Задача A. Ограбление банка

Автор задачи и разработчик: Антон Вдовин

Обозначим i-й символ j-й полученной строки за s_i^j. Тогда исходная строка равна $s^1 = \overline{s_0^1, \ldots, s_{n-1}^1}$, а для каждой следующей $s_i^{j+1} = s_i^j \oplus s_j^{i+1}$. Сходу стоит заметить, что эта формула чем-то напоминает формулу для подсчета числа сочетаний: $C_{n+1}^k = C_n^k + C_{k-1}^n$.

Также далее будем пользоваться эквивалентностью операции исключающего ИЛИ сложению по модулю 2. Поэтому будем записывать $x \oplus x \oplus \ldots oplusx$, где x встречается y раз, как $y \cdot x$.

Распишем процесс полностью:

- Изначально строка равна s^1.
- Строка s^2 получается как
$$s_i^2 = s_i^1 \oplus s_1^{i+1}$$
- На следующем шаге s^3 так же выражается через s^2. Раскроем то, что получили выше, и получим, что
$$s_i^2 = s_1^i + 2 \cdot s_{i+1}^1 + s_{i+2}^1$$
- Если сложить полученные выражения для s_i^3 и s_{i+1}^3, получим
$$s_i^4 = s_i^1 + 3 \cdot s_{i+1}^1 + 3 \cdot s_{i+2}^1 + s_{i+3}^1$$
- И так далее, на каждом шаге будем выписывать XOR соседних элементов предыдущей строки.

Несложно заметить, что коэффициенты, получающиеся при битах исходной строки — это значения из треугольника Паскаля. А как известно, значения в треугольнике Паскаля — это биномиальные коэффициенты C_n^k. Таким образом:

$$s_1^{n-1} = C_{n-1}^0 s_0^1 \oplus C_{n-1}^1 s_1^1 \oplus \ldots \oplus C_{n-1}^{n-1} s_{n-1}^1.$$

Чтобы посчитать это значение, вспомним, что на самом деле мы считаем не сумму, а XOR, а значит нам важен только остаток полученной суммы по модулю 2. А найти четность биномиального коэффициента можно по *теореме Люка*, которая утверждает, что C_n^k нечетно тогда и только тогда, когда нет бита, равного 1 в k и 0 в n. Это можно проверить за $\mathcal{O}(1)$ следующим выражением: $k \wedge \neg n \neq 0$ (либо же просто проитерироваться по битам и сравнить каждый).

Таким образом, посчитаем четность каждого биномиального коэффициента C_{n-1}^i, умножим на соответствующий бит исходной строки, и сложим. Получим наш ответ за время $\mathcal{O}(n)$.

Задача B. Массивы Росомахи: математическая ловкость и фантазия

Авторы задачи и разработчики: Антон Вдовин и Даниил Орешников

Сделаем несколько наблюдений:

1. из любых трех подряд идущих чисел будет не более одного четного; следовательно, не более $\frac{n}{3}$ чисел будут четными;
2. если нечетные числа расположить подряд, то они всегда будут удовлетворять условию: действительно, разница между двумя нечетными числами на расстоянии не больше двух будет четной и будет не превосходить 4, следовательно, общих делителей у них не будет.

Докажем, что в большей части случаев задача решается и для $n \geqslant \frac{3m}{4}$. Для этого достаточно построить массив, содержащий все нечетные числа от l до r и ровно половину четных. Сначала расставим нечетные: это логично сделать в соответствии с наблюдением выше, упорядочив их подряд. Оставим при этом каждое третье место под какое-то четное число. Получим последовательность вида

$$1, 3, ?, 5, 7, ?, 9, 11, ?, 13, 15, \ldots$$

(здесь для примера взято $l = 1$). Как расположим четные числа? Заметим, что чтобы каждое четное было различным, достаточно расположить на месте ? четные, равные полусумме стоящих рядом нечетных. То есть:

$$1, 3, 4, 5, 7, 8, 9, 11, 12, 13, 15, \ldots,$$

но такое распределение не работает — на расстоянии 2 оказались числа 12 и 9.

