Открытая олимпиада школьников по программированию 2019

Разбор задач

Задача Polo. «Небоскребы»

Автор задачи: Жюри олимпиады
Разработчик задачи: Андрей Чулков, ВШЭ

Формальная постановка

- Дана матрица размера *n x m* из целых положительных чисел, нужно для каждой пары из строки и столбца независимо решить следующую задачу:
  - Можно изменять числа в столбце и строке на произвольные целые положительные, при условии сохранения относительного порядка в строке и столбце (**независимо**).
  - Нужно минимизировать максимальное число в выбранных столбце и строке

O(1) (разбор случаев, 14 баллов)

- Если $n = m = 1$, то ответ всегда *1*
- Если $\{n, m\} = \{1, 2\}$, то ответ *1* в случае равенства чисел в матрице, и *2* иначе
- Если $n = m = 2$, то для случая справа имеем:
    - ответ *1*, если $a = b = c$
    - ответ *2*, если $a \geq b, c \geq b$ или $a \leq b, c \leq b$
    - ответ *3*, если $a > b > c$ или $c > b > a$

	c
a	*b*

Ключевая идея

- Рассмотрим *i*-ю строку и *j*-й столбец, пусть элемент на их пересечении равен *x*
- Обозначим количество **различных** чисел меньше x в строке как L_{row}, а в столбце как L_{col}
- Аналогично скажем, что число различных больших x в строке — это G_{row}, а в столбце G_{col}.
- Тогда ответ равен ans = $\max(L_{row}, L_{col}) + 1 + \max(G_{row}, G_{col})$
- Действительно, перенумеруем число x в $x' = \max(L_{row}, L_{col}) + 1$, все меньшие числа в различные до $x' - 1$ включительно, а все большие – в различные от $x' + 1$ до *ans* включительно

$O(nm(n+m))$ / $O(nm(n+m) \log (n+m))$ (38 баллов)

- Просто посчитаем числа, описанные в предыдущем слайде, для каждой пары из строки и столбца втупую:
  - Будем проходить каждый раз по строке или столбцу и сравнивать с числом *x* на пересечении
  - Следим за тем, что считаем количество **различных** чисел с помощью хеш-таблицы или любого бинарного дерева (set / unordered_set в C++)

$O(nmA)$ (66 баллов)

- В этой подгруппе числа небольшой величины
- Заранее посчитаем посчитаем множество (без кратных элементов) чисел, для каждой строки и столбца. Размер таких множеств не превосходит A
- Теперь искомые значения $(L_{row}, G_{row}, L_{col}, G_{col})$ для каждой пары из строки и столбца можно находить за $O(A)$, проходясь по предпосчитанным массивам
- Асимптотика:
  - $O(nm + nmA)$, то есть просто $O(nmA)$

$O(nm \log nm)$ (100 баллов)

- Заранее отсортируем, удалим кратные элементы и сохраним в таком виде множество чисел для каждой строки и столбца
- Теперь искомые значения $(L_{row}, G_{col}, L_{row}, G_{col})$ для каждой пары из строки и столбца можно находить двоичным поиском по сохраненным массивам
- Асимптотика:
  - $O(nm(\log n + \log m)) = O(nm \log nm)$

Задача Krusenstern. «Трубка с шариками»

Автор задачи: Максим Ахмедов, Яндекс
Разработчик задачи: Григорий Резников, МГУ, Яндекс

Формальная постановка

- Есть трубка из n секций, в ней *m* шариков и *k* переключателей
- Когда шарик задевает переключатель, тот включается и остаётся таковым навсегда
- За один шаг можно либо подвинуть все шарики влево, либо подвинуть все шарики вправо
- Шарик не может вылететь из трубки или переместиться в секцию, где уже есть шарик
- Нужно включить все переключатели за минимальное число действий
- Либо сначала x раз двигаем влево, а потом y раз вправо, либо наоборот – важное ограничение!

