Задача 1. Сикс-севен-дзидай

Заметим два важных свойства из условия задачи:

- Каждое из трёх действий (перекуп ронинов, найм сказителя, эвакуация святынь) можно совершить не более одного раза.
- Слава за «Песнь павшим» начисляется при любом исходе битвы, тогда как слава за эвакуацию святынь начисляется только при поражении.

Далее предполагается, что все стоимости и награды неотрицательны (если в задаче возможны отрицательные значения славы, следует использовать Вариант 1 — он корректно обрабатывает любые знаки чисел).

Вариант 1. Полный перебор состояний (рекомендуемый)

Поскольку для каждого из трёх действий есть ровно два варианта (совершать или нет), общее число возможных стратегий равно $2^3 = 8$. Мы можем перебрать все 8 комбинаций и выбрать лучшую среди допустимых по бюджету.

Шаги алгоритма:

1. Переберём три бинарных флага $\text{buy_ronin}, \text{buy_song}, \text{buy_esc} \in \{0, 1\}$. Для текущей комбинации вычислим суммарные затраты:

$$\text{cost} = \text{buy_ronin} \cdot (M_2 + 1) + \text{buy_song} \cdot C_{\text{song}} + \text{buy_esc} \cdot C_{\text{esc}}.$$

   Если $\text{cost} > M_1$, монет не хватает, и данный вариант пропускается.

2. Определим итоговые силы сторон: если $\text{buy_ronin} = 1$, наша сила равна $F_1 + R$, а сила врага — F_2; иначе наша сила равна F_1, а сила врага — $F_2 + R$.

3. Подсчитаем итоговую славу glory:
   - Если наша итоговая сила больше или равна силе врага, мы побеждаем и прибавляем V_{prov}.
   - В противном случае (при поражении), если $\text{buy_esc} = 1$, прибавляем V_{esc}.
   - Независимо от исхода битвы, если $\text{buy_song} = 1$, прибавляем $(F_1 + F_2 + R) \cdot V_{\text{song}}$.

4. Обновим ответ: $\text{ans} = \max(\text{ans}, \text{glory})$ (изначально $\text{ans} = 0$).

Вариант 2. Разбор случаев

Обозначим $V_{\text{song_total}} = (F_1 + F_2 + R) \cdot V_{\text{song}}$. Инициализируем ответ значением $\text{ans} = 0$ (соответствует отказу от всех трат) и рассмотрим возможные сценарии:

1. **Победа без перекупа ронинов ($F_1 \geqslant F_2 + R$).** Клан гарантированно побеждает и получает V_{prov}. Перекупать ронинов или готовить эвакуацию бессмысленно, поэтому при $C_{\text{song}} \leqslant M_1$ дополнительно нанимаем сказителя:

$$\text{ans} = V_{\text{prov}} + [C_{\text{song}} \leqslant M_1] \cdot V_{\text{song_total}}.$$

2. **Исходной силы не хватает ($F_1 < F_2 + R$).** Переберём подходящие стратегии и возьмём максимум из доступных по бюджету:
   - *Попытка победить с помощью ронинов:* если $M_1 \geqslant M_2 + 1$ **и** $F_1 + R \geqslant F_2$, перекупаем ронинов и получаем V_{prov}. Если после этого хватает остатка средств на сказителя ($M_1 - (M_2 + 1) \geqslant C_{\text{song}}$), прибавляем $V_{\text{song_total}}$.
   - *Отказ от ронинов (поражение):* проверяем все допустимые комбинации песни и эвакуации:
     - при $M_1 \geqslant C_{\text{esc}}$ получаем V_{esc};
     - при $M_1 \geqslant C_{\text{song}}$ получаем $V_{\text{song_total}}$;

– при $M_1 \geqslant C_{\text{song}} + C_{\text{esc}}$ получаем $V_{\text{song_total}} + V_{\text{esc}}$.

Асимптотика и детали реализации

Оба алгоритма работают за $O(1)$ времени и используют $O(1)$ дополнительной памяти.

Замечание: при больших ограничениях на входные данные произведение $(F_1 + F_2 + R) \cdot V_{\text{song}}$, а также суммарная стоимость действий могут превысить $2^{31} - 1$. Во избежание переполнения при реализации на C++ следует использовать 64-битный целочисленный тип `long long`.

Задача 2. B2B AI SaaS Six-Seven Startup

Заметим, что из любого числа можно получить 67 тремя последовательными действиями:

1. $x \to x \bmod 67$ (гарантированно получаем число меньше 67),
2. $x \to \lfloor x/67 \rfloor$ (гарантированно получаем 0),
3. $x \to x + 67$ (гарантированно получаем 67).

