Разбор задачи «Нарисуй ломаные»

Для перестановки p обозначим за $a(p)$ количество острых углов среди углов $\angle A_{p_{i-1}} A_{p_i} A_{p_{i+1}}$.

$task = 1$

Нужно найти перестановку p, такую что $a(p) = n - 2$.

Можно за $O(n!)$ перебрать все перестановки.

Для подзадач со случайными точками можно использовать оптимизационные подходы (обсудим позже).

Полное решение этого случая:

- Давайте набирать перестановку, первая точка будет A_1.
- На каждой итерации будем выбирать точку $A_{p_{i+1}}$ (из еще не взятых), такую что длина отрезка $A_{p_i} A_{p_{i+1}}$ максимальная. То есть самую далекую точку.
- Заметим, что $\angle A_{p_i} A_{p_{i+1}} A_{p_{i+2}}$ острый, потому что $A_{p_i} A_{p_{i+1}} \geqslant A_{p_i} A_{p_{i+2}}$.
- Получим перестановку, такую что $a(p) = n - 2$, поэтому это максимум.

Оптимизационные идеи

В подзадачах со случайными точками хорошо работает оптимизация!

Нужно минимизировать функционал $f(p) = -a(p)$ (если $task = 1$) или $f(p) = |a(p) - k|$ (если $task = 3, 4$).

Можно, например, менять случайные 2 элемента перестановки и пересчитывать функционал за $O(1)$. Далее используя локальную оптимизацию или отжиг получится ответ.

Чтобы покрывать все значения k, можно использовать «почти дискретную непрерывность». При локальном изменении функция меняется на $\leqslant 6$. Поэтому можно минимизировать функцию, а потом наоборот ее максимизировать. Таким образом, если запоминать ответы для всех значений, которые получаются, довольно скоро покроются все нужные k.

$x_i < x_{i+1}, y_i < y_{i+1}$

Основная идея про монотонность:

- Допустим у нас есть три точки $A_1(x_1, y_1)$, $A_2(x_2, y_2)$, $A_3(x_3, y_3)$, такие что либо $x_1 < x_2 < x_3$, либо $x_1 > x_2 > x_3$ и либо $y_1 < y_2 < y_3$, либо $y_1 > y_2 > y_3$.
- Тогда угол $\angle A_1 A_2 A_3$ всегда тупой, а углы $\angle A_2 A_1 A_3$ и $\angle A_2 A_3 A_1$ всегда острые.

Тогда работает такое решение:

- Для максимизации просто рассмотрим перестановку $1, n, 2, n-1, 3, n-2, \ldots$.
- Чтобы получать конкретное k подойдет

$$[1, k+2, 2, k+1, 3, k, \ldots], [k+3, k+4, \ldots, n]$$

или

$$[k+2, 1, k+1, 2, k, 3, \ldots], [k+3, k+4, \ldots, n]$$

Монотонные подпоследовательности

- Любую перестановку длины $n \leqslant 2x^2$ можно разделить на $\leqslant 2x$ возрастающих или убывающих подпоследовательностей. Алгоритм:
  - Найдем НВП длины k за $O(n \log n)$.
  - Если $k \leqslant 2x$, то автоматически получаем разбиение на k НУП (например, из алгоритма с бинпоиском).
  - Если $k > 2x$, то удалим эту НВП и перейдем к перестановке длины $\leqslant 2x^2 - 2x - 1 \leqslant (x-1)^2$.
- Общее время построения $O(n\sqrt{n}\log n)$.

$task = 2$

- Нам нравятся монотонные последовательности — можно делать много тупых углов.
- Найдем разбиение множества точек на $\leqslant \sqrt{2n} \leqslant 400$ монотонных подпоследовательностей.
- Можно соединить подпоследовательности, количество острых углов будет $\leqslant 2\sqrt{2n}$.

$task = 3$

- В монотонных подпоследовательностях мы можем получать любые количества острых углов.
- Хочется склеить ломаные между подпоследовательностями. Проблемы будут на границах (непредсказуемое изменение количества острых углов).
- Если в каждой подпоследовательности зафиксировать первый, второй, последний, предпоследний элемент, тогда при склеивании количество острых углов будет предсказуемым.
- Теперь можно будет получать любые k, но определенной четности (из-за фиксирования).
- Если попробовать несколько случайных вариантов, покроются все четности.

