Разбор задач Московского четвертьфинала NEERC ACM ICPC 2016

Problem A. Altitude

Идея задачи: Андреева Елена Владимировна. Разработка: Олег Христенко.

В данной задаче требовалось в зацикленной последовательности различных чисел $a_1, a_2, \ldots, a_n$ выбрать тройку позиций (i, j, k), такую что $a_i < a_j$, $a_j > a_k$ и сумма расстояний от i до j и от j до k при движении по часовой стрелке минимальна. Ключевым элементом задачи является условие различности всех элементов последовательности, благодаря чему в качестве j всегда можно взять максимальный элемент последовательности, а в качесвте i и k первый перед j и первый после j соответственно, то есть $i = (j-1)\ mod\ n$ и $k = (j+1)\ mod\ n$.

Величина целевой функции в ответе такого вида равна 2, что является минимальным возможным значением, и, следовательно, оптимальным ответом.

Problem B. Blocking Buffer

Идея задачи: Андрей Шестимеров. Разработка: Олег Христенко.

При решении задачи первым делом отметим следующий факт:

- Рассмотрим объем данных, которые записаны в буфер, но еще не считаны от туда. В любой момент времени размер этих данных кратен числу $\texttt{НОД}(r, w)$

Это утверждение верно, поскольку все записанные и считанные блоки кратны этому числу. Пересчитаем входные числа следующим образом:

$r = \frac{r}{\texttt{НОД}(r,w)}, r = \frac{w}{\texttt{НОД}(r,w)}, l = \lfloor \frac{l}{\texttt{НОД}(r,w)} \rfloor$

Теперь r и w `взаимопросты`. Размер буфера l в свою очередь мог быть не кратен $\texttt{НОД}(r, w)$, но этот остаток мы никогда не могли заполнить, поэтому новое l округляем вниз.

Из определения следует, что дедлок наступит если в буфере сейчас находится меньше чем r (то есть не больше чем $r-1$), и в то же время больше чем $l-w$ (то есть не меньше чем $l-w+1$). Получается, что дедлок может наступить только если выполняется следующее неравенство для некоторого x:

$l - w + 1 \le x \le r - 1 \Leftrightarrow l - w + 1 \le r - 1 \Leftrightarrow r + w - 2 \ge l$

Теперь покажем, что если выполняется последнее условие, то дедлок всегда можно достичь. Будем выполнять следующие действия:

- читаем из буфера блоки размера r пока это возможно

- пишем в буфер `ровно один` блок размера w

до тех пока мы не придем в дедлок. Это всегда произойдет, потому что иначе в какой то момент в буфере оказалось ровно $r-1$ (что и есть дедлок по неравенству $r-1+w>l$). Так как r и w взаимопросты и по сути мы просто много раз прибавляем w по модулю r, то в буфере будут достижимы все остатки по модулю r, в том числе и $r-1$

Итого, задача сводится к тому чтобы поделить все входные числа на $\texttt{НОД}(r,w)$ и после этого проверить неравенство $r+w-2\geq l$

Problem C. Catch Me If You Can

Идея задачи: Александр Хохлов. Разработка: Олег Христенко, Дмитрий Жуков.

Пусть Карл изначально находится в точке A, а покемон сидит в точке P.

Карл может ошибиться с первым направлением (идя во внешнюю полуплоскость от границы $3R$), но второй шаг в противоположном направлении точно приведет его к первому результативному движению — попаданию в точку B. Если не приведёт — было выбрано направление касательной к внешней окружности в точке A. Тогда просто движемся в ортогональном направлении (нужная полупрямая с двух попыток угадывается) — точка P находится на этой прямой (так как радиус к точке касания перпендиуклярен касательной).

В случае, если точка B расположена на границе $3R$, то следующий ход снова может попасть не в ту полуплоскость; таким образом, останется два хода, последний из которых уже должен попадать в требуемый радиус от покемона. В случае, если второе направление будет выбрано произвольно, двух ходов может не хватить: если полученная в итоге точка C расположена на границе $2R$, то в некоторых случаях могут остаться два варианта для точки P (которая строится как пересечение серединного перпендикуляра к AB и окружности Аполлония для отрезка BC и соотношения $3/2$) и один ход на проверку.