Можно полностью перебрать все возможные последовательности действий длины 0, 1, 2 (всего их 21) и проверить, не получится ли число 67 меньше, чем за 3 действия. Иначе, вывести 3.

Таким образом, временная сложность решения $O(21Q)$.

Задача 3. Великая чайная церемония

Основная идея

Как подсказывает эпиграф к задаче, наиболее выгодная стратегия — использовать силу самого сильного из доступных кланов для того, чтобы «вытянуть» до уровня Великого союза как можно более слабый клан, не растрачивая сильные дома друг на друга без необходимости.

Отсортируем массив сил кланов по неубыванию: $A_1 \leqslant A_2 \leqslant \ldots \leqslant A_N$. Заведём два указателя: $l = 1$ (на самый слабый из ещё не рассмотренных кланов) и $r = N$ (на самый сильный из ещё не рассмотренных кланов). Пока $l < r$, будем рассматривать пару (l, r):

- Если $A_l + A_r \leqslant K$, то клан l слишком слаб: даже в паре с самым сильным из оставшихся кланов r он не образует Великий союз. Так как массив отсортирован, для любого другого доступного клана j $(l < j \leqslant r)$ выполняется $A_l + A_j \leqslant A_l + A_r \leqslant K$. Следовательно, клан l не сможет войти ни в один Великий союз среди оставшихся кланов. Исключим его из рассмотрения, сдвинув левый указатель: $l \leftarrow l + 1$.
- Если $A_l + A_r > K$, то кланы l и r могут образовать Великий союз. Объединим их в пару, увеличим ответ на 1 и сдвинем оба указателя: $l \leftarrow l + 1$, $r \leftarrow r - 1$.

Оставшиеся после завершения цикла кланы (если таковые есть) не могут образовать ни одного Великого союза; их можно разбить на произвольные обычные пары, чтобы общее число союзов равнялось $\lfloor \frac{N}{2} \rfloor$.

Доказательство корректности

Покажем по индукции, что на каждом шаге для текущего подотрезка $[l, r]$ существует оптимальный набор Великих союзов, согласованный с выбором жадного алгоритма.

1. *Случай* $A_l + A_r \leqslant K$. Как показано выше, ни один Великий союз на отрезке $[l, r]$ не может содержать элемент l. Значит, переход к отрезку $[l+1, r]$ не уменьшает максимально возможное число Великих союзов.
2. *Случай* $A_l + A_r > K$. Возьмём произвольный оптимальный набор Великих союзов S на отрезке $[l, r]$. Докажем, что его можно перестроить так, чтобы он содержал пару (l, r), не уменьшив размер S:
   - Если $(l, r) \in S$, перестройка не требуется.
   - Если ни l, ни r не входят ни в один Великий союз из S, мы можем добавить пару (l, r) в S и увеличить ответ, что противоречит оптимальности S.

- Если ровно один из элементов (например, l) входит в Великий союз $(l, x) \in S$, а r свободен, заменим (l, x) на (l, r). Поскольку $A_l + A_r > K$, новый союз также является Великим, а размер S не изменился. Аналогично поступаем, если занят только r.
- Если оба элемента входят в различные Великие союзы: $(l, x) \in S$ и $(y, r) \in S$, где $l < x, y < r$ и $x \neq y$. Так как l — минимальный элемент на $[l, r]$, выполнено $A_y \geqslant A_l$. Прибавив A_x к обеим частям неравенства и воспользовавшись тем, что (l, x) — Великий союз, получаем:

  $$A_y + A_x \geqslant A_l + A_x > K.$$

  Следовательно, пара (y, x) также образует Великий союз. Заменив в S две пары (l, x) и (y, r) на (l, r) и (y, x), мы получим корректный набор Великих союзов того же размера, содержащий пару (l, r).

Зафиксировав пару (l, r) в оптимальном ответе, мы сводим задачу к отрезку $[l + 1, r - 1]$.

Асимптотика и детали реализации

Сортировка массива занимает $O(N \log N)$ времени, а проход двумя указателями выполняется за $O(N)$. Итоговая временная сложность алгоритма составляет $O(N \log N)$, дополнительная память — $O(1)$ (без учёта хранения входного массива).

Замечание: при больших ограничениях на A_i и K (например, до 10^9 и выше) сумма $A_l + A_r$ может превысить $2^{31} - 1$, поэтому для хранения сил кланов и значения K следует использовать 64-битный целочисленный тип данных (`long long` в C++).

Задача 4. Магические кристаллы

Подзадача 2 ($X = 0$ или $X = N - 1$)

При $X = 0$ применяются только заклинания «Минимус», поэтому ответ равен минимальному элементу массива. При $X = N - 1$ применяются только заклинания «Максимус», поэтому ответ равен максимальному элементу массива.

