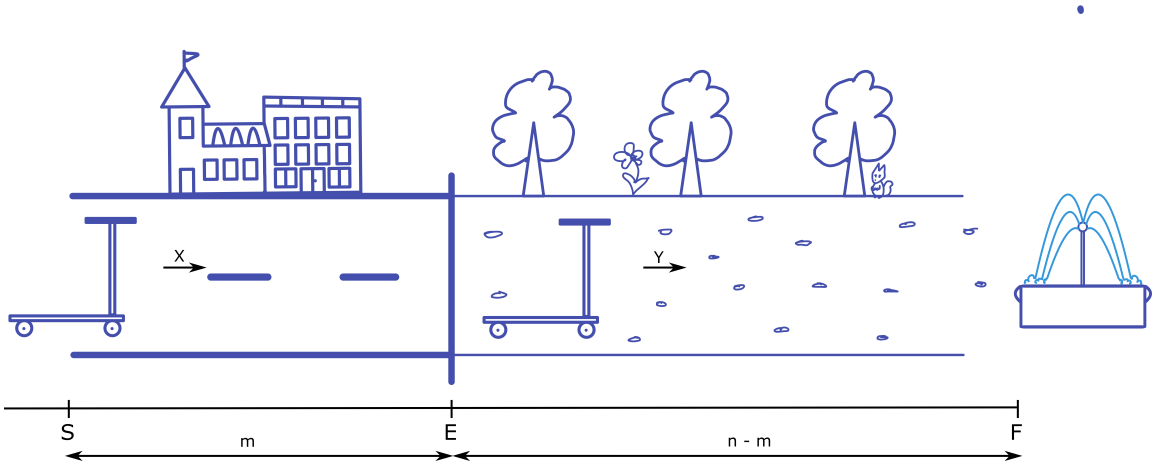
# Задача A. Встреча у фонтана

*Автор и разработчик задачи: Рита Саблина*

Скорость в задаче дана в м/с, а время в минутах, ответ требуется дать в минутах, поэтому переведем значения x и y в м/мин, для этого их нужно умножить на 60. $X$, $X = x \cdot 60$, и $Y$, $Y = y \cdot 60$, — скорость в м/мин электросамоката до входа в парк и после входа в парк, соответственно.

Путь Киры на электросамокате от точки старта до фонтана показан на рисунке.




Рассмотрим первую часть пути — от старта (точка $S$) до входа в парк (точка $E$). Расстояние, которое проехала Кира, — $m$ метров, скорость $X$ м/мин. Таким образом, путь до входа в парк занял у Киры $\frac{m}{X}$ минут.

В момент старта Киры до встречи с Аней оставалось $T$ минут, $\frac{m}{X}$ минут Кира потратила на первую часть пути, на путь по парку, чтобы быть на встрече вовремя, у Киры осталось $T - \frac{m}{X}$ минут.

Рассмотрим вторую часть пути. Кира проедет путь по парку со скоростью $Y$ м/мин, от входа в парк до встречи остается $T - \frac{m}{X}$ минут. За это время Кира проедет $Y \cdot (T - \frac{m}{X})$ метров. С другой стороны, до фонтана осталось проехать $n - m$ метров — стартовое расстояние составляло $n$ метров, $m$ метров Кира преодолела до входа в парк.

Если расстояние, которое Кира проедет за время, оставшееся после поездки до парка, будет больше или равно оставшемуся расстоянию до фонтана, Кира сможет быть на встрече вовремя, и ответ на задачу будет 0. Другими словами, если $Y \cdot (T - \cdot mX) \geqslant (n - m)$, то требуется вывести 0.

Данные значения можно сравнить в целых числах: для этого умножим обе части неравенства на $X$ ($X > 0$ по условию). Если $Y \cdot T \cdot X - Y \cdot m - (n - m) \cdot X \geqslant 0$, то ответ на задачу 0.

Если же оставшегося времени недостаточно для преодоления пути от входа в парк до фонтана, найдем количество минут, которые Аня будет ждать Киру. Чтобы преодолеть оставшееся расстояние $n - m$ по парку со скоростью $Y$ м/мин, потребуется $\frac{n-m}{Y}$ минут. После первой части пути Кире остается $T - \frac{m}{X}$ минут. Значит, искомое значение — $\frac{n-m}{Y} - (T - \frac{m}{X})$. Ответ требуется округлить вверх, для этого приведём дроби к общему знаменателю: $\frac{(n-m)\cdot X + m \cdot Y}{X \cdot Y} - T$. Если числитель делится на знаменатель без остатка, то мы получим правильный ответ при целочисленном делении, если же нет, то получим ответ, округлённый вниз, к нему нужно добавить 1. Частное целых чисел $A$ и $B$ с округлением вверх будет равен результату целочисленного деления $A + B - 1$ на $B$. Если $A \bmod B = 0$, то полученный результат останется тем же, что и при целочисленном делении $A$ на $B$. Если $A \bmod B! = 0$, то полученный результат будет больше на 1. Таким образом, мы получим правильнй ответ.

Итоговая формула: $\lfloor \frac{(n-m)\cdot X + m \cdot Y + X \cdot Y - 1}{X \cdot Y} \rfloor - T$

# Задача B. Взлом навигатора

*Автор и разработчик задачи: Владимир Рябчун*

Обозначим за $d_s[v]$ длину кратчайшего пути из вершины $s$ в вершину $v$, а за $c_s[v]$ обозначим количество различных кратчайших путей из $s$ в $v$. Аналогично определим величины $d_t[v]$ и $c_t[v]$. Кратчайший путь из $s$ в $t$ занимает $d_s[t]$ времени, поэтому если $L < d_s[t]$, то ответ 0, в случае $L = d_s[t]$ ответ будет $c_s[t]$.

