Построение суффиксного дерева за линейное время

Лекция N 1 курса
"Алгоритмы для Интернета"

Юрий Лифшиц

ПОМИ РАН - СПбГУ ИТМО

Осень 2006

Esko Ukkonen

План лекции

1. Введение в суффиксные деревья
   - Определение
   - Два применения
   - Наивный кубический алгоритм
2. Квадратичный алгоритм
3. Линейный алгоритм

Часть I

Что такое суффиксное дерево?

Для чего оно может быть полезно?

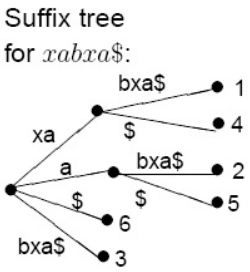
Как построить ST за кубическое время?

Определение суффиксного дерева

Определение: для текста $T = t_1 \dots t_n$ каждое окончание $t_i \dots t_n$ называется **суффиксом**.

Неформально, чтобы построить **суффиксное дерево** (ST), нужно приписать специальный символ \$ к тексту, взять все $n+1$ суффикс, подвесить их за начала и склеить все ветки, идущие по одинаковым буквам. В каждом листе записывается номер суффикса, который в нем заканчивается.

Пример



Проблема: как хранить ST используя линейную память?

Ответ: хранить длинные длинные ребра как ссылки на сегмент текста $T[u..v]$

Поиск подстрок

Задача: дан текст T. Нужно так его "подготовить" за время $O(n)$, чтобы поиск любого шаблона P занимал время $O(|P|)$

Решение с помощью ST:
Построим суффиксное дерево для T
Прочитаем шаблон вдоль дерева от корня (пришли в V)
Прочитаем числа, записанные в листьях-потомках V
Эти числа - все начала вхождений P в T

Сложность: $O(|P| + |Output|)$

Наибольшая общая подстрока

Задача: даны тексты T_1 и T_2. Найти длину их наибольшей общей подстроки.

Самый непосредственный алгоритм?

Решение с помощью ST:
Построим суффиксное дерево для $T_1 T_2$.
Для каждой внутренней вершины выясняем:
Есть ли у нее одновременно "короткий" потомок и "длинный" потомок?
Находим самую нижнюю такую вершину
Ее глубина - ответ для задачи

Сложность: $O(|T_1| + |T_2|)$

On-line подход

Будем строить ST не только для всего текста, но и всех его префиксов:

Строим суффиксное дерев для t_1
Расширяем его до дерева для $t_1 t_2$
.
.
.
Расширяем дерево $t_1 \dots t_{n-1}$ до дерева $t_1 \dots t_n$
Расширяем дерево $t_1 \dots t_n$ до дерева $t_1 \dots t_n\$$

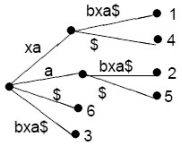
Каждый шаг этого списка называется **фазой**

Все деревья, кроме последнего — **неявные**

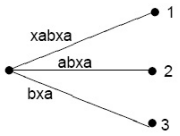
Неявные суффиксные деревья

Неявное суффиксное дерево (IST) - это ST для текста, в котором забыли дописать \$ на конце. Некоторые суффиксы текста в нем заканчиваются не в листьях, и их номер нигде не хранится. Пример:

Suffix tree for $xabxa\$$:



Implicit suffix tree for $xabxa$:

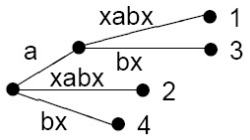


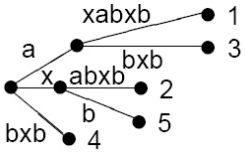
Фаза = последовательность продлений

В фазе i мы перестраиваем IST для $t_1 \dots t_i$ в IST для $t_1 \dots t_i t_{i+1}$:

Для каждого j от 1 до i
Находим в дереве конец суффикса $t_j \dots t_i$
и **продляем** его, дорисовывая (если нужно) букву t_{i+1}

Пример одной фазы





Три типа продлений

Продления суффикса могут быть трех видов:

1. Продление листа
2. Ответвление буквы
   (тут возможно создание новой вершины ST)
3. Пустое правило
   Ничего не дорисовываем, так как буква уже есть

Кубическая оценка времени

Оценка времени нашего алгоритма:

- Всего n фаз
- Фаза i требует i продлений
- Продление j в фазе i требует время $O(i-j)$
- Итого: $O(\sum_{i=1}^{n} \sum_{j=1}^{i} i-j) = O(n^3)$

Часть II

Как улучшить кубический алгоритм до квадратичного?

Сейчас пойдут
кошмарные технические подробности

Удвойте внимание!

