ЗАДАЧА A АРИФМЕТИЧЕСКИЙ ПОКЕР

Перебор с отсечением и мемоизацией. Перебираем все возможные порядки чисел (в худшем случае 6!=720 вариантов). Между каждой парой чисел – одна из 4 операций (в худшем случае 4^5=1024 вариантов). Генерируем все возможные расстановки скобок (до 88 вариантов для 6 чисел), итого 64880640 вариантов в худшем случае.

Для обработки вариантов с делением нужно хранить числа в виде пары – числитель и знаменатель.

ЗАДАЧА В БАТАРЕЙКИ

Решение для $M \leq 10, N \leq 100$

Для каждого подмножества автобусов мы проверяем, могут ли они завершить весь маршрут. При рассмотрении конкретного подмножества мы отправляем автобусы один за другим. Каждый автобус стремится достичь конца, используя как можно меньше дополнительных единиц. Если это необходимо, он возьмет единицы в последний возможный момент. Если цель достижима, он не лишит другие автобусы ресурсов. Временная сложность составляет $O(2^M \cdot M \cdot N)$.

Решение для $N,M \leq 10^4$

Мы строим на основе предыдущего подхода. Вместо того чтобы рассматривать каждое подмножество, мы замечаем, что автобусы с большим начальным количеством единиц имеют больше шансов завершить маршрут, сохраняя больше единиц. Поэтому мы сортируем их по этому параметру и обрабатываем в порядке убывания. Обратите внимание, что мы не знаем размер группы заранее; он определяется после неудачного маршрута или когда все автобусы завершают поездку. Наибольшая группа будет последней "текущей" группой, найденной этим алгоритмом. Временная сложность составляет $O(M \log M + M \cdot N)$.

Решение для $N,M \leq 10^5$

Оптимальный подход заключается в выборе нескольких автобусов с наибольшим начальным количеством единиц. Мы выполняем бинарный поиск по значению решения. Предположим, мы решили выбрать группу из K автобусов. Мы моделируем их параллельное путешествие через каждую станцию. Когда они прибывают на станцию, они используют все доступные единицы, при этом приоритет отдается "самому пустому" автобусу. Это можно смоделировать с помощью структуры мин-кучи (очередь с приоритетом).

Куча хранит количество единиц и количество автобусов с этим количеством. Если достаточно единиц доступно, мы можем "объединить" два наименьших значения, уменьшая размер кучи на один. Во время такой операции уменьшение может происходить несколько раз, но добавления происходят не более двух раз (при делении недостаточных единиц с остатком). Поскольку мы не можем удалить больше, чем добавляем, сложность каждой проверки составляет $O((N + M)\log M)$.

С бинарным поиском общая временная сложность становится $O((N + M)\log^2 M)$.

Основное решение

Вместо мин-кучи мы используем стек. Это делает каждую проверку линейной, снижая общую сложность до $O((N + M) \log M)$.

ЗАДАЧА С ВОЗМОЖЕН ЛИ РАЗМЕН?

ЗАДАЧА D ГИПЕРПРОСТРАНСТВЕННЫЕ ПЕРЕХОДЫ

Подзадача 1 (8 баллов): n ≤ 16
Общее количество возможных назначений равно 2^N. Таким образом, имея граф, мы можем найти назначение, вмещающее максимальное количество обитаемых систем, за $O(2^N \times N)$ времени. Сначала мы находим оптимальное назначение для исходного графа, а затем проверяем, остается ли размер оптимального назначения неизменным для всех N(N-1)/2 дополнительных рёбер. Временная сложность составляет $O(2^N \times N^3)$.

Подзадача 2 (6 баллов): n ≤ 80
Подзадача 3 (18 баллов): n ≤ 400
Задача нахождения размера оптимального назначения обычно известна как задача о максимальном независимом множестве, которая в общем случае является трудной для произвольных графов. Однако, поскольку галактика представляет собой дерево, мы можем решить её эффективно.

