Разбор задачи «Валя и письмо»

Если письмо нужно повернуть, то это можно сделать перед тем, как начать его складывать. При фиксированном расположении листа, нужно сгибать письмо по горизонтали, пока высота письма больше высоты конверта, и по вертикали, пока ширина письма больше ширины конверта. Нужно рассмотреть два начальных поворота листа, для каждого посчитать ответ и взять из них минимальный.

Теперь, чтобы решить исходную задачу, достаточно рассмотреть два возможных начальных расположения письма, посчитать для каждого ответ и выбрать из них минимальный.

Важно заметить, что не верно, что перед складыванием короткая сторона письма должна быть параллельна короткой стороне конверта. Например, если стороны письма равны 5 и 4, а стороны конверта равны 4 и 3, то выгодно расположить письмо так, чтобы его длинная сторона было параллельна короткой стороне конверта. В данном случае достаточно согнуть письмо только один раз, чтобы его можно было поместить в конверт. Если же расположить письмо так, чтобы его короткая сторона была параллельна короткой стороне конверта, то в этом случае письмо придется сложить два раза: один раз по вертикали и один раз по горизонтали.

Разбор задачи «Экскурсия»

Рассмотрим пожелания любого человека. Сопоставим каждому пожеланию v две вершины — v_{true} и v_{false}. v_{true} означает, что пожелание v будет выполнено, v_{false} — пожелание v не будет выполнено.

Если не выполнено пожелание v_i, то должны быть выполнены все остальные. Проведем рёбра-следствия из $v_{i,false}$ в $v_{j,true}$ для всех других пожеланий v_j этого человека.

После проведения всех ребер нужно проверить, можно ли выбрать выполненные и невыполненные пожелания так, чтобы все ребра-следствия выполнялись. Это можно сделать алгоритмом, который используется при решении задачи $2-SAT$ за $\mathcal{O}(V+E)$, V — число вершин, E — число ребер в полученном графе. Алгоритм можно прочитать, например, на сайте `http://e-maxx.ru/algo/2_sat`.

Мы провели $\mathcal{O}(k^2)$ ребер. Для того, чтобы решить задачу, нужно уменьшить число проведенных ребер. Вместо того, чтобы проводить ребра во все пожелания кроме текущего v, воспользуемся следующей идеей:

Пусть P_i — вершина, из которой есть ребра-следствия в первые i пожеланий $v_1, v_2, \ldots, v_i$. Тогда из P_i можно провести всего два ребра.

1. $P_{i+1} \to P_i$
2. $P_{i+1} \to v_{i+1,true}$

Аналогично S_i — вершина, из который есть ребра-следствия во все пожелания с номером больше либо равным i.

Тогда для пожелания нужно провести всего два ребра: $v_{i,false} \to P_{i-1}$, $v_{i,false} \to S_{i+1}$.

Кроме того, если есть пожелание v_1 посетить город a и пожелание v_2 не посещать город a, то оба пожелания одновременно не смогут быть выполнены. Чтобы выполнить это условие для каждого города t создадим две дополнительных вершины t_{true} и t_{false}. Тогда для пожелания v, в котором требуется посетить город t, добавим ребра $v_{true} \leftrightarrow t_{true}$, $v_{false} \leftrightarrow t_{false}$. А для пожелания v, в котором не требуется посещать город t добавим ребра $v_{true} \leftrightarrow t_{false}$, $v_{false} \leftrightarrow t_{true}$.

Заметим, что суммарное число проведенных ребер $\mathcal{O}(k+m)$.

Разбор задачи «Как проиграть контест»

Переформулируем условие в более удобном виде. Даны n предметов с весами w_i и стоимостями c_i. Нужно выбрать множество предметов с суммарным весом не более W, так что к нему нельзя добавить ни один из оставшихся предметов, не превысив при этом ограничение на вес. Нужно найти минимальную суммарную стоимость выбранных предметов.

Если суммарный вес всех предметов не превышает W, то придется взять все предметы, и ответ в этом случае равен суммарной стоимости всех предметов.

