Problem A. Another Brick in the Wall

Идея: Георгий Корнеев
Разработка: Георгий Корнеев

Если l нечетно, то в каждом ряду кирпичей как минимум один должен быть 1×3, и этого достаточно. В этом случае ответ h.

Если l четно, то в рядах поочерёдно нужно делать либо все кирпичи 1×2, либо два крайних кирпича 1×3. Нам потребуется как минимум $\lfloor \frac{h}{2} \rfloor$ рядов с двумя кирпичами 1×3. Ответ в этом случае будет $2\lfloor \frac{h}{2} \rfloor$.

Problem B. Brick in the Wall, Part 2

Идея: Георгий Корнеев
Разработка: Михаил Первеев

Научимся решать задачу при условии, что построенная стена должна быть горизонтальной. Далее необходимо запустить полученное решение для исходной матрицы, а также для транспонированной матрицы, чтобы учесть вертикальные стены, и выбрать наилучший из двух ответов.

Пусть мы можем строить только горизонтальные стены. Выделим в каждой строке множество наибольших по включению отрезков, состоящих только из свободных клеток. Также создадим для стартовой и финишной клеток отрезки длины 1. Теперь построим граф, в котором вершинами будут выделенные отрезки. Соединим две вершины ребром, если в отрезках, соответствующих этим вершинам, существуют две клетки, которые являются соседними по стороне.

Теперь научимся определять, можно ли построить одну стену таким образом, чтобы не существовало ни одного пути из стартовой клетки в финишную. Так как на этом этапе мы не стремимся минимизировать длину стены, будем строить стену, которая полностью перекроет один из выделенных ранее отрезков. В терминах построенного графа это означает, что некоторая вершина будет удалена, так как теперь соответствующий ей отрезок будет заполнен заблокированными клетками, а значит перемещаться по ним будет нельзя.

Мы свели задачу к классической задаче: «можно ли удалить одну вершину в графе таким образом, чтобы не существовало ни одного пути из вершины s в вершину t». Для решения этой задачи выделим компоненты вершинной двусвязности и построим Block-Cut Tree. Теперь легко заметить, что удалить одну вершину требуемым образом возможно тогда и только тогда, когда в дереве на пути строго между стартовой и финишной вершиной есть хотя бы одна точка сочленения.

Вернемся к исходной задаче и поймем, как минимизировать длину построенной стены. Переберем все точки сочленения, лежащие на пути между стартовой и финишной вершинами в дереве, и для каждой из них решим задачу независимо.

Рассмотрим некоторую точку сочленения v, а также две инцидентные ей компоненты вершинной двусвязности c_1 и c_2, которые лежат на рассмотренном пути в дереве. Рассмотрим некоторое ребро в графе (u, w). Скажем, что $seg(u, w)$ — это отрезок столбцов, в которых отрезки строк таблицы, соответствующие вершинам u и w, «соединяются». Иными словами, $seg(u, w) = [l, r]$, где $[l, r]$ — отрезок столбцов, принадлежащих обоим отрезкам, соответствующим вершинам u и w.

Определим отрезок $[l_{c_1}, r_{c_1}]$ следующим образом:

$$l_{c_1} = \min_{(v,u) \in c_1} seg(v, u).l$$

$$r_{c_1} = \max_{(v,u) \in c_1} seg(v, u).r$$

Аналогично определим отрезок $[l_{c_2}, r_{c_2}]$, рассмотрев ребра, принадлежащие компоненте c_2. Нетрудно заметить, что в качестве ответа достаточно построить стену, покрывающую отрезок столбцов $[l_{c_1}, r_{c_1}]$, или стену, покрывающую отрезок столбцов $[l_{c_2}, r_{c_2}]$, так как в каждом из этих случаев будут заблокированы все клетки, по которым можно из вершины v перейти в компоненту c_1 или в

компоненту c_2. Однако, данные способы не всегда являются оптимальными. Поэтому далее необходимо рассмотреть случаи, когда оба отрезка не перекрываются полностью.

Пусть $L = \max(l_{c_1}, l_{c_2})$, а $R = \min(r_{c_1}, r_{c_2})$. Нетрудно заметить, что левая граница перекрываемого отрезка должна быть не правее L, а правая граница перекрываемого отрезка — не левее R. Если это не так, то всегда будет существовать клетка, через которую можно напрямую перейти из c_1 в c_2. Однако, перекрыть отрезок $[L, R]$ не всегда достаточно, так как могут существовать ребра, инцидентные вершине v, используя которые, можно «перепрыгнуть» заблокированный отрезок клеток и переместиться из c_1 в c_2.

