Задача A. Устойчивый луч

Идея решения

Требуется найти непрерывный неубывающий отрезок массива. Будем просматривать массив слева направо и поддерживать самый длинный подходящий отрезок, заканчивающийся в текущей позиции.

Алгоритм

Будем использовать нумерацию элементов с единицы. Обозначим через l начало текущего отрезка, через b начало лучшего найденного отрезка, а через L его длину. Изначально $l = b = 1$ и $L = 1$.

Для каждого i от 2 до N выполним следующие действия:

1. Если $a_i < a_{i-1}$, положим $l = i$: продолжить предыдущий отрезок нельзя.
2. Иначе сохраним прежнее значение l: текущий отрезок продолжается.
3. Если $i - l + 1 > L$, запишем $L = i - l + 1$ и $b = l$.

Выведем L, затем b, затем элементы $a_b, \ldots, a_{b+L-1}$. При равенстве длин ответ не обновляем: ранее найденный отрезок начинается раньше.

Доказательство корректности

Докажем, что после обработки позиции i отрезок $[l, i]$ является самым длинным неубывающим отрезком с правым концом i. Для $i = 1$ утверждение очевидно. Если $a_i < a_{i-1}$, никакой подходящий отрезок, заканчивающийся в i, не может включать позицию $i-1$, поэтому его длина равна единице. Если $a_i \geqslant a_{i-1}$, самый длинный подходящий отрезок с правым концом $i - 1$ можно продлить на один элемент. Более длинного отрезка быть не может: после удаления последнего элемента он дал бы более длинный отрезок с правым концом $i - 1$. Инвариант доказан.

Каждый возможный правый конец рассматривается, поэтому алгоритм находит максимальную длину. Для одинаковой длины более ранний правый конец означает более раннее начало. Обновление только при строгом увеличении длины сохраняет требуемый ответ. При $N = 1$ начальные значения уже задают правильный ответ.

Сложность

Время работы составляет $O(N)$, включая вывод. Для хранения массива требуется $O(N)$ памяти; помимо массива алгоритм использует $O(1)$ памяти.

Задача B. Красивый маршрут

Идея решения

Нужно найти строго возрастающую подпоследовательность максимальной длины, а среди всех таких подпоследовательностей выбрать лексикографически минимальную по значениям. При $N \leqslant 2000$ достаточно динамического программирования за $O(N^2)$ и жадного восстановления.

Динамическое программирование

Пусть dp_i обозначает максимальную длину строго возрастающей подпоследовательности, первый элемент которой находится в позиции i. Тогда

$$dp_i = \max\bigl(\{1\} \cup \{1 + dp_j \mid i < j \leqslant N,\ a_j > a_i\}\bigr).$$

Вычислим значения в порядке убывания i, чтобы все нужные dp_j уже были известны. Длина ответа равна $L = \max_{1 \leqslant i \leqslant N} dp_i$.

Восстановление ответа

Пусть p обозначает позицию последнего выбранного элемента, v его значение, а r количество элементов, которые ещё требуется выбрать. Изначально $p = 0$, $v = -1$ и $r = L$. Значение -1 допустимо, поскольку все $a_i \geqslant 0$.

Пока $r > 0$, рассмотрим все позиции i, удовлетворяющие условиям

$$i > p, \qquad a_i > v, \qquad dp_i \geqslant r.$$

Выберем среди них позицию с минимальным a_i, а при равных значениях с минимальным индексом i. Добавим a_i в ответ и положим $p = i$, $v = a_i$, $r = r - 1$.

Условие $dp_i \geqslant r$ означает, что начиная с выбранной позиции можно набрать нужное количество элементов. Если подпоследовательность длиннее, достаточно взять её первые r элементов.

Доказательство корректности

Любая возрастающая подпоследовательность с первым элементом в позиции i либо состоит только из него, либо следующим содержит некоторый элемент $a_j > a_i$ с $j > i$. Поэтому переход динамики перебирает все варианты и верно вычисляет dp_i. Следовательно, L является максимальной длиной.

При восстановлении множество рассматриваемых позиций в точности описывает возможные следующие элементы, после которых ответ ещё можно достроить до длины L. Оно непусто: до каждого шага существует подходящее продолжение. Выбор наименьшего значения даёт минимально возможный очередной элемент при уже зафиксированном префиксе ответа.

Если одинаковое значение можно выбрать в нескольких позициях, самая ранняя из них оставляет все продолжения, доступные после более поздних позиций. Поэтому выбор ранней позиции не ухудшает ответ. Повторяя рассуждение для каждой позиции ответа, получаем лексикографически минимальную подпоследовательность максимальной длины.

