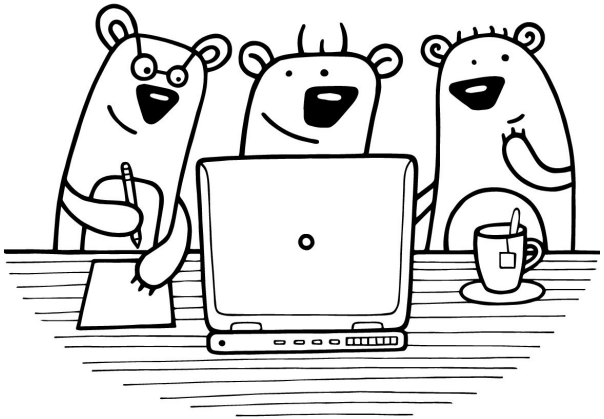


# Индивидуальная олимпиада по информатике и программированию. Заключительный этап.

24 марта 2019 года

# Задача A
# Ослабление флота

Автор задачи:
**Григорий Шовкопляс**
Разработка задачи:
**Григорий Шовкопляс**

# Постановка задачи

- Есть массив из n чисел
- На каждой итерации из массива удаляется медиана
- Нужно вывести элементы в порядке удаления

# Решение на 47 баллов

- Напишем ровно то, что просят в условии
- $n$ раз будем искать медиану в массиве, после чего удалять ее, получая новый массив. Как найти медиану?
  - Отсортировать массив за $O(n \log n)$
    - $n/2$-й элемент — медиана
    - Итоговая сложность решения будет $O(n^2 \log n)$
  - Воспользоваться алгоритмом поиска к-ой порядковой статистики, который работает за $O(n)$
    - Реализован в стандартной библиотеке в некоторых языках
    - Итоговая сложность решения будет $O(n^2)$

# Решение на 100 баллов

- Модифицируем первый способ решения предыдущей подгруппы
  - После того, как мы отсортировали массив и удалим медиану, следующая медиана это соседний элемент от удаленного
  - Достаточно отсортировать массив ровно один раз
  - Методом двух указателей пройдемся по нему и выведем нужные элементы
- Для удобства можно отдельно рассмотреть случай нечетной длины.
- Итоговая сложность данного решения $O(n \log n)$

# Задача B
# Рапорт

Автор задачи:
**Николай Будин**
Разработка задачи:
**Николай Будин**

# Постановка задачи

- Рапорт состоит из двух частей
- Каждая часть — последовательность слов
- Также, есть рулон бумаги ширины `w`
- Нужно разделить рулон вертикальной чертой на две части
- В левой части будет записана первая часть рапорта, а в правой – вторая
- Текст записывается жадно
- Соседние слова на одной строке разделяются пустой клеткой

# Решение на 51 балл

- Переберем положение вертикальной черты, при котором рапорт возможно написать
- Проэмулируем процесс написания слов и вычислим длину рулона для данной позиции вертикальной черты
- Ответом является минимальная длина рулона по всем положениям черты

# Решение на 100 баллов

- При сдвигании вертикальной черты вправо, длина первой части рапорта будет уменьшаться, а длина второй части рапорта будет увеличиваться
- Используя двоичный поиск можно найти две соседних вертикальных черты, что если провести одну из них, левая часть будет длиннее правой, а если вторую, то наоборот
- Среди ответов для этих двух черт обязательно есть оптимальный ответ
- Итоговая асимптотика: $O((n + m) \log w)$

# Задача C
# Декодирование сообщения

Автор задачи:
**Дмитрий Филиппов**
Разработка задачи:
**Дмитрий Филиппов**

# Постановка задачи

- Было сообщение, состоящее только из строчных и заглавных латинских букв
- Каждую его букву заменили на ее ASCII-код, если он трехзначный. Если двузначный, в случайное место кода добавили цифру d
- Вам дано закодированное сообщение s, однако d неизвестно
- Посчитать количество способов восстановить сообщение

# Решение задачи

- Каждый символ исходной строки кодируется в трехзначное число
- Значит s можно разбить на группы из трех последовательных символов, каждая из которых соответствует закодированной букве

# Решение задачи (21 балл)

- Длина исходной строки не превосходит 4
- Переберем все возможные исходные строки
- Для каждой проверим, можно ли ее закодировать в $s$ при $d = 1$
- Всего перебираемых строк $(26 + 26)^4 < 10^7$

# Решение задачи (43 балла)

- Блоки из трех цифр в s *независимы*
- Для каждого блока b найдем все возможные символы c, которые можно при d = 1 закодировать в b
- Полученные числа по всем блокам перемножим

# Решение задачи (75-100 баллов)

- Каким бы ни было число d, из двузначного кода символа нельзя получить число в диапазоне `[100, 122]` (соответствует `[d, z]`)
- Эти символы декодируются однозначно