$O(n^4)$, 30 баллов

- Сделаем то, что написано в условии
- Переберём x и y в пределах до n
- Будем эмулировать каждое движение шариков в трубке за $O(n)$
- Тогда проверка каждой пары (x, y) будет за $O(n^2)$
- А значит решение работает за $O(n^4)$

$O(n^3)$, 44 балла

- Обозначим за *opt(x)* минимальное для данного *x* такое *y,* что все переключатели будут включены
- Заметим, что при росте x на 1, оптимальный y либо увеличивается на 1 (если не случилось покрытия какого-то нового переключателя), либо уменьшается.
- А значит можно применить ту же идею, что и в префикс-функции и проверять только $O(n)$ пар *(x, y)*.
- Тогда решение работает за $O(n^3)$

$O(n^2 \log n)$ / $O(n^2)$, 55–76 баллов

- Переберем x
- Определим для каждого шарика позицию, на которую он попадёт за x сдвигов влево – для шарика с номером i это $max(i, pos[i] - x)$ (в 1-индексации)
- Для каждого выключателя, который не был включен за это время, найдём ближайший справа шарик
- Тогда y равен максимальному расстоянию от выключенного выключателя до ближайшего справа шарика
- Это решение работает за $O(n^2 \log n)$ или $O(n^2)$ в зависимости от реализации

O(n log n), 100 баллов

- Будем двигаться по возрастанию *x* и поддерживать оптимальный *y*
- Для каждого выключателя запомним минимальный *x*, при котором он будет включён во время движения влево
- Для тех выключателей, которые не будут включены во время движения вправо, ближайший слева шарик будет таким же, который был ближайшим при *x = 0*
- Будем поддерживать ещё не включённые переключатели в std::set и выбирать максимальный
- Получили решение за *O(n log n)*

Задача Bering. «Спички детям не игрушка»

Автор задачи: Глеб Евстропов, НИУ ВШЭ, Яндекс
Разработчик задачи: Дмитрий Саютин, ИТМО

Формальная постановка

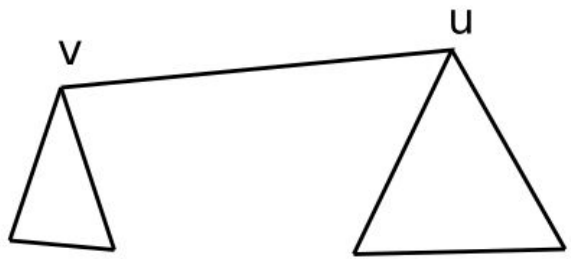
- Есть дерево, в каждой вершине записано число
- Поджечь дерево: в каждый момент сгорает лист с минимальным приоритетом
- Запрос “присвоить вершине максимальный приоритет”
- Запрос *when[v]* — выяснить когда сгорит вершина *v*
- Запрос сравнения *when[v]* и *when[u]*

$O(q_{up} n \log n)$ (9 + 12 баллов)

- В данных группах после каждой операции изменения приоритета выясняем новый порядок сгорания за $O(n \log n)$
- При сгорании дерева множество его листов нужно держать в структуре данных, позволяющей быстрое извлечение минимума (очередь с приоритетом aka std::priority_queue, сбалансированное BST aka std::set)
- Обозначим за q_{up} количество запросов типа up
- Тогда такое решение работает за $O(q_{up} n \log n)$

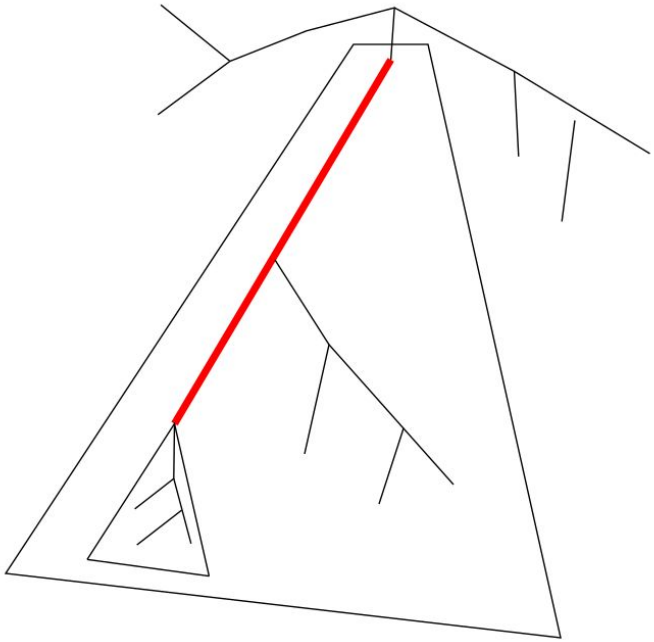
O(q log n) (+23 балла, без “when”)

