Задача A. Бармен Базилий

Автор и разработчик: Леонид Ожегов

Рассмотрим последовательность b_i, в на i-м месте будет стоять количество литров пива, которые попросил i-й математик. Заметим, что $b_i = x^{i-1}$. Значит, b_i — геометрическая прогрессия. $q = \frac{b_i}{b_{i-1}} = \frac{x^{i-1}}{x^{i-2}} = x$, а по условию $x \in (0; 1)$. Значит, b_i — бесконечно убывающая геометрическая прогрессия, а знания из математики подсказывают нам, что сумма такой геометрической прогрессии равна $\sum_{i=1}^{\infty} b_i = \frac{1}{1-q}$.

Значит, для решения задачи надо было считать число x и вывести $\frac{1}{1-x}$ с точностью до 10^{-6}.

Задача B. Системное тестирование

Автор и разработчик: Матвей Кориненко

Заметим, что решения, получившие 0 баллов на претестах не будут перетестированы. Тогда можно убрать их из рассмотрения.

Далее нам необходимо отсортировать оставшиеся решения сначала по возрастанию номера задачи, потом по убыванию количества набранных баллов.

Существует несколько подходов. Сложность решения в каждом из подходов: $O(n \log n)$

Подход 1

Можно использовать std::sort с своим компаратором. Пример компаратора:

```
bool cmp(const pair<int, int>& a, const pair<int, int>& b) {
    if (a.first < b.first) {
        return true;
    } else if (a.first > b.first) {
        return false;
    } else {
        return a.second > b.second;
    }
}
```

Подход 2

Можно заменить количество набранных баллов какой-то величиной так, чтобы по ней необходимо было сортировать по возрастанию. Тогда можно будет отсортировать массив пар по возрастанию. Такой величиной является $-s_i$. Сортируем пары $(t_i, -s_i)$ по возрастанию, потом аккуратно выводим.

Задача C. Социальная сеть

Автор: Матвей Кориненко
Разработчики: Григорий Базилевич, Матвей Кориненко

Сформулируем, что от нас требуется. Для каждой команды *register* необходимо найти и удалить минимальный элемент из множества свободных номеров, а для команды *remove* — вернуть обратно в множество.

Нам нужна структура данных, которая умеет добавлять любой элемент, а также находить и удалять минимальный элемент за время $O(\log n)$. Примерами таких структур являются: куча (std::priority_queue), std::set.

Можно было написать наивную реализацию — хранить для каждого номера его состояние (свободен или занят), тогда запрос добавления работает за $O(n)$, а запрос удаления — за $O(1)$. В худшем случае асимптотика: $O(n^2)$, такое решение получало 78 баллов.

Сложность решения: $O(n \log n)$.

Задача D. Пружинки Виталия Михайловича

Автор и разработчик: Матвей Кориненко

Для начала заметим, что первый раз все k пружинок находятся в растянутом или сжатом состоянии в момент времени $lcm(t_1, t_2, t_3, ..., t_k)$, где lcm — наименьшее общее кратное. Тогда, в общем виде время, когда каждая пружинка полностью растянута или сжата равно $C \times lcm(t_1, t_2, t_3, ..., t_k)$, где C — какая-то целая константа. Все пружинки будут в растянутом состоянии, когда для каждого i от 1 до k будет выполняться условие $\frac{C \times lcm(t_1, t_2, t_3, ..., t_k)}{t_i} \mod 2 = 1$ (т.к. растянутое состояние характеризуется нечетным количеством растяжений и четным количеством сжатий пружины).

Заметим, что если какая-то пружина оказалась в сжатом состоянии (то есть ее общее число сжатий и растяжений является четным), то при любом значении константы C она также будет сжата, т.к. при умножении четного числа на любое другое получается четное число. Тогда нам остается проверить, для всех ли t_i выполняется равенство $\frac{lcm(t_1, t_2, t_3, ..., t_k)}{t_i} \mod 2 = 1$. Если да, то ответ равен $lcm(t_1, t_2, t_3, ..., t_k)$, иначе его нет.

Задача E. Упаковка подарков

Автор и разработчик: Матвей Кориненко

Решим задачу методом динамического программирования.

