Разбор задачи «Золотые слитки»

Для начала нужно проверить, можно ли разделить добычу поровну, не производя распил. При этом, одному разбойнику достанется один слиток, а другому — два.

Если же поделить добычу без распила невозможно, сделать это с помощью одного распила можно всегда. Это можно проиллюстрировать следующим образом. Отложим последовательно отрезки длин x_1, x_2 и x_3. Отметим точку $\frac{x_1+x_2+x_3}{2}$, она поделит какой-то отрезок на две части. Если разрезать слиток, соответствующий этому отрезку, на такие же части, на какие точка бьет отрезок, добычу станет возможно поделить на две равные части.

Разбор задачи «Автобус»

Рассмотрим некоторого пассажира. Заметим, что тот факт, будет ли он стоять или сидеть, не зависит от того, куда сядет Антон. Кроме того, если пассажир стоит, то место, рядом с которым он стоит, тоже не зависит от выбора Антона. Действительно: человек выбирает, будет ли он стоять или сидеть, только на основе количества сидящих и стоящих пассажиров в момент его входа, а это не зависит от выбора места Антона, который всегда сидит. А встаёт пассажир только тогда, когда все сидения заняты, причём выбирает себе место только на основе того, как стоят другие.

Таким образом, можно посадить Антона на любое место, промоделировать процесс, для каждого места посчитать, в течение какого времени рядом с ним будут стоять, и выбрать лучшее.

Моделирование делается следующим образом. Заведём set, в котором будем хранить свободные сидячие и стоячие места. Отсортируем события «пассажир вошёл» и «пассажир вышел» по порядку. При входе пассажира нужно выбрать ему место из множества свободных, запомнить его и удалить из множества. При выходе — добавить в множество.

Разбор задачи «Горные лыжи»

Для начала научимся решать более простую задачу: поиск максимального количества зимних дней, если Таня не могла вернуться с курорта после начала зимы.

Отсортируем последовательность дней. Теперь такую задачу можно решать при помощи динамического программирования. Пусть dp_i — это максимальное количество зимних дней в городе, если Таня вернулась в i-ый день про который точно известно, что она была на курорте и среди первых i дней условия не нарушились.

Будем считать значения dp_i от 1 до n. Пусть мы посчитали все значения до $i-1$, и на текущем шаге мы считаем dp_i. Рассмотрим максимальное значение l такое, что $d_l < d_i - k$. Если такого значения не существует, то других поездок не было и $dp_i = max(d_i - k, 0)$. Иначе $dp_i = dp_l + d_i - d_l - k$.

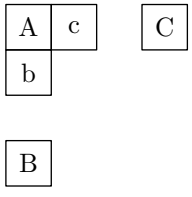
Необходимое значение l мы можем найти бинарным поиском или двумя указателями.

Вернемся к исходной задаче. Заметим, что только в одну последнюю поездку Таня могла вернуться весной. Причем эта поездка в оптимальной расстановке будет начинаться в один из дней, про который мы точно знаем. Переберем номер поездки i. Проверим, что если Таня поедет на курорт в i-ый день, про который мы точно знаем, то эта поездка будет последней и она посетит курорт во все известные дни после i. Это условие означает, что $d_i + k > d_n$. Теперь воспользуемся значением динамики. Если мы фиксировали начало последней поездки, то максимальное количество зимних дней равно $dp_{i-1} + max(n - d_i - k, 0)$. Ответ это максимум по всем i, для которых выполняется ограничение $d_i + k > d_n$.

Сортировка дней занимает $O(n \log n)$ времени, а подсчет dp_i и дальнейшее восстановление ответа — $O(n)$ времени.

Разбор задачи «IQ тест для роботов»

Докажем, что одна из клеток в ответе будет соседней с клеткой запроса. Пусть это не так, тогда рассмотрим следующую конструкцию:

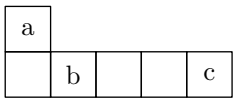


Клетка A — клетка запроса и клетки B и C — клетки в ответе, обе из которых не соседствуют с A. Рассмотрим несколько случаев:

1. Если клетки b и C имеют разный цвет, тогда b и C — более оптимальный ответ.
2. Аналогично, если B и c разного цвета, тогда взять клетки B и c оптимальнее.
3. Если же оба этих случая не выполняются, то цвет b совпадает с C, а также цвет c совпадает с B. Поскольку, по предположению, цвета B и C не совпадают, то и цвета b и c не совпадают, а значит, можно получить более оптимальную пару клеток.

