Задача A. Битва с боссами

Автор задачи и разработчик: Даниил Голов

Давайте формализуем условие задачи. Есть подвешенное дерево из n вершин и процесс, состоящий из двух чередующихся стадий — сначала из дерева убираются все вершины, являющиеся единственным ребенком своего родителя, а затем вершина с самым большим числом детей становится вершиной с одним ребенком.

Соответственно,

- если в дереве осталась ровно одна вершина, то процесс заканчивается;
- если в дереве есть хотя бы одна вершина с единственным ребенком, этот ребенок будет удален («побежден»);
- если у каждой вершины хотя бы два ребенка, на второй стадии процесса появится хотя бы одна новая вершина с единственным ребенком (возвращаемся к первой стадии).

Таким образом, ситуация, когда Зельда не может победить никого из боссов, может возникнуть исключительно тогда, когда босс остался всего один.

Заметим, что корень дерева при любом перестроении останется корнем. Действительно, если у корня максимальное число детей, то выберется сын корня и на следующей стадии он же удалится. Если у корня не максимальное число детей, перестроений с ним не будет. Также заметим, что сам корень дерева никогда не удаляется, так как Зельда атакует только боссов, у которых есть предок.

Таким образом, всё, что требовалось в задаче — найти корень заданного дерева и вывести его номер. Для этого достаточно считать список детей каждой вершины и найти единственную вершину, которая не вошла ни в один из них.

Задача B. Оптимизация заклинаний

Автор задачи и разработчик: Егор Юлин

Для начала заметим, что в терминах строк требуется найти количество пар (s_i, s_j), для которых $\texttt{lcp}(s_i, s_j)$ лежит в наборе t. Здесь за `lcp` обозначен наибольший общих префикс двух строк.

Данную задачу можно решать при помощи бора. Причем бор можно строить как на строках s_i, и затем обрабатывать t_i и считать ответ, так и наоборот.

Построим бор на строках второго набора (t), и в каждой вершине бора будем хранить $\texttt{term}_v$ — заканчивается ли в ней какой-то t_i, и $\texttt{cnt}_v$ — количество строк s_i, имеющих префикс v. Чтобы посчитать все `term`, достаточно запомнить их при построении, а чтобы посчитать `cnt`, будем по очереди обрабатывать s_i, проходить по бору и увеличивать это количество на 1 в каждой пройденной вершине.

Тогда обработка строки s_i по мере спуска в боре выглядит следующим образом: если в текущей вершине есть строка из второго набора, то к ответу прибавим $\texttt{cnt}_v - \texttt{cnt}_{v+c}$, где c — следующий символ s_i. Это количество соответствует числу строк s_j, которые начинаются на v, но не на $v + c$, то есть `lcp` которых с s_i в точности равен v. После этого перейдём в следующую вершину бора.

Такое решение работает за суммарную длину всех строк.

Задача C. Застывший Мир

Автор задачи: Даниил Голов, разработчик: Константин Бац

Суть задачи заключалось в том, чтобы найти центр некоторого круга неизвестного радиуса $2d$. Для этого у нас есть единственная операция: проверить, лежит ли некоторая точка (x, y) внутри искомого круга или нет.

Задачу можно решить, если найти три точки, лежащие на границе круга с центром в искомой точке. Проведем из точки $(0, 0)$ три луча в разных направлениях, которые пересекутся с границей круга. При этом, условие, что точка $(0, 0)$ находится на расстоянии не больше d от (x, y), гарантирует нам, что точки пересечения с границей круга будут различными.

Выбор направления лучей не имел значения. Например, можно было взять направления $(-1, 0)$, $(0, 1)$ и $(0, 1)$. Далее нужно найти точку пересечения каждого из лучей с границей круга. Для этого будем искать максимальное удаление от точки $(0, 0)$ в направлении луча, при котором она все

еще находится на расстоянии не больше $2d$ от центра круга. По условию задачи, такое удаление нужно искать на отрезке $[0, 10^6]$, а достаточная точность при бинарном поиске — 10^{-3} (если взять направления, параллельные осям координат).