Пусть помимо компонент c_1 и c_2 существуют компоненты $d_1, d_2, \ldots, d_k$, инцидентные вершине v. Для каждой из этих компонент вычислим отрезок $[l_{d_i}, r_{d_i}]$ аналогично отрезкам, рассмотренным ранее. Теперь заметим, что для того, чтобы отрезок $[L, R]$ нельзя было «перескочить», используя компоненту d_i, необходимо и достаточно, чтобы левая граница перекрытого отрезка была не правее, чем l_{d_i}, или правая граница перекрытого отрезка была не левее r_{d_i}.

Таким образом, если перекрываемый отрезок обозначить как $[l_{ans}, r_{ans}]$, мы получаем следующий набор ограничений:

- $l_{ans} \le L$;
- $r_{ans} \ge R$;
- Для любого $i \in [1, k]$ $l_{ans} \le l_{d_i}$ или $r_{ans} \ge r_{d_i}$.

Переберем все способы выбрать левую границу l_{ans}, и для каждого варианта найдем минимальную границу r_{ans}, такую что отрезок $[l_{ans}, r_{ans}]$ будет удовлетворять всем ограничениям. Для этого будем перебирать l_{ans} от начала отрезка, соответствующего вершине v до L и поддерживать минимально возможную правую границу r_{ans}. При движении левой границы направо для некоторых d_i условие $l_{ans} \le l_{d_i}$ перестает выполняться, поэтому для этих d_i необходимым становится условие $r_{ans} \ge r_{d_i}$, что не уменьшает минимально возможную правую границу ответа. Таким образом поддерживать правую границу r_{ans} при увеличении r_{ans} можно при помощи метода двух указателей, предварительно отсортировав отрезки $[l_{d_i}, r_{d_i}]$.

Суммарно нахождение ответа для всех точек сочленения v работает за $\mathcal{O}(nm \log m)$, так как отрезки, соответствующие вершинам, не пересекаются. Таким образом, все решение работает за $\mathcal{O}(nm \log m + nm \log n)$, то есть за $\mathcal{O}(nm \log nm)$.

Часть решения, в которой выполняется сортировка отрезков $[l_{d_i}, r_{d_i}]$, можно реализовать аккуратнее, получив время работы $\mathcal{O}(nm)$. Впрочем, для решения задачи это не требовалось.

Problem C. Capybara Cozy Carnival

Идея: Захар Яковлев
Разработка: Захар Яковлев

Про эту задачу удобнее думать в терминах графа: дан цикл размера n и m непересекающихся ребер-хорд. Необходимо покрасить вершины в k цветов так, чтобы никакие две вершины, соединенные ребром, не были одного цвета.

Заметим, что цикл эквивалентен отрезку с проведенным ребром из 1 в n, поэтому далее будем рассматривать задачу на отрезке.

Решим для начала задачу о количестве способов раскрасить “бамбук” без исходящих ребер. Для этого зафиксируем цвета на его концах. При этом вся информация задается двумя различными числами: количеством способов раскрасить отрезок так, чтобы оба конца были одинаковыми или различными. При рассмотрении двух последовательных отрезков, граничащих по вершине, достаточно перебрать цвет вершины на границе этих отрезков: он может совпадать с цветом одного из

оставшихся концов отрезков или иметь новый цвет. При этом сам по себе цвет промежуточной вершины значения не имеет, а важно только количество возможных цветов. Тогда количество способов покрасить вершины пути из ребер можно посчитать с помощью двоичных подъемов за $O(\log n)$.

Теперь осталось учесть наличие хорд. Для этого заметим, что замыкание пути в цикл приводит лишь к обнулению количества способов раскраски, при которой два противоположных конца имеют равный цвет.

Оказывается, описанных выше операций достаточно для полного решения задачи. Действительно, будем рассматривать непрерывные отрезки в порядке вложенности от меньшего к большему. Тогда мы последовательно будем вычислять количество способов покраски для пути из ребер, замыкания в цикл и объединения двух последовательных отрезков. При этом при рассмотрении в таком порядке в любой момент все ребра на текущем отрезке будут приняты во внимание, и не будет исходящих наружу ребер.