Поэтому в случае, если точка B расположена на границе $3R$, то следующий ход делается под очень малым углом (у автора это 0.01 градуса); если ответ нулевой, то делаем ход по направлению, симметричному относительно AB. Если полученная точка C расположена на границе $3R$, то строим точку P как пересечение двух серединных перпендикуляров. Если точка C расположена на границе $2R$, то хорда AB находилась достаточно близко к границе $2R$ для того, чтобы точка C была на расстоянии, меньшем $R/10$ от серединного перпендикуляра к хорде AB, так что, двигаясь по нормали к направлению первого хода, попадаем в требуемый радиус.

Если точка B находится на окружности с радиусом $2R$, продолжаем движение в том же направлении, получаем ответ 0 (в случае, если AB — касательная к окружности с радиусом $2R$), 2 или 1. В первом случае есть два возможных направления — ортогональные к AB из точки B, во втором строим пересечение серединного перпендикуляра к BC с окружностью Аполлония для AB и $3/2$, в третьем — пересекаем две окружности Аполлония — для AB и $3/2$ и для BC и $2/1$. Во всех случаях имеем два варианта направления, симметричных относительно AB и оставшихся двух ходов на то, чтобы выбрать нужное, достаточно.

Problem D. Demolition Time

Идея задачи: Василий Астахов. Разработка: Василий Астахов, Дмитрий Жуков.

Строку до выполнения каких-либо удалений мы обозначим как s, после выполнения всех удалений — как t, а шаблон, который мы ищем как h. Последовательность $a_1 < a_2 < \ldots < a_k$ означает позиции элементов для удаления, фунция $f(i)$ означает количество символов до i включительно, которые не будут подвергнуты удалению. Заранее прибавим к ответу всех вхождения строки h в s и все вхождения h в t, не совпадающие с каким-либо вхождением в s. Заметим, что остались только вхождения, которые обязательно образованы какой-то подстрокой s и какой-то подстрокой t.

Обозначим через $z(x)$ длину максимальной подстроки t, начинающейся с позиции x и совпадающей с префиксом h соответствующей длины. Через $p(x)$ обозначим длину максимальной подстроки s, заканчивающейся в позиции x и совпадающей с суффиксом h соответствующей длины.

Аккуратно выпишем все условия, которые необходимы, чтобы пара (x, y) определяла подходящее вхождение:

- Разумеется, длины подходящих фрагментов должны быть достаточно большими, то есть $z(x) + p(y) \geq |h|$.
- Должен существовать такой момент удаления j, что после этого удаления расстояние между элементами в точности равно длине нужно фрагмента, то есть $(f(a_j) - x + 1) + (y - a_j) = |h|$.
- И $z(x)$, и $p(y)$ должны «дотягиваться» до момента j, то есть $f(a_j) - x + 1 \leq z(x)$ и $(y - a_j) \leq p(y)$.

Заметим, что первое условие полностью покрывается третьим, а второе условие может быть заменено на два независящих от j неравенства, поскольку все промежуточные значения длины будут присутствовать. Максимальная длина подстроки с конца в x и y равняется $y - x$, а минимальная — $f(y) - x + 1$.

Теперь совместим ограничения на минимальную и максимальную длину с условиями из третьего пункта. С одной стороны нам подойдут моменты удаления не большие максимального j, такого что $f(a_j) - x + 1 > z(x)$, а с другой это должен быть момент не раньше чем минимальное j, такое что $y - a_j + 1 > p(y)$.

Перегруппировав неравенства из двух предыдущих параграфов мы получим с одной стороны отрезок событий, подходящий для x, а с другой — отрезок событий, подходящий для y, осталось найти количество перекрывающихся пар отрезков, что легко можно сделать эффективно с помощью таких структур данных как дерево отрезков или дерево Фенвика.