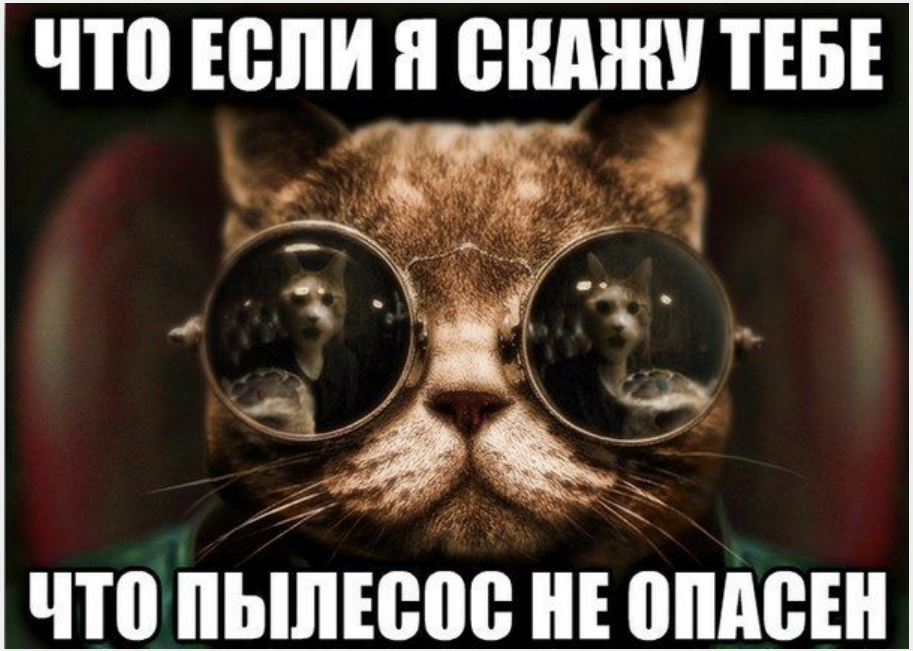
# Решение задачи (75-100 баллов)

- Переберем значение цифры `d`
- Для каждого блока `b` найдем все символы, которые могут кодироваться в `b` с помощью цифры `d`
- Это можно сделать за `O(1)`, попробовав последовательно удалить каждую цифру из блока и проверив, что оставшееся число лежит либо в `[65, 90]`, либо в `[97, 99]`

# Решение задачи (75-100 баллов)

- Полученные в каждом блоке количества возможных символов перемножим по всем блокам
- И сложим по различным значениям d
- Итоговая асимптотика: $O(10 \cdot |s|) = O(|s|)$
- В зависимости от оптимальности реализации можно было получить 75 или 100 баллов

# Задача D
## Кот Гусь и случайная матрица



# Постановка задачи

- Дана матрица случайных чисел от `0` до `p - 1`
- Нужно найти ее подматрицу с максимальной суммой такой, что эта сумма делится на `p`

# Решение первой подзадачи (10 баллов)

- Достаточно реализовать любое решение за $O((nm)^2)$,
- Например:
  - Перебор координат двух углов и подсчет с помощью частичных сумм суммы на подматрице

# Решение третьей подзадачи (10 баллов)

- На одной прямой (вертикальной или горизонтальной) необходимо найти подотрезок с максимальной суммой делящейся на p
- Это можно сделать, посчитав префиксные суммы, для каждого остатка по модулю p, а затем рассмотреть наибольшую и наименьшую суммы с таким остатком

# Решение второй подзадачи (10 баллов)

- Необходимо реализовать решение за $O(n^2m)$
  - Переберем первую и последнюю строки подматрицы
  - Подзадача сводится к подзадаче с $n = 1$

# Случайность входных данных

- Первые три подзадачи возможно решить без использования случайности входных данных

- Чтобы решить четвертую подзадачу (10 баллов), нужно реализовать неоптимально правильное решение
- Сразу перейдем к решению, набирающему 100 баллов

# Решение на 80-100 баллов

- Рассмотрим подматрицы в порядке убывания суммы, от большей к меньшей
  - Возьмем всю матрицу полностью
  - Очевидно, она является подматрицой с наибольшей суммой
  - Какой может быть вторая по сумме матрица?
    - Конечно, это вся матрица, из которой удалили какую-либо крайнюю строку/столбец

# Решение на 80-100 баллов

- Можно развить эту идею, поддерживая подматрицы в `set`
- Каждый раз будем:
  - Доставать матрицу, которая является максимальной на данный момент
  - Обрезать у нее по одному строке и столбцу
  - Класть новые матрицы в `set`
  - Как только мы нашли матрицу с суммой делящейся на `p` это ответ

# Решение на 100 баллов

- Реализация с помощь `priority queue`
  - Не используя проверку, что  такую матрицу уже рассматривали
  - Поддерживаем указатель `ptr` на текущую границу, которую требуется отрезать
    - сначала увеличивать `x1`,
    - затем, возможно, сдвигать указатель
    - затем уменьшать `x2`, если указатель сейчас стоит на этой координате
    - и так далее...

