# Московская олимпиада школьников по информатике 2021

Разбор задач

# Задача «Три пловца»

Автор задачи: Михаил Пядёркин
Разработчик: Егор Чунаев

# Формальная постановка

- Есть три пловца, они проплывают бассейн туда-сюда за $a$, $b$ и $c$ минут, все они начали заплыв в момент времени $0$
- Дан момент времени $t$, сказать через сколько времени мы увидим какого-нибудь пловца около стартового борта бассейна

# Ключевая идея

- Ответ равен разности такого минимального числа $x \geq t$, что $x$ делится на $a$, $b$ или $c$, и числа $t$

# Решение за *min(a, b, c)*

- Можно перебирать числа *t*, *t + 1*, … *x*, не более чем через *min(a, b, c)* шагов мы получим ответ.
- Решение работает за *O(min(a, b, c))* и набирает 40 баллов

# Решение за $\sqrt{t}$

- Если есть число меньше $\sqrt{t}$, то действуем как в предыдущем решении
- Иначе перебираем числа $a$, $2a$, $3a$, …, $pa$. $b$, $2b$, $3b$, …, $kb$ и $c$, $2c$, $3c$, …, $zc$.
- В каждом из трех случаев не более чем за $\sqrt{t}$ шагов мы дойдем до числа не меньше $t$
- Решение работает за $O(\sqrt{t})$ и набирает 60 баллов

# Полное решение

- Ответ равен *min(⌈t / a⌉ * a, ⌈t / b⌉* b, ⌈t / c⌉ * c) - t*
- Надо быть осторожным с вещественными числами, функция *ceil* может ошибаться на *+/-1*
- Один из самых лучших способов округлить вверх число *a / b* в коде это *(a + b - 1) / b*
- Решение работает за *O(1)*

# Задача «Аквапарк»

Автор и разработчик задачи: Филипп Грибов

# Формальная постановка

- Сеанс аквапарка длится *a* минут
- Бассейны чистятся каждые *m* минут
- Чистка бассейнов длится *b* минут
- Требуется узнать максимальное число сеансов за *n* минут непрерывной работы аквапарка

# Решение за $O(n)$

- Просимулируем работу аквапарка
- Если после конца очередного сеанса прошло хотя-бы $m$ минут с момента запуска или последней чистки, то будем прибавлять $b$ к текущему времени

# Решение за $O(1)$

- Оценим число сеансов до первой чистки бассейнов.
- Это $cnt0 = min(\lceil m/a \rceil, \lfloor n/a \rfloor)$.
- После этого останется $n1 = max(0, n - cnt0 \cdot a)$ времени

# Решение за $O(1)$

- Оценим число сеансов между чистками бассейнов.
- Это $cnt1 = \lceil (m - b)/a \rceil$
- Тогда между чистками пройдёт $t = b + cnt1 \cdot a$ времени
- Получаем, что всего чисток будет $1 + n1 / t$, а всего сеансов между чистками бассейнов - $cnt1 \cdot n1 / t$
- После последней чистки бассейнов останется $n1 \% t$ времени
- Тогда вычев из него $b$ и поделив на $a$ мы получим сколько сеансов будет после последней чистки.

# Задача «Максимальная ширина»


Автор и разработчик задачи: Александр Шеховцов

# Формальная постановка задачи

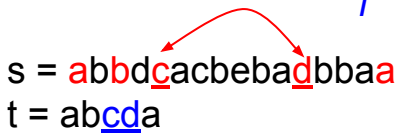
- Есть две строки $s$ и $t$
- Рассмотрим все подпоследовательности $s$ равные $t$
- Среди них нужно найти подпоследовательность которая максимизирует расстояние между двумя какими-то соседними символами

# Решение за $O(n2^n)$

- Переберем все подпоследовательности строки $s$
- Проверим, что текущая подпоследовательность равна $t$
- Обновим ответ максимальным расстоянием между соседними символами

# Решение за $O(n^2)$

- Заметим, что максимальное расстояние достигается между какой-то парой символов $t_i$ и $t_{i+1}$




- Переберем $i$
- Тогда позиция символа $t_i$ в строке $s$ должна быть как можно левее, а позиция символа $t_{i+1}$ в строке $s$ должна быть как можно правее

# Решение за $O(n^2)$

- Пусть $left_i$ самое левое положение символа $t_i$ в $s$.
- Тогда $left_i$ можно вычислить как самый первый символ в $s$ после $left_{i-1}$ равный $t_i$
- Аналогично определим и вычислим $right_i$
- Тогда максимальное возможное расстояние между символами $t_i$ и $t_{i+1}$ равно $right_{i+1} - left_i$