В противном случае, существуют предметы, которые не войдут в оптимальный ответ. Отсортируем предметы по убыванию веса и пусть k-индекс последнего (а значит самого легкого) предмета,

который не войдет в искомое множество. Тогда в этом множестве будут все предметы с большими индексами. Пусть их суммарный вес $s_k = w_{k+1} + w_{k+2} + ... + w_n$.

Тогда из оставшихся предметов (с индексами меньше k) нужно выбрать множество с суммарным весом не меньше $W - s_k - w_k + 1$ и не больше $W - s_k$. Нетрудно видеть, что в этом случае к этому множеству можно добавить предметы с индексами большими k, и суммарный вес не превысит W, но при добавлении любого из оставшихся предметов, суммарный вес превысит W.

Для того, чтобы уметь выбирать из нескольких первых предметов подмножество определенного веса и минимальной стоимости, воспользуемся идеей динамического программирования. Пусть $dp[i][j]$ - это минимальная стоимость некоторых предметов среди первых i, таких что их суммарный вес в точности равен j. Эти значения можно посчитать по формуле: $dp[i][j] = min(dp[i-1][j], dp[i-1][j-w_i] + c_i)$.

Теперь можно перебрать индекс последнего предмета, который не войдет в оптимальное множество, для каждого случая посчитать ответ и взять среди них минимальный.

Разбор задачи «Железные дороги Берляндии»

Дано n чисел $d_1, d_2, \ldots, d_n$. Нужно построить дерево, в котором степень вершины i равна d_i.

Отсортируем все степени d_i в обратном порядке, то есть будем считать, что $d_1 \geqslant d_2 \geqslant \ldots \geqslant d_n$.

Алгоритм построения дерева:

- Перебираем вершины в порядке увеличения их номера;
- Сейчас рассматриваем вершину i. Проводим ребра из нее в такие вершины справа, в которые еще не проводили ребер до этого, пока степень вершины i не станет равной d_i.

Несколько первых шагов алгоритма:

- Рассматриваем вершину 1. Из нее мы проведем ребра в вершины $2, 3, \ldots, d_1 + 1$;
- Рассматриваем вершину 2. Из нее мы проведем ребра в вершины $d_1 + 2, d_1 + 3, \ldots, d_1 + d_2$ и одно ребро уже провели до этого из вершины 1;
- Рассматриваем вершину 3. Из нее мы проведем ребра в вершины $d_1 + d_2 + 1, \ldots, d_1 + d_2 + d_3 - 1$ и одно ребро уже провели до этого из вершины 1 или 2.

Докажем, что таким образом мы получим дерево. Заметим, что $d_1 + d_2 + \ldots + d_n = 2 \cdot n - 2$. Значит $d_1 + (d_2 - 1) + \ldots + (d_n - 1) = n - 1$. Но из каждой вершины мы будем проводить направо $(d_i - 1)$ ребер, если $i \neq 1$, а иначе d_i ребер. Поэтому суммарно мы проведем $n - 1$ ребро, сколько и нужно ребер в дереве. Также заметим, что, приходя в вершину i, во все вершины $2, \ldots, i$ уже провели ровно одно ребро, если $i \geqslant 2$. Это так, потому что для любого $i \geqslant 2$ верно неравенство $d_1 + (d_2 - 1) + \ldots + (d_{i-1} - 1) \geqslant (i - 1)$, что равносильно $d_1 + d_2 + \ldots d_{i-1} \geqslant 2 \cdot i - 3$. Это очевидно следует из того, что $d_1, d_2, \ldots, d_{i-1}$ — это $(i-1)$ максимальных чисел из всех чисел массива d, сумма чисел в котором равна $2 \cdot n - 2$. Таким образом, будет построено дерево.

Будем называть дерево хорошим, если степень i-ой вершины равна d_i для любого i. Докажем, диаметр построенного нашим алгоритмом дерева будет минимальным среди всех хороших деревьев.

Обозначим за l диаметр построенного дерева, за h максимальное расстояние из вершины 1 до какого-нибудь корня.

