Краткий конспект

Лекция 4. Суффиксные деревья

версия 0.1(draft)

Д. Ищенко* Б. Коварский* И. Алтухов* Д. Алексеев*

7 марта, 2016

*МФТИ

1 Задача поиска k мотивов в геноме

Вернемся к задаче поиска мотива m в геноме g. В лекции о Z-алгоритме мы выяснили, что ее можно решить за линейное время $O(|m|+|g|)$, где $|m|$ - длина мотива, $|g|$ - длина генома. Представим, что необходимо найти k различных мотивов в геноме g. Используя предыдущий метод, мы будем конструировать «Z-ящики» для каждой объединенной строки $m\$g$ и производить поиск. Соответственно временная сложность алгоритма вырастет до $O(k(|m| + |g|))$, в виду того, что $|m| \ll |g|$, сложность - $O(k|g|)$.

В реальных задачах k может иметь достаточно большое значение, например, при картировании ридов (коротких последовательностей) на референсный геном, значения k могут быть порядка $10^6 - 10^8$. Возникает необходимость оптимизации алгоритма. В этой лекции мы рассмотрим такую конструкцию, как суффиксное дерево и покажем, как с помощью него решить задачу поиска мотива и несколько других задач, существенно уменьшив временную сложность вычислений относительно Z-алгоритма.

2 Суффиксное дерево

По аналогии с ранее введенными префиксами строки S длиной n, назовем **k-ым суффиксом** строки S ее подстроку, начинающуюся с k-го и заканчивающуюся последним символом: $S[k..n]$. Например, для строки TATATG все множетсво суффиксов будет представлять из себя:

```
1: TATATG
2: ATATG
3: TATG
4: ATG
5: TG
6: G
```

Отметим, что количество всех суффиксов строки S равно ее длине, а также тот факт, что полная строка также является суффиксом самой себя (первый суффикс).

Введем понятие *суффиксного дерева*. В общем виде – это древовидный граф с набором вершин и ребер, создающийся из всех суффиксов последовательности S. В графе к каждому ребру приписана «метка», соответствующая некоторой подстроке из S. Двигаясь по ребрам от выделенной вершины суффиксного дерева, называемой корнем, к одному из листьев дерева (вершине, к которой ведет только

одно ребро) и соединяя последовательно «метки» ребер в одну последовательность, в конечном итоге мы получаем один из суффиксов исходной последовательности S. Неформально процесс создания суффиксного дерева можно описать следующим образом:

(i) допишем в конец последовательности S символ, не встречающийся в самой последовательности, например: **\$**, тем самым получим последовательность $S^* = S\$$

(ii) возьмем все суффиксы последовательности S^*, «закрепим» их начала в одной вершине, называемой корнем дерева (Рис. 1), в «концы» суффиксов поместим вершины и соединим их ребрами с корнем дерева. Вершины пронумеруем от 1 до n, ребрам припишем «метки» равные соответствующим суффиксам, тем самым «направив» ребра от корня к листьям.

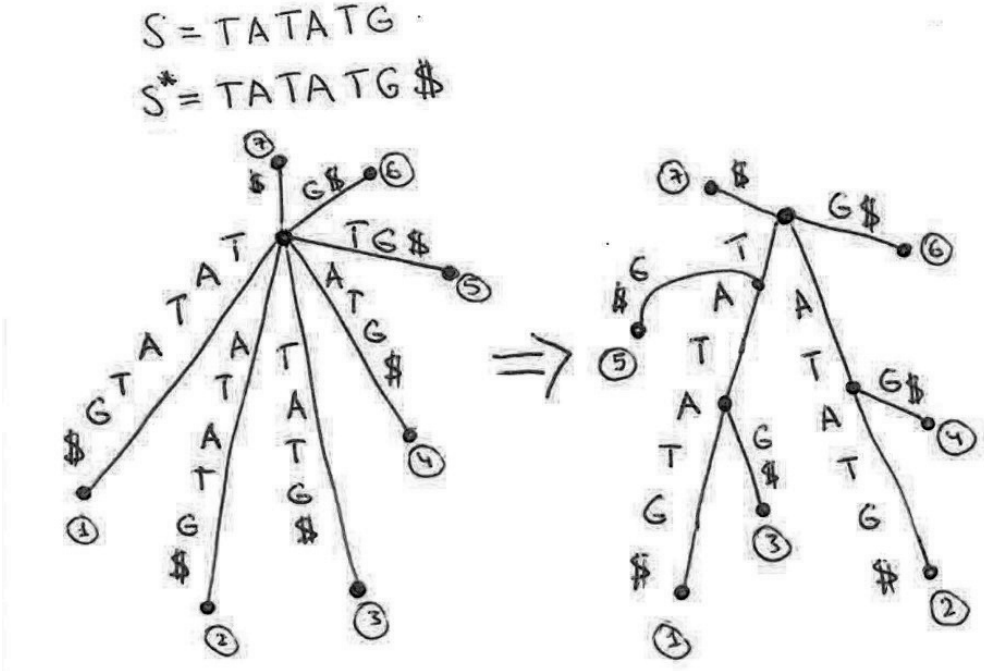

Рис. 1: Создание суффиксного дерева для последовательности S.