Problem E. Economy Printing

Идея задачи: Андреева Елена Владимировна. Разработка: Дмитрий Жуков.

Отсортируем массив номеров страниц, которые нужно напечатать, и разобьём его на блоки. Если номера соседних страниц отличаются более, чем на 2, то такие страницы будут в разных блоках, иначе — в одном. Будем решать задачу независимо для каждого блока.

Пусть блок состоит из K страниц, которые нужно напечатать $(p_l, \ldots, p_r)$. Тогда его можно представить в виде последовательности A длины $L = p_r - p_l + 1$ из нулей и единиц, i-ый

элемент которой равен 1, если страницу с номером $p_l + i$ нужно печатать, и 0 - в противном случае. Заметим, что $L < 2K$.

Рассмотрим, как выглядит ответ. Он представляет собой объединение четырёх множеств печатаемых страниц — множества одиночных страниц, и трёх множеств диапазонов страниц — по одному множеству для каждого вида (все страницы, только чётные, только нечётные). Заметим, что внутри каждого множества диапазоны страниц не пересекаются.

Рассмотрим элемент последовательности A. Пусть его номер i. Тогда в искомом ответе существует не более одного диапазона каждого вида, который начинается в позиции с номером меньше, либо равным i, и заканчивается в позиции с номером больше i. Более того, если в искомом ответе есть диапазон, вида $l-r$, включающий в себя все страницы, и $l \le i$, $r > i$, то не может быть диапазонов вида $i_1\%i_2$ и $i_1\#i_2$ таких, что $i_1 \le i$ и $i_2 > i$. Следовательно, если взять искомый ответ и рассмотреть в нём все диапазоны, которые начинаются в позиции с номером меньше, либо равным i, и заканчиваются в позиции с номером больше i, то возможны всего 5 вариантов множеств типов этих диапазонов:

- пустое множество,
- диапазон вида $i_1\%i_2$,
- диапазон вида $i_1\#i_2$,
- диапазон вида $i_1\%i_2$ и диапазон вида $i_1\#i_2$,
- диапазон вида $i_1 - i_2$.

Отсюда следует решение этой задачи с помощью динамического программирования.

Есть набор состояний, каждое из которых характеризуется парой чисел — длиной i префикса последовательности A и видом множества j, которое образуют диапазоны, которые начинаются в позиции с номером меньше, либо равным i, и заканчиваются в позиции с номером больше i. Для каждого такого состояния нужно посчитать значение, равное минимальному количеству символов, из которых будет состоять строка, которая соответствует заданному множеству страниц для печати. Здесь в случае начавшихся, но ещё не закончившихся диапазонов, будем учитывать только количество символов в левой границе диапазона и разделительный знак.

Есть начальное состояние, соответствующее префиксу последовательности A длины 0 и пустому множеству начавшихся диапазонов. Будем перебирать состояния в порядке увеличения длины префикса i. Далее остаётся только аккуратно выписать все возможные переходы между состояниями — из $(i,\ j)$, где $0 \le i < L, 0 \le j < 5$, в $(i+1,\ j')$. Если $A_i = 1$, то начинаем, заканчиваем или продолжаем некоторый диапазон, либо печатаем одиночную страницу, иначе $A_i = 0$, тогда проверяем, что i-ая страница не попадает в начавшиеся диапазоны. При пересчёте нужно сохранять не только значение длины ответа, но и состояние, из которого пришли при достижении такой длины.

Значение длины ответа для всего блока будет лежать в элементе, соответствующем состоянию $(L, 0)$, — распечатаны все страницы и нет начавшихся, но ещё не закончившихся диапазонов.

Далее остаётся восстановить и распечатать в требуемом формате саму строку ответа.

Problem F. Format

Идея задачи и разработка: Олег Христенко.

Пункты 1 и 2 условия эквивалентны следующей задаче: задать с помощью указанных в задаче правил множество S всех символов, встречающихся в данной строке.

