Лекция по суффиксным структурам Moscow International Workshop ACM ICPC 2015

Александр Кульков

19 ноября 2015

1 Введение

Рассмотрим задачу поиска образцов в тексте. Одна из её возможных формулировок выглядит так: дан текст T и n образцов S_i. Для каждого из образцов необходимо сказать, встречается ли он в тексте. Существует два принципиально разных подхода к решению такой задачи. Первый (считается проще) — алгоритм Ахо-Корасик, который строит для набора образцов автомат, распознающий строки, которые содержат данные образцы как подстроки. Второй (считается сложнее) — использование суффиксных структур, под которыми обычно подразумевают суффиксный массив, суффиксный автомат или суффиксное дерево. Данная лекция по большей части будет посвящена последним двум из них, их связи и применении в решении задач.

2 Наивное решение

Рассмотрим произвольную подстроку $s[pos, pos + len - 1]$. Она является префиксом длины len суффикса строки, который начинается в позиции pos. Учитывая это, можно использовать следующее наивное решение для поставленной задачи: возьмём все суффиксы строки и объединим их в префиксное дерево (бор). Тогда в силу структуры префиксного дерева, каждому префиксу любого суффикса будет соответствовать ровно одна вершина и мы сможем за $O(|S_i|)$ проверять, входит ли строка S_i в текст.

Недостатки очевидны — такое решение требует $O(|T|^2)$ времени и памяти. Существует два пути исправления данной проблемы, которые ведут соответственно к суффиксному дереву и суффиксному автомату.

3 Суффиксное дерево

Мы можем заметить, что любой нисходящий путь в таком боре будет подстрокой строки T. Тогда можно выкинуть из бора вершины, которые не являются корнем или вершиной, соответствующей суффиксу и степень которых при этом равна 2 (то есть, вершины, которые не являются развилками — в них одно ребро входит сверху и одно идёт вниз). Вместо пути из таких вершин, который соединяет две вершины-вилки, мы можем записать на ребре границы подстроки, которой он соответствует. Такая сжатая структура называется суффиксным деревом. Пример суффиксного дерева приведён далее.

Покажем, что суффиксное дерево занимает $O(|T|)$ памяти. Будем последовательно добавлять суффиксы, сразу поддерживая бор сжатым. Тогда на каждом этапе мы добавим либо одну, либо две новые вершины. Действительно, если мы добавили первую вершину, то нам пришлось расщепить какое-то ребро. Очевидно, после этого мы ничего расщеплять не сможем, т.к. после этого мы просто сразу подвесим к дереву очередной лист.

Существует несколько алгоритмов быстрого построения суффиксного дерева. Наиболее известный из них — алгоритм Эско Укконена, однако, на данной лекции из-за своей сложности он рассматриваться не будет.

4 Суффиксный автомат

Если задавать автомат формально, то детерминированный конечный автомат (ДКА) — это пятёрка $A = (Q, \Sigma, \delta, q_0, F)$. Q — множество состояний, Σ — алфавит, δ — множество переходов, q_0 — начальное состояние, F — множество финальных состояний.

Мы же будем рассматривать автомат как ориентированный граф, в котором на каждой дуге написана буква (дуги в автомате называют переходами, а вершины — состояниями). Кроме того из одного состояния не может быть переходов по одному и тому же символу (что и означает детерминированность).

Будем говорить, что состояние q принимает строку s, если существует путь из начального состояния в q, такой что если последовательно выписать все символы, которые мы встретили на этом пути, то мы получим строку s. В силу детерминированности это биекция — каждому пути соответствует строка и обратно каждой строке соответствует путь. Автомат принимает строку s, если её принимает хотя бы одно из финальных состояний.

Итак, суффиксным автоматом строки s будем называть *минимальный* автомат, который принимает все суффиксы строки и только их. Под минимальностью подразумевается, что число состояний в нём должно быть наименьшим возможным. Заметим, что он будет являться не только ориентированным, но и ациклическим графом, т.к. множество принимаемых им строк конечно (а если бы был цикл, мы могли бы сколь угодно нарастить какую-то строку).

