Задача A. Страшные числа

Автор задачи и разработчик: Сергей Щербаков

Воспользуемся решетом Эратосфена. Реализация решета Эратосфена за время $\mathcal{O}(n)$ также находит для каждого числа в отрезке минимальный простой делитель. Зная для каждого числа его минимальный простой делитель, можно факторизовать (разложить на простые) любое число t за время $\mathcal{O}(\log t)$. Для этого достаточно просто делить число на минимальное простое в разложении, пока оно не станет равно 1.

Заметим, что максимальное возможное количество простых в разложении числа до $2 \cdot 10^5$ не превосходит 25. Для каждого k от 0 до 25 построим массив префиксного предподсчета, $\texttt{cnt}_{k,l}$ — количество чисел $\leqslant l$, имеющих ровно k простых в разложении. Посчитать такой массив можно с помощью перехода $\texttt{cnt}_{k,l} = \texttt{cnt}_{k,l-1} + (\text{primes}(l) \stackrel{?}{=} k)$.

Теперь на запросы можно отвечать за $\mathcal{O}(1)$: на запрос (k_i, l_i, r_i) достаточно вывести ответ $\texttt{cnt}_{k,r} - \texttt{cnt}_{k,l-1}$.

Задача B. Кошмар наяву

Автор задачи: Александр Гордеев, разработчик: Мария Жогова

Рассмотрим несколько случаев расположения точек относительно круга:

1. Обе точки находятся на круге. Тогда при вращении круга прямая между этими двумя точками будет поворачиваться на тот же угол, значит гарантированно существует такой угол поворота, при котором две интересующие нас прямые будут параллельны.

2. Обе точки находятся вне круга. Тогда при вращении круга вообще не происходит изменения в положении прямых, и достаточно проверить параллельность их исходного положения.

3. Одна точка (не теряя общности, (x_1, y_1)) находится на круге, а вторая — вне его. В таком случае прямая может поворачиваться в некотором интервале доступных углов. Заметим, что экстремальные значения угол наклона прямой принимает, когда вектор $\overrightarrow{CQ_1}$ перпендикулярен вектору $\overrightarrow{CQ_2}$, где $C(x, y)$ — центр окружности, а Q_1 и $Q_2(x_2, y_2)$ — наши точки.

   Обозначим за Q_1^1 и Q_1^2 соответствующие этим двум экстремальным углам положения Q_1 на повернутом круге. Тогда, чтобы проверить, что прямая Q_1Q_2 может быть параллельной второй прямой, достаточно проверить, что направляющий вектор второй прямой лежит между векторами $\overrightarrow{Q_2Q_1^1}$ и $\overrightarrow{Q_2Q_1^2}$.

Все описанные выше проверки можно осуществлять с помощью скалярного и векторного произведений векторов.

Альтернативное решение третьего случая — вместо вычисления Q_1^1 и Q_1^2 в явном виде (для чего понадобится поворот вектора на угол), можно повернуть первую прямую относительно Q_2 так, чтобы она стала параллельна второй (то есть на самом деле параллельно сдвинуть вторую до точки Q_2), после чего проверить, что точка Q_1 может оказаться на ее пересечении с кругом. Для этого достаточно проверить, что $|CQ_1|$ не меньше расстояния от C до полученной прямой. Все описанные вычисления в таком случае можно совершать в целых числах, не переходя к арифметике чисел с плавающей точкой.

Задача C. Давайте разделимся!

Автор и разработчик задачи: Мария Жогова

Переберем в какое помещение пойдет какой лидер.

Пусть первый лидер имеет безрассудство a_1 и пошел к помещение b_1, а второй a_2 и b_2 соответственно.

Пусть в первое помещение пошло x человек,тогда во второе $n - x$, тогда надо минимизировать $\max(a_1 \cdot b_1 \cdot x, a_2 \cdot b_2 \cdot (n - x))$. Если нарисовать графики $a_1 \cdot b_1 \cdot x$, $a_2 \cdot b_2 \cdot (n - x)$ и максимум из этих

величин. Нетрудно убедиться, что оптимальный ответ находится в точке пересечения первых двух функций, но ответ должен быть целым, поэтому проверим два соседних целых значения и найдем минимум из них.