Исправим возникшую проблему:

- заметим, что единственный общий делитель, который мог появиться — это 3, так как элементы вокруг четного числа t равны $t-3$, $t-1$, $t+1$, $t+3$;
- заметим также, что такая проблема возникает у каждого третьего четного числа, а также что четные числа идут с шагом 4, то есть ни для какого взятого четного t не использовано ни $t-2$, ни $t+2$;
- если же мы заменим t на $t-2$ или $t+2$, мы получим «блок» $[t-3, t-1, t \pm 2, t+1, t+3]$, в котором общие простые делители могут быть только 3 или 5.

Остается только заметить, что если t делилось на 3, то ни $t-2$, ни $t+2$ не делятся на 3, а также хотя бы одно из них не делится на 5. Выберем его и поставим вместо t. В итоге мы использовали все нечетные и ровно половину четных, то есть ровно $\frac{3m}{4}$ различных чисел.

Чтобы решить исходную задачу, повторим тот же шаблон, пока числа не закончатся, а затем просто зациклим последнюю тройку: $x, y, z, x, y, z, \ldots$. С учетом округления вниз и -1 в требовании к количеству различных чисел, такое решение проходит все тесты.

Задача C. Награда за спасение мультивселенной

Эта задача решается как перебором с мемоизацией, так и динамическим программированием (что по сути довольно близкие методы).

С динамическим программированием будет чуть больше оптимизационных проблем, потому что будет много недостижимых состояний, которые на самом деле даже не надо обрабатывать, а вот перебор с мемоизацией проходит по времени с достаточным запасом.

Давайте заметим, что у каждой награды есть три возможных варианта — либо отдать ее одному герою, либо другому, либо оставить и потом раздвоить. Перебирать все 3^n состояний при этом нет смысла — если есть несколько способов прийти к одному и тому же «состоянию», нас не интересует их количество. А состояниями в этой задаче можно считать пары (i, d), где i — количество рассмотренных наград (рассматриваем их по порядку), а d — абсолютная разница между тем, что получил Дэдпул, и тем, что получил Росомаха.

Из состояния (i, d) мы можем получить

- состояние $(i+1, d)$, если просто не отдадим i-ю награду никому;
- состояние $(i+1, d+a_i)$, если отдадим i-ю награду тому, у кого сейчас их суммарная стоимость больше;
- и состояние $(i+1, |d-a_i|)$, если отдадим тому, у кого сумма меньше.

При чем всего возможных состояний не более $5 \cdot 10^7$, из которых на самом деле многие могут оказаться недостижимыми (но не обязательно). Сделаем рекурсивный перебор с описанными переходами между состояниями, при чем при получении ответа для какого-то состояния будем запоминать его. Посещая это же состояние в следующий раз, мы сможем сразу вернуть ответ, поэтому общее время работы будет не больше, чем количество состояний плюс количество переходов, чего достаточно, чтобы пройти в ограничения по времени.

Ответом для каждого состояния будет минимальная сумма наград, не отданных никому. Запомнить из рекурсивного перебора необходимо только те пути, которые приводят нас в состояние $(n, 0)$ — «все награды рассмотрены, и герои получили поровну». Ответом тогда будет $0.5(\texttt{ans} + \sum a_i)$, где `ans` мы получили перебором.

Задача D. Милпул и Догпуля

Автор задачи: Егор Юлин, разработчик: Даниил Орешников

К ответу можно было прийти, сначала придумав достаточно наивное решение. Оно заключается в том, чтобы выписать n побитово в двоичной системе счисления, дописывая перед каждым битом 0, а в конце дописав бит 1. Тогда при восстановлении числа достаточно найти первую единицу на четной позиции, обрезать полученную строку q по ней, и перевести левую часть из двоичной системы в десятичную.

Однако такое решение требует от строки s быть длины $2\log_2 n + 1$. Альтернативный способ — закодировать число так, чтобы в его коде какая-то последовательность бит гарантированно не встречалась, и завершить код этой последовательностью. Тогда процесс слабо отличается от описанного выше.