Осталось домножить ответ на количество способов выбрать цвета вершины 1 и вершины n: $k(k-1)$.

Асимптотика работы алгоритма — $O(m \log m)$ операций для сортировки отрезков и $O(m \log n)$ для вычисления количества способов раскраски путей. Общая сложность работы алгоритма $O(m \log m + m \log n)$.

Problem D. Defective Script

Идея: Геннадий Короткевич
Разработка: Никита Голиков

Мы легко можем сделать нагрузку равной 0 на всех серверах, поэтому дальше при поиске ответа лучшего, чем 0, будем считать, что нагрузка на каждом сервере всегда должна оставаться строго положительной.

Заметим, что применив операцию к каждому серверу, мы понизим нагрузку каждого сервера на 3. Таким образом, если существует ответ, в котором итоговая нагрузка каждого сервера равна x, то существует и ответ, в котором итоговая нагрузка равна $x \bmod 3$ (или 3, если x делится на 3). Решим задачу для каждого остатка по модулю 3 независимо и выберем максимальный ответ.

Пусть мы хотим, чтобы в итоге все нагрузки равнялись r ($1 \le r \le 3$). Обозначим за x_i количество операций, проделанных над сервером i. Тогда итоговая нагрузка на сервере i равняется $a_i - 2x_i - x_{i+1}$, если $i < n$, иначе $a_n - 2x_n - x_1$. Каждая итоговая нагрузка должна равняться r, что приводит нас к системе уравнений.

Перепишем уравнения следующим образом: $x_{i+1} = a_i - 2x_i - r$. Это позволяет нам выразить x_{i+1} через x_i. Выразим все x_i через x_1 следующим образом: скажем, что $x_i = k_i x_1 + b_i$. Для $i = 1$ получаем $k_1 = 1, b_1 = 0$, для $i \ge 2$ подставим выражение для x_{i-1} в уравнение для x_i. Теперь, используя уравнение для x_1, мы получаем, что $x_1 = a_1 - 2k_n x_1 - 2b_n - r$, из чего можно выразить x_1 как $\frac{a_1 - 2b_n - r}{2k_n + 1}$. Используя найденное x_1, можно вычислить все остальные x_i.

Для того чтобы найти максимальную возможную нагрузку для данного остатка, сделаем следующее: найдя x_i, проверим, что все итоговые значения действительно равны r. Если это так, то для нахождения максимальной нагрузки поступим следующим образом: пусть минимальное x_i равняется m, тогда, вычтя из всех x_i число m, мы получим корректный ответ. Тогда максимальное значение нагрузки для данного остатка равняется $r + 3m$.

В данном решении возникает проблема с переполнением, так как коэффициенты k_i и b_i становятся слишком большими. Чтобы решить эту проблему, поступим следующим образом: заметим, что все x_i целые, неотрицательные и не превосходят 10^9, тогда решим систему над полем остатков по модулю $10^9 + 7$. Заметим, что $k_i = (-2)^{i-1}$, и для $i \le 2 \cdot 10^5$ это значение никогда не равно -1 по модулю $10^9 + 7$, поэтому деление на $2k_n + 1$ всегда корректно. Получаем решение за $O(n)$.

Problem E. Eight-Shaped Figures

Идея: Геннадий Короткевич
Разработка: Геннадий Короткевич

Для решения задачи воспользуемся методом сканирующей прямой. Наша прямая будет вертикальной и идти слева направо. Разделим каждую окружность на две половины — верхнюю и нижнюю. В процессе сканирования будем поддерживать упорядоченное сверху вниз множество дуг, пересекаемых текущей прямой.

Заметим, что поскольку никакие две окружности не пересекаются по двум точкам, любые две дуги, активные на одной и той же вертикальной прямой, всегда будут упорядочены одинаково. Следовательно, нужно только научиться вставлять в множество две половины окружности i, когда сканирующая прямая достигнет координаты $x_i - r_i$, и удалять их, когда прямая достигнет координаты $x_i + r_i$.

Для эффективной обработки операций вставки и удаления в качестве упорядоченного множества следует использовать сбалансированное дерево поиска. В частности, в языке C++ можно использовать `std::set`, если самостоятельно определить компаратор для дуг.

Поскольку окружности и дуги могут касаться, требуется аккуратность в сравнении дуг между собой.