$task = 4$

- Делаем такие же примеры.
- Для всех них вместе можно аккуратно посчитать хеши.
- Ранее описанное решение сразу обеспечивает любое $k \in [5\sqrt{2n}, n - 5\sqrt{2n}]$.
- Можно аккуратно улучшить константу до 2, разобрав случаи склеивания.
- Общая асимптотика $O(n\sqrt{n}\log n + qn)$.

Разбор задачи «Доска улик»

Общее замечание: Развернём задачу. Вместо того, чтобы «собирать» дерево с нуля, будем наоборот, удалять рёбра из дерева. А после развернём полученный нами ответ. Далее решаем задачу именно в формате «удалить дерево». Список стикеров в вершине дерева далее будем называть стеком, на вершине стека всегда будет стикер, который должен быть удалён первым.

- **Подгруппа №1**

  Подгруппа решается полным перебором. Есть $(n-1)!$ возможных порядков удаления рёбер. Каждый из порядков тривиально проверяется за $\mathcal{O}(n)$.

  Асимптотика: $\mathcal{O}(n!)$

- **Подгруппа №2**

  Бамбук.

  Динамика по префиксу. $dp[v][i]$ — булевое значение: можно ли удалить бамбук из вершин от 1 до v, но если стек для вершины v содержит единственный элемент $c_{v,i}$ ($i \in \{0, 1\}$).

  База динамики: $dp[1][0] = dp[1][1] = true$

  Пересчёт динамики: надо перебрать все четыре варианта с какими элементами стеков удалится ребро между v и $v-1$, если это ребро можно удалить пересчитываемся через $dp[v-1][i]$.

  Ответ существует, если $dp[n][0] = true$, иначе — нет.

  Чтобы восстановить ответ надо для каждого ребра запомнить с какими i_1, i_2 оно удалялось. Сначала удалить все рёбра вида $i_1 = 0, i_2 = 0$, потом $i_1 = 1, i_2 = 0$, $i_1 = 0, i_2 = 1$ и $i_1 = 1, i_2 = 1$.

  Асимптотика: $\mathcal{O}(n)$

- **Подгруппа №3**

  Звезда.

  Пройдёмся по листам от 2 до n. Пусть сейчас рассматриваем лист v. В массиве c_1 найдём минимальное число $\geqslant w_v - c_{v,1}$. Несложно понять, что именно с этим числом оптимально будет удалить ребро ведущее к v. После чего удалим найденное число из c_1. В реализации поддерживаем c_1 в `std::multiset`, и используем метод `lower_bound`.

  Нужный порядок удаления рёбер соответствует исходному порядку чисел в c_1.

  Асимптотика: $\mathcal{O}(n \log n)$

- **Подгруппа №4**

  Две звезды, основания соединены ребром.

  Решаем две звезды аналогично группе 4. После в обоих основаниях останется по одному числу. Если они позволяют удалить ребро между основаниями, ответ Yes, иначе No. Подходящий порядок удаления рёбер: все рёбра первого ежа до вершины основания, все рёбра второго ежа до вершины основания, ребро между основаниями, все рёбра первого ежа после вершины основания, все рёбра второго ежа после вершины основания.

  Асимптотика: $\mathcal{O}(n \log n)$

- **Подгруппа №5**

  Значения чисел в вершинах возрастают.

  Если в какой-то момент какое-то ребро можно удалить, то и далее его всегда можно будет удалить. А значит в любой момент времени допустимо удалять любое возможное ребро. Итого решением будет жадный алгоритм: ищем любое ребро которое можно удалить и удаляем из дерева, повторяем $n-1$ раз. Если в какой-то момент нет готовых к удалению рёбер, ответ No.

  Асимптотика: $\mathcal{O}(n^2)$

- **Подгруппа №6**

  Заметим, что для произвольного теста, верен следующий факт: если развернуть все стеки оригинального теста и назвать это *новым* тестом, то если для оригинального теста ответа не существует, то не существует и для нового. А если ответ существует, то ответом на *новый* тест будет развёрнутый ответ для оригинального теста. Из этого наблюдения решение группы 6 напрямую следует из решения группы 5: нужно развернуть все массивы, а также итоговый ответ.

  Асимптотика: $\mathcal{O}(n^2)$

- **Полное решение**

  Полное решение основано на одной ключевой идее: «Для любого способа назначить каждому ребру пару элементов стеков из смежных вершин (без повторений), найдется порядок удаления рёбер такой, что каждое ребро будет удалено вместе с назначенными ему элементами».

