Разбор задачи «Parking Problem»

Чтобы не дать Паулине припарковать свой автомобиль на как можно большей парковке, транспортные средства нужно размещать оставляя одно свободное место между ними.

Пусть у нас есть часть парковки вида: `.M.M.CC.M.CC.` — между транспортными средствами одно свободное место, слева и справа по одному свободному месту. Если есть два автомобиля и три мотоцикла, то это максимальная парковка, в которой они не оставят ни одного свободного места для автомобиля. Если взять парковку хотя бы на одну зону больше, то по принципу Дирихле хотя бы в одном из шести блоков свободных зон будет хотя бы две свободные зоны.

Пусть мы хотим научиться понимать, если мы добавим еще одно транспортное средство, как изменится максимальная парковка, которую мы сможем покрыть. `.M.M.CC.M.CC.CC.` — мы поставим еще одно транспортное средство и одну свободную зону. Получается $+3$, если это автомобиль, $+2$, если это мотоцикл. В общем случае, если есть a автомобилей и m мотоциклов, то максимальная парковка, которую можем покрыть имеет размер: $3a + 2m + 1$.

Теперь решим задачу, сделаем двоичный поиск, поймем, что у нас есть x автомобилей и y мотоциклов. А также нам заданы занятые зоны. Разобьем на последовательности связных свободных зон, у нас получились размеры блоков свободных зон: $s_1, s_2, \ldots, s_k$. Каждый из блоков свободных зон надо покрыть так, чтобы Паулина не смогла поставить автомобиль, и при этом мы не потратили лишнее число транспортных средств.

Если s_i четное, то $3a + 2m + 1$ следует быть четным, то есть a нечетным, чтобы не пришлось брать лишних транспортных средств. Если s_i нечетное, то a четным. Поэтому в первую очередь надо взять все четные s_i и в каждую из них определить по одному автомобилю, уменьшив s_i на 3. Если какому-то s_i не хватило автомобилей, то все будет покрываться мотоциклами, то есть должно быть $y \geqslant \sum \lceil \frac{s_i - 1}{2} \rceil$.

Если всем хватило автомобилей, то у нас есть еще лишние автомобили, а все s_i стали нечетными (так как из четных мы вычли 3). Выгоднее всего брать сразу по два автомобиля, чтобы нечетность сохранилась, при этом должно быть $a_i > 6$. Берем по два автомобиля, и уменьшаем a_i на 6. Если не осталось блоков размером больше 3 или если автомобилей осталось меньше двух, тогда можно оставшиеся автомобили переквалифицировать в мотоциклы (y увеличить на x), так как они уже нигде не помогут тем, что их ширина больше на единицу. После этого надо снова проверить $y \geqslant \sum \lceil \frac{a_i - 1}{2} \rceil$.

Разбор задачи «Diamond Hands»

Для того, чтобы было проще думать над этой задачей, изобразим точки (d_i, p_i) на плоскости. Тогда график цены представляет собой ломаную, начинающуюся в $(0, 0)$, у которой все отрезки идут вправо под углом 45 или -45 градусов.

Сначала поймем, в каком случае ответа не существует. Каждый день цена акции изменяется на единицу, поэтому сумма $d_i + p_i$ всегда должна быть четной. Если во вводе дается хотя бы одна пара с нечетной суммой, то ответ -1. Также, цена не может слишком сильно измениться между соседними точками: для двух пар (d_i, p_i) и (d_{i+1}, p_{i+1}) должно выполняться неравенство $|p_{i+1} - p_i| \leqslant d_{i+1} - d_i$. Если оба условия выполняются, то ответ всегда существует.

Есть два подхода к нахождению ответа с минимальным k. Первое решение использует динамическое программирование: обозначим `up[i]` как минимальное число отрезков, нужное, чтобы построить ломаную до i-й точки включительно, и последний отрезок шел вверх, аналогично `down[i]`. Сделать переход от `up[i]` к `down[i + 1]` можно, один раз изменив направление. При переходе от `up[i]` к `up[i + 1]` есть два случая: если $p_{i+1} - p_i = d_{i+1} - d_i$, то можно просто продолжить предыдущий отрезок и ничего не добавлять, иначе, придется сделать два изменения направления внутри отрезка. Переходы из `down[i]` аналогичны. В восстановлении ответа нужно аккуратно написать формулы, чтобы посчитать, в каких точках нужно менять направление. Получается решение за $O(n)$.

Оказывается, в предыдущем решении переход от `up[i]` к `up[i + 1]` (и симметричный с `down`), когда $p_{i+1} - p_i \neq d_{i+1} - d_i$ можно не рассматривать. Если мы в оптимальном решении все-таки используем такой переход, его можно заменить на переход `up[i]` $\rightarrow$ `down[i + 1]`, от этого ответ уменьшился на один, но следующий переход в динамике, возможно, стоит дороже. Если это был

переход `up[i + 1]` → `up[i + 2]` (а стал `down[i + 1]` → `up[i + 2]`), то ответ стал не хуже, а если дальше был переход `up[i + 1]` → `down[i + 2]` (а стал `down[i + 1]` → `down[i + 2]`), то количество отрезков могло увеличиться, если переход `down[i + 1]` → `down[i + 2]` стоил 2; в этом случае применим к нему аналогичную логику.