Подзадача 3 ($N \leqslant 10$)

Переберём все последовательности действий с помощью рекурсивного перебора. На каждом шаге выберем любую соседнюю пару, заменим её максимумом или минимумом в зависимости от номера операции и продолжим перебор для получившегося массива.

Когда останется один элемент, сравним его с лучшим найденным ответом. Всего существует $(N - 1)!$ последовательностей выбора соседних пар, поэтому при $N \leqslant 10$ такой перебор успевает отработать.

Подзадачи 4 и 5

Заклинание «Максимус» фактически удаляет меньший из двух выбранных элементов. После всех таких операций останется $N - X$ элементов, а последующие заклинания «Минимус» дадут минимум среди них.

Поэтому выгоднее всего удалить X наименьших элементов массива. Это всегда возможно: найдём минимальный элемент текущего массива и применим «Максимус» к нему и любому его соседу. В результате удалится именно минимальный элемент. При равных значениях удалится одно из их вхождений.

Повторим это действие X раз, после чего выведем минимум оставшегося массива. Каждый поиск минимума и удаление занимают $O(N)$, поэтому решение работает за $O(N^2)$.

Подзадача 6 ($N \leqslant 2 \cdot 10^5$)

Из решения предыдущей подзадачи следует, что ответ — минимальный элемент, который останется после удаления X наименьших элементов массива.

Вместо последовательных удалений отсортируем массив по неубыванию и выведем элемент на позиции $X + 1$ при нумерации с единицы.

Сложность решения составляет $O(N \log N)$.

Задача 5. Опять все перепутали...

Назовём позицию *неправильной*, если символ текущей строки A в ней отличается от соответствующего символа строки B, и *правильной* в противном случае. Обозначим количество неправильных позиций через D. Каждое изменение символа меняет правильность позиции на противоположную.

Подзадача 2 ($N \leqslant 7$)

Если $D = 0$, ответ равен нулю.

При $N < 6$ нельзя выполнить ни одной операции, поэтому при несовпадающих строках ответ равен -1.

При $N = 6$ можно только изменить все позиции одновременно. Следовательно, при $D = 6$ достаточно одной операции, а при $1 \leqslant D \leqslant 5$ преобразование невозможно.

При $N = 7$ каждая операция изменяет либо все позиции, либо все позиции, кроме одной. Оптимальные последовательности можно выписать по значению D:

- $D = 1$: изменить все позиции, затем все, кроме единственной неправильной в исходной строке;
- $D = 2$: для каждой неправильной в исходной строке позиции изменить все позиции, кроме неё;
- $D = 3$: изменить все позиции, затем для каждой неправильной в исходной строке позиции изменить все, кроме неё;
- $D = 4$: для каждой правильной в исходной строке позиции изменить все позиции, кроме неё;
- $D = 5$: изменить все позиции, затем для каждой правильной в исходной строке позиции изменить все, кроме неё;
- $D = 6$ или $D = 7$: изменить все неправильные позиции одной операцией.

Подзадача 3 ($N \leqslant 8$)

Рассмотрим граф всех строк длины N из символов `6` и `7`. Вершины графа — строки, а ребро соединяет две строки, если одна получается из другой за одну допустимую операцию. Всего в графе 2^N вершин.

В простейшей реализации для каждой обрабатываемой вершины переберём все 2^N строки и проверим, существует ли соответствующее ребро. Для этого нужно убедиться, что строки отличаются ровно в шести или семи позициях. Проверка занимает $O(N)$.

Для нахождения оптимальной последовательности операций запустим обход в ширину из строки A. Для восстановления ответа сохраним предыдущую вершину для каждой посещённой строки. Если строка B недостижима, выведем -1.

Время работы такой реализации составляет $O(N \cdot 4^N)$.

Подзадача 4 ($N \leqslant 15$)

Ускорим предыдущее решение с помощью битовых масок. Единичные биты маски состояния будут соответствовать неправильным позициям. Целевое состояние — нулевая маска.

Заранее сохраним все маски операций, содержащие ровно шесть или семь единичных битов. Применение операции с маской P к состоянию M даёт состояние $M' = M \oplus P$, где $\oplus$ — побитовое исключающее ИЛИ.

Теперь каждый переход вычисляется одной битовой операцией. Остаётся выполнить BFS и восстановить ответ по предыдущим состояниям.

С учётом предварительного перебора масок время работы составляет

$$O\left(2^N \cdot \left(N + \binom{N}{6} + \binom{N}{7}\right)\right).$$

При $N = 15$ полный обход графа потребует около 375 миллионов операций.