Таким образом, для решения достаточно запустить три бинарных поиска в различных направлениях, а затем найти центр круга по трем точкам. Пусть (x_1, y_1), (x_1, y_1), (x_1, y_1) — найденные точки на окружности, а (x, y) — центр круга. Тогда, если обозначить $m_1 = \frac{y_2-y_1}{x_2-x_1}$, и $m_2 = \frac{y_3-y_2}{x_3-x_2}$, то можно построить следующую систему уравнений

$$\begin{cases} y = -\frac{1}{m_1} \cdot \left(x - \frac{x_1+x_2}{2}\right) + \frac{y_1+y_2}{2} \\ y = -\frac{1}{m_2} \cdot \left(x - \frac{x_2+x_3}{2}\right) + \frac{y_2+y_3}{2} \end{cases}$$

Для решения системы уравнений выразим x следующим образом, а затем с его помощью найдем y.

$$x = \frac{m1 \cdot m2 \cdot (y_1 - y_3) + m2 \cdot (x_1 + x_2) - m1 \cdot (x_2 + x_3)}{2 \cdot (m2 - m1)}.$$

Задача D. Простая игра

Автор задачи и разработчик: Павел Скобелин

В игре выигрывает тот, после чьего хода сумма чисел становится простым числом. Давайте обозначим: $s = A + B$. Тогда, если мы уменьшаем/увеличиваем числа A/B на какие-то значения, то s изменится на такое же значение. Обозначим за l наибольшее простое число, меньшее s, а за r — наименьшее простое число, большее s. Заметим, что в пределах игры (кроме, может быть, последнего хода) сумма текущих A и B будет лежать в пределах от l до r. Действительно, если в какой-то момент игрок «перепрыгнул» l или r, то он мог этим ходом выиграть.

Как найти l и r? Очень просто, для этого нужно понять факт, что l и r — соседние простые числа, а расстояние между соседними простыми числами до 10^9 **не превосходит** 300 (эмпирическое наблюдение). Тогда можно найти эти границы наивно, проверяя числа на простоту проверкой делителей до корня. Эта часть будет работать за $\mathcal{O}(\sqrt{A+B} \cdot 300)$.

В начале стоит проверить, что Зельда сможет выиграть за один ход. В каком случае она может это сделать? Она может или прибавить что-то к числу B, в таком случае должно выполняться неравенство $s + k \geqslant r$. Также она может вычесть что-то из числа A, в таком случае должно выполняться $s - k \leqslant l$, но в этом случае нужно не забыть про то, что A должно оставаться натуральным: то есть $A - (s - l) \geqslant 1$.

Если Зельда не может выиграть за один ход, подумаем про выигрышные и проигрышные позиции. Выигрышной суммой назовем такую, что, начиная с такой суммы, игрок может выиграть, проигрышной суммой — не выигрышную. В таком случае можем считать r проигрышной суммой. Тогда все суммы в промежутке $[r - k, r)$ — выигрышные. Но тогда сумма $r - (k + 1)$ — проигрышная. Рассуждая так далее, несложно понять, что проигрышными суммами являются только суммы вида $r - (k + 1) \cdot z, z \in \mathbb{N}$ (удобно представить в виде таблицы, в которой помечать выигрышные и проигрышные позиции). Тогда, если $(r - s) \bmod (k + 1) = 0$, то Зельда не сможет выиграть, иначе сможет.

Итого получаем решение за $\mathcal{O}(\sqrt{A+B} \cdot 300)$ на один набор входных данных.

Задача E. Чудеса природы

Автор задачи: Егор Юлин, разработчик: Константин Бац

Для решения этой задачи вспомним, как выглядит динамика для поиска наибольшей общей подпоследовательности:

$$\mathtt{dp}_i = \max_{i<j, a_i<a_j} \mathtt{dp}_j + 1,$$

где $\mathtt{dp}_i$ соответствует длине НВП, заканчивающейся в i-м элементе.