  Доказательство ключевой идеи: Рассмотрим любое назначение. Каждая вершина указывает на одно смежное ребро, которому назначен верхний элемент стека, как «наполовину готовое к удалению». Так как вершин n, а рёбер $n-1$, найдётся ребро на которое будут указывать дважды, обе смежные вершины. Именно это ребро и можно будет удалить. После удаления дерево распадается на два других, и в них применяется то же рассуждение.

  После ключевой идеи задача распалась на две части.

  1. Построить любое корректное назначение рёбер элементам стеков
  2. Восстановить порядок удаления рёбер

  Для построения назначения используем жадный алгоритм, похожий на решение группы 3. Выпишем порядок вершин любого обхода дерева. Будем перебирать вершины в перевёрнутом порядке обхода (с листьев). Пусть сейчас перебираемая вершина v. Тогда единственному оставшемуся числу в стеке вершины v назначим минимальное подходящее число ($\geqslant w_v - c_v$) из вершины родителя.

  Для реализации будем в каждой вершине хранить `std::multiset` чисел. Будем делать `lower_bound` и `erase` в нём.

  Восстановление: для каждого ребра поддерживаем насколько оно готово к удалению (0/1/2). Рёбра готовые к удалению храним в стеке. Итеративно достаём любое ребро из стека, удаляем его и увеличиваем «готовность к удалению» у не более 2 других рёбер. Если какое-то из них становится готовым к удалению, добавляем его в стек.

  Финальная асимптотика: $\mathcal{O}(n \log n)$

Разбор задачи «Больше подарков хороших и разных»

Подгруппа 1. $n \leqslant 100, k \leqslant 10$.

Используем динамическое программирование.$dp[i]$ = наименьшее количество подотрезков на префиксе длины i, при этом $dp[1] = 1$.

Пересчёт для состояния i: $dp[i] = \min(dp[i], dp[j]+1)$, если подотрезок $[j+1, i]$ содержит не более t различных чисел.

Если пересчитывать данную динамику с конца, то можно проверять условие за $O(1)$, поддерживая количество различных чисел в подотрезке. Таким образом получается решение за $O((nk)^2)$ или $O((nk)^2 \log(nk))$

Подгруппа 2. $t = 1$.

Разобъём исходный массив на блоки одинаковых подряд идущих чисел. Пусть количество блоков будет — cnt. Тогда, ответом будет являться $cnt \cdot k$, если первый блок не совпадает с последним, иначе $(cnt-1) \cdot k + 1$

Доказательство работы Жадного решения.

Рассмотрим некоторое оптимальное разбиение, пусть оно отличается от жадного. Найдём первую позицию в которой есть различия. В таком случае длина подотрезка в оптимальном оптимальном разбиении меньше, чем в жадном. Сдвинем правую границу данного подотрезка до позиции в нашем жадном решении. Разбиение останется корректным и количество подотрезков не увеличится. Продолжая данный процесс, оптимальное разбиение превратится в жадное, причём количество подотрезков не увеличится. Из этого следует, что жадное разбиение является оптимальным.

Подгруппа 3. $n \leqslant 1000, k \leqslant 1000$.

Построим полностью повторённый массив размера nk. С помощью жадного алгоритма наберём минимальное число подарков. Решение за $O(nk)$ или $O(nk \log(nk))$.

Подгруппа 4. $n \leqslant 1500, k \leqslant 10^6$.

Рассмотрим следующий алгоритм — с позиции i в исходном массиве будем жадно набирать подарки до конца текущей стопки. После чего будем добирать из следующей стопки, пока это не увеличивает количество подотрезков. Для каждой позиции i в исходном массиве запомним позицию следующую за той, на которой мы остановимся, а так же количество подотрезков полученных в процессе и назовём эти величины $go[i]$ и $cnt[i]$.

Насчитаем массивы go и cnt за $O(n^2 \log n)$ втупую. Теперь, чтобы посчитать ответ заведём параметр cur, который будет отвечать за позицию в массиве, до которой мы оптимально разбили массив. Идём циклом от 1 до k, на каждой итерации прибавляем к ответу $cnt[cur]$, а cur меняем на $go[cur]$. Подсчёт ответа получается за $O(k)$, прибавляем предпосчёт и получаем $O(n^2 \log n + k)$

Подгруппа 5. $k \leqslant 10^6$.

Научимся находить массивы go и cnt за $O(n)$. Для этого воспользуемся методом двух указателей, чтобы для позиции i находить максимальную длину подотрезка, который содержит не более, чем t чисел. Назовём эту длину — len. Считая массивы go и cnt с конца исходного массива, пересчёт для позиции i можно производить следующим образом:

- Если $i + len \geqslant n - 1$, то $go[i] = (i + len)\%n + 1$, где % операция взятия числа по модулю, а $cnt[i] = 1$
- Иначе, $go[i] = go[i+len+1]$, где % операция взятия числа по модулю, а $cnt[i] = cnt[i+len+1]+1$

Считая ответ как в подгруппе 4, получаем решение за $O(n + k)$ или $O(n \log n + k)$

Полное решение.

