XXX Всероссийская олимпиада школьников по информатике

Ульяновск

1 – 7 апреля 2018 года

Поэзия — та же добыча радия.
Владимир Маяковский

Задача 1 «Добыча радия»

- Идея задачи — Андрей Календаров
- Подготовка тестов — Андрей Календаров, Максим Ахмедов
- Разбор задачи — Александр Кленин

Постановка задачи

- Дан массив $n \times m$ различных чисел
- *Подходящий* элемент — максимум одновременно в строке и столбце
- Элементы увеличиваются с сохранением уникальности
- После каждого увеличения вывести количество *подходящих* элементов

Решение на 25 баллов

- После каждого изменения заново найдём все подходящие элементы
- Сложность $O(n^2 \times m^2 \times q)$

Решение на 100 баллов

- Хранить и обновлять позицию максимума для каждой строки и столбца и общий счётчик подходящих элементов
- Для каждого увеличения неподходящего элемента
  - если максимум строки обновился и в той же строке был подходящий элемент, уменьшаем счётчик
  - аналогично для столбца
  - если элемент стал подходящим, увеличиваем счётчик
- Сложность $O(n \times m + q)$

Неэффективное решение на 100 баллов

- Хранить и обновлять множество элементов для каждой строки и столбца и общий счётчик подходящих элементов
- Обновлять аналогично
- Сложность $O(n \times m + q \times (\log n + \log m))$

Решение на 50 баллов

- Аналогично полному, но без динамических массивов
- Сложность по памяти $O(\max(n, m)^2)$

Вопросы?

Задача 2 «Вирусы»

- Идея задачи — Михаил Тихомиров
- Подготовка тестов — Дмитрий Иващенко
- Разбор задачи — Дмитрий Иващенко

Постановка задачи

- n вирусов и n клеток, i-я заражена вирусом i.
- Клетка, заражённая вирусом i, может атаковать клетку, заражённую вирусом j. Если вторая более восприимчива к вирусу i, чем к j, то она перезаразится
- Атаки происходят, пока возможны перезаражения
- Вирус *стабилен*, если остаётся при **любых** последовательностях атак
- Вирус *жизнеспособен*, если остаётся при **какой-то** последовательнсти атак

Решение для $n \leqslant 5$ (33 балла)

- Рекурсивный перебор с состоянием $a[i]$ = «каким вирусом заражена клетка i»
- Перебираем атакующую клетку и клетку, которую атакуют
- Не будем заходить в уже посещённые состояния
- Число состояний $O(n^n)$, итоговая сложность $O(n^{n+2})$

Решение для $t = 1$ (32 балла)

- Когда вирус i стабилен?
- Необходимое условие: если с самого начала клетку i будут атаковать **все** остальные, перезаражения быть не должно
- Значит клетка i наиболее восприимчива как раз к вирусу i
- Заметим, что это условие достаточное. Проверить его можно за $O(n)$.

Ключевая идея решения для $t = 2$

- Если вирус i жизнеспособен, то существует вариант развития событий, где некоторая клетка j заражена им в конце. Назовём такую клетку *«хранителем»*
- Пусть вирус i занимает позицию $pos[j][i]$ в списке клетки j
- Переберём «хранителя» j для вируса i (при этом $pos[j][i] < pos[j][j]$)
- Вирусы k такие, что $pos[j][k] < pos[j][i]$ назовём «опасными», они должны вымереть

Решение за $O(n^4)$ (85 баллов)

- Наблюдение: если вирус «опасный», то соответствующая клетка может никого не атаковать (иначе этот вирус придется «убирать» из двух клеток)
- Итого, нужно проверить, что все «неопасные» вирусы могут как-то перезаразить все опасные
- Проверку можно реализовать за $O(n^2)$: для каждого «опасного» вируса перебрать того, кто его перезаразит

Полные решения

- Сменим порядок перебора: сначала переберём «хранителя», а потом в его списке переберём с конца к началу вирус
- Тогда пересчёт можно организовать за $O(n)$: убирается один «опасный» вирус, добавляется один «неопасный». Достаточно посмотреть, кого может «убрать» новый «неопасный» вирус
- Сложность $O(n^3)$
- Используя технику битового сжатия, можно ускорить решения за $O(n^4)$ и $O(n^3)$ в 32-64 раза, но это не требовалось в полном решении

Вопросы?

С годами я перестал покупать многие вещи просто потому, что они теперь кажутся мне нелепыми.

Стив Джобс

Задача 3 «Иннофон»

- Идея задачи — Павел Маврин
- Подготовка тестов — Павел Маврин, Григорий Резников
- Разбор задачи — Михаил Пядеркин

Постановка задачи

- Для каждого покупателя указана максимальная цена, за которую он готов приобрести «Иннофон», и максимальная цена, за которую он готов приобрести «Иннофон Плюс».
- Необходимо выбрать цены для «Иннофона» и «Иннофона Плюс» так, чтобы максимизировать суммарную прибыль.

Решение на 9 баллов ($n, a_i \leqslant 100$)

- Переберем цену «Иннофона Плюс» x, от 0 до 100
- Переберем цену «Иннофона» y, от 0 до 100
- Вычислим, сколько в таком случае заплатит каждый из покупателей, и сложим
- Покупатель с номером i заплатит
  - x, если $a_i \geqslant x$,
  - y, если $a_i < x$, но $b_i \geqslant y$,
  - 0 иначе.

Решение на 19 баллов ($n \leqslant 300$)

- Отметим, что не имеет смысла назначать цену z для любого из телефонов, если это число не совпадает ни с одним из a_i или b_i
- Составим список *осмысленных* цен: это все числа a_i и b_i
- Общее количество осмысленных цен равно $2n$

Решение на 19 баллов ($n \leqslant 300$)

- Переберем цену «Иннофона Плюс» x, это одна из осмысленных цен ($2n$ вариантов)
- Переберем цену «Иннофона» y, это также одна из осмысленных цен ($2n$ вариантов)
- Вычислим, сколько в таком случае заплатит каждый из покупателей, и сложим
- $\sim 2n \times 2n \times n = O(n^3)$ действий

Решение на 11 баллов ($b_i = 0$)

- Никто не готов покупать обычный «Иннофон»
- Необходимо подобрать цену на «Иннофон Плюс» так, чтобы максимизировать прибыль
- Отсортируем значения a_i **по убыванию**, пусть это список c
- Если мы назначим цену c_i, которая в отсортированном списке находится на позиции i, то наша прибыль будет равна $c_i \times (i + 1)$

$i+1$	1	2	3	4	5
c_i	27	25	20	10	5

Решение на 11 баллов ($b_i = 0$)

- Отсортируем все значения a_i
- В отсортированном списке c найдем максимум значений $c_i \times (i + 1)$
- $O(n \log_2 n) + O(n) = O(n \log_2 n)$

Решение на 35 баллов ($n \leqslant 3000$)

- Предположим, что цена на «Иннофон Плюс» зафиксирована и равна x
- Все люди, которые могут купить «Иннофон Плюс», его купят
- Составим список людей, которые не могут приобрести «Иннофон Плюс» (те, для которых $a_i < x$)
- Решим вспомогательную задачу: подобрать для этого списка людей цену на «Иннофон» так, чтобы максимизировать прибыль от оставшихся людей

Решение на 35 баллов ($n \leqslant 3000$)

- Вспомогательная задача: подобрать для списка людей цену на «Иннофон» так, чтобы максимизировать прибыль
- Это та самая подзадача на 11 баллов!
- Всех людей, которые не могут купить «Иннофон Плюс», необходимо отсортировать **по убыванию** значений b_i
- В полученном отсортированном списке c выбрать максимум из значений $c_i \times (i + 1)$

Решение на 35 баллов ($n \leqslant 3000$)

- Переберем цену на «Иннофон Плюс»
- Составим список людей, который не могут приобрести его по такой цене
- Упорядочим их по убыванию b_i
- В полученном отсортированном списке c выберем максимум $c_i \times (i + 1)$
- $O(n \times n \log_2 n) = O(n^2 \log_2 n)$

Решение на 35 баллов ($n \leqslant 3000$)

- Так как числа не меняются, можно изначально отсортировать всех людей по значению b_i, а на следующих шагах просто выписывать их в уже отсортированном порядке
- $O(n \log_2 n + n \times n) = O(n^2)$

Решение на 16 баллов ($a_i = b_i$)

- Отсортируем всех покупателей **по возрастанию** a_i, пусть это список c
- Тогда, если цена на «Иннофон Плюс» равна c_i, то наша прибыль от продажи «Иннофона Плюс» равна $c_i \times (n - i)$

$n - i$	5	4	3	2	1
c_i	5	10	20	25	27

Решение на 16 баллов ($a_i = b_i$)

- Покупатели, которые не смогут купить «Иннофон Плюс» по этой цене, стоят на позциях до i (невключительно) в отсортированном списке
- Для них необходимо выбрать оптимальное j такое, что $a_j \times (i - j)$ максимально

$i - j$	4	3	2	1			
c	5	10	20	25	27	29	30
					i		

Решение на 16 баллов ($a_i = b_i$)

- Будем перебирать цену «Иннофона Плюс» в порядке возрастания
- При переходе к следующему индексу один из элементов переходит в «левую часть»

$i - j$	4	3	2	1			
c	5	10	20	25	27	29	30
					i		

Решение на 16 баллов ($a_i = b_i$)

- Для «левой части» необходима структура данных, позволяющая выполнять две операции:
- Добавить элемент a_i в конец
- Найти максимум по всем j значений $(i - j) \times a_j$

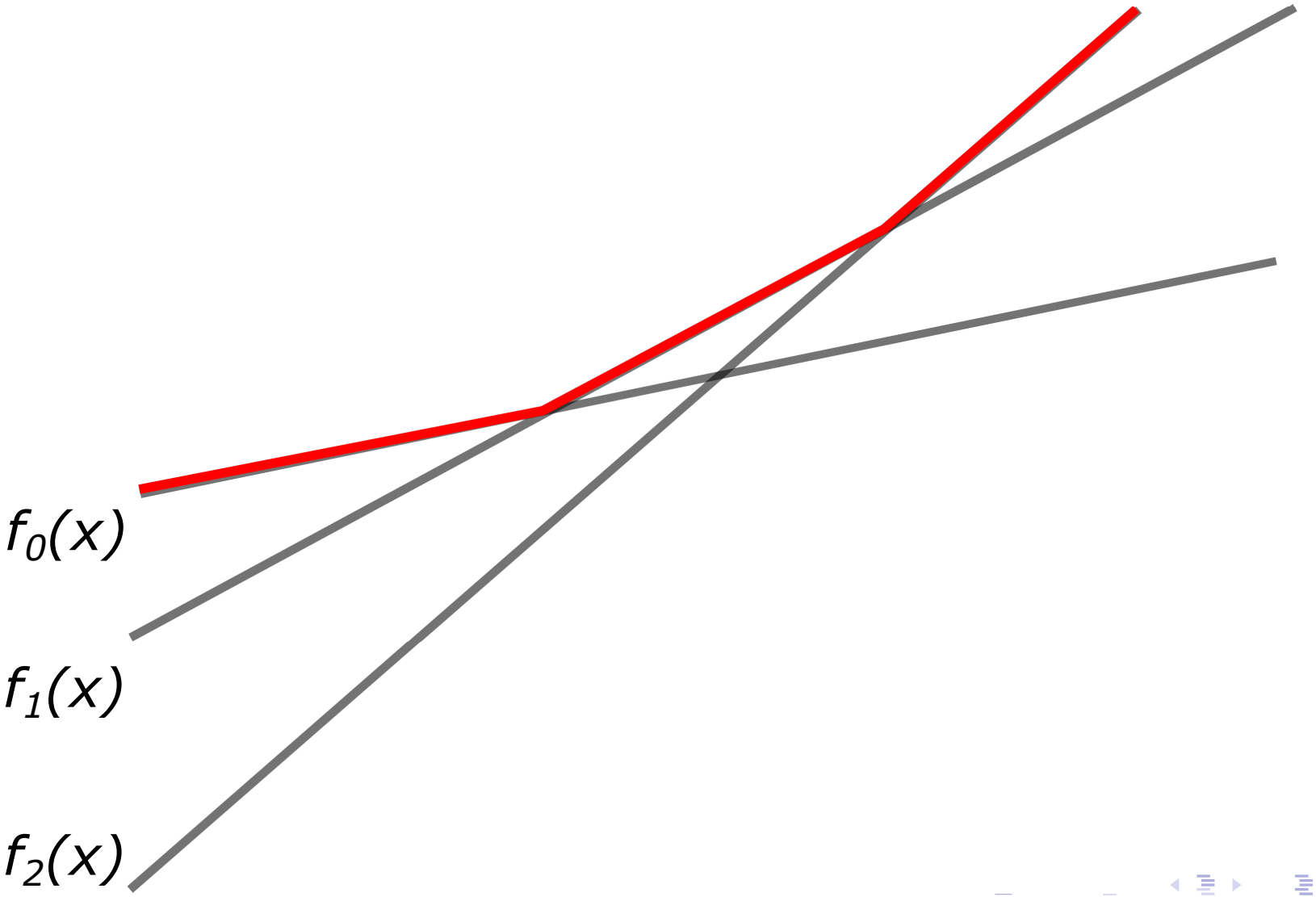
$i-j$	4	3	2	1			
c	5	10	20	25	27	29	30
					i		

Решение на 16 баллов ($a_i = b_i$)