Подзадача 5 ($N \leqslant 2000$)

Главное наблюдение, которое нужно было сделать для решения следующих подзадач: номера неправильных позиций не важны, важно только их количество. Действительно, операция разрешает выбирать любые позиции. Если две текущие строки имеют одинаковое число несовпадений с B, то

в обеих можно выбрать одинаковое количество правильных и неправильных позиций. После этого число несовпадений снова будет одинаковым. Следовательно, возможности дальнейших преобразований определяются только значением D.

Рассмотрим операцию размера $K \in \{6, 7\}$. Пусть она затрагивает X неправильных и $K - X$ правильных позиций. Первые станут правильными, вторые — неправильными, поэтому

$$D' = D + K - 2X.$$

Для выполнения операции должно хватать позиций каждого типа:

$$0 \leqslant X \leqslant D, \qquad 0 \leqslant K - X \leqslant N - D.$$

Таким образом, допустимые значения X задаются границами

$$\max(0, K - (N - D)) \leqslant X \leqslant \min(K, D).$$

Получаем граф с вершинами $0, 1, \ldots, N$, обозначающими количество несовпадений между текущей и желаемой строками. Из каждой вершины перебираем $K \in \{6, 7\}$ и допустимые значения X, переходя в вершину $D + K - 2X$. Если допустимых значений X нет, переходы для такого K отсутствуют.

Каждый переход можно выполнить, выбрав любые подходящие позиции. Поэтому кратчайший путь от исходного D до нуля задаёт оптимальное количество операций. Для поиска такого пути снова используем BFS. Из вершины выходит не более 15 переходов, поэтому поиск работает за $O(N)$. Если ноль недостижим, ответ равен -1.

Для восстановления ответа сохраним предыдущие состояния и параметры переходов (K, X). Затем будем двигаться по найденному пути, каждый раз просматривая текущую строку и выбирая X неправильных и $K - X$ правильных позиций. Кратчайший путь содержит не более N операций, а восстановление каждой занимает $O(N)$. Итоговая сложность — $O(N^2)$.

Подзадача 6 ($N \leqslant 2 \cdot 10^5$)

Используем тот же BFS по количеству несовпадений, но ускорим восстановление операций.

Будем хранить номера неправильных позиций в массиве `BAD`, а правильных — в массиве `GOOD`. Для операции (K, X) извлечём с конца первого массива X позиций, а с конца второго — $K - X$ позиций.

После операции выбранные неправильные позиции перенесём в `GOOD`, а выбранные правильные — в `BAD`. Важно сначала извлечь все позиции и только затем переносить их между массивами, чтобы не выбрать одну позицию дважды в одной операции.

Поскольку $K \leqslant 7$, восстановление одной операции занимает $O(1)$ времени. Операций не более N, поэтому всё решение работает за $O(N)$.

Дополнительные наблюдения

Любое целое число $D \geqslant 30$ представимо суммой шестёрок и семёрок. Таким образом, случаи, где $D \geqslant 30$, можно решить, просто набрав необходимую сумму, максимизируя количество семёрок. При $D < 30$ решать с помощью BFS.

Можно построить и полностью конструктивное решение без BFS, например при $N \geqslant 12$ минимальное количество операций определяется формулой:

$$\begin{cases} 0, & D = 0, \\ 1, & D \in \{6, 7\}, \\ \max\left(2, \left\lceil \frac{D}{7} \right\rceil\right), & \text{иначе.} \end{cases}$$

Задача 6. Странный калькулятор

Подзадачу 2 можно было решать динамическим программированием. Пусть $dp[i]$ — количество способов получить число i. База: $dp[1] = 1$. Чтобы получить $dp[i]$, нужно просуммировать все такие

$dp[j]$, что из числа j одним нажатием можно получить число i. В качестве j подходят только числа вида $\frac{i}{k}$, где $2 \leqslant k \leqslant 9$, и при этом i нацело делится на k. Ответом для пары чисел l, r будет сумма $\sum_{i=l}^{r} dp[i]$ Чтобы решение проходило по времени, во-первых, необходимо заранее предподсчитать dp, а для быстрого ответа на каждый запрос предподсчитать значения префиксных сумм для dp.

Для решения подзадачи 3 можно для каждого запроса перебирать все числа $l \leqslant i \leqslant r$, и надо уметь быстро вычислять количество способов получить число. Далее заметим очень важное наблюдение, которое понадобится и далее: а именно, что при умножении на число от 2 до 9 из простых делителей получившегося числа разве что могли быть добавлены только $2, 3, 5, 7$. Значит, любое число, которое может получиться на экране, имеет разложение $2^a 3^b 5^c 7^d$, причём a, b, c, d могут быть нулевыми. Так, например, можно проверить на соответствие числа i заданной форме, деля на $2, 3, 5, 7$. Если заданной формы, то можно рекурсивно считать количество способов, с запоминанием ранее подсчитанных значений.