# Решение на 100 баллов

- Таким образом “путь” до каждой матрицы будет ровно один
- В итоге, в priority queue будет храниться `(x1,x2,y1,y2,ptr)`
  - ptr принимает значения от `0` до `3`
- Можно доказать, что с очень высокой вероятностью, количество матриц, которые придется рассмотреть, достаточно мало (порядка `O(p)`)
- Также возможны решения, которые перебирают матрицы только “достаточно” большого размера.

# Задача E
# Упорядочивания

Автор задачи:
**Григорий Шовкопляс**
Разработка задачи:
**Григорий Шовкопляс**

# Постановка задачи

- Есть дерево на n вершинах
- Каждое ребро ориентировано
- Посчитать количество топологических сортировок дерева

# Решение задачи (16 баллов)

- Переберем все перестановки чисел от 1 до n
- Для каждого ребра проверяем, что его начало в перестановке раньше конца
- Сложность – $O(n! \cdot n)$

# Решение задачи (32 балла)

- Динамическое программирование по подмножествам
- `dp[S]` — количество перестановок вершин из множества `S`, которые удовлетворяют необходимым условиям
- Чтобы пересчитывать динамику, переберем, какую вершину мы добавляем в множество
- Если не существует ребер из новой вершины в текущее множество, ее можно добавить и обновить значение динамики
- Сложность – $O(2^n \cdot n)$

# Решение задачи за полиномиальное время

- Подвесим дерево за произвольную вершину
- Пусть `sz[u]` — размер поддерева вершины `u`

# Решение для частного случая

- Пусть все ребра направлены вниз, то есть от родителя к ребенку
- Динамическое программирование по поддеревьям
- Пусть `dp[u]` — количество топологических сортировок поддерева вершины u
- Пусть вершина u имеет k детей: $v_1, v_2, \ldots, v_k$
- Рассмотрим какую-нибудь топологическую сортировку поддерева u
- На первом месте всегда обязана стоять вершина u
- Остается `sz[u]-1` свободных позиций

# Решение для частного случая

- Для каждого поддерева $v_1, v_2, \ldots, v_k$ можно произвольно выбрать множество позиций, на которых будут стоять вершины из этого поддерева
- Для `i`-го из этих поддеревьев количество вариантов подмножества позиций будет составлять `C(sz[u]-1-sz[`$v_1$`]-sz[`$v_2$`]-...-sz[`$v_{i-1}$`], sz[i])`
- Посчитаем все значения `C(n, k)` заранее
- Каждое из поддеревьев можно упорядочить произвольным образом
- Надо умножить ответ на `dp[`$v_1$`],dp[`$v_2$`],...,dp[`$v_k$`]`

# Полное решение

- Пусть теперь некоторые ребра ведут наверх
- Удалим из дерева все ребра, ведущие наверх, и решим задачу для оставшихся ребер
- Для каждого из получившихся деревьев решим задачу в частном случае
- Для каждого дерева в итоговом упорядочивании можно произвольным образом выбрать множество мест
- Скомбинируем ответы

# Полное решение

- Мы посчитали некоторое количество неправильных упорядочиваний
- Пусть $A_i$ — множество упорядочиваний вершин, в которых концы ребра вверх под номером $i$ расположены в неправильном порядке
- Нам нужно найти размер объединения множеств $A_i$
- Воспользуемся формулой включений-исключений
- Каждое из ребер вверх можно либо убрать, либо перевернуть
- Хотим посчитать для каждой четности количества перевернутых ребер итоговое количество упорядочиваний

# Полное решение

- Модифицируем динамику из предыдущего решения
- Новое состояние — `dp[u][size][parity]`
  - u — вершина
  - `size` — размер поддерева вершины u по ребрам, ведущим вниз
  - `parity` — четность количества ребер вверх, которые были перевернуты
- Значение динамики — количество упорядочиваний всего исходного поддерева вершины u за исключением ее текущего поддерева по ребрам вниз

# Полное решение

- Когда встречаем ребро вниз, можем только добавить его и увеличить параметр `size`
- Ребро вверх можно либо развернуть, увеличив параметр `size` и изменив параметр `parity`, либо удалить, изменив пересчитав значение динамики

# Полное решение

- При наивной реализации время работы составит $O(n^3)$
- Если считать динамику только для достижимых состояний, время работы составит $O(n^2)$

```
printf("%s\n", "Спасибо за внимание")
```

Материалы олимпиады

http://neerc.ifmo.ru/school/io