Мы доказали, что одна из клеток в ответе — сосед клетки запроса. Поскольку соседей у каждой клетки не более четырех, его можно перебрать. Научимся быстро находить вторую клетку в ответе. Для этого посчитаем для каждой клетки ближайшую в каждом из четырех (вверх, вниз, влево и вправо) направлений клетку другого цвета. Это можно сделать с помощью динамического программирования.

Возьмем какое-то конкретное направление, например, вверх. Будем перебирать клетки в каждом столбце сверху вниз. При рассмотрении очередной клетки рассмотрим два случая: если клетка непосредственно сверху отличается цветом от текущей цветом, то, то искомой ближайшей отличающейся клеткой будет она. В противном случае, ближайшей отличающейся клеткой для текущей будет клетка, найденная ранее для ее верхнего соседа. Эта процедура просматривает каждую клетку только один раз, поэтому она работает за $O(nm)$. Аналогично сделаем для трех других направлений.



Окончательно, чтобы ответить на запрос, переберем клетку, которая является соседней с клеткой запроса, назовем ее a. Теперь переберем одно из двух направлений, в котором находится относительно клетки запроса вторая клетка пары. Посмотрим на соседнюю с клеткой запроса в этом направлении клетку b. Если они отличаются по цвету, то возьмем эти две клетки в качестве ответа. Иначе, в качестве второй клетки пары возьмем ближайшую от b в этом направлении клетку, отличную по цвету от b (клетка c).

В итоге, для ответа на запрос, необходимо рассмотреть не более восьми (четыре начальных и два перпендикулярных к нему направлений) случаев и выбрать среди них пару с наименьшим расстоянием.

Разбор задачи «Расшифровка ДНК»

Так как множество G является префиксным, разбивать строку s на строки из G можно жадным алгоритмом: пока строка s не пуста, находим строку из G, являющуюся префиксом s (такой строки всегда будет не более одной). Если такая строка нашлась, можно отрезать этот префикс от s и повторять до тех пор, пока строка s не станет пустой, либо в какой-то момент не найдется префикса s, находящегося в G. Во втором случае строку s нельзя разбить на строки из G. Отсюда, в частности, следует, что у любой строки s существует не больше одного такого разбиения.

Для решения задачи будем эмулировать описанный процесс сразу для всех строк из D, которые не были разбиты к текущему моменту. Также мы будем поддерживать бор на строках из массива G. Для всех же неразбитых строк из D будем поддерживать самую глубокую вершину в боре, которая соответствует самому длинному их префиксу, который присутствует в боре. Рассмотрим наши действия для обоих типов запросов.

- Запрос «? s». Будем последовательно обрабатывать символы строки s и переходить по ним в боре. Если мы пришли в терминальную вершину (вершину, в которой заканчивается какая-то строка из G), перейдем в корень бора и продолжим процесс.

  Если встретили символ, перехода по которому в боре нет, в последней вершине запомним номер этой строки и количество обработанных символов. В случае, если мы обработали все символы

строки, тогда, если мы закончили в корне бора, строка разбилась на строки из G, иначе — нет, и никогда не разобьется.

При этой операции в ответе может лежать максимум одна строка — только что добавленная.

- Запрос «`+` s». Новую строку необходимо добавить в бор. Каждый раз, когда мы добавляем в бор новый переход из вершины v, необходимо обновить все строки из D, которые находились в тот момент в вершине v. Все строки, у которых первый нерассмотренный символ был равен символу на новом переходе, следует переместить в новую созданную вершину в боре.

  После того, как мы полностью добавили строку s в бор, посмотрим на все строки из D, которые теперь лежат в вершине, соответствующей концу s, переместим их в корень бора и продолжим обрабатывать символы каждой из них, аналогично запросу `?` s. Все строки, которые в результате этого оказались успешно разбиты, следует вывести в ответ.

При эффективной реализации такое решение работает за $O\left(\sum\limits_{g_i \in G} |g_i| + \sum\limits_{d_i \in D} |d_i|\right)$, то есть за суммарную длину всех строк во входных данных. Построение бора из строк из G занимает $O\left(\sum\limits_{g_i \in G} |g_i|\right)$ времени. Обработка же строк из D, в свою очередь, занимает $O\left(\sum\limits_{d_i \in D} |d_i|\right)$ времени, поскольку каждый раз, когда мы двигаем строку из D по бору, мы уменьшаем количество оставшихся в ней символов, поэтому для каждой строки d_i будет выполнено не больше d_i передвижений.