Заметим, что $l = 2 \cdot h - 1$ или $l = 2 \cdot h$. Это верно, так как мы в нашем алгоритме всегда подвешиваем новую вершину к листу с минимальной глубиной, следовательно разность максимальной и минимальной глубины листьев в дереве всегда не будет превосходить 1. Очевидно, что $d_1 \geqslant 2$ (за исключением случая $n = 2$, в котором есть только одно хорошее дерево), а значит из вершины 1 выходит как минимум 2 ребра, а значит существует как минимум 2 непересекающихся пути из вершины 1 до листьев длины $\geqslant h - 1$, один из которых будет иметь длину h. Значит в дереве есть путь длины как минимум $2 \cdot h - 1$. То, что $l \leqslant 2 \cdot h$ следует из того, что все пути из вершины 1 до других вершин имеют длину $\leqslant h$.

Рассмотрим любое хорошее дерево и вершину v в нем. Докажем, что существует путь из нее в какой-нибудь лист длины $\geqslant h$. Пусть есть вершина a, которая имеет высоту (расстояние до вершины v) меньше, чем вершина b, но при этом $d_a < d_b$. Пусть a и b не совпадают с v. Тогда сделаем

следующее перестроение: из вершины a выходит $d_a - 1$ ребер вниз, из вершины b выходит $d_b - 1$ ребер вниз. Рассмотрим любые $d_b - d_a$ ребер вниз из вершины b, и поддеревья, которые подвешены этими ребрами к вершине b. Отрежем эти ребра от вершины b и подвесим к вершине a вместе с поддеревьями. Тогда степень вершины a станет равной d_b, а степень вершины b станет равной d_a. Поменяем номера вершин a и b. Получится снова хорошее дерево. Но его высота не увеличится, так как мы перевесили некоторые поддеревья выше к вершине v. Также можно менять v с вершиной a, если $d_v < d_a$. Аналогично, можно менять вершины, которые имеют одинаковую высоту. Именно так можно привести хорошее дерево к нашему дереву, не увеличивая высоту. Для этого сначала поменяем максимальную степень с v, потом второй максимум с сыном v и так далее. Значит изначальная высота была как минимум h.

Рассмотрим любое другое хорошее дерево. Пусть его диаметр равен l'. Хотим доказать, что $l' \geqslant l$. Рассмотрим любой диаметр в этом дереве и его середину, какую-то вершину v. Тогда из этой вершины максимальная длина пути до остальных $\leqslant \lceil l'/2 \rceil$. Но при этом мы доказали, что оно $\geqslant h$. Значит, $h \leqslant \lceil l'/2 \rceil$, откуда следует, что $2 \cdot h - 1 \leqslant l'$. Если $l = 2 \cdot h - 1$, то мы доказали. Значит $l = 2 \cdot h$.

В таком случае докажем следующее утверждение. Пусть у нас есть хорошее дерево и ребро между u и v в нем. Тогда либо расстояние от u до какой-то вершины, лежащей в ее поддереве, если удалить это ребро, будет $\geqslant h$, либо расстояние от v до какой-то вершины, лежащей в ее поддереве, если удалить это ребро, будет $\geqslant h$. Для этого просто подвесим дерево не за вершину, а за две вершины u и v. Тут можно делать абсолютно такие же замены, если у более глубокой вершины степень больше, чем у менее глубокой. Поэтому аналогично можно свести рассматриваемое хорошее дерево к нашему, не уменьшив высоту. Но так как $l = 2 \cdot h$, то в нашем дереве есть вершина высоты $\geqslant h$.

Рассмотрим любое другое хорошее дерево. Пусть его диаметр равен l'. Хотим доказать, что $l' \geqslant 2 \cdot h$. Уже знаем, что $l' \geqslant 2 \cdot h - 1$. Рассмотрим любой диаметр в этом дереве и его среднее ребро. Тогда любая вершина лежит на расстоянии $\leqslant \lceil (l' - 1)/2 \rceil$ до этого ребра. Но при этом мы доказали, что это расстояние $\geqslant h$. Значит, $h \leqslant \lceil (l' - 1)/2 \rceil$, откуда следует, что $2 \cdot h \leqslant l'$, что и требовалось.

