Всероссийская олимпиада школьников по информатике

Казань

10–16 апреля 2012 года

Задача «Пароль»

Задача «Пароль»

- Идея задачи — Юрий Петров
- Подготовка тестов — Сергей Мельников
- Разбор задачи — Сергей Мельников

Математическая формулировка

Входные данные:

- Две строки из цифр, первая состоит из n цифр
- В первой строке подстроку заменили на сумму цифр

Результат:

- Найти позиции подстроки, которую заменили на сумму цифр

Пример:

$s = 1\,\underbrace{234}\,5$

$t = \quad 1\;9\;5$

Решение за $O(n^3)$

- Переберём все подстроки первой строки.
- Для каждой посчитаем сумму цифр, и проверим получается ли вторая строка.
- Такое решение получало 30 баллов

Частичные суммы на префиксе

- $a[k]$ - сумма цифр с первой по k-ую
- Вычислим $a[k] = \sum_{i=1}^{k} s[i]$
- Можно вычислять последовательно: $a[k] = a[k-1] + s[k]$
- Сумма цифр с L по $R = a[R] - a[L-1]$.

Пример:

$s = 1\ 2\ 3\ 4\ 5$

$a = 1\ 3\ 6\ 10\ 15$

Сумма цифр с второй по четвёртую:

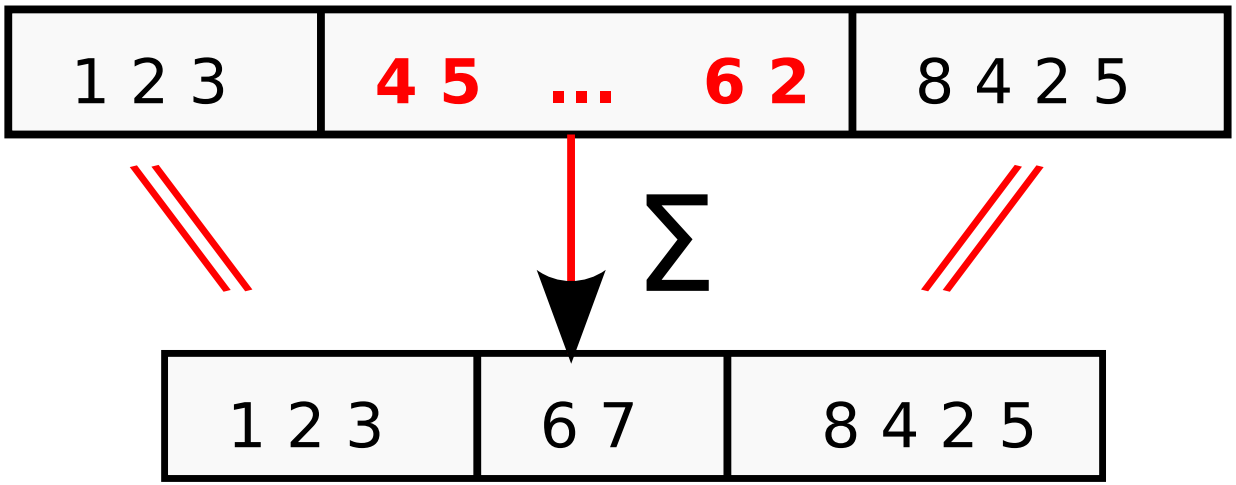
$a[4] - a[1] = 10 - 1 = 9$

Проверка сопадения префикса

- Найдём наибольший общий префикс двух строк
- Теперь за $O(1)$ можем проверить совпадает ли префикс длины p
- Аналогично суффикс

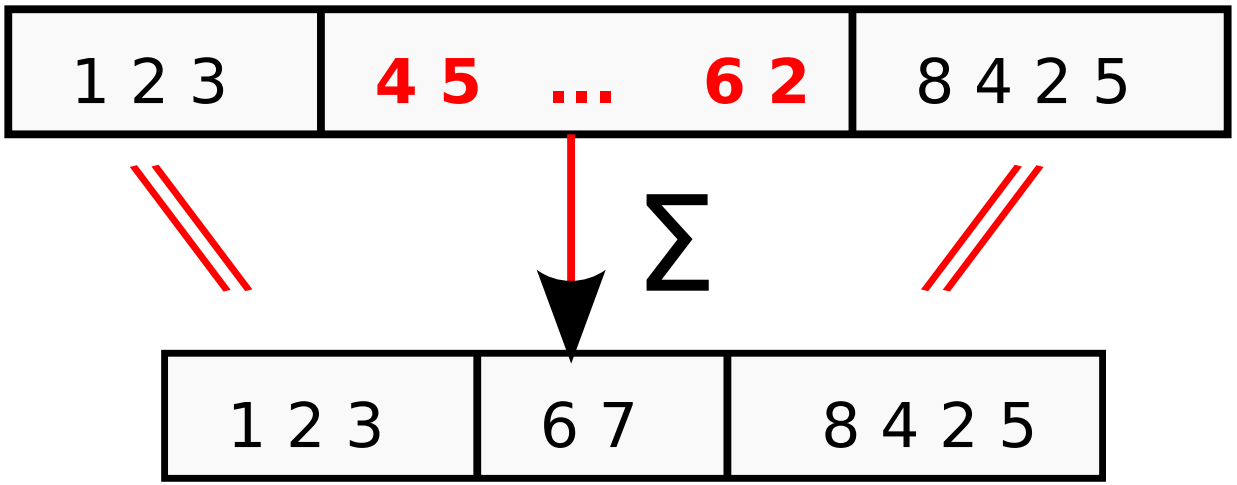
$O(n^2)$: подходит ли строка

- Перебираем подстроки первой строки
- Проверяем, что все символы перед ней совпадают с соответствующими символами второй
- Аналогично для символов после рассматриваемой подстроки



$O(n^2)$: подходит ли строка

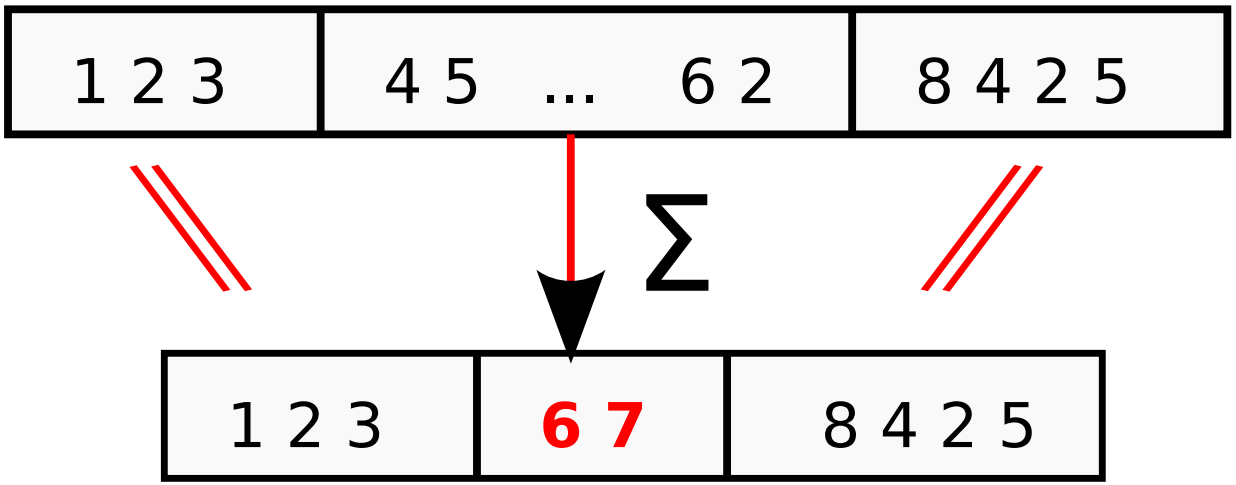
- Проверяем, совпадает ли сумма символов подстроки с нужным числом во второй строке
- Заметим, что сумма цифр не больше 10 $\times$ длину строки
- Длина суммы цифр подстроки не превосходит $log(n)$, в ограничениях задачи 8.



- Такое решение получало 60 баллов

Решение за $O(n \log n)$

- Перебираем подстроки второй строки, которые могут быть суммой
- Однозначно определяем совпадающие префикс и суффикс
- Проверяем, сопадает ли сумма цифр остатка первой строки с рассматриваемой подстрокой.



Вопросы?

Задача «Вирусы и антивирусы»

Задача «Вирусы и антивирусы»

Задача «Вирусы и антивирусы»

Задача «Вирусы и антивирусы»

- Идея задачи — Глеб Евстропов, Михаил Дворкин, Павел Кунявский
- Подготовка тестов — Юрий Петров
- Разбор задачи — Андрей Станкевич

Задача

- Задано два ориентированных исходящих дерева на одном и том же множестве вершин
- Требуется найти количество пар (u, v), чтобы v была потомком u в обоих деревьях

Решение за N^3

- Для каждой пары вершин проверим, что есть путь из u в v в обеих деревьях
  - Либо каждый раз запустим поиск в глубину
  - Либо один раз алгоритм Флойда и затем выполним проверку за $O(1)$
- 25 баллов

Решение за N^2

- Для каждой вершины u запустим из нее поиск в глубину в обоих деревьях
  - Число ребер в дереве с N вершинами равно $N - 1$, поэтому поиск в глубину работает за $O(N)$
- Прибавим к ответу число вершин, достижимых в обоих деревьях
- 50 баллов

Решение за $N \log N$

- Запустим обход в глубину от корня дерева
- Для каждой вершины посчитаем «время входа» и «время выхода»
- Сделаем это в каждом дереве
- Каждой вершине можно сопоставить две пары чисел:
  - $(enter1_u, enter2_u)$
  - $(leave1_u, leave2_u)$

Как определить, что вершина является предком?