Задача D. Самая страшная история

Автор задачи: Тимур Гараев, разработчик: Даниил Орешников

Будем хранить всю строку в виде декартова дерева по неявному ключу, элементами которого являются символы строки. Помимо этого, будем в каждой вершине хранить информацию о ее поддереве:

- `size` — размер поддерева (количество символов);
- `cnt` — количество пробелов;
- `last` — позицию последнего пробела (или -1, если их нет).

Сведем все описанные в задаче запросы к запросам на таком декартовом дереве:

1. Чтобы по позиции символа в строке i вернуть номер слова и его номер в этом слове, достаточно заметить, что при выполнении $\texttt{split}(i)$, в левом поддереве l содержатся все предшествующие слова и следующие за ними пробелы, таким образом номер слова будет равен $\texttt{cnt}(l) + 1$, а номер позиции в слове будет равен $i - \texttt{last}(l)$.
2. Наоборот, чтобы по номеру слова w и позиции в нем p найти позицию в строке, найдем $(w-1)$-й пробел, после чего отступим от него p позиций вправо. Чтобы найти позицию определенного пробела в строке, можно спускаться из корня в левое или правое поддерево в зависимости от того, чему равно $\texttt{cnt}(l)$.
3. Для удаления или вставки символа достаточно несколько раз воспользоваться стандартными `split` и `merge` по позиции.

Таким образом, осталось только реализовать все это, после каждого действия пересчитывая все три параметра в изменившихся вершинах. Время работы равно $\mathcal{O}(q \log n)$.

Задача E. Trick or Treat!

Авторы задачи: Захар Черемных и Даниил Орешников, разработчик: Даниил Орешников

При рассмотрении точки отдельно рассмотрим ближайшие к ней слева-сверху, слева-снизу, справа-сверху и справа-снизу. Такое разделение позволяет перейти к формуле расстояния без модулей.

Будем обрабатывать точки слева направо сканирующей прямой. В таком случае от точки (x_i, y_i) расстояние до точек (x_j, y_j) слева-сверху будет равно $(x_i - x_j) + (y_j - y_i) = (x_i - y_i) - (x_j - y_i)$. Для фиксированного i кратчайшее расстояние достигается при максимальном $x_j + y_j$. Симметрично, для точек слева-снизу кратчайшее расстояние можно будет посчитать как $(x_i + y_i) - \max(x_j + y_j)$. Обозначим $d_1(A) = A_x + A_y$ и $d_2(A) = A_x - A_y$.

Будем хранить на сканирующей прямой все встреченные до этого точки в декартовом дереве по явному ключу по координате Y. Тогда при рассмотрении очередной точки (x_i, y_i) будет достаточно разбить все точки на сканирующей прямой на нижнюю и верхнюю часть с помощью $\texttt{split}(y_i)$, после чего найти $\max d_1$ в нижней части и $\max d_2$ в верхней. Для этого можно просто поддерживать максимальные значения d_1 и d_2 в каждой вершине дерева.

Осталось только рассмотреть точки строго на той же вертикали и находящиеся справа от текущей. Точки, находящиеся справа, можно, рассмотрев симметричную картину (заменив все (x_i, y_i) на $(\texttt{X\_MAX} - x_i, y_i)$) и запустив еще раз ту же самую сканирующую прямую. Рассмотреть точки с той же X-координатой, что и у текущей, можно за $\mathcal{O}(1)$, так как после сортировки точек как пар, ближайшие по Y-координате с той же X-координатой будут соседними с текущей.

Задача F. Стрелочник

Автор задачи и разработчик: Владислав Власов

Заметим, что нам нужно найти кратчайший путь в некотором графе, где ребро может весить 1 или 0. Для решения таких задач подойдет алгоритм Дейкстры или 0-1 bfs.