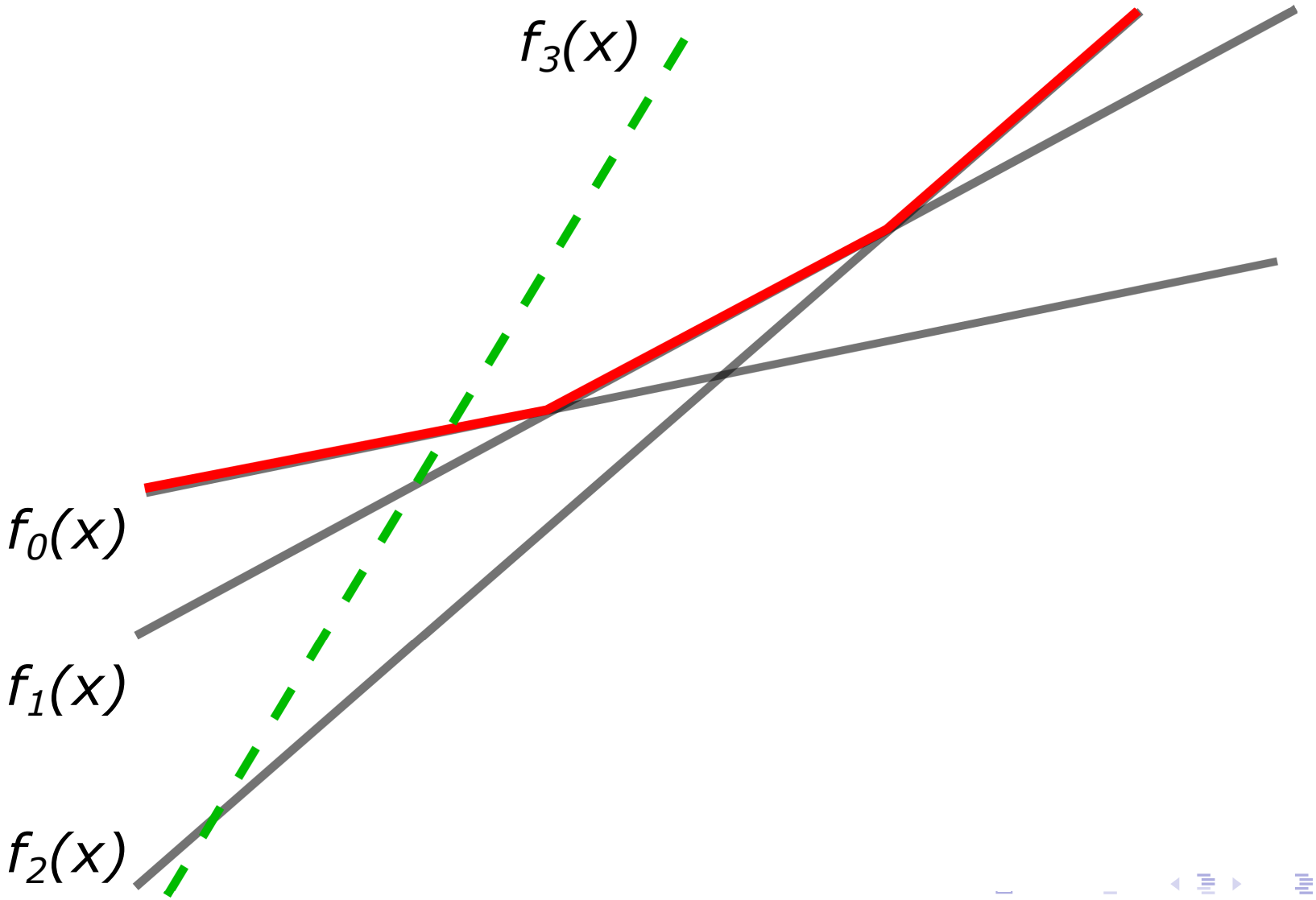
- Каждое j задает линейную функцию $f_j(x) = (x - j) \times a_j = a_j \cdot x - j \cdot a_j$
- Необходимо уметь добавлять одну новую функцию, а также находить максимум по всем j значений $f_j(i)$

$f_j(x)$	$5x$	$10x - 10$	$20x - 40$	$25x - 75$			
c	5	10	20	25	27	29	30
					i		

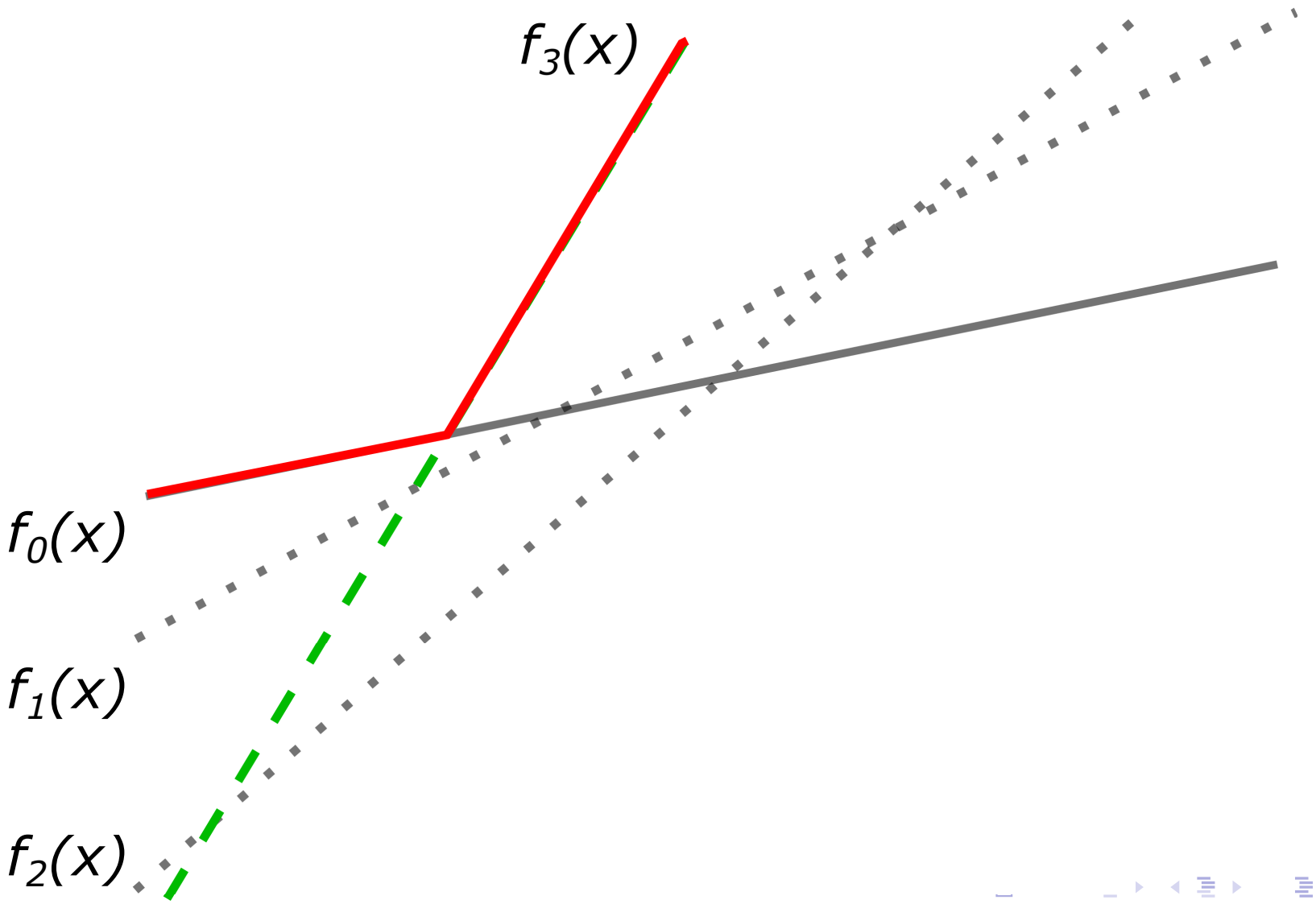
Решение на 16 баллов ($a_i = b_i$)



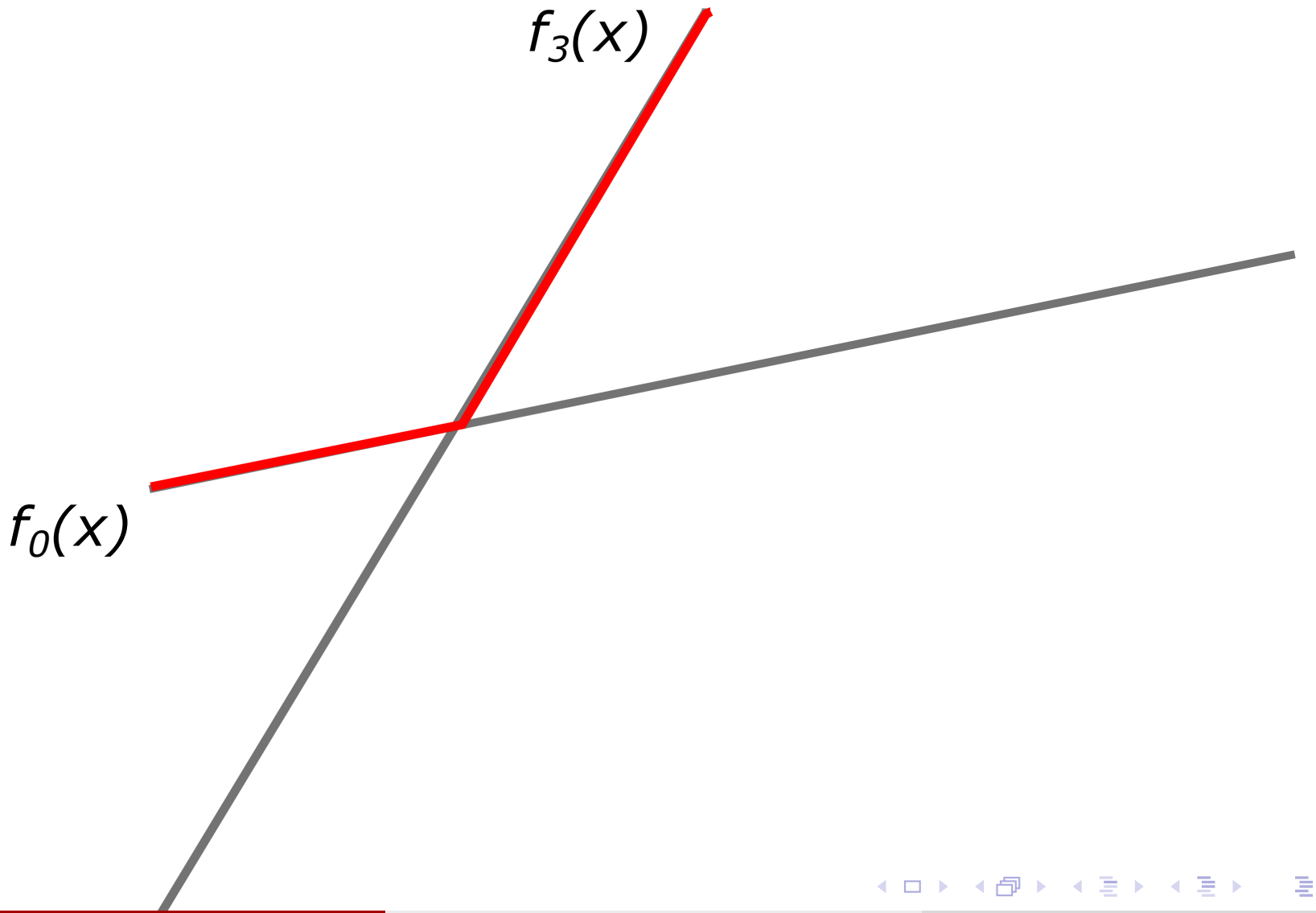
Решение на 16 баллов ($a_i = b_i$)