Сложность

Вычисление динамики занимает $O(N^2)$ времени. Восстановление делает не более N проходов по массиву, то есть также требует $O(N^2)$ времени. Общая сложность составляет $O(N^2)$ по времени и $O(N)$ по памяти.

Задача C. Правильная цепочка времени

Идея решения

Переведём каждый момент времени в число секунд от начала суток:

$$t = 3600h + 60m + s.$$

Отсортируем полученные числа: $t_1 < t_2 < \cdots < t_N$. Исходный порядок записей не ограничивает выбор цепочки, поэтому сортировка допустима. Теперь требуется найти самую длинную арифметическую прогрессию с положительной разностью среди имеющихся значений.

Динамическое программирование

При $N = 1$ выведем единственный момент и завершим работу. Для $i < j$ обозначим через $dp_{i,j}$ максимальную длину цепочки, первые два элемента которой равны t_i и t_j.

Следующий элемент, если он существует, определяется однозначно:

$$x = t_j + (t_j - t_i) = 2t_j - t_i.$$

Если $x = t_k$ для некоторого $k > j$, то

$$dp_{i,j} = 1 + dp_{j,k}.$$

Если такого значения нет, положим $dp_{i,j} = 2$. Вычислим строки таблицы в порядке убывания i. Тогда при обработке пары (i, j) строка с первым индексом j уже вычислена.

Для быстрого поиска следующего значения можно использовать массив pos длины 86400: $pos[x]$ хранит индекс момента x либо специальную отметку, если такого момента нет. Значения $x \geqslant 86400$ сразу считаем отсутствующими. Так поиск выполняется за $O(1)$ без хеширования.

Выбор и восстановление ответа

Среди всех пар (i, j) выберем пару с максимальным $dp_{i,j}$. При равной длине сравним пары (t_i, t_j) лексикографически: сначала минимизируем первый момент, затем второй. Этого достаточно, поскольку первые два элемента однозначно задают разность и всю дальнейшую прогрессию.

Для выбранной пары положим $L = dp_{i,j}$ и $d = t_j - t_i$. Ответ состоит из значений

$$t_i,\ t_i + d,\ \ldots,\ t_i + (L-1)d.$$

Каждое значение x переведём обратно в часы, минуты и секунды:

$$h = \left\lfloor \frac{x}{3600} \right\rfloor, \qquad m = \left\lfloor \frac{x \bmod 3600}{60} \right\rfloor, \qquad s = x \bmod 60.$$

Каждую компоненту выводим двумя цифрами, при необходимости с ведущим нулём.

Доказательство корректности

Для фиксированных первых двух элементов возможен только один следующий момент $2t_j - t_i$. Если его нет в журнале, цепочка не продолжается и имеет длину два. Если он есть, задача о продолжении совпадает с задачей для пары (j, k), а перед ней добавляется один элемент t_i. Это доказывает переход динамики. Порядок вычисления обеспечивает готовность всех зависимостей, следовательно, вся таблица вычисляется правильно.

При $N \geqslant 2$ любая пара различных моментов уже образует допустимую цепочку, поэтому оптимальная цепочка содержит хотя бы два элемента. Все возможные первые пары перебираются, а значит, максимальное значение таблицы равно длине оптимального ответа. Правило сравнения первых двух элементов точно совпадает с правилом выбора из условия. Восстановление выводит именно элементы найденной прогрессии.

Сложность

Сортировка занимает $O(N \log N)$ времени, динамика — $O(N^2)$. С учётом инициализации массива поиска общая оценка составляет $O(N^2 + 86400)$ по времени и памяти. Размер суток является константой, поэтому обычно эту оценку записывают как $O(N^2)$. Длина цепочки не превышает 2000, поэтому для элемента таблицы достаточно 16-битного беззнакового целого типа.

Задача D. Особый квадрат

Идея решения

При $N, M \leqslant 200$ можно перебрать все квадратные фрагменты. Чтобы проверять каждый квадрат за $O(1)$, заранее вычислим префиксные суммы отдельно для каждой строки и каждого столбца.

Префиксные суммы

Используем нумерацию строк и столбцов с единицы. Определим

$$R_{i,0} = 0, \qquad R_{i,j} = \sum_{q=1}^{j} a_{i,q},$$

$$C_{0,j} = 0, \qquad C_{i,j} = \sum_{p=1}^{i} a_{p,j}.$$

Тогда сумма участка строки i от столбца l до r равна $R_{i,r} - R_{i,l-1}$, а сумма участка столбца j от строки u до v равна $C_{v,j} - C_{u-1,j}$.