Как метод сканирующей прямой поможет нам найти ответ на задачу? Заметим, что если две окружности касаются (внутренним или внешним образом), то их соответствующие дуги в какой-то момент окажутся соседними в множестве. Давайте в каждый момент, когда какие-то две дуги окажутся соседними в множестве, проверим, не касаются ли соответствующие окружности, и если касаются, сохраним их точку касания и номера окружностей, касающихся в этой точке.

После окончания процесса для каждой точки касания посчитаем, сколько окружностей касаются эту точку с одной и с другой стороны, и добавим к ответу произведение этих двух чисел.

Чтобы избежать проблем с точностью вычислений, можно хранить все точки касаний в рациональных координатах — обе координаты точек касания могут быть представлены в виде дробей с целыми числителями порядка 10^{18} и знаменателями порядка 10^9.

Так как добавление и удаление в упорядоченное множество можно выполнить за $O(\log n)$, сложность решения составит $O(n \log n)$.

Следует обратить внимание, что, по аналогии со вторым примером, можно построить тест с ответом 10^{10}, не влезающим в 32-битный целочисленный тип (10^5 окружностей, касающихся внутренним образом в одной и той же точке слева, и 10^5 таких же окружностей справа), поэтому решения, подсчитывающие все подходящие пары окружностей по одной, недостаточно эффективны.

Problem F. False Alarm

Идея: Геннадий Короткевич
Разработка: Геннадий Короткевич

Переведём время в минуты. Пусть a_i — число минут от 7:00, на которое установлен i-й будильник ($0 \le a_1 < a_2 < \ldots < a_n \le 120$).

Ответ равен 0, если для некоторого i верно, что $a_{i+2} - a_i \le 10$. Это значит, что будильники i, $i+1$ и $i+2$ находятся на расстоянии не больше 10 минут и подходят под условие.

Ответ равен 1, если для некоторого i верно, что $a_{i+1} - a_i \le 10$. Это значит, что к будильникам i и $i+1$ можно добавить третий так, чтобы условие выполнялось.

В противном случае ответ равен 2 — можно взять любой имеющийся будильник и добавить два новых на достаточно небольшом расстоянии от него.

Problem G. Game of Annihilation

Идея: Михаил Иванов
Разработка: Михаил Иванов

Если у обоих игроков поровну фишек, то сразу заключаем, что исход — ничья. Допустим, у них не поровну фишек; назовём убегающим того игрока, у которого меньше фишек (и все его фишки будем иногда называть убегающими), и атакующим того игрока, у которого больше (и его фишки будем называть атакующими). Ясно, что исходом игры будет либо ничья, либо победа атакующего игрока. Основная идея решения заключается в том, что у убегающего есть лишь один шанс на ничью: добиться того, чтобы у него появилась фишка, которая правее всех фишек противника, к моменту начала его хода. Тогда убегающий просто ею сбежит в сторону $+\infty$.

Оказывается, критерий, что убегающий может добиться ничьей, такой. Пусть у убегающего игрока n фишек, у атакующего игрока m фишек, $m > n$. Отсортируем все фишки убегающего игрока по убыванию координаты, пусть их координаты $e_1 \geqslant e_2 \geqslant \ldots \geqslant e_n$, а у атакующего — $a_1 \geqslant a_2 \geqslant \ldots \geqslant a_m$. Пусть $E_i = \sum_{j=1}^{i} e_j$, $A_i = \sum_{j=1}^{i} a_j$. Критерий таков: если сейчас ход убегающего игрока, он может добиться ничьей, если и только если существует i от 1 до n, для которого $E_i \geqslant 1 + A_i$. Если сейчас ход атакующего, то убегающий может добиться ничьей, если и только если существует i от 1 до n, для которого $E_i \geqslant 2 + A_i$.

Стратегия за убегающего, если по критерию он может добиться ничьей: взять самое маленькое i, для которого $E_i \geqslant 1 + A_i$, и подвинуть вправо любую фишку с индексом от 1 до i (если отсчитывать фишки справа налево). Легко доказать, что если ход не привёл к аннигиляции, то выполнен критерий ничьей для i ($E_i \geqslant 2 + A_i$), а если привёл к аннигиляции, то для $i - 1$ ($E_{i-1} \geqslant 2 + A_{i-1}$). Действительно:

- если взаимно уничтожились две фишки с индексами, не превосходящими i (из всех атакующих фишек при отсчёте справа налево и из всех убегающих фишек при отсчёте справа налево), то сначала к сумме E_i добавили единицу, а потом из обеих сумм E_i и A_i выкинули равные слагаемые, отчего разность между ними не поменялась, но они стали E_{i-1} и A_{i-1} в новой конфигурации;
- если взаимно уничтожились убегающая фишка с индексом, не превосходящим i, и атакующая фишка с индексом $j > i$, то можно заключить, что $e_i < a_j \leqslant a_i$, что значит, что $E_i - A_i < E_{i-1} - A_{i-1}$, поэтому i не было наименьшим индексом, для которого выполнен критерий ничьей.