- Вершина u является предком вершины v в дереве, если
  - $enter_u \leqslant enter_v \leqslant leave_u$
- Число предков u — число таких вершин
- Рассмотрим числовую прямую
- Вершине v сопоставим точку на прямой с координатой $enter_v$
- Число потомков вершины u — число точек на отрезке $[enter_u, enter_v]$

Что делать, если два дерева?

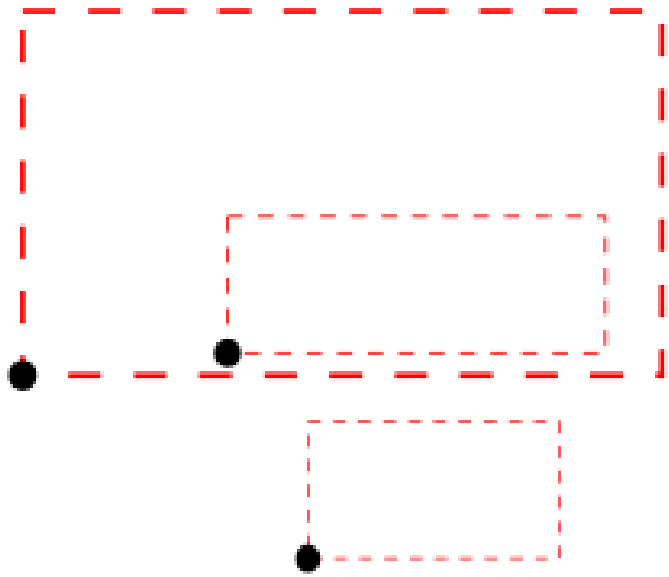
- Вершина u является предком вершины v в первом дереве, если
  - $enter1_u \leqslant enter1_v \leqslant leave1_u$
- Вершина u является предком вершины v во втором дереве, если
  - $enter2_u \leqslant enter2_v \leqslant leave2_u$
- Сопоставим вершине v точку на плоскости с координатами $(enter1_v, enter2_v)$
- Количество потомков вершины u в обоих деревьях — количество точек в прямоугольнике
  - $[enter1_u, leave1_u] \times [enter2_u, leave2_u]$

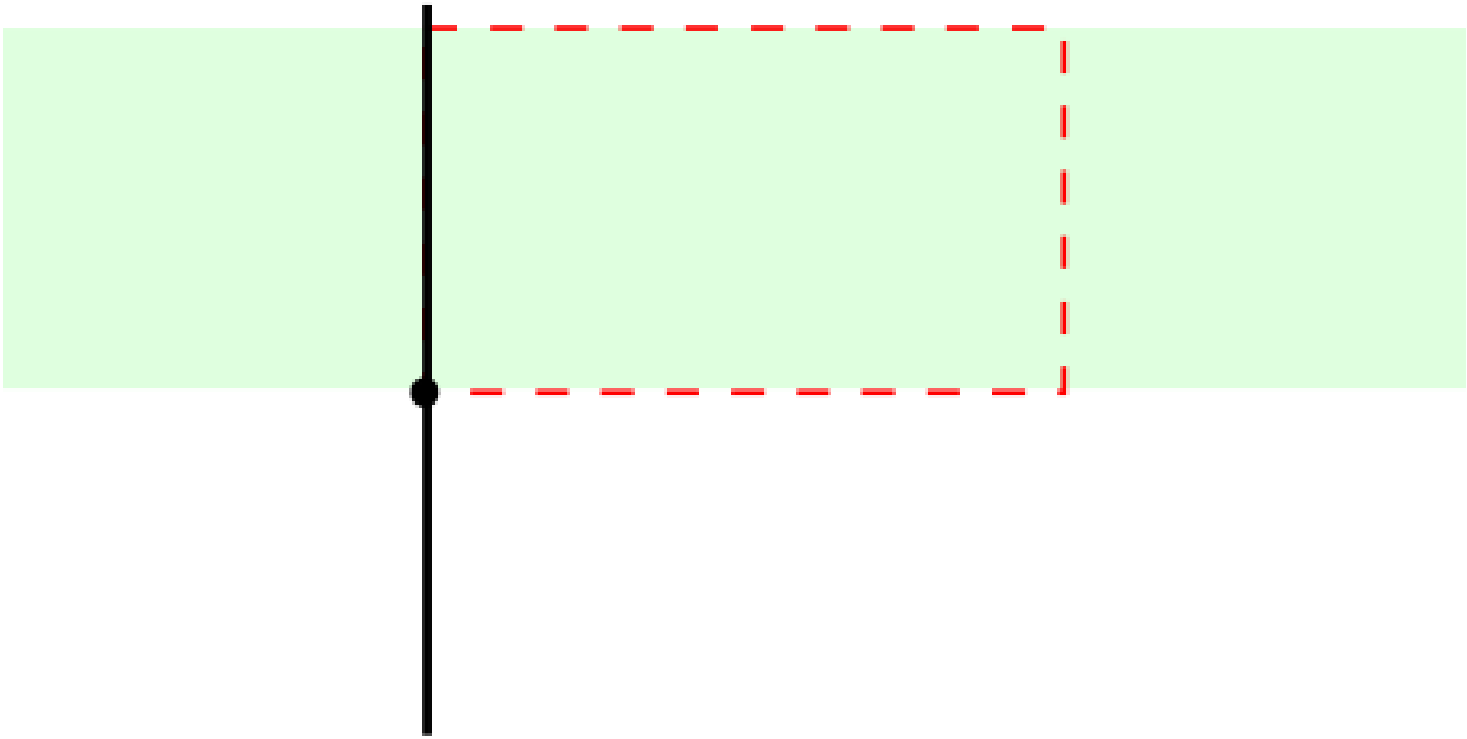
Как посчитать число точек в прямоугольнике?

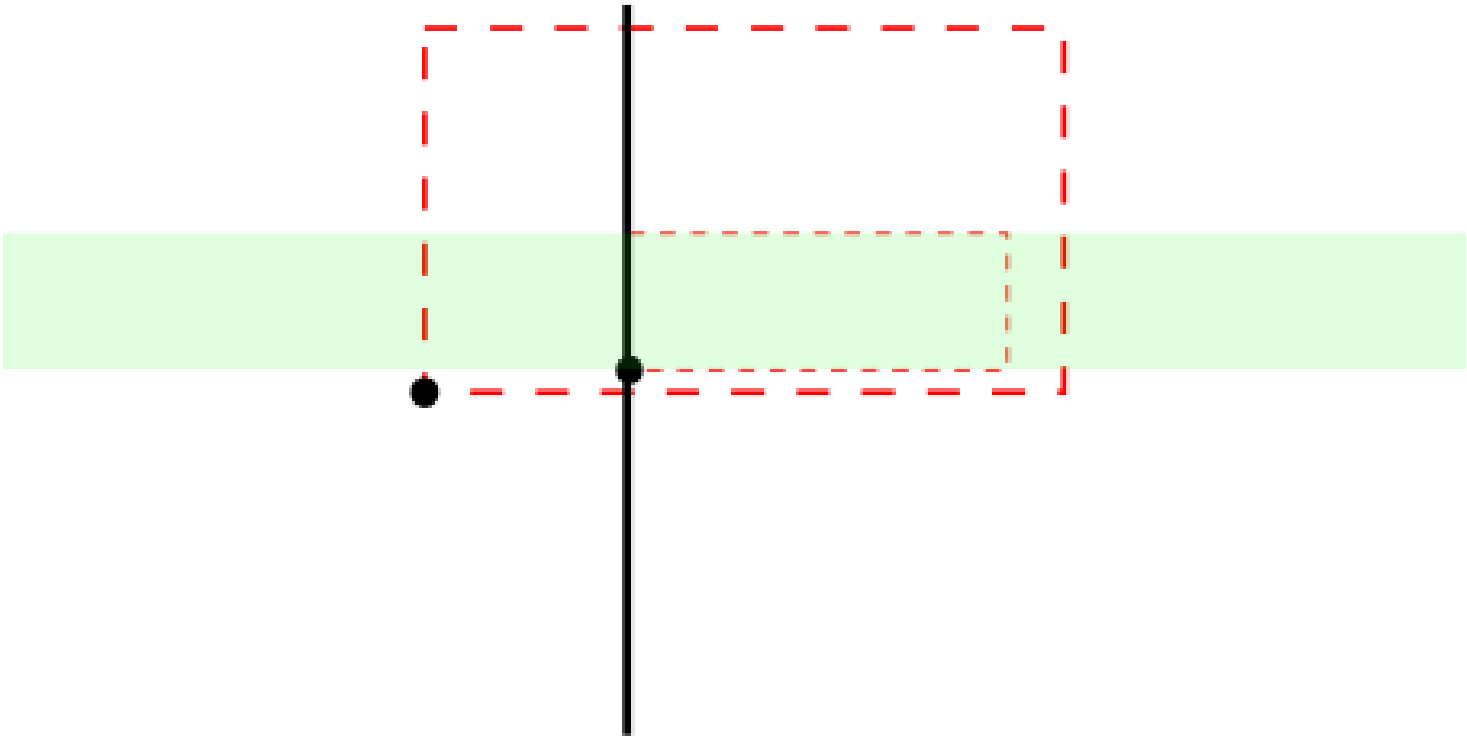
- Задача: даны точки на плоскости и прямоугольники. Требуется подсчитать сумму числа точек внутри для всех прямоугольников
- Решение:
  - Сканирующая прямая
  - Деревья отрезков с операцией «сумма»

