Московская городская командная олимпиада 2017 года

Разбор задач

Задача A. «В свете софитов»

Автор и разработчик задачи: Никита Сендерович

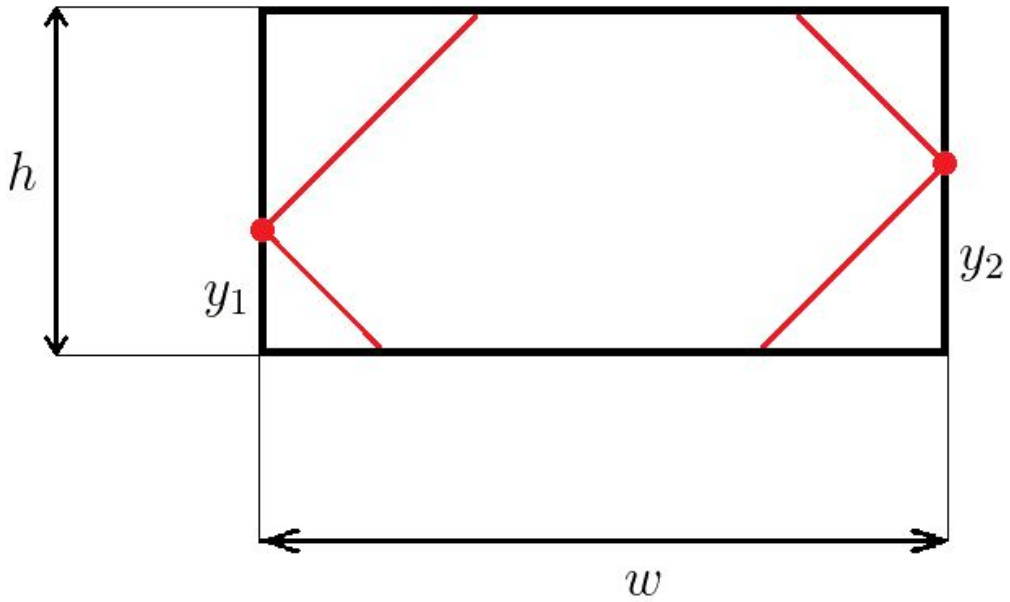
Постановка задачи

- К противоположным боковым сторонам прямоугольника *w* на *h* прикреплены 2 прожектора на заданных высотах y_1 и y_2.
- Оси прожекторов горизонтальны, прожектор освещает угол величиной 90 градусов.
- Требуется проверить, освещен ли весь прямоугольник.

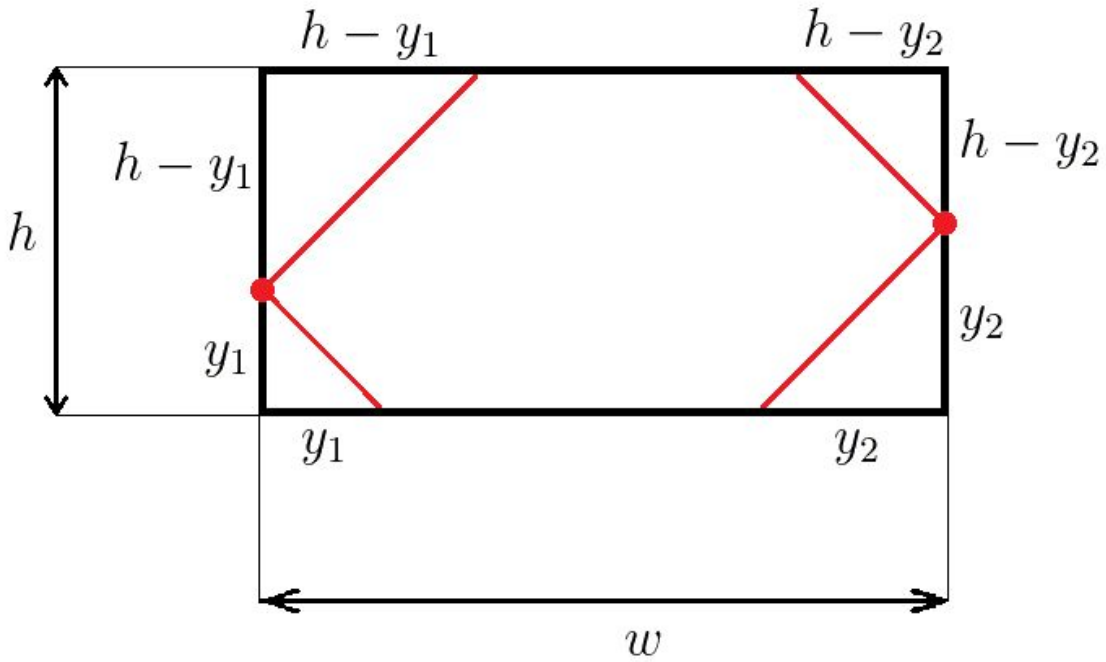
Идея решения

- Прямоугольник полностью освещен в том и только в том случае, когда полностью освещены его горизонтальные стороны.
- Это так, потому что любую неосвещённую точку можно подвинуть либо вверх до верхней стороны, либо вниз до нижней.
- По входным данным определяем пересечения углов с горизонтальными сторонами прямоугольника.
- Для каждой стороны проверяем, что она полностью освещена.

