XXXI Всероссийская олимпиада школьников по информатике

Разбор задач

Иннополис, 11 – 17 апреля 2019 г.

Задача 1. Постановочное фото

- Идея задачи — Михаил Пядеркин
- Подготовка тестов — Павел Маврин, Егор Чунаев
- Разбор задачи — Михаил Пядеркин

- Изначально имеется нераскрашенная полоска
- При выходе на сцену некоторый отрезок целиком перекрашивается в новый цвет
- Требуется определить отрезки и их порядок так, чтобы получилась требуемая раскраска

- Рассмотрим итоговую раскраску полоски
- В ней обязательно существует цвет, который образует непрерывный отрезок (он соответствует последней делегации)
- Если существует цвет, который образует непрерывный отрезок, то он может быть последним
- Найдем любой такой цвет и «вырежем» его из полоски

- Научимся искать начало и конец каждого отрезка для каждого из цветов
- Для этого заметим, что для каждого цвета можно рассмотреть отрезок от первого вхождения этого цвета до последнего вхождения
- Отрезок длинее брать нет смысла, а отрезок короче нам не подойдет

- Любые два из полученных отрезков либо не пересекаются, либо вложены (из двух пересекающихся отрезков один всегда соответствует более поздней делегации)
- Таким образом, они образуют конструкцию, похожую на правильную скобочную последовательность

- Будем идти по полоске слева направо и поддерживать множество уже открытых цветов в стеке
- При встрече нового цвета добавляем его в стек
- При встрече цвета, который уже находится в стеке, будем удалять цвета с вершины стека и выписывать их в ответ, пока не дойдем до нужного
- Если при этом мы пытаемся выписать в ответ цвет, который еще не закончился целиком, то ответа не существует (`rgrgr`)

Задача 2. Беспилотное такси

- Идея задачи — Николай Калинин
- Подготовка тестов — Егор Чунаев
- Разбор задачи — Егор Чунаев, Дмитрий Саютин

Постановка задачи

- Есть поле размера $n \times m$
- Есть три типа запросов
  - Присвоить всем числам в строке число 0
  - Присвоить всем числам в столбце число 0
  - Найти кратчайший путь между двумя клетками, если можно ходить только по клеткам, число в которых не превосходит k_i
- Перед каждым запросом ко всем клеткам поля прибавляется число 1

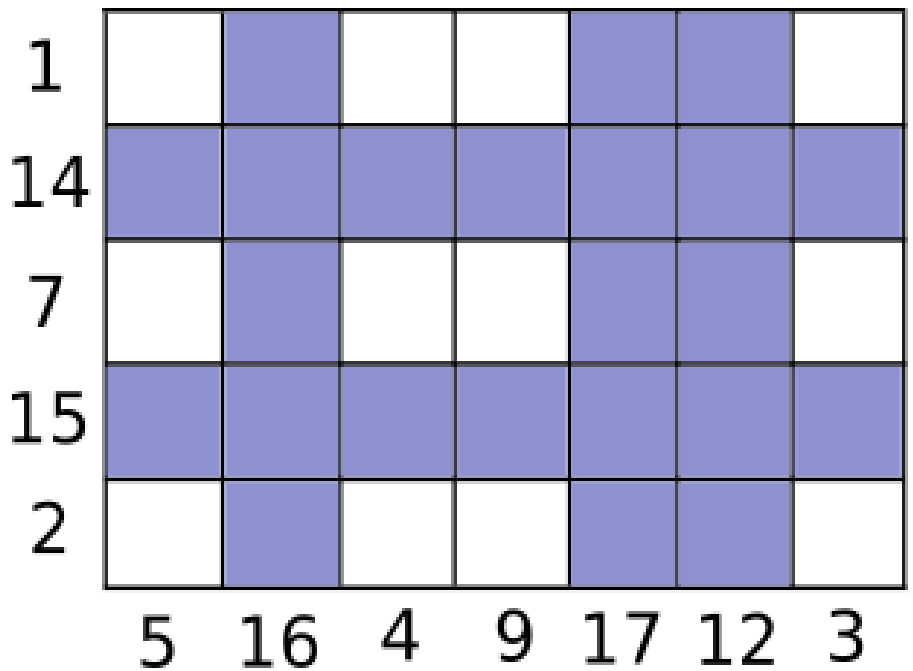
Первая подзадача

- Все поле можно явно хранить в памяти и на каждый запрос кратчайшего пути запускать поиск в ширину
- Асимптотика $O(nmq)$

Ключевая идея

- Немного изменим модель задачи
- Будем записывать в строках и столбцах последний момент времени, когда строка или столбец была очищена.
- Изначально все строки и столбцы были очищены в момент времени 0
- Пусть нам в момент времени x надо ответить на запрос для такси с проходимостью k
- Тогда это такси может ездить по всем строкам и столбцам, в которых записанное число не меньше $x - k$.

