Задача A. Цепная реакция

Автор задачи и разработчик: Григорий Хлытин

Посмотрим на задачу под таким углом: узлы — вершины графа, связи — ребра, и требуется найти кратчайший путь из вершины u в вершину v, если некоторые ребра могут «закрываться» в определенные периоды времени.

Чтобы найти кратчайший путь во взвешенном графе, можно воспользоваться *алгоритмом Дейкстры*. Остается только учесть тот факт, что по некоторым ребрам нельзя перемещаться сразу же после попадания в их начало. Например, если первый момент времени, в который энергия добирается до некоторой вершины a, равен t, а ближайший период времени, в течение которого ребро $a \to b$ веса w (проход по которому занимает w времени) пропускает энергию, равен $[x, y]$, то энергия попадет в вершину b по этому ребру в момент времени $x + w$.

Давайте запустим алгоритм Дейкстры, но будем считать, что вес такого ребра равен не w, а $(x - a) + w$, то есть сумме времени ожидания до «открытия» ребра плюс его исходный вес. В таком случае после того, как такой алгоритм отработает, в каждой вершине будет отмечено кратчайшее расстояние от вершины u до нее.

Недостающая деталь — необходимость быстро находить для каждого такого ребра ближайший интервал времени, в который энергия может по нему перемещаться. Для этого, если алгоритм Дейкстры запоминает кратчайшие расстояния в массив `d`, достаточно в вершине a научиться быстро находить такой минимальный $i \leqslant k$, что $y_i \geqslant \mathtt{d}[a]$. Это можно было сделать двоичным поиском (с небольшой вероятностью это решение могло пройти), либо, заметив, что расстояния в алгоритме Дейкстры не уменьшаются, двигаться по i аналогом метода *двух указателей*.

Время работы такого решения — $\mathcal{O}(m \log n + k)$.

Задача B. Пилтовер и Заун

Автор задачи и разработчик: Даниил Орешников

Пусть изначально Пилтовер и Заун имели одинаковое целое значение *статуса* x. Тогда, если история верна, то $x_1 = x + \sum_{i=1}^{n} a_i$, а $x_2 = x - \sum_{i=1}^{n} a_i$. Обозначим сумму всех a_i за A.

По условию, любое $a_i \geqslant 1$, поэтому $A \geqslant n \geqslant 0$. Отсюда вытекает первое условие на x_1 и x_2: если история верна, то $x_1 \geqslant x_2$. С другой стороны, можно сказать, что $x_1 - x_2 = 2 \cdot A \geqslant 2n$. Поэтому должно выполняться $(x_1 - x_2) \bmod 2 = 0$ и $x_1 - x_2 \geqslant 2n$.

Таким образом, имеем три условия, необходимых для того, чтобы история была правдой:

1. $x_1 \geqslant x_2$
2. $(x_1 - x_2) \bmod 2 = 0$
3. $x_1 - x_2 \geqslant 2n$ (так что первое условие на самом деле избыточно, потому что $n \geqslant 0$)

Заметим, что указанные условия так же являются достаточными для существования последовательности a_i, такой, что $x_1 = x + \sum_{i=1}^{n} a_i$ и $x_2 = x - \sum_{i=1}^{n} a_i$. В частности, можно взять первые $n - 1$ значений a_i равными единице, а последнее — равное $\frac{x_1 - x_2}{2} - (n - 1)$.

Считать данные и проверить выполнение всех условий можно за $\mathcal{O}(1)$, поэтому и общее время работы программы будет $\mathcal{O}(1)$.

Задача C. Опасные игры

Автор задачи и разработчик: Даниил Голов

Будем решать задачу бинпоиском. Пойдем в одну из центральных клеток квадрата и спросим, где относительно нее находится искомая. Заметим, что результаты «`NW`», «`NE`», «`SW`», «`SE`» однозначно обозначают верхний левый, верхний левый, нижний левый и нижний правый квадраты от точки запроса, и можно сократить область для следующего запроса в 4 раза. Таким образом, достаточно поддерживать текущие левую, правую, нижнюю и верхнюю границы актуальной зоны поиска.