Это означает, что если $p_{i+1} - p_i = d_{i+1} - d_i$, то обязательно нужно сделать переход в `up[i + 1]`, а если нет, то оптимально изменить текущее направление движения. На основе этого наблюдения можно составить следующее жадное решение: переберем начальное направление движения и, если можем продолжить текущий отрезок, продолжаем его, иначе меняем направление на противоположное. Такое решение также работает за $O(n)$.

Разбор задачи «RPS string»

Для начала решим задачу для $d = 1$ и $\sum n \leqslant 5000$. Воспользуемся методом динамического программирования. Пусть $dp[i][j][c], c \in \{r, p, s\}$ будет являться истиной тогда и только тогда, когда мы можем удалить все элементы на отрезке кроме одного, и оставшийся элемент будет иметь символ c. Тогда $dp[i][j][c] = true$, если можно разделить наш отрезок на два подотрезка, в одном из них останется c, а в другом какой-то символ, который проиграет (или будет равен) символу c. Эту динамику можно пересчитать за $O(n^3)$.

Как теперь проверить, может ли в конце остаться символ на позиции i? Понятно, что i должен выиграть всех слева и всех справа, причём независимо. Поэтому слева и справа должны остаться какие-то элементы, которые проиграют или будут равны i-му. Тогда для какого-то c, который не выиграет s_i, $dp[1][i-1][c] = true$, и для какого-то другого c, который не выиграет s_i, $dp[i+1][n][c] = true$. Таким образом ответ по динамике можно посчитать за $O(n)$.

Теперь давайте посмотрим, что изменится в случае $d = 2$. Теперь нам запрещено удалять одинаковые символы. А давайте посмотрим, когда нам это мешает? Давайте теперь $dp[i][j][c] = true$, если в конце может остаться несколько элементов с символом c. Тогда эта динамика пересчитывается точно также, как и первая. Действительно, две группы с одинаковыми элементами объединяются в одну, а если одна группа побеждает другую, то можно это сделать поэлементно.

Как теперь проверить, может ли в конце остаться символ на позиции i? Теперь слева и справа должны остаться группы элементов, которые проиграют i-му. Ответ по динамике также можно посчитать за $O(n)$.

Давайте посмотрим, когда мы не можем оставить символ позиции i в конце, при $d = 1$. Это значит, что слева или справа есть элементы, которым мы проигрываем, и при этом нет элементов которые могут эти элементы выиграть. Если допустим у $s_i =$`'r'`, то слева либо нет элементов `'p'`, либо есть хотя-бы один элемент `'s'`. Поэтому группу $d = 1$ мы можем решить за $O(n)$, просто подсчитав префиксные суммы.

Теперь осталось понять, как решать группу с $d = 2$. На самом деле это тоже не очень сложно. Слева и справа должна остаться какая-то группа с символом c. Как проверить что можно оставить этот символ. Снова вспомним наши условия. Чтомы мы могли оставить на каком-то префиксе символ c, пред ним и после него должен быть хотя-бы один «хороший» элемент, либо не должно быть «плохих». Тогда нам достаточно рассмотреть всего 8 потенциальных позиций символа c — до первого плохого, после первого плохого, до первого хорошего, после первого хорошего, до последнего плохого на префиксе, после последнего плохого на префиксе, до последнего хорошего на префиксе, после последнего хорошего на префиксе. Все эти позиции можно легко найти за $O(1)$, а также проверить за $O(1)$ с помощью массива префиксных сумм, если просто для каждой позиции и для каждого символа хранить следующий и предыдущий такой символ для данной позиции. Тогда суммарно решение будет работать за $O(n)$.

Разбор задачи «Long puzzle»

Для начала, учтем в ответе детали, у которых обе границы являются плоскими. Такие детали нельзя ни с чем состыковать. Если длина детали равна l, она образует множество сама по себе, иначе — нет.

Рассмотрим граф на 4 вершинах. Каждой детали сопоставим ориентированное ребро в этом графе. Пусть для детали i ребро ведет из вершины v_i в вершину u_i. Сделаем так, что деталь x можно состыковать с деталью y (в порядке сначала x, потом y) тогда и только тогда, когда $u_x = v_y$. Это можно сделать, если вершины будут соответствовать, например:

1. Плоская левая граница
2. Выпуклая левая граница или вогнутая правая
3. Вогнутая левая граница или выпуклая правая
4. Плоская правая граница

Тогда множество деталей можно соединить в один кусок, если ребра, соответствующие деталям в этом множестве, образуют граф, в котором есть Эйлеров путь из вершины 1 в вершину 4.

Чтобы в ориентированном графе существовал Эйлеров путь, необходимо и достаточно, чтобы:

- У всех вершин, кроме крайних, входящая и исходящая степени должны быть равны.
- Для стартовой вершины исходящая степень должна быть на 1 больше входящей.
- Для конечной вершины входящая на 1 больше исходящей.
- Граф должен быть связен, если убрать ориентацию ребер.