# Полное решение $O(n)$

- Заметим, что в предыдущем решении за $O(n^2)$ величины $left_i$ и $right_i$ можно предподсчитать за $O(n)$
- Тогда осталось пройти по $i=1…(m - 1)$ и обновить ответ значением $right_{i + 1} - left_i$

# Задача «Ход гения»


Автор задачи: Владимир Романов
Разработчик: Евгений Антрушин

# Формальная постановка

- Требуется найти такие числа $x$, $y$, что каждое из них в двоичной записи состоит из ровно $a$ нулей и $b$ единиц, а $x - y$ содержит ровно $k$ единиц
- $x$, $y$ не содержат ведущие нули

# Решение за $(a + b) * 4^{a+b}$

- Переберем $x$ и $y$ до $2^{(a+b)}$, будем за двоичную длину чисел искать разность

# Ключевая идея

- Почти всегда представимы только $k$ принадлежащие отрезку $[0; a + b - 2]$ (за исключением примитивных случаев $a = 0$ или $b = 1$, когда числа задаются однозначно и $k$ может быть равно только $0$)
- Если зафиксировать число $x$ максимально возможным, то мы все еще сможем набрать любое $k$ из представимых
- Нужно конструктивно построить ответ

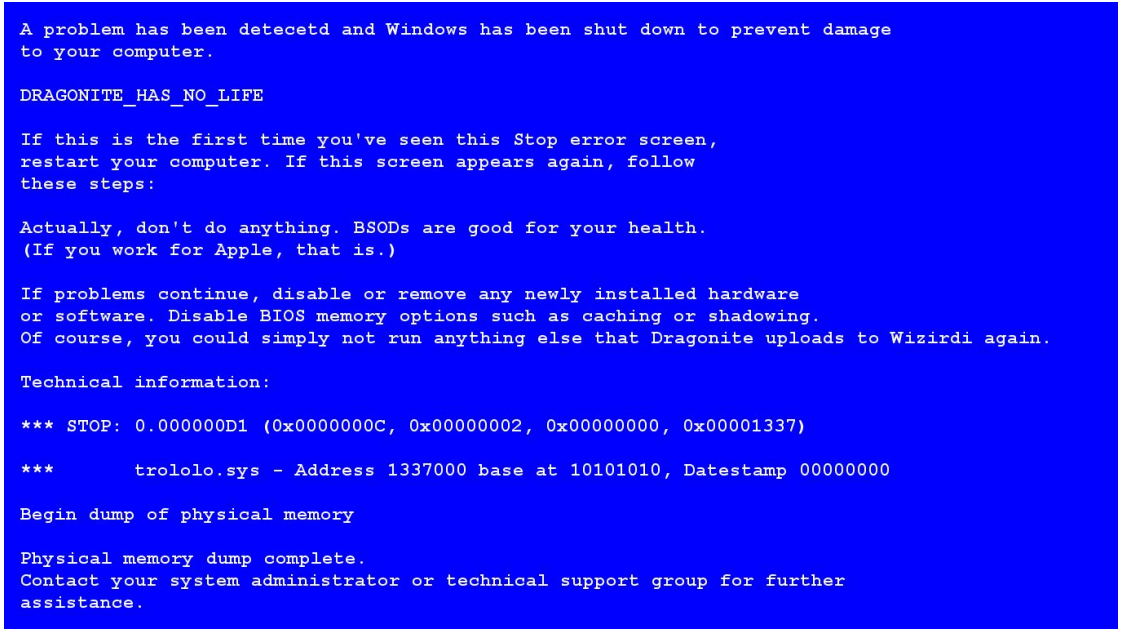
# Возможные полиномиальные решения

- Пусть $y = x = 1111...1000...000$. Будем брать самую правую единичку в $y$ из образующих префикс и жадно пытаться поставить ее поочередно на все остальные места правее. Ставим на позицию, где итоговое число единиц как можно больше возрастет. После каждой попытки поставить проверяем разность на количество единиц за $O(a + b)$. Такое решение работает за $O(b(a + b)^2)$ и набирает не менее $50$ баллов.
- Можно заметить, что нет смысла ставить новую единичку правее предыдущей, это будет работать за $O((a + b)^2)$ и наберет не менее $60$ баллов

# Полное решение

- Пусть у нас есть число *y = 1..10..0*, заметим, что при движении самой правой единички префикса еще правее, с каждой позицией число единичек в *x - y* возрастает на *1* (рассматриваем числа вида *1111000*, *1110100*, *1110010*, *1110001*)
- Теперь возьмем следующую единицу, заметим, что сдвинув ее ровно на одну позицию вправо мы увеличим число единиц в *x - y* на *1*. Заметим, что мы можем делать так до тех пор, пока единичный префикс состоит из хотя бы двух единиц (рассматриваем числа вида *1101001*, *1011001*).
- Решение работает за *O(a + b)*
- Довольно просто заметить, что представленные ранее полиномиальные решения являются неаккуратной реализацией полного
- Не забываем про *a = 0* или *b = 1* где ответ существует только для *k = 0*