Теперь рассмотрим случай, когда $L > d_s[t]$. Предположим, что Денис доехал до перекрёстка $v$ по кратчайшему пути, после чего его направили в перекрёсток $u$ по дороге, которая требует $w_{vu}$ времени. Тогда его путь суммарно займёт $d_s[v] + w_{vu} + d_t[u]$ времени, поскольку после этой дороги путь всегда будет выбираться оптимально. Эта величина должна совпадать с $L$. Кроме этого должна быть возможность доехать до перекрёстка $v$, для этого необходимо и достаточно выполнения условия $d_s[v] + d_t[v] == d_s[t]$. Кроме этого необходимо явно запретить $v$ совпадать с $t$. Если все эти условия выполнены, то хакеры могли направить Дениса по этому ребру, а количество способов доехать от $s$ до $t$, проезжая по дороге от $v$ до $u$, равно $c_s[v] \cdot c_t[u]$.

Для получения ответа на задачу необходимо просуммировать все произведения $c_s[v] \cdot c_t[u]$ по всем дорогам $vu$, которые удовлетворяют описанным выше условиям. Вычислить массивы $d_s$, $d_t$, $c_s$ и $c_t$ можно при помощи алгоритма Дейкстры. Итоговая сложность решения $O(m \log n)$.

# Задача C. Турнир

*Автор задачи: Елена Андреева, разработчики: Михаил Анопренко, Андрей Станкевич*

Отсортируем уровни учеников, пусть $a_1 \leqslant a_2 \leqslant \ldots \leqslant a_n$.

Рассмотрим сначала альтернативную версию задачи, когда в классе четное число учеников и выбирать ведущего не требуется. Докажем, что оптимально собрать такие пары: $a_1+a_n$, $a_2+a_{n-1}$, ...

Пусть $a_1$ в паре с $a_i$, $i \neq n$, посмотрим, с кем в паре $a_n$, пусть с $a_j$.

Тогда: $a_1+a_i \leqslant a_1+a_n \leqslant a_j+a_n$. Аналогично, $a_1+a_i \leqslant a_j+a_i \leqslant a_j+a_n$. Поэтому, заменив пары на $a_1+a_n$ и $a_i+a_j$, мы не увеличим максимум и не уменьшим минимум среди пар. Значит существует оптимальное решение, в котором $a_1$ в паре с $a_n$, применим теперь аналогичное рассуждение для $a_2$ и $a_{n-1}$, и так далее.

Вернемся теперь к решению нашей задачи. Переберем по очереди, кто из учеников станет ведущим. Пусть это ученик с уровнем игры $a_1$.

Тогда по предыдущему рассуждению оптимально собрать пары $a_2 + a_n$, $a_3 + a_{n-1}$, и так далее. Сложим соответствующие суммы уровней в «`std::multi_set`» и выберем минимум и максимум. Это ответ, если выгодно назначить ученика с уровнем игры $a_1$ ведущим.

Пусть теперь ведущий — ученик с уровнем игры $a_2$. Тогда оптимально собрать пары $a_1 + a_n$, $a_3 + a_{n-1}$, и так далее, то есть из мультимножества уровней пар надо удалить $a_2 + a_n$ и добавить $a_1 + a_n$. Обновим ответ, если так поступить оптимальнее, и перейдем к следующему ученику в качестве ведущего.

Так сделаем для всех учеников, общее время работы алгоритма $O(n \log n)$.

# Задача D. Автомобильный туризм

*Автор задачи: Андрей Станкевич, разработчики: Михаил Анопренко, Андрей Станкевич*

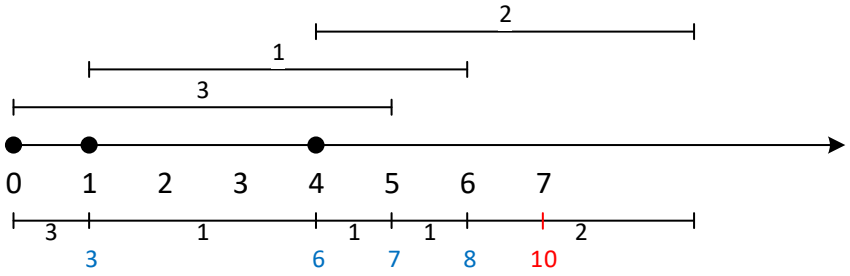
Для решения этой задачи можно применить разные жадные алгоритмы. Рассмотрим один из них.

Для каждой заправки создадим два события: «начало отрезка» для этой заправкив точке $x_i$ и «конец отрезка» этой заправки в точке $x_i + C$. Рассмотрим по очереди эти события. Утверждается, что отрезок между двумя соседними событиями можно ехать на бензине, купленном на одной из заправок, отрезками которой он покрыт, а из них оптимально выбрать самую дешевую.

Это дает следующее решение: будем хранить все цены в `std::multi_set`, обрабатывая событие «начало отрезка», добавим в мультимножество цену на текущей заправке, а обрабатывая «конец отрезка» — удалим. При этом перед изменением мультимножества возьмем из него минимальный

элемент и заплатим столько за единицу бензин для отрезка между текущим событием и предыдущим. Если денег не хватает, либо если доступных заправок нет, то мы нашли самую дальнюю точку, до которой можно доехать.

На рисунке приведены отрезки для заправок из примера, внизу приведены отрезки между событиями и оптимальная стоимость бензина для каджого из них. Синим цветом показан суммарный расход денег к очередному событию, на седьмом километре деньги заканчиваются.