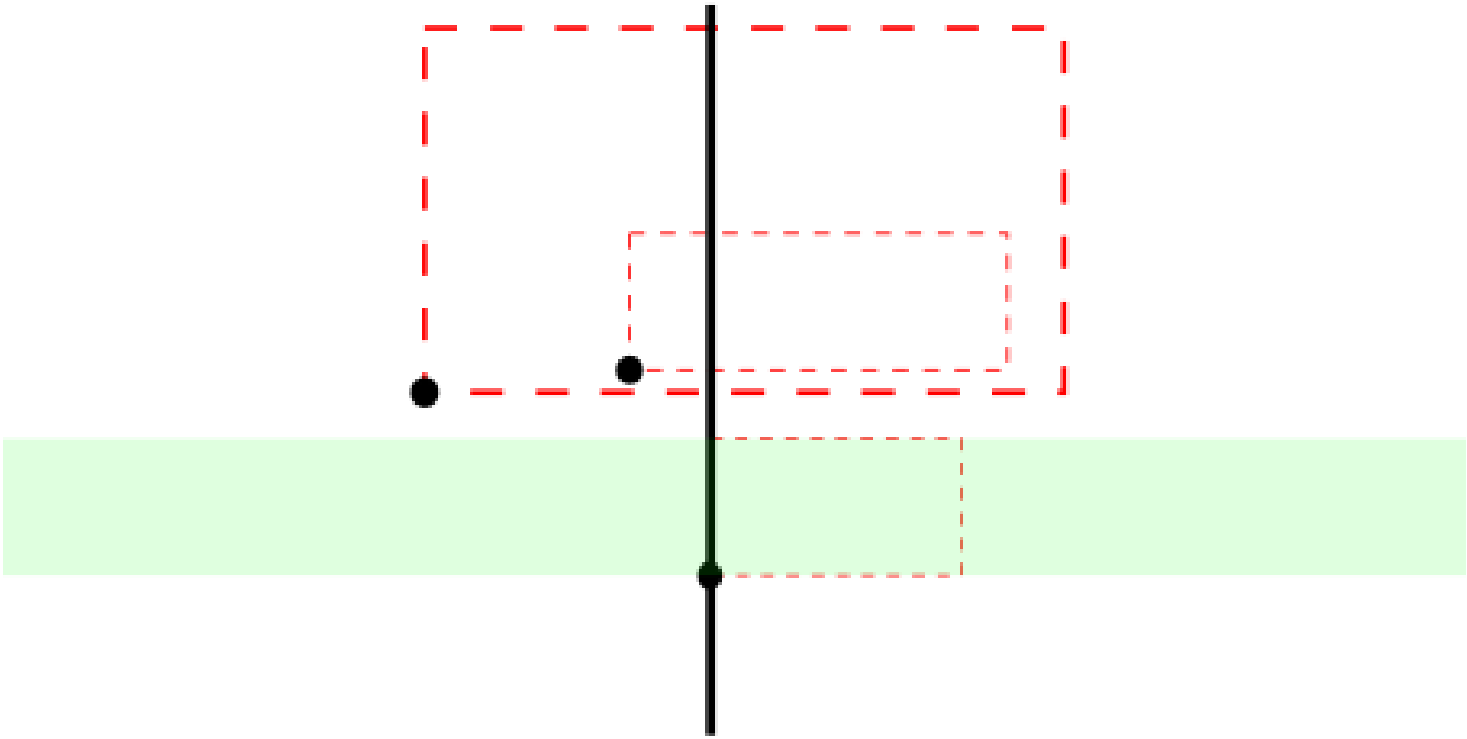
Подсчет числа точек в прямоугольнике

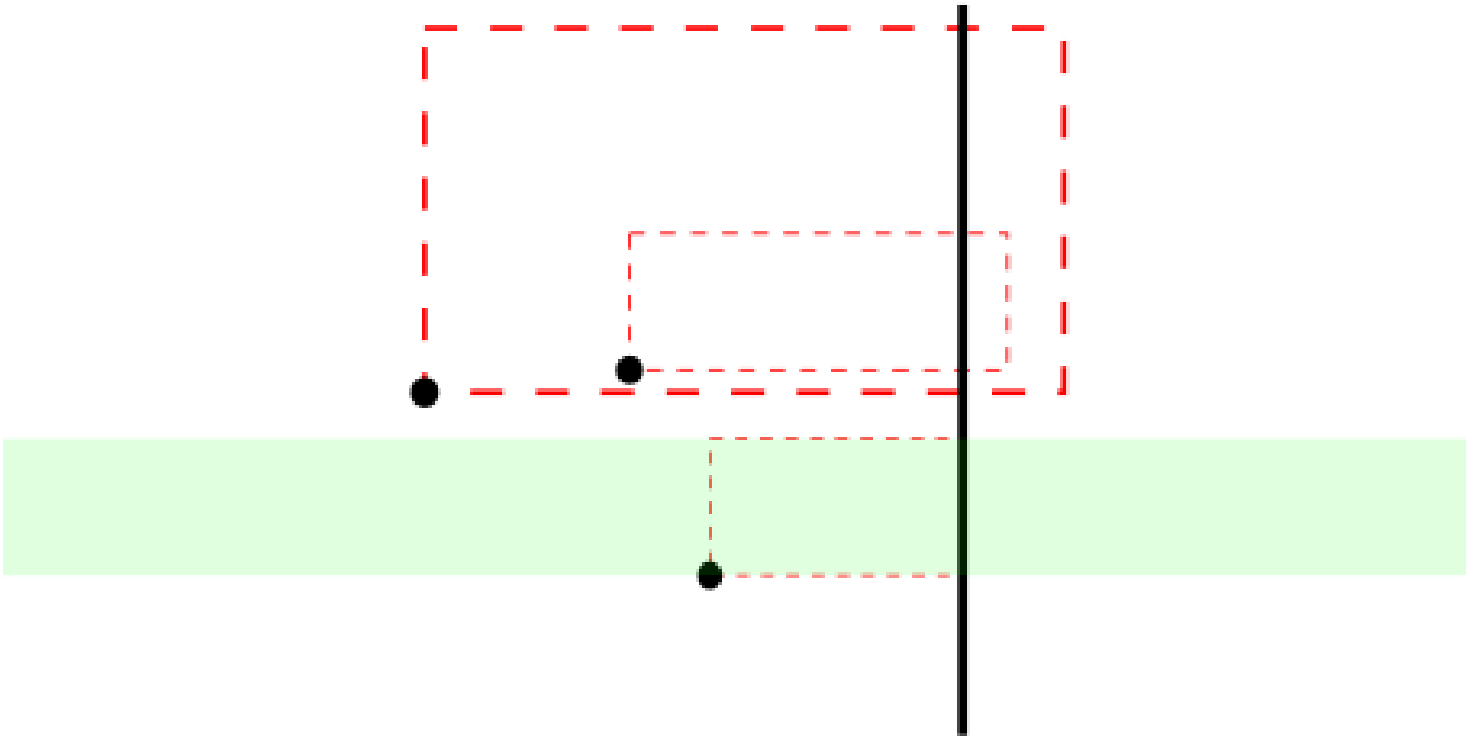
- Пройдем «сканирующей прямой» по возрастанию первой координаты
- Создадим «события» —
  - начало прямоугольника
  - конец прямоугольника
  - появление точки
- В начале прямоугольника вычитаем из ответа сумму на отрезке, соответствующей его второй координате
- В конце прямоугольника — прибавляем к ответу эту сумму
- При появлении точки добавляем 1 к значению в этой позиции в дереве отрезков