- Рассмотрим мысленно процесс сгорания
- Чтобы выяснить, кто сгорит раньше, нужно сравнить максимум приоритета в двух поддеревьях
- Реализовать это можно с помощью дерева отрезков на эйлеровом обходе дерева



O(q log n) (+23 балла, без “when”)

- В случае вертикального пути есть небольшое отличие
- Одно из поддеревьев превращается во "всё кроме поддерева"



Ключевая идея

- Как меняется порядок сгорания после операции “up”?
- На самом деле, очень просто: заметим что путь от старого максимума до нового сгорает последним.
- Итого: всё кроме этого пути сгорает в том же порядке, что и до этого; после чего сгорает путь в порядке от старого максимума к новому

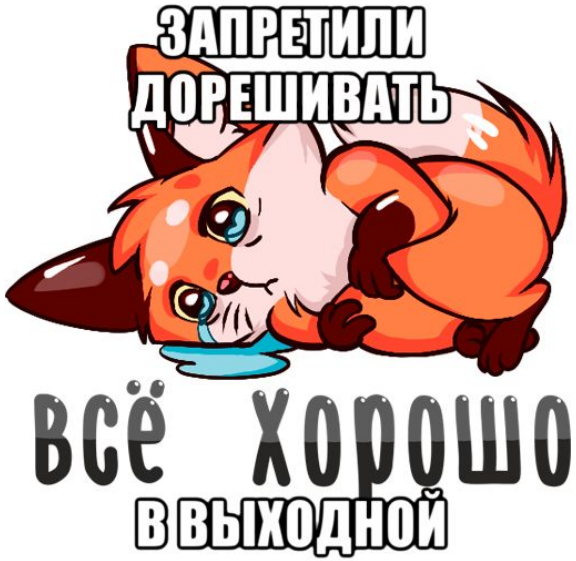
$O(q \log^2 n)$ (100 баллов)

- Давайте реализовывать только операцию when; операция compare реализуется как два запроса when.
- Давайте скажем, что up красит путь в цвет i (и i++)
- Чтобы узнать when[v], для начала узнаем цвет вершины
- when[v] это количество вершин меньших цветов плюс количество вершин того же цвета, но которые сгорят раньше
- Последнее это количество вершин на пути от старого максимума до нового в запросе, который породил этот цвет

$O(q \log^2 n)$ (100 баллов)

- Чтобы реализовать покраску на пути нужно Heavy-Light-Decomposition
- Подгруппа на бамбук: декомпозиция выглядит как один путь
- Внутри пути храним set отрезков подряд идущих одного цвета
- Количество вершин с меньшим цветов: фенвик по цветам
- Технические детали: нужно ещё учесть изначальный порядок до всех up. Но так как вершина не более одного раза меняет цвет с нулевого на ненулевой то можно это делать за O(количество таких вершин)

Задача Gagarin. «Расписание смены»



Автор и разработчик задачи: Александр Курилкин, ВШЭ, Яндекс

Формальная постановка

- Даны две бинарные строки — *s* и *t*
- Необходимо найти такую анаграмму *s* (перестановку символов), чтобы количество вхождений *t* в нее было максимально

Любая сложность (16 баллов)

- Напишем любое переборное решение
- Строку будем проверять за $O(|s|^2)$, наивно проверяя каждое возможное вхождение за линейное время
- Переберём все возможные перестановки *s* (на C++ – например, с помощью **std::next_permutation)** и найдём среди них оптимальную – $O(C(|s|,|s|/2) \cdot |s|^2)$
- Переберём перебрать все битовые маски длины |s| из нужного количества единиц и нулей и выбрать из них наилучшую – $O(2^{|s|} |s|^2)$

$O(|s|^3)$ (43 балла)

- Подгруппа для оригинальных полиномиальных решений от отдельных умельцев :)
- Заметим, что нам не важен точный вид исходной строки *s*, важно только количество единиц *a* и нулей *b* в ней
- Считаем динамику *dp[i][j][k]*:
  - значение – максимальное возможное количество вхождений *t* в текущую строку *current;*
  - в текущей строке *i* единиц и *j* нулей;
  - *k = π(t + '#' + current),* то есть, *k* есть длина максимального суффикса строки *current*, являющегося префиксом t;
  - ответ на задачу – максимальное из значений *dp[a][b][·]*
  - переходы с использованием значений префикс-функции *t*

