

Задача А. Распределение нагрузки

Рассмотрим два случая:

- Бегимай идет читать в направлении L :
 - Виктор точно прочтёт все задачи от n -й до s -й не включительно — всего $(n - s)$.
 - Айдар и Бегимай прочтут одинаковое количество задач, равное $\lceil \frac{s}{2} \rceil$
- Бегимай идет читать в направлении R :
 - Айдар точно прочтёт все задачи от 1 до s -й не включительно — всего $(s - 1)$.
 - Бегимай и Виктор прочтут одинаковое количество задач, равное $\lceil \frac{n-s+1}{2} \rceil$

P.S. Запись $\lceil \frac{x}{2} \rceil$ обозначает целочисленное деление x на 2 с округлением **вверх**. Оно может быть вычислено следующими способами:

- либо с применением условного оператора на «делится / не делится»;
- либо как $(x \text{ div } 2) + (x \text{ mod } 2)$;
- либо как $(x + 1) \text{ div } 2$.

где div — целочисленное деление, а mod — остаток от деления.

P.P.S. Подумайте, как будет выглядеть формула в общем виде для $\lceil \frac{x}{y} \rceil$.

Задача В. Что решать дальше?

Сразу обратим внимание, что все входные данные разделялись пробелами, что позволяло либо читать поэлементно (`std::cin` в C++), либо разделять уже введенную строку (`split` в Java, C#, Python и т.д.).

Сама задача не представляла какой-то идейной или реализационной сложности:

- Будем хранить для каждой задачи три счётчика:
 - S_j — количество решивших задачу j ;
 - T_j — количество пытавшихся решить задачу j ;
 - F_j — суммарное количество попыток по задаче j .
- Для результата каждой команды по задаче j будем проверять следующие условия:
 - F_j всегда увеличиваем на указанное количество попыток.
 - Если команда сдала задачу, то увеличиваем счётчики S_j и T_j на 1.
 - Если команда не сдала задачу, но сделала хотя бы одну попытку, то увеличиваем счётчик T_j на 1.

Задача С. Целочисленное переполнение

Решение можно разделить на две части:

- Вычислить «минимальный» тип для каждой из переменных A , B , $C = A \cdot B$.
- Адаптировать тип переменной A или B для соответствия типу C при необходимости.

Рассмотрим первый шаг — вычисление «минимального» типа:

- Для переменных A и B это было проделать несложно — просто сравнить значения с заданными в условии константами.

- Обратите внимание, что в языках C++, Java, C# и т.д. необходимо было использовать 64-битный тип как для хранения переменных A и B , так и для хранения констант — границ типов.
- Также важно помнить, что целочисленные константы в таких языках по умолчанию имеют 32-битный тип, поэтому необходимо было использовать суффикс L в Java / C# или суффикс LL в C++.
- На языках с длинной арифметикой (например, Python) проверка для переменной C была такой же, как и для A и B .
- На языках без длинной арифметики (например, C++) можно было воспользоваться утверждением:

Из $A \cdot B \leq X$ следует, что $A \leq \lfloor \frac{X}{B} \rfloor$, после чего подставить в качестве X верхние границы для 32-битного и 64-битного типов.

Так как все значения были положительными, то дополнительных проверок и условий не требовалось.

Рассмотрим второй шаг — адаптация типов A или B для соответствия типу C :

- Выберем больший из типов для A и B .
- Сделаем только его равным типу C , не изменяя меньший из типов.
- Таким образом мы и выполним необходимое условие, и достигнем оптимальной суммы битностей:
 - если типы A и B равны, то нет разницы, какой увеличивать;
 - если типы A и B не равны (то есть 32 и 64 в каком-то порядке), то:
 - * либо тип C уже равен 64 и ничего не изменится;
 - * либо тип C равен 128 — и сумма $32 + 128 + 128$ будет меньше, чем $128 + 64 + 128$.

Задача D. TL, ML или OK?

В задаче требовалось аккуратно провести вычисления по формулам, описанным в условии:

- количество операций на одной итерации равнялось $(2 \cdot q + 5 \cdot k)$;
- количество чисел, добавляемых на одной итерации равнялось k .
- для вычисления суммарных количеств необходимо было обе величины умножить на n .

P.S. Конечно же, необходимо было использовать 64-битный тип для вычислений (привет задаче C);

Задача E. Итоговые места

Решение можно разделить на три подзадачи:

- определить для каждой команды итоговое количество решённых задач и суммарное штрафное время;
- отсортировать команды в правильном порядке;
- вычислить итоговые места команд.

Первая часть решения — определение результатов для каждой команды:

- Для каждой пары (команда i , задача j) будем хранить две величины:
 - $AC_{i,j}$ — решена ли уже командой i задача j или ещё нет;
 - $PN_{i,j}$ — суммарное штрафное время, которое команда i получит, если сдаст задачу j .
- Если текущая посылка в логе сделана по уже решённой задаче — ничего не делаем вовсе.
- Если посылка неудачная, то увеличиваем $PN_{i,j}$ на 20 — указанное в условии штрафное время за неверную попытку.
- Если посылка удачная, то увеличиваем $PN_{i,j}$ на указанное в логе количество минут, после чего обновляем флаг $AC_{i,j}$.