Ключевая идея



$x = 22, k = 12$

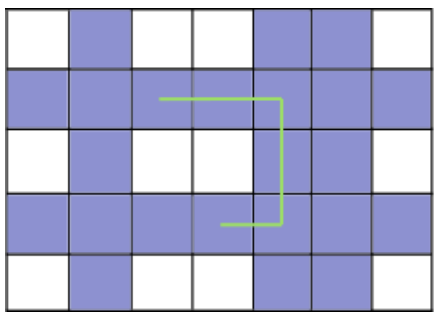
$x - k = 10$

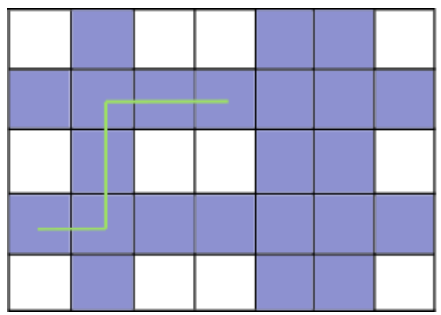
Ключевая идея

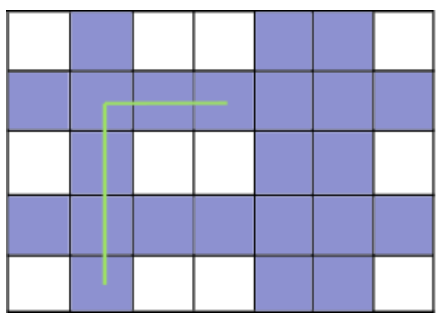
- В каждом из запросов область допустимых клеток представляет из себя «решетку», то есть допустимые клетки являются объединением некоторого количество строк и столбцов.
- Кратчайший путь между узлами этой решетки равен $|r_1 - r_2| + |c_1 - c_2|$ и похож на русскую букву «Г».

Вид кратчайшего пути

- Кратчайший путь между клетками лежащими на звеньях решетки, с точностью до поворота, имеет один из следующих видов:







Вид кратчайшего пути

- Поскольку разбирать все возможные случаи сложно, самый простой способ обработать этот случай — перебрать ближайший узел для каждой из двух точек и взять минимум по всем 4 вариантам пути.
- Пути не существует, если либо начальная клетка, либо конечная не проходима, или не выполняется не один из способов выше.

Как получать ответ из ключевой идеи

- Сначала проверим что ответ существует, для этого проверим что изначальные клетки проходимы, а затем проверить одно из двух условий:
  - Одновременно есть хоть одна активна строка и один активный столбец
  - Обе клетки принадлежат одной группе строк или столбцов.
- Чтобы проверить что клетки лежат в одной группе строк или столбцов надо проверить что минимум на отрезке строк или столбцов превосходит $x - k$.

Как получать ответ из ключевой идеи

- Если клетки находятся не в одной группе строк или столбцов, то надо найти ближайшие узлы к этим клеткам.
- Для этого надо уметь находить ближайшие слева и справа активные строки или столбцы.

Вторая подзадача

- Чтобы решить вторую подзадачу надо реализовать описанные выше операции за линейные проходы по массивам строк и столбцов
- Асимптотика $O((n + m) \cdot q)$

Третья подзадача

- Чтобы решить третью подзадачу для запросов с $x - k \leqslant 0$ надо сразу выводить сумму разности координат
- Иначе нас интересует не более q строк и столбцов
- Асимптотика $O(q^2)$.

Пятая подзадача

- Чтобы решить пятую подзадачу надо реализовать минимум на отрезке при помощи какой-нибудь структуры данных, искать ближайших слева и справа не нужно, ибо ответ существует только для клеток находящихся в одной группе строк.

Шестая подзадача

- Чтобы решить шестую подзадачу, в которой все k равны, надо все так же реализовать минимум на отрезке
- Для поиска ближайших слева и справа можно использовать `std::set`, поскольку каждая строка и столбец добавляется в него и активна только для следующих k запросах, поэтому искать ближайших слева и справа можно при помощи `lower_bound`

Четвертая подзадача и полное решение

- Для решения четвертой и седьмой подзадачи надо реализовать спуск по дереву для поиска ближайшего слева и справа
- В зависимости от эффективности решения, оно пройдет либо обе подзадачи, либо только четвертую

Задача 3. Экспресс 20/19

- Идея задачи — Глеб Евстропов
- Подготовка тестов — Николай Будин, Дмитрий Саютин
- Разбор задачи — Николай Будин

Постановка задачи

- Есть ориентированный взвешенный ацикличный граф
- Есть запросы, каждый запрос характеризуется вершиной f_i и расстоянием r_i
- На каждый запрос нужно ответить, существует ли путь из вершины 1 в вершину f_i с длиной лежащей в отрезке $[r_i, r_i \cdot \frac{p}{p-1}]$ $(2 \leqslant p \leqslant 20)$