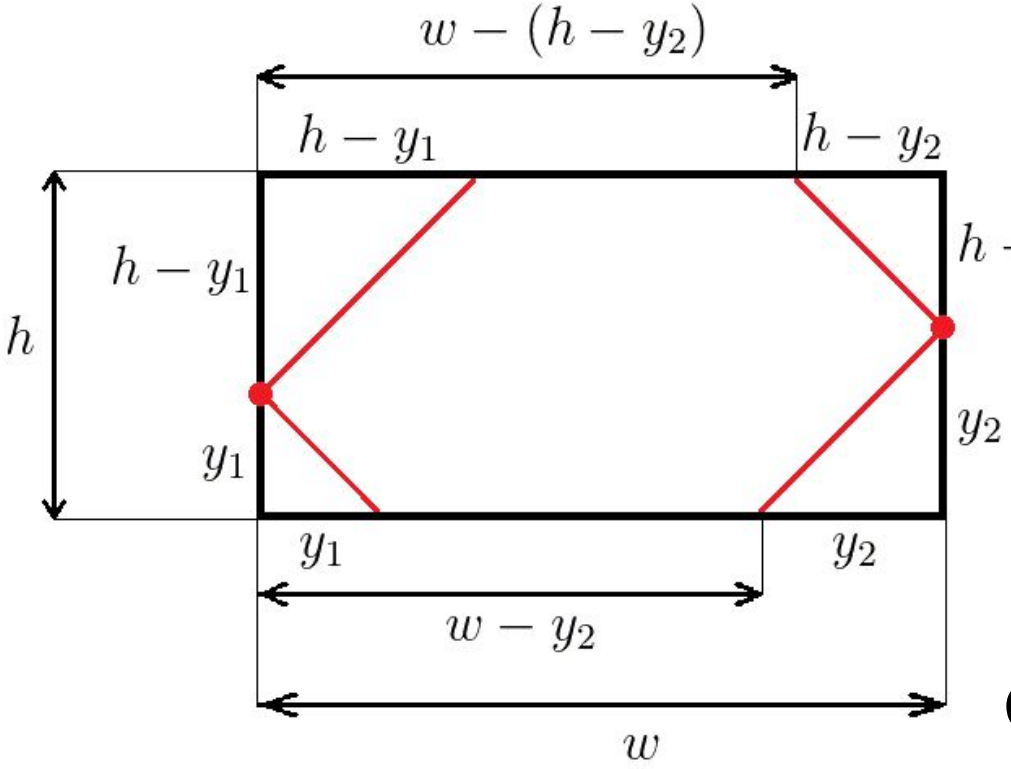
Решение: считаем отрезки



Решение: считаем отрезки



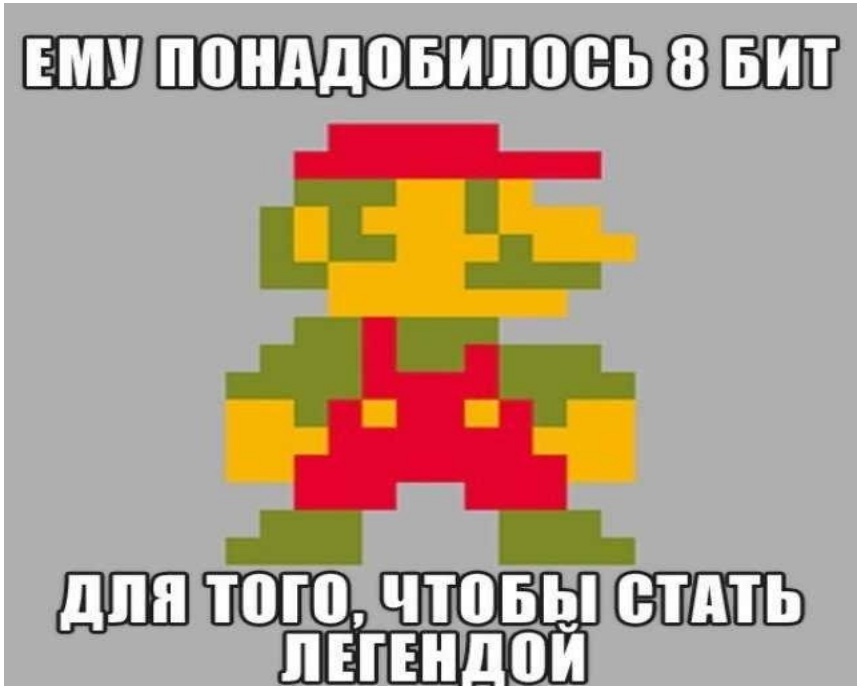
Решение: считаем отрезки




```
if y1 <= w - y2 and
   h - y1 <= w - (h - y2):
| print "Yes"
else:
| print "No"
```

Сложность решения – *O(1)*.

Задача J. «Компания и побитовое И»



Автор задачи: Михаил Пядёркин

Разработчик: Георгий Халин

Постановка задачи

- Дан массив *a*, состоящий из *n* чисел.
- Требуется посчитать сумму побитовых И среди всех пар чисел.
- Строго говоря: посчитать сумму *(a[i] & a[j])* по всем парам *(i, j)*, таким, что $i < j$, где & – операция побитового И.

Ключевая идея

- Будем решать задачу по разрядам.
- Определим вклад в ответ *k*-го разряда слагаемых в двоичной системе счисления.
- Каждая пара чисел, в двоичной записи которых *k*-й бит равен единице, добавляет к ответу 2^k.
- Так как числа в исходном массиве не превосходят 10^9, то нас интересуют значения k от *0* до *29*

Решение

- Для каждого *k* от *0* до *29* посчитаем величину *cnt[k]*, равную количеству чисел в массиве *a*, содержащих единицу в *k*-м разряде двоичной системы счисления.
- Мы хотим посчитать количество **пар** чисел в массиве *a*, содержащих единицу в *k*-м разряде.
- Легко видеть, что оно равно
  $C(cnk[k], 2) = cnt[k] \cdot (cnt[k] - 1) / 2.$

Решение

- Для того, чтобы насчитать массив *cnt* для, будем для каждого *k* проходиться по всем элементам массива и для каждого числа проверять, равен ли *k*-й разряд единице.
- В языке C++ можно проверить, равен ли *k*-й разряд в числе *a[i]* единице, следующим образом:

$$((1 << k) \,\&\, a[i]) > 0.$$

Решение

- Не забываем, что ответ не помещается в 32-битный int.
- Асимптотика: $O(n \log c)$, где n – количество чисел в массиве, а c – максимально возможное число.

Задача G. «Коробка конфет»

Автор и разработчик задачи: Дмитрий Саютин

Постановка задачи

- Есть коробка из синих и красных конфет. Пока в ней есть конфеты, из неё вытаскивается и съедается та конфета, которой больше всего. При равенстве выбирается красная.
- Дана строка из символов “`r`”, “`b`” и “`?`”, обозначающих тип съеденной в каждый момент времени конфеты.
- Посчитать число способов составить коробку конфет, чтобы она подходила под данную строку.

Ключевая идея

- Заметим, что любая корректная последовательность выглядит как “`rr…rr|rbrb…rb`” или “`bb…bb|rbrb…rb`”, то есть сначала будет съедаться самая частая конфета, а потом они будут чередоваться.
- Какая-то из двух частей может быть пустой, например: “`rr…rr|`”, “`bb…bb|`”, “`|rbrb…rb`”.
- Нужно посчитать количество последовательностей такого вида, подходящих под заданную строку.