Результаты «N», «S», «W» и «E» смещают лишь только одну из данных границ вместо двух. Однако по ним можно однозначно понять, в каком столбце или какой строке находится искомая клетка, так что на самом деле область уменьшается намного больше.

Таким образом, при каждом запросе область поиска будет сокращаться в 4 раза, пока конечная точка поиска не будет локализована в одном столбце или одной строке, после чего зона поиска будет сокращаться в 2 раза. А тогда итоговая асимптотика составит $\mathcal{O}(\log_4(n^2)+\log_2(n)) = \mathcal{O}(\log_2 n)$. Для ограничений задачи количество запросов в любом случае не превысит 50.

Задача D. Тренировки миротворцев

Автор задачи и разработчик: Даниил Орешников

Рассмотрим какую-то последовательность действий, приводящую к площади $\frac{s}{2}$ за минимальное число шагов.

Заметим, что если какие-то два миротворца совершали движения в одну и ту же сторону, можно было бы добиться той же площади, подвинув вместо этого третьего в противоположную сторону. Такая последовательность шагов была бы на один шаг короче. По аналогичным причинам никакой миротворец не совершает два шага в противоположные стороны, иначе оба эти шага можно просто удалить из последовательности.

Не теряя общности, будем считать, что первый миротворец ходит только влево и вниз, второй — только вверх, а третий — только вправо. Тогда если первый прошел расстояние влево x_1, вниз — y_1, второй прошел вверх y_2, и третий прошел вправо x_2, формула для площади получившегося треугольника будет выглядеть как

$$D = (x_1 + x_2)(y_1 + y_2) - \frac{x_2 y_2}{2} - \frac{x_1(y_1 + y_2)}{2} - \frac{y_1(x_1 + x_2)}{2} = \frac{x_1 y_2 + x_2 y_1 + x_2 y_2}{2}$$

Несложно заметить, что если $x_1 > 0$, то если рассмотреть последовательность шагов, в которой x_1 на один меньше, а x_2 на один больше, итоговая площадь увеличится на y_1 и окажется не меньше D. Аналогично для y_1 и y_2. Таким образом, если искомая площадь достижима за какое-то число шагов, то она достижима и за то же число шагов, но при условии, что ходят только второй и третий миротворец.

В таком случае формула для площади получается $D = \frac{x_2 y_2}{2}$, что максимально при фиксированной сумме $x_2 + y_2$, если $x_2 = y_2$. Получаем, что достаточно было взять число $x_2 = \lceil \sqrt{s} \rceil$ — округленный вверх квадратный корень из s. Соответствующий ему y_2 либо равен ему, либо на один меньше, это можно было проверить одним условием на их произведение. В ответ затем следовало вывести $x_2 + y_2$. Время работы решения — $\mathcal{O}(1)$.

Задача E. Взрывоопасная лестница

Автор задачи и разработчик: Даниил Орешников

Будем собирать лексикографически минимальный ряд поэлементно. Заметим, что можно добиться того, чтобы на i-м месте в самом нижнем ряду в конце стоял минимальный из всех элементов в i-м столбце лестницы. Опишем действие, которое позволяет поставить минимальный из элементов i-го столбца в произвольный ряд длины $i - 1$ (на самом деле в ряд любой длины хотя бы $i - 1$, но для данного решения достаточно такого ограничения).

В качестве первого шага просто рассмотрим тот ряд, в котором находится минимальный первый элемент. Поставим первым действием его самым верхним, а все остальные ряды под ним в произвольном порядке. Все элементы, кроме первого, упадут вниз, и на первом месте останется стоять ряд из одного элемента, минимального в своем столбце. За $\mathcal{O}(n^2)$ надо аккуратно просимулировать это падение элементов вниз.