# Задача «Почти отказоустойчивая база данных»




Автор задачи: Александр Курилкин
Разработчик: Евгений Шульгин

# Формальная постановка

- Дано *n* массивов, в каждом по *m* чисел
- Надо найти какой-нибудь массив длины *m* такой, чтобы все данные *n* массивов отличались бы от него не более чем в *1* позиции
- Или определить, что такого массива нет

# Решение за $2^m nm$ (все числа равны $1$ или $2$)

- Можно полностью перебрать все возможные варианты ответа и проверить их пригодность
- Если в массиве находятся только числа $1$ или $2$ (первая группа), то ответом будет также массив из единиц и двоек
- Всего таких массивов $2^m$
- Такое решение может пройти первую группу и дать не менее $10$ баллов

# Ключевая идея

- Можно взять любой из $n$ исходных массивов, и попытаться изменить в нем один элемент так, чтобы измененный массив стал ответом
- Так как искомый ответ не должен отличаться от выбранного массива более чем в одном элементе, то мы таким образом переберем все возможные ответы

# Решение за $(n+m)nm^2$

- Берём один из $n$ исходных массивов, проходим все элементы от $1$-го до $m$-го
- Каждый элемент можно заменить на какой-нибудь другой, встречающийся в массивах. Всего элементов $nm$, но в тестах с ответом “Yes” уникальных элементов максимум $O(n + m)$. Это возможный ответ
- Проверка возможного ответа после замены занимает $nm$ операций
- Итоговая сложность $(n+m)nm^2$, достаточно для $20$ баллов

# Решение за $n^2m$

- Не имеет смысл заменять элемент на такой, который не встречается в этом столбце, так как это невыгодно
- В частности, если в столбце все числа одинаковые, то его можно проигнорировать и ничего в нем не менять
- Можно доказать, что в тестах с ответом “Yes” надо рассмотреть и проверить не более $O(n)$ замен элементов
- Итоговая сложность $n^2m$, дает минимум $40$ баллов

# Решение за *Snm*

- Все числа от *1* до *50* (т. е. *S = 50*).
- Фиксируем первый массив, и находим позицию *j* такую, что у какого-нибудь *k*-го массива *j*-й элемент не такой, как в первом массиве
- Нужно попробовать поменять *j*-й элемент первого массива на другой. Все возможные варианты можно проверить перебором за *Snm*
- Если это не сработало, это значит, что в ответе *j*-м элементом может быть число из первого массива. Возможным ответом является *k*-й массив с измененным *j*-м элементом. Проверка этого варианта займет *nm* операций
- Такое решение получает минимум *40* баллов

# Решение за *min(S,n)nm* (продолжение *Snm*)

- Все числа от *1* до $10^9$ (т. е. $S = 10^9$).
- Для достаточно маленьких *n* и *m* можно применить решение из прошлого слайда, но итерироваться не от *1* до *S*, а по всем уникальным числам в заданном столбце
- Такое решение получает минимум *70* баллов

# Решение за *nm*

- Зафиксируем первый массив. Назовем “плохими” остальные массивы, с которыми он отличается в двух позициях (отличия в трех и более позициях означают ответ “No”).
- Если “плохих” массивов не найдено, то ответом является первый массив (все остальные массивы отличаются от него не более чем в одной позиции).
- Иначе рассмотрим любой “плохой” массив и попытаемся изменить первый массив так, чтобы снизить кол-во отличающихся с ним позиций.
- Будет не более двух вариантов для изменения. После изменения нужно проверить ответ (что никаких “плохих” массивов больше нет) за *nm*.
- Это решение получает *100* баллов.

# Другое решение за $nm$

- Хорошей идеей является наблюдение, что в ответе на позиции $j$ скорее всего будет “популярный” элемент $p_j$, встречающийся чаще всего среди массивов на $j$-й позиции.
- Если в каком-то массиве $k$ на $j$-й позиции находится “непопулярный” элемент, то нужно проверить в качестве ответа этот массив $k$ с $p_j$ на позиции $j$.
- Для более уверенного ответа можно удалить “повторяющиеся” массивы, оставив только уникальные.
- Алгоритм может плохо работать для $n = 2$, это тоже можно учесть.
- Подобные решения могут получать близко к $100$ баллам в зависимости от реализации.