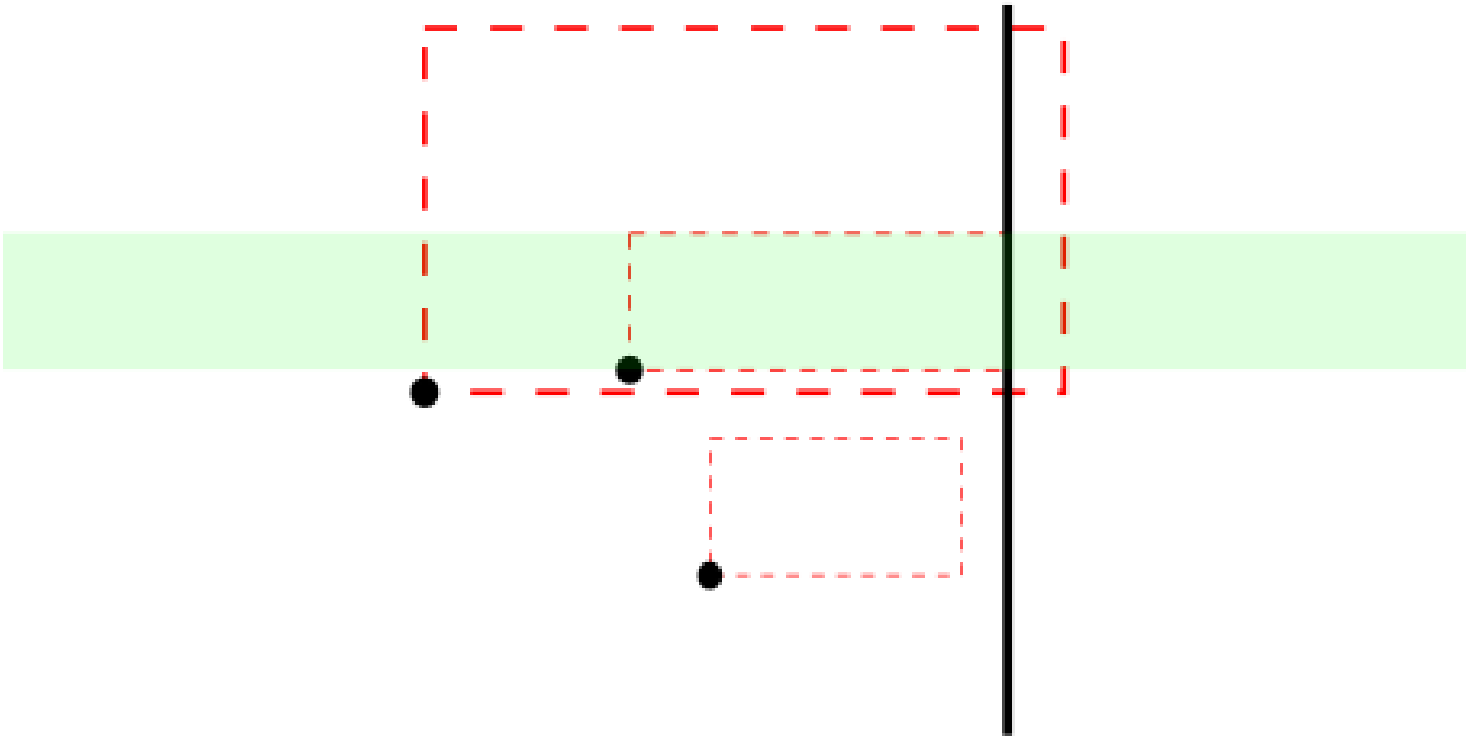


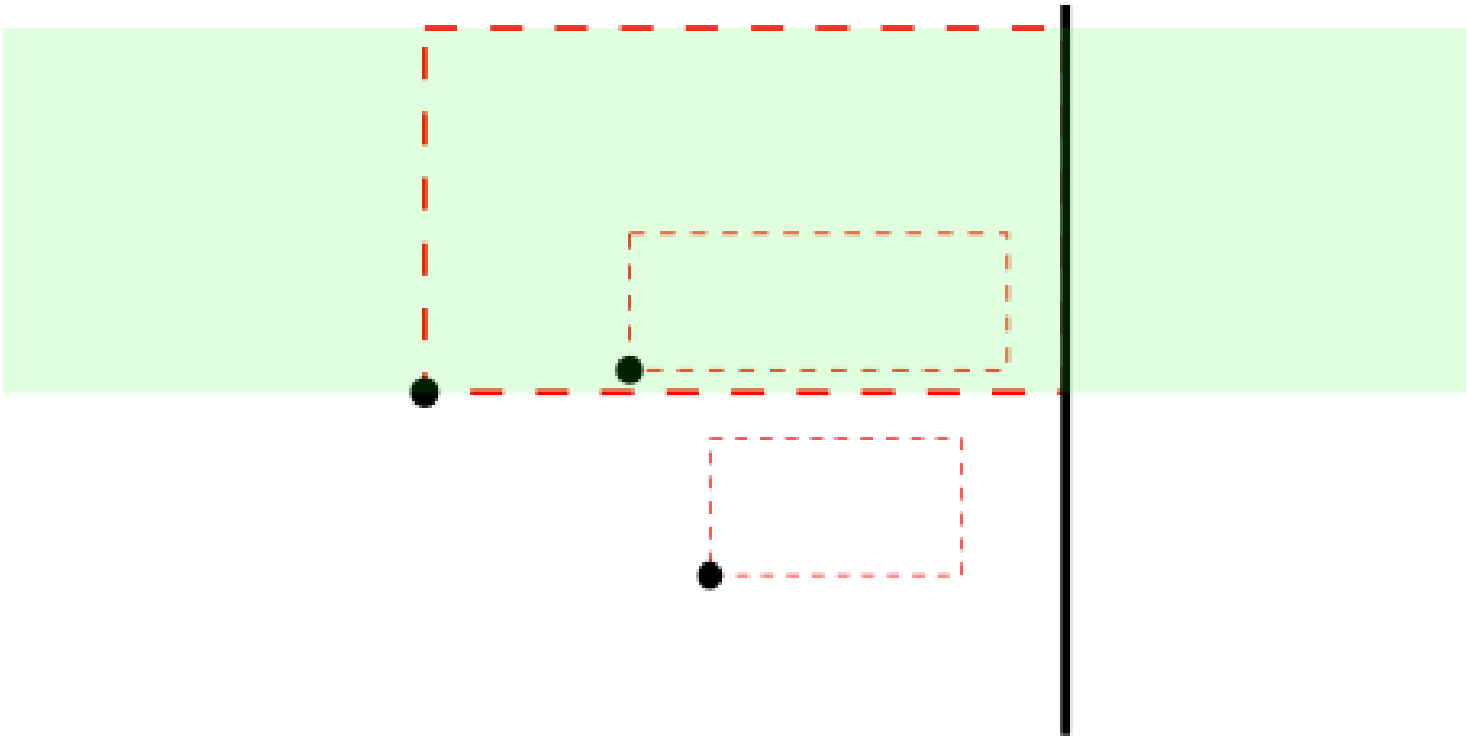












Вопросы?

Задача «Урюк»

Задача «Урюк»

Задача «Урюк»

Задача «Урюк»

- Идея задачи — Сергей Волчёнков
- Подготовка тестов — Юрий Петров
- Разбор задачи — Юрий Петров

Математическая формулировка

Входные данные:

- Количество монет n
- Стоимость взвешивания при равенстве a
- Стоимость взвешивания при неравенстве b

Результат:

- Минимальная стоимость, необходимая для определения более лёгкой монеты

Решение

Воспользуемся методом динамического программирования

- Параметр: количество гарантированно настоящих монет k
- Вычисляемая функция: минимальная стоимость f_k
- База: при $k = n - 1$ фальшивая монета однозначно определена

Решение

Воспользуемся методом динамического программирования

- Параметр: количество гарантированно настоящих монет k
- Вычисляемая функция: минимальная стоимость f_k
- База: при $k = n - 1$ фальшивая монета однозначно определена

Решение

Воспользуемся методом динамического программирования

- Параметр: количество гарантированно настоящих монет k
- Вычисляемая функция: минимальная стоимость f_k
- База: при $k = n - 1$ фальшивая монета однозначно определена

Переход

Переберём очередное взвешивание

- На первую чашу весов положим x настоящих и y новых
- На вторую чашу весов положим z настоящих и w новых
- Остаётся незадействованными $t = (n - k) - (y + w)$ монет
- Взвешивание имеет смысл, если $x + y = z + w$

Переход

Переберём очередное взвешивание

- На первую чашу весов положим x настоящих и y новых
- На вторую чашу весов положим z настоящих и w новых
- Остаётся незадействованными $t = (n - k) - (y + w)$ монет
- Взвешивание имеет смысл, если $x + y = z + w$

Переход

Переберём очередное взвешивание

- На первую чашу весов положим x настоящих и y новых
- На вторую чашу весов положим z настоящих и w новых
- Остаётся незадействованными $t = (n - k) - (y + w)$ монет
- Взвешивание имеет смысл, если $x + y = z + w$

Переход

Переберём очередное взвешивание

- На первую чашу весов положим x настоящих и y новых
- На вторую чашу весов положим z настоящих и w новых
- Остаётся незадействованными $t = (n - k) - (y + w)$ монет
- Взвешивание имеет смысл, если $x + y = z + w$

Переход

Результат взвешивания:

- Если чаши равны, платим a и переходим в состояние $k + y + w$, все монеты на весах настоящие
- Если первая чаша легче, платим b и переходим в состояние $k + w + t$
- Если вторая чаша легче, платим b и переходим в состояние $k + y + t$

Переход

Результат взвешивания:

- Если чаши равны, платим a и переходим в состояние $k + y + w$, все монеты на весах настоящие
- Если первая чаша легче, платим b и переходим в состояние $k + w + t$
- Если вторая чаша легче, платим b и переходим в состояние $k + y + t$

Переход

Результат взвешивания:

- Если чаши равны, платим a и переходим в состояние $k + y + w$, все монеты на весах настоящие
- Если первая чаша легче, платим b и переходим в состояние $k + w + t$
- Если вторая чаша легче, платим b и переходим в состояние $k + y + t$

Переход

Значение функции:

- Из трёх исходов нужно выбрать худший, потому что нам он не известен
- Из значений для каждого взвешивания нужно выбрать минимальное

Переход

Значение функции:

- Из трёх исходов нужно выбрать худший, потому что нам он не известен
- Из значений для каждого взвешивания нужно выбрать минимальное

Оценка времение работы

Оценка времени работы:

- Всего $O(n)$ состояний
- Из каждого состояния $O(n^3)$ переходов
- Итоговое время есть $O(n^4)$

Оценка времение работы

Оценка времени работы:

- Всего $O(n)$ состояний
- Из каждого состояния $O(n^3)$ переходов
- Итоговое время есть $O(n^4)$

Оценка времение работы

Оценка времени работы:

- Всего $O(n)$ состояний
- Из каждого состояния $O(n^3)$ переходов
- Итоговое время есть $O(n^4)$

Оптимизации

Оптимизации:

- Заметим, что одинаковое количество настоящих монет можно убрать с обеих чаш, поэтому $x = 0$ или $z = 0$
- Количество переходов снижается до $O(n^2)$
- Такое решение получало 40 баллов

Оптимизации

Оптимизации:

- Заметим, что одинаковое количество настоящих монет можно убрать с обеих чаш, поэтому $x = 0$ или $z = 0$
- Количество переходов снижается до $O(n^2)$
- Такое решение получало 40 баллов

Оптимизации

Оптимизации:

- Заметим, что одинаковое количество настоящих монет можно убрать с обеих чаш, поэтому $x = 0$ или $z = 0$
- Количество переходов снижается до $O(n^2)$
- Такое решение получало 40 баллов

Оптимизации

Оптимизации:

- Заметим, что f_k монотонно неубывает
- Значит, из y и w в случае неравенства всегда будет происходить переход в большую
- Значит, выгодно делать $|y - w| \leqslant 1$
- Количество переходов снижается до $O(n)$
- Такое решение получало 70 баллов

Оптимизации

Оптимизации:

- Заметим, что f_k монотонно неубывает
- Значит, из y и w в случае неравенства всегда будет происходить переход в большую
- Значит, выгодно делать $|y - w| \leqslant 1$
- Количество переходов снижается до $O(n)$
- Такое решение получало 70 баллов

Оптимизации

Оптимизации:

- Заметим, что f_k монотонно неубывает
- Значит, из y и w в случае неравенства всегда будет происходить переход в большую
- Значит, выгодно делать $|y - w| \leqslant 1$
- Количество переходов снижается до $O(n)$
- Такое решение получало 70 баллов

Оптимизации

Оптимизации:

- Заметим, что f_k монотонно неубывает
- Значит, из y и w в случае неравенства всегда будет происходить переход в большую
- Значит, выгодно делать $|y - w| \leqslant 1$
- Количество переходов снижается до $O(n)$
- Такое решение получало 70 баллов

Оптимизации

Оптимизации:

- Заметим, что f_k монотонно неубывает
- Значит, из y и w в случае неравенства всегда будет происходить переход в большую
- Значит, выгодно делать $|y - w| \leqslant 1$
- Количество переходов снижается до $O(n)$
- Такое решение получало 70 баллов

Оптимизации

Оптимизации:

- Вспомним, что f_k монотонно неубывает
- А ответ — это максимум из $b + f_x$ и $a + f_{k-2x}$ для некоторого $1 \leqslant x \leqslant k$
- Значит, при переходе от k к $k+1$ оптимальное разбиение по чашам может сдвинуться только вправо
- Общее количество переходов из всех состояний становится $O(n)$
- Такое решение получало 100 баллов

Оптимизации

Оптимизации:

- Вспомним, что f_k монотонно неубывает
- А ответ — это максимум из $b + f_x$ и $a + f_{k-2x}$ для некоторого $1 \leqslant x \leqslant k$
- Значит, при переходе от k к $k+1$ оптимальное разбиение по чашам может сдвинуться только вправо
- Общее количество переходов из всех состояний становится $O(n)$
- Такое решение получало 100 баллов

Оптимизации

Оптимизации:

- Вспомним, что f_k монотонно неубывает
- А ответ — это максимум из $b + f_x$ и $a + f_{k-2x}$ для некоторого $1 \leqslant x \leqslant k$
- Значит, при переходе от k к $k+1$ оптимальное разбиение по чашам может сдвинуться только вправо
- Общее количество переходов из всех состояний становится $O(n)$
- Такое решение получало 100 баллов

Оптимизации

Оптимизации:

- Вспомним, что f_k монотонно неубывает
- А ответ — это максимум из $b + f_x$ и $a + f_{k-2x}$ для некоторого $1 \leqslant x \leqslant k$
- Значит, при переходе от k к $k + 1$ оптимальное разбиение по чашам может сдвинуться только вправо
- Общее количество переходов из всех состояний становится $O(n)$
- Такое решение получало 100 баллов

Оптимизации

Оптимизации:

- Вспомним, что f_k монотонно неубывает
- А ответ — это максимум из $b + f_x$ и $a + f_{k-2x}$ для некоторого $1 \leqslant x \leqslant k$
- Значит, при переходе от k к $k + 1$ оптимальное разбиение по чашам может сдвинуться только вправо
- Общее количество переходов из всех состояний становится $O(n)$
- Такое решение получало 100 баллов

Вопросы?

Задача «Древний календарь»

Задача «Древний календарь»

Задача «Древний календарь»

Задача «Древний календарь»

- Идея задачи — Владимир Гуровиц
- Подготовка тестов — Виталий Гольдштейн, Александр Тимин
- Разбор задачи — Елена Андреева

$1 \leqslant N \leqslant 1000$, $1 \leqslant M \leqslant 100$. В каждом столбце таблицы есть, по крайней мере, одна цифра.

- Найдем цифру в последнем столбце и восстановим его, циклически расставляя цифры по порядку до начала и до конца столбца:

```
...3 4 5 6 7 8 9  0 1 2 3...
    * * * * 6 * * 17 * 7 *
```

попутно добавляя рядом с нулем последнего столбца, например 10, в предпоследний столбец или просто 10 вместо $*$

- Во втором и каждом последующем столбце поступаем следующим образом:
- Проходим столбец в двух направлениях.
- Сверху вниз
- Если текущий элемент *, а предыдущий нет, то текущий элемент равен предыдущему
- Если текущий элемент равен 10, то он равен предыдущему $+1$
- Аналогично проходим снизу вверх, заменяя оставшиеся звездочки и 10 на предыдущее или на 1 меньшее значение.
- Вычитаем 10 из элементов больше 9

$1 \leqslant N \leqslant 1000$, $1 \leqslant M \leqslant 100$. Хотя бы один столбец содержит только символы «*»

- **Частичное решение**
- Если в столбце есть хотя бы одна цифра, то поступаем также, как и раньше
- Если цифры нет, то переберем первую цифру
- Решение быстро работает, когда пустых столбцов мало и когда пустых столбцов много
- От 60 до 70 баллов

$1 \leqslant N \leqslant 100\,000$, $1 \leqslant M \leqslant 100\,000$

- Динамическое программирование
- Состояние — [столбец] [сдвиг]
- Сдвиг — количество цифр в первом (неполном) блоке
- Второе число меняется от 0 до $\min\{N, 10j\} - 1$, если столбцы нумеровать справа с нуля
- Для каждого состояния храним, допустимо ли оно, с какой цифры может начинаться, из какого состояния $(j-1)$-го столбца мы в него попали

Пересчет состояния

- Если хотя бы одна цифра в столбце есть, то по величине сдвига S мы однозначно проверяем допустимость состояния и первую цифру D. Это состояние делает допустимым ровно одно состояние в столбце $(j+1)$:
  $[j+1]\,[S+(9-D)\cdot(10j)]$
  (если $S+(9-D)*(10j) \leqslant N$, иначе допустимо состояние $[j+1]\,[0]$)
- Если цифр нет, то пока первую цифру считаем любой, а допустимыми все соответствующие состояния столбца $(j+1)$ для разных D

Проверка состояния

- Необходимо выяснить, не противоречат ли уже имеющиеся цифры текущему сдвигу
- Для этого для каждой цифры предпосчитаем на префиксе, сколько раз она встречается. Далее за одно действие мы сможем проверять, сколько определенных цифр на интервалах
- $0 \dots S-1, S \dots S+10j-1, S+10j \dots S+2 \cdot 10j-1$, до N
- различных цифр на интервале быть не должно
- Однозначно определяем первую цифру по первому непустому интервалу и еще раз проверяем, что на каждом интервале находится нужная цифра за количество интервалов

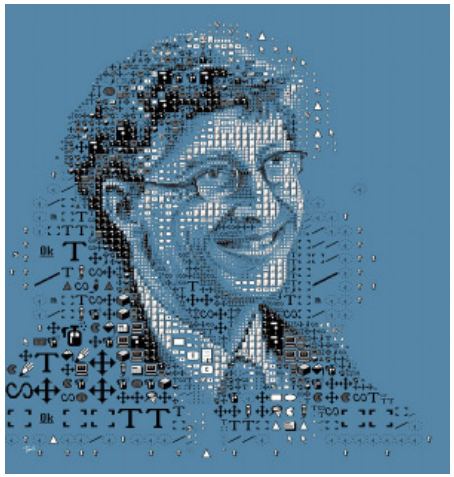
Восстанавливаем ответ с конца

- В левом столбце нам подходит любой допустимый сдвиг
- По нему мы определяем сдвиг и, соответственно, цифру следующего столбца и т. д.
- Общее число действий в решении не превосходит $O(NM)$, причем около $10NM$ в худшем случае, так как:
- Проверка состояния производится за $\frac{10N}{10^j}$, сумма по всем состояниям $10NM$

Вопросы?

Задача «Мозаика»

Задача «Мозаика»



Задача «Мозаика»

Задача «Мозаика»

- Идея задачи — Сергей Поромов и Елена Андреева
- Подготовка тестов — Юрий Петров
- Разбор задачи — Сергей Поромов

Простое решение

- Для каждого запроса отдельно на каждом отрезке перебираем все пары элементов
- Нашли ответ — выводим, нет — выводим «0 0»
- Работает за $O(n^3)$
- Получаем 20 баллов

Добавляем предподсчет

- Для каждого элемента заранее найдем ближайший следующий, который негармоничен с ним
- При очередном запросе для каждого элемента на отрезке смотрим, попадает ли следующий негармоничный с ним в отрезок
- Работает за $O(n^2)$
- Получаем 70 баллов

Ответ на запрос за $O(\log)$

- При каждом запросе находим на отрезке элемент, у котором минимальный номер следующего негармоничного с ним
- Это можно делать деревом отрезков
- $O(n \log n)$
- 100 баллов

Можно быстрее и проще

- Для каждого элемента заранее найдем следующий, отличающийся от него по длине ($nexta_i$) и по ширине ($nextb_i$)
- Для каждого запроса на отрезке с L по R достаточно взять три элемента: L, $nexta_L$ и $nextb_L$
- Если $nexta_L \leqslant R$ и $nextb_L \leqslant R$, то из этих трёх элементов найдётся подходящая пара, иначе — такой пары нет
- Работает за $O(n)$

Решения со случайным поиском

- На каждом отрезке несколько раз случайно выбирается пара элементов и проверяется
- Если за несколько запросов не нашли — считаем, что подходящей пары нет
- Получаем 20 баллов
- Если сжать подряд идущие одинаковые, то 50 баллов

Вопросы?