Для полного решения поймём, почему количество чисел, которые могут получиться на экране (как мы уже знаем, это числа вида $2^a 3^b 5^c 7^d$) до 10^{18} невелико.

Заметим, что если $a \geqslant 60$, то $2^a 3^b 5^c 7^d \geqslant 2^a \geqslant 2^{60} > 10^{18}$.

Если $b \geqslant 40$, то $2^a 3^b 5^c 7^d \geqslant 3^b \geqslant 3^{40} > 10^{18}$.

Если $c \geqslant 30$, то $2^a 3^b 5^c 7^d \geqslant 5^c \geqslant 5^{30} > 10^{18}$.

Если $d \geqslant 24$, то $2^a 3^b 5^c 7^d \geqslant 7^d \geqslant 7^{24} > 10^{18}$.

Вот уже получили верхнюю оценку в $60 \times 40 \times 30 \times 24 = 1728000$ чисел. В действительности такая оценка довольна грубая, так как не учитывает зависимости между a, b, c, d. Несложно проверить, что реальное количество чисел такого вида равно 66061. На полное решение надо было предподсчитать все возможные пары (число на экране, количество способов), затем упорядочить их по возрастанию по первому элементу пары. Далее ответом на любой запрос будет сумма вторых чисел пар какого-то подотрезка. Чтобы для каждого запроса быстро считать суммы на подотрезке, нужно предподсчитать значения префиксных сумм количеств способов, а границы отрезка для каждого запроса можно искать бинарным поиском по первому числу пары. Осталось научиться быстро предподсчитывать все возможные пары. Это можно было сделать, например, предподсчитав 4-мерную динамику $dp[a][b][c][d]$ — количество способов получить на экране число, равное $2^a 3^b 5^c 7^d$, размеры dp определены оценками выше. Можно было сделать и альтернативным способом: решать задачу динамикой "вперёд", а именно по уже известному значению $dp[i]$ обновить значения чисел, которые можно получить из i за один шаг, а именно $dp[i * k]$, где $2 \leqslant k \leqslant 9$. Для последнего способа можно было даже не проводить рассуждения с возможным представлением числа на экране, а просто взять "на веру", что количество возможных чисел на экране — невелико.

Для прохождения подзадач 4 и 5 можно было использовать некоторые допущения. Например, для подзадачи 5 не использовать бинарный поиск и префиксные суммы количеств способов, а просто для каждого запроса пройтись по всем парам, и учитывать только подходящие для данного запроса.

Задача 7. Нормы гроссмейстера

Для того чтобы найти максимальное количество подходящих отрезков (попарно непересекающихся и каждый из которых представляет собой норму), будем действовать жадным алгоритмом: а именно, идти по партиям слева направо и на каждом шаге брать подходящий отрезок, представляющий собой норму и не пересекающийся с уже ранее взятыми отрезками, с наименьшим правым концом. Таким образом, можно считать динамику $dp[i]$ — максимальное количество норм, которые можно получить, если в рассмотрение можно брать только первые i партий. Переходы тут довольно понятные: либо мы не берём i-ю партию, и тогда получим переход из $dp[i]$, либо мы берем i-ю партию в какую-то норму. Пусть эта норма содержит партии с j-й по i-ю включительно. Тогда количество партий будет равно $dp[j-1]+1$. Значение $dp[i]$ будет равно максимуму по всем переходам. Ответ на задачу будет лежать в $dp[n]$.

Для решения *подзадачи* 2 можно для каждого i перебрать все $j < i$ и затем проверять отрезок партий с j-й по i-ю на соответствие норме.

Такое решение будет работать за $O(n^3)$.

Для решения *подзадачи* 3 достаточно для каждого i перебирать $j < i$ от большего к меньшему,

тем самым добавляя по одной партии и быстро проверяя отрезок на соответствие норме.

Такое решение будет работать за $O(n^2)$.

Для решения *подзадачи* 4 можно заметить, что для каждого i достаточно проверить лишь $k_{max} - k_{min} + 1$ вариантов, с количеством партий от k_{min} до k_{max}. Чтобы быстро проверить отрезок партий на наличие достаточного среднего рейтинга соперников и на количество побед, можно предподсчитать значения префиксных сумм рейтингов и значения префиксных сумм результатов (по очкам). С помощью этого каждый отрезок будет проверяться за $O(1)$.

Такое решение будет работать за $O(n * (k_{max} - k_{min}))$.

