Задача A. Крепление парусов

Разобьем задачу на две подзадачи: генерировать все 3^n возможных последовательностей состояний и проверять для каждой из них, возможно ли ее получить описанными действиями, будет слишком долго. Для этого заметим, что некоторые последовательности состояний «эквивалентны»: они отличаются только длинами отрезков узлов в одном состоянии, но имеют одинаковые последовательности этих состояний. Например, «`rrbwwbbrrr`» и «`rbbbbbwbbbbr`» эквивалентны — в них последовательности состояний описываются строкой «`rbwbr`».

Таким образом,

1. назовем такие последовательности состояний «масками», и найдем для каждой маски, сколько существует соответствующих ей ответов;
2. найдем сумму полученных ответов по всем достижимым за q действий маскам.

Но и это еще не все. Попробуем еще больше сократить количество различных масок.

- Заметим, что узлы в состоянии '`w`' ни разу не переходили в другие состояния. Это означает, что задачу можно независимо рассматривать на отрезках, которые получаются при разбиении маски по символам '`w`'. Назовем полученные отрезки маски «независимыми отрезками».
- Для каждого независимого отрезка мы знаем, что в нем символы '`r`' и '`b`' чередуются, поэтому всего их может быть не более 140 различных: по две конфигурации каждой длины. Найти минимальное число действий, за которое какой-то отрезок может быть получен, тоже просто: первое действие — всегда смена статуса на '`r`' на всем отрезке, а затем за каждое действие добавляется не более двух различий в статусе на соседних позициях (то есть количество действий примерно равно половине длины такого независимого отрезка).
- Сопоставим каждому независимому отрезку его длину. Затем заметим, что маски, отличающиеся порядком их независимых отрезков, не отличаются ни количеством соответствующих им ответов, ни достижимостью за q действий. А независимые отрезки, не отличающиеся по длине, также эквивалентны. Тогда можно каждую маску задать отсортированной последовательностью длин ее независимых отрезков.

Если задавать каждую маску последовательностью длин ее независимых отрезков без учета порядка, то всего различных масок будет не больше, чем число разбиений чисел от 1 до 70 на слагаемые без учета порядка, что не превосходит $\approx 4.2 \cdot 10^5$. Соответственно, для решения, проходящего по времени, достаточно было перебрать все такие маски, для каждой проверить достижимость за q действий, и в случае достижимости, прибавить к ответу число последовательностей статусов, соответствующих этой маске.

Последнее (поиск числа последовательностей, соответствующих данной маске), можно было решать также через оптимизированный перебор, либо через динамическое программирование. Оба варианта были примерно эквивалентны по времени работы.

Задача C. Петух Хей-Хей и камни

Автор задачи и разработчик: Даниил Голов

Заметим следующий факт: относительный порядок петухов никогда не меняется. Действительно, для петухов, движущихся в одном направлении, это очевидно, а для движущихся в разном известно, что они движутся к общей цели — первый из них, достигший цели, достигнет ее раньше, чем встретится со вторым, значит они не успеют поменяться местами.

А в таком случае задачу можно переформулировать следующим образом: есть n позиций петухов a_i и m камней. Камни надо обрабатывать в порядке уменьшения их k_i, и для очередного камня на позиции b:

- надо найти $d = \min(|a_i - b|)$ — расстояние, которое пройдут все петухи, пока один из них не окажется рядом с камнем;
- после чего все $a_i \leqslant b$ увеличить на d;

- а все $a_i > b$ уменьшить на d.

Поиск очередного камня можно выполнять за $\mathcal{O}(1)$ — достаточно их заранее отсортировать по k_i. Для выполнения оставшихся операций достаточно поддерживать декартово дерево всех позиций петухов. Чтобы найти d, достаточно сделать `split`(b) и рассмотреть крайние элементы двух полученных деревьев. Операции увеличения или уменьшения позиций петухов на d — это просто отложенные массовые операции, которые можно применять к полученным двум деревьям за $\mathcal{O}(1)$.

Итого решение работает за $\mathcal{O}((m+n)\log(n+m))$.