Для квадрата размера k с левым верхним углом (i, j) положим $b = i + k - 1$ и $r = j + k - 1$. Суммы его сторон вычисляются так:

$$T = R_{i,r} - R_{i,j-1}, \quad B = R_{b,r} - R_{b,j-1},$$
$$L = C_{b,j} - C_{i-1,j}, \quad Q = C_{b,r} - C_{i-1,r}.$$

Квадрат подходит тогда и только тогда, когда $T = B = L = Q$. Каждая сторона включает обе угловые клетки. Вычитать углы не нужно: мы сравниваем четыре отдельные суммы, а не сумму всей границы.

Алгоритм

1. Построим обе таблицы префиксных сумм.
2. Переберём k от $\min(N, M)$ до 1 в порядке убывания.
3. Для каждого k переберём $i = 1, \ldots, N - k + 1$, а внутри этого цикла $j = 1, \ldots, M - k + 1$, оба индекса в порядке возрастания.
4. Как только найдём подходящий квадрат, выведем k и (i, j) и завершим алгоритм.

Доказательство корректности

Разность префиксных сумм в точности равна сумме нужного участка строки или столбца. Поэтому проверка четырёх равенств верно определяет, является ли квадрат особым.

Размеры рассматриваются от большего к меньшему, следовательно, первый найденный квадрат имеет максимальный возможный размер. Для этого размера сначала рассматриваются меньшие номера строк, а при одинаковой строке меньшие номера столбцов. Значит, первый подходящий квадрат удовлетворяет обоим правилам выбора ответа.

Алгоритм обязательно найдёт ответ: квадрат размера один имеет четыре одинаковые суммы сторон, каждая из которых равна его единственному элементу.

Сложность

Построение префиксных сумм требует $O(NM)$ времени и памяти. Число проверяемых квадратов не превышает

$$\sum_{k=1}^{\min(N,M)} (N-k+1)(M-k+1) \leqslant NM\min(N,M).$$

Итоговая сложность составляет $O(NM\min(N,M))$ по времени и $O(NM)$ по памяти. При $N = M = 200$ существует 2 686 700 квадратов.

Бинарный поиск по размеру здесь неприменим: существование подходящего квадрата не является монотонным свойством размера.

Задача E. Фестиваль огней

Идея решения

Каждому фонарю сопоставим целочисленный вектор (x, y) из начала координат. Два фонаря лежат на одном луче тогда и только тогда, когда их векторы отличаются положительным множителем.

Найдём каноническое представление направления, сократив обе координаты на их наибольший общий делитель:

$$g = \gcd(|x|, |y|), \qquad (u, v) = \left(\frac{x}{g}, \frac{y}{g}\right).$$

По условию $(x, y) \neq (0, 0)$, поэтому $g > 0$ и деление всегда определено.

Алгоритм

Создадим пустое множество пар. Для каждого фонаря вычислим нормализованную пару (u, v) и добавим её в множество. Ответ равен размеру множества.

Знаки координат обязательно сохраняем. Например, $(1, 2)$ и $(-1, -2)$ задают противоположные лучи и должны учитываться отдельно. Для точек на осях получаются направления $(1, 0)$, $(-1, 0)$, $(0, 1)$ и $(0, -1)$. Вычислять углы или отношения координат с плавающей точкой не требуется.

Доказательство корректности

Нормализованный вектор имеет взаимно простые абсолютные значения координат и направлен в ту же сторону, что исходный, поскольку делитель g положителен.

Покажем единственность такого представления на луче. Пусть два нормализованных целочисленных вектора лежат на одном луче. Множитель между ними можно записать как положительную несократимую дробь p/q. Чтобы после умножения первого вектора обе координаты остались целыми, q должен делить обе его координаты. Их НОД равен единице, значит, $q = 1$. Тогда второй вектор получается умножением на целое p, и взаимная простота его координат требует $p = 1$. Следовательно, векторы совпадают.

Обратно, если нормализованные пары совпадают, исходные векторы являются положительными целыми кратными одной пары и лежат на одном луче. Значит, совпадение пар в точности соответствует совпадению направлений, а размер множества равен искомому количеству лучей.

Сложность

Пусть $A = \max(1, \max_i |x_i|, \max_i |y_i|)$. Алгоритм Евклида вычисляет один НОД за $O(\log A)$ времени. С хеш-множеством ожидаемое общее время составляет $O(N \log A)$, а память — $O(N)$. Альтернативно можно отсортировать все нормализованные пары и посчитать различные: это даёт гарантированную оценку $O(N \log A + N \log N)$ по времени и $O(N)$ по памяти.

