Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Дискретный анализ»

Студент: С. М. Бокоч
Преподаватель: А. А. Кухтичев
Группа: М8О-204Б

Дата:
Оценка:
Подпись:

Москва, 2018

Лабораторная работа №5

Задача: Необходимо реализовать алгоритм Укконена построения суффиксного дерева за линейное время. Построив такое дерево для некоторых из входных строк, необходимо воспользоваться полученным суффиксным деревом для решения своего варианта задания.

Алфавит строк: строчные буквы латинского алфавита (т.е., от a до z). Входные данные: текст располагается на первой строке, затем, до конца файла, следуют строки с образцами.

Выходные данные: для каждого образца, найденного в тексте, нужно распечатать строчку, начинающуюся с последовательного номера этого образца и двоеточия, за которым, через запятую, нужно перечислить номера позиций, где встречается образец в порядке возрастания.

Вариант алгоритма: поиск с использованием суффиксного массива.

1 Метод решения

Алгоритм **Укконена** строит неявное суффиксное дерево строки S с m символами для каждого префикса $S[1..i]$ строки S в режиме *online*, то есть происходит m фаз расширения суффикного дерева.

Неявное суффиксное дерево — суффиксное дерево для текста S без терминирующего символа(\$) на конце. Некоторые суффиксы текста в нем заканчиваются не в листьях, и их номер нигде не хранится.

Определение. Суффиксное дерево T для *m-символьной строки* S:

1. Ориентированное дерево, имеющее ровно m листьев, пронумерованных от 1 до m.
2. Каждая внутренняя вершина, отличная от корня, имеет не меньше двух детей.
3. Каждая дуга помечена непустой подстрокой строки S(дуговая метка).
4. Никакие две дуги, выходящие из одной вершины, не могут иметь меток, начинающихся с одинаковых символов.
5. Для каждого листа i конкатенация меток от корня составляет $S[i..m]$.

Тонкости алгоритма: добавляется терминальный символ, который больше нигде в строке S не встречается('\$'), потому что может возникнуть проблема: если будет существовать суффикс, совпадающий с префиксом другого суффикса, то не будет выполнено условие о количестве листьев.

Определение. Неявное суффиксное дерево может быть получено из суффиксного дерева строки S:

1. удалением всех вхождений терминального символа;
2. затем удалением всех дуг без меток;
3. затем удалением всех вершин, имеющих меньше двух детей(кроме корня).

Простыми преобразованиями из суффиксного дерева можно получить алгоритм Укконена, который работает за **линейное время**.

Правила продолжения суффиксов:

1. В текущем дереве путь x кончается в листе. Это значит, что путь от корня с меткой x доходит до конца некоторой «листовой» дуги. При изменении дерева нужно добавить к концу метки этой листовой дуги символ *S[i+1]*.

2. Ни один путь из конца строки x не начинается символом *S[i+1]*, то по крайней мере один начинающийся оттуда путь имеется. В этом случае должна быть создана новая листовая дуга, начинающаяся в конце x и помеченная символом *S[i+1]*. При этом, если x кончается внутри дуги, должна быть создана новая вершина. Листу в конце листовой дуги сопоставляется номер *j*.

3. Некоторый путь из конца строки x начинается символом *S[i+1]*, то строка *xS[i+1]* уже имеется в текущем дереве, так что ничего не надо делать.

Определение. Пусть *xa* обозначает произвольную строку, где x – её первый символ, а a – оставшаяся подстрока. Если для внутренней вершины u с путевой меткой *xa* существует другая вершина *s(u)* с путевой меткой a, то указатель из u в *s(u)* называется **суффиксной связью**.
Первое ускорение. Суффиксные связи. Алгоритм отдельного продолжения. Скачок по счетчику. $\Theta(\mathrm{n}^2)$.

Второе ускорение. Сжатие дуговых меток. $\Theta(\mathrm{n})$.

Суффиксный массив — лексикографически отсортированный массив всех суффиксов строки. В данной структуре можно находить все вхождения подстроки строки S за время $\Theta(n \log n)$ помощью двоичного поиска. Суффиксный массив работает немного медленнее алгоритма Укконена, но требует намного меньше памяти $\Theta(n)$.

Преобразовать суффиксный массив в суффиксное дерево можно с помощью рекурсивного обхода от корня, до каждого листа дерева.
При поиске образца в тексте я использовал функцию из библиотеки algorithm – equal_range, которая возвращает пару итераторов: первый представляет значение итератора, возвращаемое алгоритмом lower_bound(), второй – алгоритмом upper_bound(). В параметрах передается искомое значение и функция сравнения. Использовал лямбда-функцию, ввиду простоты и компактности.