# Задача «Древесные жабы (и лягушки)»

Автор задачи: Владислав Макеев
Разработчик: Максим Деб Натх

# Формальная постановка

- Дано дерево на $n$ вершинах
- Надо ответить на запросы вида

*“найти сумму номеров всех вершин, которые находятся ближе к вершине $u$, чем к вершине $1$”*

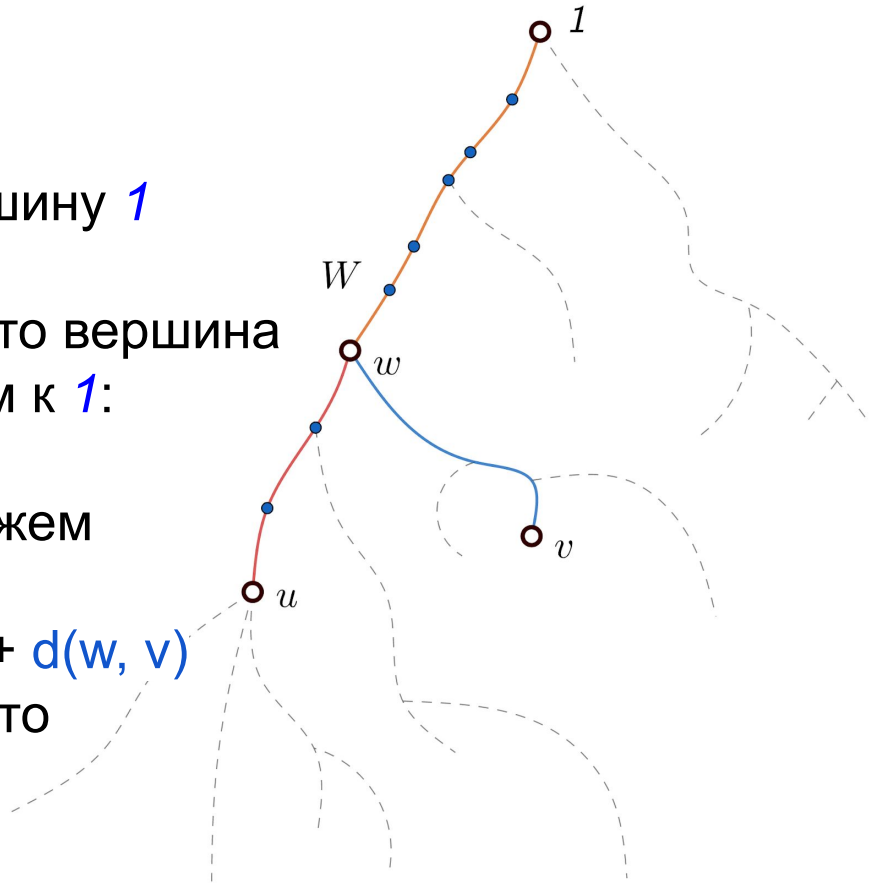
# Частичные решения

- Решение за $O(n^2m)$: ответ на каждый из запросов найдём пройдясь по всем вершинам и вычислив расстояния до них от вершин *1* и *u*

- Решение за $O(n^2+m)$: предподсчитаем ответ для каждой вершины, запустив из каждой из вершин DFS или BFS

