Второй курс, осенний семестр 2018/19
Практика по алгоритмам #5

Суффиксные массив, дерево

3 октября

Собрано 3 октября 2018 г. в 22:04

Содержание

Суффиксные массив, дерево

1. **Задачи про суффиксный массив**
   a) Общая подстрока двух строк за $\mathcal{O}(|s| + |t|)$.
   b) Реализуйте сжатие `LZSS` за $\mathcal{O}(n \log n)$.

2. **Сортировка строк**
   Дан набор строк суммарной длины n над алфавитом Σ.
   a) Отсортируйте за время $\mathcal{O}(n \log |\Sigma|)$.
   b) Покажите, что $\nexists$ алгоритма сортировки строк за $\mathcal{O}(n)$.
   c) `(*)` Отсортируйте за время $\mathcal{O}(n + |\Sigma|)$.

3. **Строка по суффмассиву**
   a) Построить такую строку s, что её суффиксный массив совпадает с данным, за $\mathcal{O}(n)$.
   b) `(*)` Минимизировать размер алфавита.

4. **Хитрый поиск**
   Дан словарь, постройте структуру, чтобы быстро отвечать на запросы
   a) `get(s)` – число строк в словаре, которые начинаются с s, заканчиваются на $\overleftarrow{s}$.
   b) `get(s,t)` – число строк в словаре, которые начинаются с s, заканчиваются на t, $|s| = |t|$.
   c) `get(s,t)` – число строк в словаре, которые начинаются с s, заканчиваются на t.

5. **Разбиение на словарные слова**
   Дан словарь слов суммарной длины L и текст T.
   a) Слова длины $\leqslant l$. Представить T в виде конкатенации минимального числа словарных слов за $\mathcal{O}(L + l|T|)$. Слова можно использовать более одного раза.
   b) Представить T в виде конкатенации минимального количества **подстрок** словарных слов за $\mathcal{O}(\text{poly}(L) + |T|)$.

6. **Один бор хорошо, а два – лучше!**
   Даны два бора A и B. Найдите для каждой вершины $u \in A$ самую глубокую вершину B, путь до которой равен суффиксу $\text{path}(A)$. $\mathcal{O}(|A| + |B|)$. Размер алфавита $\mathcal{O}(1)$.

7. **Задачи про суффиксное дерево**
   a) Найти количество различных подстрок.
   b) Найти самый длинный рефрен – подстроку s: $\text{count}(s) \cdot |s| \to \max$.
   c) Самая длинная подстрока, входящая в s дважды, причём вхождения не пересекаются.
   d) Общая подстрока двух строк за $\mathcal{O}(|s| + |t|)$.
   e) Общая подстрока k строк за суммарную длину всех строк.

8. **Амортизация в Укконене**
   Приведите пример, когда Укконен при добавлении одного символа спустится вниз $\Omega(n)$ раз.

9. **Бажный Укконен**
   При подсчёте суффиксных ссылок в алгоритме Укконена маленький Петя делает спуск не по рёбрам (пройти всё ребро за шаг), а по символам (пройти один символ за шаг). Приведите пример строки, на которой полученный алгоритм будет работать $\omega(n)$.

10. (*) **XOR** $\to \max$
Дан массив a длины n. Найдите пару a_i, a_j: $a_i \hat{} a_j = \max$.
a) $\mathcal{O}(n \log M)$. $M = \max(a_1, \ldots, a_n)$.
b) $\mathcal{O}(\text{sort} + n)$

11. (*) **XOR** $\geqslant k$
Дан массив a длины n и число k. Длина чисел в массиве $\mathcal{O}(1)$. За время $\mathcal{O}(n)$ посчитайте количество
a) пар индексов таких, что побитовый `XOR` элементов по этим индексам $\geqslant k$.
b) отрезков последовательности, побитовый `XOR` всех чисел из которых $\geqslant k$.

12. (*) **Обратное преобразование Барроуза-Уилера**
Дан последний столбец отсортированного массива циклических сдвигов строки. Также дан номер исходной строки в этом порядке. Восстановить строку.
a) $\mathcal{O}(n^2)$
b) $\mathcal{O}(n)$

Разбор задач практики

1. **Задачи про суффиксный массив**
   a) **Общая подстрока.** Строим суфмас $s\#t\#$ и считаем LCP. Для каждого суффикса s ищем ближайший слева и справа суффикс t, два указателя. Смотрим их LCP – минимум в очереди или Фарах-Колтон-Бендер.
   А можно смотреть только позиции, где рядом суффиксы из s и из t, все равно их LCP не меньше, чем у не соседних.
   b) **LZSS.** Пусть мы уже выписали i символов. Нужно быстро найти $j < i$: $\texttt{LCP}(i, j) = \max$. И жадно из i перейти в $i + \texttt{LCP}(i, j)$. Раньше мы перебирали все j за $\mathcal{O}(i)$. Теперь мы можем посмотреть на ближайший слева/справа в суффмассиве.
   Способ за $\mathcal{O}(n \log n)$. Будем держать позиции в суфмассиве всех $j < i$ в `set<int>` (нужны `insert`, `lower_bound`).
   Способ за $\mathcal{O}(n)$. Каждой позиции суфмассива соответствует начало суффикса $sa[i]$. Нужно найти ближайший справа и слева меньший элемент массива sa.
   Умеем это делать стеком за $\mathcal{O}(n)$.
   Минимум `LCP` можно насчитывать, запоминая минимум между соседями на стеке. А можно написать Фараха-Колтона-Бендера.