Оптимизируем подсчёт ответа. Заметим, что если $k > n$, то при подсчёте ответа величина cur примет некоторые значения несколько раз. Найдём первое повторение. Дальше найдём количество шагов, за которое произошло это повторение, а так же величину прибавленную к ответу за это время. Разделив оставшееся число стопок на количество шагов, найдём сколько ещё раз произойдет повторение, а оставшиеся в конце стопки обработаем отдельно

Итого решение за $O(n)$ или $O(n \log n)$

Разбор задачи «Большая хурма»

Для начала заметим две полезные вещи:

- Максимизировать сумму взятых элементов, это то же самое, что максимизировать разность между тем, что взял ты, и что взял твой оппонент.
- Пока на столе есть ещё хотя бы один кусочек, оба человека съели одинаковое количество, то есть разность между ними равна нулю.

Для начала решим задачу за $O(n^2 \cdot \max\limits_i(w_i))$, используя динамическое программирование:

Обозначим dp[l][r][dif] = разность между тем, сколько возмет первый и тем, сколько возьмет второй, если будут играть на отрезке элементов [l, r], при чем первый начнет есть через dif секунд, после второго (dif может быть отрицательным)

Тогда база динамики: dp[i][i-1][dif] = 0, и ответ лежит в dp[0][n-1][0]

От знака dif зависит, кто будет первым принимать решение о том, какой кусочек взять. Таким образом пересчеты:

Для dif $\leqslant 0$: dp[l][r][dif] = max из

- $dp[l + 1][r][dif + w[l]] + w[l]$
- $dp[l][r - 1][dif + w[r]] + w[r]$

Для для dif > 0: dp[l][r][dif] = min из

- $dp[l + 1][r][dif - w[l]] - w[l]$
- $dp[l][r - 1][dif - w[r]] - w[r]$

В такой динамике $|\text{dif}| \leqslant \max\limits_i(w_i)$, то есть всего $O(n^2 \cdot \max\limits_i(w_i))$ состояний и по 2 перехода из каждого.

Чтобы решить подгруппы с ограничением $w_{i+1} \leqslant 2 \cdot w_i$ докажем следующий факт: в таких ограничениях, для любого достижимого состояния (l, r, dif) выполнено $|\text{dif}| \leqslant w_{r+1}$, (исключение: если $r = n - 1$, то $|\text{dif}| \leqslant w_{n-2}$)

Чтобы это доказать, заметим, что при любом переходе dif меняется в сторону нуля, то есть его модуль либо уменьшается, либо становится не больше, чем w_i (где w_i - последний взятый кусочек). Таким образом, при переходах [l, r] $\to$ [l+1, r] соотношение сохраняется в любом случае, а при переходе [l, r] $\to$ [l, r-1] сохраняется, так как $w_{r+1} \leqslant 2 \cdot w_r$

Таким образом, для каждого l существует всего $O(\sum\limits_{i=0}^{n-1} w_i) = O(W)$ достижимых пар r, dif, то есть всего $O(n \cdot W)$ состояний, и два перехода из каждого.

Теперь, чтобы решить полную задачу, поменяем переходы в динамике так, чтобы они сохраняли соотношение $|\text{dif}| \leqslant w_{r+1}$

Для этого заметим, что если один человек взял несколько кусочков до передачи хода оппоненту, он всегда мог это сделать, сначала взяв самые маленькие кусочки, а затем самые большие. В связи с этим, оставим переход [l, r] $\to$ [l+1, r], так как они сохраняют инвариант, а вместо переходов [l, r] $\to$ [l, r-1], добавим переходы [l, r, dif] $\to$ [l, r', dif'], то есть возьмем столько наибольших элементов, чтобы ход перешел оппоненту, или игра закончилась.

Такие переходы сохраняют инвариант $|\text{dif}| \leqslant w_{r+1}$, то есть в такой динамике $O(n \cdot W)$ состояний, и всё ещё два перехода из каждого.

Заметим так же, что значения r' зависят только от r и dif, но не от l, то есть их можно насчитать за $O(W \cdot \log(n))$.

Полученное решение работает за $O(n \cdot W)$, чего достаточно, чтобы пройти все тесты.