Задача D. Очередная игра

Автор задачи: Артемий Блинов, разработчик: Константин Бац

Основная идея в этой задаче — понять, как вовремя помешать противникам выиграть. Оказывается, что это достаточно просто сделать — поскольку каждый игрок поставит в сумме только три свои маркера на поле, у каждого есть ровно один шанс на победу — выстроить линию своим третьим ходом.

Заметим, что поскольку мы ходим вторыми, то

- к моменту нашего второго хода первый игрок уже поставил два своих маркера; в таком случае если существует свободная клетка, образующая с ними линию, мы просто походим в нее, забрав у второго шанс на победу;
- к моменту нашего третьего хода, третий игрок уже поставил два своих маркера; аналогично, если они выстраиваются в линию с пустой клеткой, мы можем заблокировать ее.

Первый же ход мы можем сделать произвольным образом — от него ничего не зависит. Все, что осталось — аккуратно реализовать описанную стратегию. Поскольку поле всего лишь 3×3, перебирать выигрышные линии и доступные ходы можно было произвольным образом, даже полным перебором всех возможных действий.

Задача E. Сломанный крюк

Автор задачи: Антон Вдовин, разработчик: Егор Юлин

Для начала заметим, что нам достаточно рассматривать подстроки длины только 2 или 3. Действительно, пусть есть подстрока-палиндром большей длины, но тогда в ней точно есть подстрока-палиндром длины 2 или 3 (мы можем удалить одинаковое количество символов с концов, пока не получим необходимые длины).

Далее есть два решения. Рассмотрим решение, которое использует динамическое программирование.

В $\texttt{dp}[i][f_1][f_2]$ будем хранить минимальное число изменений на префиксе длины i, где f_1 и f_2 — флаги того, были ли изменены последние два символа. Тогда если текущая позиция i, $s_i = s_{i-1}$ и символ на позиции $i-1$ не был изменен, мы должны поменять символ на позиции i. Аналогично с позицией $i-2$. Получается пересчет всей динамики за $\mathcal{O}(n)$.

Покажем, как восстанавливать ответ — это менее тривиально, чем просто посчитать динамику. Пусть мы знаем, что символ на позиции i нужно заменить, тогда найдем любой символ, которого нет в подстроке $\overline{s_{i-2}s_{i-1}s_is_{i+1}s_{i+2}}$, и присвоим в s_i этот символ. Такой подход работает, так как в нам достаточно добиться отсутствия палиндромов длины 2 и 3, то есть необходимо и достаточно избавиться от повторяющихся символов в таких строках. Собственно, ровно этого мы и добиваемся такими заменами.

Задача F. Отчетность

Автор задачи: Даниил Голов, разработчик: Егор Юлин

Обозначим за $A = \max(a)$.

Немного переформулируем задачу. Запишем $a_i \bmod t$ как $a_i - \lfloor \frac{a_i}{t} \rfloor \cdot t$, и обозначим $\lfloor \frac{a_i}{t} \rfloor$ за d_i. Тогда для максимизации $\sum\limits_{i=1}^{n}(a_i \bmod t)$ нам нужно минимизировать $\sum\limits_{i=1}^{n}(d_i \cdot t) = t \cdot \sum\limits_{i=1}^{n} d_i$.

Посмотрим, как меняется d_i при различных t.

- если $t \leqslant \sqrt{A}$, то $\sqrt{a_i} \leqslant d_i \leqslant a_i$;
- если $t > \sqrt{A}$, то $d_i \leqslant \sqrt{A}$.

Тогда для $t \leqslant \sqrt{A}$ переберем t, и для каждого найдем ответ; это будет работать за $\mathcal{O}(n\sqrt{A})$.

Для всех остальных t мы знаем, что d_i принимает не более $\sqrt{A}$ различных значений; найдем отрезки, на которых оно принимает определенные значения. После чего будем обрабатывать различные d_i при помощи сортировки событий, при этом будем обрабатывать все d_i одновременно.

Так как каждое d_i принимает не более $\sqrt{A}$ различных значений, то всего будет не более $n\sqrt{A}$ событий и время работы будет равно $O(n\sqrt{A})$. Интервал t, при котором d_i неизменно и равно фиксированному числу, можно найти либо формулой, либо двоичным поиском. Решение через формулу гарантированно проходило по времени.