От нас же по задаче требуется найти, во-первых, не длину максимальной подпоследовательности, а их количество (стандартное изменение пересчета динамики с $\max(\ldots)$ на $\mathrm{sum}(\ldots)$), а во-вторых, не просто возрастающих последовательностей, а сначала возрастающих, затем убывающих.

Стандартная идея, применимая в таких ситуациях — разбить искомые последовательности по «центральному» элементу. Заметим, что если перебирать a_i, слева от которого подпоследовательность возрастает, а справа — убывает, то ответ на задачу равен

$$\sum_{i=1}^{n} \#\texttt{lis}_i \cdot \#\texttt{lds}_i,$$

где $\#\texttt{lis}_i$ — число возрастающих подпоследовательностей, заканчивающихся в a_i, а $\#\texttt{lds}_i$ — число убывающих подпоследовательностей, начинающихся в a_i.

Более того, заметим, что если мы научимся считать $\#\texttt{lis}$, то $\#\texttt{lds}$ можно просто посчитать как $\#\texttt{lis}$ на развернутом массиве a.

Ну а для подсчета $\#\texttt{lis}$ вспомним решение задачи о поиске НВП через дерево отрезков: будем обрабатывать элементы массива по очереди слева направо, а значения динамики $\texttt{dp}_i$ хранить по индексу a_i. Тогда все элементы с $i < j$ и $a_i < a_j$ — это просто префикс нашего дерева отрезков. Поскольку мы хотим находить число подпоследовательностей, а не максимальную из них, пересчет будет выглядеть как $\texttt{dp}_i = \sum\limits_{x<a_i} \texttt{segment\_tree}_x$, то есть как запрос к ДО на префиксе.

Всего на подсчет $\#\texttt{lis}$ и $\#\texttt{lds}$ уйдет $\mathcal{O}(n \log n)$ времени, после чего ответ можно посчитать за линейное время, перебрав центральный элемент холма (см. формулу выше).

Задача F. Закрыть порталы

Автор задачи и разработчик: Даниил Орешников

Это вариация классической задаче о дереве Штейнера — минимальном остовном дереве графа, соединяющем множество заранее заданных вершин. Для произвольного числа выбранных вершин эта задача является NP-трудной, однако данная задача сводится к задаче о дереве Штейнера для трех вершин.

Нам требуется выбрать некоторое множество вершин, чтобы по ним можно было добраться от любого из трех изначально заданных **связных** множеств вершин до другого. Обозначим изначальные множества за A_1, A_2 и A_3, а искомое множество связывающих их вершин — за B. Тогда заметим, что каждая из вершин B лежит на каком-то пути между A_i и A_j для $i \neq j$, иначе эту вершину можно удалить, и ответ станет меньше.

Посмотрим, как множество B может выглядеть.

1. Рассмотрим путь $p_{1,2} \subseteq B$ между A_1 и A_2, и аналогичные пути $p_{1,3}$ и $p_{2,3}$. Заметим, что $B = p_{1,2} \cup p_{1,3} \cup p_{2,3}$, все остальные вершины можно удалить.

2. Пусть какие-то два из этих путей, например, $p_{1,2}$ и $p_{1,3}$, не пересекаются. Тогда заметим, что существует путь от A_2 до A_3 через A_1, поэтому $B = p_{1,2} \cup p_{1,3}$.

3. Если же любые два выбранных пути пересекаются, то рассмотрим v — какую-то из вершин на пересечении $p_{1,2}$ и $p_{1,3}$. Заметим, что если оставить в B только $p_{1,2}$ и часть $p_{1,3}$ от v до A_3, то также все множества будут взаимодостижимы, поэтому $B = \bigcup\limits_{i=1}^{3} \texttt{path}(v, A_i)$.