Решение

- Перебираем в порядке возрастания от 0 число пар “rb”, из которых будет состоять вторая часть.
- Этому может помешать буква “r” на позиции, где должна быть “b” или наоборот. Если находим такую мешающую букву, прерываем цикл.

```
?r??rr???r??brb
 ^ плохо
```

Решение

- Осталось проверить, можно ли всё слева от зафиксированного суффикса заменить на буквы “r” или на буквы “b”.
- Мы можем поменять всё слева на букву “r” тогда и только тогда, когда слева нет букв “b”. Аналогично для буквы “b”.
- Проверять это условие циклом уже нельзя, иначе получится квадратичная сложность и около 10^{10} операций, что вряд ли уложится в 1 секунду.

Решение

- Запомним позицию самых левых букв “`b`” и “`r`” (а если таких нет, запомним *n+1*). Теперь можно за *O(1)* понять, есть ли на префиксе хотя бы одна буква “b” или буква “`r`”.
- Финальная версия нашего решения: перебираем суффикс чётной длины, пока он выглядит как “`rb`...`rb`” и прибавляем по единичке на каждый из вариантов заполнить остаток буквами “`b`” или “`r`”.

Решение

- Важный случай: если зафиксированный суффикс совпадает со всей строкой, то надо прибавить к ответу 1, а не 2, так как нет разницы, чем заполнять пустой префикс: “`rbrb…rbrb`”.
- Сложность: *O(n)*.

Задача E. «Сортировка монет»

Автор задачи: Глеб Побегайло
Разработчик: Егор Чунаев

Постановка задачи

- Дан массив из нулей и единиц и запросы изменения нуля на единицу.
- После каждого запроса надо посчитать количество итераций внешнего цикла сортировки пузырьком, которое потребуется, чтобы отсортировать массив.

Решение

- Решим задачу для фиксированного массива.
- Если массив состоит только из единиц, он уже отсортирован ⇒ ответ равен 1:

1111111…1111111

- Иначе рассмотрим самый правый нуль.
- Если левее самого правого нуля нет единиц, то массив уже отсортирован ⇒ ответ равен 1:

0000...0001111…1111

Ключевая идея

- Пусть левее самого правого нуля ровно *k* единиц.
- За одну итерацию ближайшая слева единица сдвинется на позицию этого нуля, а сам нуль сдвинется на одну позицию влево

```
до:    0001001000011111   k=2
после: 0000010000111111   k=1
```

- После одной итерации у нас будет *k – 1* единиц левее самого правого нуля. Значит, ответ на задачу это *k + 1*.

Решение

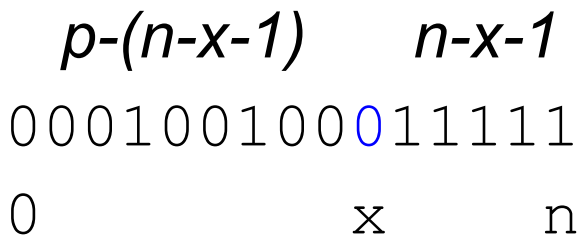
- Вернемся к исходной задаче.
- Будем поддерживать указатель на самый правый нуль.
- Поскольку в результате запросов нули только исчезают, указатель перемещается только влево.

```
000100100011111
      ^
```

- Если самый правый ноль исчез, двигаем указатель влево циклом, пока не найдём очередной ноль.

Решение

- Пусть указатель стоит на позиции *x* (в нумерации с нуля), а в массиве всего *p* единиц.
- Справа от *x* все символы – единицы, значит справа ровно *n – x – 1* единиц.
- Значит слева ровно *p – (n – x – 1)* единиц.



Анализ сложности

- Цикл, которым мы двигаем указатель на самый правый ноль, суммарно сделает $O(n)$ итераций.
- Иными словами, амортизированная сложность – $O(1)$ на запрос.
- Зная указатель, мы отвечаем на запрос за $O(1)$.
- Итоговая сложность решения: $O(n + q)$.

Задача D. «Национальное достояние»

Автор задачи: Глеб Евстропов
Разработчик: Роман Горбунов

Формальная постановка

- Дано *n* слов.
- Заглавная буква считается лексикографически меньшей, чем строчная буква: “`B`” < “`b`” (как в реальном ASCII).
- Можно ли некоторые буквы сделать заглавными во всех *n* словах так, чтобы слова стали идти в лексикографическом порядке?

Ключевая идея

- Если строки s_i и s_{i+1} не являются префиксами друг друга, нужно, чтобы $s_{i,k}$ было меньше, чем $s_{i+1,k}$, где k – первая позиция, где строки отличаются.

Ключевая идея

- Пусть мы рассматриваем строки s_i и s_{i+1}, а k – позиция их первого различия. Тогда есть два случая:
  - $s_{i,k} > s_{i+1,k}$ – тогда нам требуется капитализировать букву $s_{i,k}$, и $s_{i+1,k}$ не должна быть капитализированной;
  - $s_{i,k} < s_{i+1,k}$ – тогда должно быть верно, что если мы капитализируем букву $s_{i+1,k}$ когда-нибудь, то и букву $s_{i,k}$ мы тоже капитализируем.

Решение за $O(n + m)$

- Заведём граф, вершинами которого являются буквы.
- Если верно $s_{i,k} > s_{i+1,k}$, тогда пометим вершину $s_{i,k}$ как капитализированную.
- Пусть $s_{i,k} < s_{i+1,k}$, тогда проведём ребро, ведущее из $s_{i+1,k}$ в $s_{i,k}$. Оно будет означать, что если мы капитализировали букву $s_{i+1,k}$, то надо капитализировать и букву $s_{i,k}$.

Решение за $O(n + m)$

- Нужно распространить капитализацию вдоль рёбер.
- Заметим, что граф будет ациклическим, так как рёбра ведут из больших букв в меньшие.
- Можем пройтись с больших букв к меньшим и вдоль рёбер капитализировать буквы.
- После всех капитализаций, нужно проверить, что наши строки идут в лексикографическом порядке, иначе вывести “No”.

Задача H. «Соляной рудник»

Автор задачи: Михаил Пядёркин
Разработчик: Григорий Резников

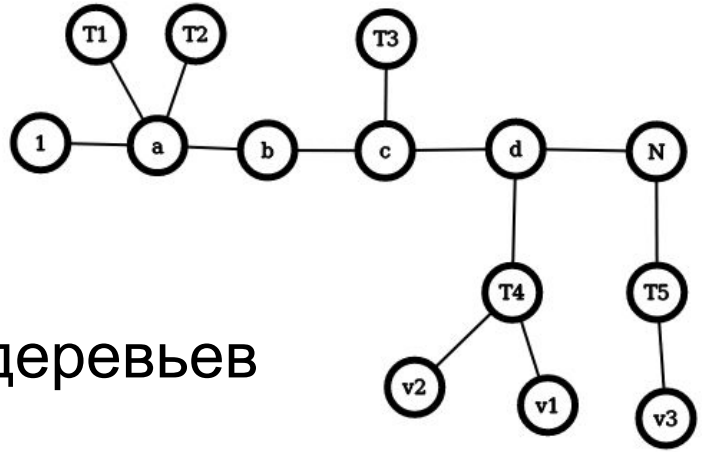
Постановка задачи

- Дано дерево с весами на рёбрах, причём вес ребра зависит от направления движения по нему.
- Найти путь из вершины *1* в вершину *n*, не проходящий ни по какому ребру дважды в одну сторону максимального веса.