Итоговое время работы равно $\mathcal{O}(n\sqrt{A})$.

Задача G. Безопасное мореплавание

Переформулируем задачу. Дан неориентированный граф, требуется для каждого ребра определить максимальный его вес, при котором оно будет входить в минимальное остовное дерево.

Задача разбивается на несколько похожих случаев. Проще всего начать со случая ребер, которые не входят в MST. Для таких ребер достаточно заранее построить произвольное MST, после чего воспользоваться леммой о безопасном ребре: чтобы ребро входило в MST, оно должно быть одним из минимальных в каком-то разрезе графа, разделяющем концы этого ребра.

Иными словами, построим MST и обозначим множество ребер в нем M. Рассмотрим какое-то ребро $e \notin M$. Если добавить это ребро в MST, мы получим множество $M \cup \{e\}$, в котором есть цикл, проходящий по e. Чтобы существовал минимальный остов, содержащий e, как следствие из леммы о безопасном ребре, необходимо и достаточно, чтобы на цикле существовало ребро веса не меньше p_e. Таким образом, ответ для $e \notin M$ — это вес максимального ребра на образующемся цикле, который можно найти с помощью двоичных подъемов и `lca` за время $\mathcal{O}(\log n)$.

В случае с ребрами $e \in M$ все аналогично: до какого-то момента можно просто увеличивать вес ребра, но в какой-то момент появится «замена», которую можно добавить вместо e, уменьшив вес дерева. Пусть при удалении e дерево M разбивается на M_1 и M_2. Тогда заметим, что $p_e = \min\limits_{u \in M_1, v \in M_2, (uv) \notin M} w_{uv}$ — это вес минимального ребра между M_1 и M_2, на которое можно заменить удаляемое e. При большем весе e эта замена приводит к уменьшению веса остова, что означает, что e никакому минимальному остову не принадлежит.

Осталось эту величину посчитать для каждого $e \in M$. Для этого есть два способа:

- Проще всего ее посчитать подобием динамики по дереву: будем для каждой v поддерживать множество ребер, ведущих из поддерева v в «над-дерево» $\texttt{parent}_v$. Для этого необходимо идти по дереву снизу вверх, и для каждой v объединять такие множества для детей, после чего удалять все ребра, смежные с v.

  При таком алгоритме каждое ребро будет ровно один раз добавлено в какое-то множество и ровно один раз удалено, поэтому в сумме это займет $\mathcal{O}(m \log m)$ времени. А объединения множеств с помощью техники small-to-large займут в сумме $\mathcal{O}(m \log^2 m)$ времени.

- Возможно, решение за такую асимптотику не будет проходить по времени — тогда достаточно заметить, что любое ребро $g \notin M$ может являться «заменой» для каждого ребра на пути в M между его концами. То есть вместо того, чтобы для каждого $e \in M$ считать $p_e = \min\limits_{u \in M_1, v \in M_2, (uv) \notin M} w_{uv}$, будем рассматривать $(uv) \notin M$ и выполнять обновление p_e для всех e на пути в M между v и u. Такие обновления можно выполнять с помощью отложенной информации на тех же самых двоичных подъемах, за суммарное время $\mathcal{O}(m \log m)$.

Задача H. Целая медиана

Автор задачи и разработчик: Даниил Голов Будем рассматривать описанный в задаче процесс с

конца. Представим, что все числа уже добавлены в множество. Тогда медиана этого набора вычисляется однозначно — это просто медиана всех чисел из a. Если эта медиана нецелая, что возможно только при четной длине a, то ответ -1, так как вне зависимости от порядка добавления нецелая медиана будет достигнута.

Теперь будем удалять числа из нашего множества так, чтобы медиана оставалась целой. Этот процесс будет аналогичен добавлению, только протечет в обратном порядке, соответственно, в конце ответ нужно будет развернуть.

Если множество четного размера и имеет целую медиану, то мы можем удалить сначала минимум, потом максимум, и медиана все еще останется той же, либо множество опустеет. То есть мы сократим размер множества на 2, оставив медиану целой на любом шаге.