Разбор задачи «Преобразование таблицы»

Будем хранить перестановки столбцов и строк таблицы — массивы pc, pr соответственно. $pc_i = x$ означает, что в i-м столбце сейчас на самом деле стоит столбец номер x изначальной таблицы. Аналогично работаем с перестановкой строк pr. Теперь рассмотрим, как будут выполняться операции:

- «`c` x y» — просто меняем столбцы в перестановке: `swap`(pc_x, pc_y);
- «`r` x y» — просто меняем строки в перестановке: `swap`(pr_x, pr_y);
- «`f` a b c d» — исходя из нашей информации, в клетке (a, b) сейчас на самом деле находится значение $matrix[pr_a][pc_b]$ изначальной матрицы (аналогично для клетки (c, d)). Поэтому просто поменяем значения в исходной матрице и продолжим выполнения запросов: `swap`($matrix[pr_a][pc_b]$, $matrix[pr_c][pc_d]$);

В конце, когда все запросы обработаны, составим из наших перестановок pr, pc новую матрицу, посчитаем ее контрольную сумму, это и будет ответом.

Разбор задачи «Архивы джедаев»

Первым запросом посмотрим, что записано в ячейке 1. Пусть это сведения о планете y. Мысленно вычтем из всех номеров планет $y - 1$. Если получилось число, меньшее 1, прибавим к нему 10^{18} (то есть, мы просто сдвинули номера по кругу, чтобы в первой ячейке оказалось 1). Получаем возрастающий массив чисел, начинающийся с 1. x в нём можно искать бинарным поиском, но на это потребуется 60 запросов.

Однако этот не произвольный массив — он получается из массива, содержащего все числа от 1 до 10^{18}, выкидыванием m чисел. Изначально число x стояло на позиции x, а затем оно не более m раз сдвинулось на 1 влево, либо было удалено. Поэтому границы бинарного поиска — не 1 и $10^{18} - m$, а $x - m$ и x.

$m \leqslant 500$, поэтому на бинарный поиск потребуется не больше 9 запросов. Всего, таким образом, будет не больше 10 запросов.

Разбор задачи «Музей»

Определим $opt_{A,B}$, где AB — диагональ зала, как треугольник минимальной площади, который содержит в себе все сувениры, которые находятся слева от AB, с вершиной в точке A. Тогда ответом будет являться минимум по всем $area(opt_{A,B}) + area(opt_{B,A})$.

Как считать opt? Зафиксируем точку A и будем перебирать точки B от неё по часовой стрелке. Хотим найти $opt_{A,B}$. Рассмотрим случай, когда AB является стороной треугольника. Нужно понять, в каких вершинах зала можно расположить третью вершину треугольника C.

Заметим, что для какая-то вершина C подходит тогда и только тогда, когда слева от диагонали AC нет сувениров и справа от диагонали BC нет сувениров. Оба этих условия по отдельности дают два отрезка подходящих вершин, каждый из которых можно предподсчитать заранее (суммарно за $O(nm)$), а вместе эти условия дают пересечение этих отрезков, что тоже является отрезком.

Обозначим его концы за L и R. Заметим, что при движении точки C от L к R расстояние от C до прямой AB сначала неубывает, а затем невозрастает (так как зал выпуклый), а значит минимум расстояния достигается в одном из концов отрезка, то есть в качестве точки C необходимо взять либо L, либо R (поскольку площадь треугольника равна половине расстояния от C до AB, умноженной на $|AB|$).

Теперь рассмотрим случай, когда AB не является стороной треугольника. Но все такие треугольники мы уже рассматривали, когда находили $opt_{A,B'}$, где AB' — предыдущая рассмотренная диагональ. Единственная проблема может быть лишь в том случае, когда между AB и AB' есть сувениры. Чтобы проверить этот случай, отсортируем все сувениры по полярному углу относительно A и будем поддерживать указатель на последний сувенир, находящийся левее текущей диагонали. Тогда, переходя к следующей диагонали, просто проверим, изменится ли этот сувенир.

Разбор задачи «Обычный мальчик»

Заметим, что если у числа x есть 100 делителей, то для любого натурального t, $x \cdot t$ имеет не менее 100 делителей. Для решения задачи воспользуемся этим фактом. Возьмем за x минимальное число, которое имеет 100 делителей — 45360.

Теперь если число n не менее, чем $100 \cdot x$ у нас существует решение вида $x \cdot t$, где $t = \lceil \frac{n}{x} \rceil$.