Решение (начало)

- Так как граф является деревом, существует единственный простой путь из *1* в *n*. По рёбрам, лежащим на этом пути, придётся пройти один раз, при этом второй раз пройти не получится, потому что иначе Влад не сможет добраться до *n*.
- Будем рассматривать дерево как путь с подвешенными к нему поддеревьями. Внутри поддерева можно пройти по маршруту, начинающемуся и заканчивающемуся в корне поддерева

Пример



1-a-b-c-d-N – Путь в дереве

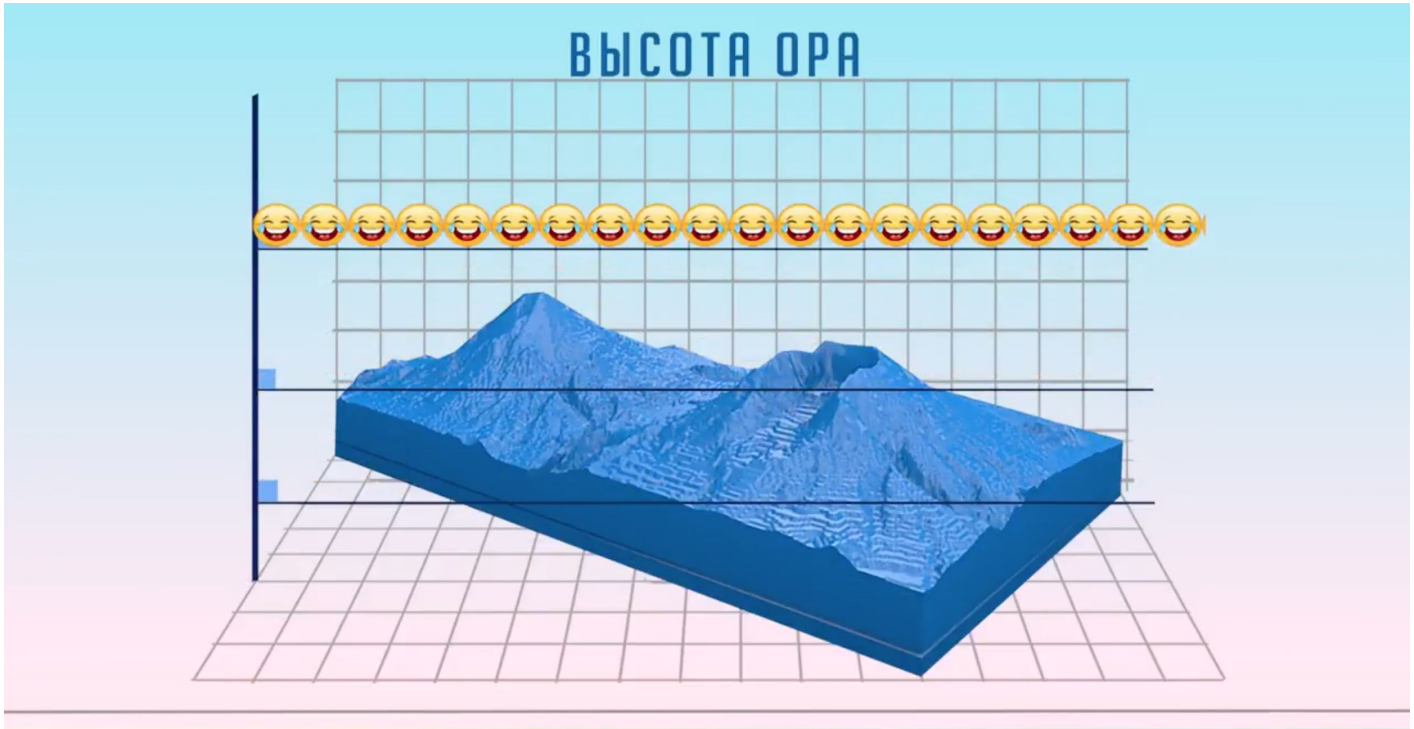
T1, T2, T3, T4, T5 – Корни поддеревьев

v1, v2, v3 – Вершины в поддеревьях

Решение (окончание)

- Посчитаем *dp[v]* – максимальный вес пути в поддереве вершины v, который начинается и заканчивается в *v*.
- *dp[v] = Σ max(0, dp[to] + w(v, to) + w(to, v))*
- Ответ на задачу: вес пути из 1 в *n* плюс сумма значений *dp[v]* по всем поддеревьям, у которых корень лежит на пути.

Задача F. «Ор выше гор»



Автор и разработчик задачи: Олег Мингалёв

Постановка задачи

- Дано n целых неотрицательных чисел $a_1, a_2, \ldots, a_n$.
- Сколько существует подотрезков, побитовое ИЛИ значений на которых больше любого из этих значений?

Решаем за $O(n^3)$

- Назовём искомые отрезки *высокоорными*.
- Назовём *ором* на отрезке побитовое ИЛИ чисел на нём.
- Переберём все подотрезки, каждый проверим на высокоорность за линию: найдём ор на подотрезке, проверим, что он больше каждого из чисел.

Решаем за $O(n^2)$

- Or на отрезке больше любого числа на отрезке тогда и только тогда, когда or на отрезке больше максимума на нём.
- За $O(n^2)$ посчитаем максимум и or для каждого подотрезка:
- $max_{i,i} = or_{i,i} = a_i$
- $max_{i,j} = max(max_{i,j-1}, a_j)$
- $or_{i,j} = or_{i,j-1} \mid a_j$

Решаем за $O(n \log^2 n)$

- Or выше максимума m на отрезке, если $\exists\ x$ на отрезке, такое что x содержит хотя бы один бит, которого не содержит m.
- Почему правда: допустим, нет таких чисел x, что они содержат хотя бы один бит, не содержащийся в m, тогда все числа на подотрезке – битовые подмножества m, и or чисел на подотрезке равен m.