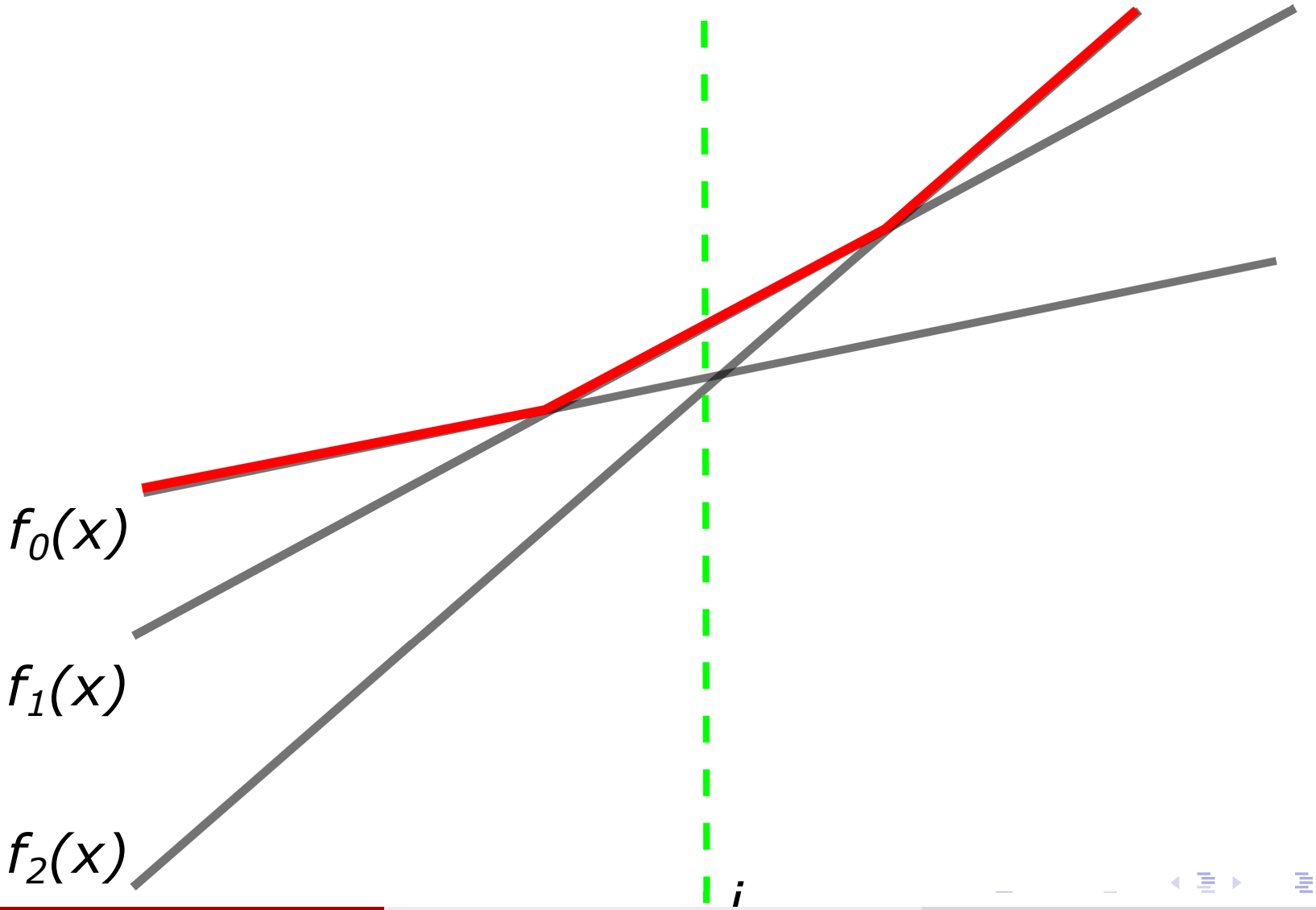
Решение на 16 баллов ($a_i = b_i$)



Решение на 16 баллов ($a_i = b_i$)



Решение на 16 баллов ($a_i = b_i$)



Решение на 16 баллов ($a_i = b_i$)

- Так как a_i возрастают, то прямые добавляются с конца
- Для поиска оптимального j можно воспользоваться бинарным поиском
- Но так как i возрастает, можно хранить указатель на оптимальное значение j, и он движется только вправо

Решение на 100 баллов

- Так же, как и в решении на 16 баллов, будем перебирать цену «Иннофона Плюс» в порядке возрастания
- Для «левой части» необходима структура данных, позволяющая выполнять две операции:
- Добавить элемент b_i (возможно, в середину отсортированного списка c)
- Найти максимум по всем j значений $j \times c_j$

Решение на 100 баллов

- Исходно разобьем все значения b_i, упорядоченные **по убыванию**, на группы размера $\sqrt{n}$
- Пусть какие-то элементы уже расположены в «левой части», назовем их *активными*

Решение на 100 баллов

- Внутри каждой группы каждый активный элемент x имеет свой индекс в упорядоченном по убывании массиве активных элементов этой группы, пусть это индекс t
- Для каждой группы мы знаем общее число уже добавленных элементов активных элементов в предыдущие группы, пусть это индекс g
- Тогда индекс элемента в упорядоченном массиве определяется как $t + g$

Решение на 100 баллов

- Нас интересует максимум по всем j величины $j \cdot c_j$
- $f_j(x) = (j + x)c_j = ((t + g) + x)c_j = (t + (x + g))c_j$
- Таким образом, внутри каждой группы мы можем хранить набор линейных функций $f'(x) = (t + x)c_j$, где важно, что индекс t определяется элементами внутри группы
- Это означает, что при добавлении элементов в какую-то другую группу линейные функции $f'(x)$ внутри нашей группы не меняются

Решение на 100 баллов

- При добавлении нового активного элемента ровно в одну группу требуется добавить новую линейную функцию
- К сожалению, эта линейная функция не добавляется в конец
- Поэтому, мы перестроим выпуклую оболочку с нуля, потратив на это $O(\sqrt{n})$ действий

Решение на 100 баллов

- Для вычисления оптимального ответа переберем группу, в которой он находится
- Внутри группы нам необходимо выбрать максимум линейных функций f' в точке g, где g — суммарное число активных элементов во всех предыдущих группах
- Таким образом, оптимальный ответ можно найти за количество групп, умноженное на время поиска ответа внутри одной группы

Решение на 100 баллов

- Число групп равно $O(\sqrt{n})$
- Внутри каждой группы максимум можно находить бинарным поиском за $O(\log_2 n)$
- Сложность такого решения составляет $\sim n \times \sqrt{n} \times \log_2 n = O(n\sqrt{n}\log_2 n)$

Решение на 100 баллов

- Так как для каждой группы число активных элементов только возрастает, то внутри каждой группы можно также поддерживать указатель на оптимальную функцию
- При добавлении новой функции, во всех группах, кроме одной, указатель может сдвинуться лишь вправо
- Внутри этой группы указатель же может сдвинуться на первую из них
- Суммарное число движений указателей составляет $O(n\sqrt{n})$

Вопросы?

Задача 4 «Квантовая телепортация»

- Идея задачи — Георгий Корнеев, Глеб Евстропов
- Подготовка тестов — Борис Минаев, Павел Кунявский
- Разбор задачи — Глеб Евстропов

Постановка задачи

- Дано клетчатое поле $n \times m$, в нём выбрано k клеток.
- Расстояние между двумя выбранными клетками (x_1, y_1) и (x_2, y_2) определяется как $2^{d_{i,j}}$, где $d_{i,j} = \max(|x_1 - x_2|, |y_1 - y_2|)$.
- Найти кратчайший путь из $(1, 1)$ в (n, m).

Постановка задачи

- Дано клетчатое поле $n \times m$, в нём выбрано k клеток.
- Расстояние между двумя выбранными клетками (x_1, y_1) и (x_2, y_2) определяется как $2^{d_{i,j}}$, где $d_{i,j} = \max(|x_1 - x_2|, |y_1 - y_2|)$.
- Найти кратчайший путь из $(1, 1)$ в (n, m).

Постановка задачи

- Дано клетчатое поле $n \times m$, в нём выбрано k клеток.
- Расстояние между двумя выбранными клетками (x_1, y_1) и (x_2, y_2) определяется как $2^{d_{i,j}}$, где $d_{i,j} = \max(|x_1 - x_2|, |y_1 - y_2|)$.
- Найти кратчайший путь из $(1, 1)$ в (n, m).

Решение на 21 балл

- Поскольку $n, m \leq 20$, веса всех рёбер графа не превосходят 2^{20}.
- Рёбер $O(k^2)$.
- Воспользуемся любым алгоритмом поиска кратчайшего пути.

Решение на 21 балл

- Поскольку $n, m \leq 20$, веса всех рёбер графа не превосходят 2^{20}.
- Рёбер $O(k^2)$.
- Воспользуемся любым алгоритмом поиска кратчайшего пути.

Решение на 21 балл

- Поскольку $n, m \leq 20$, веса всех рёбер графа не превосходят 2^{20}.
- Рёбер $O(k^2)$.
- Воспользуемся любым алгоритмом поиска кратчайшего пути.

Решение на 34 балла

- При ограничениях $n, m \leq 500$, ответ не превосходит 2^{500}.
- Реализуем длинную арифметику. Нам потребуются операции прибавления степени двойки и сравнения.
- Будем отдельно хранить каждый бит числа, выполнять сравнение за $O(len)$.
- Алгоритмом Дейкстры, время работы составим $O(k^2 \cdot len)$.
- $len \leq \max(n, m)$.

Решение на 34 балла

- При ограничениях $n, m \leq 500$, ответ не превосходит 2^{500}.
- Реализуем длинную арифметику. Нам потребуются операции прибавления степени двойки и сравнения.
- Будем отдельно хранить каждый бит числа, выполнять сравнение за $O(len)$.
- Алгоритмом Дейкстры, время работы составим $O(k^2 \cdot len)$.
- $len \leq \max(n, m)$.

Решение на 34 балла

- При ограничениях $n, m \leq 500$, ответ не превосходит 2^{500}.
- Реализуем длинную арифметику. Нам потребуются операции прибавления степени двойки и сравнения.
- Будем отдельно хранить каждый бит числа, выполнять сравнение за $O(len)$.
- Алгоритмом Дейкстры, время работы составим $O(k^2 \cdot len)$.
- $len \leq \max(n, m)$.

Решение на 34 балла

- При ограничениях $n, m \leq 500$, ответ не превосходит 2^{500}.
- Реализуем длинную арифметику. Нам потребуются операции прибавления степени двойки и сравнения.
- Будем отдельно хранить каждый бит числа, выполнять сравнение за $O(len)$.
- Алгоритмом Дейкстры, время работы составим $O(k^2 \cdot len)$.
- $len \leq \max(n, m)$.

Решение на 34 балла

- При ограничениях $n, m \leq 500$, ответ не превосходит 2^{500}.
- Реализуем длинную арифметику. Нам потребуются операции прибавления степени двойки и сравнения.
- Будем отдельно хранить каждый бит числа, выполнять сравнение за $O(len)$.
- Алгоритмом Дейкстры, время работы составим $O(k^2 \cdot len)$.
- $len \leq \max(n, m)$.

Длинная арифметика

- База длинной арифметики w — беззнаковые 32-битные числа.
- Ответ не более $10\,000$. Итого примерно 300 чисел в длинной арифметике.
- Можно хранить только позиции единичных бит.
- Различных прыжков больше x не больше $\frac{n \cdot m}{x^2}$.
- $O(n^{\frac{2}{3}})$ единичных бит $(n \approx m)$.
- Работа с числами в такой форме за линейное время.

Длинная арифметика

- База длинной арифметики w — беззнаковые 32-битные числа.
- Ответ не более $10\,000$. Итого примерно 300 чисел в длинной арифметике.
- Можно хранить только позиции единичных бит.
- Различных прыжков больше x не больше $\frac{n \cdot m}{x^2}$.
- $O(n^{\frac{2}{3}})$ единичных бит $(n \approx m)$.
- Работа с числами в такой форме за линейное время.

Длинная арифметика

- База длинной арифметики w — беззнаковые 32-битные числа.
- Ответ не более $10\,000$. Итого примерно 300 чисел в длинной арифметике.
- Можно хранить только позиции единичных бит.
- Различных прыжков больше x не больше $\frac{n \cdot m}{x^2}$.
- $O(n^{\frac{2}{3}})$ единичных бит $(n \approx m)$.
- Работа с числами в такой форме за линейное время.

Длинная арифметика

- База длинной арифметики w — беззнаковые 32-битные числа.
- Ответ не более $10\,000$. Итого примерно 300 чисел в длинной арифметике.
- Можно хранить только позиции единичных бит.
- Различных прыжков больше x не больше $\frac{n \cdot m}{x^2}$.
- $O(n^{\frac{2}{3}})$ единичных бит $(n \approx m)$.
- Работа с числами в такой форме за линейное время.

Длинная арифметика

- База длинной арифметики w — беззнаковые 32-битные числа.
- Ответ не более $10\,000$. Итого примерно 300 чисел в длинной арифметике.
- Можно хранить только позиции единичных бит.
- Различных прыжков больше x не больше $\frac{n \cdot m}{x^2}$.
- $O(n^{\frac{2}{3}})$ единичных бит $(n \approx m)$.
- Работа с числами в такой форме за линейное время.

Длинная арифметика

- База длинной арифметики w — беззнаковые 32-битные числа.
- Ответ не более $10\,000$. Итого примерно 300 чисел в длинной арифметике.
- Можно хранить только позиции единичных бит.
- Различных прыжков больше x не больше $\frac{n \cdot m}{x^2}$.
- $O(n^{\frac{2}{3}})$ единичных бит $(n \approx m)$.
- Работа с числами в такой форме за линейное время.

Решение на 67 баллов

- Рассмотрим переходы из позиции (i, j) в область $(x > i, y > j)$.
- Пусть d — расстояние до ближайшей точки (a, b) в этой области.
- Все точки с таким расстоянием находятся на отрезках $(i + 1 \leq x \leq i + d, y = j + d)$ и $(x + d, j + 1 \leq y \leq j + d)$.
- Пусть есть точка (x, y), причём $x > i$, $y > j$ и $c = \max(x - i, y - j) > d$.
- Тогда $2^c > 2^d + 2^{\max(|x-a|,|y-b|)}$.

Решение на 67 баллов

- Рассмотрим переходы из позиции (i, j) в область $(x > i, y > j)$.
- Пусть d — расстояние до ближайшей точки (a, b) в этой области.
- Все точки с таким расстоянием находятся на отрезках $(i + 1 \leq x \leq i + d, y = j + d)$ и $(x + d, j + 1 \leq y \leq j + d)$.
- Пусть есть точка (x, y), причём $x > i$, $y > j$ и $c = \max(x - i, y - j) > d$.
- Тогда $2^c > 2^d + 2^{\max(|x-a|,|y-b|)}$.

