Задача A. Нечётный букет

Автор идеи: Иван Белоногов
Разработка: Иван Белоногов
Разбор задачи: Иван Белоногов

Каждого вида цветов в букете должно быть нечётное количество, поэтому если определённого вида цветов чётное количество, то это число следует уменьшить на единицу $a_i := a_i - 1$, всё равно последний «чётный» цветок взять не удастся.

Теперь каждого типа цветов нечётное количество и нет смысла брать какого-то вида не все цветы.

Так как требуется составить букет из нечётного количества цветов, где каждого вида цветов тоже нечётное количество, количество различных видов цветов в букете должно быть тоже нечётным.

Если число видов цветов нечётно, просто возьмём все цветы. Иначе цветы какого-то типа придётся не брать. Логично не брать тот вид цветов, для которого количество цветов этого вида минимально.

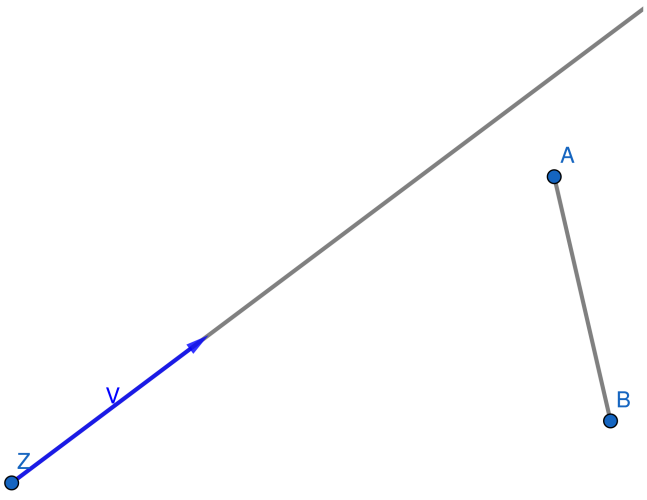
Итого, после того, как число цветов каждого типа сделано нечётным, чтобы получить ответ для нечётного n нам достаточно посчитать сумму всех a_i, а если n чётно, то после подсчёта суммы необходимо найти минимум в массиве a и вычесть его из суммы.

Задача B. Перекрытие отрезков

Автор идеи: Демид Кучеренко
Разработка: Антон Гардер
Разбор задачи: Антон Гардер

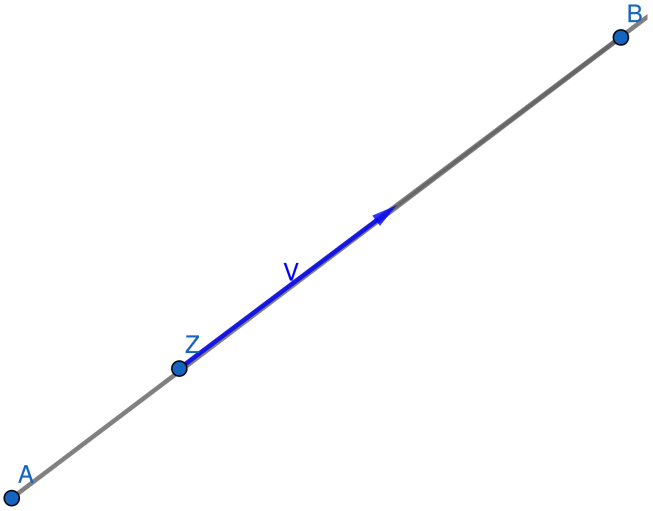
Отрезок AB перекрывает отрезок CD по направлению $\vec{v}$ только тогда, когда точка A или точка B попадает на отрезок CD, двигаясь по направлению $\vec{v}$, или когда точка C или точка D попадает на отрезок AB, двигаясь по противоположному направлению $-\vec{v}$. Траектория движения точки по неизменяющемуся направлению — это луч, то есть для решения задачи нужно четыре раза проверить, пересекается ли определённый луч с определённым отрезком.

Научимся проверять, что луч из точки Z в направлении $\vec{v}$ пересекает отрезок AB. Для удобства рассмотрим ещё прямую l, содержащую луч. Рассмотрим случаи.