Пример суффиксного автомата для строки *abbcbc*, а также суффиксного дерева для строки *cbcbba* (на первой картинке двойной обводкой обозначены финальные состояния):

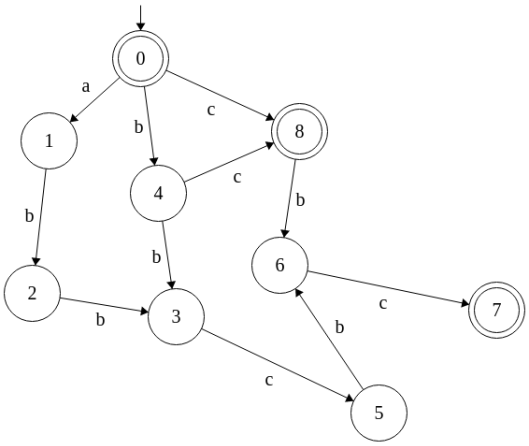

Рис. 1: Автомат для *abbcbc*

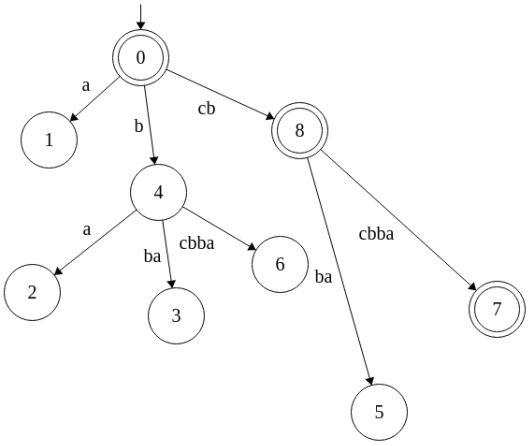

Рис. 2: Суффиксное дерево *cbcbba*

Как вы уже могли догадаться, эти две структуры неслучайно размещены рядом. Изучив автомат слева, вы легко сможете понять, что множество строк, которое принимает состояние k в точности совпадает с множеством развернутых строк, для которых соответствующая вершина в суффиксном дереве находится на ребре, которое ведёт в вершину k дерева. Данный факт отнюдь не является совпадением и будет рассмотрен ниже.

Чтобы иметь возможность строить и использовать данную структуру, жизненно необходимо понимать, как она устроена.

Первый тривиальный факт, к которому мы можем прийти — если две строки a, b принимаются одним состоянием q произвольного автомата, то для любой строки x строки ax и bx принимаются или не принимаются автоматом одновременно. Действительно, независимо от того, как мы "пришли" в состояние q, если мы пройдём из него по пути, соответствующему строке x, мы сможем точно сказать, в какое состояние мы попадём (в частности будет ли оно финальным).

Значит, любому состоянию q соответствует множество строк $X(q)$, которые переводят его в одно из конечных состояний. Данное множество называют правым контекстом состояния. Оно определено не только для состояния, но и для строк, которые оно принимает (их правый контекст совпадает с правым контекстом состояния). Отсюда можно сделать вывод, что состояний в автомате *не меньше*, чем различных правых контекстов у подстрок строки, для которой он построен (т.к. любой строке, которую можно дополнить до принимаемой соответствует какой-то путь в автомате, а значит, какое-то состояние).

Покажем, что в минимальном автомате эта нижняя оценка достигается. Допустим, что в автомате есть два состояния q_1, q_2 такие что $X(q_1) = X(q_2)$. Мы можем удалить состояние q_2 и перевести переходы, ведущие в него в состояние q_1. Очевидно, множество принимаемых строк от этого не изменится, следовательно, мы можем продолжать эту процедуру, пока число состояний не будет равно числу различных правых контекстов. Таким образом, *ДКА является минимальным тогда и только тогда, когда правые контексты всех его состояний попарно различны.*

Наконец, зная данные факты, верные для произвольных автоматов, вернёмся к автомату суффиксному.