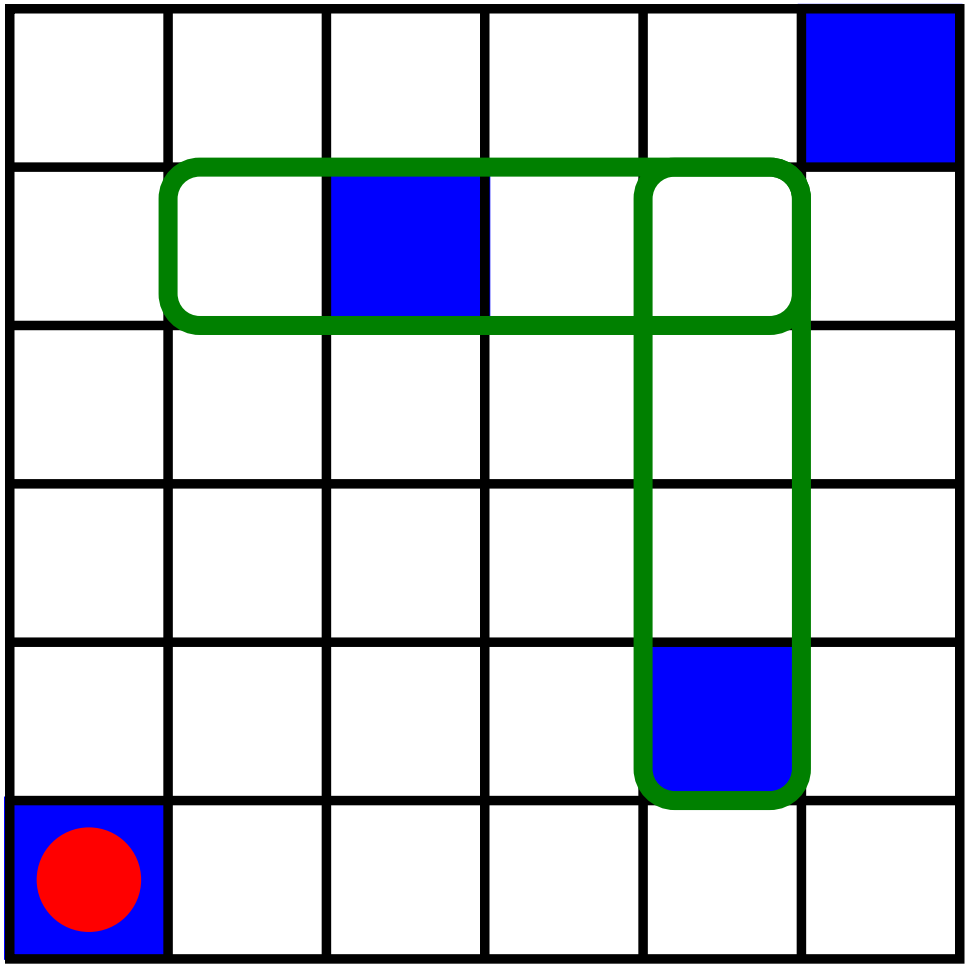
Решение на 67 баллов

- Рассмотрим переходы из позиции (i, j) в область $(x > i, y > j)$.
- Пусть d — расстояние до ближайшей точки (a, b) в этой области.
- Все точки с таким расстоянием находятся на отрезках $(i + 1 \leq x \leq i + d, y = j + d)$ и $(x + d, j + 1 \leq y \leq j + d)$.
- Пусть есть точка (x, y), причём $x > i$, $y > j$ и $c = \max(x - i, y - j) > d$.
- Тогда $2^c > 2^d + 2^{\max(|x-a|,|y-b|)}$.

Решение на 67 баллов

- Рассмотрим переходы из позиции (i, j) в область $(x > i, y > j)$.
- Пусть d — расстояние до ближайшей точки (a, b) в этой области.
- Все точки с таким расстоянием находятся на отрезках $(i + 1 \leq x \leq i + d, y = j + d)$ и $(x + d, j + 1 \leq y \leq j + d)$.
- Пусть есть точка (x, y), причём $x > i$, $y > j$ и $c = \max(x - i, y - j) > d$.
- Тогда $2^c > 2^d + 2^{\max(|x-a|,|y-b|)}$.

Решение на 67 баллов



Решение на 67 баллов

- Точка (a, b) позволяет сократить путь в любую другую точку области дальше d.
- Проведём рёбра во всех точки на расстоянии d.
- В третьей группе не бывает двух точек на одной вертикали или одной горизонтали.
- Не более двух рёбер в одну область.
- Не более восьми рёбер из одной вершины.

Решение на 67 баллов

- Точка (a, b) позволяет сократить путь в любую другую точку области дальше d.
- Проведём рёбра во всех точки на расстоянии d.
- В третьей группе не бывает двух точек на одной вертикали или одной горизонтали.
- Не более двух рёбер в одну область.
- Не более восьми рёбер из одной вершины.

Решение на 67 баллов

- Точка (a, b) позволяет сократить путь в любую другую точку области дальше d.
- Проведём рёбра во всех точки на расстоянии d.
- В третьей группе не бывает двух точек на одной вертикали или одной горизонтали.
- Не более двух рёбер в одну область.
- Не более восьми рёбер из одной вершины.

Решение на 67 баллов

- Точка (a, b) позволяет сократить путь в любую другую точку области дальше d.
- Проведём рёбра во всех точки на расстоянии d.
- В третьей группе не бывает двух точек на одной вертикали или одной горизонтали.
- Не более двух рёбер в одну область.
- Не более восьми рёбер из одной вершины.

Решение на 67 баллов

- Точка (a, b) позволяет сократить путь в любую другую точку области дальше d.
- Проведём рёбра во всех точки на расстоянии d.
- В третьей группе не бывает двух точек на одной вертикали или одной горизонтали.
- Не более двух рёбер в одну область.
- Не более восьми рёбер из одной вершины.

Решение на 67 баллов

- Время работы алгоритма Дейкстры составит $O(n \log n \cdot \frac{n}{w})$.
- Как построить граф?
- Возьмём пару точек i и j, d равно расстоянию между ними.
- Не существует способа «срезать», если нет точек на расстоянии $d - 1$ от обеих.
- С помощью структур данных, можно построить граф за $O(k \log k)$.

Решение на 67 баллов

- Время работы алгоритма Дейкстры составит $O(n \log n \cdot \frac{n}{w})$.
- Как построить граф?
- Возьмём пару точек i и j, d равно расстоянию между ними.
- Не существует способа «срезать», если нет точек на расстоянии $d - 1$ от обеих.
- С помощью структур данных, можно построить граф за $O(k \log k)$.

Решение на 67 баллов

- Время работы алгоритма Дейкстры составит $O(n \log n \cdot \frac{n}{w})$.
- Как построить граф?
- Возьмём пару точек i и j, d равно расстоянию между ними.
- Не существует способа «срезать», если нет точек на расстоянии $d-1$ от обеих.
- С помощью структур данных, можно построить граф за $O(k \log k)$.

Решение на 67 баллов

- Время работы алгоритма Дейкстры составит $O(n \log n \cdot \frac{n}{w})$.
- Как построить граф?
- Возьмём пару точек i и j, d равно расстоянию между ними.
- Не существует способа «срезать», если нет точек на расстоянии $d - 1$ от обеих.
- С помощью структур данных, можно построить граф за $O(k \log k)$.

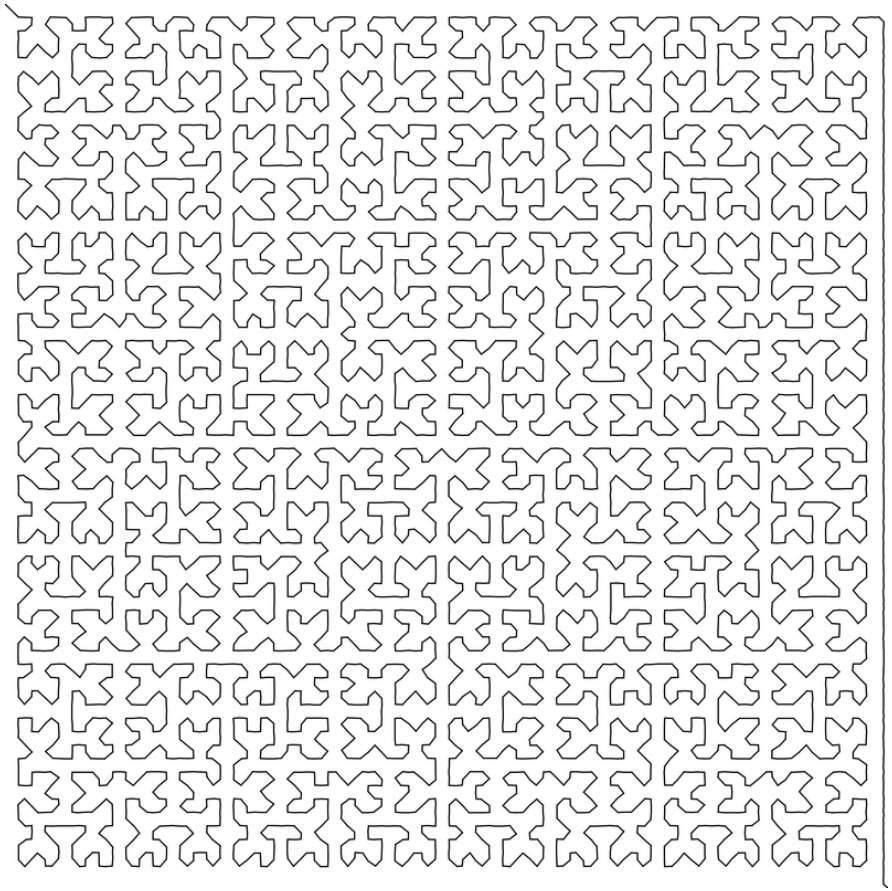
Решение на 67 баллов

- Время работы алгоритма Дейкстры составит $O(n \log n \cdot \frac{n}{w})$.
- Как построить граф?
- Возьмём пару точек i и j, d равно расстоянию между ними.
- Не существует способа «срезать», если нет точек на расстоянии $d - 1$ от обеих.
- С помощью структур данных, можно построить граф за $O(k \log k)$.

Решение на 100 баллов

- Множество вершин, расстояние до которых следует релаксировать лежит на двух отрезках.
- Построим ДО для каждой вертикали и каждой горизонтали.
- В ДО хранится оптимальное значение, по лучшим элементам каждого ДО построим кучу.
- При извлечении минимума, удаляем вершину из всех структур.
- При релаксации применяем обновление на отрезке.
- Итоговое время работы $O(n \log n \cdot \frac{n}{w})$.

Как может выглядеть ответ?



Вопросы?

Задача 5 «Расшифровка»

- Идея задачи — Дмитрий Саютин
- Подготовка тестов — Дмитрий Саютин, Дмитрий Иващенко
- Разбор задачи — Андрей Календаров

Постановка задачи

- Дан массив натуральных чисел.
- Разбить его на минимальное количество непересекающихся отрезков, так чтобы в каждом отрезке минимум находился на левой границе, а максимум – на правой.

ДП (30 баллов)

- Посчитаем dp_i – ответ на задачу для префикса массива длины i.
- Пересчет: переберем левую границу последнего отрезка, проверим выполнение условий.
- $dp_i = dp_{j-1} + 1$, для таких $j < i$, что $a_j = \min(a[j:i])$ и $a_i = \max(a[j:i])$.
- $O(n^2)$ переходов, в каждом поиск минимума и максимума на отрезке за $O(n)$.

ДП (30 баллов)

- Посчитаем dp_i – ответ на задачу для префикса массива длины i.
- Пересчет: переберем левую границу последнего отрезка, проверим выполнение условий.
- $dp_i = dp_{j-1} + 1$, для таких $j < i$, что $a_j = \min(a[j:i])$ и $a_i = \max(a[j:i])$.
- $O(n^2)$ переходов, в каждом поиск минимума и максимума на отрезке за $O(n)$.

ДП (30 баллов)

- Посчитаем dp_i – ответ на задачу для префикса массива длины i.
- Пересчет: переберем левую границу последнего отрезка, проверим выполнение условий.
- $dp_i = dp_{j-1} + 1$, для таких $j < i$, что $a_j = \min(a[j:i])$ и $a_i = \max(a[j:i])$.
- $O(n^2)$ переходов, в каждом поиск минимума и максимума на отрезке за $O(n)$.

ДП (30 баллов)

- Посчитаем dp_i – ответ на задачу для префикса массива длины i.
- Пересчет: переберем левую границу последнего отрезка, проверим выполнение условий.
- $dp_i = dp_{j-1} + 1$, для таких $j < i$, что $a_j = \min(a[j:i])$ и $a_i = \max(a[j:i])$.
- $O(n^2)$ переходов, в каждом поиск минимума и максимума на отрезке за $O(n)$.

ДП (60 баллов)

- Оптимизация: будем находить минимум на отрезке с помощью дерева отрезков или разреженных таблиц.
- Другая оптимизация: будем перебирать j в порядке $i - 1 \,..\, 0$, поддерживая минимум и максимум на отрезке.
- $O(n^2)$ переходов, в каждом $O(\log n)$ или $O(1)$ действий.

ДП (60 баллов)

- Оптимизация: будем находить минимум на отрезке с помощью дерева отрезков или разреженных таблиц.
- Другая оптимизация: будем перебирать j в порядке $i - 1 \,..\, 0$, поддерживая минимум и максимум на отрезке.
- $O(n^2)$ переходов, в каждом $O(\log n)$ или $O(1)$ действий.

ДП (60 баллов)

- Оптимизация: будем находить минимум на отрезке с помощью дерева отрезков или разреженных таблиц.
- Другая оптимизация: будем перебирать j в порядке $i - 1 \,..\, 0$, поддерживая минимум и максимум на отрезке.
- $O(n^2)$ переходов, в каждом $O(\log n)$ или $O(1)$ действий.

Жадные идеи

- Исследуем первый отрезок $a[0 : r]$.
- Для каждого i рассмотрим индекс mn_i ближайшего справа элемента, меньшего a_i.
- Индекс mn_0 является левой границей одного из отрезков.
- $\Longrightarrow r < mn_0$.