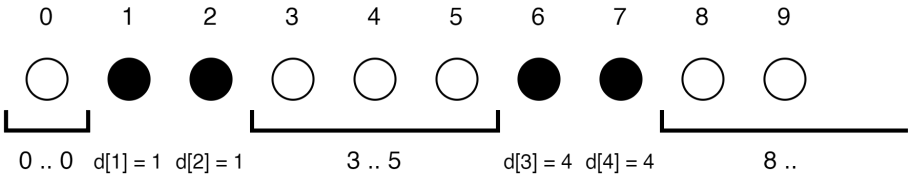

Докажем корректность алгоритма. Для каждого километра дороги, который мы проехали, соединим его отрезком с заправкой, на которой был куплен бензин, использованый на этом километра. Из-за объема бензобака каждая точка покрыта не более чем $C$ отрезками.

Заметим, что можно считать, что не существует двух отрезков $[L_1, R_1]$ и $[L_2, R_2]$, для которых $L_1 < L_2 < R_2 < R_1$, так как их можно заменить на отрезки $[L_1, R_2]$ и $[L_2, R_1]$, количество отрезков, покрывающих каждую точку, не возрастает. Нет отрезка $[L, R]$ с длиной больше $C$, иначе все отрезки, соответствующие километрам внутри него должны начинаться в точке $L$ или левее и точка $L$ покрыта более чем $C$ отрезками. Значит любой километр проезжается на бензине, купленном не более чем за $C$ километров до него, а среди таких решений наш алгоритм строит самое дешевое.

# Задача E. kex

*Автор задачи: Андрей Станкевич, разработчик: Константин Бац*

Рассмотрим, как располагаются числа из множества $A$ и $\text{kex}(A, k)$ на числовой оси. Заметим, что kex–числа образуют отрезки между числами из множества $A$.




Пример расположения чисел множества $A$ для теста из условия.

Занумеруем элементы множества $A$ в порядке возрастания, начиная с 1. Давайте для $j$-го элемента множества $A$ предподсчитаем $d[j]$ — количество kex–чисел, меньших его. Это делается за один проход по элементам множества: $d[0] = 0$, пусть мы знаем $d[j-1]$, тогда $d[j] = d[j-1] + len$, где $len$ – длинна отрезка kex–чисел, который идет между элементами множества $A$ с номерами $j-1$ и $j$. Так же будем считать, что $d[n+1] = \infty$.

Теперь, зная $d[j]$, для любого $k$ мы легко можем найти отрезок kex–чисел, в котором лежит $\text{kex}(A, k)$. Для этого нужно воспользоваться бинарным поиском. Это возможно, так как $d[i]$ не убывает и нам надо найти минимальный $j$ такой, что $d[j] \geqslant k$.

Пусть мы нашли номер отрезка и он равен j, тогда $\text{kex}(A, k)$ является $(k - d[j])$–м по порядку элементом найденного отрезка. Таким образом, $\text{kex}(A, k) = left[j] + (k - d[j])$, где $left[j]$ — левая граница $j$–го отрезка, то есть $A_j + 1$ или 0, если $j = 0$.

Время работы такого решения — $O(n + q \log n)$.

Также существует решение, не использующие бинарный поиск — некоторый аналог двух указателей. Отсортируем $k_i$ по возрастанию и получим такой же массив $d$. Будем последовательно находить

$\mathrm{kex}(A, k_i)$: если $\mathrm{kex}(A, k_i)$ не лежит в текущем отрезке kex–чисел, значит оно и все $k_i$, большего текущего, лежат в следующем отрезке или позже, перейдем к следующему отрезку и посмотрим на него.

Асимптотика такого решения — $O(n + q \log q)$.

# Задача F. Среднее по Бобу

*Автор задачи: Владимир Новиков, разработчики: Владимир Новиков, Виктор Романенко*

Для начала решим данную задачу для случая $q = 1$.

Заметим, что, если мы научимся решать задачу для строки из 0 и 1, то с помощью бинарного поиска, можно свести исходную задачу к строке из 0 и 1, заменив все числа меньшие чем $mid$ на 0, а равные или большие на 1. Если для текущего значения $mid$ можно получить 1 в итоге, то заменим левую границу поиска на $mid$, иначе правую. Тогда ответом будет итоговая левая граница.

Для решения задачи на строке из 0 и 1, введём следующий жадный алгоритм. Будем хранить возможный префикс из какого-то числа 1 и 0 именно в таком порядке. Пусть в текущий момент есть префикс из $x$ единиц и $y$ нулей.

Если очередное число, которое мы рассматриваем, равно 1, то:

- если $y \neq 0$, то можно просто перейти в состояние $x, y - 1$, так как медиана подмассива $[0, 1, c]$ равна $c$.
- если $y = 0$, то переходим в состояние $x + 1, y$.

Если же очередное число равно 0, то переходим в состояние $x, y+1$, если $y \neq 2$, иначе в состояние $x, 1$, так как последние три 0 можно превратить в один.

Пусть итоговый ответ будет 1, если $x = 1$ и $y = 0$ или если $x \geqslant 2$.

Докажем, что данный алгоритм корректен.

Так как, все действия либо увеличивают сумму $x + y$ на 1, либо уменьшают на 1, то из-за нечётности длины отрезка, сумма $x + y$ тоже будет нечётной. Заметим, что если решение возвращает 1, то на исходном массиве можно было получить 1, потому что все действия с парой $(x, y)$ представляют собой какие-то действия с исходным массивом. Осталось доказать, что если можно получить на исходном массиве 1 тогда и алгоритм получит 1, из чего будет следовать то, что алгоритм корректен. Показать это можно с помощью метода математической индукции по длине подотрезка.

Для длины подотрезка $n = 1$ это тривиально.