Довольно легко понять, что в случае суффиксного автомата правый контекст $X(a)$ строки a взаимооднозначно соответствует множеству правых позиций вхождений строки a в строку s. Действительно, если ax принимается автоматом, то есть, является суффиксом, то $s = yax$, а строке x мы можем сопоставить позицию $|s|-|x|-1$. Таким образом, каждое состояние автомата принимает строки с одинаковым множеством *правых* позиций их вхождений и обратно, все строки с таким множеством позиций принимаются этим состоянием.

5 Связь между суффиксным автоматом и суффиксным деревом

Рассмотрим ребро в суффиксном дереве строки s, а точнее все подстроки s, которым соответствует "внутренняя" вершина ребра (т.е., одна из "выброшенных" при сжатии) или вершина, в которую ведёт ребро. Покажем, что *для любой строки x и любой пары строк a, b из рассматриваемого множества строк, строки xa и xb являются или не являются префиксами строки s одновременно* (иначе говоря, их множества *левых* позиций вхождений совпадают).

Пусть $|a| < |b|$. Тогда a является предком b в боре, то есть, её префиксом, значит, её множество вхождений точно содержит множество вхождений строки b. Допустим, существует позиция $|x|$, в которой есть вхождение строки a, но не строки b. Рассмотрим строку a', которая является максимальным префиксом строки b, который можно встретить в той позиции. Если a' не упирается в конец строки, то её можно продолжить, как минимум, двумя различными символами и она останется подстрокой s. Значит, соответствующая ей вершина в дереве имеет степень больше двух и должна разбивать ребро, на котором находится, что конфликтует с предположением о том, что a и b взяты с одного ребра. Если же a' нельзя продолжить, то она всё ещё должна разбивать ребро, т.к. является суффиксом строки и её вершина не будет удалена при сжатии рёбер.

Аналогичным образом от противного можно показать, что для любой строки a все строки b с таким же множеством левых позиций вхождений находятся на соответствующем строке a ребре. Таким образом, мы получи-

ли, что *для любого состояния q суффиксного автомата строки s найдётся вершина q' суффиксного дерева развернутой строки s такая, что множество строк, принимаемых состоянием q, совпадает с множеством строк, таких что соответствующая им вершина в дереве лежит на ребре, ведущем в q' (включая строку, соответствующую q').*

Отсюда мгновенно вытекают такие свойства, основанные на структуре развернутых подстрок в суффиксном дереве:

1. Если строки a и b принимаются состоянием q и $|a| < |b|$, то a — суффикс b.
2. Если строки a и b принимаются состоянием q и $|a| < |b|$, то состоянием q также принимаются все строки c такие что $|a| < |c| < |b|$ и c является суффиксом b.
3. Рассмотрим строки a и b, $|a| \leq |b|$. Если a — суффикс b, то $X(b) \subseteq X(a)$ или же $X(a) \cap X(B) = \varnothing$ в противном случае.

Таким образом, мы полностью описали состояния автомата.

6 Построение суффиксного автомата

Наконец, рассмотрим алгоритм построения суффиксного автомата. Для этого введём понятие суффиксной ссылки. Пусть длина самой короткой строки, которая принимается состоянием q равна k. Тогда суффиксная ссылка $link(q)$ будет вести из этого состояния в состояние, которое принимает эту же строку без первого символа. Вновь обратимся к суффиксному дереву и поймём, что таким образом в нём она будет вести в предка вершины q. Таким образом, суффиксные ссылки образуют дерево, которое соответствует суффиксному дереву развернутой строки. Также будем обозначать длину самой длинной строки, которая принимается состоянием q как $len(q)$. Очевидно, длина самой короткой строки из q в таком случае будет равна $len(link(q)) + 1$. Просто по определению суффиксной ссылки.

