Квалификационный тур четвертьфинала 2016. Разбор задач

После конца соревнования, если вы захотите, решить какие-либо задачи, несданные на соревновании, то можете перейти по данной ссылке (http://codeforces.com/blog/entry/47622) и получить алгоритм, как это сделать.

a. Контур-кампус

В этом тексте 23 знака препинания, 15 латинских букв, 8 цифр.
Ответ: 15 + 8 - 23 = 0

b. Сильный программист

Общее время на подходы: N * T
Общее время на перерывы (N - 1) * P, т.к. перерывов меньше на единицу, чем подходов
Ответ: сумма этих двух величин

c. Волчок

Есть всего 24 перестановки из четырёх цветов. Для каждой из них вы заранее можете глазами на картинке найти ответ и записать отдельным if'ом в программе

d. Судоку

Мы специально выбрали судоку попроще. Вы могли написать программу или заранее решить судоку на бумаге. Если вы не решили эту задачу и не знаете, как это сделать, вбейте в поисковик "как решать судоку" и найдёте множество различных алгоритмов.

e. Конфеты

Пустим цикл по i от 1 до n. На каждом шаге будем находить НОД(i, n - i) за линейное время (простым перебором всех делителей). Такой алгоритм даст решение задачи за O(n * n). Однако если остановить программу после первого найденного подходящего i, то алгоритм будет работать за O(log n * log n). Дело в том, что если некоторое простое i не подошло под ответ, то n делится на i. Поэтому когда вы дойдёте до 30го простого, ваше n должно будет делится уже на произведение всех простых от 1 до 30, что больше 1e9, а значит невозможно.

f. Влад в армии

Проведём n + 2 вертикальные прямые с x от 0 до n + 1. Они разобьют данные схемы на n + 1 полосу. Решим задачу для каждой полосы независимо, а после просуммируем. В каждой полосе будет ровно по одному отрезку от каждой схемы. Если они не пересекаются, то нам нужно найти площадь четырёхугольника (или треугольника в случае крайней левой или крайней правой полосы). Если они пересекаются, то мы получаем два треугольника с общей вершиной и параллельными сторонами, не содержащими эту вершину. Такие треугольники подобны и из подобия можно найти их высоты, а значит и площади.

g. Открытый кубок

Будем решать задачу n раз: отдельно для каждого нового запроса. Переберём все подстроки первой строки за время O(L * L), для каждой из них проверим подходит ли она. Для этого: пробежимся по всем остальным строкам и найдём, входит ли она в них. Общая сложность такого решения: O(n * n * L^4), что не превосходит нескольких миллионов действий и требует меньше 0.1 секунды

h. Пасьянс

Построим ориентированный граф, вершинами которого будут все карты из колоды. Из вершины A в вершину B проведём ребро, если карту, соответствующую A, нужно переложить в стопку раньше, чем карту, соответствующую вершине B (каждую из карт в свою стопку). Соответственно, ребра между вершинами возникают в двух ситуациях: карты одной масти, и ребро ведет из карты с меньшим достоинством в карту с большим; карта лежит в ряду правее другой карты. Заметим, что топологическая сортировка вершин данного графа, которую можно получить с помощью поиска в глубину, непосредственно задает требуемый порядок карт. А если в графе будут циклы, то разложить пасьянс не получится. Кроме того, можно заметить, что для целей топологической сортировки достаточно оставить только ребра между соседними по достоинству и соседними в рядах картами. В свете последнего замечания в графе будет O(nm) ребер, и ответ будет найден за O(nm).

i. Сумма степеней

Давайте рассматривать разбиения на суммы, в которых все слагаемые упорядочены по невозрастанию. Тогда мы можем действовать следующим образом: в каждый момент у нас есть какая-то текущая степень, сумма которую осталось добить, и мы знаем сколько слагаемых осталось поставить. Мы можем либо взять ещё одно слагаемое текущей степени, либо перейти на степень на одну меньшую.
Наивный способ оформить данное решение в виде динамического программирования:
dp(sum, deg, count) = dp(sum, deg - 1, count) + dp(sum - k^deg, deg, count - 1)
И база: dp(0, 0, 0) = 1.
Ответ лежит в dp(n, log_k(n) + 1, l).
Проблема такой наивной динамики: sum до 10^18. Но это не проблема на самом деле: мы знаем что sum % k^deg = n % k^deg, так как sum получена из n вычитанием степеней k больше либо равных deg. А также мы знаем что если sum / k^deg больше count, то ответ для такой dp равен 0. Это потому что нам надо использовать count слагаемых каждое из которых не больше k^deg и набрать sum, и при этом k^deg * count < sum. Таким образом вместо sum можно хранить sum / k^deg, и этот параметр не превосходит l.
Тогда модифицированная dp выглядит так:
dp(s, deg, count) = dp(s * k + n_k[deg - 1], deg - 1, count) + dp(sum - 1, deg, count - 1),
Где n_k[deg - 1] = n / k^(deg - 2) % k. (то есть (deg - 1)-ый разряд в k-ичной записи числа n).
База остаётся прежней: dp(0, 0, 0) = 1.
Ответ лежит в dp(0, log_k(n) + 1, l).
Состояний в dp - l * log_k (n) * l, каждое вычисляется за O(1).
Таким образом общая сложность - O(l^2 log(n)).

j. Динамическая сложность строки

Давайте за 2^n переберём все бинарные строки длины n. После этого за O(n * n) посчитаем их динамическую сложность. Это делается следующим образом:
1)Для каждого префикса посчитаем его период за линейное время с помощью префикс-функции или z-функции. Это суммарно делается за O(n * n)
2)Теперь переберём все префиксы строки. Для каждого префикса будем за время O(n) находить его сложность: все грани строки можно перечислить за линейное время с помощью алгоритма префикс-функции, после чего взять его заранее посчитанный период и добавить в словарь. Сложностью данной строки будет размер словаря. Такой словарь можно реализовать за константное время, храня его в одном integer’е и добавлять элементы, добавляя бит в нужную позицию.

Общая сложность решения: O(n * n * 2^n). Оно отработает несколько минут, после чего вы можете создать новое решение, в котором занести ответы на все 25 тестов в массив констант.

Этого уже достаточно, чтоб сдать задачу, однако, у жюри также есть решения за O(n * 2^n) и O(2^n), но мы их не будем приводить здесь, т.к. их описание требует гораздо больший объём текста.

k. Открытый кубок - 2

Построим общее суффиксное дерево для всех строк из входа, при этом при построении отметим в каждой вершине, каким строкам она принадлежала. Далее используем классический приём объединения множеств вдоль дерева -- обходим суффиксное дерево в глубину и поддерживаем множество, содержащее все номера строк, которые могут быть встречены в поддереве, сливая эти множества снизу. При этом при объединении множеств будем добавлять элементы меньшего множества к большему, что в сумме отработает за nlog^2n.

Далее, имея такое множество, мы должны узнать наименьший номер строки из второго типа, в котором встречается ребро (logn) и наименьший номер строки из первого типа, в котором оно не встречается. Второй запрос также может быть сделан за logn, но в авторском решении он реализован с помощью бинарного поиска и сравнения номера элемента с количеством элементов, меньших него за log^2 n.

Итоговая асимптотика: O(n log^2 n).