Разбор задачи «Missing Letters»

Есть множество способов решить эту задачу, самый простой способ — жадный алгоритм.

Заметим, что заменять знаки вопроса имеет смысл только на буквы из инициалов. Пройдемся по строке слева направо. Если встречаем знак вопроса, то если перед ним стоит первая буква инициалов, ставим вторую, если нет — то первую.

Почему это верно? Пусть мы ошиблись и в оптимальном ответе вместо второй буквы нужно было поставить первую, тогда мы испортили одну хорошую подстроку, но зато добавили другую, поэтому суммарное число хороших подстрок осталось таким же.

Разбор задачи «Julia and Flower Beds»

Оказывается, что в оптимальном решении разность всегда равна 0 или 1 (0, если общее количество цветков чётное, и 1 иначе). Существуют несколько жадных стратегий, позволяющих построить оптимальную рассадку. Разберем одну из них.

Будем идти с конца, перебирая $i = n,\ n-1,\ \ldots,\ 1$, и поддерживать количество цветков на каждой из клумб (изначально на каждой клумбе 0 цветков). Все цветы очередного вида i посадим на ту клумбу, где меньше цветов. Утверждается, что после итерации на вида i разность цветков на клумбах не превосходит i, и соответственно, в конце разность не превосходит 1.

Это утверждение можно доказать методом математической индукции. Базовый случай: после итерации $i = n$ разность равна $a_n \leqslant n$. Если разность меньше или равна i после итерации $i > 1$, то на следующей итерации к меньшему прибавится положительное число, не превосходящее $i - 1$, и разность не может стать больше $i - 1$.

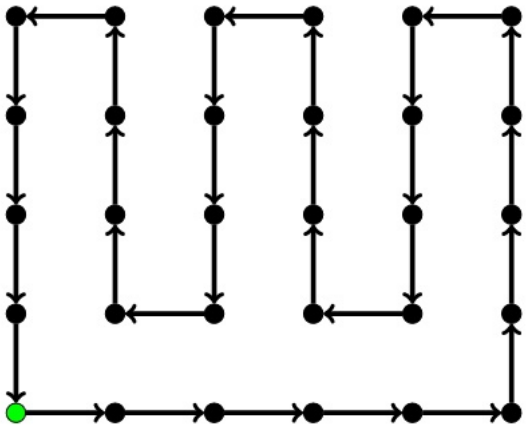
Разбор задачи «Divisor Circle»

Пусть $n = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$, где p_i — различные простые числа. Тогда любой делитель $d = p_1^{\beta_1} p_2^{\beta_2} \ldots p_k^{\beta_k}$ можно представить как точку $(\beta_1, \beta_2, \ldots, \beta_k)$ в k-мерном пространстве. И по условию нам нужно обойти по циклу как можно больше точек так, чтобы любые две соседние были на расстоянии 1 друг от друга.

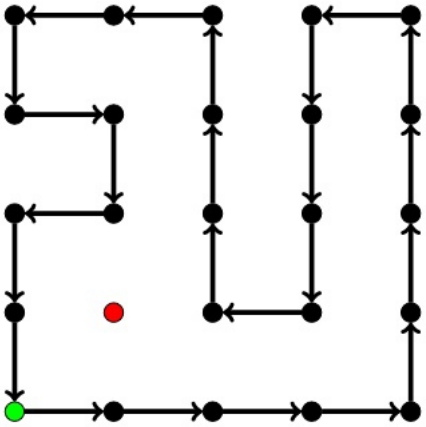
Если $n = p^{\alpha}$, то у нас одномерный случай, и больше двух делителей мы никак не сможем поставить.

В ином случае, у нас есть хотя бы два измерения. Утверждается, что всегда можно построить цикл из всех точек или без одной. Понятно, что если у нас нечетное суммарное количество точек, то мы не сможем обойти все: на каждом шаге четность суммы координат изменяется.

Для простоты сначала рассмотрим двумерный случай. Тогда наши точки образуют прямоугольник. Можно сделать обход, например, следующим образом:



Сначала мы обошли прямоугольник вдоль первого измерения, после чего вернулись обратно «змейкой». Причем в случае с нечетным количеством точек в самом конце мы сделали еще одну «змейку».



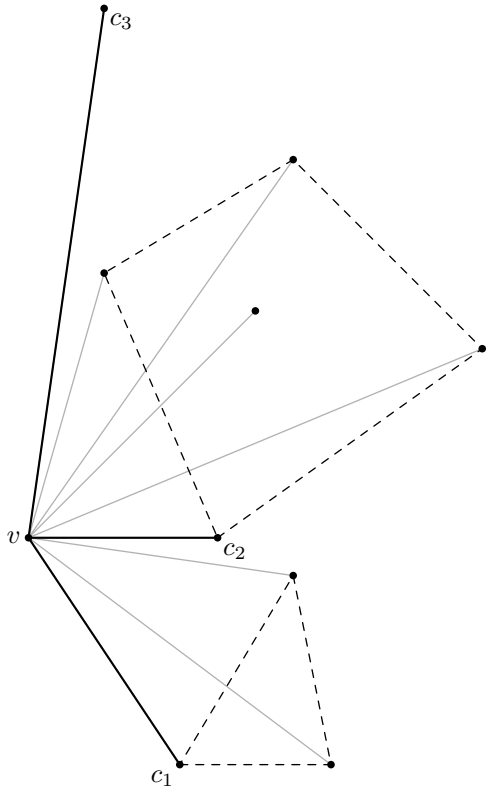
На самом деле также можно сделать и в общем случае. Сначала сделаем обход (не обязательно циклический) всех точек без последнего измерения. после чего сделаем этот же обход в обратном порядке, добавив такую же «змейку» в последнем измерении. Если последнее измерение будет четным, то мы посетим все точки, если нечетным - все точки, кроме одной.