Следующая теорема особенно полезна:

Для любого связного графа G = (V, E) с |V| ≥ 2 и любой вершины v степени 1 существует максимальное независимое множество, которое включает v.

Доказательство:
Предположим обратное. Для любого максимального независимого множества единственной смежной вершиной w к v должно быть включено (в противном случае v можно было бы добавить, что противоречит максимальности). Если мы удалим w и добавим v, то не будет выбрано ни одной пары смежных вершин. Таким образом, мы находим максимальное независимое множество, включающее v, что противоречит предположению.

Используя это, мы можем легко решить задачу для деревьев без дополнительных рёбер. Работает рекурсивный подход на основе DFS: всегда включайте листовую вершину в независимое множество, а для других вершин включайте их только если ни одна из их дочерних вершин не выбрана. Этот алгоритм эквивалентен многократному удалению вершин степени ≤ 1 и их соседей, что обеспечивает корректность.

Когда одно ребро добавляется к дереву, граф становится циклом с деревьями, прикрепленными к его вершинам. Для вершин, не находящихся на цикле, решение остается корректным, так как они рассматриваются как вершины степени 1. Однако, вершины цикла могут нарушить условие независимого множества, поскольку их смежные рёбра принудительно игнорируются.

Если ни одна вершина цикла не выбрана, смежные нециклические вершины включаются, и вершина цикла может рассматриваться как удаленная. Если выбрана вершина цикла, она считается сохраненной. Возникают два случая:

- Если все вершины цикла сохранены, задача сводится к выбору независимого множества на цикле, добавляя ⌊размер цикла / 2⌋ к ответу.
- Если некоторые вершины цикла удалены, оставшиеся вершины образуют пути. Размер независимого множества для каждого пути составляет ⌈размер пути / 2 ⌉, суммируемый к ответу.

Оба случая могут быть решены за O(N) времени, что приводит к общей сложности $O(N^3)$ для $O(N^2)$ итераций.

Подзадача 4 (18 баллов): n ≤ 2000

Решение с временной сложностью O(N^2) достаточно для этой подзадачи. Предполагаемый подход основывается на следующей теореме:

> Пусть T - дерево, а T + (i,j) обозначает T с добавленным ребром (i, j). Определим $S = \{v \mid v \in V(T), v \text{ принадлежит всем максимальным независимым множествам}\}$. Тогда (i, j) является неприемлемой парой тогда и только тогда, когда i ∈ S и j ∈ S.

Доказательство:

(⇒) Если (i, j) неприемлема, все максимальные независимые множества T должны включать как i, так и j; в противном случае такое множество останется допустимым в T + (i, j), что противоречит неприемлемости.

(⇐) По контрапозиции: если (i, j) мирная, размер максимального независимого множества в T + (i, j) равен размеру в T. По крайней мере одна из i или j исключена в некотором максимальном независимом множестве T, так что они не могут обе принадлежать S.

Чтобы вычислить S, для каждой вершины v удалите v и вычислите размер максимального независимого множества в получившихся поддеревьях, используя метод подзадачи 3. Если общий размер уменьшается, v ∈ S. Временная сложность составляет O(N^2).

Подзадача 5 (6 баллов): n ≤ 10000

Подзадача 6 (8 баллов): n ≤ 50000

Подзадача 7 (36 баллов): Без дополнительных ограничений

Из подзадачи 3 алгоритм вычисляет максимальное независимое множество для каждого поддерева, когда оно имеет корень в произвольной вершине. Для подзадачи 4 мы предварительно вычисляем это для всех поддеревьев, не ориентированных на корень. С этой информацией мы можем решить для оставшихся поддеревьев за O(N) времени.

В частности, алгоритм подзадачи 3 вычисляет для каждой вершины v:

- Размер независимого множества в поддереве v.
- Выбрана ли v в этом множестве.

Теперь мы вычисляем для каждой v, когда она рассматривается как корень:

- Размер независимого множества в поддереве par(v), исключая поддерево v.
- Выбрана ли par(v) в этом множестве.

Это делается сверху вниз. Ключевые наблюдения таковы:

- Результат для поддерева par(v) исключает поддерево v.
- Он включает поддерево par(par(v)), которое уже известно.