Асимптотика: $O(n \cdot log(n))$ или $O(n)$ (если сортировать массив d подсчетом).

Разбор задачи «Сложные задачи»

Сначала посчитаем число букв «А» в строке s — больше нам от неё ничего не нужно. Пусть их будет n. Тогда, если ответ равен k, то существует такая последовательность различных натуральных чисел $a_1, \ldots, a_k$, что $\sum_{i=1}^{k} a_i = n$. Заметим, что тогда существуют и последовательность $b_1, \ldots, b_k$, такая что $b_i = i$ для всех $i \neq k$ и $\sum_{i=1}^{k} b_i = \sum_{i=1}^{k} a_i = n$. Почему? Отсортируем последовательность a_i; теперь $i \leqslant a_i < a_{i+1}$. Тогда можно сделать $b_i = i$ (для $i < k$) и $b_k = k + \sum_{i=1}^{k-1} (a_i - i) = n - \sum_{i=1}^{k-1} i$.

Разбор задачи «Гонка роботов»

Развернем задачу: перевернем поле и будем считать, что нам нужно посчитать количество способов расставить барьеры так, чтобы расстояние от клетки $(1, 1)$ до всех клеток последнего столбца было одинаковым.

Теперь для решения задачи можно воспользоваться динамическим программированием по профилю. Будем рассматривать клетки по возрастанию номера столбца, а внутри столбца — по возрастанию номера строки. Каждый раз, рассматривая очередную клетку, будем перебирать варианты расположения стенок между этой клеткой и ее соседями, которые уже были добавлены (это клетки слева и сверху от добавляемой). Для того, чтобы пересчитывать состояние динамики, нам потребуется помимо текущей клетки хранить также профиль: для каждой добавленной клетки, справа от которой нет добавленной клетки, будем хранить расстояние от нее до клетки $(1, 1)$. Таким образом, получается следующая динамика: $dp[j][i][prof]$ — количество способов расставить барьеры между клетками первых j столбцов и первых i клеток j-го столбца, где $prof$ — расстояния до клеток профиля. Добавляя новую клетку в профиль, мы можем пересчитывать расстояния до клеток профиля.

Ответ можно получить, сложив все значения $dp[m][0][prof]$, где в массиве $prof$ все значения равны. Для того, чтобы хранить состояния динамики, можно использовать ассоциативный массив.

Разбор задачи «Гибкие отрезки»

Заметим, что если существует гибкий отрезок длины n, то существует гибкий отрезок длины $n+2$. Почему? Если отрезок $[l;r]$ является гибким, то отрезок $[l;r+2]$ также является гибким. Для того, чтобы это понять, достаточно положить $a_{r+1} = r+2$ и $a_{r+2} = r+1$. В таком случае $(r+1)\cdot(r+2) = a_{r+1}\cdot a_{r+2}$.

Решим задачу для $n = 2$. Воспользуемся той же идеей, что при переходе от отрезка длины n к отрезку длины $n+2$. Достаточно заметить, что отрезок $[1;2]$ — гибкий, $a_1 = 2$, $a_2 = 1$. Тогда, последовательно добавляя к этому отрезку справа по два элемента, мы сможем решить задачу для каждого чётного n.

Теперь решим задачу для $n = 3$. Можно перебрать на компьютере отрезок длины 3 и способы прибавить к его элементам ± 1 или найти решение на бумаге. Так или иначе, отрезок $[3;5]$ — гибкий, $a_3 = 2$, $a_4 = 5$, $a_5 = 6$. Добавляя к этому отрезку по два элемента, мы найдем гибкие отрезки для всех нечётных n, начиная с 3.

Разбор задачи «Секретный шифр»

Переберём первые две цифры ответа. Если в числе уже есть хотя бы две цифры, то можно однозначно определить остаток следующей добавленной цифры по модулю 3.

123...ab c

Если мы добавим цифру c, то $(a+b+c) \equiv 0 \mod 3$ Среди цифр с нужным остатком, выберем максимальную, чтобы максимизировать составленное число.