Основная идейная сложность этой задачи — это понять, при чем тут $\sqrt{2}$. Правильный ответ — ни при чем, кроме того, что значения $\sqrt{2}$ и $\frac{1}{\log_2 \phi}$ очень близки друг к другу (≈ 1.41 и ≈ 1.44). Здесь за ϕ обозначено золотое сечение — решение уравнения $x^2 - x - 1 = 0$.

Поэтому значение $\sqrt{2}\log_2 n$ достаточно близко к значению $\log_\phi n$, особенно учитывая, что при $n \leqslant 10^{18}$, $\log_\phi n < 90$.

А этот вывод (либо дальнейшие рассуждения от наивного решения, что нам хочется гарантировать отсутствие какой-то последовательности бит в коде) уже должен натолкнуть на *фибоначчиеву систему счисления*. Ее полезное свойство в том, что в ней никогда не встречаются два единичных бита подряд.

Таким образом:

1. Для первого запуска посчитаем все числа Фибоначчи до 10^{18} и пройдемся по их уменьшению: если n меньше этого числа, выпишем 0, иначе выпишем 1 и уменьшим n. В конец допишем последовательность бит 110 или 011, после чего выведем полученный результат.

2. Для декодирования разрежем полученную строку по 110 или 011 (в зависимости от того, что было использовано), и декодируем тем же образом часть слева от этой тройки бит.

В ограничениях данной задачи такой метод позволял уложиться в ограничение на длину строки, так как числа Фибоначчи растут со скоростью $F_n \approx \phi^n$, а поправка на разницу между $\sqrt{2}\log_2 n$ и $\log_\phi n$ не превосходит 2.

Задача E. Потеря медальона

Автор задачи: Артемий Блинов, разработчик: Павел Скобелин

Для начала заметим, что не более чем за $x - 1$ запрос мы можем узнать остаток при делении на x текущей координаты медальона с помощью следующего алгоритма:

1. сделаем запрос числом x;

2. если получим ответ 1 — значит координата делится на x, то есть имеет остаток 0 при делении на x;

3. если получим ответ 0, то продолжим алгоритм, пока не встретим число 1.

Тогда, если мы потратили y действий, значит $y - 1$ раз мы получили ответ 0, на y-й запрос получили ответ 1. Тогда понятно, что изначальная константа имеет остаток $y - 1$ при делении на x. Также заметим, что последний (x-й) запрос можно не делать, если мы до него дошли, ведь в таком случае мы гарантировано получим ответ 1. Тогда, не более чем за $x - 1$ запрос можно получить остаток при делении на x начального числа.

Важно отметить, что также можно находить остатки при делении на несколько чисел — после нахождения остатка начального числа (назовем его s) при делении на x_1 (назовем остаток p_1) можно запомнить, сколько мы вычли из изначального числа, и после этого найти остаток при делении на x_2 числа $s - p_1$, тогда $s - p_1 \equiv p_2 \pmod{x_2}$, значит $s \equiv p_1 + p_2 \pmod{x_2}$.

Тогда, с помощью описанного выше алгоритма найдем остатки при делении начальной координаты s на первые семь простых чисел: 2, 3, 5, 7, 11, 13, 17. После будем иметь систему: $\forall 1 \leqslant i \leqslant 7,\ s \equiv r_i \pmod{p_i}$. Всего на это мы потратим $\sum_{i=1}^{7}(p_i - 1) = 1 + 2 + 4 + 6 + 10 + 12 + 16 = 51$ запрос.

В конце воспользуемся Китайской теоремой об остатках, которая говорит о единственности такого x, что $0 \leqslant x < \prod_{i=1}^{n} m_i$ и $\forall 1 \leqslant i \leqslant n$ можно записать $x \equiv r_i \pmod{m_i}$, при условии взаимной простоты m_i. В нашем случае модули очевидно взаимно просты (это различные простые), тогда в границах от 0 до $\prod_{i=1}^{7} p_i = 510510$ существует единственное число s, подходящее заданным условиям.

Для поиска этого ответа воспользуемся алгоритмом Гарнера для решения системы в условиях КТО, или даже просто переберем все возможные числа от 0 до $5 \cdot 10^5$ и проверим совпадение остатков. Даже решение без алгоритма Гарнера укладывается в ограничения по времени, так как сумма ответов по всем тестовым случаям не превосходит $5 \cdot 10^5$.