Аналогично, если выполнено $E_i \geqslant 2 + A_i$ для какого-то i и сейчас ход атакующего игрока, то он ничего с этим не поделает: рассмотрим наименьшее i, для которого $E_i \geqslant 2 + A_i$, тогда после хода атакующего игрока будет выполнено $E'_i \geqslant 1 + A'_i$, если не произошло аннигиляции (или аннигиляция не затронула правые i фишек убегающего игрока), и $E'_{i-1} \geqslant 1 + A'_{i-1}$, если атакующий игрок аннигилировал одну из правых i убегающих фишек.

Если же по критерию атакующий игрок побеждает, то его стратегия такая: если не существует двух атакующих фишек, между которыми расположена убегающая, то атакующий игрок может как угодно наступать своими фишками налево, пока не погубит все убегающие фишки. В противном случае он должен взять самую правую атакующую фишку, справа от которой есть убегающая фишка, и её-то и подвинуть вправо. Нетрудно понять, что если он ходом не произведёт аннигиляции, то инвариант сохранится, а если произведёт аннигиляцию, и инвариант нарушится, то он и до хода нарушался. Убегающий игрок ничего не сможет сделать во время своего хода, если по критерию атакующий побеждает, доказательство аналогично.

Problem H. Hanoi Towers Reloaded

Идея: Артем Васильев
Разработка: Артем Васильев

Решим частный случай: переложить все n дисков со стержня 1 на стержень 3, что в терминах задачи соответствует состояниям `1 1 ... 1` и `3 3 ... 3`. Обозначим нужное число ходов за $f(n)$. Аналогично классической задаче про Ханойские башни, приведем рекурсивный алгоритм. Для нуля дисков нужно ноль ходов: $f(0) = 0$.

Для n дисков рассмотрим, как мы перемещаем самый большой: сначала $1 \to 2$, затем $2 \to 3$. Для хода $1 \to 2$ необходимо, чтобы все остальные диски были на стержне 3, а для хода $2 \to 3$ необходимо, чтобы все остальные диски были на стержне 1. Получаем следующую последовательность действий для n дисков:

- Переложить диски $1 \dots n-1$ со стержня 1 на стержень 3, рекурсивно за $f(n-1)$ шагов;
- Переложить диск n с 1 на 2;
- Переложить диски $1 \dots n-1$ с 3 на 1, рекурсивно за $f(n-1)$ шагов;
- Переложить диск n с 2 на 3;
- Переложить диски $1 \dots n-1$ с 1 на 3, рекурсивно за $f(n-1)$ шагов.

Всего шагов получается $f(n) = f(n-1)+1+f(n-1)+1+f(n-1) = 3f(n-1)+2$. Решением этого рекуррентного соотношения является $f(n) = 3^n - 1$.

Заметим, что в этой последовательности нет повторяющихся состояний. Всего различных состояний в этой задаче 3^n: каждый из дисков может быть на одном из трех стержней, порядок дисков на стержне фиксирован. Также, из каждого состояния (кроме `1 1 ... 1` и `3 3 ... 3`) существует ровно два возможных хода: если самый маленький диск находится на стержне 2, его можно переложить на 1 или 3, а если он лежит с краю, то между оставшимися двумя стержнями можно сделать один ход.

Из всех утверждений, описанных выше, можно сделать вывод, что граф переходов между состояниями — гамильтонов путь: все состояния лежат на одной цепочке из `1 1 ... 1` в `3 3 ... 3`, и других переходов нет. Для того, чтобы найти расстояние между двумя произвольными состояниями, найдем позицию каждого из них в этой цепочке и посчитаем разность.