Задача «Театр начинается с актеров»

Задача «Театр начинается с актеров»



«Полонез», театр Моссовета

Задача «Театр начинается с актеров»

Задача «Театр начинается с актеров»

- Идея задачи — Виталий Гольдштейн
- Подготовка тестов — Виталий Гольдштейн
- Разбор задачи — Виталий Гольдштейн

Различать попарно

- Найдем для каждой пары актеров номер действия, когда мы сможем их различить
- Ответом для актера будет номер действия, когда мы сможем отличить его от каждого актера (то есть максимум)

Обновляем после действия

- Изначально для всех пар ответ бесконечность
- Пусть в очередном действие участвуют актеры $P = \{p_1, p_2, \ldots, p_k\}$
- Тогда все актеры p из P и q из P стали различимы. Переберем все такие пары актеров и отметим, что мы их можем отличить после текущего действия (если они не были отличимы ранее)

Реализация

- Отметим в массиве про каждого актера, участвовал ли он в действие. Переберем все пары актеров (i, j) и в случае различия отметок в матрице установим $A_{i,j}$ текущее действие
- Такое решение работает за $O(N^2M)$. Это решение первой подзадачи.
- Перебираем актеров, участвовавших в действии i, и всех остальных актеров j. Если актер j не участвовал в действие, то в матрице установим $A_{i,j}$ текущее действие
- Такое решение работает за $O(N \cdot \sum K_i)$
- ~~Это решение второй подзадачи.~~ Память $O(N^2)$

Проверка для конкретной пары

- Найдем первое действие, в которых два актера различаются (один участвует, а другой нет).
- Первый участвовал в сценах $P = \{p_1, p_2, \ldots, p_k\}$
- Второй участвовал в сценах $Q = \{q_1, q_2, \ldots, q_t\}$
- Найдем такое первое такое i, что q_i не равно p_i
- Меньшее из этих двух чисел и будет искомым действием.
- Время определение за $O(k + t)$

Подзадача 2

- Для каждого актера переберем остальных актеров и решим задачу только для этой пары.
- Для этого актера выберем максимум.
- Время работы $O(N \cdot \sum k_i)$
- Памяти требуется $O(\sum k_i)$

Разделение на классы эквивалентности

- Изначально все актеры не различимы и находятся в одном классе эквивалентности.
- Актеры, участвовавшие в очередном действии, должны быть выделены из своего класса эквивалентности в отдельный.
- Если при разделении в одном (или обоих) из классов остается один актер, то после это действия его (или их) можно отличить после этого действия.

Реализация

- Для каждого актера будем хранить номер его класса эквивалентности.
- Для каждого класса эквивалентности будем хранить количество актеров и номер действия, после которого произошло последнее изменение.
- После очередного действия всех участников отсортируем по номеру класса эквивалентности.
- Всех актеров из одного класса эквивалентности выделим в отдельный (кроме случая, когда все актеры класса эквивалентности участвовали в действии)

Определение ответа

- Для каждого актера выведем номер действия, после которого последний раз изменился его класс эквивалентности.
- Если класс эквивалентности состоит из более, чем одного актера, то ответ для этого актера 0.

Время работы

- Сортировка участников по классам эквивалентности $O(C \cdot \log C)$, где $= \sum k_i$. Если сортировать подсчетом, то $O(C)$.
- Изменение информации для актеров $O(C)$.
- Изменение информации о классе эквивалентности $O(C)$
- Нахождение ответа $O(N)$.

Вопросы?

Задача «Ёжик в тумане»

Задача «Ёжик в тумане»

Задача «Ёжик в тумане»

Задача «Ёжик в тумане»

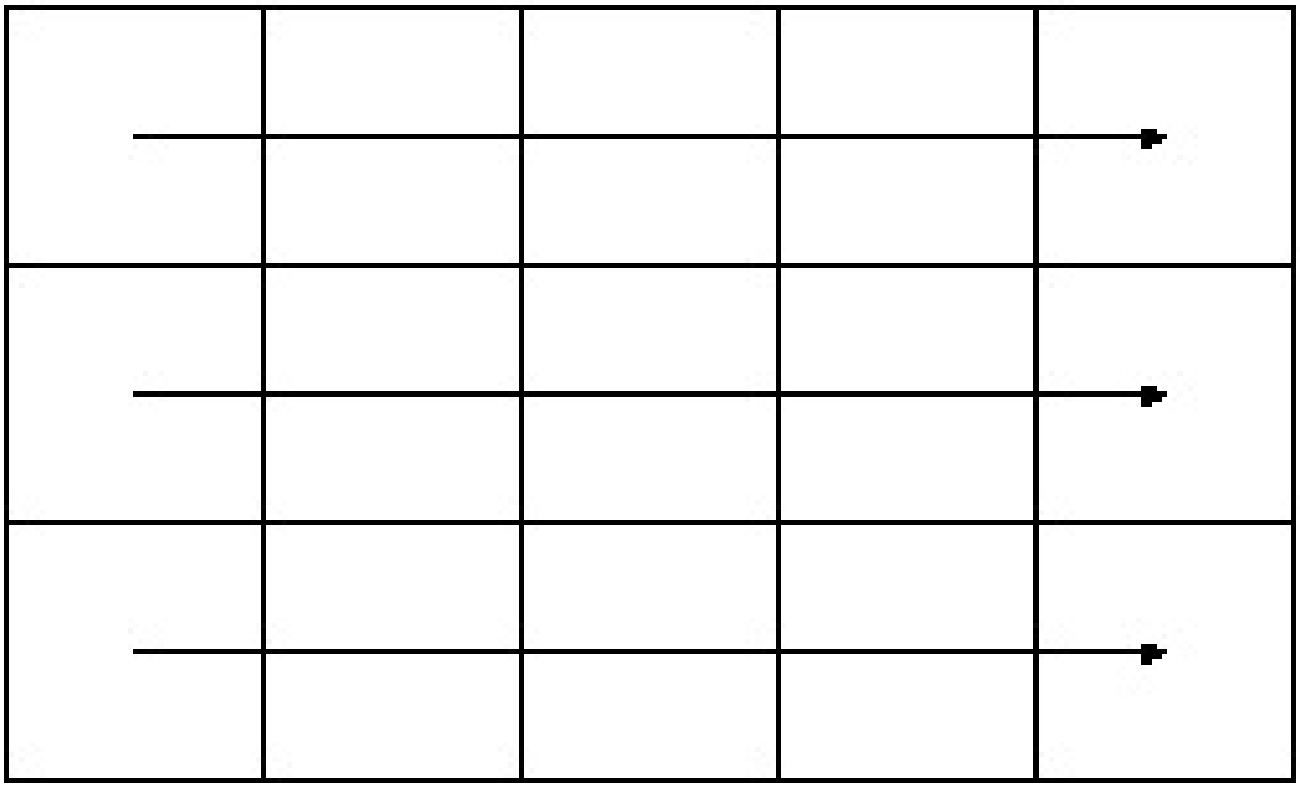
- Идея задачи — Андрей Шестимеров
- Подготовка тестов — Александр Тимин
- Разбор задачи — Александр Тимин

Основная идея

- Лошадь всегда можно поймать
- Переберем где стоит Лошадь и поймаем её
  - Это всегда получится, так как мы движемся быстрее чем она
  - Если мы с Лошадью стоим на одной горизонтали или вертикали, то долго она от нас убегать не сможет (упрется в границу)
  - В противном случае мы уменьшаем расстояние до неё, двигаясь по диагонали.

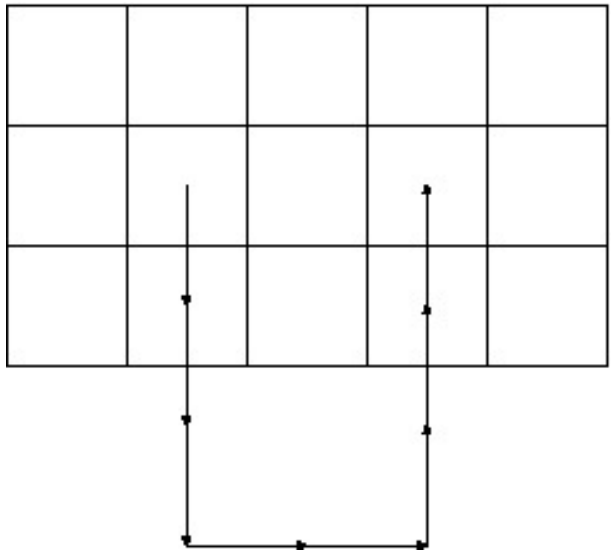
Решение на 35 баллов

Будем ловить лошадей в таком порядке:



Оптимизация

- Заметим, что для некоторых начальных позиций Лошадь выходит за границу поля а потом возвращается
- Ловить таких Лошадей не надо

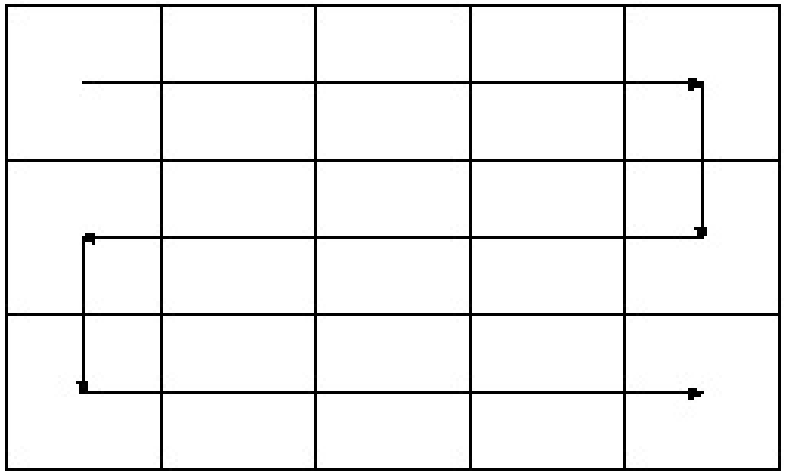


Оптимизация

- Множество Лошадей, которые не выходили за границу — прямоугольник
- Он сдвигается с каждым ходом Лошади
- Будем его поддерживать и пересчитывать на каждом ходу
- Данное решение набирает 45 баллов

Решение на 80 баллов

Будем обходить клетки в более оптимальном порядке



Решение на 95 баллов

- Если к предыдущему решению, обходящему все клетки «змейкой» применить ранее описанную оптимизацию, то оно набирает 95 баллов.