Теперь поймем, что в каждой клетке может быть выгодно быть в разные моменты, а не только в минимальный возможный. Но при этом если два момента времени t_1 и t_2 отличаются на время, кратное восьми, то достаточно знать $\min(t_1, t_2)$, так как каждые 8 единиц времени все стрелки возвращаются в исходное положение. Таким образом, вместо обычного $\texttt{dist}[v]$, минимального времени, в которое можно оказаться в вершине v, будем хранить восемь массивов $\texttt{dist}_d[v]$ — минимальное время, в которое можно оказаться в вершине v, имеющее остаток d по модулю 8.

Пересчет же будет таким же, как и в обычном поиске в ширину, только теперь каждую вершину можно воспринимать как 8 независимых вершин: (v, d) будет соответствовать вершине v, в которую мы попали в момент времени t, что $t \bmod 8 = d$. Время работы алгоритма равно $\mathcal{O}(n + m)$, но с в 8 раз большей константой.

Задача G. Уиджа

Автор задачи: Александр Голубев, разработчик: Даниил Орешников

Для решения задачи можно было для начала рассмотреть менее общие случаи расположения указателя на доске. Например, если указатель находится в центре доски, то у второго игрока есть симметричная стратегия: всегда после своего хода возвращать его в центр. Аналогично, если указатель находится в середине по одному из направлений, но не по другому, то первый игрок может получить симметричную стратегию, «обрезав» более длинную часть доски, сведя задачу к предыдущему случаю.

Рассмотрев такие случаи, можно было прийти к мысли, что стоит рассматривать не позицию указателя, а количество строк сверху и снизу от него, и количество столбцов слева и справа. Обозначим соответствующие количества за u, d, l и r соответственно. Заметим, что за свой ход игрок может просто уменьшить любое из этих чисел на произвольное значение. Говоря формально, наша игра является суммой четырех нимов из u, d, l и r камней. Функция Гранди для такой игры принимает значение $u \oplus d \oplus l \oplus r$, где $\oplus$ — побитовое исключающее «или».

Выигрышная победа есть у первого игрока тогда и только тогда, когда $u \oplus d \oplus l \oplus r \neq 0$. Сама выигрышная стратегия заключается в том, чтобы после текущего хода это значение оставалось нулевым. Можно отдельно доказать, что всегда существует ход, приводящий из ненулевого состояния в нулевое, однако это также напрямую следует из свойств функций Гранди. Таким образом, выбирать сторону следовало, опираясь на начальное значение такого выражения, а каждым своим ходом сводить игру к нулевому значению. Чтобы понять, какую сторону следовало резать, можно было проверить существование нулевого состояния для каждой стороны отдельно. Например, чтобы проверить, что существует такое $l_1 < l$, что $u \oplus d \oplus l_1 \oplus r = 0$, достаточно было проверить, что $u \oplus d \oplus r < l$.

Задача H. Расстановка тыкв

Автор разбора: Даниил Орешников

Будем решать задачу методом динамического программирования. Пусть $\texttt{dp}[i]$ — «оптимальный», то есть дающий максимальное значение удовлетворенности, способ расставить тыквы на первых i местах, при котором на месте номер i **обязательно стоит тыква**. Переберем предыдущее место j, на котором стоит тыква. Для фиксированного j формула пересчета удовлетворенности будет иметь вид

$$\texttt{dp}[i] = \texttt{dp}[j] - c_i + \sum_{t=1}^{n} |x_i - x_j - d_t|.$$

Заметим, что максимизация этой величины равносильна минимизации $\texttt{dp}[j] + \sum\limits_{t=1}^{n} |x_i - x_j - d_t|$.

Определим $f_j(x)$ как $\texttt{dp}[j] + \sum\limits_{t=1}^{n} |x - x_j - d_t|$. Тогда для фиксированного x_i мы ищем $\max\limits_{j<i} f_j(x_i)$. Заметим, что все f_j имеют не более одной точки пересечения каждый с каждым, так как

- каждая из них является суммой модулей линейных функций;
- модуль линейной функции выпуклый вверх, поэтому сумма таких модулей тоже выпукла вверх;
- график f_j является параллельным переносом f_1 на $x_j - x_1$ по оси X и на $\texttt{dp}[j] - \texttt{dp}[1]$ по оси Y.