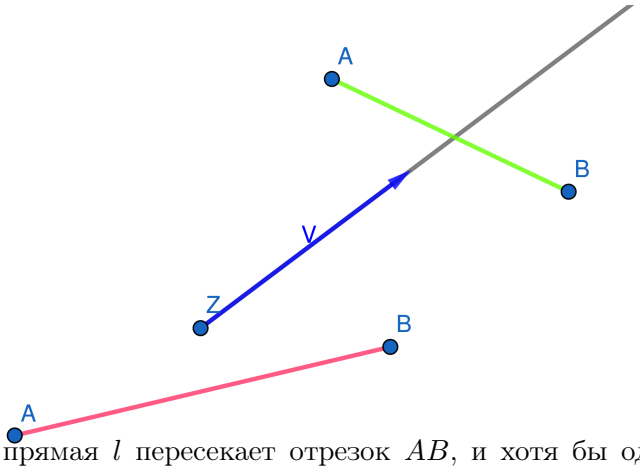


Если точки A и B не лежат на прямой l и находятся по одну сторону от неё, то прямая l не пересекает отрезок, а значит и луч не пересекает. Проверить это можно с помощью знаков векторных

произведений.



Если точки A и B обе лежат на прямой l, то луч пересекает отрезок если хотя бы один из векторов $\overrightarrow{ZA}$ или $\overrightarrow{ZB}$ сонаправлен с вектором $\vec{v}$. Проверить это можно с помощью знаков скалярных произведений.



Теперь мы знаем, что прямая l пересекает отрезок AB, и хотя бы одна из точек отрезка не принадлежит прямой. Пусть A находится слева от l или лежит на ней, а B находится справа от l или лежит на ней. Тогда луч пересекает AB, если точка Z находится справа от вектора $\overrightarrow{AB}$. Это также можно проверить с помощью знака векторного произведения. Аналогично рассматривается зеркальный случай.

Итого, задача сводится к вычислению знака нескольких векторных и скалярных произведений, это можно сделать полностью в целых числах, чтобы не пришлось опасаться потерь точности.

Задача C. Великая теорема Ферма

Автор идеи:	Михаил Дворкин
Разработка:	Арсений Кириллов
Разбор задачи:	Михаил Дворкин

Для начала рассмотрим процедуру, которая бесконечно выводит описанные в задаче неравенства, игнорируя значения l и r. Её несложно реализовать, например, в виде пяти вложенных циклов for (самый внешний — по максимуму из четырёх чисел, затем по a, по b, по c и по n).

Теперь модифицируем этот код, чтобы он выводил только неравенства с l-го по r-е. Для этого на каждой итерации каждого цикла for вычислим сперва, сколько неравенств будут выведены в рамках этой итерации. Если это число s строго меньше l, то эта итерация не породит неравенств, которые следует вывести. Следовательно, можно просто уменьшить l и r на величину s и перейти к следующей итерации.

Кроме того, конечно, как только очередная итерация самого вложенного цикла вывела последнее неравенство, требуемое в данном тесте, весь процесс следует прервать.

Последняя сложность, которую необходимо обсудить, — какой знак неравенства следует выводить в очередном неравенстве. Стандартного типа данных с плавающей точкой не хватает, чтобы

вычислить допустимые в этой задаче степени, например 999^{999}. Для многих неравенств можно воспользоваться простыми правилами: если $max(a, b) \geqslant c$ или $n \cdot (\log(c) - \log(\max(a, b))) > \log(2)$, то ответ понятен. В остальных случаях можно воспользоваться библиотекой для работы с «длинной арифметикой».

Задача D. Угадай путь

Автор идеи: Иван Сафонов
Разработка: Иван Сафонов
Разбор задачи: Иван Сафонов

Давайте обозначим за l_i и r_i (для всех $1 \leqslant i \leqslant n$) — наименьший номер строки клетки пути, которая лежит в i-м столбце и наибольший номер строки клетки пути, которая лежит в i-м столбце. Заметим, что тогда весь путь это $(l_1, 1), (l_1 + 1, 1), \ldots, (r_1, 1), \ldots, (l_i, i), (l_i + 1, i), \ldots, (r_i, i), \ldots, (l_n, n), (l_n + 1, n), \ldots, (r_n, n)$. Будем находить значения l_i, r_i, делая запросы.

Заметим, что изначально мы знаем, что $l_1 = 1$, $r_n = m$, $r_i = l_{i+1}$ (для всех $1 \leqslant i \leqslant n - 1$). Будем называть столбец хорошим, если мы угадали оба значения l_i, r_i для этого столбца. Если $n = 1$ или $m = 1$, то задача тривиальная, иначе изначально никакой столбец не является хорошим.