Пусть onchild(v) будет количеством детей v в независимом множестве. Предварительное вычисление этого позволяет выполнять проверки O(1) для определения, выбрана ли par(v), когда v является корнем. Размер независимого множества выводится из этих значений, что дает временную сложность O(N).

ЗАДАЧА Е АРИФМЕТИЧЕСКАЯ ПРОГРЕССИЯ НАНОСИТ ОТВЕТНЫЙ УДАР

ЗАДАЧА F БЫСТРЫЙ ПОИСК МЕТЕОРИТА

ЗАДАЧА G ВЬЮЩИЕСЯ ПОСЛЕДОВАТЕЛЬНОСТИ

ЗАДАЧА H ГИПЕРПРОСТРАНСТВЕННЫЙ КОНВЕЙЕР

Подзадача 1: M = 0. Нельзя добавлять порталы
Как указано в условии задачи, при заданном массиве векторов игра полностью определена. Для решения этой подзадачи достаточно симулировать процесс и подсчитать количество прыжков.

```
Прочитать N порталов
FOR {i = 1 to N}
   прочитать входные данные l_i, r_i
   pair[l_i] = r_i
   pair[r_i] = l_i
ENDFOR
pos = 1
res = 0
WHILE {pos ≤ 2N}
   pos = pair[pos]
   pos = pos + 1
   res = res + 1
ENDWHILE
Вывести res
```

Сложность решения: O(N).

Подзадача 2: r_i = l_i + 1. Все порталы разделены
Из-за начальной структуры порталов корабль проходит через все из них. Что можно сделать с новым порталом?

- Наблюдение 1: Фишка всегда достигает правого конца.
- Наблюдение 2: Нельзя пройти через портал в одном направлении дважды.

Максимальное увеличение счёта с новым порталом:

- Новый портал можно использовать максимум 2 раза
- Можно заставить портал, используемый 1 раз, использоваться 2 раза
- Можно заставить неиспользуемый портал использоваться до 2 раз

Решение для этой подзадачи:

```
res = N
res = res + 3 * min(N, M)
d = M - N
IF {d > 0}
   res = res + 4 * (d / 2)
   res = res + d % 2
ENDIF
Вывести res
```

Сложность: O(1).

Подзадачи 3-6
Эти подзадачи объединены, так как используют одинаковые наблюдения, различаясь только структурами данных для реализации.

- Наблюдение 3: Неиспользуемые порталы образуют циклы.
- Наблюдение 4: Граф состоит из линии (путь от начала до конца) и набора непересекающихся циклов.

- Наблюдение 5: Новый портал может добавить все узлы цикла к основному пути.

Алгоритм решения:
1. Построить модель (линия + циклы)
2. Отсортировать циклы по размеру
3. Объединять циклы с основным путём от большего к меньшему
4. Если остались новые порталы, использовать алгоритм из подзадачи 2

Инициализировать структуру Union-Find
Обнаружить циклы
Отсортировать циклы по размеру
Вычислить результат
Вывести res

Возможен ли размен?

Тривиально можно считать, что

$$1 \leqslant lq \leqslant rq \leqslant x_{lq} + x_{lq+1} + \ldots + x_{rq}$$

Решение за 5 баллов

Симуляция с `bitset` для каждого запроса.

Сложность: $O\left(\frac{qnS}{32}\right)$

Решение за 10 баллов

С помощью `bitset` можно найти возможные суммы для каждой пары (l, r) и ответить на запросы с одинаковыми (l, r) за один раз.

Сложность: $O\left(\frac{(n^2+q)\cdot S}{32}\right)$, можно модифицировать для получения 25 точек.

Решение за 15 баллов

Поддерживаем r и dp, где $dp[s]$ означает, что в $x_{dp[s]}, \ldots, x_r$ есть подмножество с суммой s, но в $x_{dp[s]+1}, \ldots, x_r$ такого подмножества нет. При переходе $r - 1 \to r$ достаточно обновить:

$$dp\_new[s] = \max(dp\_old[s], dp\_old[s - x_r])$$

и

$$dp\_new[x_r] = r$$

Запрос превращается в "найти количество чисел $lq \leqslant j \leqslant rq$ таких, что $dp[j] \geqslant l$".