Будем дописывать символы в конец строки s по одному и при этом поддерживать для неё корректный автомат с деревом суффиксных ссылок. Пусть у нас есть автомат для строки s и она принимается его состоянием $last$. Мы хотим получить автомат для строки sc. Нам нужно, чтобы для каждого суффикса новой строки существовало состояние, которое его примет. При этом нам нужно сохранить минимальность автомата.

Добавим состояние, которое принимает всю строку sc и назовём его new. Правый контекст строки sc, очевидно, состоит из единственной строки — пустой, значит, в new будут входить те и только те суффиксы, которые мы встретили в строке впервые. Все такие строки можно получить дописыванием символа c к суффиксам s, которые принимаются состоянием, из которого ещё нет перехода по данному символу. Таким образом, чтобы новые суффиксы принимались, нам необходимо будет “попрыгать” по суффиксным ссылкам и добавить переходы по символу c в состояние new, пока не

придём в корень или не обнаружим, что переход по символу c из состояния уже есть.

Если мы пришли в корень, значит, *все* непустые суффиксы строки sc принимаются состоянием new и мы можем положить $link(new) = q_0$ (напоним, q_0 — начальное состояние автомата) и завершить работу.

Иначе мы нашли состояние q', из которого переход по символу c уже есть. Значит, суффиксы длины $\leq len(q') + 1$ уже встречались в строке, и новых переходов в состояние new мы проводить не будем. Однако, для состояния new ещё нужно посчитать суффиксную ссылку. Наибольшей строкой в ней будет суффикс строки sc длины $len(q') + 1$. В данный момент он находится в состоянии t, в которое ведёт переход по символу c из состояния q'. Однако, в данный момент в нём могуть быть также строки большей длины. Таким образом, если $len(t) = len(q')+1$, то t и есть искомая суффиксная ссылка. Проведя её, мы завершим обновление автомата.

Иначе t — состояние, которое принимает как строки, являющиеся суффиксами строки, так и строки, которые ими не являются. Из-за этого мы не можем определить его финальность. Чтобы решить данный конфликт мы должны будем отщепить от t состояние t', которое примет все строки, которые принимаются t, но имеют длину $\leq len(q') + 1$, то есть, тот самый кусок с суффиксами. Для этого скопируем в t' переходы и суффиксную ссылку из t, а длину установим равной $len(q') + 1$. Затем установим $link(new) = link(t) = t'$. Наконец, чтобы "перебросить" в него нужные строки из t, пройдёмся по суффиксным ссылкам состояния q' пока переходы по c ведут в t и переправим эти переходы в t'. Таким образом, мы перенаправим все пути интересующей нас длины, которые ранее проходили через вершину t в вершину t'. На этом построение автомата заканчивается.

В некотором смысле эта процедура соответствует построению суффиксного дерева добавлением суффиксов в возрастающем (по длине) порядке.

Приведём код на языке C++, выполняющий эту процедуру:

```
const int maxn = 2e5 + 42; // Максимальное число состояний
map<char, int> to[maxn]; // Переходы
int link[maxn]; // Суффиксные ссылки
int len[maxn]; // Длины максимальных строк в состояниях
int last = 0; // Состояние, соответствующее всей строке
int sz = 1; // Общее число состояний

void add_letter(char c) // Дописываем символ в конец
{
    int p = last; // Записываем в p состояние строки s
    last = sz++; // Создаём для строки sc новое состояние
    len[last] = len[p] + 1;
    for(; to[p][c] == 0; p = link[p]) // (1)
      to[p][c] = last; // Прыгаем по ссылкам, создавая переходы
    if(to[p][c] == last)
    { // Если мы оказались здесь, то символ c встречен впервые
        link[last] = 0;
      return;
    }
    int q = to[p][c];
    if(len[q] == len[p] + 1)
    { // Если переход сплошной, то q - суфф ссылка
      link[last] = q;
      return;
    }
    // Расщепляем q на два состояния, одно из которых cl,
    // А второе получит тот же номер, что и q имело ранее
    int cl = sz++;
    to[cl] = to[q]; // (2)
    link[cl] = link[q];
    len[cl] = len[p] + 1;
    link[last] = link[q] = cl;
    for(; to[p][c] == q; p = link[p]) // (3)
      to[p][c] = cl; // Перенаправляем переходы там, где нужно
}
```

7 Время работы

Покажем линейность работы данного алгоритма. На каждом шаге есть три места, которые выполняются не за $O(1)$:

1. Прыжки по ссылкам $last$ для создания переходов в new.
2. Копирование переходов из t в t'.
3. Прыжки по ссылкам q' для перенаправления переходов из t в t'.

