подобразцов, не содержащих джокера. Если число джокеров ограничено фиксированной постоянной (не зависящей от размера образца), то решение задачи поиска возможно за линейное время $O(n + m)$, где n – суммарная длина всех подстрок образца без джокеров, m – длина текста. В противном случае неизвестно, возможно ли решение за линейное время. В моей реализации для отображения образца в дерево используется map, которая реализована на красно-черном дереве. Время обращения к его элементам пропорционально логарифму числа элементов, а это константа. Время поиска вхождений составляет $O(m + z)$, z – число вхождений. Число джокеров ограничено, независимо от размера образца, поэтому алгоритм работает за линейное время.

Этот алгоритм имеет практическое применение. Важный случай, где встречаются джокеры, – это факторы транскрипции ДНК. Фактор транскрипции – белок, который закрепляется в определенных местах ДНК и регулирует, усиливая или падавляя, транскрипцию ДНК в РНК. Так регулируется синтез белка, для которого, собственно, и предназначены коды ДНК.

Вообще, использование алгоритмов поиска на практике вопрос неоднозначный. Возможно, удобнее решать эту задачу с помощью регулярных выражений, потому что алгоритм компиляции регэкспов в автомат понятнее и регулярные выражения больше позволяют. К тому же, нынче их умеют реализовывать в высшей мере эффективно.