Решаем за $O(n \log^2 n)$

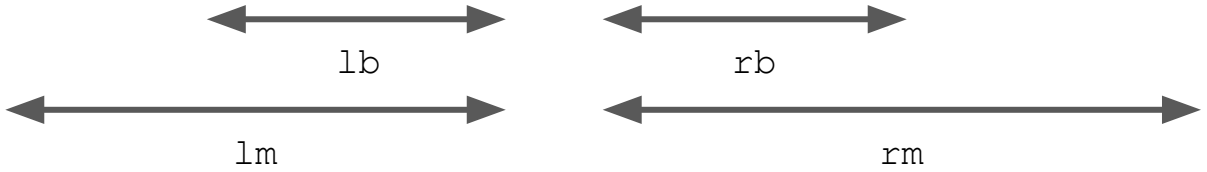
- При фиксированном левом максимуме на отрезке с индексом i все высокоорные подотрезки должны содержать или первое число слева, имеющее новый относительно a_i бит, или правое такое число, или оба.
- При этом подотрезки не должны содержать слева чисел, больших или равных a_i, а справа – больших a_i.

Решаем за $O(n \log^2 n)$

- Построим дерево отрезков для операций максимум и побитовое ИЛИ.
- Запрос максимума и ора на отрезке – $O(\log n)$
- Поиск ближайшего числа слева $\geqslant a_i$ и ближайшего справа $> a_i$ можно реализовать бинпоиском.
- Поиск ближайших чисел, содержащих дополнительные биты, тоже можно произвести бинпоиском.

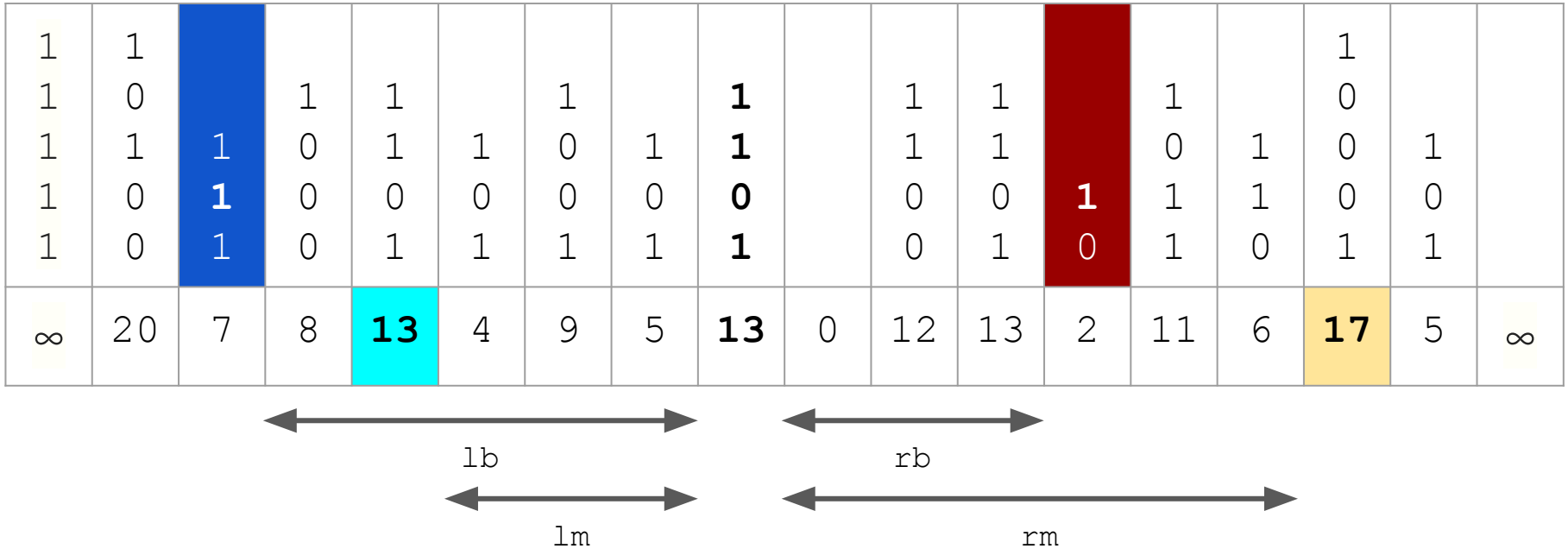
Решаем за $O(n \log^2 n)$

1 1 1 1 1	1 0 1 0 0	1 1 0 1	1 0 0 0	1 1 1	1 0 1	1 0 0 1	1 0 1	**1** **1** **0** **1**		1 1 0 0	1 1 0 1	**1** **0**	1 0 1	1 1 0	1 0 0 1	1 0 1	
∞	20	**13**	8	7	4	9	5	**13**	0	12	13	2	11	6	**17**	5	∞



(lm+1)*(rm+1) - (lb+1)*(rb+1)

Решаем за $O(n \log^2 n)$



(lm+1)*(rm-rb)

Решаем за *O(n log n)*

- Построим разреженные таблицы (sparse table) на максимуме и побитовом ИЛИ.
- По-прежнему используем бинарный поиск.
- Запрос максимума и ора на отрезке – *O(1)*

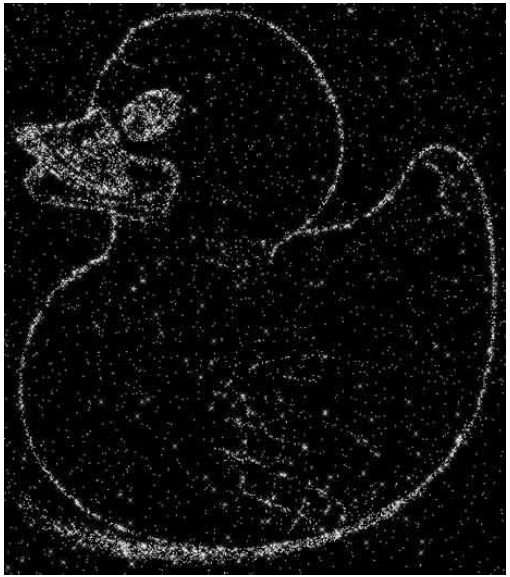
Решаем за $O(n \log c)$

- $0 \leqslant a_i \leqslant c$.
- Научимся за $O(n)$ находить для каждого числа первое слева > (⩾), чем оно.
- Заводим пустой стек, идём слева направо.
- Повторять для каждого числа:
  - Пока стек непустой и число на вершине стека не больше (меньше) удалить число из стека
  - Число на вершине стека – искомое для текущего числа
  - Добавить текущее число в стек

Решаем за $O(n \log c)$

- По каждому биту сделаем проход по всем числам и для каждого числа с нулём в текущем бите найдём первое предыдущее с единицей в этом бите.
- Для каждого числа первое число, содержащее новый бит – ближайшее из посчитанных выше чисел.

