Задача A. Скунс на поляне

Автор задачи и разработчик: Константин Бац

Скунс может выбрать любые две из трёх куч и забрать из них все ягоды. Поэтому нужно найти максимальную сумму двух куч.

Заметим, что если отсортировать количества ягод, то выгоднее всего взять две самые большие кучи. Альтернативно можно сложить все ягоды и вычесть размер наименьшей кучи.

Таким образом, ответ равен $a + b + c - \min(a, b, c)$.

Время работы решения — $\mathcal{O}(1)$.

Задача B. Звериные норы

Разработчик задачи: Константин Бац

Будем обрабатывать норы справа налево. Для каждой позиции нужно понять, существует ли правее зверь с большей массой.

Во время обхода будем хранить максимальную массу среди всех уже просмотренных зверей справа. Пусть это значение равно mx. Если масса текущего зверя меньше mx, значит, правее действительно есть более тяжёлый зверь, и ответ для этой норы равен 1. Иначе зверь никого тяжелее себя справа не видит, поэтому ответ равен 0.

После обработки очередной норы обновим mx, взяв максимум из текущего значения и массы этого зверя.

Таким образом, каждый элемент массива обрабатывается ровно один раз. Сложность алгоритма составляет $\mathcal{O}(n)$, а дополнительная память — $\mathcal{O}(1)$ помимо массива ответа.

Задача C. Филин и суслик

Автор идеи: Антон Карабанов, разработчик: Константин Бац

Рассмотрим фиксированную клетку, в которой сидит филин. Пусть её координаты (x, y).

Если филин смотрит вверх, то он видит все клетки строк с номерами не больше x. Значит, суслик может находиться только в строках ниже, то есть существует $(n - x) \cdot m$ подходящих клеток.

Аналогично: при взгляде вниз число невидимых клеток равно $(x - 1) \cdot m$, при взгляде влево — $n \cdot (m - y)$, при взгляде вправо — $n \cdot (y - 1)$.

Просуммируем количество допустимых клеток для суслика по всем четырём направлениям. Для произвольной клетки (x, y) получаем

$$(n - x) \cdot m + (x - 1) \cdot m + n \cdot (m - y) + n \cdot (y - 1) = m \cdot (n - 1) + n \cdot (m - 1).$$

Получается, что для любой клетки филина число способов выбрать направление и клетку суслика одинаково и равно $m \cdot (n - 1) + n \cdot (m - 1) = 2nm - n - m$.

Клетку для филина можно выбрать nm способами. Для каждой такой клетки существует ровно $2nm - n - m$ подходящих направлений взгляда и клеток суслика. Поэтому итоговый ответ равен $nm \cdot (2nm - n - m)$.

Остаётся вычислить это значение по модулю $10^9 + 7$. Сложность такого решения составляет $\mathcal{O}(1)$.

Задача D. Уборка леса

Авторы задачи: Маргарита Саблина и Константин Бац, разработчик: Константин Бац

Заметим, что горизонтальные и вертикальные тропинки можно рассматривать независимо. Предположим, что Светлана хочет проложить тропинку по некоторой строке. Тогда её интересуют только клетки этой строки: деревья вне неё она трогать не может, но они и не мешают движению.

Рассмотрим клетку строки, занятую стволом дерева «*». Если после поворота дерева на 180 градусов его ствол может перейти в другую клетку и эта новая клетка свободна, то дерево можно «убрать» с рассматриваемой строки. При этом важно учитывать направление дерева. Если дерево лежит горизонтально, то после поворота ствол остаётся в той же строке, просто переезжает в другую клетку этой строки. Если новая позиция свободна, обе клетки строки становятся пригодными для тропинки. Если же дерево вертикальное, то после поворота ствол вообще уходит из рассматриваемой строки, и текущая клетка также освобождается.

Для каждой строки построим её текущее состояние и отметим клетки, которые можно освободить описанным способом. После этого задача сводится к поиску максимального непрерывного отрезка клеток, которые либо уже свободны, либо могут стать свободными после поворотов деревьев. Такой отрезок легко найти одним проходом по строке.

Аналогично можно обработать все столбцы. Для вертикальной тропинки нужно рассмотреть только деревья, пересекающие данный столбец, и проверить, можно ли освободить занимаемые ими клетки. Затем следует найти максимальный непрерывный свободный участок в столбце.