Итого, B можно получить либо как объединение двух минимальных путей между двумя парами данных множеств, либо как объединение трех кратчайших путей от некоторой вершины v до каждого из множеств. Чтобы все эти кратчайшие пути вычислить, сделаем множественный $\texttt{bfs}$ из каждого из A_i и найдем $\texttt{d}_i(v)$ — длину кратчайшего пути от какой-то из вершины A_i до v.

Теперь если обозначить за $\texttt{D}_i(j)$ величину $\min\limits_{v \in A_j} \texttt{d}_i(v)$, ответ равен

$$\min\left(\texttt{D}_1(2) + \texttt{D}_1(2), \texttt{D}_2(1) + \texttt{D}_2(3), \texttt{D}_3(1) + \texttt{D}_3(2), \min_{v=1}^{n}(\texttt{d}_1(v) + \texttt{d}_2(v) + \texttt{d}_3(v) + 1)\right).$$

Только надо аккуратно учесть, что расстояния надо считать в вершинах, а не в ребрах.

Задача G. Лечебный смузи

Автор задачи и разработчик: Егор Юлин

Отметим некоторые факты, после чего предложим алгоритм построения массива, в котором будет максимальный ответ.

Если в массиве встречается число a_i, при этом оно уже было раньше, то ничего не изменится. Из-за этого все повторения чисел можно расположить в конце в любом порядке.

Обозначим $b_i = \texttt{mex}(a_1, \ldots, a_i)$. Заметим, что массив b не убывает. Кроме того, все повторения чисел мы переместили в конец, тогда рассмотрим массив a, в котором все числа различны. Чтобы получить максимальный $\mathrm{mex}(b)$, мы хотим получить массив b вида $(0, 1, 2, 3, \ldots)$.

Докажем, что если отсортировать массив a, после чего расположить его элементы как $(a_n, a_1, a_2, a_3, \ldots)$, то массив b получится оптимальным.

Действительно,

- первое число должно быть не равно 0, т.к. иначе $b_1 = 1$ и итоговый mex будет равен 0;
- чтобы на второй позиции в массиве b получить 1, нужно расположить где-то 0, но это не первая позиция, поэтому 0 должен стоять на второй позиции;
- и так далее;
- среди всех элементов на первой позиции должен стоять максимальный, потому что число на первой позиции в массиве a никогда не появится в массиве b.

Тогда можно отсортировать a, все неуникальные числа перенести в конец, f на первую позицию поставить максимум. После этого останется посчитать mex на префиксах (это сделается с помощью `std::map`), после этого посчитать итоговый mex. Время работы: $\mathcal{O}(n \log n)$.

Задача H. Как провести время

Разработчик: Даниил Орешников

Самое примитивное решение задачи заключается в эмуляции процесса, описанного в условии. К сожалению, количество действий, необходимое, чтобы завершить процесс, может быть слишком большим, поэтому требуется эту эмуляцию оптимизировать.

Для полного решения достаточно одного наблюдения: с каждым следующим шагом в последовательности становится все больше и больше одинаковых чисел. Пусть у нас x минимальных чисел и y максимальных, тогда

- если $x \leqslant y$, то через $2x$ шагов все минимальные числа превратятся в следующие по величине, а максимальных останется $y - x$ (x из них станут предыдущими по величине);
- если $x > y$, то через $2y$ шагов симметрично исчезнут все максимальные.

Таким образом, давайте поддерживать дек (`deque`) упорядоченных по величине чисел и количеств их вхождений в последовательность. За одно действие будем вынимать из дека минимальное и максимальное число, после чего массово обрабатывать $2\min(x, y)$ ходов.

Останется только аккуратно отследить момент, когда различных чисел становится не больше двух. Время работы такого решения пропорционально n.