Жадные идеи

- Исследуем первый отрезок $a[0 : r]$.
- Для каждого i рассмотрим индекс mn_i ближайшего справа элемента, меньшего a_i.
- Индекс mn_0 является левой границей одного из отрезков.
- $\Longrightarrow r < mn_0$.

Жадные идеи

- Исследуем первый отрезок $a[0 : r]$.
- Для каждого i рассмотрим индекс mn_i ближайшего справа элемента, меньшего a_i.
- Индекс mn_0 является левой границей одного из отрезков.
- $\Longrightarrow r < mn_0$.

Жадные идеи

- Исследуем первый отрезок $a[0 : r]$.
- Для каждого i рассмотрим индекс mn_i ближайшего справа элемента, меньшего a_i.
- Индекс mn_0 является левой границей одного из отрезков.
- $\Longrightarrow r < mn_0$.

Жадные идеи (продолжение)

- Рассмотрим индекс p максимального элемента на отрезке $a[0 : mn_0 - 1]$. Если таких несколько, выберем самый правый.
- Индекс p является правой границей одного из отрезков.
- $\Longrightarrow r \leqslant p$.
- Заметим, что $a_0 = \min(a[0 : p])$, $a_p = \max(a[0 : p])$.
- $\Longrightarrow r = p$ оптимально.

Жадные идеи (продолжение)

- Рассмотрим индекс p максимального элемента на отрезке $a[0 : mn_0 - 1]$. Если таких несколько, выберем самый правый.
- Индекс p является правой границей одного из отрезков.
- $\Longrightarrow r \leqslant p$.
- Заметим, что $a_0 = \min(a[0 : p])$, $a_p = \max(a[0 : p])$.
- $\Longrightarrow r = p$ оптимально.

Жадные идеи (продолжение)

- Рассмотрим индекс p максимального элемента на отрезке $a[0 : mn_0 - 1]$. Если таких несколько, выберем самый правый.
- Индекс p является правой границей одного из отрезков.
- $\Longrightarrow r \leqslant p$.
- Заметим, что $a_0 = \min(a[0 : p])$, $a_p = \max(a[0 : p])$.
- $\Longrightarrow r = p$ оптимально.

Жадные идеи (продолжение)

- Рассмотрим индекс p максимального элемента на отрезке $a[0 : mn_0 - 1]$. Если таких несколько, выберем самый правый.
- Индекс p является правой границей одного из отрезков.
- $\Longrightarrow r \leqslant p$.
- Заметим, что $a_0 = \min(a[0 : p])$, $a_p = \max(a[0 : p])$.
- $\Longrightarrow r = p$ оптимально.

Жадные идеи (продолжение)

- Рассмотрим индекс p максимального элемента на отрезке $a[0 : mn_0 - 1]$. Если таких несколько, выберем самый правый.
- Индекс p является правой границей одного из отрезков.
- $\Longrightarrow r \leqslant p$.
- Заметим, что $a_0 = \min(a[0 : p])$, $a_p = \max(a[0 : p])$.
- $\Longrightarrow r = p$ оптимально.

Жадный алгоритм (60 баллов)

- Будем находить индексы mn_i и p линейным поиском.
- Выделив первый отрезок, повторим процесс, пока не рассмотрим весь массив.
- $O(n)$ итераций, $O(n)$ действий в каждой.

Жадный алгоритм (60 баллов)

- Будем находить индексы mn_i и p линейным поиском.
- Выделив первый отрезок, повторим процесс, пока не рассмотрим весь массив.
- $O(n)$ итераций, $O(n)$ действий в каждой.

Жадный алгоритм (60 баллов)

- Будем находить индексы mn_i и p линейным поиском.
- Выделив первый отрезок, повторим процесс, пока не рассмотрим весь массив.
- $O(n)$ итераций, $O(n)$ действий в каждой.

Жадный алгоритм (100 баллов)

- Индексы mn_i можно предподсчитать с помощью стека за $O(n)$.
- Также для каждого i предподсчитаем индекс mx_i самого левого элемента, **не меньшего** a_i.
- Теперь для левой границы отрезка l индекс p можно найти за $O(1)$ амортизировано:
  пока $mx_p < mn_l$: $p \leftarrow mx_p$.
- $O(n)$.

Жадный алгоритм (100 баллов)

- Индексы mn_i можно предподсчитать с помощью стека за $O(n)$.
- Также для каждого i предподсчитаем индекс mx_i самого левого элемента, **не меньшего** a_i.
- Теперь для левой границы отрезка l индекс p можно найти за $O(1)$ амортизировано:
  пока $mx_p < mn_l$: $p \leftarrow mx_p$.
- $O(n)$.

Жадный алгоритм (100 баллов)

- Индексы mn_i можно предподсчитать с помощью стека за $O(n)$.
- Также для каждого i предподсчитаем индекс mx_i самого левого элемента, **не меньшего** a_i.
- Теперь для левой границы отрезка l индекс p можно найти за $O(1)$ амортизировано:
  пока $mx_p < mn_l$: $p \leftarrow mx_p$.
- $O(n)$.

Жадный алгоритм (100 баллов)

- Индексы mn_i можно предподсчитать с помощью стека за $O(n)$.
- Также для каждого i предподсчитаем индекс mx_i самого левого элемента, **не меньшего** a_i.
- Теперь для левой границы отрезка l индекс p можно найти за $O(1)$ амортизировано:
  пока $mx_p < mn_l$: $p \leftarrow mx_p$.
- $O(n)$.

Другой жадный алгоритм (60 баллов)

- Рассмотрим индекс r максимального в массиве a элемента. Если таких несколько, выберем самый правый.
- Рассмотрим индекс l минимального на отрезке $a[0:r]$ элемента. Если таких несколько, выберем самый левый.
- Взять отрезок $a[l:r]$ оптимально.
- Решим ту же задачу независимо в массивах $a[0:l-1]$ и $a[r+1:n-1]$.
- $O(n)$ отрезков, обработка каждого за $O(n)$.

Другой жадный алгоритм (60 баллов)

- Рассмотрим индекс r максимального в массиве a элемента. Если таких несколько, выберем самый правый.
- Рассмотрим индекс l минимального на отрезке $a[0:r]$ элемента. Если таких несколько, выберем самый левый.
- Взять отрезок $a[l:r]$ оптимально.
- Решим ту же задачу независимо в массивах $a[0:l-1]$ и $a[r+1:n-1]$.
- $O(n)$ отрезков, обработка каждого за $O(n)$.

Другой жадный алгоритм (60 баллов)

- Рассмотрим индекс r максимального в массиве a элемента. Если таких несколько, выберем самый правый.
- Рассмотрим индекс l минимального на отрезке $a[0:r]$ элемента. Если таких несколько, выберем самый левый.
- Взять отрезок $a[l:r]$ оптимально.
- Решим ту же задачу независимо в массивах $a[0:l-1]$ и $a[r+1:n-1]$.
- $O(n)$ отрезков, обработка каждого за $O(n)$.

Другой жадный алгоритм (60 баллов)

- Рассмотрим индекс r максимального в массиве a элемента. Если таких несколько, выберем самый правый.
- Рассмотрим индекс l минимального на отрезке $a[0:r]$ элемента. Если таких несколько, выберем самый левый.
- Взять отрезок $a[l:r]$ оптимально.
- Решим ту же задачу независимо в массивах $a[0:l-1]$ и $a[r+1:n-1]$.
- $O(n)$ отрезков, обработка каждого за $O(n)$.

Другой жадный алгоритм (60 баллов)

- Рассмотрим индекс r максимального в массиве a элемента. Если таких несколько, выберем самый правый.
- Рассмотрим индекс l минимального на отрезке $a[0:r]$ элемента. Если таких несколько, выберем самый левый.
- Взять отрезок $a[l:r]$ оптимально.
- Решим ту же задачу независимо в массивах $a[0:l-1]$ и $a[r+1:n-1]$.
- $O(n)$ отрезков, обработка каждого за $O(n)$.

Другой жадный алгоритм (100 баллов)

- Оптимизация: будем находить минимум на отрезке с помощью дерева отрезков или разреженных таблиц.
- $O(n)$ отрезков, обработка каждого за $O(\log n)$ или $O(1)$.

Другой жадный алгоритм (100 баллов)

- Оптимизация: будем находить минимум на отрезке с помощью дерева отрезков или разреженных таблиц.
- $O(n)$ отрезков, обработка каждого за $O(\log n)$ или $O(1)$.

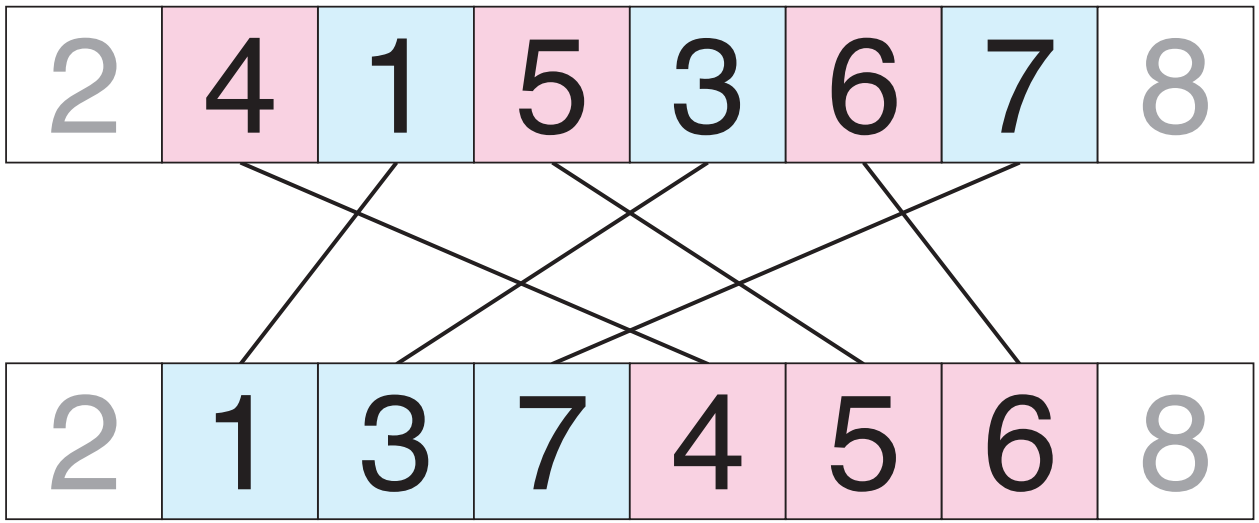
Вопросы?

Задача 6 «Быстрая сортировка»

- Идея задачи — Жюри
- Подготовка тестов — Павел Кунявский, Павел Маврин
- Разбор задачи — Павел Маврин

Постановка задачи

- Отсортировать массив с помощью операции «расслоения» отрезка массива

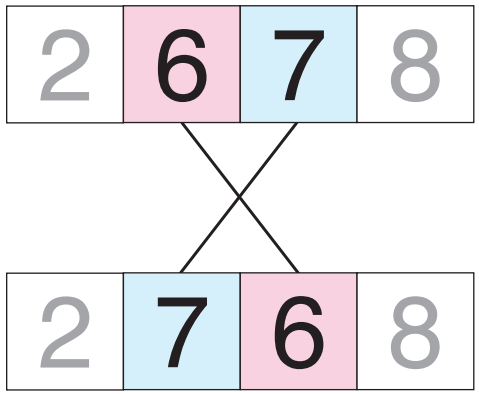


Решение на 20 баллов

- Гарантируется, что массив можно отсортировать с помощью одной операции расслоения
- Переберем все возможные операции и проверим, сортируют ли они массив
- Время работы $O(n^3)$

Решение на 50 баллов

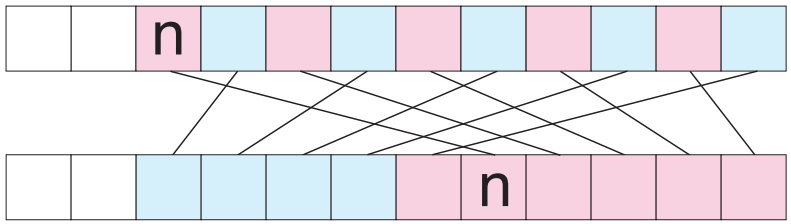
- Операция расслоения на отрезке размера 2 — это просто обмен соседних элементов



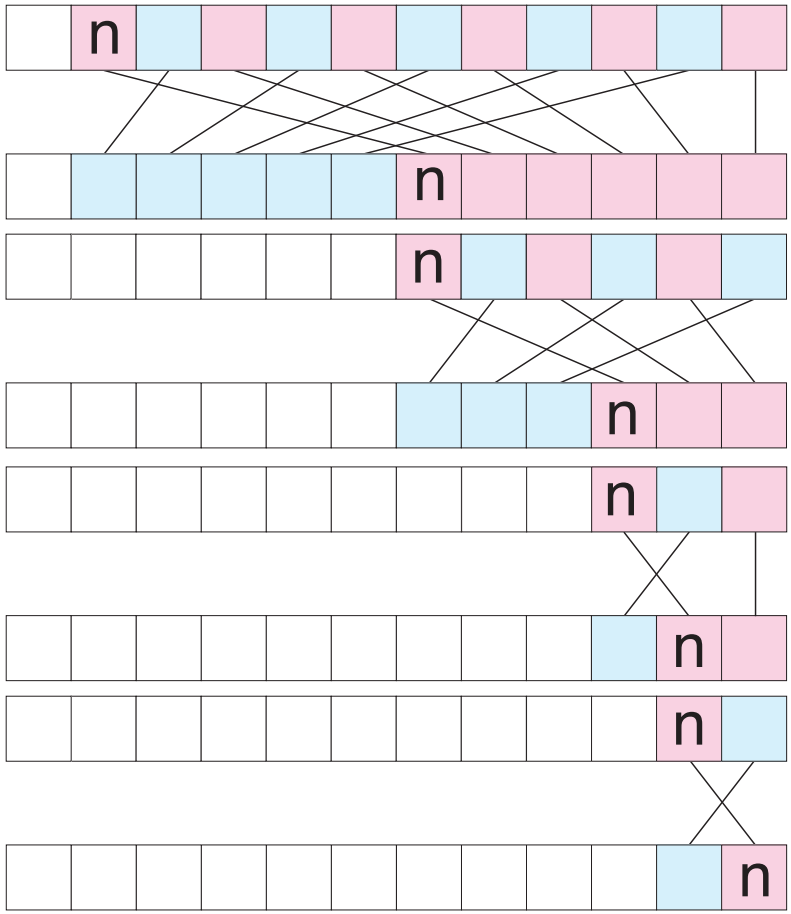
- Применим сортировку вставками или пузырьком
- Число операций $O(n^2)$