2. **Сортировка строк**
   a) $\mathcal{O}(n \log |\Sigma|)$. Строим бор, в вершине не массив, а `map`, иначе нужно выделить $\mathcal{O}(n|\Sigma|)$ памяти. `dfs` по бору, из вершины идем в детей в порядке возрастания символа.
   b) Если большой алфавит и строки длины 1, то это не проще сортировки n чисел.
   c) `(*)` $\mathcal{O}(n + |\Sigma|)$. Отсортируем поразрядно пары $\langle j, s_i[j] \rangle$, т.е. «позиция в строке, символ». Также про каждую пару помним, из какой она строки.
   Теперь можно класть в бор сразу упорядоченные ребра. Берем пару, смотрим, в какую вершину бора пришли по соответствующей строке. Добавляем в нее соответствующее ребро, либо оно там уже есть, тогда это ровно последнее добавленное.

3. **Строка по суффмассиву**
   a) `s[sa[i]] = i`.
   b) Суфмассив – p, $q \circ p = \mathrm{id}$, s – строка.
   Пусть это $p_k = i$ и $p_{k+1} = j$. Когда можно взять $s_i = s_j$? Ровно тогда, когда $q_{i+1} < q_{j+1}$.
   Алгоритм: $s_{p_0} = 0, s_{p_{i+1}} = s_{p_i} + (q_{p_i+1} > q_{p_{i+1}+1})$.

4. **Хитрый поиск**
   a) `get(s)` – бор из $\texttt{LCP}(w, \overleftarrow{w})$. Спускаемся по s, смотрим число конечных вершин в поддереве.
   b) `get(s,t)` – бор из строк $w_1 w_n w_2 w_{n-1} w_3 w_{n-2} \dots w_n w_1$. Спускаемся попеременно по символам s и $\overleftarrow{t}$. Смотрим число конечных вершин в поддереве.
   c) `get(s,t)` – бор прямых строк, отдельно бор обратных строк. Конечные вершины помечены номерами строк.
   Спускаемся в первом боре по s, во втором по $\overleftarrow{t}$, получили два поддерева. Ответ – пересечение множеств пометок в этих поддеревьях. В каждом дереве все пометки различны, умеем решать такую задачу 2D-запросом.

5. **Разбиение на словарные слова**
   a) Строим бор из словаря. `f[i]` – минимальная стоимость выписать префикс длины `i`.
   Динамика вперёд: спускаемся по бору суффиксом `s[i:]`, если очередная вершина бора конечная, то `relax(f[i + dep], f[i] + 1)`.
   Максимальная глубина бора $l \Rightarrow$ время $\mathcal{O}(L + l|T|)$.
   b) Строим бор суффиксов словарных слов за $\mathcal{O}(L^2)$, либо суфдерево для $s_1\#s_2\#\ldots\#s_n$ за $\mathcal{O}(L)$.
   Жадно разбиваем текст за $\mathcal{O}(|T|)$: пока текст не пуст, отрезаем самый длинный префикс текста, по которому можем спуститься в боре.

   Способ #2: Ахо-Корасиком. Он находит самую глубокую вершину бора, в которую можно прийти суффиксом текста.
   Считаем динамику, держим очередь с минимумом для окна, покрытого путем до текущей вершины бора.

6. **Один бор хорошо, а два – лучше!**
   Замкнем бор B до полного автомата за $\mathcal{O}(|B|)$. Для каждой вершины $u \in A$ ответ пересчитываем через предка: `ans[a[v][c]] = b[ans[v]][c]`.

7. **Задачи про суффиксное дерево**
   a) **Число подстрок.** В боре это число вершин. В сжатом боре это сумма длин ребер.
   b) **Рефрен.** count(s) – число суффиксов, кончающихся в поддереве, если спуститься по s. Заметим, что достаточно смотреть на строки, кончающиеся в вершинах.
   c) **Дважды входящая.** Ищем v: $\min(R[v] - L[v], \text{depth}[v]) \to \max$. Ответ может быть посреди ребра.
   d) **Общая подстрока.** Суфдерево для $s\#t\#$. Самая глубокая вершина, в поддереве которой есть и суффикс s, и суффикс t. Чтобы это определить, храним min и max суффиксы в поддереве.
   e) **Общая подстрока k строк.** Для каждой вершины считаем битмаску номеров строк, суффиксы которых есть в ее поддереве. Ищем самую глубокую, в поддереве которой есть все k.
   Еще можно решать для больших k, для этого надо за $\mathcal{O}(n)$ посчитать для каждой вершины количество различных чисел в поддереве.