Задача I. Фабрика эхо

Автор задачи: Даниил Орешников, разработчик: Павел Скобелин

Давайте воспользуемся декартовым деревом для решения задачи. Будем поддерживать декартово дерево по неявному ключу из текущих сегментов, каждый из которых также будет декартовым деревом по неявному ключу, в котором будем хранить сами элементы очереди в нужном порядке. Научимся обрабатывать запросы всех типов:

1. Чтобы добавить элемент в конец очереди, нужно достать последний сегмент из дерева и добавить в его конец новый элемент;

2. Заметим, что запросы удаления можно обрабатывать «наивно», ведь суммарно удалений произойдет не больше количества добавленных элементов. То есть для обработки этого запроса нужно пройти по всем сегментам и удалить первый элемент из каждого, после этого удалить пустые сегменты;

3. Чтобы разделить сегмент, достаточно разделить декартово дерево по неявному ключу для данного сегмента на два дерева, и вставить новый сегмент сразу за разделенным;

4. Запрос объединения всех сегментов также можно выполнять «наивно», ведь их суммарно будет не более, чем запросов разреза. То есть для обработки этого запроса просто объединим все сегменты в один.

Таким образом эту задачу можно решать с помощью вложенных ДД.

Также есть решение, не использующее декартово дерево или хотя бы два вложенных ДД. Давайте хранить все элементы очереди в обычном массиве и помечать уже использованные элементы.

Тогда сегменты можно хранить в виде встроенного дерева поиска (например, `ordered_set` в C++, или также в декартовом дереве, в данном решении оно будет одномерное). Каждый сегмент будет задаваться парой чисел: индекс его начала в текущем массиве и количество элементов в нем. Тогда решение остается таким же, но нужно по другому обрабатывать запрос номер три.

Теперь тяжело искать индекс начала второго сегмента, если мы разрежем сегмент после i-го элемента. Заметим, что тогда индекс начала нужного сегмента будет индексом $i + 1$-й единицы, начиная с индекса начала текущего сегмента. Этот запрос, например, можно обрабатывать техникой спуска по дереву отрезков. Тогда если поддерживать использованные элементы в дереве отрезков, то мы сможем эффективно отвечать на все запросы.

Задача J. Великий комбинатор Зельда

Автор задачи: Даниил Орешников, разработчик: Владимир Рябчун

Математическая модель данной задачи представляет из себя дерево размера n, где в каждой вершине записано число k_v (будем называть это цветом вершины). Заметим, что минимальное количество атак не изменится, если сжать все множества смежных вершин с одинаковым цветом. Более того, заметим, если убрать из условия «жадное» распространение атаки и позволить каждой атаке покрывать произвольное подмножество вершин двух цветов вместо максимального по включению, оптимальный ответ не изменится.

В сжатом графе все ребра соединяют вершины разных цветов. Подвесим граф за любую вершину чтобы у всех кроме корня был родитель. Множество детей вершины v будем обозначать $g[v]$, а родителя — p_v.

Теперь если мы хотим посчитать ответ для поддерева вершины v, можно рассмотреть два случая: v была уничтожена вместе с родителем или нет. Количество атак в первом случае будем обозначать как $\mathtt{dp}_1[v]$, а во втором — $\mathtt{dp}_2[v]$. Тогда имеют место следующие формулы:

$$\mathtt{dp}_1[v] = \sum_{u \in g[v], k_u = k_{p_v}} \min\left(\mathtt{dp}_1[u] - 1, \mathtt{dp}_2[u]\right) + \sum_{u \in g[v], k_u \neq k_{p_v}} \left(\mathtt{dp}_2[u]\right) + 1 \tag{1}$$

$$\mathtt{dp}_2[v] = \min\left(1 + \sum_{u \in g[v]} \mathtt{dp}_2[u], \min_{c \in g[v]} \left(1 + \sum_{u \in g[v], k_u = k_c} \left(\min(\mathtt{dp}_1[u] - 1, \mathtt{dp}_2[u])\right) + \sum_{u \in g[v], k_u \neq k_c} \left(\mathtt{dp}_2[u]\right)\right)\right) \tag{2}$$

Фактически в первом случае мы уже знаем, на каких детей перейдет атака (на тех, у которых цвет совпадает с цветом родителя текущей вершины), и выбираем лучший вариант у них, остальные дети считаются независимо через $\mathtt{dp}_2[u]$. Во втором случае можно сгруппировать всех детей v по цвету (за линейное время), и перебрать пару (k_v, k_{child}). Для каждого ребенка выбранного цвета для него можно взять как $\mathtt{dp}_1$, так и $\mathtt{dp}_2$ (выбираем оптимальное), для всех остальных мы обязаны выбрать $\mathtt{dp}_2$.