Для решения задачи заметим следующие моменты (везде, где используется отношение порядка на строках, оно введено в соответствии с условиями задачи, то есть сначала идёт сравнение длин, а потом лексикографическое сравнение).

- Два идущих подряд символа задавать как интервал невыгодно — длина будет равна двум вместо трёх.
- Три и более идущих подряд символа задавать как интервал выгодно всегда. Если количество символов больше трёх, то в случае задания интервалом длина будет меньше; если количество символов равно трём, то код минуса 0x2C меньше любого из разрешённых кодов, кроме пробела, и тем самым меньше любого из кодов «среднего» символа.
- При построении описания интервала, начинающегося с некоторого разрешённого символа A_n с кодом C_n, надо выводить в качестве стартового символа интервала символ с кодом $C_{n-1}+1$, где C_{n-1} — код предыдущего алфавитноцифрового символа. Действительно, в этом случае множество описываемых алфавитноцифровых символов остаётся тем же, а форматная строка получается наименьшей.
- После этого строим форматную строку F_1 — описание множества S как совокупности интервалов и отдельных букв и форматную строку F_2 — описание дополнения множества S до множества всех алфавитноцифровых символов, добавляем ко второй строке знак '^' и сравниваем эти строки в соответствии с требованиями задачи.
- Отметим, что если S совпадает с множеством всех алфавитноцифровых символов, то, так как пустая строка не разрешена, требуется взять наименьшую непустую строку, состоящую из неалфавитноцифровых символов. Такой строкой является строка, составленная из одного восклицательного знака, соответственно, форматная строка будет иметь вид '^!'

Problem G. Great Guest Gathering

Идея задачи: Александр Хохлов. Разработка: Олег Христенко и Александр Хохлов.

Представим исходные тарелки с пиццами как вершины графа, а перекладывание куска с одной тарелки на другую — как ориентированная дугу в этом графе.

Исходная ситуация отличается от финальной тем, что из каждой тарелки перекладывается $n-1$ кусок пиццы на $n-1$ другую тарелку. Таким образом перекладываний будет не меньше чем $(n-1)\cdot n+2$, где два дополнительных хода — это первый ход перекладывания первого куска на дополнительную тарелку и последний ход перекладывания этого же куска на его финальное место. Таким образом оценка сверху количества ходов найдена, осталось постоить искомое решение, которое требовалось вывести в задаче.

Заметим, что за исключением первого и последнего хода все остальные ходы взаимосвязаны между собой — пустое место последовательно "гуляет" по тарелкам, а последовательность перекладываний кусков пицц формируют путь в ориентированном графе (правда направление будет противоположным ориентации перекладываний). Заметим также, что наш ориентированный граф — полный, без петель и повторов. Таким образом задача поиска оптимального решения за $(n-1) \cdot n + 2$ ходов сводится к построению Эйлерова цикла в полном ориентированном графе (плюс два хода с дополнительной тарелкой).

Построение Эйлерова цикла в полном графе можно было написать либо конструктивным рекурсивным образом для данного случая полного графа, либо применив общий алгоритм построения Эйлерова цикла в ориентированном графе.

Problem H. Hockey Cup

Идея задачи и разработка: Александр Калужин

Для решения данной задачи достаточно было перебрать результат последней встречи и рассчитать итоговую таблицу для каждого такого результата. Каким количеством забитых шайб можно было ограничить результат последней встречи? У жюри есть тест где это число 29, но можно было ограничить с запасом, например, 100 шайбами. Также при переборе результата необходимо было не забыть, что у овертайма разница забитых шайб ровно 1.