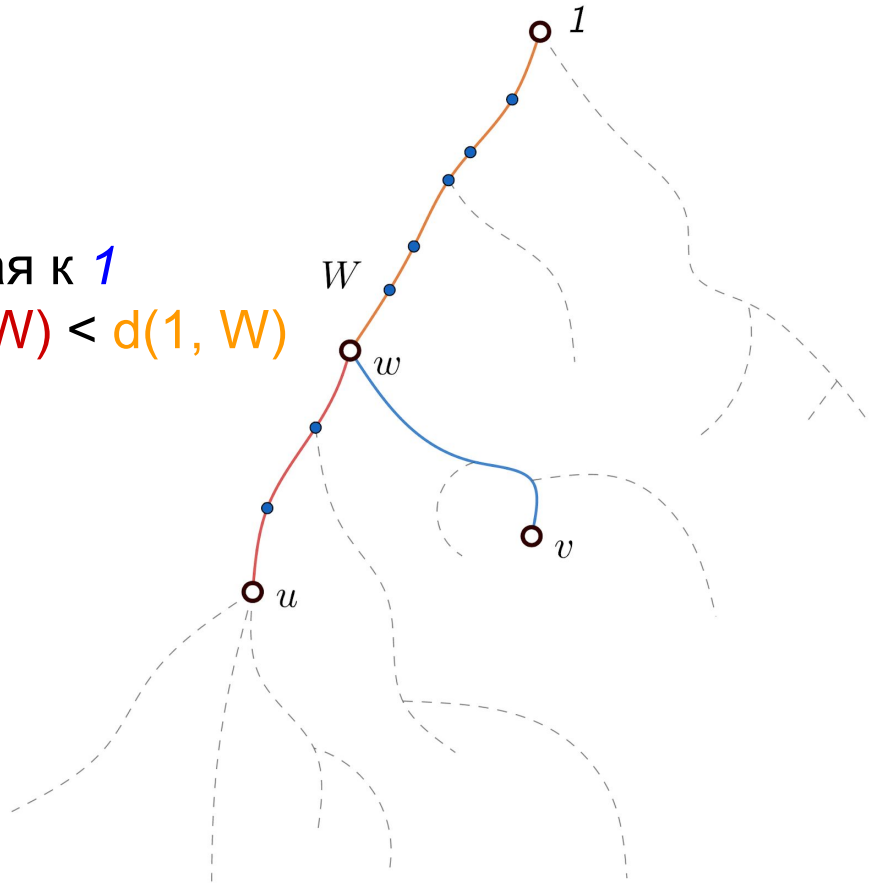
# Ключевая идея

- Подвесим дерево за вершину *1*
- Пусть *w = lca(v, u)*
- Запишем, что означает, что вершина *v* ближе к вершине *u*, чем к *1*: *d(u, v) < d(1, v)*
- Каждый из двух путей можем разбить на два: d(u, w) + d(w, v) < d(1, w) + d(w, v)
- Это эквивалентно тому, что d(u, w) < d(1, w)



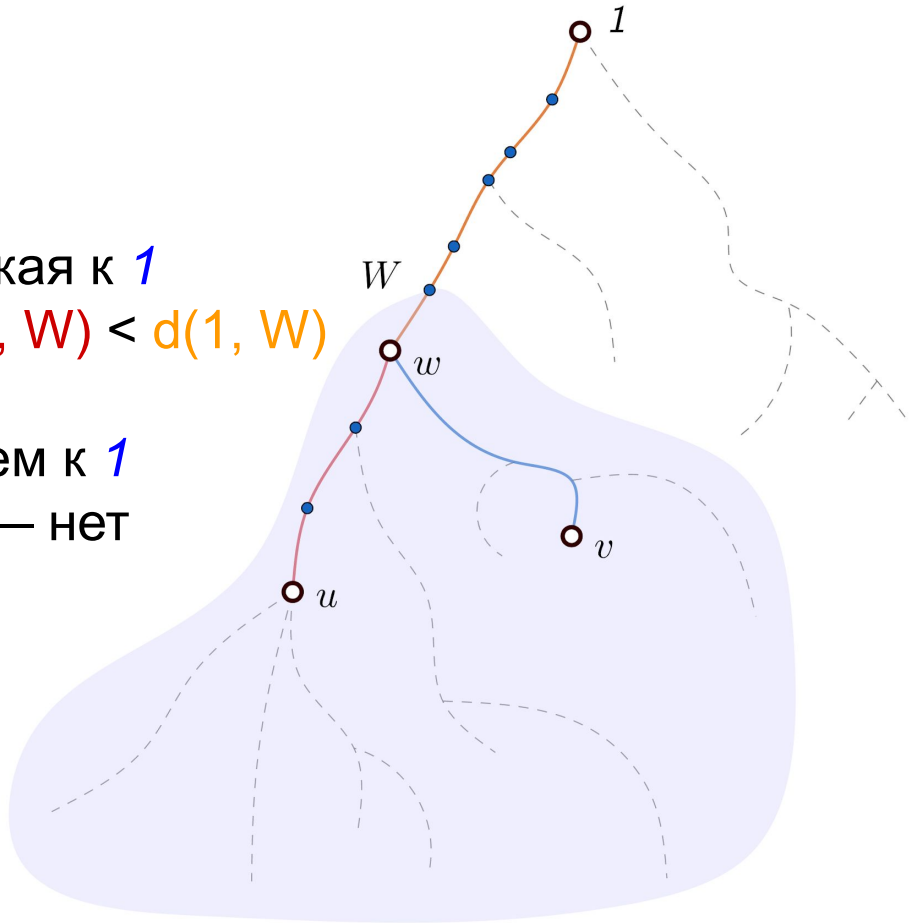
# Ключевая идея

- d(u, w) < d(1, w)
- Пусть *W* — самая близкая к *1* вершина, такая, что d(u, W) < d(1, W)