Решение первой подзадачи

- Будем хранить в вершине список длин путей из вершины 1 в текущую
- Рассматриваем вершины в порядке топологической сортировки, для ребра v u d, добавляем в вершину u все возможные расстояния до вершины v, увеличенные на d

Решение второй подзадачи

- Не будем хранить одну и ту же длину несколько раз
- Не будем хранить длины больше чем $maxr \cdot \frac{p}{p-1}$

Решение задачи

- Рассмотрим множество расстояний из запросов, которые будут удовлетворены длиной x
- Это расстояния лежащие в отрезке $[x \cdot \frac{p-1}{p}, x]$
- Мы храним в вершине множество длин, им соответствует множество отрезков расстояний, которые удовлетворены
- Несложно доказать, что достаточно оставить любое подмножество этих отрезков, которые в объединении будут давать то же множество удовлетворенных расстояний

Решение задачи

- Будем набирать отрезки жадно, перебирая их правые границы от больших к меньшим
- Если правая граница очередного отрезка лежит левее левой границы последнего взятого, возьмем его и предыдущего перед ним
- При этом, мы взяли два отрезка, и величина правой границы уменьшилась в $\frac{p}{p-1}$ раз
- Значит, количество отрезков, которое мы оставим в каждой вершине, не превышает $2 \cdot \log_{\frac{p}{p-1}} maxr$

Задача 4. Чёрная дыра

- Идея задачи — Михаил Тихомиров
- Подготовка тестов — Михаил Тихомиров, Павел Кунявский, Илья Збань
- Разбор задачи — Михаил Тихомиров

Постановка задачи

- Загадано целое число x от 1 до n.
- Можно задавать запросы вида «верно ли, что $x \leqslant y$?».
- Не более, чем один ответ на запрос может быть неверным.
- Программа жюри может не загадывать x заранее и отвечать на запросы адаптивно, чтобы хотя бы одно значение x оставалось корректным.
- Требуется точно определить x, уложившись в ограничение на число запросов.

Подзадачи 1 и 2

$q \leqslant 3\lceil \log_2 n \rceil$: обычный бинпоиск, повторяем каждый запрос три раза.

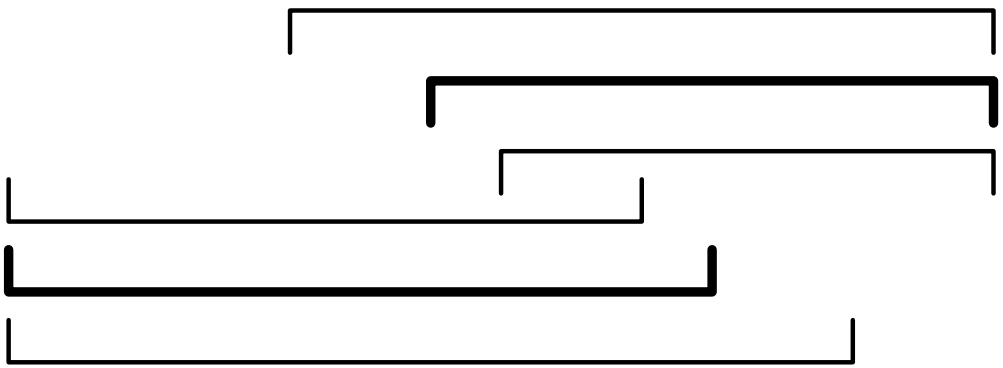
$q \leqslant 2\lceil \log_2 n \rceil + 1$: делаем третий запрос, только если первые два ответа были разными, после этого нам не могут соврать.

Подзадача 3 ($n \leqslant 4$) в подарок!

Подзадачи 3–6: переборы с оптимизациями

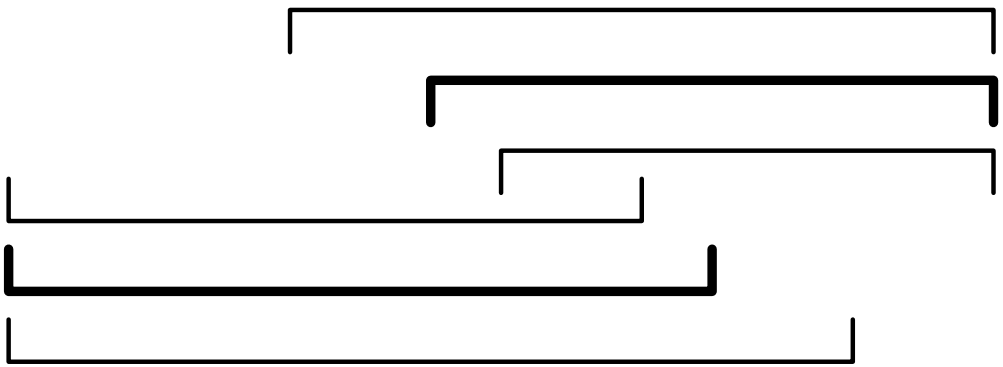
Перебираем стратегии, пользуемся тем, что запросов мало (не больше 9), сжимаем состояния, и т.д. и т.п.