После того, как счет последней встречи зафиксирован, начинаем рассчитывать итоговую таблицу. Для этого для каждой команды рассчитаем ее показатели: очки, победы, победы в основное время, разница заброшенных и пропущенных шайб и, наконец, количество заброшенных шайб. Сначала соритируем по количеству очков, выделяем множества команд с одинаковым показателем. Если множество состоит из одной команды - ее итоговая позиция найдена. Если же из двух - упорядочиваем их по личной встрече, после чего для них также определены итоговые позиции. Если же три или четыре команды имеют одинаковый показатель количества очкой, упорядочиваем их между собой по следующему показателю – победам. И так же, как в первом случае, разрешаем множества из одной и двух команд с одинаковым количество побед. И так далее последовательно повторяем процесс для всех пяти перечисленных показателей.

Если после всех процедур разрешения позиций для трех команд не удалось определить относительный порядок, то считаем что этот порядок мог быть любым (так как при подбросе монетки все команды равны). При этом надо не забыть, что у четвертой команды позиция уже однозначно определена. Также можно заметить, что для описанных правил не может произойти ситуации, что для всех четырех команд их места определяются по монетке (но на решение задачи этот факт не влияет)

Далее исходя из занятой Россией позиции при всех возможных исходах последнего матча определяем, всегда ли команда выходит из группы и может ли она выйти из группы.

Problem I. Interesting Interactive Idea

Идея задачи и разработка: Дмитрий Жуков.

Пусть l_{x_i} и r_{x_i} — ограничения на значение x_i ($l_{x_i} \le x_i \le r_{x_i}$), соответственно l_m и r_m — ограничения на значение m ($l_m \le m \le r_m$). В начале известно, что $l_{x_0} = 0, r_{x_0} = n-1, l_m = 1, r_m = n$. После ответа на очередной запрос длина интервала возможных значений для x_i или

m может уменьшиться. Идея решения — задавать такие запросы, чтобы прийти к ситуации $l_{x_i} = r_{x_i}, l_m = r_m$.

Для начала сделаем так, чтобы выполнялось неравенство $r_{x_i} < l_m$. Для этого возьмём, например, такую последовательность b_k:

- $b_k = 1$, k = 1
- $b_k = min(max(1, \lfloor \frac{\sum_{t=1}^{k-1} b_t}{3} \rfloor), n)$, k > 1

Будем задавать запросы $b_1, b_2, \ldots$, пока после очередного запроса b_p не получим ответ, отличный от '>'. Сделаем после этого то же самое ещё раз — задаём запросы $b_1, b_2, \ldots, b_q$, где b_q — первый запрос, на который ответ — не '>'. Заметим, что на таком наборе запросов ответ '>' всегда означает, что $x_i = (x_{i-1} + a_i) = (x_{i-1} + a_i) mod m$, а другие ответы означают, что $x_i = (x_{i-1} + a_i) - m = (x_{i-1} + a_i) mod m$. Далее также будем задавать только такие запросы, которые не нарушают это свойство. Пусть $p + q = i$. Теперь можно оценить x_i и m так: $l_{x_i} = 0$, $r_{x_i} = b_q - 1$, $l_m = (\sum_{t=1}^{q-1} b_t) + 1$, $r_m = b_p + (\sum_{t=1}^{q} b_t) - 1$. При этом выполняются неравенства $r_{x_i} < l_m$, $l_m > \frac{m}{3}$, $r_m < \frac{5m}{3}$ (т.к. $b_p < \lceil \frac{m}{3} \rceil$, $b_q < \lceil \frac{m}{3} \rceil$).

Далее будет работать алгоритм, на каждой итерации которого будет происходить следующее. Если $r_{x_i} < l_m - 1$, задаём запрос $a = l_m - 1 - r_{x_i}$. Теперь $r_{x_i} = l_m - 1$. Далее два варианта. Если $3(r_{x_i} - l_{x_i} + 1) \le r_m - l_m + 1$, то будем задавать запросы $a = max(1, (r_m - l_m + 1)/3)$, иначе $a = max(1, (r_{x_i} - l_{x_i} + 1)/2)$, пока не получим ответ, отличный от '>'. При этом после каждого ответа на запрос обновляем значения $l_{x_i}, r_{x_i}, l_m, r_m$.