Будем увеличивать множество хороших столбцов, с помощью запроса. Пусть сейчас столбцы $1 \leqslant c_1 < c_2 < \ldots < c_k \leqslant n$ хорошие. Тогда построим путь для того, чтобы сделать запрос, следующим образом, сконкатенировав его из нескольких последовательных путей: специальный путь из $(1, 1)$ в (l_{c_1}, c_1), специальный путь из (l_{c_1}, c_1) в (l_{c_2}, c_2), ..., специальный путь из (l_{c_k}, c_k) в (m, n).

Специальный путь из клетки (x_1, y_1) в клетку (x_2, y_2) построим так: если они лежат в одной строке или одном столбце, то путь однозначен, иначе сделаем такую конструкцию: пусть $mid = \lfloor \frac{y_1 + y_2}{2} \rfloor$, тогда:

- путь направо из (x_1, y_1) в (x_1, mid);
- путь вниз из (x_1, mid) в (x_2, mid);
- путь направо из (x_2, mid) в (x_2, y_2).

После этого сделаем запрос для полученного пути. Всеми клетками, в которых загорится датчик обновим значения l_i, r_i, того столбца, в котором стояла клетка. Заметим, что если какой-то столбец был столбцом mid для какого-то специального пути, то этот столбец будет хорошим.

Тогда будем продолжать делать запросы, пока количество хороших столбцов не станет равно $n - 1$. После этого значение l_n можно вычислить как r_{n-1}, то есть мы получим корректные значения l_i, r_i для всех столбцов.

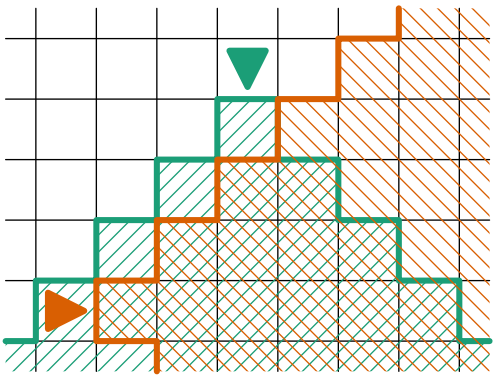
Утверждается, что предоставленный алгоритм всегда сделает $\leqslant \lceil \log_2 n \rceil$ запросов, что нам подходит. Для этого заметим, что всегда между любыми двумя соседними хорошими столбцами столбец в середине тоже становится хорошим, поэтому за логарифм от количества столбцов, все столбцы кроме последнего окажутся столбцами mid в каком-нибудь запросе.

Задача E. Робо-прятки

Автор идеи: Николай Будин
Разработка: Николай Будин
Разбор задачи: Николай Будин, Андрей Станкевич

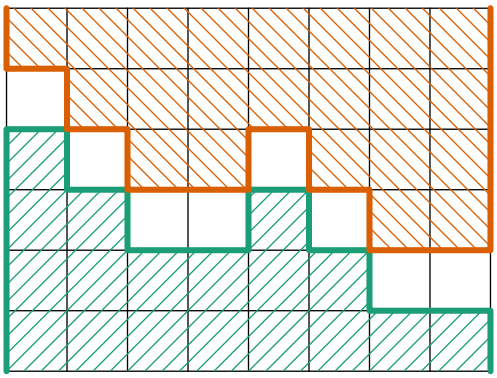
Во-первых, научимся выбирать наибольшее по размеру множество позиций исходного поля, что роботы в них друг друга не видят, а потом докажем, что всех остальных можно повернуть на 90 градусов, чтобы позиция стала хорошей.

Будем решать задачу независимо для смотрящих влево/вправо и смотрящих вверх/вниз: другие пары роботов не могут видеть друг друга.

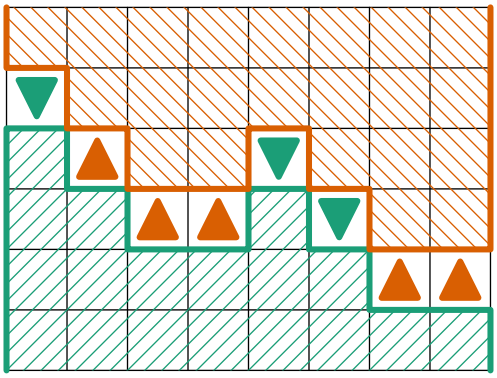


Рассмотрим случай роботов, смотрящих вверх/вниз. Заметим, что если один робот смотрит вверх, а другой — вниз, то для них отношение «один видит другого» симметрично. Значит никто из смотрящих вниз не должен видеть смотрящих вверх и наоборот.