Задача B. «Звёздное небо»



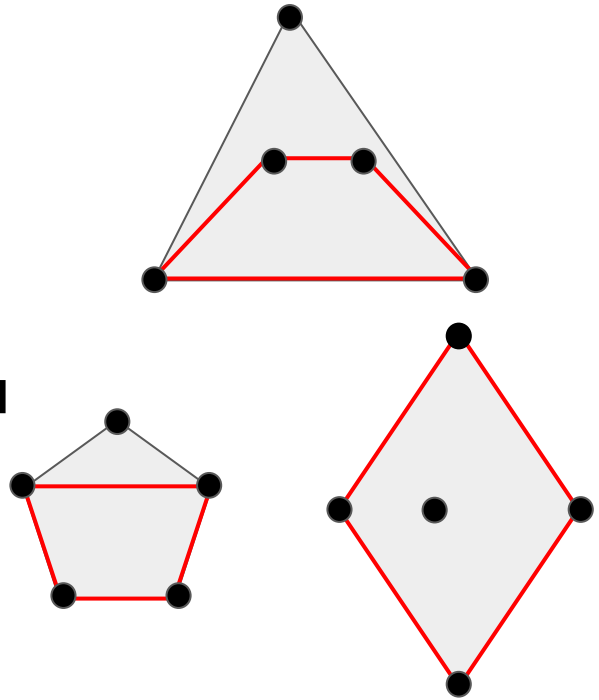
Автор и разработчик задачи: Андрей Чулков

Формальная постановка

- Дано n различных точек общего положения на плоскости.
- Требуется найти среди них k, образующих выпуклый k-угольник.
- k очень маленькое (не превосходит 6).

Ключевая идея

- Среди любых пяти точек есть выпуклый четырехугольник (см. картинки справа).
- Возможно, среди любых x_5 есть выпуклый пятиугольник, а среди любых x_6 – шестиугольник?
- Предположим, что это правда.
- Возьмем столько, чтобы успеть перебрать в TL.



Возможные расположения пяти точек на плоскости

Решение

- Для любого *k* существует такое число *n*, что среди любых *n* точек найдется выпуклый *k*-угольник (Happy Ending Problem, Erdős & Szekeres, 1935).
- А именно, для k = 6 это n = 17 (Szekeres & Peters, 2006).
- Оставим первые 17 точек, если $n > 17$.
- Переберем все *k*-элементные подмножества, проверим, что размер выпуклой оболочки равен *k*.
- Полученное решение работает за $O(C(\min(n, 17), k)\, k \log(k))$. $C(17, 6) \leqslant 17^6/120 \approx 2 \cdot 10^5$.

Задача I. «Курьерский клуб»

Автор задачи: Михаил Тихомиров
Разработчик: Игорь Краскевич

Постановка задачи

- Двум курьерам нужно выполнить доставку пакетов в заданном порядке.
- Каждую доставку должен осуществить ровно один курьер.
- Нужно минимизировать максимальное расстояние между курьерами в процессе.

Решение за $O(N^2)$

- В любой момент времени состояние полностью описывается положением обоих курьеров.
- Можно считать методом динамического программирования величину $f[pos_1][pos_2]$ = минимум максимального расстояния до текущего момента при условии, что курьеры сейчас в позициях pos_1 и pos_2.
- Переход: подвинуть одного из курьеров в следующую позицию $\max(pos_1, pos_2) + 1$, обновить максимум расстояний.

Решение с линейным числом состояний

- Пусть $pos_1 > pos_2$. Если следующую доставку сделает второй курьер, то не важно, где он сейчас стоит.
- Переходы можно делать сразу на несколько позиций вперед и считать, что следующую доставку делает курьер, позиция которого меньше.
- Теперь состояние однозначно задается положением одного курьера.

Шаг назад

- Сделаем бинпоиск по ответу, значение станет "можно ли оказаться в таком состоянии, не превосходя зафиксированного расстояния".
- Из данной точки можно сделать переход вперед на какой-то отрезок.
- Отрезок заканчивается тогда, когда появляется слишком далекая точка. Можно просто бежать вперед и хранить текущее максимальное расстояние.
- Такое решение работает за $O(n^2 \log ans)$.

Решение за $O(n \log n \log ans)$

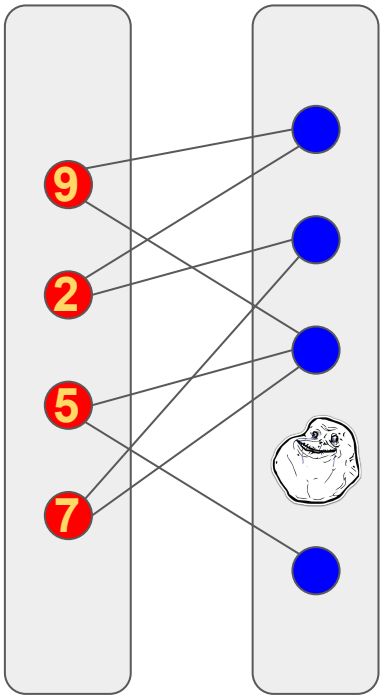
- Чтобы понять, подходит ли отрезок, достаточно знать минимум и максимум на нем
- Найти конец отрезка для перехода можно спуском по дереву отрезков на минимум и максимум
- Теперь все переходы из одного состояния делаются за $O(\log n)$ (достаточно хранить самую большую достижимую позицию)
- Итоговая сложность составляет $O(n \log n \log ans)$

Задача C. «Королевские вопросы»

Автор и разработчик задачи: Вильям Саакян

Формальная постановка

- Дан двудольный граф.
- Каждая вершина левой доли (принцессы) соединена ровно с двумя вершинами (принцами) в правой доле.
- Каждая вершина левой доли обладает весом.
- Необходимо найти паросочетание, максимизирующее суммарный вес покрытых вершин в левой доле.