В первых двух пунктах происходит создание очередного перехода в автомате. Покажем, что всего переходов будет $O(n)$. Разделим все переходы $\delta(v, c) = u$ из v в u по символу c на два класса — "сплошные", для которых $len(v) + 1 = len(u)$ и все остальные.

Т.к. в каждое состояние, кроме начального, ведёт ровно один сплошной переход, сплошных переходов будет не больше, чем состояний, а их $O(n)$.

Рассмотрим теперь несплошные переходы. Каждому такому переходу можно поставить в соответствие строку acb, где a — длиннейшая строка, которую принимает состояние v, а b — длиннейшая строка, которую можно вывести из состояния u. Данная строка является суффиксом s (иначе мы могли бы продлить b вправо). Кроме того $|a| = len(v)$, отсюда можно сделать вывод, что $|a|$ составлена исключительно из сплошных переходов. Значит, по произвольному суффиксу мы можем определить рассмотренный несплошной переход, как первый, который встретим, "скармливая" его строке. Значит, такое отображение взаимооднозначное и сплошных переходов не больше, чем суффиксов в строке. Отсюда следует, что суммарно переходов в автомате $O(n)$.

Наконец, докажем линейность третьего пункта. Для удобства назовём $link(link(q))$ второй суффиксной ссылкой состояния q. Также будем использовать такие обозначения: $last$ — состояние, соответствующее строке s, new — состояние строки sc, p — состояние, которое "прыгает" в циклах (можно видеть в коде). Т.к. sc не могла встречаться в суффиксном автомате s, из состояния $last$ изначально нет перехода по c, поэтому в цикле (1) мы сделаем хотя бы один шаг. Отсюда $len(link(p)) \leq len(link(link(last)))$ (действительно, изначально $p = last$, после первой итерации $p = link(last) \to len(link(p)) = len(link(link(last)))$, на всех последующих итерациях длина $len(link(p))$ не увеличивается, значит, упомянутое неравенство верно.

Когда мы вышли из цикла (1), мы имеем $\delta(p, c) = q$. Очевидно, если есть переход из состояния p в состояние q, то если мы допишем символ c к самой короткой строке из p, длина которой равна $len(link(p)) + 1$, то получим строку, которая будет не короче, чем самая короткая строка, принимаемая состоянием q, длина которой равна $len(link(q))+1$. То есть, $len(link(p))+2 \geq len(link(q)) + 1 \to len(link(p)) + 1 \geq len(link(q))$.

После выхода из цикла (1) куда бы мы ни поставили суффиксную ссылку из new, вторая суффиксная ссылка точно будет $link(q)$. Отсюда $link(q) = link(link(new)) \to len(link(q)) = len(link(link(new)))$.

Теперь посмотрим на цикл (3). Он будет выполняться пока $\delta(p, c) = q$, то есть, как было упомянуто выше $len(link(p)) + 1 \geq len(link(q))$. При этом изначально $len(link(link(last))) \geq len(link(p))$. Так как $len(link(q)) = len(link(link(new)))$, а также на каждом шаге $len(link(p))$ уменьшается, получаем, что весь цикл отработает не дольше, чем за $len(link(link(last))) - len(link(link(new)))$, то есть, за разность максимальных длин, принимаемых вторыми суффиксными ссылками состояний, соответствующих всей строке.