Сложность: $O((n + q) \cdot S)$

Можно заметить, что достаточно обновлять только до суммы $x_1 + \ldots + x_r$, причём это можно делать по префиксам или суффиксам (достаточно развернуть массив и найти эквивалентные запросы). Общая сумма обновлений составит $(n + 1) \cdot S$, и можно выбрать способ, при котором она меньше, что даст $\leqslant \frac{(n+1)S}{2}$ обновлений в целом.

Решение за 50 баллов

Если отсортировать $dp[s]$ и запросы по убыванию l, то будет два типа событий: "установить значение в позиции x равным 1"и "найти сумму чисел в позициях от lq до rq". Для этого можно использовать дерево Фенвика.

Сложность: $O\left(\left(\frac{nS}{2} + q\right) \cdot \log_2(S)\right)$

Решение за 80 баллов

Поскольку событий типа `update` гораздо больше, чем `query`, можно поддерживать первые за $O(1)$, а вторые за $O(\sqrt{S})$.

Сложность: $O\left(\frac{nS}{2} + q \cdot \sqrt{S}\right)$

Решение за 100 баллов

Поскольку в позициях поддерживаются только значения 0 и 1, можно снова использовать `bitset` и уменьшить время выполнения запроса до $\sqrt{\frac{S}{32}}$.

Сложность: $O\left(\frac{nS}{2} + q \cdot \sqrt{\frac{S}{32}}\right)$

Арифметическая прогрессия наносит ответный удар

Подзадача 1

$N \leqslant 3$. Поскольку $N \geqslant 2$, рассмотрим два случая: $N = 2$ и $N = 3$.

- Для $N = 2$ изменения не требуются. Ответ: 0.
- Для $N = 3$ пусть числа на картах слева направо будут a, b и c. Если $c - b = b - a$, изменения не нужны, и ответ: 0. Иначе, пусть $d = b - a$. Изменение последнего числа на $b + d$ делает последовательность корректной, поэтому ответ: 1.

Сложность по времени: $O(1)$, что достаточно для заданных ограничений.

Подзадача 2

После изменения чисел последовательность на картах должна иметь вид $a, a + d, \ldots, a + (N - 1)d$ для некоторых целых a и d.

Для $N \leqslant 5$ хотя бы два исходных числа остаются неизменными. Рассматриваем все возможные пары карт и проверяем, могут ли их числа остаться без изменений. Если да, определяем a и d и подсчитываем количество карт, которые нужно изменить. Минимальное количество среди всех случаев будет ответом.

Для $N > 5$ существует хотя бы одна пара соседних карт, числа которых остаются неизменными. Применяем аналогичный подход, как для $N \leqslant 5$, но фокусируемся на соседних парах.

Сложность по времени: $O(N^2)$, что эффективно для заданных ограничений.

Подзадача 3

Как и в Подзадаче 2, итоговая последовательность должна иметь вид $a, a+d, \ldots, a+(N-1)d$. Хотя бы одно исходное число остаётся неизменным.

Для каждой карты i ($1 \leqslant i \leqslant N$), если её число остаётся неизменным и d определено, то $a = x_i - (i - 1)d$. Перебираем все возможные значения d и подсчитываем количество карт, которые нужно изменить. Минимальное количество будет ответом.

Количество возможных значений d: $O(d)$, подсчёт изменений для каждого случая занимает $O(N)$ времени. Общая сложность: $O(N^2 d)$. Учитывая небольшой диапазон d, это выполнимо в рамках ограничений.

Подзадача 4

Ответ не превышает $N - 2$. Это объясняется тем, что если оставить первые два числа a и b неизменными и задать $d = b - a$, первые две карты не требуют изменений. Остальные $N - 2$ карт могут потребовать изменений для корректности последовательности.