Для решения следующих подзадач переформулируем условия на средний рейтинг соперников и на процент очков. Чтобы соблюдалось ограничение для отрезка партий с j-й по i-ю, должно соблюдаться условие:

$$\frac{\sum_{ind=j}^{i} r_{ind}}{i - j + 1} \geqslant R_{avg}.$$

Перепишем:

$$\sum_{ind=j}^{i} r_{ind} \geqslant R_{avg} \times (i - j + 1)$$

$$\sum_{ind=1}^{i} r_{ind} - \sum_{ind=1}^{j-1} r_{ind} \geqslant R_{avg} \times i - R_{avg} \times (j - 1)$$

$$\sum_{ind=1}^{i} r_{ind} - R_{avg} \times i \geqslant \sum_{ind=1}^{j-1} r_{ind} - R_{avg} \times (j - 1)$$

Заметим, что слева и справа стоят схожие выражения, отличие только в том, что слева i, а справа $j - 1$.

Тогда введем:

$$prsum_r[i] = \sum_{ind=1}^{i} r_{ind} - R_{avg} \times i$$

Условие теперь будет выглядеть так: $prsum_r[i] \geqslant prsum_r[j - 1]$

Для процента очков, рассуждая аналогично, введем:

$$prsum_s[i] - \sum_{ind=1}^{i} score_{ind} - p \times i$$

Здесь $score_{ind} = 100$, если $s[ind] = W$, $score_{ind} = 50$, если $s[ind] = D$ и $score_{ind} = 0$, если $s[ind] = L$.

Для решения *подзадачи* 5 можно заметить, что при $p = 100$ можно разбить партии на последовательности из одних побед, убрав партии с ничьими и поражениями, а затем просуммировать значения по всем таким последовательностям. После этого для каждой последовательности исчезает ограничение на процент очков, остаются ограничения только на количество партий и средний рейтинг соперников. Теперь можем идти жадно, как описано выше, и для того, чтобы проверить, можем ли добавить следующую норму, заканчивающуюся этой партией, достаточно проверить, будет ли $prsum_r[i]$ больше или равен хотя бы одному значению $prsum_r[j - 1]$, или, другим языком, проверить, будет ли $prsum_r[i]$ больше или равен минимуму $prsum_r$ на соответствующем отрезке (по j). Для быстрого нахождения минимума на отрезке можно, например, построить дерево отрезков на минимумах или разреженные таблицы.

Решение будет работать за $O(n \times log(n))$.

Для решения *подзадачи* 6 можно заметить, что при упорядоченных рейтингах соперников, если отрезок с j-й по i-ю партию подходит по среднему рейтингу, то и отрезок с $j+1$-й по i-ю партию также будет подходить по среднему рейтингу. Это так, потому что, убирая слабейшего соперника (а он с номером j), средний рейтинг соперников не уменьшается. Будем действовать жадным алгоритмом, как описано в самом начале. Для партии i, среди всех отрезков, заканчивающихся на эту партию, ограничение на средний рейтинг соперников будет соблюдаться только на отрезках с левой границей не меньше определенной. Получаем, что все отрезки, для которых соблюдаются условия на k_{min}, k_{max} и на средний рейтинг соперников, все их левые границы образуют непрерывный отрезок. Аналогично подзадаче 5, чтобы проверить, можем ли добавить следующую норму, заканчивающуюся этой партией, достаточно проверить, будет ли $prsum_s[i]$ больше или равен минимуму $prsum_s$ на соответствующем отрезке (по j).

Решение будет работать за $O(n \times log(n))$.

Для *полного* решения будем использовать тот же жадный алгоритм, что и раньше. Отрезок с j-й по i-ю партию будет подходить, если соблюдаются условия: $k_{min} \leqslant i - j + 1 \leqslant k_{max}$, $prsum_r[j-1] \leqslant prsum_r[i]$, $prsum_s[j-1] \leqslant prsum_s[i]$. Заметим, что от $prsum_r$ и $prsum_s$ нам достаточно лишь уметь их сравнивать, поэтому можем их "сжать" по отдельности, и каждое значение будет от 0 до $n-1$. Для партии с номером i, подходящие отрезки по количеству партий и непересекающиеся с предыдущими взятыми, образуют непрерывную последовательность по j. Получается, нужно проверить, существует ли такое j на отрезке, что $prsum_r[i] \geqslant prsum_r[j-1]$ и $prsum_s[i] \geqslant prsum_s[j-1]$. Заметим, что при переходе от i к $i+1$, отрезок подходящих j может очиститься (если i-я партия конец взятого отрезка), и также для ограничений на k_{min} и k_{max}, может в конец быть добавлен элемент (для нового отрезка длины k_{min}) и из начала удален элемент (отрезок длины $k_{max}+1$ быть не может). Это можно реализовать многими способами, например добавляя число (в множество) на отрезке и искать минимум на отрезке с помощью дерева отрезков. Некоторые решения используют дерево Фенвика.