Пусть верно для всех подотрезков длины $\leqslant k$, докажем для подотрезка длины $k + 2$. Если существует такая последовательность действий, что в итоге получается 1, то рассмотрим последнее действие. Оно было совершено над одним из следующих отрезков длины 3: 110, 101, 011, 111. Каждая из цифр на полученном отрезке является результатом взятия медиан на отрезке длины не более $k$. Тогда по предположению индукции — результат работы алгоритма на каждом отрезке корректен и является одним из следующих значений:

- $(1, 0)$, $(\geqslant 2, 1)$, $(\geqslant 3, 2)$ для единицы.
- $(0, 1)$, $(1, 2)$ для нуля.

Других, вариантов нет, так как $x + y$ нечётно и $y$ принимает только значения от 0 до 2.

Тогда, если в отрезке перед последним действием заменить 0 и 1 на любое из возможных значения алгоритма и на уже полученном массиве снова применить алгоритм, то можно нетрудно убедиться, что получится 1, так как выполнен инвариант, что количество единиц будет хотя бы на 1 больше чем количество 0. Из чего следует, что переход доказан и алгоритм корректен.

Заметим, что в решении можно хранить не более двух единиц. Значит в данном решении есть всего 9 возможных состояний. Причём в итоге ответ будет 1, если получится комбинация $(1, 0)$ или $(2, y)$, где $y$ любое.

Теперь за $O(n \log A)$ мы научились отвечать на один запрос. Сожмём координаты и получим решение за $O(n \log n)$

Чтобы быстро отвечать на много запросов, рассмотрим дерево отрезков, где в вершине будем хранить следующую динамику: для каждого набора $x, y$ из жадного решения храним те значения $x', y'$, которые мы получим, если с изначальными значениями жадного решения равными $x, y$ пройдём по подотрезку, которому соответствует вершина в дереве. Заметим, что мы храним всего 9 значений в каждой вершине.

Чтобы пересчитывать состояния дерева отрезков, достаточно перебрать изначальные $x, y$, после чего по значению динамики левого сына получить $x', y'$, а после по значению динамики правого сына $x'', y''$, что и будет значением динамики в данной вершине для состояния $x, y$.

Теперь, на запрос к отрезку мы умеем отвечать за $O(\log n)$, а так же за $O(\log n)$ менять значение в точке с 0 на 1.

Чтобы ответить на $q$ запросов, применим технику параллельного бинпоиска.

Для каждого запроса сохраним его индекс, границы отрезка $[l, r]$, а так же возможные границы ответа, как в решении для одного запроса $[L, R]$, а так же для каждого индекса от $0, n$ запомним все запросы, на которые надо будет ответить в данной точке. Изначально на каждый запрос надо будет ответить в точке $\frac{0+n}{2}$.

Пройдёмся по массиву, изначально инициализируя его 1, в порядке уменьшения $a_i$, меняя $i$-ю 1 на 0. Ответим на все запросы, на которые надо ответить в данной точке. Если ответом на запрос получится 0, то меняем правую границу запроса $R$ на текущий индекс, а если 1, то левую границу $L$ на текущий индекс, и добавим данный запрос в точку $\frac{L+R}{2}$.

Если сделать операцию выше $\log n$ раз, то на каждый запрос мы ответим хотя бы $\log n$ раз, а значит бинпоиск для каждого запроса сойдётся.

Посчитаем асимптотику. Для каждого запроса мы $O(\log n)$ раз сделаем запрос к дереву отрезков за $O(\log n)$, а значит для всех запрос будет $O(q \log^2 n)$. Так же мы $O(n \log n)$ раз сделаем изменение в дереве отрезков, а значит суммарно будет $O(n \log^2 n)$.

Получаем итоговую асимптотику $O((n + q) \log^2 n)$.

# Задача G. Дорожка вокруг озера

*Автор и разработчик задачи: Михаил Дворкин*

Рассмотрим клетку $A$ — верхнюю клетку озера, а если их несколько, то, например, самую левую из них. Назовём клетку сверху от неё $B$. Клетка $B$ не может быть озером, так как тогда $A$ не была бы верхней клеткой озера, и не может быть землёй, так как земля и озеро отделены газоном, следовательно, клетка $B$ — газон. Докажем, что клетка $B$ точно должна входить в оптимальную дорожку. Пускай это не так, и существует оптимальная дорожка без клетки $B$. Тогда в существует клетка дорожки на луче, идущем из клетки $B$ вверх, ведь если это не так, то от клетки $A$ можно не пересекая дорожку дойти до края парка, просто двигаясь вверх.

Возьмём теперь в новую дорожку вместо всех клеток, располагающихся выше клетки $B$, их проекции на горизонталь, содержащую $B$. Получилась новая дорожка, клеток в которой строго меньше, поскольку в каждом столбце в неё мы взяли одну клетку вместо не менее чем одной, которая была ранее, и есть столбец в котором мы взяли одну клетку вместо не менее чем двух (это столбец, в котором дорожка уходит выше горизонтали, содержащей $B$). Новая дорожка корректна, поскольку на горизонтали, содержащей клетку $B$, нет клеток озера. Получили противоречие. Таким образом, в оптимальной дорожке не могла отсутствовать клетка $B$.

Можно теперь перейти от математической части разбора к алгоритмической. С помощью обхода графа разделим клетки, не принадлежащие газону, на землю — то, что 4-связно с краем парка, и озеро — всё остальное. Среди клеток озера выберем верхнюю $A$ и зафиксируем её соседа сверху $B$. Теперь запустим обход графа в ширину, начиная из клетки $B$, который ходит только по клеткам газона, и которому запрещено ходить влево, если текущая клетка находится на одной вертикали с клеткой $B$ и находится выше неё либо совпадает с ней. Такое ограничение заставит обход в ширину найти кратчайший путь, приходящий в клетку слева от $B$ (легко доказать из условия задачи, что это тоже клетка газона) и при этом окружающий озеро. Клетки этого пути и есть ответ на задачу.

