Задача F. Безопасная дорога

В первой подзадаче значение n не превышает 20, поэтому мы можем применить полный перебор вариантов установки факелов за $O(2^n)$. Перебираем все возможные строки длины n, состоящие из 0 и 1. Пусть на блоке с индексом i установлен факел, если в i-й позиции строки стоит 1. Затем проверим, освещают ли установленные факелы всю дорогу. Если да, то это одно из допустимых решений. Из всех допустимых вариантов выбираем тот, при котором Панкрат тратит меньше всего изумрудов.

Для решения второй подзадачи применим метод динамического программирования. Пусть $dp[i]$ — это минимальное количество изумрудов, которое нужно потратить, чтобы осветить дорогу от первого до i-о блока, поставив факел на блоке i.

Базой динамического программирования являются случаи, когда нужно осветить блоки от первого до $(2k)$-о — для этого достаточно одного факела, так что базу легко определить.

Для перехода найдём минимум среди значений $dp[j]$ на отрезке от $i-2k-1$ до $i-1$. Это можно сделать линейным проходом по отрезку, что даёт сложность перехода $O(k)$. В итоге получаем общее время работы $O(nk)$. Ответом будет минимальное значение $dp[i]$ на отрезке $[\max(0, n-k-1), n-1]$.

Для решения третьей подзадачи улучшим предыдущее решение. Минимум на отрезке можно искать с помощью мультимножеств. Пройдём по массиву dp, поддерживая нужный отрезок $[i-2k-1, i-1]$ для текущего i. Операции удаления и добавления происходят за $O(\log k)$, так что общее время работы составит $O(n \log k)$.

Полное решение получаем, заметив, что отрезок, на котором нужно искать минимум, всегда имеет фиксированную длину. Тогда задача перехода в динамике сводится к задаче нахождения минимума на скользящем окне фиксированной длины. Эта задача известна и решается с помощью двухсторонней очереди, что позволяет получать минимум за константное время.

Таким образом, итоговая сложность решения $O(n)$.

Также можно написать дерево отрезков снизу. Это решение хоть и работает за $O(n \log k)$, но при аккуратной реализации проходит все тесты.

Задача G. Заполнение хордами

Все свободные точки окружности разбиваются уже нарисованными хордами на группы — в каждой группе любые две точки могут быть соединены хордой так, чтобы она не пересекала нарисованные хорды.

Очевидно, что в группе, состоящей из m вершин, можно провести $\lfloor \frac{m}{2} \rfloor$ хорд — например, разбить все точки на пары соседних точек и соединить.

Далее остается аккуратно реализовать это.

Первая подзадача ($k = 0$) имеет очевидное решение.

Во второй подзадаче ($k = 1$) все свободные точки разбиваются на две группы и их можно легко соединить максимальным количеством хорд.

В третьей подзадаче ($k = 2$) точки разбиваются на не более, чем четыре группы, поэтому каждую группу всё еще можно обработать вручную.

По мнению автора, самым простым способом решить полную задачу является рекурсивное разделение задачи на две задачи меньшей размерности по принципу «разделяй и властвуй»: каждая нарисованная хорда разбивает рассматриваемые точки на две группы, каждую из которых рекурсивно рассматриваем дальше. Рекурсия заканчивается, когда в рассматриваемой группе точек больше нет хорд — тогда эти точки просто соединяем максимальным количеством хорд.

Задача H. Погоня

Для решения первой подзадачи достаточно заметить, что если $t_1 < a$, то либо Тимур сам прилетит к Алёше, либо его невозможно поймать. Если $t_1 > a$, то Тимура невозможно поймать, так как если он находится в городе v ($v < n$), то расстояние между двумя игроками не может уменьшиться, а если он находится в городе n, то на следующий день игра заканчивается.

Для решения второй подзадачи нужно заметить, что нам дан граф, похожий на дерево, и воспользоваться тем фактом, что самый дешёвый путь до какого-либо города также является самым

коротким. Это значит, что мы можем пройтись по всем вершинам графа с помощью поиска в глубину и сохранить в каждом из них стоимость и расстояние от начальной вершины Алёши.