Для решения *подзадачи* 7 не надо уметь удалять отдельное значение, так как при переходе от i к $i+1$ элемента удаления с начала отрезка не будет из-за $k_{max} = n$ (отрезок не может быть слишком большим в данном случае).

Решение будет работать за $O(n \times log(n))$ или $O(n \times log^2(n))$ в зависимости от реализации.

Задача 8. Зеркальный лес

Обозначим начальную точку через S, конечную — через T, а положение i-го камня — через P_i. Точки можно складывать и вычитать по координатам: например, $(x, y) + (u, v) = (x + u, y + v)$.

При отражении относительно камня P_i точка Q переходит в $2P_i - Q$, поскольку P_i — середина между исходной и отражённой точками. Посмотрим, что получится после одного, двух и трёх отражений. Для удобства выбранные камни обозначим через A, B и C:

$$\begin{aligned} Q_1 &= 2A - S, \\ Q_2 &= 2B - Q_1 = S - 2A + 2B, \\ Q_3 &= 2C - Q_2 = -S + 2A - 2B + 2C. \end{aligned}$$

Каждое новое отражение меняет знаки у всех прежних слагаемых и добавляет новый камень с коэффициентом 2. Поэтому знаки при камнях чередуются, а последний знак всегда положительный:

$$\begin{aligned} &\text{чётное число камней:} && -, +, -, +, \ldots, -, +, \\ &\text{нечётное число камней:} && +, -, +, -, \ldots, -, +. \end{aligned}$$

Плюсов либо столько же, сколько минусов, либо на один больше. При чётном числе отражений перед S стоит плюс, при нечётном — минус.

Теперь разобьём использованные камни на две группы: «плюсы» и «минусы». Сложим точки из плюсов и вычтем точки из минусов; результат обозначим через V. Пусть b — число плюсов минус

число минусов. Назовём b *балансом*. Тогда попасть в T можно в одном из двух случаев:

$$b = 0: \quad T = S + 2V, \qquad V = D_0 = \frac{T - S}{2},$$

$$b = 1: \quad T = -S + 2V, \quad V = D_1 = \frac{T + S}{2}.$$

Заметим, что этих условий достаточно. При $b = 0$ начнём с камня из минусов и будем чередовать группы. При $b = 1$ начнём с плюсов и тоже будем чередовать. Получатся именно те знаки, которые нужны в формуле для конечной точки. Внутри группы камни можно брать в любом порядке: их сумма не изменится.

Значит, вместо порядка камней достаточно выбирать для каждого из них одно из трёх состояний: плюс, минус или не использовать. Когда знаки выбраны, порядок восстанавливается чередованием групп.

Сразу обработаем простые случаи. Если $S = T$, ответом будет пустой путь. Также, из формулы D_0 и D_1 видно, что координаты $S + T$ должны делиться на 2, иначе ответ NO.

Решение

Из предыдущих рассуждений становится понятно, что у каждого камня три состояния, поэтому всего распределений 3^n. Для каждого распределения за $O(n)$ посчитаем сумму V и баланс b, затем проверим два условия, описанных выше. Получили решение за $O(n \cdot 3^n)$ времени и $O(n)$ дополнительной памяти.

Если перебирать состояния рекурсивно, сумму и баланс можно не считать заново. Пропуская камень, ничего не меняем; выбирая плюс, прибавляем P_i к сумме и 1 к балансу; выбирая минус, вычитаем их. Это уменьшает время до $O(3^n)$, но для $n = 29$ всё ещё долго.

Разделим номера камней на две почти равные части левую и правую. После этого переберём все варианты расстановки знаков в каждой части отдельно. Для варианта одной половины достаточно сохранить пару (V, b): её сумму и баланс. То есть, применим *meet-in-the-middle*.

Осталось быстро соединить половины. Для правого состояния (V_R, b_R) найдём, какое левое состояние ему нужно. Так как суммы и балансы складываются, подходят

$$(D_0 - V_R,\, -b_R) \qquad \text{или} \qquad (D_1 - V_R,\, 1 - b_R).$$

Первая пара соответствует чётному числу отражений, вторая — нечётному.

Сохраним левые состояния по группам баланса. В каждой группе отсортируем точки: сначала по x, а при равенстве x — по y. Для каждого правого состояния будем искать две нужные пары бинарным поиском.

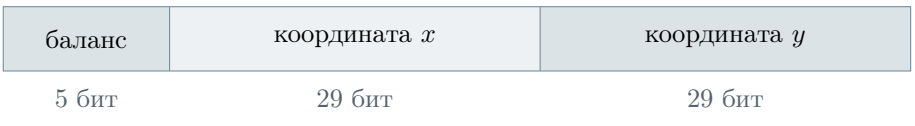
Обозначим $M = 3^{\lceil n/2 \rceil}$ — число вариантов большей половины. С сортировкой и бинарными поисками время составляет $O(M \log M)$, память — $O(M)$.