Рассмотрим пример:

- Например, набранное число $123\ldots34$. Сумма остатков последних двух цифр 1, следовательно, у следующей цифры остаток 2.
- Пусть есть неиспользованные 1, 3, 5 и 8
- Подходящими цифрами являются 5 и 8
- Добавим из них максимальное, то есть 8

Доказательство:

- Заметим, что в числе, у которого любая подстрока длины три делится на три, можно поменять две цифры с одинаковым остатком по модулю 3, и это свойство сохранится Если $x < y$ и $x \equiv y \mod 3$, то мы можем поменять x, y местами и увеличить число. Значит в оптимальном ответе числа одного остатка отсортированы по убыванию. $\Rightarrow$ Брать максимальную из возможных — верно.

Время работы: $\mathcal{O}(digits \cdot digits \cdot len)$.

Разбор задачи «Путешествие по тору»

4 большие дорогие делят каждую из стран на 4 равные части длины $\frac{2\pi r}{4}$. Очевидно, Сеня никогда не попадет на внешнюю дорогу, потому что можно, начиная с момента движения в ее сторону, заменить весь путь на симметричный относительно центральной окружности, и тем самым сократить его (потому что тогда он пойдет по внутренней). Более того, из симметрии северной и южной дорог, можно считать, что из них Сеня бывает только не северной.

Назовем любой отрезок большой дороги между соседними странами *горизонтальным шагом*, а малой между соседними городами — а *вертикальным шагом*.

Сеня побывал хотя бы по разу в каждой стране, значит сделал хотя бы $n-1$ горизонтальный шаг. Для начала, можно рассматривать только движение по часовой стрелке, так как если Сеня развернулся в какой-то момент, то он направился в страну, (1) по которой уже проехал, или (2) по которой проедет на этом обратном пути. Первое — лишние действия, которые можно удалить из

пути, второе — путь, в котором движение туда-обратно можно заменить на развернутую часть пути против часовой стрелки, и путь тоже сократится.

А это значит, что Сеня использовал ровно n или $n-1$ горизонтальных шагов. Если $n-1$, то он начал с вертикального шага, иначе — с горизонтального, и закончил вертикальным. Последнее означает, что путь можно сократить, развернув его против часовой стрелки целиком и удалив последний (ранее — первый) горизонтальный шаг.

Тепень мы знаем, что любой путь, который может быть оптимальным, состоит из не менее, чем n вертикальных и ровно $n-1$-го горизонтального шагов. Вертикальные шаги просто меняют Сенину позицию между i_k и n_k, тогда назовем *малым шагом* часть пути между соседними городами одной страны **сразу после того, как Сеня попал в эту страну** и назовем *большими шагами* все части пути между двумя соседними малыми шагами.

Теперь путь состоит ровно из n малых $n-1$-го большого чередующихся шагов (*). Большие шаги можно разбить на 4 типа:

(1) $i_k \to i_{k+1}$

(2) $n_k \to i_k \to i_{k+1}$

(3) $n_k \to n_{k+1}$

(4) $i_k \to n_k \to n_{k+1}$

(4)-й тип можно отбросить, потому что можно заменить его на (1)-й с малым шагом вида $i_{k+1} \to n_{k+1}$, следующим после. Так, не меняя общего пройденного расстояния, мы избавляемся от всех (4)-го типа. И если после этого путь не удовлетворяет описанию (*), то он исходно был не оптимальный, так как полученный можно сократить.

А еще выгодно жадно выбирать (1)-й тип при возможности, потому что при другом выборе Сеня идет из i_k в n_{k+1} и можно сначала удалить все лишние шаги туда-обратно, а потом сократить еще на $\frac{\pi r}{2}$ или 0 поменяв все i_j и n_j друг с другом (и пути между ними), начиная с $j = k+1$-й страны включительно. Это не сделает путь длиннее, а возможно и сделает его короче.

Наилучший результат дает путь вида $(3), (1), (3), (1), \ldots$, потому что (1)-й тип может следовать только за (3)-м. На самом деле очевидна оптимальность этого пути для случая, когда (2)-й тип не короче (3)-го и (1)-й самый короткий. Поскольку (1)-й самый короткий всегда, независимо от n, посмотрим, что случается, если (2)-й короче (3)-го.