Для решения третьей подзадачи нужно перебрать все возможные пути Алёши. Для этого можно воспользоваться поиском с возвратом.

Для решения четвёртой подзадачи требуется уже оптимально хранить стоимости путей от начальной вершины до всех других для всех возможных длин путей. Для начала стоит заметить, что длина пути не может превышать $n-1$, так как в противном случае это будет означать, что мы посетили какой-то город дважды. Соответственно, заведём массив `d[cnt][v]`, где `cnt` — количество дней, а `v` — город. Сначала `d[0][a] = 0`, а все остальные значения равны бесконечности. Далее требуется оптимальным образом заполнить все значения. Поскольку ограничения малы, можно воспользоваться либо алгоритмом Дейкстры для разреженных графов, либо алгоритмом Беллмана-Форда.

Задача I. Заработок

Заметим, что можно сделать бинарный поиск по значению

$$P_{i_1 i_K} = \frac{A_{i_1} + A_{i_2} + \cdots + A_{i_K}}{B_{i_1} + B_{i_2} + \cdots + B_{i_K}}.$$

Для этого будем проверять, существует ли путь в дереве, удовлетворяющий неравенству $P_{i_1 i_K} \geqslant X$, где X — значение, по которому проводится бинарный поиск. Преобразуем это неравенство:

$$A_{i_1} + A_{i_2} + \cdots + A_{i_K} \geqslant X(B_{i_1} + B_{i_2} + \cdots + B_{i_K}) \Rightarrow$$
$$\Rightarrow (A_{i_1} - XB_{i_1}) + (A_{i_2} - XB_{i_2}) + \cdots + (A_{i_K} - XB_{i_K}) \geqslant 0.$$

Таким образом, задача сводится к поиску простого пути в дереве, сумма весов рёбер которого по новой метрике $(A_i - XB_i)$ неотрицательна. Проверку можно эффективно реализовать, например, с помощью динамического программирования по поддеревьям или центроидной декомпозиции.

Поскольку бинарный поиск проводится по вещественным числам с плавающей точкой, его сложность составляет $O(60)$ шагов. Проверка на каждом шаге работает за $O(n)$, что даёт итоговую асимптотику $O(60 \cdot n)$.

Задача J. Побег из инопланетной тюрьмы

Сначала немного переформулируем условие задачи. Для каждого запроса надо найти кратчайший *безопасный* маршрут из точки A в точку B. Маршрут считается *безопасным*, если он не проходит строго через отрезок, образованный 2 вышками, а также не заходит строго внутрь треугольника, образованный 3 вышками. По сути это означает, что *безопасный* маршрут не должен заходить внутрь ***выпуклой оболочки*** всех вышек в запросе.

Подзадача 1

В этой подзадаче гарантировалось, что все вышки находятся в третей четверти плоскости, а точки A и B — в первой четверти. Это означает, что ответом будет просто длина отрезка AB.

Подзадача 2

Так как в этой подзадаче $L = R$, то это значит, что на плоскости находятся всего 2 вышки в каждом запросе. Обозначим их как точки C и D. Тогда есть два случая — отрезок AB строго пересекается с CD, и не пересекается. В первом случае ответом будет минимум из $|AC| + |CB|$ и $|AD| + |DB|$. Во втором случае ответ — $|AB|$. Также нужно было учесть частный случай, когда AB лежит строго внутри CD.

Подзадача 3

В этой подзадаче $R - L \leqslant 1$. То есть количество вышек на плоскости не превосходит 4. Здесь предполагалось аккуратно перебрать все возможные случаи.

Подзадача 4

Во всех запросах использовались все векторы для клонирования, и точка A совпадала с точкой B. То есть нам надо было как-то построить выпуклую оболочку всех 2^n вышек заранее, и в каждом запросе проверять, лежит ли A строго внутри этой выпуклой оболочки. Так как же её построить?