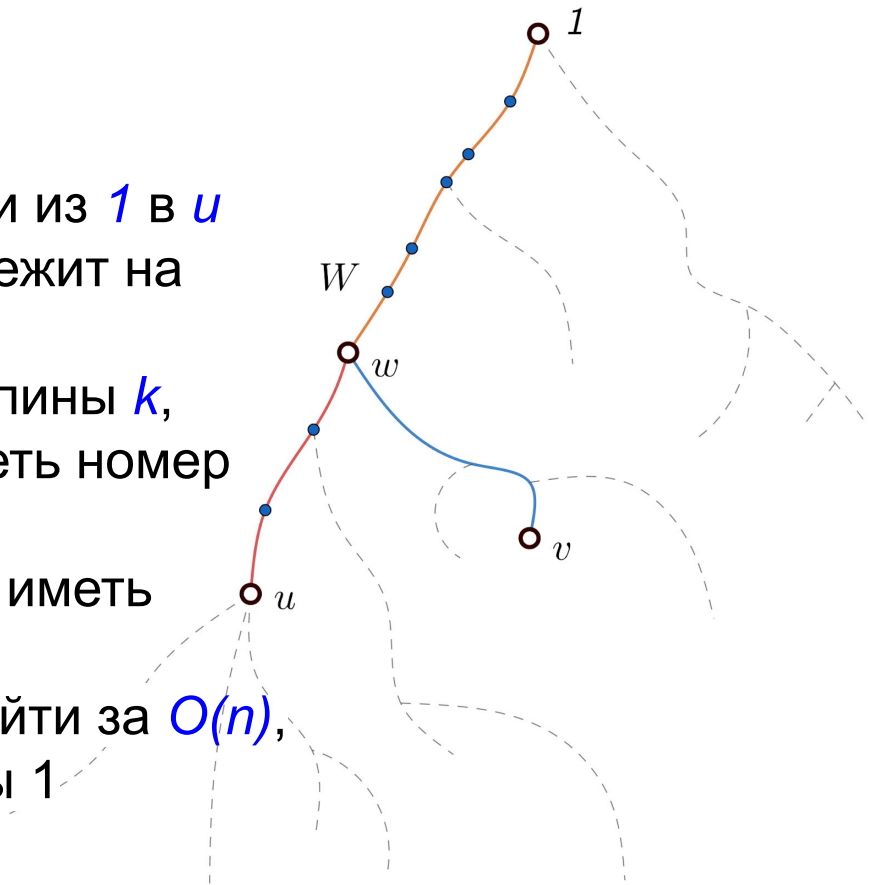
# Ключевая идея

- d(u, w) < d(1, w)
- Пусть *W* — самая близкая к *1* вершина, такая, что d(u, W) < d(1, W)
- Тогда все вершины в её поддереве ближе к *u*, чем к *1*
- А остальные вершины — нет



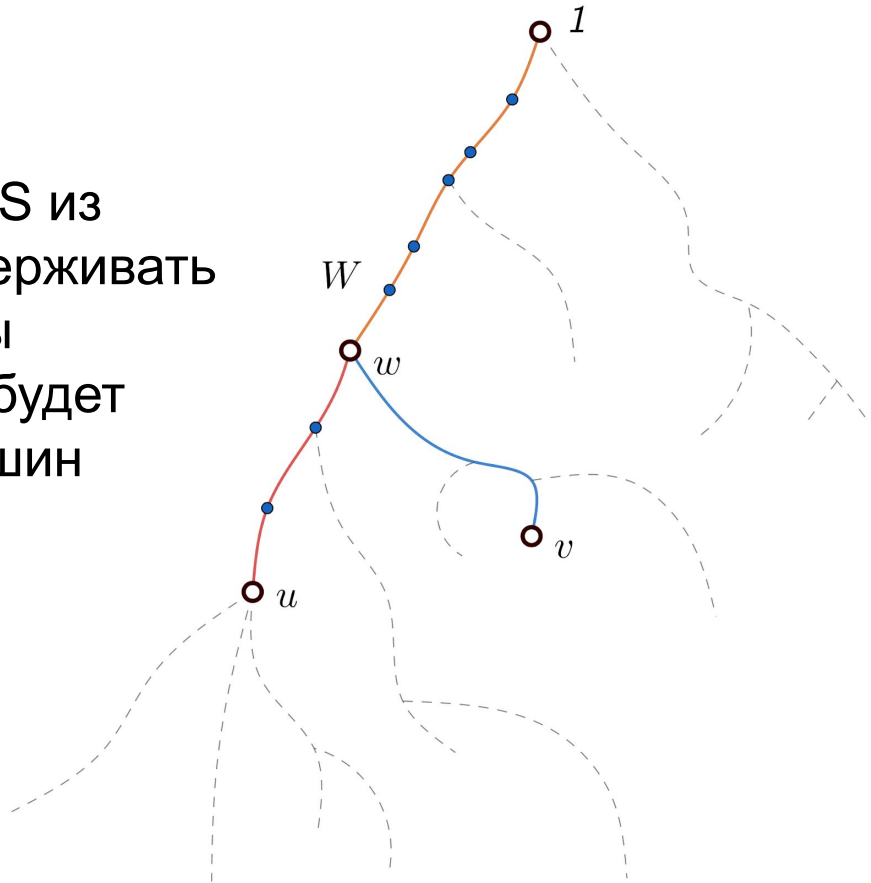
# Полное решение

- Вершина *W* лежит на пути из *1* в *u*
- И при этом, она “почти” лежит на середине пути:
  - если путь нечётной длины *k*, то вершина будет иметь номер *(k + 1)/2*
  - иначе вершина будет иметь номер *(k + 2)/2*
- Такие вершины можно найти за *O(n)*, запустив DFS из вершины 1