Разбор задачи «Planar Tree»

Подвесим дерево за какую-нибудь вершину (корень). Поддеревом вершины v назовем множество таких вершин u, что v лежит на кратчайшем пути от u до корня. Обозначим поддерево v как st_v.

Научимся рекурсивно решать следующую задачу. Рассмотрим поддерево вершины v, включая саму вершину v. Функция принимает вершину v, а также P — множество точек на плоскости ($|st_v| = |P|$). И должна расставить вершины из st_v в точках из P, при этом вершина v должна быть расположена в точке $p \in P$ (гарантируется, что p лежит на выпуклой оболочке P). Расстановка должна быть правильная, т.е. никакие два ребра не должны пересекаться, кроме как в концах.

Удалим p из P, и отсортируем оставшиеся точки P по углу относительно p. Обозначим детей v в дереве как c_1, c_2, $\ldots$, c_k. Для поддерева вершины c_1 используем точки $P[1 \ldots |st_{c_1}|]$, при этом вершину c_1 расположим в точке $P[1]$. Для поддерева c_2 используем точки $P[|st_{c_1}|+1 \ldots |st_{c_1}|+|st_{c_2}|]$, при этом вершину c_2 расположим в точке $P[|st_{c_1}|+1]$. И так далее (см. илл.).



Заметим, что ребра (v, c_1), (v, c_2), $\ldots$, (v, c_k) не пересекаются друг с другом, и не могут пересекаться ни с какими ребрами из поддеревьев c_1, c_2, $\ldots$, c_k. Следовательно, алгоритм корректен.

Изначально нужно вызвать рекурсивную функцию от корня дерева, всего множества точек, и расположить корень в любой точке, которая лежит на выпуклой оболочке. Например, в точке с минимальной парой (x, y).

Время работы $O(n^2 \cdot \log n)$.

Разбор задачи «Redundant Binary Representations»

Для начала решим прямую задачу и выведем формулу для подсчета $a(n)$.

Пусть n нечетно, $n = 2k+1$. Тогда в каждой избыточной двоичной записи числа n присутствует ровно одна единица (например, это можно видеть для $n = 21$ в условии). Если убрать эту единицу и поделить все остальные слагаемые на 2, мы получим представление числа k. Таким образом, $a(2k+1) = a(k)$.

Иначе, пусть n четно и $n = 2k+2$. Тогда число единиц в избыточном двоичном представлении n четно. Убрав эти 0 или 2 единицы и поделив остальные слагаемые на два, мы получим представление числа k или $k+1$. Таким образом, $a(2k+2) = a(k) + a(k+1)$.

Рассмотрим пару соседних значений $x = a(n)$ и $y = a(n+1)$. В паре чисел $(n, n+1)$ ровно одно число четно и одно число нечетно.

Пусть $n = 2k+1$. Тогда $x = a(2k+1) = a(k)$, а $y = a(2k+2) = a(k)+a(k+1)$. Поскольку $a(k) > 0$, то $x < y$. Аналогично, если $n = 2k+2$, то $x = a(2k+2) = a(k)+a(k+1)$, а $y = a(2k+3) = a(k+1)$. В этом случае $x > y$.

Получается, что большее из x и y соответствует четному n. Разберем случай, когда $x < y$. Мы знаем, что $x = a(2k+1) = a(k)$, $y = a(2k+2) = a(k) + a(k+1)$. Можно выразить $a(k) = x$ и $a(k+1) = y - x$. Таким образом, можно сперва решить задачу для пары $(x, y-x)$, а затем перейти от k к $2k+1$.

Если $x > y$, то мы можем решить рекурсивно задачу для пары $(x-y, y)$, а затем перейти от k к $2k+2$.

Этот процесс очень похож на алгоритм Евклида, работающий с вычитаниями. В конце, если мы пришли к паре $x = 1$, $y = 1$, то ответом будет $n = 0$; иначе, ответа не существует, что бывает в

случае не взаимно простых x и y. Алгоритм Евклида с вычитаниями работает за $O(x+y)$ и проходит все подгруппы, кроме последней.

Чтобы соптимизировать решени, нужно перейти к быстрой версии алгоритма Евклида со взятиями по модулю. Для этого нам надо уметь переходить от пары $(x, y \bmod x)$ к паре (x, y). Пусть $r = \lfloor \frac{y-1}{x} \rfloor$. Это значит, что к начальному n надо применить r раз операцию $n \to 2n + 1$. В двоичной системе счисления эта операция выглядит как сдвиг с прибавлением единицы. Таким образом, ответом будет $n \cdot 2^r + (2^r - 1)$. Похожим образом переходим от пары $(x \bmod y, y)$ к паре (x, y). Это решение работает за $O(\log^2(x + y))$, но также можно доказать, что оно работает за $O(\log(x + y))$.