Заметим, что когда мы клонируем вышки, то у нас происходит частный случай суммы Минковского с 2-сторонним многоугольником (это наш вектор клонирования). То есть, если мы продублируем все векторы в противоположном направлении, отсортируем их по полярному углу, и найдём самую нижнюю левую вышку, то мы построим искомую выпуклую оболочку. Однако есть небольшая проблема — самая нижняя левая вышка зависит от стартовой вышки, которая во всех запросах разная. Это легко решается, если мы будем считать, что стартовая вышка всегда находится в точке с координатами $(0, 0)$. При известных настоящих координатах стартовой вышки S в запросе, мы можем просто сдвинуть точки A и B в $A - S$ и $B - S$, где минус — это покоординатное вычитание.

Теперь у нас есть выпуклая оболочка всех 2^n вышек. Дальше для каждого запроса можно проверить принадлежность точки A этой оболочке за $O(\log(n))$ простым бинарным поиском.

Подзадача 5

В этой подзадаче всё то же самое, что и в подзадаче 4, но точки A и B в запросах не совпадают. Сначала надо проверить, лежат ли эти точки строго внутри выпуклой оболочки. Если да, то ответ сразу -1. Иначе, надо уметь находить касательные из точки к выпуклой оболочке. Есть несколько известных алгоритмов поиска таких касательных за $O(\log(n))$. Когда в очередном запросе мы нашли касательные из точек A и B, у нас есть 2 случая — отрезок AB строго пересекает выпуклую оболочку, и не пересекает. Проверить AB на пересечение можно следующим образом:

- Предположим, что A лежит строго снаружи оболочки. Пусть лучи AA_1 и AA_2 — касательные к этому многоугольнику из точки A (здесь A_1 и A_2 это вершины многоугольника). Тогда отрезок AB строго пересекает этот многоугольник тогда и только тогда, когда отрезки AB и A_1A_2 строго пересекаются.

Если AB не пересекается с выпуклой оболочкой, то ответом будет $|AB|$. Иначе, надо найти кратчайший маршрут, который начинается в точке A, идёт по одной из касательных, проходит по сторонам многоугольника до другой касательной, и, наконец, доходит до точки B. Таких маршрутов существует всего четыре. Длины их всех можно найти префиксными суммами по сторонам многоугольника.

Здесь много частных случаев, когда A и B лежат на границе многоугольника. Их надо аккуратно рассмотреть.

Подзадача 6

В этой подзадаче A всегда совпадает с B, но на этот раз L и R разные для всех запросов. То есть нам надо как-то эффективно строить текущую выпуклую оболочку, состоящую из векторов от L до R. Давайте отвечать на запросы в оффлайне с помощью алгоритма Мо. Заведём декартово дерево, в котором ключами будут векторы. Мы можем вставлять и удалять векторы из декартово дерева за $O(\log(n))$. А также все векторы в этом дереве всегда будут отсортированы по полярному углу. Это значит, что мы можем узнать координаты i-й точки выпуклой оболочки тоже за $O(\log(n))$, просто покоординатно просуммировав первые i векторов.

Теперь мы умеем поддерживать актуальную выпуклую оболочку суммарно за $O(q * \sqrt{n} * \log(n))$. Чтобы проверить для каждого запроса, лежит ли точка A внутри оболочки, можно воспользоваться бинарным поиском из 4 подзадачи суммарно за $O(q * \log^2(n))$.

Подзадача 7

Чтобы решить эту подзадачу, достаточно было строить выпуклую оболочку и находить касательные за $O(n)$ на запрос.

Подзадачи 8-9

Чтобы получить полное решение, объединим решения подзадач 5 и 6. Так как мы можем узнавать координаты любой вершины выпуклой оболочки за $O(\log(n))$, то мы также умеем находить касательные за $O(\log^2(n))$. Нам ещё нужно находить расстояние между вершинами выпуклой оболочки, но это тоже делается с помощью декартово дерева.

Итоговая асимптотика — $O(q * \sqrt{n} * \log(n) + q * \log^2(n))$ с огромной константой из-за декартового дерева, что даёт нам 90 баллов.

Чтобы получить все 100 баллов, необходимо было заменить декартово дерево на дерево Фенвика.