Ответом будет максимум среди всех строк и всех столбцов. Каждая клетка участвует лишь в константном числе проверок, поэтому суммарное время работы алгоритма составляет $\mathcal{O}(nm)$. По условию $n \cdot m \leqslant 2 \cdot 10^5$, поэтому такое решение укладывается в ограничения.

Задача E. Темная ночь

Разработчик задачи: Павел Скобелин

Заметим, что Маша может перелетать только от звезды с меньшим номером к звезде с большим. Поэтому если она оказалась в некоторой звезде v, то стартовать она могла только из тех звёзд, из которых существует путь до v.

Для каждой звезды будем хранить минимальное значение силы желания среди всех звёзд, из которых до неё можно добраться, включая её саму. Обозначим эту величину за $mn[v]$. Если мы знаем $mn[v]$, то лучший вариант закончить путешествие в звезде v даст счастье $a[v] - mn[v]$.

Поскольку все лучики ведут только вперёд по номерам, звёзды удобно обрабатывать в порядке от 1 до n. Изначально для каждой звезды $mn[v] = a[v]$. Затем для каждого лучика $u \to v$ можно передать в звезду v информацию о том, что до неё можно добраться через u. Поэтому обновляем $mn[v] = \min(mn[v], mn[u])$.

После этого можно попробовать закончить путешествие в звезде v и обновить ответ задачи значением $a[v] - mn[v]$:

$$ans = \max(ans, a[v] - mn[v]).$$

Таким образом, мы постепенно распространяем по лучикам минимальную стоимость старта и для каждой звезды проверяем, какое счастье можно получить, если закончить путь именно в ней.

Каждый лучик обрабатывается ровно один раз, поэтому сложность алгоритма составляет $\mathcal{O}(n + m)$.

Задача F. Белка и арифметика

Автор задачи и разработчик: Орешников Даниил

По условию массив $d_1, d_2, \ldots, d_{k-1}$ должен быть арифметической прогрессией из неотрицательных чисел. Положим $m = k - 1$. Тогда разностный массив имеет длину m.

Если $k = 1$, то разностный массив пуст. Поэтому любой один элемент от 0 до n подходит. Ответ равен $n + 1$. Если $k = 2$, то разностный массив состоит из одного числа: $d_1 = a_2 - a_1$. Требуется только $d_1 \geqslant 0$, то есть $a_1 \leqslant a_2$. Количество пар $0 \leqslant a_1 \leqslant a_2 \leqslant n$ равно $\frac{(n+1)(n+2)}{2}$.

Далее будем считать, что $k \geqslant 3$, то есть $m \geqslant 2$.

Перенумеруем элементы разностного массива в нумерации с нуля. Так как разностный массив является арифметической прогрессией, его можно записать в виде

$$d_i = p + iq,$$

где $i = 0, 1, \ldots, m - 1$, и p — первый элемент прогрессии, а q — её разность.

Обозначим сумму всех d_i за

$$S = d_0 + d_1 + \ldots + d_{m-1}.$$

Так как все разности неотрицательны, последовательность $a_1, a_2, \ldots, a_k$ неубывает. Значит, если известен разностный массив, то вся последовательность однозначно задаётся значением a_1.

Последний элемент равен

$$a_k = a_1 + (a_2 - a_1) + \ldots + (a_k - a_{k-1}) = a_1 + S.$$

Условие $0 \leqslant a_i \leqslant n$ для всех i теперь эквивалентно двум условиям:

$$0 \leqslant a_1, \quad a_1 + S \leqslant n.$$

Следовательно, при фиксированном разностном массиве можно выбрать ровно $n-S+1$ значений a_1, если $S \leqslant n$, и ни одного значения иначе.

Сумма разностного массива

Рассмотрим арифметическую прогрессию длины m:

$$p, p+q, p+2q, \ldots, p+(m-1)q.$$

Если $q \geqslant 0$, то из неотрицательности всех элементов достаточно требовать $p \geqslant 0$. Сумма элементов этой прогрессии равна

$$S = mp + \frac{m(m-1)}{2}q.$$

Обозначим $C = \frac{m(m-1)}{2}$, тогда $S = mp + Cq$.

Как учитывать $q < 0$

Шаг арифметической прогрессии может быть отрицательным. Например, $5, 3, 1$ — это арифметическая прогрессия из неотрицательных чисел с отрицательным шагом.

Любую такую прогрессию можно развернуть. Если массив $d_0, d_1, \ldots, d_{m-1}$ имеет отрицательный шаг, то массив $d_{m-1}, d_{m-2}, \ldots, d_0$ имеет положительный шаг.