1. (Состояние динамики) $dp[i]$ — количество способов упаковать i подарков в коробки размерами x, y, z.
2. (Исходные значения) $\forall i : dp[i] = 0$. Увеличим на единицу $dp[x]$, $dp[y]$ и $dp[z]$ (если какие-то из этих чисел совпадают, то значение $dp[i]$ должно быть равно количеству таких чисел).
3. (Переходы между состояниями)

   $$dp[i] = (dp[i - x] + dp[i - y] + dp[i - z]) \mod 1234567$$

   По-другому: количество способов собрать i подарков в коробки равно количеству способов собрать $i - x$, $i - y$ и $i - z$ подарков в коробки, что очевидно (задачу для $i - x$ подарков мы уже решили, а остальные x подарков поместятся в одну коробку. Аналогично для коробок размера y и z). Возможен выход за границы массива, необходима аккуратная реализация.
4. (Порядок пересчета) Логично, что пересчитывать динамику необходимо по возрастанию i из ранее посчитанного. Возможна реализация с обратным порядком пересчета, когда рассматривая $dp[i]$ будут обновляться $dp[i + x]$ и т.д.
5. (Ответ) Ответ хранится в $dp[n]$.

Сложность решения: $O(n)$.

Задача F. Крупный игрок на бирже

Автор: Матвей Кориненко
Разработчики: Григорий Базилевич, Матвей Кориненко

Переформулируем условие: дан массив из n чисел a_i, необходимо найти два индекса i, j $(i < j)$ так, что $a_j - a_i$ максимально, j минимальный из всех возможных, а если для данного j существует несколько i, то выбрать такой, что $j - i$ минимально.

Рассмотрим наивное решение: для каждого индекса j, перебираемого циклом, будем за линейное время искать такой индекс i, что $a_j - a_i$ максимально и $j - i$ минимально, и пытаться улучшить ответ (пару $(i_{ans}; j_{ans})$) заменив его парой $(i; j)$. Фактически, такое произойдет только если $a_j - a_i > a_{j_{ans}} - a_{i_{ans}}$ иначе мы берем не минимальный индекс j, что противоречит условию. Изначально в качестве ответа можно взять пару (0; 0) (указать на один и тот же индекс, получив при

этом выгоду, равную 0. Если после прохода по массиву ответ не изменится, то значит, что ответа с выгодой, большей 0 нет, а значит вкладывать в биткоин бессмысленно, ответ -1. Сложность решения: $O(n^2)$.

Оптимизируем наивное решение. Заметим, что каждый раз перебирая циклом индекс i мы находим одно и то же значение — самое правое вхождение минимального элемента на префиксе $[0; j]$. Пересчитывать данный индекс (назовем его i_{min}) можно при движении индекса j. Получается, мы избавились от вложенного прохода по массиву, что привело нас к сложности решения: $O(n)$.

Задача G. Направляющий в шеренге

Автор и разработчик: Матвей Кориненко

В данной задаче от нас требуется переставить одно число в начало последовательности, сохранив порядок остальных чисел. Задачу можно решать при вводе:

- Считаем и выведем число, которое нам необходимо переставить.
- Будем считывать изначальную перестановку и выводить считанное число, если оно не равно числу, которое нам необходимо переставить.

Сложность решения: $O(n)$

Задача H. Праздничное пирожное

Автор: Матвей Кориненко
Разработчики: Григорий Базилевич, Матвей Кориненко

Построим график зависимости $w(t)$ по точкам, точки соединим отрезками. Заметим, что масса пирожного — площадь под графиком $w(t)$. Разобьем график на треугольники и трапеции перпендикулярами, опушенными из точек на ось X. Тогда площадь под графиком $w(t)$ равна сумме площадей данных трапеций и треугольников, посчитать их сумму можно за $O(n)$. Для успешного прохождения последней подзадачи необходимо было использовать long double в C++.

Задача I. Учим билеты

Автор и разработчик: Матвей Кориненко

Представим задачу в виде ориентированного графа, вершинами которого являются задачи, а ребрами — «зависимости» билетов. Заметим, что если по условию «не существует теорем, которые каким-либо образом взаимно доказываются друг через друга», то граф является ациклическим. А так как «для доказательства теоремы может использоваться не более одного доказательства другой теоремы», то граф представляет из себя множество «деревьев»(«лес»), в котором ребра идут не от предка к потомку, а от потомка к предку.

Для удобства инвертируем все ребра, чтобы они шли от предка к потомку. Тогда ответом для каждой вершины является отношение размера ее поддерева к общему количеству вершин в графе, умноженное на 100.

Для получения ответа на эту задачу можно запускать обход в глубину для каждой вершины, в котором для вычисления размера поддерева данной вершины будем суммировать размеры поддеревьев ее потомков.

Сложность решения $O(n + m)$.