Таким образом на каждой итерации алгоритма за $O(1)$ запросов уменьшаем хотя бы в два раза длину интервала $[l_{x_i}, r_{x_i}]$, либо хотя бы на треть уменьшаем длину $[l_m, r_m]$.

В итоге приходим к ситуации $l_{x_i} = r_{x_i}, l_m = r_m$. По запросам, которые задали, и известным значениям x_i, m восстанавливаем значение x_0 и выводим ответ.

Problem J. Juice Degustation

Идея задачи: Василий Мокин. Разработка: Василий Мокин, Дмитрий Жуков и Глеб Евстропов.

Для решения данной задачи на пригодится следующее утверждение: любой исходный набор из n соков можно разлить в $n - 1$ бочку (если $n > 1$).

Действительно, вычислим количество сока в одной бочке $k = \frac{\sum_{i=1}^{n} c_i}{n-1}$. Заметим, что $\min a_i < k$ и $\min c_i + \max c_i \ge k$, из чего следует, что в одну бочку объёмом k можно полностью поместить сок того типа, которого в наличии меньше всего, после чего долить до величины c соком того, которого в наличии больше всего. Одна такая операция заполняет одну бочку и избавляет нас ровно от одного типа сока (заметим, что $\min c_i + \max c_i > k$ при $n > 2$), при этом у оставшегося набора не изменится значение k. Значит, через $n - 1$ действие все бочки будут заполнены на k, а сок всех типов будет израсходован.

Теперь мы знаем, что ответ на задачу не превосходит $n - 1$, а значит достаточно перебрать количество бочек в ответе x, вычислить величину $k = \frac{\sum_{i=1}^{n} c_i}{n-1}$, после чего (как следует из жадного решения выше) необходимо разбить множество $c_1, c_2, \ldots c_n$ на $n-x$ подмножеств $A_1, A_2, \ldots, A_{n-x}$ так, чтобы в каждом подмножестве A_i сумма его элементов $\sum_{j \in A_i} c_j$ равнялась $k \cdot (|A_i| - 1)$. Другими словами, $k = \frac{\sum_{j \in A_i} c_j}{|A_i| - 1}$.

Описанное выше можно сделать «в лоб» с помощью динамического программирования с перебором подмасок за время $O(3^n\ n^2)$, что является недостаточно эффективным решением. Решить задачу за время $O(2^n\ n^2)$ можно перебрав ответ, а затем вычислив следующие значения динамического программирования: $dp[mask]$ можно ли множество, задаваемое битовой маской $mask$ разбить на количество групп, равное $|mask| - \lfloor \frac{\sum_{i \in mask} c_i}{k} \rfloor$ и какой-то остаток, меньший k. Другими словами, можно ли множество разбить на группы со свойством из предыдущего параграфа и какую-то неполную группу. Обратите внимание, количество групп однозначно определяется неравенствами. Элементы, такие что $c_i = k$, выкинем из рассмотрения, выделив их в отдельную группу. Значение динамики для $mask$ будет *true*, если у какой-либо из подмасок формула выше даёт такое же количество групп, либо количество групп на один меньше, но для $mask$ деление в формуле выполняется без остатка (это означает, что мы как раз набрали новую полноценную группу).

Любопытному читателю предлагается также подумать над решением данной задачи с асимптотическим временем работы $O(2^n\ n)$.

Problem K. Knights of the Old Republic

Идея задачи: Василий Астахов. Разработка: Василий Астахов, Дмитрий Жуков и Глеб Евстропов.

Предположим сначала, что нам необходимо захватить не только все командные пункты, но и все коридоры их соединящие. Тогда заметим, что с одной стороны нам понадобиться как минимум $k = \max(a_1, a_2, \ldots, a_n, c_1, c_2, \ldots, c_m)$ джедаев, а с другой стороны такого количества точно хватит, так как группой такого размера можно просто обойти весь имеющийся граф. Посылать всех k джедаев мы будем в командный центр с самой дешёвой стоимостью отправки, таким образом стоимость решения составит $k \cdot \min(b_1, b_2, \ldots, b_n)$.