(iii) «склеим» ребра с совпадающими началами, объединив их общую часть в одно ребро и «ответвив» от новой внутренней вершины несовпадающие части.

Образованный граф представляет из себя суффиксное дерево для последовательности S^*.

Отдельно стоит отметить, что суффиксное дерево – «направленный» граф, т.е. по ребрам мы можем двигаться только в одном направлении. Из этого следует, что у корневого узла и внутренних узлов есть узлы-«потомки» (узлы, к которым можно прийти, двигаясь по направленным ребрам). А также у всех узлов, кроме корневого (внутренних и листьев) есть узлы-«предки» или «родительские» узлы (те, из которых приходим в эти узлы двигаясь по направленным ребрам).

Перечислим некоторые свойства суффиксного дерева:

(i) Количество листьев дерева равно количеству суффиксов последовательности S^*.

(ii) Каждый узел в дереве, кроме корневого, имеет ровно один родительский узел.

(iii) Двигаясь по ребрам от корня дерева к одному из *листьев* и объединяя метки соответствующих ребер в одну последовательность, последняя будет соответствовать одному из *суффиксов* последовательности S^*.

(iv) Двигаясь по ребрам от корня дерева к одной из *внутренних вершин* дерева и объединяя метки соответствующих ребер в одну последовательность, последняя будет соответствовать некоторой *подстроке* из S^*.

(v) Для любой подстроки S^* можно найти соответсвующий ей путь от корня дерева, причем, путь не обязательно завершается в вершине дерева, а может быть заверешен в середине одного из ребер.

(vi) Метки ребер выходящих из корня или любой внутренней вершины отличаются первыми символами (иначе они были бы склеены в одно ребро). Следовательно из любой вершины не может выходить больше $\sigma + 1$ ребер, где σ – размер алфавита (количество разных символов, для нуклеотидной последовательности $\sigma = 4 + 1 = 5$ (добавляем дополнительный символ \$)).

(vii) Если общее число вершин в дереве равно N, то число ребер: $(N - 1)$.

Предположим, что мы построили для последовательности S суффиксное дерево, как с помощью него определить количество вхождений мотива p в строку S?

3 Алгоритм построения $O(n^2)$

Хранить суффиксное дерево можно различными способами, рассмотрим один из самых простых. Все узлы храним в массиве *nodes*, причем i-ым элементом этого массива, будет в свою очередь набор (список, лист) индексов узлов, которые непосредственно наследуются от i-го. Если i-ый узел – лист (т.е. у него нет потомков), то храним в $nodes[i] = -1$ (можно хранить просто пустой массив). Во втором массиве *edges* в i-ом элементе будет находиться метка ребра, которое ведет к i-му узлу. Нулевое значение (метка к корневому узлу) оставим путсым $edges[0] = ''$.

Пример такой структуры изображен на Рисунке 2.