Задача K. Священное бревно

Автор задачи: Анна Антонова, разработчик: Владислав Власов

Решим данную задачу с помощью задачи о рюкзаке. Пусть $\texttt{dp}[i][j]$ равно `true`, если возможно удалить вершины суммарного веса j с помощью удалений вершин $1, 2, \ldots, i$, и `false` иначе.

Но дальше возникают сложности — если положить вес вершины равным ее весу из условия, такое решение не учтет требование на связность оставшегося множества, а если положить вес вершины равным весу всего ее поддерева, мы можем удалить какую-то вершину больше одного раза.

Положим вес каждой вершины равным сумме исходных весов вершин в ее поддереве. Тогда задача преобразуется в подсчет $\texttt{dp}[n][\texttt{total} - w]$ с учетом того, что если какая-то вершина была взята в рюкзак, то никакой ее потомок не может быть взят. Чтобы учесть это, выпишем вершины в специальном порядке: перед вершиной v должен идти непрерывный отрезок из всех ее потомков. Назовем массив с этим порядком `order` (на самом деле это просто развернутый Эйлеров обход дерева). Тогда положим $\texttt{dp}[i][j]$ равным `true`, если можно набрать вес j с помощью вершин $\texttt{order}_{1,\ldots,i}$.

Теперь к некоторому $\texttt{dp}[i][j]$ существует два перехода:

- из $\texttt{dp}[i-1][j]$ — если мы не берем вершину $\texttt{order}_i$;
- из $\texttt{dp}[i-\texttt{order}_i][j-\texttt{weight}_i]$ — если мы не берем вершину $\texttt{order}_i$, но тогда автоматически не берем всех ее потомков.

Если посчитать такую динамику, мы получим требуемый ответ. Время работы решения — $\mathcal{O}(nw)$.

Задача L. Очередная карточная игра

Автор задачи: Антон Вдовин, разработчик: Даниил Орешников

Для начала поймем, что за $2n$ запросов такая задача решается тривиально — первые n запросов потратим, чтобы по одному разу перевернуть каждую пару карточек с соседними номерами, после чего мы будем знать все их значения. И вторые n запросов потратим, чтобы переворачивать уже карточки с одинаковыми значениями.

Теперь осталось соптимизировать наше решение на один запрос. Для этого нам дана дополнительная информация: для двух карточек известно, значение на какой из них меньше, чем на другой. Будем делать следующее: как и в изначальном решении, будем переворачивать подряд идущие пары карточек и запоминать их значения. При этом

- если мы встречаем вторую карточку с каким-то значением, мы находим номер первой такой карточки, переворачиваем их обе и забываем про них;
- в процессе перебора карточек по очереди мы пропустим карточки a и b.

Что останется после такого прохода по последовательности карточек? Во-первых, мы потратили $n-1$ запрос на первый переворот каждой карточки, кроме a и b. Во-вторых, ко всем карточкам, кроме a и b, нашлась пара. Соответственно, осталось четыре карточки — a, b, c и d, и для c и d мы знаем их значения.

Сделаем последние два запроса — перевернем вместе a и минимальную по значению из c и d, а затем b и оставшуюся последнюю. Так как мы знаем, что все карточки разбиваются по значениям на пары одинаковых, очевидно, что за такие последние два действия мы перевернули две пары одинаковых. Всего было потрачено $2n-1$ запросов.