На $i + 1$-м шаге мы будем получать ответ длины $i + 1$, имея строку длины i, стоящую i-й сверху, состоящую из минимальных среди своих столбцов элементов. Найдем j — позицию строки, в которой находится минимальный из элементов на позиции $i + 1$. Поскольку в строке i ровно i элементов, то $i \neq j$. Расположим строку j непосредственно над i. Все оставшиеся строки больших размеров, чем i, расположим по очереди снизу от них, а все меньших размеров — сверху от них. Например, если $i = 3$ и $j = 5$ при $n = 6$, получится конструкция следующего вида:

2					
2	2				
2	2	2	**1**	2	
1	1	1			
2	2	2	2		
2	2	2	2	2	2

В итоге выделенный элемент упадет вниз, а все следующие — еще ниже, и на $i+1$-м месте будет стоять строка, в которой все $i+1$ ее элементов являются минимальными в своих столбцах. После этого можно переходить к следующему этапу. Проведя такие действия для всех i от 1 до $n-1$, мы получим решение за время от $\mathcal{O}(n^2)$ до $\mathcal{O}(n^3)$ в зависимости от реализации.

Задача F. Компонентная химия

Автор задачи: Евгений Карпович, разработчик: Константин Бац

Давайте формализуем задачу. У нас есть изначально пустой массив S. Каждый запрос добавляет в этот массив некоторое число a_i или удаляет одно вхождение a_i из него. После каждой такой операции требуется найти количество неупорядоченных пар $\{S_i, S_j\}$ $(i \neq j)$, что их сумма $S_i + S_j$ кратна m.

Заметим, что, рассматривая кратность суммы пар чисел числу m, можно рассматривать эти числа по модулю m. То есть, если $x = \overline{x} + z_x \cdot m$, $y = \overline{y} + z_y \cdot m$, $0 \leqslant \overline{x}, \overline{y} < m$, $z_x, z_y \in \mathbb{Z}$, то $(y + x) \bmod m = (\overline{x} + z_x \cdot m + \overline{y} + z_y \cdot m) \bmod m = (\overline{x} + \overline{y}) \bmod m$.

Давайте последовательно обрабатывать запросы, поддерживать массив `c` такой, что для всех k от нуля до $m-1$ `c`$[k]$ равно текущему количеству элементов в массиве M с остатком k, и текущее количество пар, сумма которых кратна m.

1. Пусть требуется добавить в массив M элемент x, $\overline{x} = x \bmod m$. Тогда количество пар, удовлетворяющих условию, увеличится на `c`$[(m - \overline{x}) \bmod m]$. Прибавим это число к счетчику пар, выведем значение на экран и увеличим `c`$[\overline{x}]$ на один.

2. Пусть требуется удалить из массива M элемент x, $\overline{x} = x \bmod m$. Найдем $\overline{y} = (m - \overline{x}) \bmod m$ – остаток чисел, которые образуют с x пару. Может быть два случая:

   - Если $\overline{x} \neq \overline{y}$, то в `c`$[\overline{y}]$ посчитано число пар, которые распадутся при удалении x из m. Значит обновим количество пар с суммой, кратной m, выведем ответ и обновим $C[\overline{x}]$, то есть вычтем оттуда единицу.
   - Если $\overline{x} = \overline{y}$, то, так как $(\overline{x} + \overline{y}) \bmod m = 0$, в `c`$[\overline{y}]$ в том числе посчитано число x, которое, как известно, само с собой пару не образует. Значит число пар, уменьшится на `c`$[\overline{x}] - 1$. После этого нужно вывести ответ и уменьшить `c`$[\overline{x}]$ на 1.

Ввиду ограничений на m (до 10^9), поддерживать массив `c` целиком не получится. Однако вместо обычного массива можно использовать произвольный словарь, например `HashMap`, `unordered_map`, `dict`. Тогда, поскольку число запросов ограниченно 10^5, то и размер словаря будет занимать $\mathcal{O}(10^5)$ памяти, а каждый запрос будет все еще обрабатываться за $\mathcal{O}(1)$.

Замечание 1. Стоит учитывать, что некоторые языки программирования, например C++ и Java, считают остаток от деления отрицательных чисел не так, как это принято в математике. Напомним, остаток от деления числа a на b — это такое $0 \leqslant c < b$, что $a = b \cdot z + c$ и $z \in \mathbb{Z}$.