Для чисел менее, чем $100 \cdot x$ будем решать задачу перебором. Для этого рассмотрим все такие i, для которых выполняется ограничение из условия задачи, количество таких значений не превосходит $\lceil \frac{n}{100} \rceil$. Теперь для каждого i проверим количество делителей за $O(\sqrt{i})$.

Таким образом решение работает за $O(\lceil \frac{n}{100} \rceil \cdot \sqrt{n})$ для чисел меньших, чем 4536000 и за $O(1)$ для больших чисел.

Разбор задачи «Полиглоты-интроверты»

Вместо графа людей будет рассматривать граф между языками. Два языка соединим ребром, если есть человек, который говорит на обоих. Найдем для каждой пары языков минимальное количество человек, если первым языком в цепочке был А, а последним Б. Стоимостью цепочки языков $A_1 \to A_2 \to \ldots \to A_m$ назовем количество людей, которые знают хотя бы один из языков A_i.

Решение за $O(nk2^k)$: переберем множество языков, которые будут использованы и посчитаем количество людей, говорящих хотя бы на одном из них. Если это множество языков связно, то обновим ответ для всех пар языков этим количеством.

Для ускорения этого решения необходимо заметить следующий факт: если рассмотреть цепочку языков $A_1 \to A_2 \to \ldots \to A_m$, то множество людей, которые знают язык A_i, не знают никакой из языков $A_1, A_2, \ldots, A_{i-2}, A_{i+2}, \ldots, A_m$, поскольку в ином случае эту цепочку можно укоротить, а ее стоимость при этом не увеличится. Таким образом, люди, которые учитываются в ответе, знают только один или два подряд идущих языка в этой цепочке.

Сопоставим ребрам в построенном графе на языках следующую стоимость: стоимость ребра $A \to B$ будет равна количеству людей, которые знают язык B, но не знают язык A. В таком графе вес кратчайшего пути из A в B плюс количество людей, знающих A будет равен стоимости оптимальной цепочки, где первый язык это A, а последний — B. Все попарные кратчайшие расстояния в этом графе можно найти с помощью алгоритма Флойда за $O(k^3)$.

Осталось по этому графу языков построить ответ на задачу. Для этого для каждой пары людей можно перебрать начальный и конечный языки в цепочке и выбрать минимальный ответ среди таких пар языков. Эта часть может быть реализована за $O(n^2k^2)$.

Для того, чтобы соптимизировать эту часть, предподсчитаем вспомогательную величину $P_{i,j}$ — минимальная стоимость цепочки языков, которая начинается с какого-то языка, который знает i-й

человек, и заканчивается в j-м языке. Это можно предпосчитать за $O(nk^2)$. Теперь, используя эту величину, для пары людей достаточно перебрать лишь последний язык в цепочке, а первый язык однозначно определится из подсчитанной величины $P_{i,j}$.

Описанное решение работает за $O(\max(n,k)^3)$ и укладывается в ограничения по времени при $n, k \leqslant 300$.

Разбор задачи ««Исключающее или» наносит ответный удар»

Первое решение. Заметим, что ответ никогда не превосходит $a < 2^{60}$, так как $a \oplus a = 0$ и делится на n. Будем генерировать b по битам, начиная со старших бит и заканчивая младшими. Пусть нам уже известны несколько старших бит числа b. Рассмотрим очередной бит: выгодно поставить там 0, если это возможно — тогда итоговое число получится меньше. Таким образом, задача сводится к определению того, существует ли искомое b с некоторым фиксированным набором старших бит, при этом оставшиеся биты могут быть любыми. Это равносильно тому, что у нас фиксированы некоторые биты числа $a \oplus b$, а оставшиеся могут быть любыми.

a	0	1	1	0	1	...
b	1	0	?	?	?	...
$a \oplus b$	1	1	?	?	?	...

Таким образом, числа вида $a \oplus b$, где b имеет фиксированное начало, образуют некоторый непрерывный отрезок $[l, r]$, и нам необходимо лишь проверить, что этот отрезок содержит число, делящееся на n. Для этого достаточно проверить, что $l \geqslant r - r \mod n$.

Второе решение. Интуитивно ясно, что искомое b не очень велико, а значит $a \oplus b$ не сильно отличается от a. Более того, оказывается, что $|(a \oplus b) - a| < n$. Так как $a \oplus b$ обязано делиться на n, то это означает, что $a \oplus b$ равно либо $a - a \bmod n$, либо $a - a \bmod n + n$. Нетрудно заметить, что $b = a \oplus (a \oplus b)$, таким образом, ответ может быть найден как

$$\min\left(a \oplus (a - a \bmod n), a \oplus (a - a \bmod n + n)\right)$$

.