Задача F. Погоня в Пустоте

Автор задачи: Даниил Голов, разработчик: Константин Бац

Заметим, что если изначально есть тоннель из хотя бы одной локации с Алиотом в локацию n, то герои не смогут от него спастись, так как при любом маршруте движения героев они окажутся в локации n либо позже, либо одновременно со своим врагом. С другой стороны, если нет ни одного тоннеля от локации с Алиотом в локацию n, то герои всегда могут спастись, так как можно построить тоннель $1 \leftrightarrow n$ (если он еще не построен) и добраться до убежища первыми.

С учетом предыдущих замечаний, можно построить тоннели между всеми парами локаций, кроме пар локаций $a_i \leftrightarrow n$. Всего может быть $\frac{n \cdot (n-1)}{2}$ тоннелей, из них k тоннелей построить нельзя (именно в стольких локациях уже есть Алиот), а m тоннелей уже построено. Поэтому, если герои могут спастись, то дополнительно можно построить $\frac{n \cdot (n-1)}{2} - k - m$ тоннелей.

Таким образом в решении было достаточно проверить, существует тоннель, связывающий какой-нибудь a_i и n. Если такой тоннель есть, то ответ равен -1. Иначе ответ на задачу — $\frac{n \cdot (n-1)}{2} - k - m$.

Для хранения и быстрой проверки, находится ли Алиот в тоннеле j (то есть есть ли такой i, что $a_i = j$), можно вместо списка a_i хранить массив значений `marks`, где `marks`$[i] = 1$, если в локации i есть Алиот, и `marks`$[i] = 0$ иначе.

Тогда асимптотика такого решения будет равна $\mathcal{O}(n + m)$.

Задача G. Ловушка поезда

Автор задачи и разработчик: Павел Скобелин

Для начала научимся проверять, правда ли, что $n \leqslant x$ для произвольного x. Для этого отметим состояние в текущем вагоне, сделаем x шагов вперед, и после каждого шага будем менять состояние на противоположное тому, которое мы оставили в первом вагоне. Затем сделаем x шагов назад — если состояние в первом вагоне поменялось, то $x \geqslant n$. Аналогичным образом можно проверять, что n лежит от x до $2x$: просто в первой половине вагонов сделаем состояние равным первому, а во второй половине — противоположным.

Теперь воспользуемся классической идеей подъема по степеням двойки. Рассмотрим по очереди $x \in [1, 2, 4, 8, 16, \ldots]$. Как только найдем $2x \geqslant n$, будем понимать, что $n \in (x, 2x]$.

Теперь, зная максимальную границу ответа (причем $x < n \leqslant 2n$), пройдем вперед $2x$ шагов и выключим все обогреватели. Так как вагонов в поезде не более $2x$, значит, теперь все обогреватели выключены. После этого включим обогреватель в текущем вагоне, пройдем x шагов вперед (все еще не вернувшись в исходный, так как $x < n$), и будем делать по одному шагу, каждый раз проверяя состояние обогревателя в текущем вагоне. Как только мы встретим первый включенный — значит, мы прошли по циклу.

При аккуратной реализации такой алгоритм займет не более $15 \cdot n - 10 < 16 \cdot n$ действий.

Задача H. Героический поступок

Автор задачи: Александр Гордеев, разработчик: Константин Бац

В задаче требовалось соединить три точки линиями так, чтобы линии шли по вертикалям или горизонталям клетчатой сетки и при этом имели минимальную суммарную длину.

Рассмотрим линию, которая в соответствии с условием соединяет две точки, например, первую и вторую. Длина такой линии — *манхэттенское расстояние* между двумя точкам, которое равно $|x_1 - x_2| + |y_1 - y_2|$. Заметим, что то же самое касается и кратчайших расстояний между остальными парами точек.