Теперь решим задачу, считая что необходимо захватить все командные пункты, а захваченные коридоры должны делать граф связным. Как и в предыдущем случае нам достаточно будет отправить большую группу джедаев в один командный пункт, а размер этой группы будет равен максимуму из всех a_i и такого числа c, что если оставить только рёбра стоимости $\leq c$, то получившийся граф будет связным. Как и до этого, всех солдат отправим в самый «дешёвый» командный пункт, то есть за каждого придётся заплатить минимум среди всех значений b_i.

Используя идеи из предыдущих абзацев мы готовы описать полиномиальное решение задачи. Заметим, что ответ разбивается на компоненты связности, обладающие таким свойством, что любое ребро внутри одной компоненты по весу (то есть по величине c_i) строго меньше, чем любое ребро соединяющее эту компоненту с какой-либо другой. Всего связных подграфов, обладающих таким свойством, линейное количество, так как любые два таких подграфа вложены или не пересекаются (то есть структура вложенности таких подграфов образует дерево). Для каждого из них будем поддерживать две величины $f(G)$ — оптимальный ответ, если подграф связен, и $ans(G)$ — ответ, если подграф разбит на части оптимальным способом. Величина $f(G)$ может быть легко вычислена используя решение из второго абзаца, а величина $ans(G)$ для всего графа будет являться ответом на задачу. Поскольку любые две интересные компоненты вложены или не пересекаются, нет смысла перебирать все способы разбить G на части, достаточно найти наибольшие компоненты, напрямую вложенные в G. Для этого выберем максимальное k, такое что удаление из компоненты всех рёбер $\geq k$ сделает её несвязной. $ans(G)$ положим

равным максимуму из $f(G)$ и $\sum ans(T)$ по всем компонентам из такого «старшего» разбиения.

Осталось лишь придумать, как описанный выше процесс реализовать достаточно оптимально. Заметим, что если рассматривать его от маленьких компонент к большим, то он будет выглядеть аналогично алгоритму Краскала — каждый раз выбирается минимальное ребро c, соединяющее две различные компоненты A и B, после чего они объединяются в компоненту $T = A \cup B$, а величины $f(G)$ и $ans(G)$ пересчитываются:

- $f(T) = \max(\max\limits_{v \in T} a_v, c) \cdot \min\limits_{v \in T} b_v$
- $ans(T) = \min(f(T), ans(A) + ans(B))$

В формулаВ конце мы получим одну единственную компоненту, значение величины ans которой и будет ответом на задачу.

Problem L. Lazy Coordinator

Идея задачи и разработка: Глеб Евстропов.

Последовательно заметим два свойства ответа:

- Пусть в некоторый момент времени у координатора в пуле находятся k предложений. Тогда они неразличимы для него в дальнейшем и математическое ожидание оставшегося времени ожидания равно для всех этих наборов вопросов.
- Количество наборов в пуле не зависит от того, какие наборы будет выбирать координатор, и определяется только разностью количества поступивших и количества использованных на текущий момент предложений.

Двигаясь по событиям от конца к началу (от большего времени к меньшему) будем поддерживать две величины: текущий баланс k, то есть количество предложений в пуле в данный момент и математическое ожидание ξ времени нахождения в пуле ожидания для предложения, которое бы поступило в данный момент. Рассмотрим как будут меняться эти две величины в зависимости от типа запроса:

- При переходе на одно событие назад (в сторону уменьшения времени) величина ξ возрастает на разность координат $x_i - x_{i-1}$.
- Если событие $i - 1$ — появление нового предложения набора вопросов, то выпишем для него в качестве ответа текущее значение ξ. Баланс k уменьшим на 1.
- Если же событие $i - 1$ — выбор одного из предложений для проведения соревнования, то увеличим значение k на 1 и пересчитаем величину ξ умножив её на $\frac{k-1}{k}$ (уже после увеличения k), так как каждый из активных к данному моменту раундов закончился бы с вероятностью $\frac{1}{k}$.