Бинарный поиск выполняется для каждого правого состояния и для максимальных ограничений является дорогой операцией, для ускорения решения сохраним и отсортируем обе половины: тогда нужные пары можно найти двумя указателями.

Указатели двигаются только вперёд по левой группе и назад по правой, поэтому все проверки вместе занимают $O(M)$ времени. Сортировка, по-прежнему, требует $O(M \log M)$, несмотря на это, такая реализация будет работать уже быстрее, но такое решение в зависимости от реализации может получить TL или ML.

Для упрощения работы научимся упаковывать точки в 64 бита.

В половине не больше 15 камней, поэтому модуль координаты суммы не превосходит $15 \cdot 10^7 < 2^{28}$. После прибавления 2^{28} координата помещается в 29 бит. Баланс лежит от -15 до 15; после прибавления 15 ему хватит 5 бит. Значит, состояние занимает 63 бита и помещается в одно 64-битное число:



Одна запись теперь занимает 8 байт. При $n = 29$ два массива, каждый на 3^{15} записей, занимают около 220 МБ.

После упаковки все состояния можно хранить в одном массиве 64-битных чисел. Однако сортировать весь массив целиком необязательно. После сортировки такой массив будет, в первую очередь, отсортирован по балансу и будет представлять собой блоки из точек с одинаковым балансом. Каждый такой блок будем сортировать отдельно по координатам. Разные балансы между собой сравнивать уже не потребуется, а сами сортировки будут работать с меньшими массивами. Такой подход даст ускорение которого при аккуратной реализации достаточно для получения 100 баллов.

Осталось научиться формировать эти блоки без предварительной сортировки всего массива по балансу. Для этого заранее посчитаем, сколько места требуется каждому блоку, а затем во время перебора будем сразу записывать состояния на нужные места.

Пусть в рассматриваемой половине m камней. Обозначим через $\mathrm{dp}[i][b]$ число способов назначить состояния первым i камням так, чтобы число плюсов минус число минусов равнялось b.

Для нуля камней существует ровно один вариант — ничего не выбирать. Его баланс равен нулю:

$$\mathrm{dp}[0][0] = 1.$$

Все остальные значения $\mathrm{dp}[0][b]$ равны нулю.

Рассмотрим очередной камень. Получить баланс b после его обработки можно тремя способами:

- раньше баланс был b, а камень мы пропустили;
- раньше баланс был $b - 1$, а камень взяли с плюсом;
- раньше баланс был $b + 1$, а камень взяли с минусом.

Эти варианты не пересекаются, поэтому

$$\mathrm{dp}[i + 1][b] = \mathrm{dp}[i][b] + \mathrm{dp}[i][b - 1] + \mathrm{dp}[i][b + 1].$$

После обработки всех m камней значение $\mathrm{dp}[m][b]$ задаёт размер блока баланса b. Здесь учитываются все назначения состояний, даже если некоторые из них дают одинаковую сумму.

Расположим блоки по возрастанию баланса. Первый начинается с позиции 0, а начало каждого следующего получаем прибавлением размера предыдущего. Таким образом, ещё до перебора известны границы всех блоков.

Для каждого блока заведём указатель на следующую свободную позицию. Получив при переборе состояние с балансом b, запишем его по указателю соответствующего блока и увеличим указатель. Все состояния сразу окажутся в нужных участках массива, без дополнительной сортировки по балансу и без увеличения контейнеров во время заполнения.

Теперь каждый блок сортируем отдельно. Поскольку баланс внутри него одинаковый, сравнение упакованных чисел соответствует сравнению сначала по x, затем по y. После сортировки удалим одинаковые записи внутри блока и запомним его новый используемый конец. Сдвигать следующие блоки при этом не обязательно.

Далее применяем два указателя, как и ранее, тут ничего менять не будем.

Осталось восстановить номера использованных камней. После нахождения подходящей пары состояний запомним сумму и баланс каждой половины. Для каждой половины повторно запустим рекурсивный перебор и остановимся на первом подходящем варианте с нужными суммой и балансом. На обратном ходе рекурсии добавим номера выбранных камней в списки плюсов и минусов.

Объединим списки обеих половин и будем чередовать камни: при общем балансе 0 начнём с минуса, при балансе 1 — с плюса. Порядок камней внутри каждого списка может быть любым. Повторный перебор занимает $O(M)$ времени, поэтому общая асимптотика решения не меняется.

Общая оценка решения остаётся $O(M \log M)$ времени и $O(M)$ памяти, но сортировки и последующий поиск выполняют меньше работы и такое решение уже получает 100 баллов.