$O(n^5)$: 30 (+15) баллов



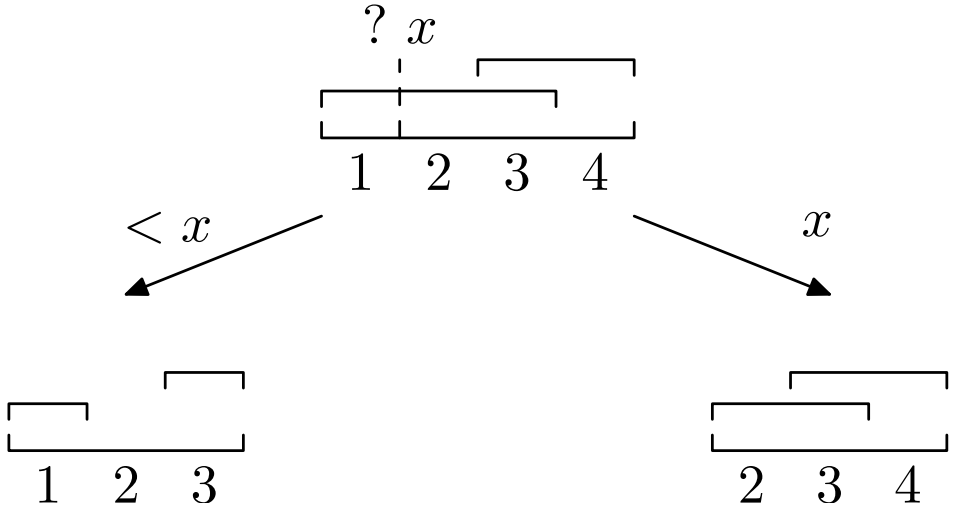
Пусть нам сказали, что x лежит на каких-то префиксах и суффиксах. Заметим, что про второй по величине префикс и суффикс нам не могли соврать, поэтому бОльшие префиксы и суффиксы не добавляют новой информации. Всего $O(n^4)$ состояний поиска.

$O(n^5)$: 30 (+15) баллов



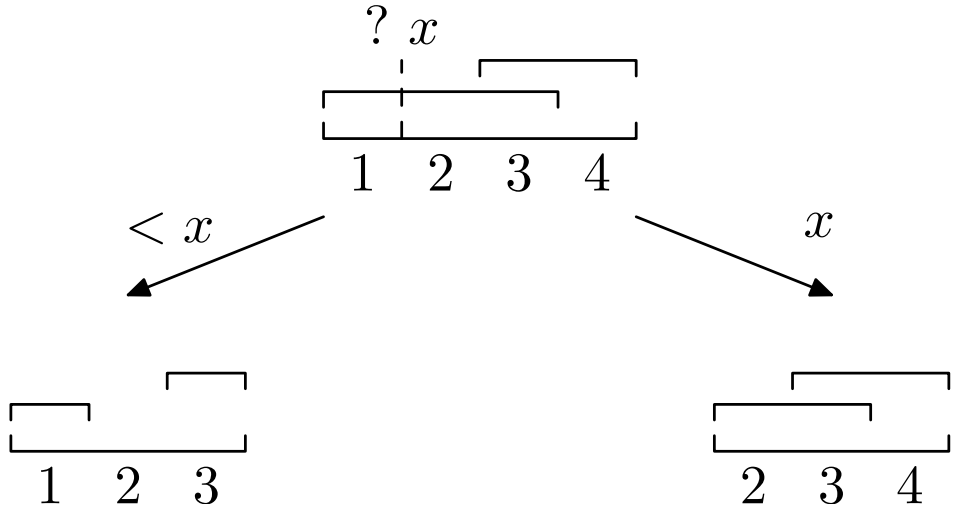
Динамическое программирование: наименьшее число запросов для данных длин двух префиксов и двух суффиксов. $O(n)$ переходов (рассматриваем все ходы), суммарная сложность $O(n^5)$.

$O(n^5)$: более внимательно



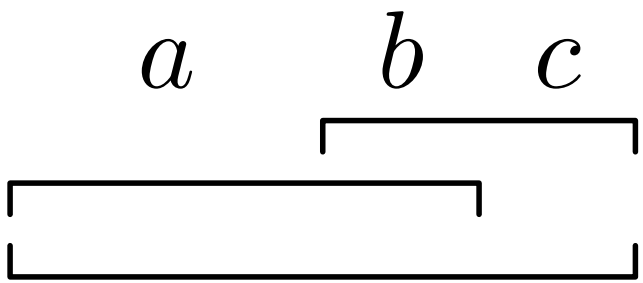
Посмотрим на состояния более внимательно.
Пересечение второго префикса и суффикса точно содержит загаданное число.

$O(n^5)$: более внимательно



Если наименьшие префикс и суффикс не пересекаются, то к этому моменту нам уже соврали, и можно искать ответ в оставшихся диапазонах обычным бинпоиском.