Замечание 2. На самом деле, при удалении числа из массива M можно избежать рассмотрения случаев, если сперва уменьшить `c`$[\overline{x}]$, а потом пересчитать счетчик пар.

Итого, время работы программы: $\mathcal{O}(n)$.

Задача G. Магический кристалл

Авторы задачи: Даниил Голов и Даниил Орешников, разработчик: Даниил Орешников

Отдельно рассмотрим случаи $n \leqslant 3$ и $n = 4$. При $n \leqslant 3$ несложным перебором можно понять, что решения нет, и следует вывести «`-1 -1`». При $n = 4$ есть тривиальное решение вида $2 + 2 = 4$. Действительно, это два слагаемых и один множитель.

Для любого $n > 4$ есть конструктивный способ построить ответ. При четных $n = 2k$ разобьем n на два различных множителя 2 и k. Их сумма равна $k + 2$. Разложим n на слагаемые как $n = (k + 2) + 1 + 1 + \ldots + 1$, то есть одно слагаемое $(k + 2)$ и еще $k - 2$ единицы. Их произведение, как можно заметить, тоже равно $k + 2$.

Случай нечетного $n = 2k + 1$ аналогичен. Представим n в виде произведения $n \cdot 1$. Сумма этих двух множителей равна $n + 1$. Тогда разложим n на слагаемые как $n = (k + 1) + 2 + 1 + 1 + \ldots + 1$, то есть $(k + 1)$, 2 и еще $k - 2$ единицы. Их произведение равно $2 \cdot (k + 1) = 2k + 2 = n + 1$.

Задача H. Магические часы

Авторы задачи: Григорий Хлытин, разработчик: Даниил Орешников

Для каждого запроса можно было бы запустить симуляцию процесса, чтобы определить, достигнется ли желаемое состояние, но такое решение работало бы слишком долго, если бы мы рассматривали каждый тик по очереди.

Заметим, что если при движении между двумя состояниями минутная стрелка хотя бы раз догоняла часовую, то существовал момент времени между двумя состояниями, когда минутная стрелка была на нуле. Обозначим за (h_1, m_1) стартовое состояние, (h_2, m_2) — конечное состояние в запросе. Тогда можно отдельно проверить, достижимо ли второе состояние из первого, без прохода минутной стрелки через ноль, а дальше отталкиваться от того, что хотя бы раз минутная стрелка перескакивала в ноль. Для проверки первого случая достаточно заметить, что часовая стрелка движется с постоянной скоростью, и проходит строго меньше целого круга, поэтому через $h_2 - h_1$ можно вычислить сколько в точности прошло времени и проверить, перешло ли m_1 в m_2 за это время.

Случай прохождения через 0 же рассмотрим так: по (h_2, m_2) однозначно восстанавливается, из какого состояния $(h_2^*, 0)$ и сколько времени назад начинала двигаться минутная стрелка (для этого должно выполняться, что $m_2 \bmod 12 = 0$), и, аналогично, сколько времени пройдет перед тем, как состояние (h_1, m_1) перейдет в $(h_1^*, 0)$, и каким будет это h_1. Это произойдет за $\left\lceil \frac{(h_1 - m_1) \bmod (60 \cdot 12)}{11} \right\rceil$ тиков.

После этого достаточно проверить, достижимо ли состояние $(h_2^*, 0)$ из $(h_1^*, 0)$, и если да, то за сколько тиков. В общем случае из состояния $(h, 0)$ минутная стрелка догонит часовую за $\lceil \frac{h}{11} \rceil$ тиков. Только надо не забыть особый случай $h = 0$, при котором пройдет 60 тиков перед достижением минутной стрелкой нуля. Проведем симуляцию, переходя из состояния вида $(h, 0)$ в следующее состояние такого же вида за $\mathcal{O}(1)$ времени, суммируя требующееся для каждого перехода количество тиков.

Либо мы придем в состояние, которое уже было посещено, либо придем в желаемое конечное состояние. В первом случае ответ, что конечное состояние недостижимо, иначе — выводим насчитанное число тиков. Время работы решения — $\mathcal{O}(720q)$.