Решение за $O(n\,(n + m))$

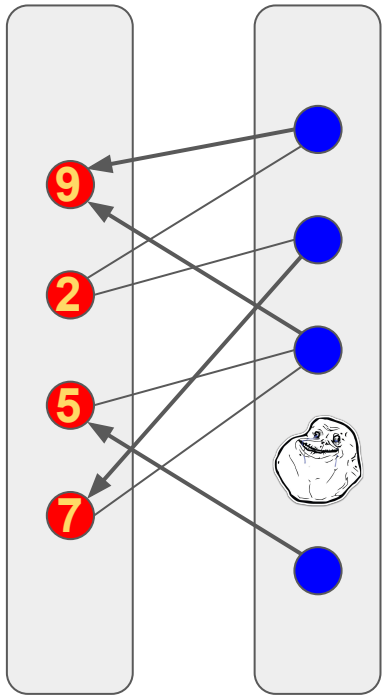
- Выбирать принцесс нужно **жадно**!
- Отсортируем вершину левой доли по их весу в порядке убывания.
- Запустим алгоритм Куна, будем искать увеличивающие цепочки из вершин левой доли в зафиксированном порядке.
- Можно доказать, что полученное паросочетание будет обладать максимальным суммарным весом покрытых вершин левой доли.

Решение за $O(n\,(n + m))$

- Поиск дополняющей цепочки работает за $O(V + E)$, где V – количество вершин в графе, равное $n + m$, а E – количество рёбер в графе, равное $2n$.
- Каждая итерация занимает $O(n + m)$ времени, суммарное время работы – $O(n\,(n + m))$.
- Заслуженный вердикт TL, так как $n \leqslant 200\,000$

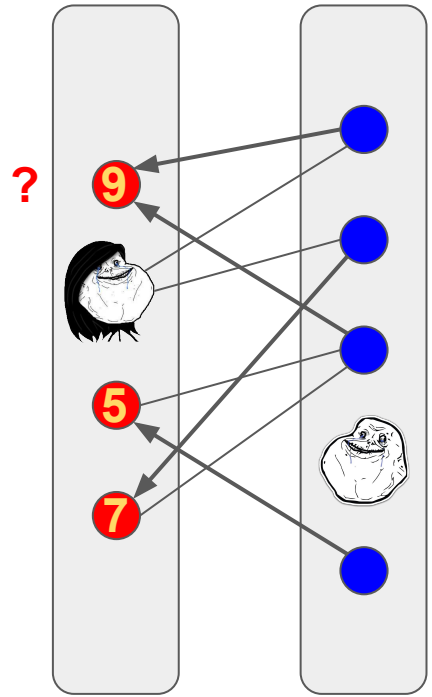
Ключевая идея (жадное назначение)

- Попробуем «назначить» каждой (не изолированной) вершине правой доли самую дорогую вершину левой доли.
- Если получилось паросочетание – мы решили задачу!
- Полученное паросочетание, очевидно, имеет наибольший возможный вес.



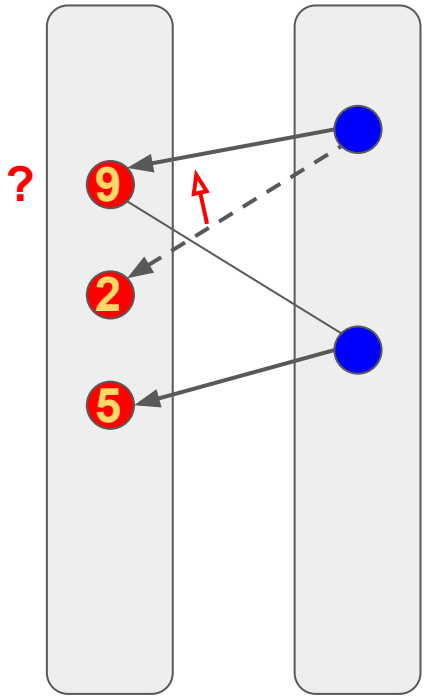
Ключевая идея (жадное назначение)

- Проблема может возникнуть с вершиной левой доли, которой окажутся назначены две вершины правой доли.



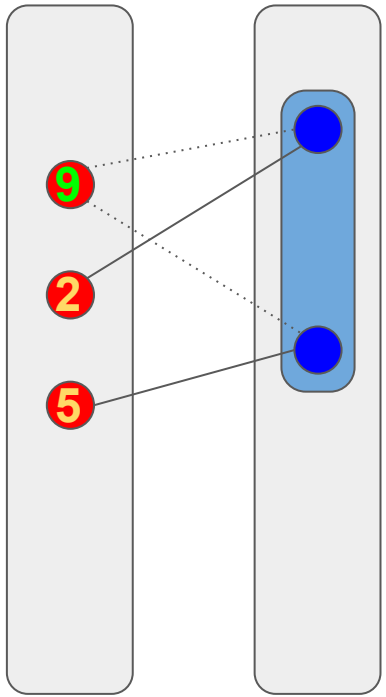
Ключевая идея (популярные вершины)

- Назовём такую вершину *популярной*.
- Утверждение: популярная вершина будет обязательно покрыта искомым паросочетанием.
- Если нет, возьмём любого соседа из правой доли и переназначим его нашей вершине, станет заведомо не хуже, так как популярная вершина была максимальной из его соседок.