Задача F. Подземные туннели

Идея решения

Обычного кратчайшего пути по сумме времён туннелей недостаточно: число пройденных туннелей определяет количество остановок. Кроме текущей башни, будем хранить число туннелей, пройденных после последней остановки, и применим алгоритм Дейкстры к расширенному графу.

Когда следует останавливаться

Для маршрута из $q \geqslant 1$ туннелей минимальное число остановок равно

$$\left\lceil \frac{q}{K} \right\rceil - 1 = \left\lfloor \frac{q-1}{K} \right\rfloor.$$

Действительно, r остановок делят маршрут на $r+1$ участков, каждый не длиннее K туннелей. Поэтому $q \leqslant (r+1)K$, что даёт указанную нижнюю границу. Она достигается, если останавливаться после каждых K туннелей, когда ещё нужно продолжать движение.

Таким образом, даже если разрешать досрочные остановки, для фиксированного маршрута они не уменьшают время: стоимость каждой остановки одинакова и $S \geqslant 0$. Достаточно рассматривать остановки после ровно K туннелей. После прибытия в конечную башню остановка не требуется.

Расширенный граф

Состояние (v, c) означает, что курьер находится в башне v и после последней остановки прошёл c туннелей, где $0 \leqslant c \leqslant K$. Начальное состояние — $(1, 0)$.

Переходы задаются следующим образом:

- Для каждого туннеля между v и u с временем w и каждого $c < K$ есть переход $(v, c) \longrightarrow (u, c+1)$ стоимости w. Поскольку туннели двусторонние, учитываем оба направления.
- Из состояния (v, K) есть переход $(v, K) \longrightarrow (v, 0)$ стоимости S: обязательная остановка перед продолжением движения.

Все стоимости неотрицательны, поэтому применим алгоритм Дейкстры. Пусть $dist_{v,c}$ обозначает найденное расстояние от $(1, 0)$. Искомое время равно

$$\min_{0 \leqslant c \leqslant K} dist_{N,c}.$$

В частности, состояние (N, K) допустимо: после последнего туннеля не нужно дополнительно платить за остановку.

Реализация

Храним исходный граф списками смежности и генерируем переходы при извлечении состояния из очереди с приоритетом. Явно строить все рёбра расширенного графа не нужно.

Изначально $dist_{1,0} = 0$, остальные расстояния бесконечны. При извлечении записи с расстоянием, не равным текущему значению $dist_{v,c}$, пропускаем её как устаревшую. Если $c = K$, релаксируем только остановку. Иначе релаксируем переходы по всем соседним туннелям. Можно завершить работу при первом извлечении неустаревшего состояния с башней N: его расстояние уже минимально среди всех конечных состояний.

Доказательство корректности

Любому пути в расширенном графе соответствует допустимое движение курьера: переходы по туннелям увеличивают счётчик, а перед его переполнением необходимо выполнить остановку и сбросить счётчик. Стоимость пути равна общему времени движения и остановок.

Для любого маршрута в исходном графе можно расставить остановки после каждых K туннелей, кроме конца маршрута. Как доказано выше, это даёт минимальное возможное время прохождения данного маршрута. Такое движение полностью представлено в расширенном графе. Следовательно, минимум по его конечным состояниям равен оптимальному времени в исходной задаче. Алгоритм Дейкстры верно находит этот минимум, поскольку веса всех переходов неотрицательны.

Сложность и типы данных

В расширенном графе $V' = N(K+1)$ состояний и $E' = 2MK + N$ переходов. С двоичной кучей и удалением устаревших записей время работы составляет $O((V'+E')\log(V'+E'))$, что при данных ограничениях можно записать как $O((N+M)K\log(NK))$.

Таблица расстояний и исходный граф занимают $O(NK + N + M)$ памяти. При реализации с добавлением новой записи в кучу на каждую успешную релаксацию сама куча в худшем случае занимает $O(E')$ памяти; общая оценка памяти тогда равна $O((N+M)K)$. Индексированная куча с операцией уменьшения ключа позволяет хранить в очереди не более $O(NK)$ записей.

Для расстояний необходим 64-битный целый тип. Например, время простого пути вместе с остановками не превосходит $2(N-1)\cdot 10^9$, что помещается в знаковый 64-битный тип. Бесконечность следует выбирать больше этой границы; при релаксации нельзя складывать вес с самой бесконечностью.