$O(n^5)$: более внимательно



В противном случае будем обозначать длину пересечения b, а длины оставшихся частей префикса и суффикса a и c соответственно.

$O(n^4)$: 35–40 (+15) баллов

Заметим, что от сдвига всех границ на x ответ не меняется, а стратегия переносится на x. $O(n^3)$ состояний, сложность $O(n^4)$.

$O(n^3 \log n)$: 40–48 (+15) баллов

Заметим, что для любого состояния оптимальное количество запросов после ответа $< x$ — неубывающая функция от x; аналогично, количество запросов после $\geqslant x$ — невозрастающая функция.

Оптимальный запрос для состояния — точка минимума для максимума из этих двух функций, ищем бинпоиском. Переход ДП за $O(\log n)$ вместо $O(n)$.

$O(n^3)$: 48–55 (+15) баллов

Рассмотрим состояние с $b > 0$. Можно заметить, что при увеличении c положение оптимального запроса может сместиться только вправо.

Переходы для всех состояний с фиксированными a, b можно сделать за $O(n)$, двигая указатель.

$O(n^2 \log n)$: 60–70 (+15) баллов

Поменяем местами ответ и один из параметров: $maxc(k, a, b)$ — максимальное c, что в состоянии (a, b, c) можно угадать число за k ходов, или $-\infty$, если нельзя ни при каком неотрицательном c.

$O(n^2 \log n)$: 60–70 (+15) баллов

- Если $maxc(k-1, a, b) \geqslant 0$, то $maxc(k, a, b) \geqslant maxc(k-1, a, b) + 2^{k-1} - b$: отрежем $2^{k-1} - b$ от c, в одной из веток запустим бинпоиск на 2^{k-1} элемент.
- Если $a \geqslant 2^{k-1} - b$, то $maxc(k, a, b) \geqslant maxc(k-1, a-(2^{k-1}-b), b)$: аналогично.

$O(n^2 \log n)$: 60–70 (+15) баллов

- Если для $0 \leqslant x \leqslant b$ верно $maxc(k-1, a, x) \geqslant b - x$, то $maxc(k, a, b) \geqslant maxc(k-1, x, b-x)$: режем часть длины b на x и $b - x$.

Можно соединить со всеми предыдущими оптимизациями, итоговая сложность $O(n^2 \log n)$. При аккуратной реализации $O(n^2)$ памяти.

$o(n^2 \log n)$?

???

Сохраняем решающее дерево

Будем пытаться искать стратегию локально и сохранить ее в код. Заметим, что если мы знаем стратегию для максимального $n = f(k)$ с фиксированным числом запросов k, мы умеем решать и все меньшие n.

k	0	1	2	3	4	5	6	7	8	9	10	11
$f(k)$	1	1	1	2	2	4	7	12	22	40	76	142

k	12	13	14	15	16	17	18	19
$f(k)$	268	500	944	1788	3389	6444	12286	23464

Уменьшаем решающее дерево

Дерево для k запросов имеет 2^k вершин, при $k \sim 10 - 12$ можно просто сохранить все дерево целиком.

При больших k будем экономить:

- если префикс и суффикс текущего состояния не пересекаются ($b = 0$), в такой ветке дерева происходит обычный бинпоиск, такую ветку можно не сохранять.

Уменьшаем решающее дерево

- Если разным вершинам дерева соответствуют одинаковые состояния ДП, сохраним их только один раз.
- Одинаковые вершины деревьев для разных k можно объединить.

Эталонное дерево

- 1978 вершин, 72 Кб кода
- построение за 3 минуты, 6 Гб используемой памяти

Вопросы?

Задача 5. Гирлянда

- Идея задачи — Александр Кленин
- Подготовка тестов — Максим Ахмедов, Георгий Корнеев
- Разбор задачи — Георгий Корнеев, Александр Кленин

Постановка задачи

- Дана строка из n символов 0 и 1.
- Требуется удалить наименьшее число символов, чтобы получить красивую строку $(0^p1)^q0^p$.
- Если $n \leqslant 15$, можно перебрать все подпоследовательности.

Общая идея

- Будем перебирать p.
- Поскольку строка из 0 и 1, для данного p бежим указателями по исходной и по красивой строке, если символы не совпадают — удаляем символ исходной строки.
- $O(n^2)$

Много или мало единиц

- Если единиц больше половины, оставляем только единицы.
- Пусть единиц k.
- Представим исходную строку как блоки подряд идущих нулей и единиц.
- Переберём подмножества блоков, для каждого варианта проверим за k — $O(k \cdot 2^k)$.
- При фиксированном p будем двигаться по блокам, а не по отдельным символам — $O(nk)$.