$O(|s|^2)$ (79 баллов)

- Разовьем идею с префикс-функцией из предыдущей подгруппы
- Будем конструировать ответ, дописывая по одному символу
- Если $k = x < |t|$, то всегда выгодно дописать символ, который даст нам $k = x+1$
- Пусть выгодно обратное – мы обязаны будем вернуться когда-нибудь в состояние с $k = x$, и мы в нём окажемся с не меньшими значениями i и j
- Иными словами, прогресс терять не выгодно

$O(|s|^2)$ (79 баллов)

- Если же префикс-функция уже была равна $|t|$ (получили очередное вхождение), то выгодно дописать тот символ из 0 и 1, после которого её значение будет максимально возможным
- Объясняется той же логикой
- Найдём наибольший бордер строки t – префикс, совпадающий с суффиксом
- Пусть его длина k, тогда оптимальным символом для дописывания будет $t[k]$ (в индексации с нуля)
- Найдем длину бордера за $O(n^2)$, просто перебрав её и проверив каждый вариант за линейное время

$O(|s|)$ (100 баллов)

- Найдем длину наибольшего бордера не за $O(n^2)$, а любимым из доступных строковых алгоритмов за $O(n)$ — z-функцией, префикс-функцией или хешами

Задача Magellan. «Выбор вагона»

Автор и разработчик задачи: Евгений Шемчик, НИУ ВШЭ, Яндекс

Формальная постановка

- Дан дек длины до 10^9, изначально заполненный нулями
- Приходят запросы на добавление блоков нулевых элементов в начало или конец
- Также есть запросы на прибавление арифметической прогрессии с положительным шагом ко всему деку
- Требуется после каждого запроса вывести индекс и значение ближайшего к голове дека из минимальных элементов

$O(ms)$, $O(m \cdot m_{12})$ (27 или 54 балла)

- Для прохождения первой группы тестов необходимо просто промоделировать процесс, описанный в условии
- Данное решение работает за $O(ms)$, где s – суммарный размер всех блоков в поезде (включая исходные вагоны)
- Заметим, что в исходном и каждом добавленном блоке элементов нас интересует только первый, так как при всех последующих операциях он будет оставаться минимальным по величине (а из таких – ближайшим к голове)
- От каждого блока оставим только первый элемент, получим решение за $O(m \cdot m_{12})$, где m_{12} – число запросов первого и второго видов

Ключевая идея

- Рассмотрим множество всех индексов, которые могут быть оптимальными
- Оно расширяется только при добавлении блока в конец поезда
- Заметим теперь, что при хранении кандидатов в порядке расположения блоков, значения, соответствующие блокам, будут убывать

$O(m_{12}^2 + m)$ (73 балла)

- Будем хранить префиксные суммы базы и шага прогрессии ($baseSum_{time}$ и $stepSum_{time}$), а также время появления каждого блока t_i
- После каждого добавления блока в конец дека будем проходить во всем блокам, строя список кандидатов
- Добавляя блок длины k в начало, вычесть из суммарной базы прогрессии величину суммарного шага, умноженного на длину добавляемого блока, также необходимо пройти по всем блокам, чтобы подсчитать для каждого из них величину, корректирующую изменение первого элемента суммарной прогрессии:
  $baseSum_{time} = baseSum_{time - 1} - k \cdot stepSum_{time}$
  $correct_i = k \cdot (stepSum_{time} - stepSum_{t[i]})$

$O(m_{12}^2 + m)$ (73 балла)

- Прибавляя прогрессию, достаточно добавить её базу и шаг к частичным суммам
- После каждого запроса будем удалять с конца списка кандидатов лишние элементы, если такие появляются, и выводить последний элемент этого списка
- Получаем решение за $O(m_{12}^2 + m)$

O(m) (100 баллов)

- Заметим теперь, что нам нет необходимости каждый раз перестраивать список кандидатов
- Мы можем действовать аналогично алгоритму построению выпуклой оболочки со стеком
- При добавлении же в начало, можно оставлять в списке кандидатов только добавленный блок, что делает корректировку базы прогрессии ненужной
- Итого получаем решение за линейное время: *O(m)*