Давайте докажем, что итоговый ответ можно найти по формуле $d(i, j) = \max(x_i) - \min(x_i) + \max(y_i) - \min(y_i)$. К этому можно прийти двумя способами:

1. Более интуитивный способ, который сложнее доказать — заметить, что так как любые две точки должны быть соединены, путь между точками i и j будет не короче $|x_i - x_j| + |y_i - y_j|$. При этом если мы проведем провода так, что они будут представлять три ломаных из трех наших точек в одну и ту же точку плоскости «между ними», то каждая эта ломаная войдет в два пути. Поэтому ответ равен

$$0.5 \cdot (d(1, 2) + d(2, 3) + d(1, 3)),$$

   что, в свою очередь, равно полусумме $(\max - \text{mid}) + (\text{mid} - \min) + (\max - \min)$, то есть просто $\max - \min$ по каждой из координат.

2. Либо можно было заметить, что нам понадобится минимум $\max x_i - \min x_i$ горизонтальных отрезков, чтобы соединить две наиболее удаленные по горизонтали точки, и то же самое по вертикали, поэтому меньшего ответа добиться нельзя. А пример с таким ответом строится достаточно тривиально (в зависимости от того, является ли «средняя» по x точка минимальной или максимальной по y, или нет).

Задача I. Нестабильность времени

Давайте для начала упростим условие и решим упрощенную версию задачи: пусть каждое измерение сообщает нам результат сравнения между двумя линиями времени. Тогда задача превращается в достаточно несложную: если мы представим каждое сравнение как ребро из большего значения в меньшее, то

- если в итоговом графе есть циклы, мы автоматически получаем противоречие;
- иначе, можно сделать топологическую сортировку, назначить всем истокам минимальное возможное значение (1), а для каждой следующей вершины посчитать ответ как $d_v = 1 + \max\limits_{v \to u} d_u$.

Действительно, нет смысла ставить значения нестабильности больше минимально подходящих, расставим минимальное число, которое удовлетворяет всем известным сравнениям, и в конце останется только проверить, что никакое значение не оказалось больше разрешенного максимума (10^9).

Таким образом, «простую» версию задачи можно решить за $\mathcal{O}(n+m)$. Теперь перейдем к оригинальному условию. Заметим, что его можно решить тем же самым методом, но проблема в том, что теперь каждое наблюдение порождает максимум $\frac{n^2}{4}$ ребер типа «одно больше другого». На графе с таким количеством ребер исходное решение получит превышение лимита времени работы.

Единственное, что нужно для полного решения задачи — придумать, как задавать тот же граф сравнений менее явно, чтобы не создавать по ребру на каждую пару сравниваемых значений. Для этого можно заметить, что мы оперируем по большей части отрезками подряд идущих по номерам линий времени, а значит можно какие-то отрезки воспринимать как самостоятельные сущности. Эта идея должна натолкнуть вас на структуру дерева отрезков, что позволяет придумать следующее решение.

1. Создадим структуру дерева отрезков на наших линиях времени;

2. Для каждого измерения $(l_i, r_i, \{x_j\})$ вместо того, чтобы проводить ребра между каждым x_j и каждым не-x, создадим еще одну фиктивную вершину u и проведем ребра $x_j \to u$ для всех j;

3. Затем посмотрим на отрезок $[l_i, r_i]$ и удалим из него все x_j. Он разобьется не более чем на k_i+1 подотрезков. Каждый из них разбивается не более чем на $\log_2 n$ отрезков из дерева. Проведем ребра из u в каждый из них.

4. Наконец, нам нужно, чтобы построенная нами конструкция соответствовала конструкции из решения простой версии задачи, то есть чтобы для каждого измерения были пути из каждого большего значения в каждое меньшее. Для этого достаточно в дереве отрезков провести ребро новой категории «больше или равно» из каждой вершины в двух ее детей.

Заметим, что в сумме мы провели не более $\sum k_i + \sum(k_i+1)\log_2 n$ ребер, что в условиях данной задачи значительно меньше n^2. И при этом теперь из любого большего значения можно попасть по ребрам в меньшее по пути через фиктивную вершину и затем вниз по дереву.