Выпишем формулы $(1) = \frac{2\pi(R-r)}{n}$, $(2) = (1) + \frac{\pi r}{2}$, $(3) = \frac{2\pi R}{n}$, и проверим когда $(2) < (3)$. Ответ: при $n < 4$. Для $n = 3$, $(3), (1)$ на самом деле останется оптимальным вариантом, потому что после (2) может следовать только (2) или (3), а $2 \cdot (2) > (1) + (3)$ при $n > 2$. Для $n = 2$ (2) даст минимальную длину пути, а при $n = 1$ большие шаги вообще отсутствуют, и нет разницы между (1), (2) и (3).

Получаем, что для всех $n \neq 2$ минимальное расстояние достигается на пути вида $(3), (1), \ldots$ $n-1$ раз, плюс n малых шагов. Для $n = 2$ ответом будет (2) плюс два малых шага. Так же надо не забыть про четность n, потому что первая формула будет отличаться для четных и нечетных.

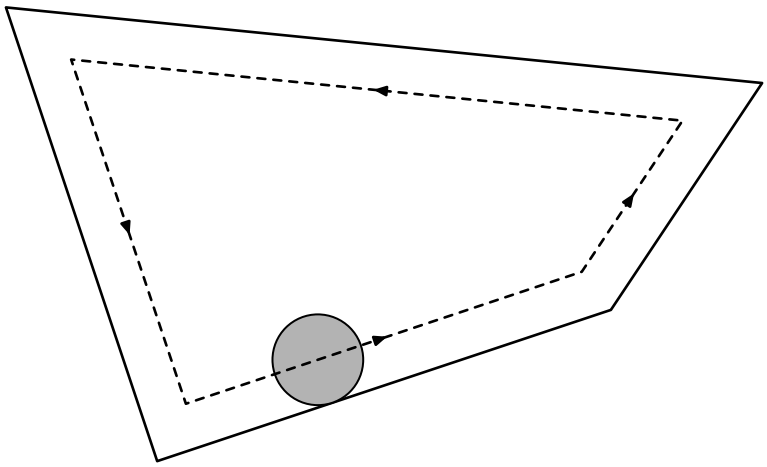
Разбор задачи «Новая прогулка Амальтеи»

Рассмотрим граф клеток, где соединим ребром две соседние по стороне клетки. Возьмем любое остовное дерево в этом графе, например дерево обхода в глубину. Если пройти вдоль этого дерева «по правилу правой руки» как раз получится гамильтонов цикл в графе, где клетки разделены на 4 части.

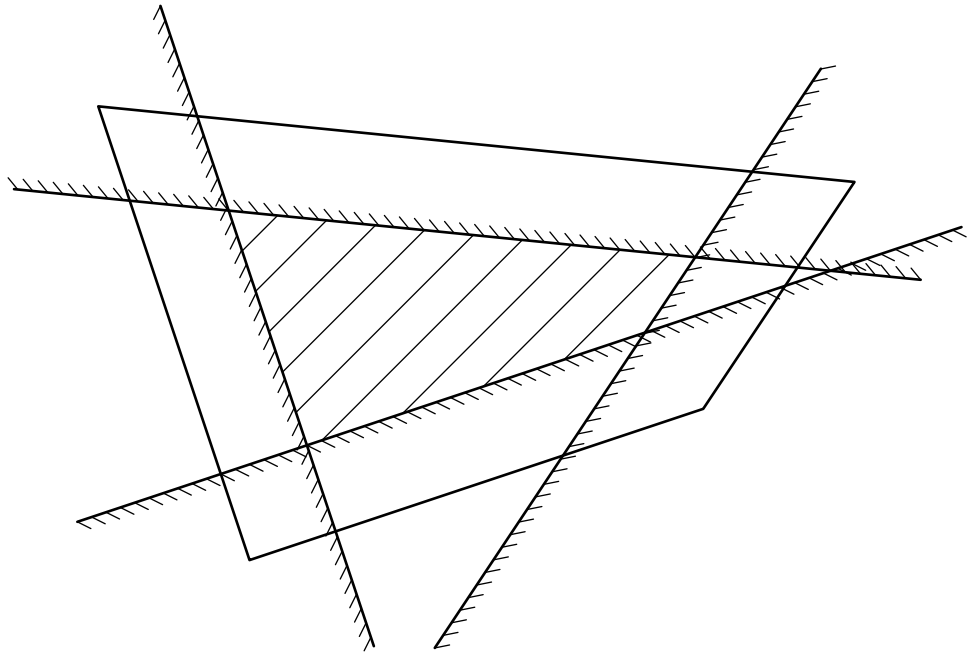
Разбор задачи «Формула 42»