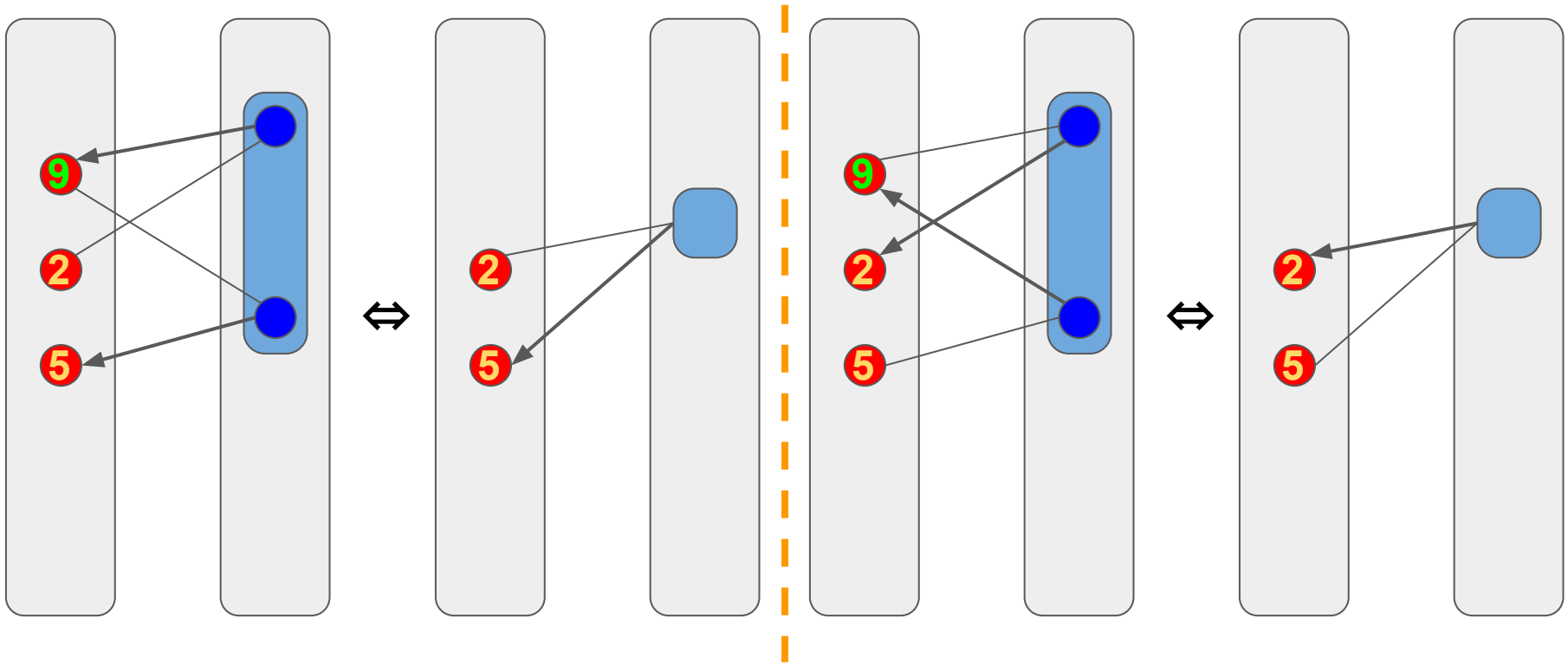


Ключевая идея (эквивалентное преобразование)

- Вес популярной вершины можно сразу прибавить к ответу.
- Удалим популярную вершину из графа.
- Стянем её соседей в одну вершину, соседками которой будут являться все соседки исходных двух вершин.
- Эта объединённая вершина будет символизировать того, кому не достанется популярная вершина.



Ключевая идея (эквивалентное преобразование)

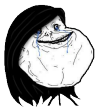


Решение за $O(n \log n)$

- По-прежнему рассматриваем вершины левой доли в порядке по убыванию веса.
- Правую долю поддерживаем в системе непересекающихся множеств (DSU), где каждое множество соответствует одной склеенной вершине.

Решение за *O(n log n)*

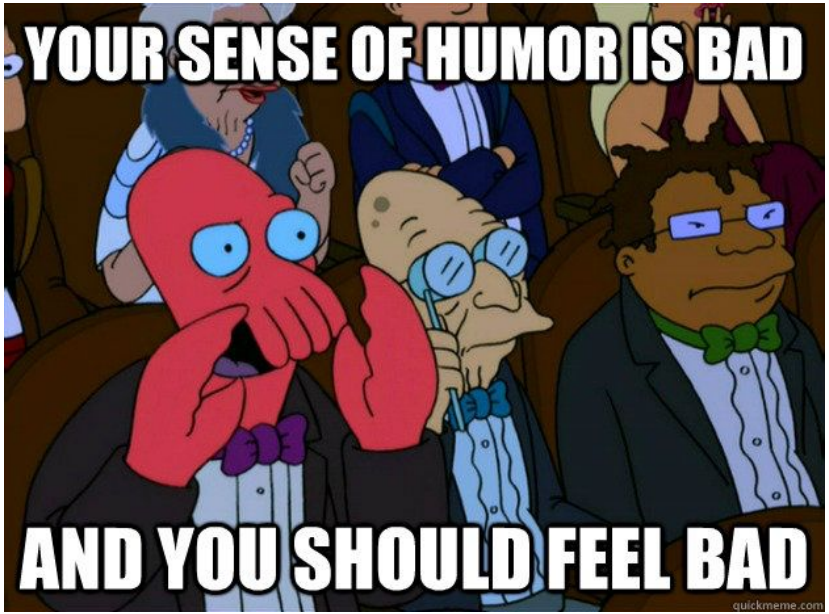
- Рассмотрим очередную вершину. Возможно три варианта.
  - У вершины два соседа в правой доле. Прибавляем к ответу её вес, выкидываем из графа и склеиваем соседей с помощью операции Join в DSU.
  - У вершины один сосед в правой доле. Прибавляем к ответу её вес и выкидываем их обоих.
  - У вершины нет соседей. Выкидываем её из графа.



Решение за $O(n \log n)$

- Полученное решение работает за $O(n \log n)$.
- Возможна альтернативная реализация: можно добавлять вершины правой доли по одной и пытаться жадно назначать им максимальную из соседок в левой доле. Потребуется структура данных для поддержания списка исходящих рёбер из склеенной вершины и вытаскивания максимума (leftish heap).
- Можно адаптировать под случай, когда граф является рёберно-взвешенным.

Задача «Тачка на дачку»



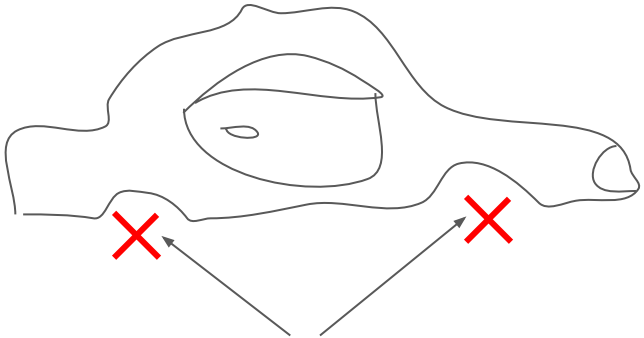
Автор и разработчик задачи: какой-то шутник

Формальная постановка

- Дано 40 квадратных метров стали, два метра проводов, ведро гаек и изображение автомобиля.
- Требуется проверить, можно ли из имеющихся материалов собрать требуемый спорткар.
- Если требуемое возможно, нужно также нарисовать портрет тачки ASCII-артом.

Ключевая идея

- Тачка не поедет без колёс.
- Колёс у нас нет.
- Однако, это не важно: в задаче не требуется, чтобы получившийся автомобиль ездил.
- Значит, можно действовать жадно.



Вот здесь ставим ширму, чтобы никто не заметил отсутствия колёс

Решение

- Объявляем тендер на сборку автомобиля.
- Ждём, пока автомобиль не построится.
- PROFIT!
- Полученное решение работает за
  $O(n \log^2 n$ + *время на поиск подрядчика)*