Решение на 80 баллов

- Научимся выставлять число n на свое место
- Пусть $p(n)$ — текущая позиция числа n
- Выберем отрезок $[p(n), n]$. Число n сдвинется максимально вправо



Решение на 80 баллов

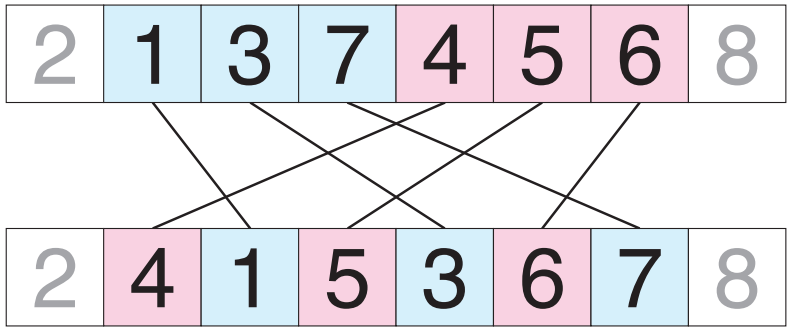


Решение на 80 баллов

- После каждой операции $n - p(n)$ уменьшается в два раза
- За $\log n$ операций число n встанет на свое место
- Общее число операций $O(n \log n)$

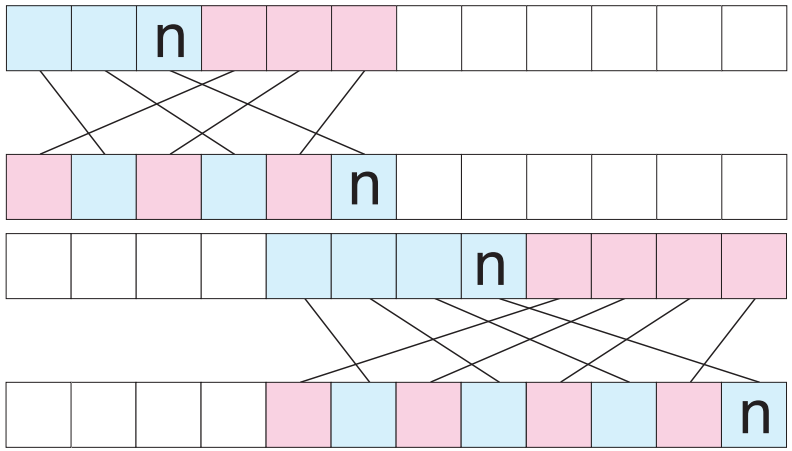
Решение на 100 баллов

- Будем строить решение с конца
- Построим обратную перестановку
- Теперь нужно отсортировать массив с помощью операции, обратной расслоению



Решение на 100 баллов

- Научимся выставлять число n на свое место
- Выберем отрезок так, чтобы число n сдвинулось максимально вправо.



Решение на 100 баллов

- Чем это решение лучше?
- Посмотрим, сколько действий нужно, чтобы поставить на место n, в зависимости от начальной позиции
- Старое решение:

4	4	4	4	4	4	4	4	3	3	3	3	2	2	1	0

- Новое решение:

4	3	3	2	2	2	2	1	1	1	1	1	1	1	1	0

Решение на 100 баллов

4	3	3	2	2	2	2	1	1	1	1	1	1	1	1	0

- Пусть массив изначально упорядочен случайно
- Среднее число операций $1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + \ldots = 2$
- Таким образом, общее число операций в среднем $2n$

Решение на 100 баллов

- Как случайно перемешать элементы массива?
- Способ 1: сделать 239 случайных операций
- Способ 2: сгенерировать случайную перестановку и переставить элементы в соответствии с ней тем же способом за $2n$ операций в среднем
- Общее число операций в среднем $4n$, с большой вероятностью не больше $5n$

Вопросы?

Задача 7 «Робомарафон»

- Идея задачи — Глеб Евстропов
- Подготовка тестов — Николай Будин, Глеб Евстропов
- Разбор задачи — Михаил Пядеркин

Постановка задачи

- Дано n бегунов. Каждый умеет преодолевать дистанцию за a_i.
- Сигнальные стартовые устройства можно включить у каких-то роботов, как минимум у одного
- Каждый робот стартует, когда до него доходит сигнал, распространяющийся со скоростью одна дорожка в секунду
- Определить лучшее и худшее возможное место для каждого робота

Решение на 20 баллов, за $O(2^n \cdot n)$

- Переберем все 2^n способов поставить сигнальные устройства
- Для каждого способа найдем место каждого бегуна
- Выберем максимум и минимум для каждого бегуна

$p = 1$ (минимальное место)

- Заметим, что спортсмен займет минимальное место в том случае, если включено только его сигнальное устройство
- Спортсмен j обгонит спортсмена i, если $a_j + |i - j| < a_i$
- За $O(n^2)$ тривиально (≈ 20 баллов)

$p = 1$, $O(n \log n)$ (до 40 баллов)

- Рассмотрим отдельно случаи $j > i$ и $j < i$
- Пусть $j < i$, второй случай аналогично
- Нужно для i найти число j, что $a_j + i - j < a_i$
- Перенесем слагаемые: $(a_j - j) < (a_i - i)$
- Сжимаем разности $(a_j - j)$, строим сортированную структуру данных на разностях $(a_i - i)$
- Добавляем элементы по увеличению i, считаем число элементов меньше $(a_j - j)$

$p = 2$ (максимальное место)

- Пусть спортсмен i стартует с задержкой D
- Устройство j выключено для всех $j : |i - j| < D$
- Место будет максимальным если устройство включено для всех $j : |i - j| \geqslant D$
- Решение за $O(n^3)$ (≈ 20 баллов): переберем i и D, посчитаем место

$p = 2$, $O(n^2)$ (≈ 30 баллов)

- Рассмотрим три случая:
  1. Устройство 1 включено, а n — выключено
  2. Устройство n включено, а 1 — выключено
  3. Включены оба устройства 1 и n
- Для каждого из случаев чем больше D, тем больше место участника i
  1. $D = i - 1$
  2. $D = n - i$
  3. $D = \min(i - 1, n - i)$
- Перебираем i и одно из трех D, считаем место

$p = 2$, $O(n \log n)$ (до 60 баллов)

- Опять рассмотрим отдельно $j < i$ и $j > i$
- Пусть $j < i$
- Участники $j < i - D$ имеют фору ровно D, для них нужно найти, сколько $j : a_j < a_i + D$
- Участники $j \geqslant i - D$ имеют фору $i - j$, для них нужно найти, сколько $j : a_j < a_i + i - j$, что эквивалентно $a_j + j < a_i + i$
- Нужны три сортированных структуры данных: для a_j, для $(a_j + j)$, для $(a_j - j)$

Пишем аккуратно ($p = 1, 2$)

- Старшее слагаемое в асимптотике — $\Theta(n \log n)$, выкидываем все лишние $\Theta(n \log n)$ из асимптотики
- Не заводим массив событий (и не сортируем его), раскладываем по векторам или на лету понимаем, какие надо делать запросы
- Предподсчитываем для каждого i позиции $a_i + i$, a_i и $a_i - i$ в своих сортированных массивах, сокращаем число бинарных поисков
- Быстрый ввод помогает совсем чуть-чуть (80 мсек под g++ под Linux)

Разные сорта быстрых решений ($p = 1, 2$)

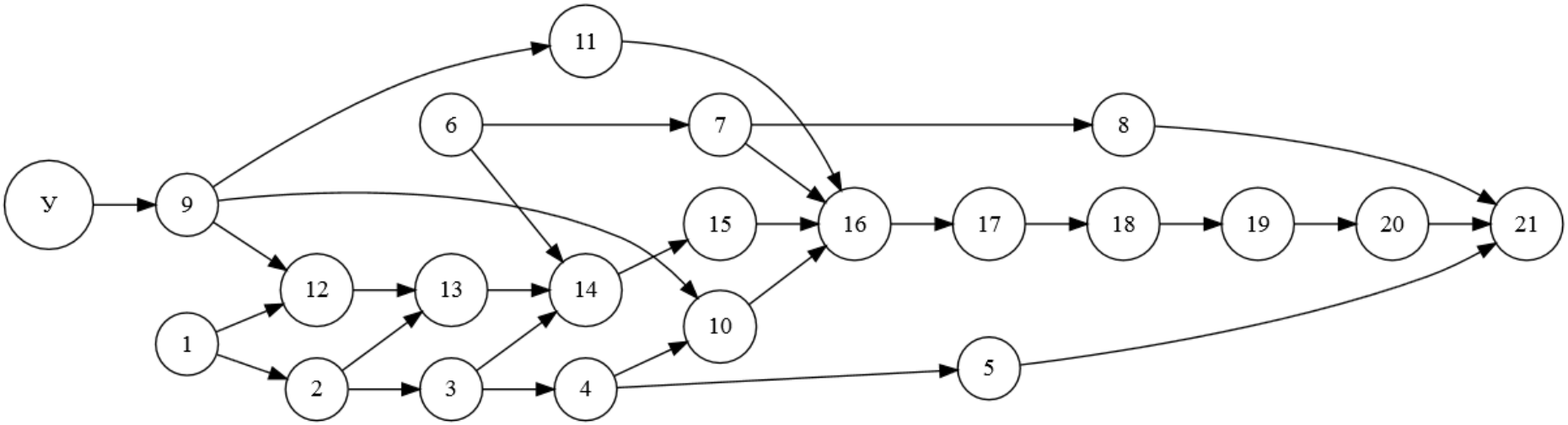
- Декартово дерево — 70–80 баллов
- Двумерное дерево отрезков — 70–80 баллов
- Дерево отрезков — 80–90 баллов
- Дерево Фенвика — 80–100 баллов

Вопросы?

Задача 8 «Сложение без переносов»

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

Наглядный рисунок



Постановка задачи

- Дано n чисел a_i
- Надо найти набор чисел b_i, такой что $b_i > a_i$ и каждый бит есть максимум в одном числе

Решение для $n = 2$ (12 баллов)

- *Увеличение* числа:
  - Один из нулевых битов ставится в единицу
  - Все младшие него сбрасываются в ноль
- Можно увеличивать только одно число:
  - Пусть увеличили оба
  - Одно из двух увеличили в более старшем бите
  - Если убрать изменение второго станет не хуже
- Найдем старший бит, равный 1 в обоих числах
- Найдем младший бит, равный 0 в обоих числах старше этого
- Переберем в каком числе его *увеличить*.

Решение для $n = 2$ (12 баллов)

- *Увеличение* числа:
  - Один из нулевых битов ставится в единицу
  - Все младшие него сбрасываются в ноль
- Можно увеличивать только одно число:
  - Пусть увеличили оба
  - Одно из двух увеличили в более старшем бите
  - Если убрать изменение второго станет не хуже
- Найдем старший бит, равный 1 в обоих числах
- Найдем младший бит, равный 0 в обоих числах старше этого
- Переберем в каком числе его *увеличить*.

Решение для $n = 2$ (12 баллов)

- *Увеличение* числа:
  - Один из нулевых битов ставится в единицу
  - Все младшие него сбрасываются в ноль
- Можно увеличивать только одно число:
  - Пусть увеличили оба
  - Одно из двух увеличили в более старшем бите
  - Если убрать изменение второго станет не хуже
- Найдем старший бит, равный 1 в обоих числах
- Найдем младший бит, равный 0 в обоих числах старше этого
- Переберем в каком числе его *увеличить*.

Решение для $n = 2$ (12 баллов)

- *Увеличение* числа:
  - Один из нулевых битов ставится в единицу
  - Все младшие него сбрасываются в ноль
- Можно увеличивать только одно число:
  - Пусть увеличили оба
  - Одно из двух увеличили в более старшем бите
  - Если убрать изменение второго станет не хуже
- Найдем старший бит, равный 1 в обоих числах
- Найдем младший бит, равный 0 в обоих числах старше этого
- Переберем в каком числе его *увеличить*.

Решение для $n = 2$ (12 баллов)

- *Увеличение* числа:
  - Один из нулевых битов ставится в единицу
  - Все младшие него сбрасываются в ноль
- Можно увеличивать только одно число:
  - Пусть увеличили оба
  - Одно из двух увеличили в более старшем бите
  - Если убрать изменение второго станет не хуже
- Найдем старший бит, равный 1 в обоих числах
- Найдем младший бит, равный 0 в обоих числах старше этого
- Переберем в каком числе его *увеличить*.

