Алгоритмы
Домашнее задание №2
Стрелец Владимир

Задача 2.

Докажем, что $\pi_{i+1} \le \pi_i + 1$ для всех допустимых i. Пусть это не верно, то есть существует i, такое что $\pi_{i+1} \ge \pi_i + 2$.

Тогда из этого неравенства получим, что

s[i-π_i-1...i+1] = s[1...π_i+2], но по определению π_i:

s[i-π_i-1...i] $\neq$ s[1...π_i+1], что уже невозможно.

Получено противоречие, откуда следует верность неравенства $\pi_{i+1} \le \pi_i + 1$.

Теперь строим следующую строку s над алфавитом {1, … , n}:

s[0] = 1.

Для всех 0 < i < n:

если π_i > 0, то это значит, что (по определению значения π_i) подстроки s[1...π_i] и s[i-π_i+1...i] равны. А так как s[π_i] уже построен, то s[i] = s[π_i];

если π_i = 0, то мы можем выбрать любой ещё не использованный символ для s[i] и он не совпадёт ни с каким прошлым (в частности, ни с s[0], ни с s[π_{i-1}+1] для i > 1).

Теперь покажем, что построенная строка имеет заданную π-последовательность. Для этого можно, например, построить верную π-последовательность (назовём её τ-последовательностью) для этой строки и показать, что построенная последовательность совпадает с заданной. Итак, рассмотрим два варианта:

1) τ_i < π_i. Это невозможно, так как по построению строки s мы строили все элементы так, чтобы было верно: s[i-π_i+1...i] = s[1...π_i]. Поэтому $\tau_i \ge \pi_i$.
2) τ_i > π_i. Тогда по построению τ-последовательности
   s[i-τ_i+1...i] = s[1...τ_i]. Но тогда подстрока s[i-τ_i+1-π_i...i] должна быть равна s[1...$\tau_i - \pi_i$]. Но… (пока не доказано).

Задача 1.

Предположим, что мы умеем по настоящей π-последовательности строить строку (это было выполнено в задании 2). Также предположим, что нам задана настоящая π-последовательность (то есть такая, для которой существует хотя бы одна строка, порождающая её). Попробуем построить строку s алгоритмом:

s[0] = 1.
Для всех 0 < i < n:

если $\pi_i > 0$, то это значит, что (по определению значения π_i) подстроки s[1...π_i] и s[i-π_i+1...i] равны. А так как s[π_i] уже построен, то s[i] = s[π_i];

если $\pi_i = 0$, то мы можем выбрать любой ещё не использованный символ для s[i] и он не совпадёт ни с каким прошлым (в частности, ни с s[0], ни с s[π_{i-1}+1] для i > 1).

Если во время построения был произведён выход за "известные" границы строки, то это будет значить, что данная последовательность не является π-последовательностью. Если такого не случилось, то после построения строки найдём её настоящую π-последовательность и сравним с данной. Если совпали, то нам была дана π-последовательность. Иначе -- нет.

Алгоритм работает за O(|π|).

Задача 3.

Задача 4.

За O(|S|) найдём π-последовательность для строки. Далее, покажем, что бордер бордера -- второй наибольший собственный бордер строки. Пусть имеется π_n. Тогда s[1...π_n] = s[n-π_n+1...n].
Бордер бордера -- s[1...π_{π_n}] = s[π_n-π_{π_n}+1...π_n], откуда следует, что s[n-π_{π_n}+1...n] = s[n-π_n+1...n-π_n+π_{π_n}]. То есть бордер бордера -- наибольший бордер подстроки s[1...π_n]. А значит, он будет наибольшим бордером подстроки s[n-π_n+1...n], то есть, по совместительству, вторым по длине бордером.

Итак, ответом на задачу является бордер бордера и его длина равна π_{π_n}.

А задача решается за O(|S|), так как для взятия бордера от бордера потребовалось константное число операций.

Задача 5.

Сначала найдём π-последовательность для строки.
Как было показано в задаче 4, для нахождения следующего по длине бордера необходимо взять бордер от бордера. Таким образом, чтобы найти все бордеры, нужно брать бордер от бордера до тех пор, пока его длина не станет равна 0.
А во время нахождения бордеров будем подсчитывать их количество.

Так как число всех бордеров строки не превосходит S-1 (все бордеры -- суффиксы), то алгоритм будет работать за O(|S|).

Задача 6.

Рассмотрим строку reverse(s). Заметим, что все суффиксы строки s -- это префиксы строки reverse(s). А строки вида s[i...i+a_i-1] -- это обращённые строки вида reverse(s)[n-i-a_i+2...n-i+1] Поэтому искомое значение a_i строки s равно значению префикс-функции строки reverse(s) в позиции (n-i+1). Поэтому для нахождения массива a_i i=1, … , n мы перевернём строку s, найдём для неё π-последовательность и перевернём этот массив. Всё это будет выполнено за O(|S|) (будем переворачивать массив и строку, например, через стек).

Задача 7.

Задача 8.

Пусть второй игрок будет составлять строку S$T ($ -- символ не из алфавита), то есть считать, что изначально имеется строка S$T и к ней в конец приписываются символы первым игроком. Теперь второй игрок будет пытаться подсчитывать z-последовательность для строки S$T, начиная с S$T[n + 1] (то есть с символов строки T). Каждое вхождение строки S в строку T будет давать значение z-функции, равное n.

Так как при подсчёте z-последовательности для строки имеют место лишь "удачные" и "неудачные" сравнения, одни из которых "сдвигают границу рассмотренных символов строки", а вторые -- обрывают подсчёт z-функции, а также все сравнения порождают максимум лишь одно дополнительное сравнение то число сравнений заключено в пределах $\Theta(2|T|) - \Theta(4|T|)$.