Задача Dezhnev. «Путешествие по музеям»

Автор задачи: Егор Чунаев, МГУ, Яндекс
Разработчик задачи: Даниил Николенко, НИУ ВШЭ

Формальная постановка

- Дан ориентированный граф
- Перемещение по ребру занимает *1* единицу времени
- Известно, в какие моменты каждая из вершин активна
- Для каждой вершины моменты, когда она активна, повторяются с периодом *d*
- При посещении вершины, когда она активна, мы получаем *1* очко, но не более одного раза от каждой вершины
- Можем перемещаться по графу столько, сколько необходимо, возможно, посещая некоторые вершины несколько раз
- Максимизировать полученный бонус

$O(2^n \times n^2 \times d)$ (9 баллов)

- $n \le 10, d \le 10$
- Можем посчитать *dp[mask][t][last]*
- *dp[mask][t][last]* = *true*, если можно посетить музеи из маски *mask* так, что сейчас момент времени *t* (по модулю *d*), и мы находимся в вершине *last*
- Такая динамика насчитывается с помощью BFS
- $O(2^n \times n^2 \times d)$

Ключевая идея

- Рассмотрим граф, где каждая вершина задаётся парой *(v, t)*, где *v* — вершина исходного графа, *t* — момент времени (по модулю *d*).
- Если в исходном графе было ребро *(u, v)*, в новом графе должно быть ребро из *(u, t)* в *(v, (t + 1) % d))* для всех *t* от *0* до *d - 1*.
- Про каждую вершину мы теперь знаем, работает ли там музей или нет. Однако есть проблема: мы не должны учитывать несколько раз музей, посещённый в разные дни

O(md) (каждый музей открыт ровно 1 день)

- В этой группе проблемы с учётом музеев, посещённых несколько раз в разные дни недели, нет
- Задача сводится к стандартной
- Есть ориентированный граф, где у каждой вершины задан вес (в нашем случае *0* или *1*)
- Необходимо найти путь (возможно, не простой) с максимальным весом, притом вес каждой вершины учитывается ровно *1* раз
- Научимся решать эту задачу за время, пропорциональное размеру графа

O(md) (каждый музей открыт ровно 1 день)

- Построим конденсацию графа
- Вес компоненты сильной связности равен сумме весов её вершин
- Теперь задача свелась к той же с дополнительным условием, что граф ацикличен
- На ациклическом графе задача решается простым динамическим программированием
- *dp[v]* — максимальный вес, если мы начинаем путь в вершине *v*
- Получили решение за *O(md)*

$O(md)$ ($d = 2$, 13 баллов)

- Построим конденсацию графа
- Заметим, что какие-то компоненты двудольные, а какие-то нет
- В недвудольных компонентах можно посетить любую вершину с любым остатком
- *dp[v][t]* — ответ, если начать в компоненте сильной связности v с остатком t

Полезное наблюдение

- Пусть есть путь из (v, t_1) в (v, t_2), тогда есть и путь из (v, t_2) в (v, t_1)
- Действительно, рассмотрим путь из (v, t_1) в (v, t_2)
- Он соответствует какому-то циклу в исходном графе
- Представим, что мы стоим в вершине (v, t_2)
- Пройдём ещё $d - 1$ раз по этому циклу
- Это всё равно, что пройти d раз по циклу, начиная в вершине (v, t_1)
- Следовательно, после этого мы окажемся в $(v, (t_1 + d \times l) \% d)$, где l — длина цикла
- Однако $(v, (t_1 + d \times l) \% d) = (v, t_1)$ вне зависимости от l

Полное решение

- Из наблюдения следует, что вершины, соответствующие одному музею, либо находятся в одной компоненте сильной связности, либо вовсе не достижимы друг из друга
- Решим всё ту же подзадачу, что и в решении за $O(md)$, но теперь вес компоненты сильной связности равен количеству различных музеев, находящихся в ней
- Получили решение за время $O(md)$