Таким образом, нужно выбрать разрез от левой границы до правой, такой, что робот, стоящий в любой клетке выше разреза и смотрящий вверх, не будет видеть ни одну клетку ниже разреза. Аналогично, робот, стоящий в любой клетке ниже разреза и смотрящий вниз, не будет видеть ни одну клетку выше разреза. Следовательно можно оставить всех роботов, смотрящих вверх, выше разреза, и всех, смотрящих вниз, ниже разреза.



Верно и в обратную сторону, если мы оставили какой-то неконфликтующий набор смотрящих вверх/вниз, всегда можно между ними провести такую границу. Например, в качестве части выше разреза взять объединение областей видимости роботов, смотрящих вверх, а в качестве части ниже разреза – все остальное.



Заметим, что разрез должен изменяться не более, чем на один при переходе от столбца к следующему, для этого используем динамическое программирование. Будем считать, сколько роботов можно оставить, если мы рассмотрели k столбцов и в очередном столбце провели разрез после i-й строки.

Как теперь повернуть удаленных роботов, чтобы ничего не сломалось? Рассмотрим удаленного робота, который смотрел налево/направо. В зависимости от того, как проходит разрез между верхними/нижними роботами в его столбце, его всегда можно повернуть на 90 градусов по или против часовой стрелки, чтобы он стал смотрящим вверх/вниз по правильную сторону от разреза.

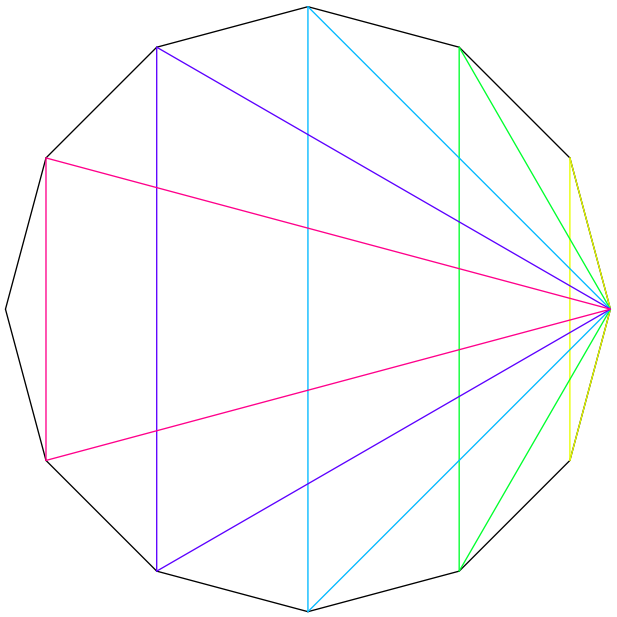
Задача F. Равнобедренные треугольники

Автор идеи: Егор Горбачев
Разработка: Егор Горбачев
Разбор задачи: Егор Горбачев

У равнобедренного треугольника есть вершина напротив основания — будем называть её первой — и две другие, которые равноудалены от первой. Давайте зафиксируем первую вершину. Тогда почти для любой другой вершины существует единственная такая третья, что они образуют равнобедренный треугольник. Эта третья — это симметричная второй относительно оси симметрии многоугольника, проходящей через первую вершину.

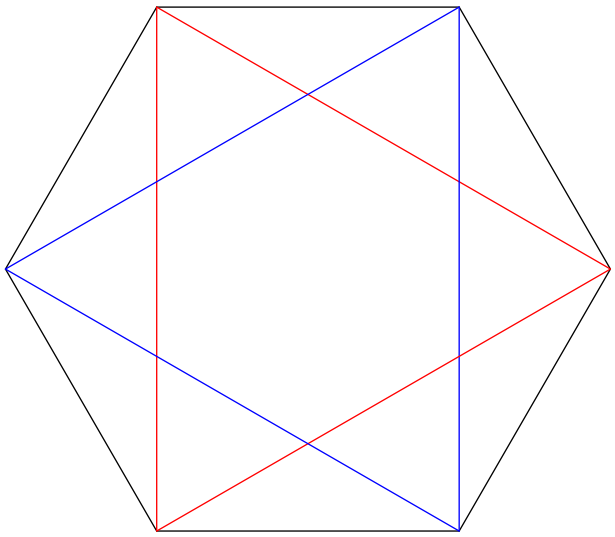
Единственные ограничения — это чтобы вторая вершина не была равна или диаметрально противоположна первой. Так что есть $n-1$ вариант при нечетном n и $n-2$ при четном, потому что при нечётном нет противоположной вершины, а при чётном есть. Заметим также, что каждый треугольник мы посчитали дважды, потому что от того, что мы поменяем местами вторую и третью вершины, треугольник не поменяется.