Вторая часть решения — сортировка команд:

- Так как команд было не более 10^3 , то подходил любой алгоритм сортировки — как за $O(n^2)$, так и за $O(n \cdot \log n)$.
- Компаратор, используемый в задаче довольно стандартный — лексикографическое сравнение кортежей в порядке:
 - S по убыванию;
 - X по возрастанию;
 - ID по возрастанию.
- В языках C++ и Python можно было воспользоваться встроенными типами кортежей / массивов, для которых лексикографическое сравнение по возрастанию реализовано по умолчанию (S необходимо было хранить с отрицательным знаком).
- Подробнее про компараторы и сортировки в языках C++, Java, Python вы можете прочесть в данном наборе лекций: https://github.com/Slamur/lectures/tree/main/ru/DSA/data_structures/sorting (часть курса «Структуры данных»).

Третья часть решения — определение итоговых мест:

- У команды на позиции 1 в отсортированном порядке место равно 1 по определению.
- Для определения места команды на позиции i в отсортированном порядке:
 - рассмотрим пару (S_i, X_i) ;
 - сравним её на **равенство** с парой (S_{i-1}, X_{i-1}) :
 - * если пары равны — место у i -й команды совпадает с местом $(i-1)$ -й команды;
 - * иначе — место равно i .
- Места можно было хранить как в отдельном массиве, так и как часть кортежа из второй части решения.

Задача F. Компромисс

Для начала обратим внимание на следующий факт: символы в ответе, стоящие на позициях i и $n-i+1$, всегда будут равны (по определению палиндрома) и зависеть только от символов S_i , T_i , S_{n-i+1} , T_{n-i+1} .

Для данной задачи существовало два подхода к определению оптимального символа в ответе:

- либо перебрать все варианты;
- либо вычислить.

Первый способ решения — перебор всех вариантов:

- Заметим, что в латинском алфавите всего $A = 26$ символов.
- Переберём предполагаемую позицию в алфавите для символа в ответе (от 1 до A).
- Вычислим суммарное расстояние от четвёрки символов в исходных строках до текущего (используя ASCII-коды символов).
- Итоговая сложность такого решения будет $O(n \cdot A)$.

Второй способ решения — вычисление оптимального символа:

- Обозначим через c_1, c_2, c_3, c_4 исходную четвёрку символов в порядке **возрастания** их позиций в алфавите (ASCII-кодов).
- Утверждается, что оптимальным будет любой символ из промежутка $[c_2, c_3]$ — будем использовать всегда либо c_2 , либо c_3 .
- Итоговая сложность такого решения будет $O(n \cdot k \log k)$, где $k = 4$.

P.S. Для языков Python, Java, C# и т.д. необходимо было помнить, что посимвольное увеличение строки в цикле не являлось асимптотически эффективным — необходимо использовать либо классы наподобие `StringBuilder`, либо собирать ответ в динамический массив и использовать методы наподобие `join`.

P.P.S. Данная задача является частным случаем задачи о **минимизации суммарного расстояния на прямой** является классической.

Известно, что её решением является **медиана** массива — элемент, стоящий в середине массива в **сортированном порядке**.

В случае чётного количества элементов — любая точка в промежутке между двумя медианами. Приведём простое доказательство (хотя и не совсем формальное):

- Расположим ответ на прямой в промежутке между точками x_i и x_{i+1} , обозначим суммарное расстояние через S .
- Переместим ответ на 1 влево — S уменьшится на i (так как до первых i точек ответ стал ближе), но увеличится на $(n - i)$ (так как до последних $(n - i)$ точек ответ стал дальше).
- Аналогично рассмотрим перемещение ответа на 1 вправо.
- Заметим, что ответ будет в оптимальной позиции, если перемещение и влево, и вправо будет увеличивать S .
- Легко понять, что оптимум достигается, когда $|i - (n - i)| \leq 1$, откуда следует, что $|2 \cdot i - n| \leq 1$ — отсюда легко получить утверждения, описанные выше.

Задача G. $A + B = C$

Ограничение 3 на длину строк SA и SB (и, соответственно, верхнее ограничение 999 на A и B) позволяло решить данную задачу простым перебором всех возможных вариантов за $O(10^{|SA|+|SB|} \cdot (|SA| + |SB| + |SC|))$:

- Переберём двумя вложенными циклами все возможные пары A и B , подходящие по количеству цифр.

- Вычислим ожидаемое значение $C = A + B$.
- Проверим соответствие строковых масок зафиксированным числам и отсутствие нарушение биекции «одна буква — одна цифра»:
 - заведём два словаря — один будет хранить ожидаемую цифру для буквы, другой — ожидаемую букву для цифры;
 - пробежимся по очереди по числам и проверим, что для текущей пары (буква, цифра) не было ранее найдено иных соответствий в словарях;
 - ведущие нули будут обработаны сами собой, так как числа A , B и C их не содержат по построению.

Задача Н. Параллельная проверка

Обозначим через $F(d)$ суммарное количество квантов времени при фиксированном количестве потоков d :

$$F(d) = \sum \lceil \frac{A_i}{d} \rceil.$$

Обозначим через $G(d)$ предикат $F(d) \leq T$.

Наша цель — найти минимальный d такой, что $G(d)$ принимает истинное значение.

Обратим внимание, что $F(d) \geq F(d+1)$ для всех $d \geq 1$, откуда следует, что $F(d)$ — монотонно невозрастающая функция.

Из монотонности $F(d)$ легко понять, что:

- если $G(d)$ истинно, то и $G(d+1)$ истинно;
- если $G(d)$ ложно, то и $G(d-1)$ ложно.

Значит функция $G(d)$ тоже монотонная, причём существует такой D , что:

- для всех $0 < d < D$ предикат $G(d)$ является ложным;
- для всех $D \leq d$ предикат $G(d)$ является истинным.

Воспользуемся техникой **двоичного поиска по ответу**, чтобы найти величину D за $O(\log \max A \cdot n)$.