Заметим, что если можно расположить барьеры так, чтобы трассу можно было использовать для болида с радиусом r, то можно расположить барьеры так, чтобы трассу можно было использовать для любого болида с меньшим радиусом. Поэтому, можно сделать двоичный поиск по ответу.

Для того, чтобы проверить, можно ли сделать трассу для болида с радиусом r, в качестве замкнутой ломанной, по которой будет двигаться центр круга с радиусом r, возьмем ломанную на расстоянии r от внешнего барьера. То есть, такую ломанную, что круг, центр которого движется по ней, будет всегда касаться внешнего барьера.



Тогда, множество точек, лежащих внутри внешнего барьера, и никогда не оказывающихся внутри круга, образует выпуклый многоугольник. А конкретнее, многоугольник, получающийся смещением каждой из сторон внешнего барьера на расстояние $2 \cdot r$, перпендикулярно стороне, внутрь многоугольника.



После этого, нужно проверить, можно ли расположить внутренний барьер так, чтобы он лежал внутри получившегося многоугольника. Научимся проверять, можно ли сместить параллельным переносом выпуклый многоугольник B, чтобы он оказался внутри выпуклого многоугольника A. B лежит внутри A, если каждая вершина B лежит внутри A. Рассмотрим множество V_1 векторов v, что если B переместить параллельным переносом на вектор v, первая вершина B окажется внутри многоугольника A. Формально $V_1 = \{v : b_1 + v \in A\}$, где b_1 — первая вершина многоугольника B. Заметим, что $V_1 = A - b_1$, то есть V_1 получается из многоугольника A параллельным переносом на вектор $-b_1$. Аналогично определим $V_2, \dots V_m$. Пусть $V = \bigcap_{i=1}^{m} V_i$. Тогда, если V не пусто, возьмем в качестве v любой вектор из V. Если сместить B на вектор v, каждая из вершин B окажется внутри A, а значит и весь многоугольник B будет лежать внутри многоугольника A. Если же V пусто, то такого вектора, при сдвиге на который B окажется внутри A, не существует.

Для того, чтобы проверить пустоту V достаточно пересечь $n \cdot m$ полуплоскостей.

Разбор задачи «Сколько тестов»

Заметим, что количество тестов в задаче — это максимальный номер теста, который есть в этой задаче.

Пусть все данные в файле строки имеют длину l. Тогда легко понять, что все номера тестов не превосходят $10^l - 1$. Значит, максимальный номер теста также не превосходит этой величины, причем достигать этой величины он может. Значит, максимальное возможное количество тестов — это $10^l - 1$.

Чтобы узнать минимальное возможное количество тестов, нужно посмотреть на максимальный номер теста среди данных и разобрать два случая:

1. Пусть у максимального номера теста среди данных есть ведущие нули (то есть первый символ в номере теста «0»). Тогда легко понять, что максимальный номер теста в задаче должен быть не меньше, чем 10^{l-1}, так как в противном случае длина всех строк была бы меньше. Также заметим, что так как максимальный из данных номеров имеет ведущий ноль, все номера тестов не превосходят 10^{l-1}. Значит, в этом случае минимальное возможное количество тестов — это 10^{l-1}.

2. Пусть у максимального номера теста среди данных ведущих нулей нет. Тогда этот тест может являться максимальным среди всех номеров тестов, и ответ равен максимальному номеру теста среди данных.