Это решение содержит из трудозатратных частей только два обхода графа с $h \times w$ вершинами, поэтому асимпитотика его работы $O(hw)$.

# Задача H. Одномерная игра

*Автор и разработчик задачи: Алексей Михненко*

Будем смотреть за отрезком $[L, R]$, как за точкой $(L, R)$. Тогда вложенные отрезки это просто все точки ниже и правее данной.

В какие точки мы можем перейти из точки (x, y)? Во все точки $(x_1, y_1)$, что нет другой точки, лежащей в прямоугольнике с углами $(x, y)$ и $(x_1, y_1)$.

Давайте обрабатывать точки в порядке уменьшения координаты $x$ (то есть левой границы), а в случае равенства $x$ координат — в порядке возрастания координаты $y$ (то есть правой границы). Мы будем поддерживать дерево отрезков по оси $y$ координат (предполагаем, что мы сжали все координаты и теперь они не больше $n$). В вершине дерева отрезков, отвечающей за отрезок $[y_l, y_r]$ мы будем поддерживать все точки, в которые мы бы могли перейти из точки $(-\infty, \infty)$, если бы в исходной задаче были только обработанные точки с $y$ координатами из отрезка $[y_l, y_r]$. Если изобразить это на плоскости, то это будет лесенка, проходящая через данные точки, содержащая под собой все остальные точки.

Поддерживать эти точки для каждой вершины мы будем в стеке, где точки хранятся в порядке убывания $x$ координаты от низа до верха стека. Нетрудно заметить, что в этом случае $y$ координаты у точек будут тоже убывать. Тогда что мы должны уметь делать с этим деревом отрезков?

1. Мы должны добавлять очередную точку и пересчитывать значения во всех вершинах дерева отрезков (так как на момент рассмотрения очередного отрезка мы точно обработаем все отрезки, в которые он мог бы переходить). Для пересчёта заметим, что изменится не больше $O(\log n)$ вершин в дереве отрезков, так как только в $O(\log n)$ вершинах будет содержаться очередная точка. А в каждой такой вершине надо просто удалить верхние вершины из стека до тех пор, пока их $y$ координата будет не больше текущей $y$ координаты. После этого надо просто добавить эту точку в вершину стека.

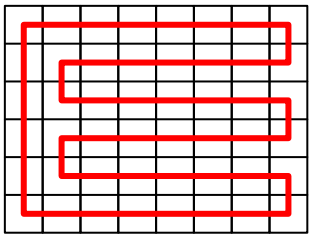
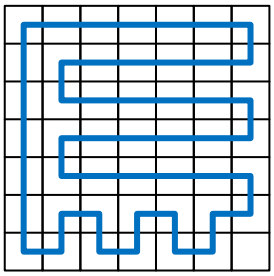
2. Мы должны по очередной точке найти для неё ответ. Пусть очередная точка равна $(x, y)$. Тогда декомпозируем отрезок $[0, y]$ на $O(\log n)$ вершин дерева отрезков (как в обычном запросе к дереву отрезков). Обработаем эти отрезки сверху вниз (по $y$ координате). При обработке очередной вершины можно перейти в какой-то суффикс стека точек, которые лежат в этой вершине. Для определения этого суффикса надо найти первую вершину в стеке, что её $x$ координата строго меньше $x$ координаты последней точки, в которую мы могли перейти в предыдущих (уже обработанных) вершинах (если таких точек ещё не было, то это просто все точки в стеке). Такую точку можно находить бинпоиском. Теперь, чтобы обновить ответ надо просто к нему прибавить сумму ответов по всем точкам на этом суффиксе, а для этого в стеке можно поддерживать префиксные суммы по ответам для точек, что не повлияет на время работы. Так же надо учесть путь, состоящий из одного отрезка, для этого надо просто не забыть прибавить единицу к ответу очередной точки.

Таким образом, получили алгоритм, в котором первая часть суммарно работает не дольше $O(n \log n)$, а вторая часть работает не больше $O(n \log^2 n)$ из-за бинпоиска в каждой вершине, поэтому итоговое время работы равно $O(n \log^2 n)$.

# Задача I. Советские ясли

*Автор и разработчик задачи: Александр Бабин*

Заметим, что в любой таблице $n \times m$ существует простой цикл, который содержит как минимум $nm - 1$ вершину. Такой цикл можно построить конструктивно за время $O(nm)$. Рисунок слева показывает устройство такого цикла, если хотя бы одно из измерений четно, рисунок справа — если оба измерения нечетны.

Пускай стоимость непосещенной клетки равнялась $w_x$ (если цикл посещает все клетки, то $w_x = 0$). Найдем начальную и конечную клетки на цикле, есть две «дуги» цикла, которые их соединяют, пускай сумма стоимостей клеток на одной дуге равна $w_L$, а на другой — $w_R$, при этом оказалось так, что $w_L \geqslant w_R$, тогда утверждается, что путь соединяющий начальную и конечную клетку по соответствующей дуге подходит.

Действительно, суммарная стоимость вершин на таком пути равна $w_a + w_b + w_L$, суммарная стоимость всех остальных вершин — $w_R + w_x$. Легко видеть, что $w_a + w_b + w_L > w_x + w_L \geqslant w_x + w_L$, таким суммарная стоимость вершин на таком пути строго больше $\frac{1}{2}W$.