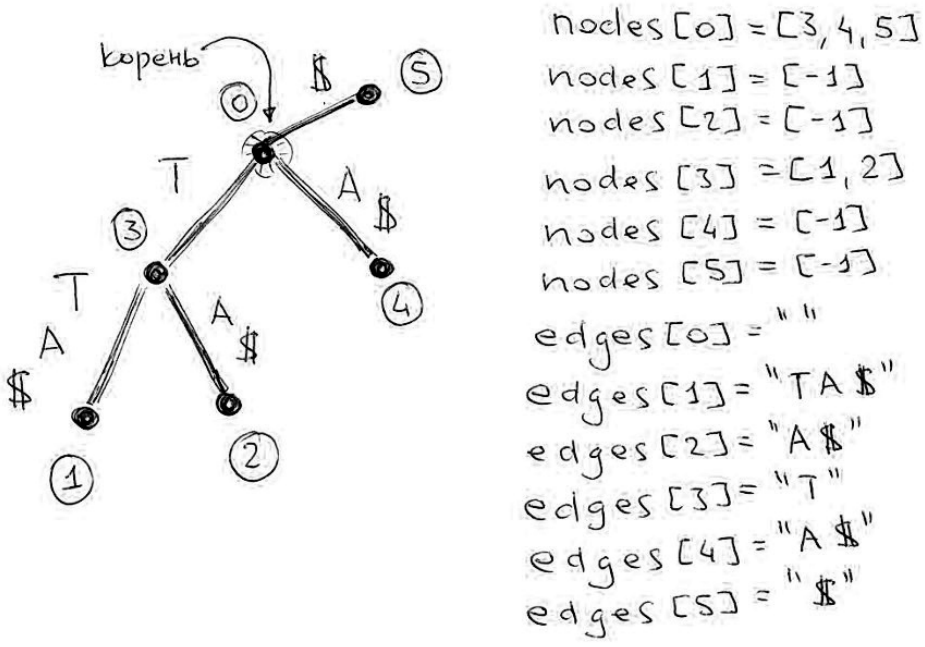

Рис. 2: Пример массивов *nodes* и *edges* для дерева последовательности TTA$.

Ниже приведен псевдокод создания этих двух массивов для строки S (код в стиле Python, индексация массивов и строк с нулевого индекса, x[i:] все элементы с i-го *включительно* до последнего, x[:i] – все элементы с нулевого до i-го *исключая*):

```
1: procedure TreeCreation(S)
2:     S = S + "$"
3:     nodes ← [[1], [−1]]
4:     edges ← ["", S]

5:     for i ← 1, len(S) − 1 do
6:         suf ← S[i :]
7:         j ← 0
8:         cur_node ← 0
9:         isAdded ← False

10:        while not isAdded do
11:            to_node ← −1
12:            for k in nodes[cur_node] do
13:                if suf[j] = edges[k][0] then
14:                    to_node ← k
15:                end if
16:            end for

17:            if to_node = −1 then
18:                nodes.append([−1])                          ▷ Добавляем новый лист
19:                edges.append(suf[j :])                      ▷ Ребро к новому листу
20:                nodes[cur_node].append(len(nodes) − 1)      ▷ Ссылка на новый
21:                isAdded ← True
22:            else
23:                for p ← 1, len(edges[to_node]) − 1 do
24:                    if suf[j + p]! = edges[to_node][p] then
25:                        nodes.append([−1])                  ▷ Добавляем новый лист
26:                        edges.append(suf[j + p :])          ▷ Ребро к новому листу
27:                        nodes.append([to_node, len(nodes) − 1])  ▷ Внутренний узел
28:                        edges.append(edges[to_node][: p])   ▷ Ребрко к узлу

                                                               ▷ Обновить ссылку из текущего на внутренний
29:                        nodes[cur_node].remove(to_node)
30:                        nodes[cur_node].append(len(nodes) − 1)
31:                        edges[to_node] = edges[to_node][p :]   ▷ Обновить ребро
32:                        isAdded ← True
33:                        break
34:                    end if
35:                end for
```

```
                j ← j + len(edges[to_node])
                cur_node = to_node
            end if
        end while
    end for

    return [nodes, edges]
end procedure
```

4 Поиск в глубину

Решаем задачу поиска мотива, как посчитать количество вхождений? Допустим, мы ищем мотив T в строке S, очевидно, что для каждого вхождения T в S будет существовать суффикс строки S, который начинается с T. Т.е. для каждого вхождения есть путь от корня суффиксного дерева к одному из листов, причем начинается этот путь со строки T. Количество вхождений – это количество суффиксов, которые начинаются с T, а это в свою очередь количество листьев в дереве, пути к которым начинаются с T. Т.е. для нахождения мотива T, необходимо найти путь (если он существует, то он будет единственным) от корня дерева, который начинается с T, остановившись либо во внутреннем узле, либо в середине одного из ребер (возможна ситуация, когда мы остановимся и в листе дерева, *подумайте, когда это может быть*). Чтобы определить количество вхождений, необходимо посчитать количество листьев потомков узла, на котором мы остановились или узла, от которого идет ребро, на котором мы остановились.

Задача подсчета количества листьев потомков одного из узлов (i-го узла) решается алгоритмом поиска в глубину, который достаточно просто реализуется через рекурсию:

```
1: procedure LeavesCount(i, nodes)
2:     if nodes[i][0] = −1 then
3:         return 1
4:     end if

5:     lCount ← 0
6:     for k in nodes[i] do
7:         lCount = lCount + LeavesCount(k, nodes)
8:     end for

9:     return lCount
10: end procedure
```

Для определения координат вхождения мотива T в строку S нам достаточно знать индексы (номера) листьев, которые являются потомками внутреннего узла, на котором мы остановились. Они же и будут являться координатами вхождения (1-ый суффикс – соответствует первому символу, 2-ой – второму и т.д.). Определение координат решается небольшой модернизацией вышеописанной рекурсивной функции.

5 Какие еще задачи можно решить с помощью дерева

Поиск повтора, поиск максимальной общей подстроки, нечеткий поиск.

6 Ссылки