Замечания

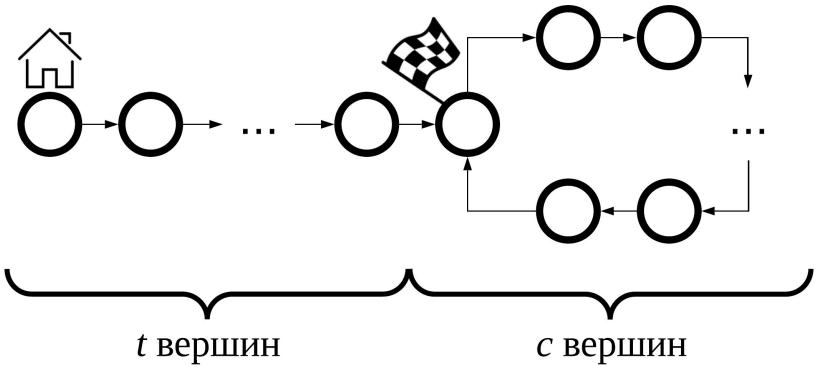
- Не стоит явно хранить граф такого большого размера
- Достаточно использовать списки смежности исходного графа
- Для большего быстродействия рекомендуется использовать массивы вместо векторов
- Группа на 12 баллов *(n ≤ 100, d ≤ 10)* рассчитана на решения, использующие неэффективный алгоритм поиска компонент сильной связности
- Возможно получить полное решение на основе решения для *d=2*

Задача Cook. «Кооперативная игра»

Автор и разработчик задачи: Олег Мингалёв, МАИ

Постановка задачи

- Загадан ориентированный граф следующего вида:



- В стартовой вершине находятся 10 маркеров
- За ход можно передвинуть любое подмножество маркеров, в ответ вы получаете информацию про то, какие маркеры оказались в одной вершине, а какие — в разной

Постановка задачи

- Программа должна уложиться в неизвестное ей заранее число запросов, но известно, что это ограничение не менее чем в три раза превосходит количество вершин в загаданном графе

Используем открытые тесты

- Общая идея: ходить разными маркерами по графу, пытаясь вычислить загаданный граф из набора заранее известных
- Пример: заметим, что параметр *c* за редкими исключениями уникален для теста
- Для того, чтобы узнать *c*, нужно привести два маркера в одну вершину на цикле, а затем пустить один из них вперёд до тех пор, пока он не вернётся в вершину к другому
- Чтобы попасть на цикл, можно достаточно долго идти вперёд, но тогда можно не уложиться в лимит запросов

Как дёшево попасть на цикл

- Давайте двигать маркер *FAST* каждый ход, а маркер *SLOW* — каждые два хода
- *FAST* уйдёт вперёд, первым попадёт на цикл и начнёт на нём крутиться
- Когда до цикла доберётся *SLOW*, *FAST* начнёт догонять его, каждые два хода сокращая расстояние на 1, пока они не встретятся
- Исходное расстояние на цикле < c, значит меньше чем за $2c$ ходов после попадания *SLOW* на цикл они с FAST встретятся
- Также *SLOW* потребовалось $2t$ ходов, чтобы дойти до цикла
- Значит, мы можем передвинуть два маркера в вершину на цикле меньше чем за $2c+2t = 2(c+t)$ ходов.

Как этим воспользоваться

- Раз мы умеем за $2(c+t)$ ходов попадать на цикл, то за ещё $+c$ мы можем узнать его размер
- По размеру попытаемся угадать тест и привести все маркеры в нужную вершину
- Такие решения уже получают порядка 70 баллов
- Если смотреть не только на размер цикла, но и на количество ходов, которое потребовалось для встречи *FAST* и *SLOW*, то останется только две пары неотличимых тестов, то есть такое решение набирает 98 баллов
- Отправим его несколько раз в систему, случайно выбирая при необходимости один из неотличимых тестов, и получим 100.