# Задача J. Строительство пирамиды

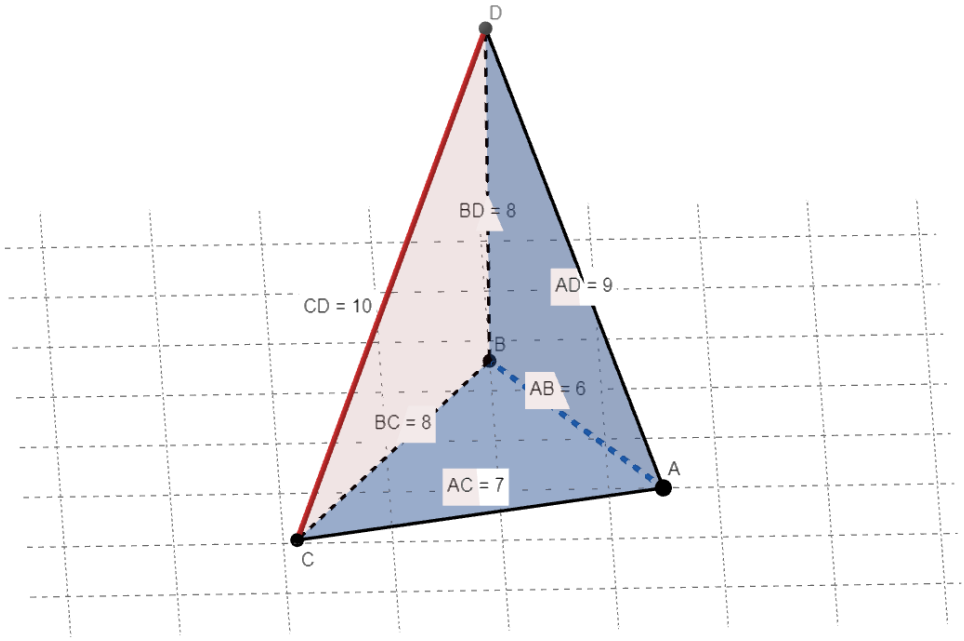
*Автор задачи: Даниил Орешников, разработчики: Даниил Орешников, Андрей Станкевич, Никита Голиков*

В задаче требуется из данных $n$ треугольников выбрать четыре так, чтобы из них можно было составить тетраэдр. Рассмотрим грани произвольного тетраэдра $ABCD$, сгруппируем в пары грани $ABC$ с $ABD$ и $BCD$ с $ACD$.

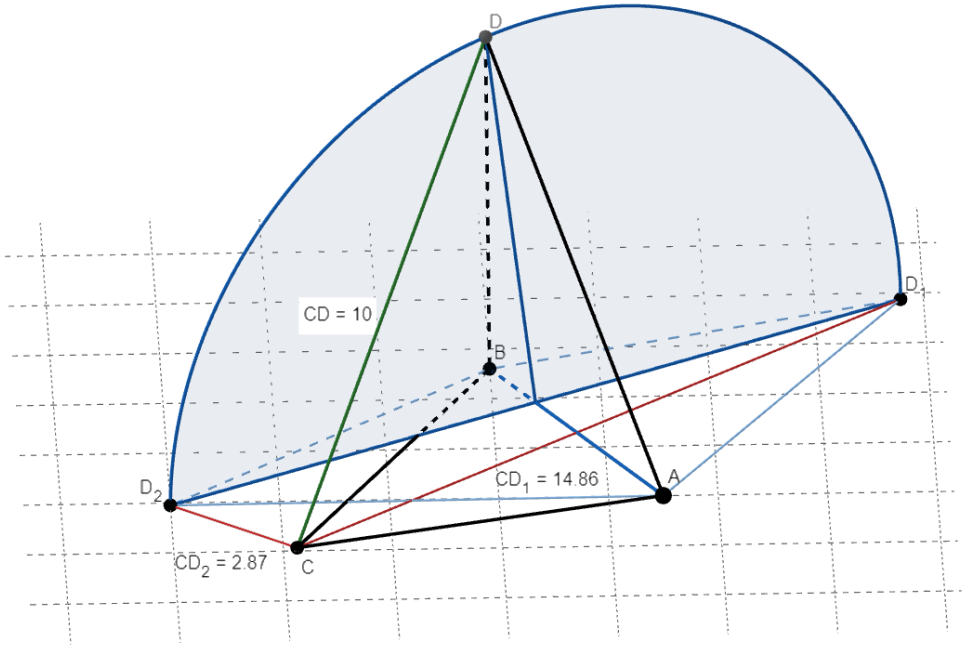
Заметим, что

- если «развернуть» грани в одной паре, чтобы они находились в одной плоскости, получатся два четырехугольника с равными сторонами ($AC$, $BC$, $AD$ и $BD$) и одной из диагоналей, равной $d_1 = |AB|$ или $d_2 = |CD|$, соответственно;
- при повороте $ABD$ относительно «крепления» его с $ABC$ (то есть относительно $AB$) расстояние между $C$ и $D$ в какой-то момент достигает величины $d_2$, ровно в этом состоянии два «согнутых» относительно своих диагоналей четырехугольника можно соединить в тетраэдр.

На следующем рисунке грани $ABC$ и $ABD$ (выделены синим) тетраэдра $ABCD$ имеют внешние стороны 7, 8, 8 и 9. Диагонали $d_1$ (синяя) и $d_2$ (красная) равны 6 и 10, соответственно.




После склеивания этих двух граней по $AB$, при вращении грани $ABD$ относительно $AB$, точка $D$ будет описывать дугу окружности (см. второй рисунок). Минимум величины $|CD|$ достигается в $|CD_2|$, когда оба треугольника лежат в одной плоскости и направлены в одну полуплоскость относительно $AB$, а максимум — в $|CD_1|$, когда они лежат в одной плоскости, но направлены в противоположные полуплоскости. Обе эти величины можно посчитать, найдя площади треугольников с помощью формулы Герона, а их углы — с помощью теоремы косинусов и теоремы синусов.