При этом сумма элементов разностного массива S не меняется, значит, и количество подходящих значений a_1 не меняется.

Поэтому можно перебирать только $q \geqslant 0$:

- прогрессии с $q = 0$ учитываются один раз;
- прогрессии с $q > 0$ учитываются два раза: для положительного и отрицательного шага.

Подсчёт вклада фиксированного q

Зафиксируем $q \geqslant 0$. Тогда нужно перебрать все $p \geqslant 0$, для которых $mp + Cq \leqslant n$.

Обозначим $R = n - Cq$. Если $R < 0$, то подходящих значений p нет. Иначе $0 \leqslant p \leqslant \left\lfloor \frac{R}{m} \right\rfloor$. Обозначим $P = \left\lfloor \frac{R}{m} \right\rfloor$.

Для каждого p вклад в ответ равен

$$n - (mp + Cq) + 1.$$

Значит, суммарный вклад фиксированного q равен

$$\sum_{p=0}^{P} \left(n - (mp + Cq) + 1\right).$$

Раскроем сумму:

$$\sum_{p=0}^{P} (n - Cq + 1 - mp) =$$

$$= (P+1)(n - Cq + 1) - m\sum_{p=0}^{P} p =$$

$$= (P+1)(n - Cq + 1) - m \cdot \frac{P(P+1)}{2}.$$

Итоговый алгоритм

Нужно рассматривать только такие q, для которых хотя бы при $p = 0$ сумма разностей не превосходит n, то есть

$$q = 0, 1, \ldots, \left\lfloor \frac{n}{C} \right\rfloor.$$

1. Если $k = 1$, вывести $n + 1$.
2. Если $k = 2$, вывести $\frac{(n+1)(n+2)}{2}$.
3. Иначе положить $m = k - 1$ и $C = \frac{m(m-1)}{2}$.
4. Перебрать все q от 0 до $\left\lfloor \frac{n}{C} \right\rfloor$.
5. Для каждого q посчитать $P = \left\lfloor \frac{n-Cq}{m} \right\rfloor$.
6. Посчитать вклад:
$$(P + 1)(n - Cq + 1) - m \cdot \frac{P(P + 1)}{2}.$$
7. Если $q = 0$, добавить этот вклад один раз.
8. Если $q > 0$, добавить этот вклад два раза.

Все вычисления ответа выполняются по модулю 998 244 353. Время работы алгоритма: $\mathcal{O}\left(\frac{n}{k^2}\right)$ с константной памятью.

Задача G. Ёжик и дупло

Автор задачи: Константин Бац, разработчик: Даниил Орешников

Для каждого бука i известно число p_i. Если $p_i = 0$, то здесь Феликс может взять нектар сразу. Если $p_i \neq 0$, то прежде Феликс должен взять нектар в дупле p_i. Поэтому, чтобы взять нектар в дупле i, Феликс должен сначала посетить дупло p_i.

Рассмотрим дупло v, в котором Феликс должен взять нектар. Если $p_v = 0$, то Феликс может взять нектар в нём сразу. Если $p_v \neq 0$, то перед этим Феликс должен взять нектар в дупле p_v. Но для дупла p_v тоже прежде может понадобиться принести нектар из другого дупла. Поэтому надо пройти по цепочке:

$$v, p_v, p_{p_v}, \ldots$$

Эта цепочка заканчивается в 0, потому что по условию циклов, которые бы мешали найти ответ, нет. Во всех деревьях этой цепочки Феликс обязан побывать.

Феликсу нужно взять нектар из буков с номерами

$$b_1, b_2, \ldots, b_k..$$

Для каждого из них нужно пройти по цепочке зависимостей до 0. Ответом будет количество различных деревьев, которые встретились хотя бы в одной из этих цепочек. Важно, что один и тот же бук не нужно посещать несколько раз. Если Феликс уже был в этом дупле, то считать его повторно не надо.

Поэтому просто пройдемся для каждого бука b_i по цепочке $b_i \to p_{b_i} \to p_{p_{b_i}} \to \ldots$ и пометим каждый из них. Если такой бук уже помечен как «необходимый», то остановим наш перебор: вся следующая часть цепочки уже была рассмотрена и выписана.