Как решать задачу без открытых тестов

- За $2(c+t) + c$ приведём два маркера в вершину на цикле и узнаем его размер
- Чтобы узнать t, выведем третий маркер из стартовой вершины на c шагов вперёд, а затем будем одновременно двигать этот третий маркер и остальные, поддерживая между ними дистанцию c, тогда они встретятся в требуемой вершине
- Приведём оставшиеся маркеры к итоговой вершине
- Если делать поэтапно — суммарно не более $4(c+t)$ запросов, такие решения набирали порядка 70 баллов
- Если объединить этапы, “склеивая” маркеры, то решение будет укладываться в $3(c+t)$ и набирать 100 баллов

Простое решение на 100 баллов

- Пусть до встречи *FAST* и *SLOW* прошли расстояния *fast* и *slow* соответственно, тогда *fast* = 2*slow*
- Пусть они встретились в вершине на цикле на расстоянии x от финишной, тогда *slow* = *t* + *x*, а *fast* = *t* + **?** · *cycle* + x

$$2t + 2x = t + \mathbf{?} \cdot cycle + x$$

$$t + x = \mathbf{?} \cdot cycle$$

- То есть если FAST и SLOW додвинуть на расстояние *t*, они окажутся в финишной вершине, так же как если двигать оставшиеся маркеры

Простое решение на 100 баллов

```
делать
    next 0 1
    next 0
пока 0 и 1 не окажутся в одной вершине


делать
    next 0 1 2 3 4 5 6 7 8 9
пока все не окажутся в одной вершине
```

Задача Armstrong. «Билеты на поезд»

Автор задачи: Глеб Евстропов, НИУ ВШЭ, Яндекс
Разработчики задачи: Антон Кваша, Артур Петуховский, ВШЭ

Формальная постановка

- Есть поезд из m вагонов со стоимостью одного билета c_i и вместимостью p_i в каждом
- На группу из k детей нужно купить еще 1 билет для сопровождающего
- Нужно купить билеты для n детей как можно дешевле

$O(mn^2)$ 19 баллов

- Решим динамическим программированием "задачу о рюкзаке", в котором мы хотим набрать детей с помощью вагонов
- Тогда мы хотим набрать суммарный вес n используя m элементов рюкзака
- Это легко сделать динамикой, как в обычном рюкзаке, учитывая в пересчете дополнительного сопровождающего
- Это работает за $O(mn^2)$

$O((mk)^2)$, 49 баллов

- Отсортируем все вагоны по возрастанию стоимости билета
- Назовем полным блоком набор из k детей и одного сопровождающего
- Тогда каждый вагон описывается количеством полных блоков и размером остатка, “неполного” блока
- Заметим, что оптимальный ответ заполняет полные блоки от дешевых вагонов к дорогим

$O((mk)^2)$, 49 баллов (продолжение)

- Насчитываем ДП для “неполных” блоков за $O((mk)^2)$
- Перебираем позицию последнего полного блока и обновляем ответ используя просчитанное ДП
- Цикл работает за $O(mk)$

$O(mk^3)$, 77 баллов

- Возьмем в ответ полные блоки, которые точно туда входят
- У нас останется совсем немного блоков под вопросом:
  - $O(m)$ полных блоков которые могут входить в ответ
  - $O(m)$ неполных блоков, по 1 от каждого вагона
- На этих блоках нужно найти ответ для $O(mk)$ детей

$O(mk^3)$, 77 баллов (продолжение)

- Используем жадный подход
- Размером *size* блока будем называть количество детей
- Сортируем блоки по удельной стоимости:
  - $(size + 1) / size \cdot cost$
- Набираем блоки жадно пока не наберется на хотя бы n детей
- Выкидываем k блоков из набранных для каждого *size*

$O(mk^3)$, 77 баллов (продолжение)

- Выкидываем k блоков из набранных для каждого size
- Почему это верно?
- Поля этого слайда слишком узки, чтобы вместить это доказательство.

$O(mk^3)$, 77 баллов (продолжение)

- Оставшихся $O(k^3)$ детей нужно набрать с помощью ДП на $O(m)$ блоках
- Заметим, что если какой-то из блоков использован в ответе частично, то более дорогие блоки не используются
  - Иначе можно просто перекинуть дорогие билеты в дешевые и ничего не потерять
  - Следовательно, не более одного блока используется частично
- Сортируем по стоимости билета
- Добавляем в ДП блоки полностью за $O(mk^3)$ и пытаемся взять последний блок частично

$O(m \log m + k^7)$, 100 баллов

- Делаем жадность из предыдущей группы
- Нам все еще нужно найти ответ для $O(k^3)$ детей
- Будем обновлять ДП для каждого size отдельно
  - В пределах одного *size* ответ получается жадным набором от дешевых к дорогим
- Итого для каждого *size* обновляем ДП за $O(k^6)$