Если же множество нечетного размера и содержит более одного элемента, то посмотрим на три центральных элемента в отсортированном порядке. Какие-то два из них точно дадут четную сумму по принципу Дирихле. Тогда удалим третий элемент и получим множество четного размера с целой медианой, которое мы строить уже умеем.

Задача I. Язык племени Мотунуи

Когда в задаче требуется найти множество лексикографически минимальных строк, имеет смысл воспользоваться бором. Разумеется, просто построить бор на всех возможных звучаниях слов не особо реалистично, так как нас интересуют k минимальных, а всего их порядка 2^n, что сильно больше.

Однако можно использовать несколько оптимизаций, чтобы строить k лексикографически минимальных строк быстрее. Для этого необходимо понимать, как строится лексикографически минимальная строка, а также как по любой строке построить следующую в лексикографическом порядке.

Построить минимальную строку не сложно: достаточно заметить, что она состоит из одного единственного минимального звука. Соответственно, эта задача просто эквивалентна поиску минимума в массиве a.

Поиск лексикографически следующей строки устроен следующим образом:

1. найти ближайший к концу текущей строки символ, который можно увеличить;
2. увеличить его на минимальное возможное значение, большее текущего;
3. дополнить минимальным возможным суффиксом.

Итого: в качестве минимальной строки выберем самое левое вхождение минимального звука в a. Далее k раз сделаем следующее:

1. если последняя буква текущего слова — не последняя буква алфавита, можно увеличить слово, дописав к нему в конец минимум из оставшихся букв; хеш строки при этом меняется понятным образом;
2. иначе — предподсчитав для каждого символа строки минимальный, больший его, справа от него (что можно сделать одним проходом со стеком), бинпоиском найдем первый с конца символ, для которого справа от него есть какой-то больший, после чего сделаем замену; минимальный возможный суффикс, который можно дописать — пустой.

Для эффективного выполнения этих операций достаточно иметь дерево отрезков на операцию «минимум» на алфавите (или просто суффиксные минимумы), а также хранить текущую строку в декартовом дереве по неявному ключу (в вершине храним флаг «есть ли в поддерева буква, которую можно увеличить», а также хеш поддерева). Хеши пересчитываются автоматически, а все описанные операции сводятся к операциям `split` и `merge` для ДД по неявному ключу — общее время работы $\mathcal{O}((n+k)\log n)$.

Задача J. Спасение Полинезии

В данной задаче требовалось построить минимальное остовное дерево на графе специального вида: в нем может быть слишком большое количество неявно заданных ребер, однако эти ребра

следуют определенным правилам из условия: из вершины $n + i$ ребра одинакового веса ведут в вершины с l_i по r_i.

Поскольку на таком графе может быть порядка nq ребер, просто построить этот граф в явном виде не получится. Но посмотрим, как будет вести себя алгоритм Краскала, если такой граф все же построить:

- ребра будут рассматриваться в порядке возрастания c_i;
- то есть ребра, ведущие куда-то из $n + i$, для каждого фиксированного i, будут рассмотрены подряд;
- и все эти ребра, соединяющие вершины из разных компонент связности, будут взяты в MST.

Поэтому сначала отсортируем все данные в условия описания ребер по возрастанию c_i. А затем будем выполнять шаги из алгоритма Краскала, но оптимизированным образом.

1. Будем рассматривать ребра $(n + i, l_i)$, $(n + i, l_i + 1)$, $\ldots$, $(n + i, r_i)$ именно в таком порядке (порядок ребер одного веса на корректность алгоритма Краскала не влияет).
2. Заметим, что вершина $n+i$ пока что ни с чем не соединена, поэтому первое такое ребро (между $n + i$ и l_i) гарантированно будет добавлено в MST.
3. Для каждого следующего ребра заметим, что $n + i$ и $j + 1$ находятся в разных компонентах связности тогда и только тогда, когда j и $j + 1$ находятся в разных компонентах связности: действительно, если мы рассматриваем ребра по возрастанию их конца, то к моменту рассмотрения $j + 1$ вершины $n + i$ и j уже связаны ребром или путем.