Быстрая проверка

- Бежим двумя указателями при фиксированном p.
- Быстро найти позицию следующей единицы — предрасчёт ближайших справа единиц.
- Быстро найти p следующих нулей — двоичный поиск или дерево отрезков, $O(\log n)$ на запрос.
- Для каждого p имеется $\frac{n}{p}$ блоков.
- $O(\sum_{p=1}^{n} \frac{n}{p} \log n) = O(n \log^2 n)$ (как сумма гармонического ряда).
- 100 баллов при аккуратной реализации.

Очень быстрая проверка

- Для каждого j предрассчитаем позиции j-й единицы и j-го нуля.
- При пробеге будем накапливать количество удалённых нулей и единиц (d_0 и d_1).
- Для префикса из i блоков вычислим номер последней единицы $(d_1 + i)$ и последнего нуля $(d_0 + (i + 1) \cdot p)$, получим их позиции из предрасчёта.
- $O(\sum_{p=1}^{n} \frac{n}{p}) = O(n \log n)$.
- Эталонное решение.

Очень быстрая проверка. Вариант 2

- Для каждого j предрассчитаем: позиции j-го нуля, ближайшую справа единицу, номер нуля по позиции.
- Для каждого очередного префикса получим номер первого нуля, добавиим $p - 1$, получим позицию последнего нуля по номеру.
- Тоже $O(n \log n)$, 100 баллов.

Задача 6. Классные парты

- Идея задачи — Дмитрий Михайлин, Елена Андреева
- Подготовка тестов — Павел Маврин, Дмитрий Саютин
- Разбор задачи — Андрей Станкевич, Дмитрий Саютин

Постановка задачи

- Есть m классов по $2n$ человек
- Есть k видов парт, характеризующиеся числами L_i, R_i
- Закупить n парт так, чтобы минимизировать суммарное неудобство

Наблюдение 1

- Если отрезок одной парты $[L_1, R_1]$ полностью вкладывается в отрезок $[L_2, R_2]$ другой парты ($L_2 \leqslant L_1 \leqslant R_1 \leqslant R_2$), то можно использовать парту $[L_2, R_2]$ вместо $[L_1, R_1]$
- Тем самым, можно избавиться от всех вложенных парт
- Тогда все парты можно упорядочить слева направо — то есть получить сортировку одновременно по L_i и R_i

Наблюдение 2

- Предположим, что мы уже закупили парты, и осталось только распределить школьников по ним
- Очевидно, эту задачу стоит решать независимо в каждом классе
- Как парты отсортированы по возрастанию, так и школьников отсортируем по возрастанию роста
- Можно показать, что всегда оптимально дать самую низкую парту двум самым низким школьникам, затем двум следующим школьникам отдать следующую парту, и т.д.

Решение за $\mathcal{O}(nmk)$

- Значит для каждой парты мы знаем кто за ней будет сидеть
- Давайте для каждой такой группы людей просто найдём оптимальную парту перебором парты
- Тем самым получается решение за $\mathcal{O}(nmk)$, что набирает 70 баллов
- Полное решение, а также частичные решения на следующие группы получаются ускорением этого решения

Решение за $\mathcal{O}(nk \log m)$

- Соптимизируем вычисление неудобства парты для конкретной группы людей
- Это можно сделать за логарифмическое время: давайте отсортируем всех людей в этой группе, а также насчитаем префиксные суммы их ростов
- Как вычислить величину неудобства для парты $[L; R]$? Сначала идут школьники меньше чем L, затем идут школьники, которым удобно сидеть за этой партой, после чего идут школьники, которым эта парта велика

Решение за $\mathcal{O}(nk \log m)$

- Давайте с помощью двух бинарных поисков найдём вышеописанные префикс и суффикс, после чего величина неудобства может быть выражена через сумму ростов в префиксе (суффиксе) и количество школьников в этом префиксе (суффиксе)
- Тем самым получаем решение за $\mathcal{O}(nk \log m)$, что проходит 10 группу и набирает 80 баллов

Полное решение

- Мы всё ещё для каждой группы перебираем вид парты, которую нужно купить.
- Если мы вспомним, как получились группы людей, которым мы покупаем парты, то мы увидим, что более поздним групп, нужно купить более «высокую» парту.
- Формально, если пронумеровать парты в порядке от более низких к более высоким, и обозначит за p_i оптимальную парту для i-й группы, то верно, что $p_i \leqslant p_{i+1}$.
- Можно применить оптимизацию «Разделяй и властвуй»

Полное решение

- Давайте перебором всех k парт определим тип парты, которую нужно купить для средней ($x = n/2$) группы школьников
- Затем рекурсивно решим задачу для меньших групп и больших групп, при этом для меньших подгрупп будем искать оптимальную парту только среди $\leqslant p_x$, а для больших только среди $\geqslant p_x$
- Несложно видеть, что глубина рекурсии составляет $\log n$, а суммарный размер задач на уровне линеен

Полное решение

- Тем самым мы получим решение за $\mathcal{O}(sort(nm) + mk \log n)$ если вычислять неудобство парты за линию, что проходит 9-ю группу и получает 78 баллов
- Или за $\mathcal{O}(sort(nm) + (m + k) \log m \log n)$, что является полным решением