Также заметим, что из этого выстраивается последовательность обхода буков: для каждой такой (возможно неполной) цепочки достаточно просто развернуть ее и выписать в конец массива ответа. Действительно, если цепочка полная, то ее развернутый порядок просто задает возможный порядок посещения буков. А если цепочка «оборвалась» на уже встреченном ранее дереве, то оно уже было посещено (выписано в ответ) ранее, и можно продолжить посещать дупла с того места, на котором мы остановились.

Алгоритм

Будем хранить массив `used` и массив ответа `ans`. Значение `used`$[v]$ равно 1, если бук v уже был посчитан и добавлен в ответ, тогда как `ans` — буквально список деревьев, которые надо посетить, в нужном порядке.

Изначально все значения `used`$[v]$ равны 0. Для каждого нужного бука b_i сделаем следующее:

1. положим $v = b_i$ и заведем `seq` — цепочку этого дерева;
2. пока $v \neq 0$ и `used`$[0] = 0$, отметим бук v, добавив его в конец `seq`;
3. перейдём к дереву p_v.

Если в какой-то момент мы пришли в уже отмеченное дупло, можно остановиться. Вся цепочка выше него уже была обработана раньше. В этот момент надо выписать весь `seq` в конец `ans`.

Каждое дупло может быть отмечено не более одного раза. После того как бук отмечен, алгоритм больше не проходит дальше через него при последующих обработках. Поэтому суммарное число шагов по всем цепочкам не превосходит $n + k$. Итоговая сложность: $\mathcal{O}(n)$.

Задача H. Секретные сигналы

Разработчик задачи: Константин Бац

Рассмотрим все подстроки длины k в заданной записи. Если некоторый сигнал встречается хотя бы два раза, то среди этих подстрок найдутся две одинаковые.

Будем последовательно перебирать все подстроки длины k и сохранять уже встреченные в множество. Для очередной подстроки проверяем, находится ли она в множестве. Если находится, значит, такой сигнал уже встречался раньше, и ответ равен «`YES`».

Если подстроки ещё не было, добавляем её в множество и продолжаем обработку. После просмотра всех позиций, если повторов не найдено, выводим «`NO`».

Так как $k \leqslant 6$, получение очередной подстроки занимает константное время. Всего рассматривается $n - k + 1$ подстрок, поэтому время работы алгоритма составляет $\mathcal{O}(n)$.

Задача I. Лесная полянка

Автор задачи: Маргарита Саблина, разработчик: Константин Бац

Заметим, что условие задачи связывает только симметричные полянки: первую с последней, вторую с предпоследней и так далее. Изменения в одной паре никак не влияют на остальные, поэтому минимальную стоимость можно искать для каждой пары независимо, а затем сложить результаты.

Рассмотрим пару значений (x, y). Нам нужно изменить их так, чтобы одно число стало кратно другому. Каждое увеличение или уменьшение на единицу стоит одно заклинание, поэтому цена перехода в новые значения (u, v) равна $|u - x| + |v - y|$.

После изменений числа должны оставаться в диапазоне от 1 до 9. Ключевое наблюдение состоит в том, что возможных значений очень мало — всего девять. Поэтому для пары (x, y) можно просто перебрать все пары чисел (u, v) от 1 до 9, оставить только те, для которых выполняется условие делимости (u кратно v или v кратно u), и выбрать вариант с минимальной стоимостью.

Для удобства реализации полезно выделить функцию, например, $q(x, y)$, которая будет вычислять ответ для заданной пары чисел. Для некоторых простых случаев ответ известен заранее: например, если одно из чисел равно 1 или числа уже равны, условие выполнено без изменений. Во всех остальных случаях перебираются все допустимые конечные значения и находится минимальное число заклинаний.

После этого остаётся пройти по всем симметричным парам массива и вызывать функцию q для каждой такой пары. Тогда остается сложить сумму необходимых изменений в каждой паре, и это и будет ответом на задачу.

Так как значения на полянках лежат только в диапазоне от 1 до 9, перебор ответа для каждой пары чисел работает за константное время (меньше 100 итераций цикла). Поэтому общее время работы решения определяется только числом пар и составляет O(n).

Задача J. Цветущий сад

Авторы задачи: Маргарита Саблина, разработчик: Даниил Голов

Вместо того чтобы рассматривать каждый цветок отдельно, будем смотреть на дни, в которые необходимо поливать цветы. Если в некоторый день цветут несколько растений, то у Ёжика есть ровно два варианта: либо полить каждый цветок отдельно, потратив сумму их c_i, либо позвать Слонёнка и потратить ровно k миллилитров. Поэтому в этот день требуется $\min(\sum c_i, k)$, где сумма c_i считается по цветкам, которые взошли и ещё не завяли в этот день.