Более того, итоговый граф будет связен тогда и только тогда, когда для любого j от 1 до $n - 1$ вершины j и $j+1$ находятся в одной компоненте связности. Поэтому мы свели всю задачу к проверке связности соседних из первых n вершин.

А для этого достаточно поддерживать множество всех несвязных соседних вершин, и при обработке всех требуемых ребер с помощью бинпоиска в этом множестве находить все несвязанные пары из отрезка $[l_i, r_i]$ и удалять их. В сумме такое решение работает за $\mathcal{O}((n + q) \log n)$.

Задача K. Первые шахматы

Автор задачи и разработчик: Павел Скобелин

В этой задаче нужно было рассмотреть много случаев. В начале удобно проверить, что первый игрок не выигрывает за один ход.

Отметим ряд неочевидных случаев:

- Доска 2×2, две ладьи стоят по диагонали.
- Доска $2 \times n$, $n > 2$, два слона одного цвета (игра на четность).
- Доска 3×3, слон стоит на клетке 1, 1; ладья стоит на клетке 2, 1; слон ходит первый — ничья. (также всевозможные развороты/исходы из этой партии являются ничейными).
- Доска $3 \times n$, $n > 3$, ладья всегда выигрывает слона (можно построить конструктивный способ загнать слона в угол доски.
- Доска $3 \times n$, играют 2 слона с координатами (2, 1) и (2, 3). Если первый слон ходит первым, то он проигрывает. Аналогично, если второй слон ходит первым и может сделать ход в клетку (2, 3), то он выигрывает.
- Доска 4×4. Если ладья ходит второй и стоит в клетке (1, 3), а слон стоит в клетке (3, 4), то ладья выигрывает. Аналогично, если ладья ходит первой и ей доступна для хода клетка (1, 3), то она выигрывает. Также особыми случаями являются все зеркальные отражения/повороты этой ситуации.

- Доска $4 \times n$. Возможно ситуация, аналогичная ситуации выше, если слон стоит на краю доски.

Все остальные случаи являются ничейными.

Все эти случаи довольно сложно заметить, поэтому для маленьких досок можно было написать ретроанализ позиции, представив как вершину графа позицию первого игрока, второго игрока и очередность хода. Рассмотрев результаты игр, можно было заметить все крайние случаи и заметить неочевидные случаи на досках $2 \times n$, $3 \times n$, $4 \times n$.

Задача L. Магические ракушки

Формально, в задаче требуется максимизировать сумму двух элементов массива a, на индексы которых наложено ограничение: их побитовое «ИЛИ» не должно превосходить k.

Ключевая идея задачи в том, что разбивать пары индексов на множества, в которых $i \mid j = t$ для всех $t < k$, может быть достаточно нетривиально, а вот посчитать ответ для множеств, в которых $i \mid k \subset k$, то есть $i \subset k$ и $j \subset k$ (как множества единичных бит), уже проще. А $\bigcup\limits_{t \leqslant k} \{(i, j) : i, j \subset t\} = \bigcup\limits_{t \leqslant k} \{(i, j) : i \mid j = t\}$. Ведь если побитовое «ИЛИ» двух чисел меньше k, то каждое из них тоже является подмаской некоторого числа от 0 до k.

А вот для каждого $t < k$ посчитать максимальную сумму двух элементов массива, побитовое «ИЛИ» индексов которых является подмаской t, уже не очень сложно с помощью динамического программирования. Обозначим за $\texttt{dp}[t]$ пару максимальных элементов массива с индексами $i, j \subset t$, тогда

$$\texttt{dp}[t] = \text{two\_max}\left(a_t, \bigcup_{b \in t} \texttt{dp}[t - 2^b]\right),$$

то есть объединение всех ответов для t без произвольного бита, добавить a_t, и оставить только два максимальных элемента.

Такую динамику можно посчитать за $\mathcal{O}(2^n \cdot n)$, после чего для каждого t достаточно посчитать сумму найденных двух максимумов, а по полученным величинам посчитать префиксные суммы — k-я из них и будет ответом.