Решение на 100 баллов

- Заметим, что можно начинать обходить поле не из угла $(1, 1)$ а из ближайшего к начальному положению Ежа

Вопросы?

Задача «Ордынское войско»

Задача «Ордынское войско»

Задача «Ордынское войско»

Задача «Ордынское войско»

- Идея задачи — Александр Калужин
- Подготовка тестов — Сергей Мельников
- Разбор задачи — Юрий Петров

Математическая формулировка

Входные данные:

- Порядок перестановки n
- Последовательность $1 \leqslant a_1 < a_2 < \ldots < a_k \leqslant n$

Результат:

- Количество перестановок p порядка n, таких что a является наибольшей возрастающей подпоследовательностью (НВП) p

Перебор

Переберём все перестановки порядка n одним из двух способов:

- Переходя от перестановки к следующей лексикографически
- Или рекурсивно

Перебор

Переберём все перестановки порядка n одним из двух способов:

- Переходя от перестановки к следующей лексикографически
- Или рекурсивно

Проверка

Проверим, что перестановка p подходит:

- Найдём НВП
- Проверим, что её длина равна k
- Проверим, что a является подпоследовательностью p

Проверка

Проверим, что перестановка p подходит:

- Найдём НВП
- Проверим, что её длина равна k
- Проверим, что a является подпоследовательностью p

Проверка

Проверим, что перестановка p подходит:

- Найдём НВП
- Проверим, что её длина равна k
- Проверим, что a является подпоследовательностью p

Подзадача: поиск НВП

Как найти НВП, первый вариант:

- Использовать метод динамического программирования
- Вычисляемая функция f_i — длина НВП, заканчивающейся в позиции i
- Переход: $f_i = 1 + \max_{j<i \text{ и } p_j<p_i} f_j$
- Сложность: $O(n^2)$ операций

Подзадача: поиск НВП

Как найти НВП, первый вариант:

- Использовать метод динамического программирования
- Вычисляемая функция f_i — длина НВП, заканчивающейся в позиции i
- Переход: $f_i = 1 + \max_{j<i \text{ и } p_j<p_i} f_j$
- Сложность: $O(n^2)$ операций

Подзадача: поиск НВП

Как найти НВП, первый вариант:

- Использовать метод динамического программирования
- Вычисляемая функция f_i — длина НВП, заканчивающейся в позиции i
- Переход: $f_i = 1 + \max_{j<i \text{ и } p_j<p_i} f_j$
- Сложность: $O(n^2)$ операций

Подзадача: поиск НВП

Как найти НВП, первый вариант:

- Использовать метод динамического программирования
- Вычисляемая функция f_i — длина НВП, заканчивающейся в позиции i
- Переход: $f_i = 1 + \max_{j<i \text{ и } p_j<p_i} f_j$
- Сложность: $O(n^2)$ операций

Подзадача: поиск НВП

Как найти НВП, второй вариант:

- Использовать метод динамического программирования
- Вычисляемая функция f_l — минимальное значение, которым может заканчиваться ВП длины l
- Исходно: $f_l = +\infty$ для всех $1 \leqslant l \leqslant n$ и $f_0 = -\infty$
- Переход: добавим элемент p_i
- Найдём максимальную длину l, такую что $f_l < p_i$
- Обновим значение $f_{l+1} = \min\{f_{l+1}, p_i\}$
- Сложность: $O(n^2)$ или $O(n \log n)$ с использованием двоичного поиска

Подзадача: поиск НВП

Как найти НВП, второй вариант:

- Использовать метод динамического программирования
- Вычисляемая функция f_l — минимальное значение, которым может заканчиваться ВП длины l
- Исходно: $f_l = +\infty$ для всех $1 \leqslant l \leqslant n$ и $f_0 = -\infty$
- Переход: добавим элемент p_i
- Найдём максимальную длину l, такую что $f_l < p_i$
- Обновим значение $f_{l+1} = \min\{f_{l+1}, p_i\}$
- Сложность: $O(n^2)$ или $O(n \log n)$ с использованием двоичного поиска

Подзадача: поиск НВП

Как найти НВП, второй вариант:

- Использовать метод динамического программирования
- Вычисляемая функция f_l — минимальное значение, которым может заканчиваться ВП длины l
- Исходно: $f_l = +\infty$ для всех $1 \leqslant l \leqslant n$ и $f_0 = -\infty$
- Переход: добавим элемент p_i
- Найдём максимальную длину l, такую что $f_l < p_i$
- Обновим значение $f_{l+1} = \min\{f_{l+1}, p_i\}$
- Сложность: $O(n^2)$ или $O(n \log n)$ с использованием двоичного поиска

Подзадача: поиск НВП

Как найти НВП, второй вариант:

- Использовать метод динамического программирования
- Вычисляемая функция f_l — минимальное значение, которым может заканчиваться ВП длины l
- Исходно: $f_l = +\infty$ для всех $1 \leqslant l \leqslant n$ и $f_0 = -\infty$
- Переход: добавим элемент p_i
- Найдём максимальную длину l, такую что $f_l < p_i$
- Обновим значение $f_{l+1} = \min\{f_{l+1}, p_i\}$
- Сложность: $O(n^2)$ или $O(n \log n)$ с использованием двоичного поиска

Подзадача: поиск НВП

Как найти НВП, второй вариант:

- Использовать метод динамического программирования
- Вычисляемая функция f_l — минимальное значение, которым может заканчиваться ВП длины l
- Исходно: $f_l = +\infty$ для всех $1 \leqslant l \leqslant n$ и $f_0 = -\infty$
- Переход: добавим элемент p_i
- Найдём максимальную длину l, такую что $f_l < p_i$
- Обновим значение $f_{l+1} = \min\{f_{l+1}, p_i\}$
- Сложность: $O(n^2)$ или $O(n \log n)$ с использованием двоичного поиска

Подзадача: поиск НВП

Как найти НВП, второй вариант:

- Использовать метод динамического программирования
- Вычисляемая функция f_l — минимальное значение, которым может заканчиваться ВП длины l
- Исходно: $f_l = +\infty$ для всех $1 \leqslant l \leqslant n$ и $f_0 = -\infty$
- Переход: добавим элемент p_i
- Найдём максимальную длину l, такую что $f_l < p_i$
- Обновим значение $f_{l+1} = \min\{f_{l+1}, p_i\}$
- Сложность: $O(n^2)$ или $O(n \log n)$ с использованием двоичного поиска

Подзадача: поиск НВП

Как найти НВП, второй вариант:

- Использовать метод динамического программирования
- Вычисляемая функция f_l — минимальное значение, которым может заканчиваться ВП длины l
- Исходно: $f_l = +\infty$ для всех $1 \leqslant l \leqslant n$ и $f_0 = -\infty$
- Переход: добавим элемент p_i
- Найдём максимальную длину l, такую что $f_l < p_i$
- Обновим значение $f_{l+1} = \min\{f_{l+1}, p_i\}$
- Сложность: $O(n^2)$ или $O(n \log n)$ с использованием двоичного поиска

Оценка сложности перебора

Время работы перебора

- Перебирается $n!$ перестановок
- Проверка требует $O(n^2)$ или $O(n \log n)$ операций
- Такие решения работают для $n \leqslant 8$ и набирают от 40 баллов

Оценка сложности перебора

Время работы перебора

- Перебирается $n!$ перестановок
- Проверка требует $O(n^2)$ или $O(n \log n)$ операций
- Такие решения работают для $n \leqslant 8$ и набирают от 40 баллов

Оценка сложности перебора

Время работы перебора

- Перебирается $n!$ перестановок
- Проверка требует $O(n^2)$ или $O(n \log n)$ операций
- Такие решения работают для $n \leqslant 8$ и набирают от 40 баллов

Оптимизации перебора

Перебор можно улучшить

- После выбора очередного элемента проверять, что a может являться подпоследовательностью
- Количество перебираемых перестановок сокращается до $(n - k)!$
- После выбора очередного элемента проверять, что длина НВП не превзошла k
- Отсечёнными окажутся многие ветви при небольших k
- Можно проверить, что такое решение работает для $n \leqslant 10$ и набирает 50 баллов

Оптимизации перебора

Перебор можно улучшить

- После выбора очередного элемента проверять, что a может являться подпоследовательностью
- Количество перебираемых перестановок сокращается до $(n - k)!$
- После выбора очередного элемента проверять, что длина НВП не превзошла k
- Отсечёнными окажутся многие ветви при небольших k
- Можно проверить, что такое решение работает для $n \leqslant 10$ и набирает 50 баллов

Оптимизации перебора

Перебор можно улучшить

- После выбора очередного элемента проверять, что a может являться подпоследовательностью
- Количество перебираемых перестановок сокращается до $(n - k)!$
- После выбора очередного элемента проверять, что длина НВП не превзошла k
- Отсечёнными окажутся многие ветви при небольших k
- Можно проверить, что такое решение работает для $n \leqslant 10$ и набирает 50 баллов

Оптимизации перебора

Перебор можно улучшить

- После выбора очередного элемента проверять, что a может являться подпоследовательностью
- Количество перебираемых перестановок сокращается до $(n - k)!$
- После выбора очередного элемента проверять, что длина НВП не превзошла k
- Отсечёнными окажутся многие ветви при небольших k
- Можно проверить, что такое решение работает для $n \leqslant 10$ и набирает 50 баллов