Задача 7. Робогольф

- Идея задачи — Павел Кунявский
- Подготовка тестов — Николай Будин, Егор Чунаев
- Разбор задачи — Николай Будин

Постановка задачи

- Есть поле $n \times m$
- На нем расположенны k ловушек
- Два игрока, начиная с некоторой клетки, ходят по очереди
- За ход можно сдвинуть фишку вниз или вправо

Постановка задачи

- Если фишка попадает в ловушку, ответ — стоимость ловушки, если выходит за границы поля, ответ — 0
- Первый игрок хочет минимизировать ответ, а второй — максимизировать
- Нужно посчитать сумму результатов игр по всем возможным стартовым позициям

Первая подзадача

- Решим с помощью метода динамического программирования
- Состояние — текущая клетка и текущий игрок
- $d_{i,j,1} = \min(d_{i+1,j,2}, d_{i,j+1,2})$
- $d_{i,j,2} = \max(d_{i+1,j,1}, d_{i,j+1,1})$
- Ответ — сумма $d_{i,j,1}$ по всем i и j

Вторая подзадача

- Рассмотрим клетку (i, j)
- Пусть в клетках $(i + d_i, j + d_j)$ нет ловушек $(d_i, d_j \leqslant n + 2)$
- Тогда ответ в клетке (i, j) равен 0
- Можно сжать более $n + 2$ подрядыдущих столбцов, не содержащих ловушки, в $n + 2$ столбца, не содержащих ловушки
- Теперь ширина поля равна $O(n \cdot k)$

Решение за $O(n+m+k)$

- Сделаем шахматную раскраску поля
- Решим отдельно для случаев, когда первый игрок делает ходы на белых клетках, и когда он делает ходы на черных клетках
- Рассмотрим клетку (i, j), в которой сделает ход первый игрок
- Если
  в клетках $(i+1, j)$ и $(i, j+1)$ нет ловушек, $d_{i,j} = \min(\max(d_{i+2,j}, d_{i+1,j+1}), \max(d_{i+1,j+1}, d_{i,j+2})) = \max(d_{i+1,j+1}, \min(d_{i+2,j}, d_{i,j+2}))$

Решение за $O(n + m + k)$

- Значит, $d_{i,j}$ почти всегда равно $d_{i+1,j+1}$, кроме случаев, когда в клетке $(i + s_i, j + s_j)$ есть ловушка $(s_i, s_j \leqslant 4)$
- На самом деле, верна даже более строгая оценка $s_i, s_j \leqslant 2$
- Будем поддерживать значения на диагонали в массиве
- Тогда если $d_{i,j} = d_{i+1,j+1}$, соответствующий элемент в массиве не изменится при переходе от одной диагонали к другой

Решение за $O(n + m + k)$

- Сделаем множество интересных клеток — только в них потенциально может измениться ответ по сравнению с той же позицией в массиве на предыдущей диагонали
- Интересные клетки — такие клетки (i, j), что в клетке $(i + s_i, j + s_j)$ есть ловушка $(s_i, s_j \leqslant 2)$
- Теперь будем идти по диагоналям, поддерживать сумму на текущей диагонали, пересчитывать значения в интересных клетках
- Просуммируем сумму по всем диагоналям

Решение за $O(k \cdot \log k)$

- Заметим, что если между двумя диагоналями нет интересных клеток, значения массива между ними не изменяются
- Значит, можно пропускать диагонали, не содержащие интересные клетки
- Вместе массива теперь будем поддерживать значения в хеш таблице или дереве поиска

Задача 8. Поиск идеи

- Идея задачи – Илья Збань
- Подготовка тестов — Илья Збань, Павел Кунявский
- Разбор задачи — Михаил Пядеркин

- Дан сжатый текст и шаблон
- Необходимо посчитать количество вхождений шаблона в текст

- В первых двух подзадачах можно распаковать сжатый текст в строку
- После этого можно решить задачу о поиске подстроки в строке
- квадратичное решение для первой подзадачи и линейное для второй

- В следующей подзадаче сжатый текст представлял собой некоторую строку, повторенную несколько раз $\alpha\alpha\ldots\alpha$
- Если суммарная длина повторений не больше удвоенной длины шаблона, то текст можно полностью распаковать
- В противном случае достаточно распаковать текст до тех пор, пока текст не станет достаточно длинным, и найти явно все вхождения шаблона в этот фрагмент текста
- После этого для вычисления общего числа вхождений можно воспользоваться формулой

- В четвертой подзадаче было выполнено условие $pos_i = L_{i-1}$
- Таким образом, выполняется копирование лишь последнего символа (возможно, много раз)
- За исключением единственного случая, когда шаблон является строкой вида aa ...a, приписывать больше 2000 копий одного символа не имеет смысла — в такой строке не будет вхождений шаблона