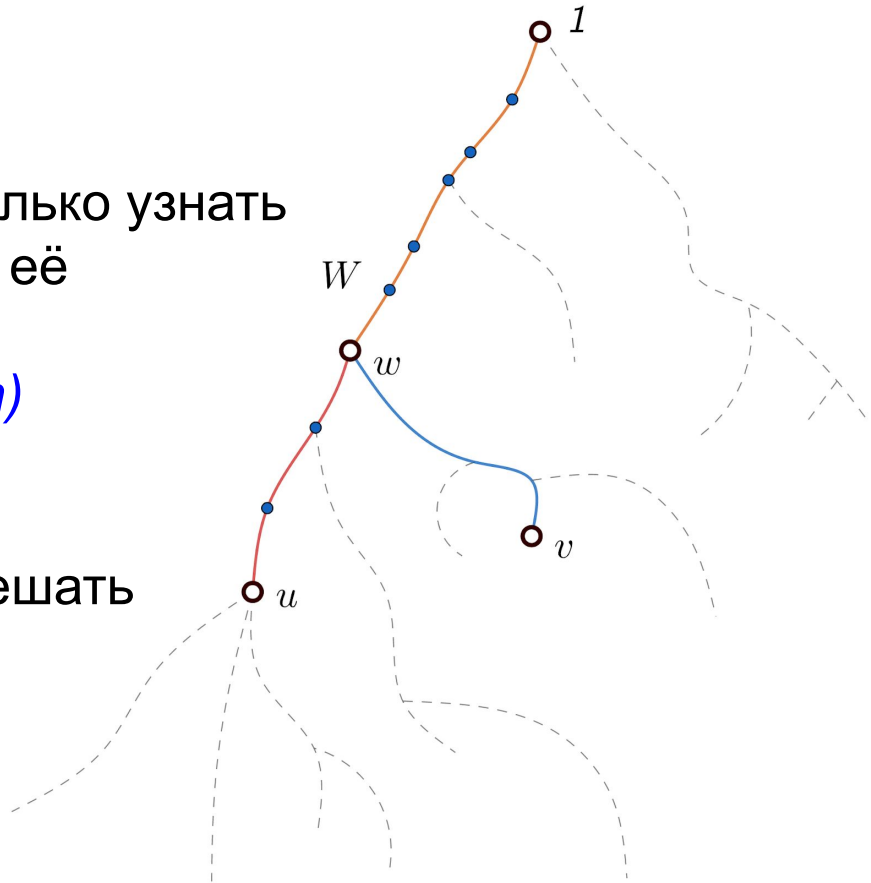
# Полное решение

- Запустим из вершины DFS из вершины 1 и будем поддерживать путь до текущей вершины
- Тогда вершину W можно будет найти для каждой из вершин