Выпуклые вверх функции, получающиеся друг из друга параллельным переносом, имеют не более одной точки пересечения, таким образом $\max\limits_{j} f_j(x)$ имеет не более n участков, на которых максимальной является конкретная f_j. Учитывая все эти условия, можно было применить дерево Ли Чао, чтобы при пересчете $\texttt{dp}[i]$ найти $\max\limits_{j} f_j(x_i)$, а затем добавить f_i в рассмотрение.

Чтобы быстро вычислять $\sum\limits_{t=1}^{n} |x - x_j - d_t|$, достаточно в самом начале отсортировать все d_t по возрастанию и посчитать их префиксные суммы. Найдем бинпоиском первое d_t, большее, чем $x - x_j$, после чего раскроем все модули для меньших d со знаком плюс, а все остальные — со знаком минус. Получившееся выражение можно будет легко посчитать с помощью префиксных сумм d_t.

Поскольку в задаче требуется найти оптимальную расстановку, в которой первое и последнее место заняты тыквами, достаточно начинать с $\texttt{dp}[1] = -c_1$ и взять в качестве ответа $\texttt{dp}[n]$.

Задача I. Интересные празднования

Автор разбора: Даниил Орешников

Рассмотрим последовательность $t = \texttt{seq}_{26}$. Заметим, что по построению, $\texttt{seq}_1$, ..., $\texttt{seq}_{25}$ являются ее префиксами с длинами 1, ..., $2^{25} - 1$, соответственно. Очевидно также, что длиной $\texttt{seq}_i$ является $2^i - 1$, так как $|\texttt{seq}_1| = 1$, а $|\texttt{seq}_{i+1}| = 2|\texttt{seq}_i| + 1$.

Посчитаем следующую величину методом динамического программирования: $\texttt{dp}[i][j]$ равно количеству подпоследовательностей из первых i символов s, равных префиксу t длины j. Ее довольно несложно пересчитывать:

- $\texttt{dp}[i][0]$ увеличивается по сравнению с $\texttt{dp}[i-1][0]$ каждый раз, когда встречается $s_i =$ 'a';
- $\texttt{dp}[i][j] = \texttt{dp}[i-1][j]$, либо же $\texttt{dp}[i-1][j] + \texttt{dp}[i-1][j-1]$, если $s_i = t_j$, так как каждый префикс длины $j-1$, встреченный ранее, можно дополнить новым символом до префикса длины j.

Такую динамику можно посчитать за время $\mathcal{O}(|s|^2)$, так как нет смысла считать $\texttt{dp}[i][j]$ для $j > i$. А в ответ достаточно вернуть $\sum\limits_{2^j-1 \leqslant n} \texttt{dp}[n][j]$, ведь префиксы именно такой длины соответствуют интересным последовательностям.

Задача J. Монстры и люди

Автор разбора: Даниил Орешников

Переформулируем задачу: требуется выбрать наибольшее по размеру множество A вершин в графе, чтобы между вершинами этого множества не было ребер. Действительно, это является достаточным и необходимым условием в соответствии с задачей. Скажем, что монстры — это вершины из A, люди — вершины из $V \setminus A$, а ребра — информация о том, кто кого обвинил. По условию разрешены все ситуации, кроме ситуации «монстр обвинил монстра», то есть кроме ребра из A в A.

Будем решать задачу жадным алгоритмом: пусть существует вершина, в которую не ведет ни одно ребро, в таком случае существует оптимальный ответ, в котором эта вершина является монстром. Если в каком-то ответе рассмотренная вершина v не является монстром, то рассмотрим ребро

$v \to u$: либо u — человек, тогда ответ можно увеличить, либо u — монстр, но тогда можно сделать u человеком и v монстром, и ответ не уменьшится.

Таким образом, просто будем поддерживать множество вершин, в которые входит 0 ребер, для каждой из них помечать, что она является монстром, после чего

1. пометим обвиненную ей вершину u человеком;
2. удалим ребро из u в обвиненную ей вершину.

В конце, когда не останется вершин со входящей степенью 0, весь граф разобьется на циклы. Очевидно, что в цикле длины k можно выбрать $\left\lfloor \frac{k}{2} \right\rfloor$ вершин. Все это можно реализовать с помощью dfs с дополнительным флагом.