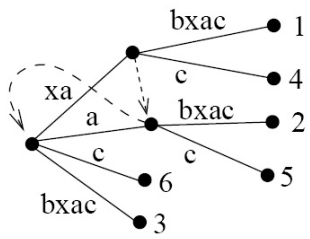
Идея вспомогательных данных

Предположим нужно вычислить массив $X_1, \ldots, X_n$. Иногда полезно поступить так:

- Определить вспомогательный массив $Y_1, \ldots, Y_n$
- Вычислить X_1 и Y_1
- С их помощью вычислить X_2, затем вычислить Y_2
- ...
- С помощью X_{n-1} и Y_{n-1} вычислить X_n

Определние суффиксных стрелок

Для каждой **внутренней** вершины, соответствующей суффиксу $a_1 \ldots a_k$ нарисуем **суффиксную стрелку** (SA) в вершину, соответствующую суффиксу $a_2 \ldots a_k$.



Два естественных вопроса: (1) как их обновлять, (2) как ими пользоваться?

Обновление SA с опозданием

Когда нужно рисовать **новую** суффиксную стрелку?

Ответ: когда мы применили продление по второму правилу (ответвление) к суффиксу $t_j \ldots t_i$ и в дереве образовалась новая внутренняя вершина

Не будем отдельно вычислять SA. Просто перейдем к следующему продлению. Это будет суффикс $t_{j+1} \ldots t_i$. Его конец и есть адрес суффиксной стрелки!

Фаза с прыжками

Будем экономить на нахождении всех "хвостов" внутри одной фазы. Пусть мы закончили работу с "хвостом" j-ого суффикса.

Как нам побыстрее найти хвост $(j+1)$-ого суффикса?

Вверх-Прыжок-Вниз:

1. Перейти от j-ого хвоста $t_j \ldots t_i$ вверх до ближайшей внутренней вершины $t_j \ldots t_k$
2. Прыгнуть по суффиксной стрелке в $t_{j+1} \ldots t_k$
3. Идти вниз, читая текст $t_{k+1} \ldots t_i$

Прыжки: подсчет высоты

Будем следить за глубиной указателя в дереве

- При переходе с хвоста на хвост мы делаем один переход вверх (глубина "-1")
- При переходе по суффиксной стрелке глубина уменьшается не более чем на 1
- Внутри фазы начальная глубина больше конечной

Вывод: глубина **увеличивалась** не более $2i$ раз.
Общая оценка навигации внутри фазы: $4i$ переходов.

Часть III

Как улучшить квадратический алгоритм до линейного?

Анализ операций продления

Напомните три вида продлений

1. Удлинение: продление листа
2. Ответвление буквы
   (тут возможно создание новой вершины ST)
3. Пустое правило
   Ничего не дорисовываем, так как буква уже есть

Наблюдение: как только мы применили пустое правило, дальше в фазе все продления - пустые

Живые ребра

Наблюдение 2: после того как мы применили правило ответвления и создали новый лист, в следующих фазах к этому листу всегда будет применятся правило удлинения.

Способ сэкономить: при создании нового листа кодировать новое ребро как $T[i+1, x]$, где x - указатель на специальную переменную. Тогда все продления уже созданных листов можно произвести одной операцией $x := x + 1$

Модификация алгоритма

Пусть "непустая часть" фазы $i-1$ закончилась на суффиксе j^*. Следовательно, к суффиксам $1,\ldots,j^*$ применялись только правила 1 и 2, и каждый из них заканчивается в своем собственном листе.

Фаза i:

1. Присвоение $x := x+1$ одновременно продляет все суффиксы $1..j^*$
2. Последовательно продляем суффиксы $j^*+1,\ldots,j'$, где j' - первое применения пустого правила
3. Присваиваем $j^* = j'-1$ и переходим к следующей фазе

Линейная оценка

Оценим время работы алгоритма:

- Участки индивидуальных продлений по фазам перекрываются не более чем по одному суффиксу
- Суммарное количество прыжков при продлениях линейно (аналогично оценкам из кваратичного алгоритма)
- Последняя фаза строит уже явное суффиксное дерево текста

Вот мы и получили оценку $O(n)$!

Главные моменты

Сегодня мы узнали:

- Суффиксное дерево: способ **представления** текста
- Применения: поиск подстрок, поиск наибольшей общей подстроки
- Основные идеи алгоритма: on-line построение, вспомогательные суффиксные стрелки, неравномерная оценка времени работы.

Вопросы?

Источники

Страница курса
http://logic.pdmi.ras.ru/~yura/internet.html

Использованные материалы:

Pekka Kilpelainen
Lecture Slides
http://www.cs.uku.fi/ kilpelai/BSA05/lectures/print07.pdf

Esko Ukkonen
On-line construction of suffix trees
http://www.cs.helsinki.fi/u/ukkonen/SuffixT1withFigs.pdf