Ключевое наблюдение: хотя бы два числа остаются неизменными. Для каждой пары карт (i, j), где $i < j$, если их числа x_i и x_j остаются неизменными, то $d = \frac{x_j - x_i}{j - i}$ должно быть целым числом. Если нет, пара пропускается. Иначе, определяется $a = x_i - (i - 1)d$, и подсчитывается количество карт, требующих изменений.

Количество возможных пар: $\frac{N(N-1)}{2}$, подсчёт изменений для каждого случая занимает $O(N)$ времени. Общая сложность: $O(N^3)$, что выполнимо для заданных ограничений.

Быстрый поиск метеорита

1 Решение 1 - 20 баллов

Пусть r — точка на плоскости, где находится R2D2, m — точка, где упал метеорит, и d — расстояние между роботом и точкой падения в начальный момент.

Выберем случайное направление и сделаем шаг в этом направлении. Если мы не приблизились, выберем противоположное направление. Затем будем делать шаги в выбранном направлении, пока приближаемся. В момент, когда снова получаем ответ 0, делаем шаг назад и замечаем, что мы только что идентифицировали полосу длиной в один метр, в которой может находиться m. Эта полоса перпендикулярна выбранному направлению, а точка, в которой мы находимся, расположена в середине отрезка, определяемого пересечением направления с полосой.

Повторим процедуру для направления, перпендикулярного предыдущему, и идентифицируем вторую полосу, перпендикулярную первой. Таким образом, мы сократили пространство, в котором может находиться m, до квадрата со стороной в один метр. Чтобы убедиться, что покрываем весь квадрат, после размещения в центре квадрата делаем по шагу к каждому углу и затем обратно к центру.

В худшем случае общее количество шагов составит $d \times 2$, плюс небольшое количество шагов для возврата, покрытия углов квадрата и т.д.

2 Решение 2 — 60 баллов

Ключевое наблюдение: любой шаг, который ведёт внутрь окружности радиуса d с центром в m, получит ответ «приблизился», а любой шаг, ведущий наружу, получит ответ «не приблизился». Таким образом, мы идентифицируем непрерывный интервал углов на плоскости, которые дадут ответ «приблизился». Эти углы ограничены двумя полупрямыми, представляющими направления шагов. Направления этих полупрямых можно найти с помощью двух бинарных поисков.

Если у нас есть две полупрямые, то направление движения задаётся биссектрисой угла между ними. Количество шагов в этом случае равно $\lceil d \rceil$. Ошибку, с которой нужно определить углы полупрямых, можно вычислить, зная, что нужно попасть в окружность радиуса 1 с центром в m.

После простого геометрического расчёта получаем, что максимально допустимая ошибка равна $\operatorname{arctg}(1/d)$, которая для больших d приблизительно равна $1/d$. Таким образом, общее количество шагов составит $2\log_2(8d) + \lceil d \rceil$ (число внутри логарифма связано с интервалом углов $[0, 2\pi]$ и расстоянием d).

3 Решение 3 - 100 баллов

При более детальном анализе оказывается, что не обязательно двигаться по биссектрисе. Если определить только одну полупрямую и двигаться перпендикулярно ей, то направление движения пересечёт окружность радиуса 1 с центром в m в двух точках. Эти точки определяют отрезок длиной $\sqrt{3} > 1$, поэтому, двигаясь в этом направлении, мы гарантированно попадём внутрь целевой окружности.

Отрезок, по которому мы движемся до попадания в окружность, является катетом в прямоугольном треугольнике с гипотенузой длиной d и вторым катетом длиной 1. Таким образом, расстояние, которое мы проходим в этом направлении, меньше расстояния, пройденного по биссектрисе.

Итоговое количество шагов составляет $\lceil d \rceil + \log_2(16d)$ (разрешение остаётся тем же, с добавлением множителя 2 для гарантии достаточной длины отрезка внутри окружности).