Рекурсивный процесс построения этой цепочки напоминает перечисление чисел в троичной системе счисления: сначала идут все состояния, в которых самый большой диск лежит на стержне 1, потом на 2, и в конце, на 3, что можно соотнести с числами, у которых старший разряд равен 0, 1 или 2. Отличие заключается в том, что если самый большой диск лежит на стержне 2, то все младшие разряды нужно инвертировать: $0 \to 2$, $1 \to 1$, $2 \to 0$. Продолжая этот процесс со всеми дисками от самого большого к самому маленькому, мы можем получить номер состояния в цепочке в троичной системе от 0 до $3^n - 1$.

Переведя два входных состояния в их номер в троичной системе счисления, необходимо их сравнить лексикографически и вычесть меньшее из большего по модулю 998 244 353.

Problem I. If I Could Turn Back Time

Идея: Георгий Корнеев
Разработка: Геннадий Короткевич

Поскольку порядок гор не имеет значения, переставим их по неубыванию текущей высоты (а при равенстве — по неубыванию старой высоты), то есть теперь, не умаляя общности, будем считать, что $h_1 \le h_2 \le \ldots \le h_n$.

Заметим, что все преобразования высот гор монотонны — а именно, если гора A раньше была выше горы B, то она никаким образом не может стать ниже горы B в результате эрозии. Значит, нужно

проверить, что выполнено условие $p_1 \le p_2 \le \ldots \le p_n$, в противном случае можно вывести -1 и перейти к следующему тесту.

Пусть снова гора A выше горы B. Тогда, если в каком-то году гора B оказалась подвержена эрозии и уменьшилась в высоте, то гора A тоже должна была быть ей подвержена (а обратное — не всегда верно). Значит, чем выше гора, тем сильнее она должна уменьшиться в размере. Формально, это условие можно записать как $p_1 - h_1 \le p_2 - h_2 \le \ldots \le p_n - h_n$ — если условие не выполнено, можно также вывести -1.

Наконец, если условия выше выполнены, можно показать, что всегда можно найти такую последовательность действий, которая приведёт высоты $p_1, p_2, \ldots, p_n$ в высоты $h_1, h_2, \ldots, h_n$. Поскольку любое действие, влияющее на горы, всегда уменьшает самую высокую гору, минимальное число таких действий (лет) равно $p_n - h_n$ — это и будет ответом на задачу.

Problem J. Just Half is Enough

Идея: Артем Васильев
Разработка: Геннадий Короткевич

Найдём k — число рёбер, удовлетворяющих условию $u_i < v_i$. Если $k \ge \lceil \frac{m}{2} \rceil$, выведем перестановку $1, 2, \ldots, n$. В противном случае, выведем перестановку $n, n-1, \ldots, 1$.

Поскольку каждое ребро удовлетворяет либо условию $u_i < v_i$, либо условию $u_i > v_i$, в сумме перестановки $1, 2, \ldots, n$ и $n, n-1, \ldots, 1$ имеют ровно m подходящих рёбер. А значит, хотя бы одна из них будет иметь хотя бы половину от m.

Так как это верно для любой пары перестановок $p_1, p_2, \ldots, p_n$ и $p_n, p_{n-1}, \ldots, p_1$, в качестве альтернативного решения можно генерировать случайные перестановки, пока не найдём ту, которая сработает. В среднем понадобится не более 2 итераций.

Problem K. Keyboard Chaos

Идея: Сергей Цаплин
Разработка: Сергей Цаплин

Основная идея

Если ответ существует, то он состоит из одинаковых букв.

Докажем это от противного. Пусть самая короткая строка имеет длину m. Будем считать, что эта строка заканчивается на $b\underbrace{a \cdots a}_{1 \le k < m}$.

Рассмотрим любой способ напечатать её префикс до последней буквы ‘b’ включительно. После этого префикса мы не можем напечатать k букв ‘a’. Это значит, что:

1. Не существует клавиши, содержащей только буквы ‘a’;
2. Если для каждой клавиши вычислить, сколько следующих букв будут ‘a’, то сумма получится меньше, чем k.

Заметим, что если удалить последнюю букву ‘b’ и отменить её печать, то клавиша, которой она была напечатана, уже не сможет использоваться для печати букв ‘a’, а состояние остальных клавиш не изменится. Поэтому сумма из последнего пункта не уменьшится и продолжит быть меньше k, то есть можно вычеркнуть последнюю букву ‘b’ и получить строку длины $m-1$, которую тоже нельзя напечатать.