8. **Амортизация в Укконене**
   $aaa\ldots ab$. Добавление последнего символа создаст n новых листьев.

9. **Бажный Укконен**
   $aaa\ldots ab$. При добавлении b для создания листа суффикса i нужно спуститься из корня по i символам.

10. **(*) XOR** $\to \max$
    a) Числа – битовые строки длины **ровно** $\log M$. Строим на них бор, старшие биты ближе к корню.
    Переберём a_i. Будем спускаться по бору, стараясь получить a_j с максимальным $a_i \hat{} a_j$: если есть ребро по биту, не равному соответствующему биту в a_i, спускаемся по нему.
    b) $\mathcal{O}(\text{sort} + n)$. Сначала заметим, что можно не перебирать a_i, а найти оба элемента параллельным спуском по бору: идем по разным битам, если можем.
    Далее, размер сжатого бора $\mathcal{O}(n)$. Имея сжатый бор, можно за $\mathcal{O}(n)$ найти ответ.

Сортируем массив и считаем LCP соседних. LCP можно посчитать битовыми операциями и предпосчитанными логарифмами степеней двоек.
По сортированному массиву и LCP умеем строить сжатый бор за $\mathcal{O}(n)$.

11. (*) **XOR $\geqslant k$**
    a) Строим бор на числах массива как битовых строках равной длины.
    Перебираем a_i, считаем число пар с ним. Для этого спускаемся по бору туда, где `XOR` даст результат, как в k. Если спустились по 0, прибавляем к ответу число листьев поддерева по 1.
    b) `XOR` подотрезка $[i, j]$ – это pref_i ^ pref_{j+1}, где pref_i – `XOR` на префиксе. Решаем предыдущий пункт для pref.

12. (*) **Обратное преобразование Барроуза-Уилера**
    a) $\mathcal{O}(n^2)$. Восстановим всю таблицу.
    Пусть уже знаем первые k столбцов (изначально 0). Тогда знаем все подстроки из $k+1$ буквы, входящие в зацикленную строку, так как есть последний столбец.
    Эти подстроки отсортированы по суффиксам длины k, можно досортировать стабильным подсчетом за $\mathcal{O}(n)$. Получили $k+1$ первых столбцов.
    b) $\mathcal{O}(n)$. Конспект ИТМО
    Википедия
    Задачу можно сдать на тимусе.
    Сортируем последний столбец подсчётом (стабильным), получим первый.
    `c[i]` – символы первого столбца, `p[i]` – их позиции в последнем.
    Алгоритм:

```
for (int i = start, j = 0; j < n; j++) {
  out << c[i]; // вывести очередной символ строки
  i = p[i]; // перейти к следующему
}
```

Домашнее задание

3.1. Обязательная часть

1. **(2.5) Общая подстрока k строк**
   Предложите алгоритм поиска max общей подстроки k строк за их суммарную длину. Можно пользоваться только суфмассивом.

2. **(2) T9**
   Дан набор n строк s_i. Для каждой s_i найдите min по длине префикс, который не является префиксом других строк. Время $\mathcal{O}(\sum |s_i|)$.
   (+1), если у вас получится $\mathcal{O}(n)$ доппамяти.

3. **(3) Количество строк**
   Дан словарь и число n. Посчитайте количество строк длины n над алфавитом $\{a, b\}$, которые не содержат ни одного словарного слова, как подстроку. Время $\mathcal{O}(nL)$, где L – суммарная длина слов в словаре.
   (+1) допбалл за $\mathcal{O}(L)$ памяти.

4. **(3) Ключевые подстроки**
   Дан набор строк s_i. Для каждой s_i найдите min по длине подстроку, которая не встречается в других.

5. **(3) Уникальные суффиксы**
   Два запроса:
   a) `addLetter(c)` – дописать в конец строки символ c.
   b) `isUnique(len)` – является ли суффикс длины len уникальной подстрокой.

6. **(3) k-я общая подстрока**
   Найти k-ю лексикографически общую подстроку s и t за $\mathcal{O}(|s| + |t|)$.

3.2. Дополнительная часть

1. **(3) Подпалиндромы на отрезках**
   Дана строка s и q запросов «самый длинный подпалиндром на отрезке строки $[l..r]$».
   Решить за $\mathcal{O}((|s| + q)\texttt{Poly}(\log))$.

2. **(3) Дерево палиндромов**
   Дана строка длины n. Посчитайте для каждого i максимальный палиндром-суффикс `s[0:i)`.
   Докажите, что различных подстрок-палиндромов не более n. $\mathcal{O}(n)$.

3. **(4) Разбить строку на 3 палиндрома**
   Дана строка s, нужно за $\mathcal{O}(|s|)$ представить её в виде конкатенации 3 палиндромов.

4. **(3) Поиск по отрезку словаря**
   Дан словарь $s_1, s_2, \ldots, s_n$. Отвечать в offline на запросы `get(t, l, r)` – сколько строк из $\{s_l, \ldots, s_r\}$ входят в текст t как подстроки. Время работы – линия от размера входа на полилог.