Решение для $n = 2$ (12 баллов)

- *Увеличение* числа:
  - Один из нулевых битов ставится в единицу
  - Все младшие него сбрасываются в ноль
- Можно увеличивать только одно число:
  - Пусть увеличили оба
  - Одно из двух увеличили в более старшем бите
  - Если убрать изменение второго станет не хуже
- Найдем старший бит, равный 1 в обоих числах
- Найдем младший бит, равный 0 в обоих числах старше этого
- Переберем в каком числе его *увеличить*.

Решение для $n = 2$ (12 баллов)

- *Увеличение* числа:
  - Один из нулевых битов ставится в единицу
  - Все младшие него сбрасываются в ноль
- Можно увеличивать только одно число:
  - Пусть увеличили оба
  - Одно из двух увеличили в более старшем бите
  - Если убрать изменение второго станет не хуже
- Найдем старший бит, равный 1 в обоих числах
- Найдем младший бит, равный 0 в обоих числах старше этого
- Переберем в каком числе его *увеличить*.

Решение для $n = 2$ (12 баллов)

- *Увеличение* числа:
  - Один из нулевых битов ставится в единицу
  - Все младшие него сбрасываются в ноль
- Можно увеличивать только одно число:
  - Пусть увеличили оба
  - Одно из двух увеличили в более старшем бите
  - Если убрать изменение второго станет не хуже
- Найдем старший бит, равный 1 в обоих числах
- Найдем младший бит, равный 0 в обоих числах старше этого
- Переберем в каком числе его *увеличить*.

Решение для $n = 2$ (12 баллов)

- *Увеличение* числа:
  - Один из нулевых битов ставится в единицу
  - Все младшие него сбрасываются в ноль
- Можно увеличивать только одно число:
  - Пусть увеличили оба
  - Одно из двух увеличили в более старшем бите
  - Если убрать изменение второго станет не хуже
- Найдем старший бит, равный 1 в обоих числах
- Найдем младший бит, равный 0 в обоих числах старше этого
- Переберем в каком числе его *увеличить*.

Решение для $n = 2$ (12 баллов)

- *Увеличение* числа:
  - Один из нулевых битов ставится в единицу
  - Все младшие него сбрасываются в ноль
- Можно увеличивать только одно число:
  - Пусть увеличили оба
  - Одно из двух увеличили в более старшем бите
  - Если убрать изменение второго станет не хуже
- Найдем старший бит, равный 1 в обоих числах
- Найдем младший бит, равный 0 в обоих числах старше этого
- Переберем в каком числе его *увеличить*.

Решение для $a_i = 2^{k_i}$ (12 баллов)

- Возьмем младший бит
- Если он установлен в нескольких числах, перенесем все кроме одного на бит старше
- Переходим к более старшим битам по очереди
- Достаточно поддерживать только количество перенесенных чисел, а не сами эти числа
- Получили решение за $O(n)$ или $O(n \log n)$ в зависимости от аккуратности

Решение для $a_i = 2^{k_i}$ (12 баллов)

- Возьмем младший бит
- Если он установлен в нескольких числах, перенесем все кроме одного на бит старше
- Переходим к более старшим битам по очереди
- Достаточно поддерживать только количество перенесенных чисел, а не сами эти числа
- Получили решение за $O(n)$ или $O(n \log n)$ в зависимости от аккуратности

Решение для $a_i = 2^{k_i}$ (12 баллов)

- Возьмем младший бит
- Если он установлен в нескольких числах, перенесем все кроме одного на бит старше
- Переходим к более старшим битам по очереди
- Достаточно поддерживать только количество перенесенных чисел, а не сами эти числа
- Получили решение за $O(n)$ или $O(n \log n)$ в зависимости от аккуратности

Решение для $a_i = 2^{k_i}$ (12 баллов)

- Возьмем младший бит
- Если он установлен в нескольких числах, перенесем все кроме одного на бит старше
- Переходим к более старшим битам по очереди
- Достаточно поддерживать только количество перенесенных чисел, а не сами эти числа
- Получили решение за $O(n)$ или $O(n \log n)$ в зависимости от аккуратности

Решение для $a_i = 2^{k_i}$ (12 баллов)

- Возьмем младший бит
- Если он установлен в нескольких числах, перенесем все кроме одного на бит старше
- Переходим к более старшим битам по очереди
- Достаточно поддерживать только количество перенесенных чисел, а не сами эти числа
- Получили решение за $O(n)$ или $O(n \log n)$ в зависимости от аккуратности

$n, L \leq 5$

- Заметим, что в ответе после увеличения все числа меньше 2^{L+n}: например, всегда можно превратить числа в $2^L, 2^{L+1}, \ldots, 2^{L+n-1}$
- Воспользуемся динамическим программированием «можно ли получить число x как ответ для поднабора первых k чисел».
  - $n2^{n+L}$ состояний
  - 2^{n+L} переходов (попробуем все варианты увеличить число)
  - Итоговая сложность $O(n4^{n+L})$.

$n, L \leq 5$

- Заметим, что в ответе после увеличения все числа меньше 2^{L+n}: например, всегда можно превратить числа в $2^L, 2^{L+1}, \ldots, 2^{L+n-1}$
- Воспользуемся динамическим программированием «можно ли получить число x как ответ для поднабора первых k чисел».
  - $n2^{n+L}$ состояний
  - 2^{n+L} переходов (попробуем все варианты увеличить число)
  - Итоговая сложность $O(n4^{n+L})$.

$n, L \leq 5$

- Заметим, что в ответе после увеличения все числа меньше 2^{L+n}: например, всегда можно превратить числа в $2^L, 2^{L+1}, \ldots, 2^{L+n-1}$
- Воспользуемся динамическим программированием «можно ли получить число x как ответ для поднабора первых k чисел».
  - $n2^{n+L}$ состояний
  - 2^{n+L} переходов (попробуем все варианты увеличить число)
  - Итоговая сложность $O(n4^{n+L})$.

$n, L \leq 5$

- Заметим, что в ответе после увеличения все числа меньше 2^{L+n}: например, всегда можно превратить числа в $2^L, 2^{L+1}, \ldots, 2^{L+n-1}$
- Воспользуемся динамическим программированием «можно ли получить число x как ответ для поднабора первых k чисел».
  - $n2^{n+L}$ состояний
  - 2^{n+L} переходов (попробуем все варианты увеличить число)
  - Итоговая сложность $O(n4^{n+L})$.

$n, L \leq 5$

- Заметим, что в ответе после увеличения все числа меньше 2^{L+n}: например, всегда можно превратить числа в $2^L, 2^{L+1}, \ldots, 2^{L+n-1}$
- Воспользуемся динамическим программированием «можно ли получить число x как ответ для поднабора первых k чисел».
  - $n2^{n+L}$ состояний
  - 2^{n+L} переходов (попробуем все варианты увеличить число)
  - Итоговая сложность $O(n4^{n+L})$.

$n, L \leq 10$

- Если в предыдущем решении рассматривать переходы только вида «перебрать позицию новой единички», останется $O(n + L)$ переходов, и сложность станет $O(n(n + L)2^{n+L})$.

$n \leq 1000, L \leq 5$

- Пусть $n > L$
  - Посмотрим на числа, которые в ответе останутся меньше 2^L. Пусть их k, тогда $k \leq L$
  - Попробуем все $k \leq L$. Для данного k выгодно выбрать k минимальных чисел. Ответ для них найдем решением для $n, L \leq 5$
  - Оставшиеся $n - k$ чисел будут равны 2^L, 2^{L+1}, ...
- Этот метод позволяет в любом тесте заменить n на $\min(n, L)$

$n \leq 1000, L \leq 5$

- Пусть $n > L$
  - Посмотрим на числа, которые в ответе останутся меньше 2^L. Пусть их k, тогда $k \leq L$
  - Попробуем все $k \leq L$. Для данного k выгодно выбрать k минимальных чисел. Ответ для них найдем решением для $n, L \leq 5$
  - Оставшиеся $n - k$ чисел будут равны 2^L, 2^{L+1}, ...
- Этот метод позволяет в любом тесте заменить n на $\min(n, L)$

$n \leq 1000, L \leq 5$

- Пусть $n > L$
  - Посмотрим на числа, которые в ответе останутся меньше 2^L. Пусть их k, тогда $k \leq L$
  - Попробуем все $k \leq L$. Для данного k выгодно выбрать k минимальных чисел. Ответ для них найдем решением для $n, L \leq 5$
  - Оставшиеся $n - k$ чисел будут равны 2^L, 2^{L+1}, ...
- Этот метод позволяет в любом тесте заменить n на $\min(n, L)$

$n \leq 1000, L \leq 5$

- Пусть $n > L$
  - Посмотрим на числа, которые в ответе останутся меньше 2^L. Пусть их k, тогда $k \leq L$
  - Попробуем все $k \leq L$. Для данного k выгодно выбрать k минимальных чисел. Ответ для них найдем решением для $n, L \leq 5$
  - Оставшиеся $n - k$ чисел будут равны 2^L, 2^{L+1}, ...
- Этот метод позволяет в любом тесте заменить n на $\min(n, L)$

$n \leq 1000, L \leq 5$

- Пусть $n > L$
  - Посмотрим на числа, которые в ответе останутся меньше 2^L. Пусть их k, тогда $k \leq L$
  - Попробуем все $k \leq L$. Для данного k выгодно выбрать k минимальных чисел. Ответ для них найдем решением для $n, L \leq 5$
  - Оставшиеся $n - k$ чисел будут равны 2^L, 2^{L+1}, ...
- Этот метод позволяет в любом тесте заменить n на $\min(n, L)$

$n \leq 5, L \leq 1000$

- Динамическое программирование по подмножествам:
  - Рассматриваем биты ответа по убыванию, начиная $n + L$, и в каждом решаем, будем ли мы ставить единичку и в какое число
  - Смотрим на числа, в которых еще не ставили единички. Выбираем лучший из возможных вариантов «ничего не ставить» и «поставить в какое-то число»
  - $dp_{k,S}$ = «ответ, если оставить биты старше k-го, и в подмножестве чисел S применяли увеличение»
  - Итоговая сложность $O(nL(n + L)2^n)$ (доп. множитель $O(L)$ на сравнение результатов)

$n \leq 5, L \leq 1000$

- Динамическое программирование по подмножествам:
  - Рассматриваем биты ответа по убыванию, начиная $n + L$, и в каждом решаем, будем ли мы ставить единичку и в какое число
  - Смотрим на числа, в которых еще не ставили единички. Выбираем лучший из возможных вариантов «ничего не ставить» и «поставить в какое-то число»
  - $dp_{k,S}$ = «ответ, если оставить биты старше k-го, и в подмножестве чисел S применяли увеличение»
  - Итоговая сложность $O(nL(n + L)2^n)$ (доп. множитель $O(L)$ на сравнение результатов)

$n \leq 5, L \leq 1000$

- Динамическое программирование по подмножествам:
  - Рассматриваем биты ответа по убыванию, начиная $n + L$, и в каждом решаем, будем ли мы ставить единичку и в какое число
  - Смотрим на числа, в которых еще не ставили единички. Выбираем лучший из возможных вариантов «ничего не ставить» и «поставить в какое-то число»
  - $dp_{k,S}$ = «ответ, если оставить биты старше k-го, и в подмножестве чисел S применяли увеличение»
  - Итоговая сложность $O(nL(n + L)2^n)$ (доп. множитель $O(L)$ на сравнение результатов)

$n \leq 5, L \leq 1000$

- Динамическое программирование по подмножествам:
  - Рассматриваем биты ответа по убыванию, начиная $n + L$, и в каждом решаем, будем ли мы ставить единичку и в какое число
  - Смотрим на числа, в которых еще не ставили единички. Выбираем лучший из возможных вариантов «ничего не ставить» и «поставить в какое-то число»
  - $dp_{k,S} =$ «ответ, если оставить биты старше k-го, и в подмножестве чисел S применяли увеличение»
  - Итоговая сложность $O(nL(n + L)2^n)$ (доп. множитель $O(L)$ на сравнение результатов)

$n \leq 5, L \leq 1000$

- Динамическое программирование по подмножествам:
  - Рассматриваем биты ответа по убыванию, начиная $n + L$, и в каждом решаем, будем ли мы ставить единичку и в какое число
  - Смотрим на числа, в которых еще не ставили единички. Выбираем лучший из возможных вариантов «ничего не ставить» и «поставить в какое-то число»
  - $dp_{k,S} =$ «ответ, если оставить биты старше k-го, и в подмножестве чисел S применяли увеличение»
  - Итоговая сложность $O(nL(n + L)2^n)$ (доп. множитель $O(L)$ на сравнение результатов)

$O(n^c)$

- Научимся проверять, можно ли получить ответ, который помещается в k битов
  - Дополним числа нулями до k битов и отсортируем по неубыванию
  - Если есть два числа с единичкой в k-ом бите, ответа нет
  - Если такое число одно, то остальные числа, а также меньшие биты этого числа, должны помещаться в $k-1$ бит, проверим рекурсивно
  - Если таких чисел нет, максимальное число можно увеличить до 2^{k-1}, для оставшихся проверить рекурсивно, помещаются ли они в $k-1$ бит

$O(n^c)$

- Научимся проверять, можно ли получить ответ, который помещается в k битов
  - Дополним числа нулями до k битов и отсортируем по неубыванию
  - Если есть два числа с единичкой в k-ом бите, ответа нет
  - Если такое число одно, то остальные числа, а также меньшие биты этого числа, должны помещаться в $k-1$ бит, проверим рекурсивно
  - Если таких чисел нет, максимальное число можно увеличить до 2^{k-1}, для оставшихся проверить рекурсивно, помещаются ли они в $k-1$ бит