2 Описание программы

suffix_tree.h	
Класс TSuffixTree.	
private:	
std::string text;	Текст, содержащийся в дереве.
TNode *root;	Указатель на корень дерева.
TNode *active_vertex;	Активная вершина при вставке.
std::size_t activeLength;	Количество символов для разбиения дуги.
std::size_t activeCharIdx;	Символ для разбиения.
void DFS(...) const;	Преобразование дерева в суффиксный массив.
public:	
explicit TSuffixTree(...)	Создание суффиксного дерева. Принимает алфавит и терминирующий символ.
void PushBack(const char &ch);	Добавление одного символа в режиме *online*.
virtual ~TSuffixTree();	Деструктор.
Класс TSuffixArray.	
private:	
std::string text;	Текст, содержащийся в массиве.
std::vector<std::size_t> array;	Отсортированный массив суффиксов.
public:	
explicit TSuffixArray(TSuffixTree &);	Преобразование в суффиксный массив.
... Find(const std::string &pattern) const;	Возвращает отсортированный массив индексов всех вхождений в текст.
virtual ~TSuffixArray();	Деструктор.

3 Консоль

```
bokoch@MacKenlly ~/DA/lab5 $ make
g++ -c -std=c++14 -Werror -pedantic -Wall -Wextra -O2 main.cpp -o main.o
g++  suffix_tree.o main.o -lm -o lab5
bokoch@MacKenlly ~/DA/LW_5 $ ./lab5
abcabxabcd
a
1: 1, 4, 7
abc
2: 1, 7
bc
3: 2, 8
d
4: 10
xa
5: 6
te
e
a
8: 1, 4, 7
bokoch@MacKenlly ~/DA/LW_5 $ ./lab5
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
a
1: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35
sd
asa
aaaaaaaaaaaaaaaaaaaaa
4: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
bokoch@MacKenlly ~/DA/LW_5 $ ./lab5
knuthandpruanic
an
1: 6, 12
knut
2: 1
```

4 Тест производительности

Тест производительности представляет из себя следующее: сравнение скорости выполнения поиска множества образцов в заданном тексте с помощью метода **std::string::find** и построения суффискного дерева для текста алгоритмом Укконена с последующим поиском в суффиксном массиве.

```
bokoch@MacKenlly ~/DA/lab5 $ ./lab5 <tests/01.t
Text size: 1000 characters
Quantity patterns length 100 characters: 1000
Suffix array: 9
std::string::find: 6
bokoch@MacKenlly ~/DA/lab5 $ ./lab5 <tests/02.t
Text size: 10000 characters
Quantity patterns length 100 characters: 1000
Suffix array: 32
std::string::find: 49
bokoch@MacKenlly ~/DA/lab5 $ ./lab5 <tests/03.t
Text size: 100000 characters
Quantity patterns length 100 characters: 10000
Suffix array: 3398
std::string::find: 7411
```

На небольших входных данных алгоритм работает медленнее. Это связано с затратами времени на преобразование суффиксного дерева в суффиксный массив. Из этого можно сделать вывод, что чем больше дерево, тем эффективней будет осуществляться поиск по сранению с функцией std::string::find из стандартной библиотеки STL.

5 Дневник отладки

1. 27.12.17; 14:45; Начал читать книгу Гасфилда по суффиксным деревьям.
2. 28.12.17; 6:12; Написал алгоритм Укконена, работающий за $\Theta(n)$.
3. 28.12.17; 16:04; Отложил суффиксные деревья, взялся реализовывать суффиксный массив.
4. 29.12.17; 10:24; Суффиксный массив ищет только одно вхождение в текст. Использование RMQ.
5. 30.12.17; 16:37; Написан алгоритм Укконена, работающий за линейное время.
6. 9.01.18; 20:55; Отлаживание ошибок, с помощью профайлеров.
7. 18.01.18; 23:31; Написание бэнчмарка и оформление программы по кодстайлу.

6 Выводы

Суффиксные деревья без сомнений эффективны при поиске образцов в строке. Особенно преимущество проявляется на больших данных(на моем компьютере от 10^6). Также алгоритм Укконена является полезным для решения различых прикладных задач, таких как поиск наибольшей подстроки, нечёткого поиска, выделение повторяющихся фрагментов и других. Но главный минус состоит в потребляемой памяти, поэтому в действие вступают суффиксные массивы, которые также способны решать задачу о подстроке почти так же эффективно, как суффиксное дерево. К сожалению, мне не до конца удалось реализовать поиск подстроки в тексте с помощью суффиксного массива за время $\Theta(\mathrm{n} + \log m$ при помощи использования массива lcp для ускорения поиска. Плюсом стоит отметить простоту реализации суффиксного массива, в отличии от алгоритма Укконена.

Список литературы

[1] Дэн Гасфилд, «Строки, деревья и последовательности в алгоитмах: Информатика и вычислительная биология», 2003 Глава 2, «Точное совпадение: клас ические методы», глава 3, «Более глубокий взгляд», стр. 19-94.(дата обращения: 28.12.2017).

[2] *Алгоритм Укконена*
URL:`http://neerc.ifmo.ru/wiki/ukkonen-algo` (дата обращения: 10.01.2018).

[3] *Визуализатор суффиксного дерева*
URL:`http://brenden.github.io/ukkonen-animation/` (дата обращения: 10.01.2018).

[4] *Алгоритм поиска в суффиксном массиве*
URL:`https://habrahabr.ru/post/115346/` (дата обращения: 8.10.2018).