Применим к получившемуся графу исходное решение. Единственный момент, который осталось отметить — в новом графе не будет циклов только из ребер «больше либо равно», поэтому наличие цикла — все так же сразу противоречие, но вот формулу пересчета надо поменять: $d_v = \max\limits_{v \to u}(d_u + w_{vu})$, где w равно нулю для ребер «больше либо равно» и 1 для ребер «больше».

Задача J. Другие

Автор задачи и разработчик: Егор Юлин

В задаче по сути требуется отвечать для вершин на запросы количества их потомков на определенной глубине от них. Будем решать задачу в оффлайне — прочитаем все запросы сразу же, и для каждой вершины дерева сохраним, на каких глубинах мы хотим получить ответ.

Тогда решение будет иметь следующий вид: спускаемся по дереву и в вершине храним `map`, где ключом является высота, а значением — количество детей на такой высоте. Достаточно хранить только информацию про глубины, которые интересны в этой вершине или в каком-то из ее предков.

Пересчитывать эту информацию для родителей через детей может потребовать много ($\Omega(n^2)$ времени, но можно при выходе из детей объединять `map` детей в одну при помощи *small-to-large* оптимизации: всегда будем сохранять наибольшего по количеству информации детей и «докладывать» в него информацию о количестве потомков других детей.

Тогда каждая такая операция обновления информации для каждого элемента `map` случится не более $\log n$ раз. Действительно, если считать «размером» `map`'ы количество детей в поддереве текущей вершины, то после переноса элементов из меньшего множества в большее, они оказываются в множестве, как минимум в два раза большем исходного. Понятно, что такое не может случиться больше $\log_2 n$ раз.

После обхода всех детей мы можем пройти по всем запросам для текущей вершины и записать ответ для них. Это решение будет работать за $\mathcal{O}(n \log^2 n + q \log n)$.

Задача K. Темпоральные дожди

Автор задачи: Артемий Блинов, разработчик: Егор Юлин

Будем решать задачу следующим образом:

Заведём СНМ, в котором будем хранить следующую информацию: количество ячеек в множестве, максимальную высоту $d_{i,j}$, соприкасается ли хотя бы одна из ячеек с границей.

Отсортируем ячейки по высоте $h_{i,j}$ и будем обрабатывать их по возрастанию высот. Пусть мы рассматриваем ячейку (i, j). Рассмотрим все соседние для нее, обозначим такие ячейки за (i', j'). Для них верно, что $|i - i'| + |j - j'| = 1$. Если соседняя еще не была обработана то пропустим её.

Теперь есть два случая:

1. Если в СНМ для (i', j') достижима граница, то высота никак не изменится, так как там уже установился максимально доступный уровень энергии, а вся остальная энергия будет просто переливаться через край.

2. Если в СНМ для (i', j') граница не достижима, то текущее множество вершин из компоненты (i', j') ограничено высотой хотя бы $h_{i,j}$ (ведь иначе мы обработали эти ячейки раньше). Тогда высота каждой ячейки изменится на $h_{i,j} - d_{i',j'}$. После этого нужно для всех ячеек компоненты установить d, равное $h_{i,j}$.

После обработки ячейки (i, j) ее нужно объединить со всеми ячейками (i', j').

В сумме решение потребует $\mathcal{O}(nm \log nm)$ времени как минимум на сортировку по высоте. Работа с СНМ займет асимптотически меньше времени.

Задача L. Эпилог

Достаточно классическая задача, похожая на задачу о рюкзаке. Основное отличие — предметов мало, но вместо этого для каждого предмета есть большое количество возможных замен.

Решение при этом тоже не сильно отличается от решения задаче о рюкзаке: заведем динамику $\texttt{dp}[w][i]$ — минимальный «штраф» за то, чтобы набрать суммарную площадь w с помощью первых i фото. Порядок пересчета динамики будет сначала по увеличению i, а затем по увеличению w. Останется только для каждого фото перебрать b_i и сделать обновление

$$\texttt{dp}[w][i] = \min_{1 \leqslant b_i \leqslant a_i} (\texttt{dp}[w - b_i^2][i-1] + (a_i - b_i)^2).$$

Общее время работы решения — $\mathcal{O}(mn \max a_i)$.