Оптимизации перебора

Перебор можно улучшить

- После выбора очередного элемента проверять, что a может являться подпоследовательностью
- Количество перебираемых перестановок сокращается до $(n - k)!$
- После выбора очередного элемента проверять, что длина НВП не превзошла k
- Отсечёнными окажутся многие ветви при небольших k
- Можно проверить, что такое решение работает для $n \leqslant 10$ и набирает 50 баллов

Правильное решение

- Воспользуемся методом динамического программирования
- Параметры:
  - Множество x уже поставленных элементов
  - Множество y элементов, являющихся минимальными конечными для ВП каждой длины от 1 до k
- Вычисляемая функция — количество способов построить префикс перестановки

Правильное решение

- Воспользуемся методом динамического программирования
- Параметры:
  - Множество x уже поставленных элементов
  - Множество y элементов, являющихся минимальными конечными для ВП каждой длины от 1 до k
- Вычисляемая функция — количество способов построить префикс перестановки

Правильное решение

- Воспользуемся методом динамического программирования
- Параметры:
  - Множество x уже поставленных элементов
  - Множество y элементов, являющихся минимальными конечными для ВП каждой длины от 1 до k
- Вычисляемая функция — количество способов построить префикс перестановки

Правильное решение

- Воспользуемся методом динамического программирования
- Параметры:
  - Множество x уже поставленных элементов
  - Множество y элементов, являющихся минимальными конечными для ВП каждой длины от 1 до k
- Вычисляемая функция — количество способов построить префикс перестановки

Правильное решение

- Воспользуемся методом динамического программирования
- Параметры:
  - Множество x уже поставленных элементов
  - Множество y элементов, являющихся минимальными конечными для ВП каждой длины от 1 до k
- Вычисляемая функция — количество способов построить префикс перестановки

Переход

- Переберём элемент, который поставим на очередное место
- Найдём длину НВП, заканчивающейся в нём, как во втором способе поиска НВП
- Если этот элемент лежит в a, проверим, что:
  - Длина НВП, заканчивающейся в нём, должна быть равна его позиции в a
  - Все предыдущие элементы a уже размещены
- Определим множества, в которые совершается переход: если элемент улучшает текущую таблицу НВП, добавим его в y

Переход

- Переберём элемент, который поставим на очередное место
- Найдём длину НВП, заканчивающейся в нём, как во втором способе поиска НВП
- Если этот элемент лежит в a, проверим, что:
  - Длина НВП, заканчивающейся в нём, должна быть равна его позиции в a
  - Все предыдущие элементы a уже размещены
- Определим множества, в которые совершается переход: если элемент улучшает текущую таблицу НВП, добавим его в y

Переход

- Переберём элемент, который поставим на очередное место
- Найдём длину НВП, заканчивающейся в нём, как во втором способе поиска НВП
- Если этот элемент лежит в a, проверим, что:
  - Длина НВП, заканчивающейся в нём, должна быть равна его позиции в a
  - Все предыдущие элементы a уже размещены
- Определим множества, в которые совершается переход: если элемент улучшает текущую таблицу НВП, добавим его в y

Переход

- Переберём элемент, который поставим на очередное место
- Найдём длину НВП, заканчивающейся в нём, как во втором способе поиска НВП
- Если этот элемент лежит в a, проверим, что:
  - Длина НВП, заканчивающейся в нём, должна быть равна его позиции в a
  - Все предыдущие элементы a уже размещены
- Определим множества, в которые совершается переход: если элемент улучшает текущую таблицу НВП, добавим его в y

Переход

- Переберём элемент, который поставим на очередное место
- Найдём длину НВП, заканчивающейся в нём, как во втором способе поиска НВП
- Если этот элемент лежит в a, проверим, что:
  - Длина НВП, заканчивающейся в нём, должна быть равна его позиции в a
  - Все предыдущие элементы a уже размещены
- Определим множества, в которые совершается переход: если элемент улучшает текущую таблицу НВП, добавим его в y

Переход

- Переберём элемент, который поставим на очередное место
- Найдём длину НВП, заканчивающейся в нём, как во втором способе поиска НВП
- Если этот элемент лежит в a, проверим, что:
  - Длина НВП, заканчивающейся в нём, должна быть равна его позиции в a
  - Все предыдущие элементы a уже размещены
- Определим множества, в которые совершается переход: если элемент улучшает текущую таблицу НВП, добавим его в y

Оценка сложности

- Каждый элемент либо не лежит в x, либо лежит в x и не лежит в y, либо лежит в обоих множествах
- Три варианта для каждого элемента дают 3^n состояний
- Каждое состояние требует $O(n)$ действий на перебор очередного элемента
- Пересчёт перехода можно осуществить за $O(n)$ или $O(1)$ действий
- Итоговая сложность $O(n \cdot 3^n)$ или $O(n^2 \cdot 3^n)$, решение набирало от 60 до 90 баллов

Оценка сложности

- Каждый элемент либо не лежит в x, либо лежит в x и не лежит в y, либо лежит в обоих множествах
- Три варианта для каждого элемента дают 3^n состояний
- Каждое состояние требует $O(n)$ действий на перебор очередного элемента
- Пересчёт перехода можно осуществить за $O(n)$ или $O(1)$ действий
- Итоговая сложность $O(n \cdot 3^n)$ или $O(n^2 \cdot 3^n)$, решение набирало от 60 до 90 баллов

Оценка сложности

- Каждый элемент либо не лежит в x, либо лежит в x и не лежит в y, либо лежит в обоих множествах
- Три варианта для каждого элемента дают 3^n состояний
- Каждое состояние требует $O(n)$ действий на перебор очередного элемента
- Пересчёт перехода можно осуществить за $O(n)$ или $O(1)$ действий
- Итоговая сложность $O(n \cdot 3^n)$ или $O(n^2 \cdot 3^n)$, решение набирало от 60 до 90 баллов

Оценка сложности

- Каждый элемент либо не лежит в x, либо лежит в x и не лежит в y, либо лежит в обоих множествах
- Три варианта для каждого элемента дают 3^n состояний
- Каждое состояние требует $O(n)$ действий на перебор очередного элемента
- Пересчёт перехода можно осуществить за $O(n)$ или $O(1)$ действий
- Итоговая сложность $O(n \cdot 3^n)$ или $O(n^2 \cdot 3^n)$, решение набирало от 60 до 90 баллов

Оценка сложности

- Каждый элемент либо не лежит в x, либо лежит в x и не лежит в y, либо лежит в обоих множествах
- Три варианта для каждого элемента дают 3^n состояний
- Каждое состояние требует $O(n)$ действий на перебор очередного элемента
- Пересчёт перехода можно осуществить за $O(n)$ или $O(1)$ действий
- Итоговая сложность $O(n \cdot 3^n)$ или $O(n^2 \cdot 3^n)$, решение набирало от 60 до 90 баллов

Оценка памяти

- Заметим, что требуется также $8 \cdot 3^n$ байт памяти для хранения ответов
- Это превышает ограничение для $n = 15$
- Заметим, что далеко не все состояния достижимы
  - Если k велико, то множество x меняется почти однозначно
  - Если k мало, то множество y имеет небольшой размер
- Можно написать программу и узнать, что достижимых состояние не больше 500 000

Оценка памяти

- Заметим, что требуется также $8 \cdot 3^n$ байт памяти для хранения ответов
- Это превышает ограничение для $n = 15$
- Заметим, что далеко не все состояния достижимы
  - Если k велико, то множество x меняется почти однозначно
  - Если k мало, то множество y имеет небольшой размер
- Можно написать программу и узнать, что достижимых состояние не больше 500 000

Оценка памяти

- Заметим, что требуется также $8 \cdot 3^n$ байт памяти для хранения ответов
- Это превышает ограничение для $n = 15$
- Заметим, что далеко не все состояния достижимы
  - Если k велико, то множество x меняется почти однозначно
  - Если k мало, то множество y имеет небольшой размер
- Можно написать программу и узнать, что достижимых состояние не больше 500 000

Оценка памяти

- Заметим, что требуется также $8 \cdot 3^n$ байт памяти для хранения ответов
- Это превышает ограничение для $n = 15$
- Заметим, что далеко не все состояния достижимы
  - Если k велико, то множество x меняется почти однозначно
  - Если k мало, то множество y имеет небольшой размер
- Можно написать программу и узнать, что достижимых состояние не больше 500 000

Оценка памяти

- Заметим, что требуется также $8 \cdot 3^n$ байт памяти для хранения ответов
- Это превышает ограничение для $n = 15$
- Заметим, что далеко не все состояния достижимы
  - Если k велико, то множество x меняется почти однозначно
  - Если k мало, то множество y имеет небольшой размер
- Можно написать программу и узнать, что достижимых состояние не больше 500 000

Оценка памяти

- Заметим, что требуется также $8 \cdot 3^n$ байт памяти для хранения ответов
- Это превышает ограничение для $n = 15$
- Заметим, что далеко не все состояния достижимы
  - Если k велико, то множество x меняется почти однозначно
  - Если k мало, то множество y имеет небольшой размер
- Можно написать программу и узнать, что достижимых состояние не больше 500 000

Оптимизация

- Ответ для состояний можно хранить в хеш-таблице, сбалансированном дереве или аналогичной структуре
- Состояния можно перебирать, обходя граф состояний в ширину
- Такое решение набирало 100 баллов

Оптимизация

- Ответ для состояний можно хранить в хеш-таблице, сбалансированном дереве или аналогичной структуре
- Состояния можно перебирать, обходя граф состояний в ширину
- Такое решение набирало 100 баллов

Оптимизация

- Ответ для состояний можно хранить в хеш-таблице, сбалансированном дереве или аналогичной структуре
- Состояния можно перебирать, обходя граф состояний в ширину
- Такое решение набирало 100 баллов

Вопросы?

EOF