Наконец, собирая всё полученное воедино, получим такое неравенство: $len(link(link(last)))+1 \geq len(link(p))+1 \geq len(link(q)) = len(link(link(new))$, отсюда $len(link(link(last)))+1 \geq len(link(link(new)))$, которое означает, что на каждом шаге длина второй суффиксной ссылки либо уменьшилась, либо увеличилась не больше, чем на единицу (а значит, её суммарное уменьшение не превосходит $O(n)$). Линейность алгоритма доказана!

8 Применение в решении задач

1. **Число различных подстрок.** Дана строка s, необходимо посчитать количество её различных подстрок. В каждом состоянии встречаются строки длины от $len(link(q)) + 1$ до $len(q)$. Всего $len(q) - len(link(q))$ строк. Просуммировав эту величину по всем состояниям, получим ответ.

   Упражнение: решите эту же задачу за $O(n)$, учитывая, что к строке s символы дописываются по одному и после каждого нового символа необходимо сказать текущее число различных подстрок строки s.

   Упражнение:* Возьмём задачу из предыдущего упражнения и скажем, что теперь мы можем не только дописывать символы *в конец*, но и удалять их *с начала* строки. Вам требуется отвечать на те же запросы. Время работы решения всё ещё должно линейно зависеть от размера входа. *Подсказка:* иногда алгоритм Укконена также бывает полезен.

2. **Поиск подстрок в тексте.** Пропустив строку через автомат мы сможем сказать, входит ли она в текст. Допустим, мы хотим узнать какую-то информацию о её вхождениях. Например, нам нужно выдать любое конкретное вхождение. Как мы уже знаем, каждому вхождению соответствует строка x такая что ax — суффикс s. Или, проще говоря, путь из состояния q в какое-то финальное состояние. Динамикой по автомату как ациклическому ориентированному графу мы можем найти длину какого-нибудь такого пути (например, для определённости минимального или максимального). Отметим, что аналогичной динамикой считаются многие другие полезные значения, например, количество строк в правом контексте состояния (или, что то же самое, количество вхождений строк из состояния в s).

   Альтернативным решением будет обратиться к дереву суффиксных ссылок, которое, как мы помним, является суффиксным деревом для s^T. Как мы упоминали в самом начале, любая подстрока строки s является префиксом одного из суффиксов исходной строки. Таким образом, если мы запишем в каждую "суффиксную" вершину индекс соответствующего ей суффикса, то все позиции вхождений строки t в s можно будет обнаружить в поддереве вершины, которая соответствует строке t. Значит, в частности, динамикой можно будет найти самое первое или самое последнее вхождение.

   Более того, учитывая, что в последнем случае мы работали с деревом, мы можем обойти его таким образом, чтобы на каждом шаге иметь в вершине множество возможных позиций, в которых встречаются строки из соответствующего состояния. Для этого нужно применить идею быстрого слияния множеств, когда мы всегда добавляем элементы из меньшего множества в большее, а не наоборот. Тогда такой обход потребует $O(n \log n)$ операций добавления в множество, т.к. каждый раз

когда мы переносим между множествами элемент k, размер нового множества будет как минимум, в два раза больше старого, в котором он хранился.

Упражнение:* дана строка s. Найти число строк t таких, что они имеют хотя бы 3 *непересекающихся* вхождения в строку s.

3. **Наибольшая общая подстрока.** Нам дано k строк $s_1, s_2, \ldots, s_k$. Нужно найти наибольшую строку t, которая встречается в каждой из строк s_i. Одно из возможных решений — построить автомат для строки $s_1 t_1 s_2 t_2 \ldots s_n t_n$, где t_i — уникальный для каждой строки символ-разделитель. Теперь мы можем завести динамику $dp[q][i]$, в которой хранить 1, если из состояния q можно добраться до состояния, из которого есть переход по t_i, не проходя при этом через другие символы-разделители. Это будет равносильно тому, что строки из q входят в s_i. Как и в прошлый раз, динамику можно пересчитывать по топологической сортировке автомата как ориентированного ациклического графа. Итого решение будет работать за $O(k \cdot \sum |s_i|)$.

   Упражнение:* решить указанную задачу за $O(\sum |s_i|)$.