Вьющиеся последовательности

1 Подзадача 1

Самый простой способ решения задачи. Для вычисления $f(x, y, z)$ рассматриваются все возможные подпоследовательности последовательности A и проверяется выполнение условий 1, 2 и 3. Для последовательности A длины N существует 2^N подпоследовательностей (каждый элемент можно либо выбрать, либо не выбрать). Для каждой подпоследовательности условия проверяются за $O(N)$ с помощью цикла. Среди подпоследовательностей, удовлетворяющих всем трём условиям, выбирается самая длинная, её длина и будет $f(x, y, z)$. Поскольку нужно вычислить $O(N^3)$ значений функции f, общая временная сложность составляет $O(N^4 \cdot 2^N)$.

Условия 1 и 2 проверяются легко, поэтому можно сначала отфильтровать подпоследовательности, удовлетворяющие этим условиям, а затем проверить условие 3.

2 Подзадача 2

В подзадаче 1 перебирались все 2^N подпоследовательностей. Основное узкое место — условие 3. Если существует более быстрый способ нахождения самой длинной зигзагообразной подпоследовательности, то временную сложность можно значительно улучшить.

Для последовательности $S_1, \ldots, S_K$ длины K самую длинную зигзагообразную подпоследовательность можно найти с помощью простого жадного алгоритма:

1. Подготовить массив arr для хранения самой длинной зигзагообразной подпоследовательности. Изначально arr пуст.
2. Для каждого $i = 1, 2, \ldots, K$:
   (a) Если добавление S_i в конец arr сохраняет зигзагообразность, добавить S_i в конец arr.
   (b) Иначе заменить последний элемент arr на S_i.

Этот алгоритм работает за $O(N)$. Сначала фильтруются элементы по условиям 1 и 2, затем применяется алгоритм для проверки условия 3. Поскольку нужно вычислить $O(N^3)$ значений f, общая временная сложность составляет $O(N^4)$.

3 Подзадача 3

Решение подзадачи 2 можно улучшить для подзадачи 3. Заметим, что в подзадаче 2 вычисляются $O(N^3)$ значений, многие из которых повторяются. Используя метод скользящего окна, можно оптимизировать вычисления и найти значения $f(x, y, y+1), f(x, y, y+2), \ldots, f(x, y, N)$ за одну итерацию. Временная сложность сокращается до $O(N^3)$, что позволяет решить подзадачу 3.

4 Подзадача 4

Для решения подзадач, начиная с подзадачи 4, требуется другой подход. Вместо вычисления каждого f и их суммирования для получения g, можно вычислить g напрямую.

Из алгоритма подзадачи 2 следует, что длина самой длинной зигзагообразной подпоследовательности для $S_1, \ldots, S_K$ может быть вычислена по формуле:

$$f(x, y, z) = \text{длина } C - (\text{количество } i : C_i < C_{i+1} < C_{i+2}) - (\text{количество } i : C_i > C_{i+1} > C_{i+2}),$$

где C — подпоследовательность $A_y, \ldots, A_z$, содержащая элементы $\leqslant x$.

Для вычисления $g(x)$ можно использовать следующий метод:

1. Для каждого элемента B_j в подпоследовательности B (элементы $\leqslant x$) вычислить его вклад:
   - Вклад в длину: $P_j \times (N - P_j + 1)$.
   - Вклад в условия 2 и 3: если $B_j < B_{j+1} < B_{j+2}$ или $B_j > B_{j+1} > B_{j+2}$, то $P_j \times (N - P_{j+2} + 1)$.
2. Суммировать вклады всех элементов B_j и вычесть вклады условий 2 и 3.

Этот метод позволяет решить подзадачу 3 за $O(N^3)$, а с использованием структур данных (например, `std::set` в C++) — подзадачу 4 за $O(N^2 \log N)$.

5 Подзадача 5

Метод вычисления вкладов можно улучшить до линейной сложности. При увеличении x до $x + 1$ в подпоследовательность B добавляется элемент $x + 1$. Вклад этого элемента и изменение вкладов соседних элементов вычисляются аналогично подзадаче 4. Это позволяет сократить временную сложность до $O(N \log N)$.