Получили противоречие. Значит, если ответ существует, то он состоит из одинаковых букв.

Вычисление ответа

Если для каждой буквы существует клавиша, содержащая только эти буквы, то такая клавиатура будет функционировать, как обычная, и на ней можно будет напечатать любую строку.

Иначе для каждой буквы, для которой нет клавиши, содержащей только эту букву, необходимо перебрать все кнопки, нажимать их, пока текущая буква на них не станет отличаться от этой буквы, и запомнить суммарное количество нажатий.

Длина ответа будет на 1 больше минимального количества нажатий по всем таким буквам.

Например, для $n = 4$, $e = 3$ и кнопок с буквами "`abc`", "`bb`", "`ccccab`" и "`aba`":

- Для буквы '`a`' 1-ю и 4-ю кнопку необходимо нажать по одному разу (т.е. сделать 2 нажатия, чтобы эта буква перестала быть текущей на всех кнопках);
- Для буквы '`b`' есть 2-я кнопка, состоящая только их этих букв, то есть строка из букв '`b`' не может быть ответом;
- Для буквы '`c`' необходимо нажимать только 3-ю кнопку, но сделать это 4 раза.

Длина ответа будет $min(2, 4) + 1 = 3$, и единственным правильным ответом будет строка "`aaa`".

Problem L. Longest Common Substring

Идея: Андрей Станкевич
Разработка: Геннадий Короткевич

Для решения этой задачи нужно придумать, как сжать информацию о каждой строке так, чтобы можно было легко проверить, что некоторая строка w действительно является наибольшей общей подстрокой s и t.

Оказывается, такое сжатие несложно выполнить. Рассмотрим все возможные 2^{k+1} битовых строк длины $k + 1$ и выпишем битовую маску $b(s)$ из 2^{k+1} бит, обозначающую, какие из них являются подстроками s. Аналогично определим $b(t)$. Тогда, если у $b(s)$ и $b(t)$ есть общие биты, то у s и t есть общая подстрока длины $k+1$, а значит, строка w длины k не может являться их наибольшей общей подстрокой. В противном случае достаточно проверить, что w входит в s и в t — а это верно, если хотя бы одна из строк w`0`, w`1`, `0`w или `1`w (длины $k + 1$) входит в $b(s)$, и то же с $b(t)$.

Теперь воспользуемся методом динамического программирования. Пусть $f(n, p, b)$ — число строк длины n, последние k бит которых равны p, а маска встреченных подстрок длины $k+1$ равна b. Для переходов "вперёд" переберём очередной $n + 1$-й бит (назовём его d), допишем d к строке p справа, добавим подстроку p в маску b, и отрежем левый бит от p.

Состояний и переходов у данной функции $O(n \cdot 2^k \cdot 2^{2^{k+1}})$, что не слишком много в заданных ограничениях.

В последней части решения — переборе масок $b(s)$ и $b(t)$ — можно перебирать только пары масок, не имеющих общих единичных битов. Таких пар ровно $3^{2^{k+1}}$, что тоже не слишком много.

Приведённый алгоритм работает только в случаях $n > k$ и $m > k$, поэтому случаи $n = k$ или $m = k$ надо рассмотреть отдельно.

Problem M. Misère

Идея: Андрей Лопатин, Геннадий Короткевич
Разработка: Андрей Лопатин

В каждой масти, если мы скидываем t карт, $0 \le t \le k$, мы можем посчитать, сколько карт нам нужно будет добрать, чтобы эта масть стала требуемого вида (естественно, что мы скидываем самые большие карты, а добираем — самые маленькие).

Применим метод динамического программирования: обозначим за $f(i, j)$ минимальное число карт, которое нужно будет добрать, если среди первых i мастей мы скинем j карт в сумме.

Для того, чтобы пересчитывать функцию, в очередной масти перебираем параметр t — количество скидываемых карт. Естественным образом рассматриваем только те масти, которые присутствуют

у нас на руках, остальные пропускаются и не участвуют в решении (можно рассматривать это, как сжатие координат).

Состояний $O(n^2)$, и суммарное число переходов тоже оценивается как $O(n^2)$ — а точнее, число переходов равно n, умноженному на сумму количеств вариантов t (которое, в свою очередь, равно $k+1$ для масти размера k).

Ответом на задачу является минимальное такое j, для которого $f(m, j) \leq j$ (где m — максимальная масть).