Остаётся посчитать эту величину для всех дней. Делать это напрямую нельзя, потому что номера дней могут достигать 10^9.

Заметим, что набор цветущих цветов меняется только в дни, когда какой-то цветок всходит или увядает. Поэтому создадим события двух типов: начало жизни цветка и конец его жизни. Отсортируем все такие события по дню и будем обрабатывать по возрастанию номера дня.

Пусть cur_sum — суммарная стоимость полива всех цветов, которые цветут в текущий момент. Между двумя соседними днями событий набор цветущих растений не меняется, а значит и значение cur_sum остаётся постоянным. Если между событиями прошло d дней, то вклад этого промежутка в ответ равен $d \cdot \min(cur\_sum, k)$.

Особенно аккуратно нужно обработать день самого события. По условию цветок нужно поливать и в день всхода, и в день увядания. Поэтому сначала для текущего дня добавляются все цветы, которые сегодня взошли, затем считается стоимость полива этого дня, и только после этого из суммы убираются цветы, которые сегодня увядают.

Таким образом, мы поддерживаем суммарную стоимость активных цветов и пересчитываем её только в дни событий. Всего событий $2n$, поэтому после сортировки решение работает за $O(n)$. Таким образом, общее время работы — $\mathcal{O}(n \log n)$.

Задача K. Уборка

Разработчики задачи: Константин Бац и Дмитрий Павлов

Поскольку коробку разрешается поворачивать произвольным образом, нас интересуют не конкретные стороны, а лишь их размеры. После любого поворота размеры коробки и контейнера можно сопоставить в некотором порядке.

Заметим, что если отсортировать размеры коробки и контейнера по возрастанию, то для существования подходящего поворота необходимо и достаточно, чтобы каждый размер коробки был строго меньше соответствующего размера контейнера.

Действительно, если после некоторого поворота коробка помещается, то наименьший размер коробки обязательно меньше наименьшего размера контейнера, второй по величине — меньше второго, и так далее. Обратно, если после сортировки выполняются все три строгих неравенства, то можно просто совместить размеры в этом порядке, и коробка поместится.

Таким образом, алгоритм очень прост: сортируем тройки (A_1, B_1, C_1) и (A_2, B_2, C_2), после чего проверяем, что каждый размер коробки строго меньше соответствующего размера контейнера. Сортировка трёх чисел занимает константное время, поэтому сложность решения равна $\mathcal{O}(1)$.

Задача L. Лесные треугольники

Авторы задачи: Сергей Матвеев и Константин Бац, разработчик: Константин Бац

Сначала отсортируем ограничения так, чтобы $a \leqslant b \leqslant c$. Тогда достаточно перебирать значения первых двух сторон i и j, а для третьей стороны k не рассматривать каждое значение отдельно, а сразу подсчитывать количество подходящих значений.

Для остроугольного треугольника достаточно проверить два условия. Во-первых, должны выполняться неравенства треугольника. Во-вторых, квадрат наибольшей стороны должен быть меньше суммы квадратов двух остальных сторон. Для фиксированных i и j удобно обозначить меньшую из них через x, а большую через y.

Далее разобьём все возможные значения k на три случая: $k \leqslant x$, $x \leqslant k \leqslant y$ и $k > y$. В каждом из них порядок сторон уже известен, поэтому условия треугольника и остроугольности превращаются в простые ограничения на отрезок допустимых значений k.

Например, если $k \leqslant x$, то наибольшей стороной является y. Из условий получаем $k + x > y$ и $k^2 + x^2 > y^2$. Обе границы задают минимально допустимое значение k, после чего все большие значения до $\min(x, c)$ подходят.

Аналогично рассматривается случай $x < k \leqslant y$, где наибольшей стороной также остаётся y, и случай $k > y$, где уже самой большой стороной становится k. В каждом из трёх случаев множество подходящих значений образует непрерывный отрезок, поэтому достаточно вычислить его границы и добавить длину этого отрезка к ответу.

Таким образом, для каждой пары (i, j) все подходящие значения k подсчитываются за $\mathcal{O}(1)$. Общая сложность алгоритма составляет $\mathcal{O}(ab)$, что укладывается в ограничения при $a, b, c \leqslant 500$.