$O(n^c)$

- Научимся проверять, можно ли получить ответ, который помещается в k битов
  - Дополним числа нулями до k битов и отсортируем по неубыванию
  - Если есть два числа с единичкой в k-ом бите, ответа нет
  - Если такое число одно, то остальные числа, а также меньшие биты этого числа, должны помещаться в $k-1$ бит, проверим рекурсивно
  - Если таких чисел нет, максимальное число можно увеличить до 2^{k-1}, для оставшихся проверить рекурсивно, помещаются ли они в $k-1$ бит

$O(n^c)$

- Научимся проверять, можно ли получить ответ, который помещается в k битов
  - Дополним числа нулями до k битов и отсортируем по неубыванию
  - Если есть два числа с единичкой в k-ом бите, ответа нет
  - Если такое число одно, то остальные числа, а также меньшие биты этого числа, должны помещаться в $k-1$ бит, проверим рекурсивно
  - Если таких чисел нет, максимальное число можно увеличить до 2^{k-1}, для оставшихся проверить рекурсивно, помещаются ли они в $k-1$ бит

$O(n^c)$

- Научимся проверять, можно ли получить ответ, который помещается в k битов
  - Дополним числа нулями до k битов и отсортируем по неубыванию
  - Если есть два числа с единичкой в k-ом бите, ответа нет
  - Если такое число одно, то остальные числа, а также меньшие биты этого числа, должны помещаться в $k-1$ бит, проверим рекурсивно
  - Если таких чисел нет, максимальное число можно увеличить до 2^{k-1}, для оставшихся проверить рекурсивно, помещаются ли они в $k-1$ бит

$O(n^c)$

Посмотрим на пример для $k = 5$:

01110
01010
00011
00001

$O(n^c)$

Посмотрим на пример для $k = 5$:

$$10000$$
$$01010$$
$$00011$$
$$00001$$

$O(n^c)$

Посмотрим на пример для $k = 5$:

$$10000$$
$$01010$$
$$00011$$
$$00001$$

$O(n^c)$

Посмотрим на пример для $k = 5$:

$$10000$$
$$01010$$
$$00100$$
$$00001$$

$O(n^c)$

Посмотрим на пример для $k = 5$:

$$10000$$
$$01010$$
$$00100$$
$$00001$$

$O(n^c)$

Посмотрим на пример для $k = 5$:

$$10000$$
$$01010$$
$$00100$$
$$00001$$

$O(n^c)$

- Теперь решим исходную задачу:
  - Пойдем от старших битов ответа к младшим, и будем пытаться жадно ставить нули в биты, когда это можно сделать
  - Можем поставить 0 в k-ый бит, если ответ помещается в $k-1$ бит
  - Если поставить нельзя, обязаны поставить в k-ый бит ответа единицу. Как и в проверке ответа, требуется либо выкинуть максимальное число (если в нем в k-ом бите стоит 0), либо откинуть у него старший бит (если там 1).

$O(n^c)$

- Теперь решим исходную задачу:
  - Пойдем от старших битов ответа к младшим, и будем пытаться жадно ставить нули в биты, когда это можно сделать
  - Можем поставить 0 в k-ый бит, если ответ помещается в $k-1$ бит
  - Если поставить нельзя, обязаны поставить в k-ый бит ответа единицу. Как и в проверке ответа, требуется либо выкинуть максимальное число (если в нем в k-ом бите стоит 0), либо откинуть у него старший бит (если там 1).

$O(n^c)$

- Теперь решим исходную задачу:
  - Пойдем от старших битов ответа к младшим, и будем пытаться жадно ставить нули в биты, когда это можно сделать
  - Можем поставить 0 в k-ый бит, если ответ помещается в $k-1$ бит
  - Если поставить нельзя, обязаны поставить в k-ый бит ответа единицу. Как и в проверке ответа, требуется либо выкинуть максимальное число (если в нем в k-ом бите стоит 0), либо откинуть у него старший бит (если там 1).

$O(n^c)$

- Теперь решим исходную задачу:
  - Пойдем от старших битов ответа к младшим, и будем пытаться жадно ставить нули в биты, когда это можно сделать
  - Можем поставить 0 в k-ый бит, если ответ помещается в $k-1$ бит
  - Если поставить нельзя, обязаны поставить в k-ый бит ответа единицу. Как и в проверке ответа, требуется либо выкинуть максимальное число (если в нем в k-ом бите стоит 0), либо откинуть у него старший бит (если там 1).

$O(t^4)$

- Тривиальная реализация этого решения имеет сложность $O(t^4)$ (где $t = n + L$) и набирает $24 - 31$ балл ($44 - 51$ баллов, если также решить все предыдущие независимые подзадачи).

$O(t^3 \log t)$, $O(t^2 \log^2 t)$

- Вместо линейного поиска максимума среди строк можно использовать `set`, а также хэши для быстрого сравнения строк (потенциально $58 - 64$ балла).

$O(t^2 \log t)$

- Заметим, что нам нужно сравнивать только суффиксы данных строк, начинающихся с 1. Пронумеруем такие суффиксы по возрастанию значений чисел (также будем рассматривать пустой суффикс, имеющий номер 0).

$O(t^2 \log t)$

- Это можно сделать за $O(S)$, где S — суммарная длина входных чисел. Пускай все суффиксы длины меньше k уже пронумерованы номерами до x. Тогда суффиксы длины k требуется пронумеровать, начиная с $x + 1$, в порядке возрастания номеров суффиксов с удаленными ведущими единицами.

$O(t^2 \log t)$

```
01111
01010
00011
00001

-????
-?-?-
---??
----?
```

$O(t^2 \log t)$

```
01111
01010
00011
00001

-???1
-?-?-
---?2
----3
```

$O(t^2 \log t)$

```
01110
01010
00011
00001

-??51
-?-4-
---62
----3
```

$O(t^2 \log t)$

```
01110
01010
00011
00001

-?751
-?-4-
---62
----3
```

$O(t^2 \log t)$

```
01110
01010
00011
00001

-9751
-8-4-
---62
----3
```

$O(t^2 \log t)$

- Теперь мы можем везде за $O(1)$ сравнивать числа вместо строк. Это решение набирает $60 - 74$ баллов.

$O(t^2 \log t)$

- От логарифма в сложности можно избавиться, если вместо set'а хранить количество и минимальную строку для каждой длины.
- На каждом шаге мы либо пропустим все строки текущей длины, либо добавим суффикс минимальной из таких строк, либо закончим проверку.
- До 80 баллов

$O(t^2 \log t)$

- От логарифма в сложности можно избавиться, если вместо set'а хранить количество и минимальную строку для каждой длины.
- На каждом шаге мы либо пропустим все строки текущей длины, либо добавим суффикс минимальной из таких строк, либо закончим проверку.
- До 80 баллов

$O(t^2 \log t)$

- От логарифма в сложности можно избавиться, если вместо set'а хранить количество и минимальную строку для каждой длины.
- На каждом шаге мы либо пропустим все строки текущей длины, либо добавим суффикс минимальной из таких строк, либо закончим проверку.
- До 80 баллов

Ускоряем квадрат

- Для полного решения требуется более оптимальная реализация того же по сути процесса.
- Снова упорядочим числа по неубыванию, пусть длина i-го этом порядке числа равна l_i.

Ускоряем квадрат

- Для полного решения требуется более оптимальная реализация того же по сути процесса.
- Снова упорядочим числа по неубыванию, пусть длина i-го этом порядке числа равна l_i.

Ускоряем квадрат

- Заметим, что длина ответа не может быть меньше, чем $b = \max_i(l_i + i - 1)$. Действительно, если проверка ответа для k битов из прошлого решения была успешна, то на j-ом шаге длина максимального числа не превосходит $k - j + 1$.
- Кроме того, длина ответа не больше, чем $b + 1$, поскольку i-ое число можно увеличить до 2^{b-i+2}.

Ускоряем квадрат

- Заметим, что длина ответа не может быть меньше, чем $b = \max_i(l_i + i - 1)$. Действительно, если проверка ответа для k битов из прошлого решения была успешна, то на j-ом шаге длина максимального числа не превосходит $k - j + 1$.
- Кроме того, длина ответа не больше, чем $b + 1$, поскольку i-ое число можно увеличить до 2^{b-i+2}.

Ускоряем квадрат

- Назовем число *критическим*, если его суффикс еще не пытались добавить в множество (даже если это пустой суффикс), и для него выполняется равенство $l_i + i - 1 = b$. Мы знаем, что длина ответа равна либо b, либо $b + 1$, попробуем отличить эти ситуации.
- Запустим проверку на то, что ответ равен b. Если k — первое критическое число, то все числа с позициями до k можно пропустить.

Ускоряем квадрат

- Назовем число *критическим*, если его суффикс еще не пытались добавить в множество (даже если это пустой суффикс), и для него выполняется равенство $l_i + i - 1 = b$. Мы знаем, что длина ответа равна либо b, либо $b + 1$, попробуем отличить эти ситуации.
- Запустим проверку на то, что ответ равен b. Если k — первое критическое число, то все числа с позициями до k можно пропустить.

Ускоряем квадрат

- После этого произойдет добавление суффикса k-го числа в множества, что приведет к перенумерации чисел. Если после этого в множестве для какого-то числа $l_i + i - 1 > b$, проверка не удалась.
- Этот процесс требуется продолжать, пока в множестве есть критические числа. Если проверка удалась (не появилось чисел с $l_i + i - 1 > b$), то проверка удалась, и все добавленные числа навсегда добавились в множество.

Ускоряем квадрат

- После этого произойдет добавление суффикса k-го числа в множества, что приведет к перенумерации чисел. Если после этого в множестве для какого-то числа $l_i + i - 1 > b$, проверка не удалась.
- Этот процесс требуется продолжать, пока в множестве есть критические числа. Если проверка удалась (не появилось чисел с $l_i + i - 1 > b$), то проверка удалась, и все добавленные числа навсегда добавились в множество.

Ускоряем квадрат

11111
11001
00110
00010

В этом примере $b = 6$, и критическим числом является только 11001. Проверим, равна ли длина ответа 6.

Ускоряем квадрат

11111
11001
01001
00110
00010

Добавим суффикс 11001 в множество, он становится критическим. Старое число перестало быть критическим, поскольку его суффикс добавили.

Ускоряем квадрат

11111
11001
01001
00110
00010
00001

После добавления суффикса 1001 сразу три числа стали критическими из-за перенумерации.

Ускоряем квадрат

11111
11001
01001
00110
00010
00010
00001

После следующей итерации два последних числа нарушают неравенство $l_i + i - 1 \leq b$, поэтому проверка не удалась, и нам требуется вернуть множество в исходное положение.

Ускоряем квадрат

- Если проверка не удалась, то длина ответа $b+1$, и мы знаем начало ответа до первого критического числа. Если критические числа еще остались, следует запустить такую же проверку с новой позиции в множестве.
- Для этой процедуры нам понадобится структура данных для поиска первого критического числа, например, дерево отрезков или декартово дерево с операцией поиска первого числа с максимальным значением $l_i + i - 1$.

Ускоряем квадрат

- Если проверка не удалась, то длина ответа $b+1$, и мы знаем начало ответа до первого критического числа. Если критические числа еще остались, следует запустить такую же проверку с новой позиции в множестве.
- Для этой процедуры нам понадобится структура данных для поиска первого критического числа, например, дерево отрезков или декартово дерево с операцией поиска первого числа с максимальным значением $l_i + i - 1$.

$O(S \log S)$?

- Остается выяснить, сколько операций нам может потребоваться для всех проверок в сумме. Покажем, что это количество составляет $O(S)$, где S — суммарная длина всех чисел во входе.
- Все успешные проверки потребуют $O(S)$ операций, поскольку каждая такая операция перманентно добавляет один суффикс в множество.

$O(S \log S)$?

- Остается выяснить, сколько операций нам может потребоваться для всех проверок в сумме. Покажем, что это количество составляет $O(S)$, где S — суммарная длина всех чисел во входе.
- Все успешные проверки потребуют $O(S)$ операций, поскольку каждая такая операция перманентно добавляет один суффикс в множество.

$O(S \log S)$!

- Неудачные проверки могут требовать большого количества операций, которые мы потом отменим. Тем не менее, в сумме они тоже потребуют $O(S)$ операций.
- Действительно, пускай мы сделали неудачную проверку для критического числа длины l. Тогда мы сделаем не более, чем l операций, поскольку все добавленные строки будут иметь разную длину.
- Более того, неудачная проверка может произойти для не более одного суффикса каждой из исходных строк. Таким образом, суммарное

$O(S \log S)$!

- Неудачные проверки могут требовать большого количества операций, которые мы потом отменим. Тем не менее, в сумме они тоже потребуют $O(S)$ операций.
- Действительно, пускай мы сделали неудачную проверку для критического числа длины l. Тогда мы сделаем не более, чем l операций, поскольку все добавленные строки будут иметь разную длину.
- Более того, неудачная проверка может произойти для не более одного суффикса каждой из исходных строк. Таким образом, суммарное

$O(S \log S)$!

- Неудачные проверки могут требовать большого количества операций, которые мы потом отменим. Тем не менее, в сумме они тоже потребуют $O(S)$ операций.
- Действительно, пускай мы сделали неудачную проверку для критического числа длины l. Тогда мы сделаем не более, чем l операций, поскольку все добавленные строки будут иметь разную длину.
- Более того, неудачная проверка может произойти для не более одного суффикса каждой из исходных строк. Таким образом, суммарное

$O(S \log S)$!

Это решение имеет сложность $O(S \log S)$ и набирает полный балл.

Вопросы?