Теперь можно перебрать две детали x и y: которые будут стоять первой и последней. У x левая граница должна быть плоской, у y — правая. Для этой пары деталей к ответу нужно прибавить количество множеств деталей без плоских границ, которые имеют суммарную длину $l - a_x - a_y$ и могут соединиться вместе с x и y в один кусок.

Для вычисления количества таких множеств, воспользуемся методом динамического программирования. Пусть мы набрали множество S деталей без плоских границ. Поймем, какая информация нас интересует про это множество. Конечно, нам нужно знать их суммарную длину, то есть $\sum_{z \in S} a_z$. Нужно знать разности между входящей и исходящей степенью вершин 2 и 3. А также, есть ли хотя бы одно ребро, соединяющее 2 и 3. А еще, в состоянии хранится длина префикса уже рассмотренных деталей.

Так как мы рассматриваем только детали без плоских границ, то $deg_{in}(2) - deg_{out}(2) = -(deg_{in}(3) - deg_{out}(3))$. Таким образом, существует всего $O(l \cdot n^2)$ различных состояний, пересчет происходит за $O(1)$: мы либо пропускаем очередную деталь, либо добавляем в множество.

Разбор задачи «Rooted MST»

Граф является вершиной 0 **соединенной с путем** $1 \to 2 \to \ldots \to n$

В этом случае, для решения задачи для одного запроса, нужно найти минимальную стоимость разбиения путь $1 \to 2 \to \ldots \to n$ на отрезки и соединения на каждом отрезке ровной одной вершины с вершиной 0.

Эта задача может быть решена с помощью тривиального линейного динамического программирования: $\mathrm{dp}_{i,0}, \mathrm{dp}_{i,1}$ будут обозначать минимальную описанную стоимость разбиения $1 \to 2 \ldots \to i$ на отрезки в случае если на этом отрезке уже выбрана ($\mathrm{dp}_{i,1}$) или еще нет ($\mathrm{dp}_{i,0}$) вершина для соединения с 0.

К сожалению, если отвечать на каждый запрос наивно за $O(n)$, решение будет работать за $O(nq)$, а это недостаточно быстро для решения подзадачи 6.

Чтобы отвечать на каждый запрос за $O(\log n)$, можно воспользоваться деревом отрезков для оптимизации ответов на запросы, по аналогии с $\mathrm{dp}_{i,0}, \mathrm{dp}_{i,1}$, мы будем хранить $\mathrm{dp}_{v,0,0}, \mathrm{dp}_{v,0,1}, \mathrm{dp}_{v,1,0}, \mathrm{dp}_{v,1,1}$ для каждой вершины v дерева отрезков, где соответствующие $\{0, 1\}$ описывают, соединен ли с нулем отрезок левой и правой границы соответственно.

С помощью аккуратного пересчета динамического программирования для каждого запроса, мы получаем решение за $O(q \log n)$, которое укладывается в ограничения подзадачи 6.

Общий случай

По аналогии с леммой о безопасном ребре, заметим, что среди ребер в индуцированном подграфе на вершинах $\{1, 2, \ldots, n\}$ можно оставить только ребра любого минимального остовного дерева.

Таким образом, в нашем графе есть фиксированный лес на вершинах $\{1, 2, \ldots, n\}$ и n ребер $0 \to i$ с динамическими весами a_i. Для простоты соединим компоненты связности ребрами веса ∞, чтобы преобразовать лес в дерево.

А далее следует ключевая идея задачи: можно преобразовать дерево на вершинах $\{1, 2, \ldots, n\}$ в путь $p_1 \to p_2 \ldots \to p_n$, не меняя ответы на запросы, а затем воспользоваться описанным выше решением для подзадачи 6.

Доказательство этого факта можно провести по индукции по размеру вершин: для одной вершины утверждение тривиально, а в противном случае рассмотрим ребро дерева с максимальным весом w, пусть его удаление разбивает дерево на две компоненты связности A и B. Построим по индукции произвольный необходимый путь для этих компонент ($P(A)$ и $P(B)$), а дальше, заметим, что одним из возможных путей для $A \cup B$ является путь $P(A) \to P(B)$ с добавлением ребра веса w между последней вершиной $P(A)$ и первой вершиной $P(B)$.

Это доказательство легко преобразуется в алгоритм с помощью сортировки ребер по весу и поддерживания системы непересекающихся множеств и списков в каждой компоненте.

Построение минимального остовного леса можно выполнить с помощью алгоритма Краскала за $O(m \log m)$, дальнейшее (описанное выше) преобразование этого леса в путь занимает $O(n \log n)$, а после перенумерации вершин пути $P(A) \to \{1, 2, \ldots, n\}$, мы получаем задачу эквивалентную подзадаче 6, решение которой за $O(q \log n)$ было описано выше.

Итого мы получаем решение за $O((n + m + q) \log(n + m + q))$.