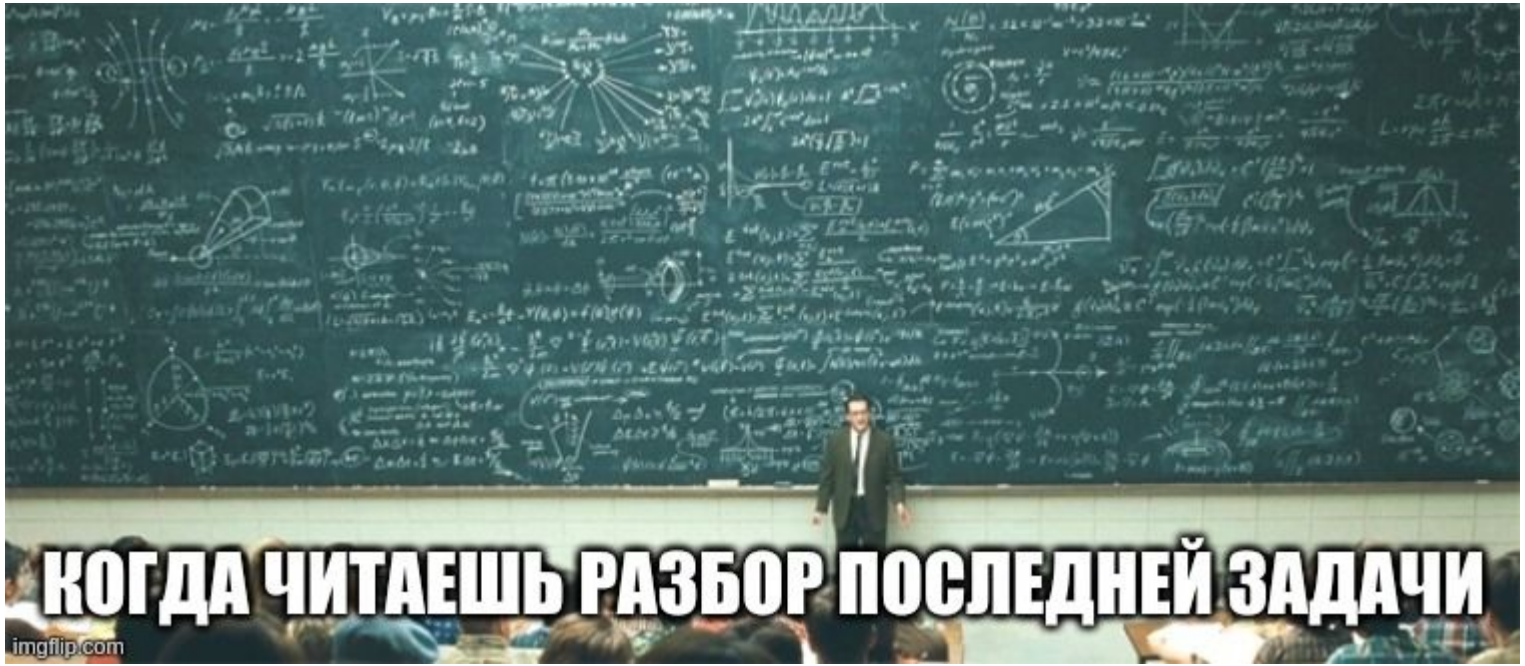


# Полное решение

- Зная вершину *W*, надо только узнать сумму номеров вершин в её поддереве
- Это тоже делается за *O(n)* с помощью динамики по поддеревьям
- Таким образом, можем решать задачу за *O(n + m)*



# Задача «Сложная задача»




Автор задачи: Филипп Грибов

Разработчик: Константин Амеличев

# Формальная постановка

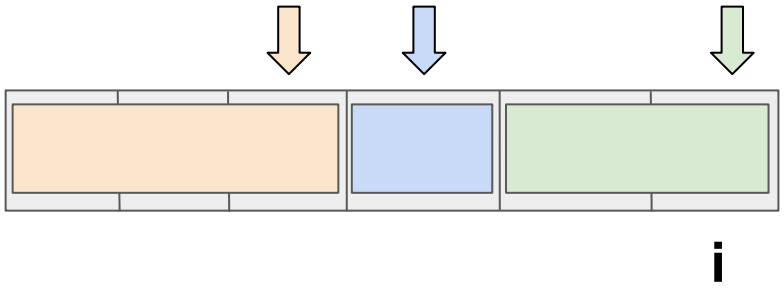
- Дан набор прямоугольников, которые надо поместить на лист бумаги
- Между некоторыми прямоугольниками можно ставить переносы
- Нужно, чтобы прямоугольники не вылезали за пределы своей строки, а суммарная высота строк была минимальной

# Решения за $n^3$ и $n^2$

- Будем заполнять лист построчно
- Пусть $dp_i$ — минимальная суммарная высота листа, в который вписываются первые $i$ прямоугольников.
- Тогда для перехода надо выбирать такое $j$, что
  - $w_j + \ldots + w_i$ не больше ширины листа
  - $dp_{j-1} + max(h_j, \ldots, h_i)$ минимально.
- Если уменьшать $j$ от $i$ до $1$, то можно поддерживать максимум и сумму.

# Ключевая идея

- Назовем рекордом такую позицию $j$, что $h_j = max(h_j, \dots, h_i)$
- Слева от каждого рекорда есть отрезок, за который он отвечает
- Внутри отрезка нужно найти минимальное значение $dp$



# Полное решение

- Будем поддерживать рекорды в окне
- При добавлении рекорда он вытесняет какой-то суффикс рекордов
- Склеим все эти отрезки в один
- Заметим, что $dp_i \le dp_{i+1} \Rightarrow$ минимум всегда слева
- Смогли определить стоимость одного отрезка. Осталось узнавать минимум в окне.
- Иногда меняется стоимость отрезков на границах окна, а также происходят добавления в конец и удаления из начала

# Полное решение

- Будем хранить *deque* с поддержкой минимума
- Для этого нужны два стека с поддержкой минимума
- Один хранит начало окна, другой конец
- Когда один из стеков пустой, переносим в него половину элементов из другого
- Получаем амортизированную сложность *O(n)*