- Таким образом, все операции копированиия можно выполнять явно, ограничив размер блока 2000, затем использовать линейный алгоритм поиска вхождений
- Для шаблона вида aa ...a нужно отдельно разобрать случай, когда приписываемый к тексту символ и есть a

- В пятой и шестой подзадаче $pos_i = 1$, то есть текст представляет собой конкатенацию префиксов стартовой строки
- Поскольку шаблон достаточно короткий ($m \leqslant 2000$), можно написать решение за $nm + 10^7$: для каждого копирования посчитать количество вхождений в начале дописанного блока за $\mathcal{O}(m)$ с префикс-функцией, а дальше воспользоваться тем, что после предпросчета мы знаем, сколько вхождений паттерна было на оставшейся части префикса

- В седьмой и восьмой подзадаче $pos_i + len_i - 1 \leqslant L_i$ и $n, m \leqslant 20$. Это дает возможность сделать решение за $2^n \cdot m$: будем хранить текст в виде объединения какого-то числа подотрезков блоков первого типа.
- Каждый блок второго типа не больше, чем удваивает текущее число отрезков. Получаем 2^n отрезков, и мы можем найти число вхождений в каждом подотрезке за $\mathcal{O}(1)$ префиксными суммами после предпросчета и вхождения на границе за $\mathcal{O}(m)$.

- В девятой и десятой подзадаче паттерн состоит лишь из букв a
- Эта подгруппа сильно приближает к полному решению задачи. Можно воспользоваться персистентным декартовым деревом, чтобы сохранить информацию о разжатой строке. Все копирования выглядят как копирование и размножение подотрезка существующей строки, персистентное декартово дерево умеет это делать за $\mathcal{O}(\log^2 L)$

- В вершине декартова дерева храним количество вхождений паттерна и число букв a на границах. Этого хватает для подсчета ответа. Решение за $\mathcal{O}(T) = \mathcal{O}(n \log^2 L)$, число вершин в декартовом дереве

- В одиннадцатой подзадаче произвольный короткий паттерн. Это позволяет сохранить в каждой вершине дерева префикс и суффикс длины m подстроки, в которую она раскрывается, и пересчитывать префикс/суффикс и число вхождений за $\mathcal{O}(m)$

- В двенадцатой подзадаче нужно научиться пересчитывать информацию про вершины дерева за околологарифмическое время
- Будем в каждой вершине хранить, какой максимальный префикс строки, в которую она раскрывается, входит в паттерн (и само вхождение в виде пары чисел). Аналогично для суффикса
- Теперь есть две проблемы — пересчитать число вхождений паттерна при склеивании двух сыновей и пересчитать вхождения префикса и суффикса

- Есть подстрока, полученная конкатенацией двух строк $p[l_1 \dots r_1] + p[l_2 \dots r_2]$, и нужно понять, сколько раз p входит в эту строку. Хотим выбрать суффикс первой подстроки и префикс второй, дающие в сумме p
- Построим дерево префикс-функции строк p и развернутой строки p^R

- Подстроке $p[1 \ldots r_1]$ соответствует вершина в дереве префикс-функции. Поднимаясь до корня в дереве префикс-функции, получаем все суффиксы $p[i \ldots r_1]$, являющиеся префиксами p, т.е. все подходящие суффиксы образуют путь в дереве
- Аналогично все префиксы $p[l_2 \ldots r_2]$ образуют путь во втором дереве

- Есть запрос — даны пути в двух деревьях от корня, нужно найти две вершины u и v на этих путях, таких, что $u + v = m$
- Это можно сделать в оффлайне (или персистентно в онлайне), обходя первое дерево поиском в глубину, и заходя в вершину u в первом дереве делая прибавление в эйлеровом обходе второго дерева в поддерево вершины $m - u$
- Так за $\mathcal{O}(\log m)$ можно пересчитывать количество вхождений паттерна

- Есть подстрока, полученная конкатенацией двух строк $p[l_1 \dots r_1] + p[l_2 \dots r_2]$, и нужно найти ее максимальный префикс, содержащийся в p
- План — построить суффиксный массив строки p, и найти суффикс с наибольшим lcp
- Суффикс с наибольшим lcp — либо первый не меньший лексикографически, либо первый меньший
- Бинпоиском находим эти два суффикса и считаем lcp

- Подсчет lcp делается за $\mathcal{O}(1)$ используя sparse table на массиве lcp суффиксного массива и разбор двух случаев
- Аналогично для поиска максимального суффикса
- Решили подзадачу за $\mathcal{O}(T \log n)$.

- В тринадцатой подзадаче предыдущее решение может получить ТЛ/МЛ из-за большого числа вершин в дереве
- Если использовать правильное дерево поиска, в нем будет не $\mathcal{O}(n \log^2 L)$ вершин, а $\mathcal{O}(n \log L)$ вершин
- Подойдет персистетное 2-3 дерево
- Итого, задача решена за $\mathcal{O}(n \log L \log m)$