Из сказанного выше, мы получаем необходимое условие на то, что треугольники $t_1$, $t_2$, $t_3$ и $t_4$ можно использовать, чтобы составить тетраэдр:

1. $t_1$ и $t_2$ должны иметь общую сторону;
2. $t_3$ и $t_4$ должны иметь общую сторону;

3. оставшиеся четыре стороны $t_1$ и $t_2$ должны совпадать с оставшимися четырьмя сторонами $t_3$ и $t_4$;

4. если соединить $t_1(ABC)$ и $t_2(ABD)$ по их общей стороне $AB$, а затем вращать $t_2$ в пространстве относительно нее, расстояние между оставшимися двумя вершинами $C$ и $D$ должно в какой-то момент достигать значения, равного общей стороне $t_3$ и $t_4$.

Эти же условия так же являются достаточными, так как треугольники с равными сторонами равны, а значит при таком повороте $t_2$ в пространстве оставшиеся две «недостающие» грани тетраэдра будут в точности равны $t_3$ и $t_4$. Поскольку расстояние между $C$ и $D$ при вращении меняется непрерывно, достаточно проверить, что длина этой общей стороны лежит строго между $\min |CD| = |CD_2|$ и $\max |CD| = |CD_1|$.

Теперь воспользуемся данной идеей, чтобы найти четверку подходящих треугольников. Рассмотрим все возможные пары треугольников, в каждой паре рассмотрим все возможные способы склеить эти два треугольника по общей стороне, чтобы получить четырехугольник. Про каждый четырехугольник запомним, чему равны его стороны и одна из диагоналей (общая сторона двух треугольников). Всего четырехугольников получится $O(n^2)$. Теперь для каждого определим, существует ли для него парный, вместе с которым они смогут образовать тетраэдр описанным выше образом. Для этого сгруппируем с помощью хэш-таблицы все четырехугольники по последовательности их сторон и проверим для каждого, существует ли в его группе пара с подходящей диагональю.

Проверять существование пары с подходящей диагональю, перебирая все четырехугольники в группе, долго. Но мы можем внутри каждой группы отсортировать все длины диагоналей по возрастанию. После чего для каждого четырехугольника мы знаем минимальную и максимальную величину искомой диагонали парного к нему, и проверять существование подходящей (лежащей между определенными для него `low` и `high`) с помощью бинпоиска.

Остается только избавиться от случаев, в которых «парный» четырехугольник имеет в своем составе общий треугольник с текущим четырехугольником. Будем перебирать массив треугольников слева-направо, рассматривая пары четырехугольников вида (текущий, еще один с префикса массива) и (два треугольника с суффикса массива): множества «префиксных» и «суффиксных» четырехугольников при переходе к следующему можно обновлять за $O(n)$.

Итоговое решение работает за время $O(n^2 \log n)$ — суммарно тратим порядка $6n^2$ времени на склеивание треугольников по общим сторонам, и еще $\log(n^2) = O(\log n)$ на поиск подходящей по длине диагонали пары каждому из полученных четырехугольников.

# Задача K. Защита крепости

*Автор задачи: Андрей Станкевич, разработчики: Андрей Станкевич, Никита Голиков*

Будем решать задачу методом динамического программирования. Определим $\text{ways}_{h,w}$ как число способов построить крепость с внешним контуром размера $h \times w$, $\text{sum}_{h,w}$ как соответствующую сумму уровней защиты по всем способам построить крепость.

Заметим, что каждая крепость либо состоит из единственного внешнего контура, либо включает в себя крепость меньшего размера. Если зафиксировать размер $i \times j$ следующего контура, то существует ровно $(h - i - 1) \cdot (w - j - 1)$ способов поставить контур такого размера внутрь внешнего. Для того чтобы пересчитать сумму уровней защиты по всем способам, отдельно учтем вклад в ответ вложенных контуров и внешнего. Таким образом, верна следующая формула для пересчёта динамики:

$$\text{ways}_{h,w} = 1 + \sum_{i=1}^{h-1} \sum_{j=1}^{w-1} (h - i - 1) \cdot (w - j - 1) \cdot \text{ways}_{i,j} \tag{1}$$

$$\text{sum}_{h,w} = h \cdot w + \sum_{i=1}^{h-1} \sum_{j=1}^{w-1} (h - i - 1) \cdot (w - j - 1) \cdot (\text{sum}_{i,j} + h \cdot w \cdot \text{ways}_{i,j}) \tag{2}$$

Реализовав подсчет динамики описанными выше формулами, получим решение за $\mathcal{O}(h^2 w^2)$. Для ускорения данного решения воспользуемся техникой префиксных сумм. Заметим, что для пересчета

очередного значения динамики нам нужно уметь вычислять сумму всех меньших значений по обеим координатам, умноженных на коэффициент $(h - i - 1) \cdot (w - j - 1)$. Раскроем скобки в данном выражении:

$$(h - i - 1) \cdot (w - j - 1) = ij \cdot 1 + i \cdot (1 - w) + j \cdot (1 - h) + (h - 1) \cdot (w - 1) \tag{3}$$

Подставив раскрытое выражение в формулу для подсчета динамики, получим четыре независимых префиксных суммы, умноженных на какой-то коэффициент, зависящий только от $h, w$. Таким образом, для пересчета динамики достаточно поддерживать префиксные суммы четырех величин $(\text{ways}(i, j), \text{ways}(i, j) \cdot i, \text{ways}(i, j) \cdot j, \text{ways}(i, j) \cdot ij)$.

Формула для пересчета $\text{sum}_{i,j}$ переписывается через аналогичные префиксные суммы схожим образом. Итоговая асимптотика решения будет $\mathcal{O}(hw)$.