Таким образом, если n выбрать достаточно большим, то будут нивелированы случаи с возможным "досрочным" обгоном первым игроком второго в том, что первый игрок предоставил слишком много информацию, которую второй игрок не может успеть обработать из-за ограничения в всего 5 сравнений за ход. Но, скорее всего, второй игрок такой стратегией в любом случае сможет победить первого.

Задача 9.

Задача 10.

Пока не решено. Но вроде бы есть решение за O((N+M) log N) = O(T log T).
Я допишу и пришлю на почту. Чуть позже. Постараюсь в течение получаса-часа.

Задача 11.

Докажем выполнимость пункта (e). И тогда пункты (a), (b), (c), (d) будут автоматически опровергнуты.
Рассмотрим любую неубывающую подпоследовательность подряд идущих значений π-последовательности, начинающуюся единицы. Пусть эта подпоследовательность "простирается" по индексам от i до j, то есть для s[i...j] выполнено $\pi_i \le \pi_{i+1} \le \ldots \le \pi_j$. Этой подпоследовательности соответствуют равенства подстрок

s[1... π_i] = s[i- π_i +1...i],
s[1... π_{i+1}] = s[i+1- π_{i+1} +1...i+1],
s[1... π_{i+2}] = s[i+2- π_{i+2} +1...i+2],
……
s[1... π_j] = s[j- π_j +1...j],

Если рассмотреть аналогичную подпоследовательность для z-последовательности, то мы получим, что……
$\Sigma_{k=i}^{j} z_k \le \Sigma_{k=i}^{j} \pi_k$ (Не получилось доказать корректно, поэтому просто написал :C).

В любом случае эти 2 примера как минимум опровергают все пункты (a)-(d).

1) S = "*abacaba*", π = "0010123", z = "0010301". Сумма z меньше суммы π.

2) S = “*abacaba*” , π = “01” , z = "01" . Сумма z равна сумме π.

Теперь опровергнем (f) ((g) может быть опровергнуто через верность (e), но, увы, его доказать пока не получилось).

(f) Рассмотрим произвольную строку s длины большей единицы.
Если у неё s[2] $\neq$ s[1], то $\pi_2 = z_2 = 0$.
Если же у неё s[2] = s[1], то $\pi_2 = 1$, а $z_2 \geq 0$.
В обоих случаях (f) не выполнено, а рассмотрены были все строки s.

Задача 12.

Нам заданы координаты x_i гвоздиков. Отсортируем из (за O(n log n)).

Нет смысла соединять гвозди не последовательно, то есть гвоздь i с гвоздём i+r, $r \geq 2$, потому что этой ниткой мы соединяем лишь гвозди с номерами i и i+r. Намного эффективнее было бы соединить гвозди с номерами от i до i+r последовательно, и длина использованной нитки была бы такой же.

Заведём массив min_length размера n. В нём на i-й позиции будет храниться минимальная необходимая длина нитки для соединения гвоздей с номерами от 1 до i включительно. Таким образом, в позиции n будет записан ответ к задаче.

Рассмотрим первый гвоздик. он обязательно будет соединён со вторым.
Рассмотрим третий гвоздь. Он может быть соединён либо со вторым, либо с четвёртым. Если соединить третий и четвёртый гвозди, то к гвоздям от 1 до 4 уже будут присоединены по одной нитке, поэтому соединять третий гвоздь одновременно со вторым и с четвёртым не имеет смысла.
То есть,

min_length[1] = 0,
min_length[2] = $x_2 - x_1$,
min_length[3] = $x_3 - x_1$,
min_length[4] = $x_2 - x_1 + x_4 - x_3$.

Далее, заметим, что вообще любые 4 гвоздя, стоящих последовательно, соединять нитками неэффективно. Также не может быть двух пропусков подряд, ведь каждый гвоздь должен быть соединён. Но иногда лучше

соединить три подряд идущих гвоздя, чем соединять гвозди только парами. Поэтому для i > 4 выполнено

```
min_length[i] = min(
        min_length[i - 2] + (x_i − x_{i−1}); min_length[i - 3] + (x_i − x_{i−2}).
```

Таким образом, за O(n) будет заполнен массив min_length и ответ будет содержаться в min_length[n].
Но итоговая сложность -- O(n log n), так как нам пришлось сортировать исходные данные.

Задача 13.

Отсортируем данные пары по первой координате a_i по убыванию (за O(n log n)). Далее составим массив разностей $c_i = a_i - a_{i+1}$ значений первых координат, причём последний элемент в этом массиве будет просто равен a_n (иногда в этом массиве могут попадаться нули).

Итак, пусть у нас есть 3 массива: c[i] разностей первых координат, a[i] значений первых координат и b[i] значений вторых координат для заданных пар.

Для нахождения искомого значения воспользуемся следующим алгоритмом:

```
max_b = 0
count = 0

for i from 1 to n:
        if c[i] > max_b:
                max_b = c[i]
        count += max_b * b[i]
```

Это работает потому что эта задача сводится к нахождению площади многоугольника, образованного наложением прямоугольников с координатами вершин (0, 0), (a_i, 0), (a_i, b_i), (0, b_i). Ведь, отсортировав по первой координате и составив массив разностей, мы будем каждый раз подсчитывать только ещё не подсчитанные "полоски". Причём будем подсчитывать каждую "полоску" целиком, так как мы обновляем максимальную высоту (или ширину, тут уж как смотреть) "полосок".

Итак, основная часть алгоритма работает за O(n), а сортировка -- за O(n log n). Итоговая сложность -- O(n log n).

Задача 14.

Ссылка на удачную посылку:

https://contest.yandex.ru/contest/1080/run-report/584746/

Задача 15.

Ссылка на удачную посылку:

https://contest.yandex.ru/contest/1080/run-report/585113/