# Задача L. Головоломка с проводами

*Автор задачи: Артем Рипатти, разработчики: Андрей Станкевич, Григорий Шовкопляс*

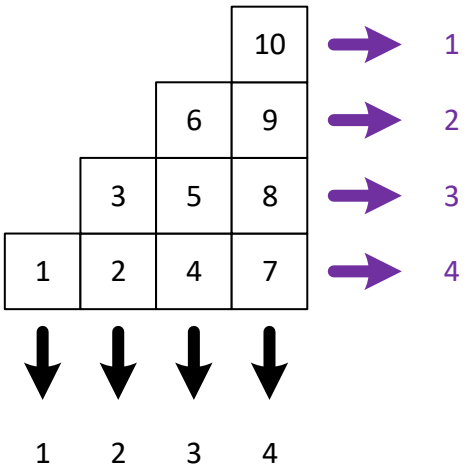
Заметим, что главная информация, которую мы получаем в ответ на запрос — в группе какого размера оказался каждый провод. Попробуем различить провода, базируясь исключительно на размерах групп.

Начнем со случая, когда $n$ является треугольным числом, иначе говоря, $n = 1 + 2 + \ldots + k$. Разобьем провода на $k$ групп, в первой будет 1 провод, во второй 2, и так далее, в $k$-й группе — $k$ проводов. Сделаем первый запрос.

На втором запрос сделаем аналогично, но разобьем иначе: первый провод, который бы в каждой группе на первом шаге отправим в группу размера $k$, второй провод — в группу размера $k - 1$ (так как в первой группе нет второго провода), и так далее. Последний, $k$-й провод в группе с $k$ проводами на первом шаге на втором шаге попадет в группу размера 1.

Заметим, что пары «размер группы на первом шаге» и «размер группы на втором шаге» различны для всех номеров левых концов проводов, значит соответствующий ему правый конец идентифицируется однозначно.

Этот процесс удобно визуализировать в виде диаграммы Юнга. На рисунке показан процесс для 10 проводов, черные стрелки соответствуют первому запросу, а фиолетовые — второму.

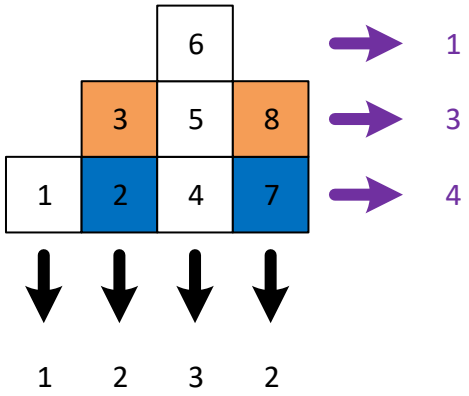



Таким образом, для треугольных чисел достаточно два запроса, но по условию необходим третий, его можно сделать фиктивным, например, определив все провода в одну группу.

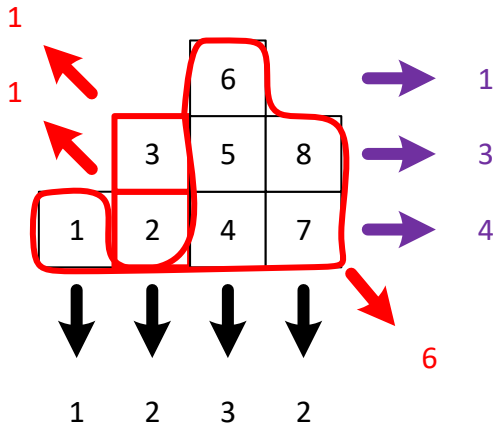
К сожалению, если $n$ не является треугольным числом, сделать в точности так не получается. Попробуем аналогично разбивать на первом запросе на группы, размером 1, 2, и так далее, всего

$k$ групп. Последняя группу в первом запросе содержит, однако, $j < k$ элементов. Второй запрос сделаем аналогично случаю для треугольного $n$, мы почти различили все провода, но провода в $j$-й и $k$-й группе на первом запросе попарно неразличимы.

На рисунке показан процесс для 8 проводов, черные стрелки соответствуют первому запросу, а фиолетовые — второму. В синий и оранжевый цвет покрашены неразличимые пары проводов.




Чтобы различить их, сделаем содержательный третий запрос. В каждой паре один из неразличимых проводов отправим в группу размера 1. А остальные, включая не попавшие в $j$-ю и $k$-ю группу на первом запросе, отправим в одну большую группу. Рисунок показывает третий запрос для $n = 8$, красным цветом обведены группы третьего запроса.




Заметим, что большая группа будет содержать строго больше одного провода, поэтому удастся различить все ранее неразличимые пары проводов.

А именно, теперь тройка «размер группы на первом шаге», «размер группы на втором шаге», «размер группы на третьем шаге» различны для всех номеров левых концов проводов, значит соответствующий ему правый конец идентифицируется однозначно.

Заметим, что у задачи есть и альтернативные решения. Например, можно сделать так. Выберем около $\sqrt{n}$ различных чисел не больше $2\sqrt{n}$ с суммой $n$. Это будут размеры групп. Сделаем $t$ копий каждого из этих чисел $t$, получим массив длины $n$. Теперь сделаем три случайных перестановку этих чисел $\pi_1$, $\pi_2$ и $\pi_3$, и проверим, что каждая тройка $(\pi_1[i], \pi_2[i], \pi_3[i])$ уникальна. Оказывается, вероятность этого достаточно высока и можно сделать не слишком много повторов случайного перемешивания размеров групп, прежде чем результат будет достигнут. После этого делаем как в решении выше, но помещаем $i$-й провод на $j$-м запросе в группу размера $\pi_j[i]$. Уникальные тройки размеров групп позволяют идентифицировать все соответствия проводов.