Итоговый результат при фиксированной первой вершине получается равным $\frac{n-1}{2}$ при нечетном n и $\frac{n-2}{2}$ при четном n, что можно упростить до $\lfloor \frac{n-1}{2} \rfloor$ (целая часть $\frac{n-1}{2}$).



Но первой вершиной может быть любая вершина многоугольника, так что ответ надо домножить на n.

Однако существуют правильные треугольники, у которых каждая вершина может выступать как первая, так что их мы посчитаем трижды. Правильный треугольник, содержащий фиксированную вершину существует, если есть еще две такие вершины, что между каждой парой одинаковое количество вершин, то есть n делится на 3. А если делится, то каждая вершина лежит ровно в одном правильном треугольнике, поэтому их $\frac{n}{3}$. Каждый из них мы посчитали трижды, а надо посчитать один раз, так что необходимо вычесть из ответа $\frac{2n}{3}$.



Соберем вместе все, что мы получили:
Если n не делится на 3, то ответ равен $\lfloor \frac{n-1}{2} \rfloor \cdot n$, а если делится, то $\lfloor \frac{n-1}{2} \rfloor \cdot n - \frac{2n}{3}$.

Задача G. Слишком много минусов

Автор идеи:	Олег Христенко, Андрей Станкевич
Разработка:	Даниил Орешников
Разбор задачи:	Даниил Орешников

Заметим, что поскольку мы рассматриваем только строки минимальной длины из всех, в которых нет двух минусов подряд, количество добавленных скобок определяется однозначно. Посчитаем количество пар стоящих рядом минусов, и возьмем минимальное четное число, не меньшее данного.

Поскольку длина строки относительно небольшая, можно воспользоваться динамическим программированием, чтобы посчитать количество защищенных строк для s с каждым возможным заданным заранее префиксом. Закодируем каждый префикс четверкой чисел ($\texttt{pref}, \texttt{brackets}, \texttt{balance}, \texttt{last}$), где `pref` — длина префикса s, вошедшего в наш префикс защищенной строки, `brackets` — количество вошедших скобок, `balance` — баланс на этих скобках, и `last` — последний символ в нашем префиксе. Заметим, что разным префиксам может соответствовать одна и та же четверка чисел, но это не имеет значения. Также в рамках этого решения будем считать, что символы строки нумеруются с 1.

Посчитаем динамическим программированием количество строк с заданным префиксом $\texttt{dp}[\texttt{pref}][\texttt{br}][\texttt{bal}][\texttt{last}]$. Заметим, что для этого достаточно перебрать следующий дописываемый символ и использовать уже посчитанное значение динамики для нового состояния. В частности,

- если $\texttt{last} \neq \texttt{'-'}$, либо если $s[\texttt{pref}+1] \neq \texttt{'-'}$, то можно спокойно дописать следующий символ из строки s, то есть добавить $\texttt{dp}[\texttt{pref}+1][\texttt{br}][\texttt{bal}][s[\texttt{pref}+1]]$
- если баланс не максимален, можно дописать открывающую скобку, то есть добавить $\texttt{dp}[\texttt{pref}][\texttt{br}+1][\texttt{bal}+1][\texttt{'\{'}]$
- если баланс строго положителен, можно дописать закрывающую скобку, то есть добавить $\texttt{dp}[\texttt{pref}][\texttt{br}+1][\texttt{bal}-1][\texttt{'\}'}]$

Эту динамику надо считать в порядке уменьшения `pref`, а внутри в порядке уменьшения `br`. После того, как значения `dp` почитаны, построим k-ю защищенную строку.

Для этого возьмем пустой префикс $(0, 0, 0, \texttt{'+'})$. После этого за одну итерацию попробуем в порядке возрастания перебрать все символы, которые можно к этому префиксу дописать, а именно $s[\texttt{pref}+1]$ (если либо он, либо `last` — не минус), `'{'` и `'}'`. Если k больше соответствующего значения `dp`, значит любая строка с таким префиксом будет иметь номер меньше k. Тогда поиск k-й — это то же самое, что и поиск $(k - \texttt{dp}[\ldots])$-й среди всех, начинающихся **не** на пропущенные префиксы.

В таком случае мы вычитаем из k это значение `dp` и переходим к следующему символу. Как только выполняется условие $k \leqslant$ `dp`$[\ldots]$, текущий символ дописывается к префиксу, и начинается перебор следующего.

Если оказалось, что после перебора всех символов ни один не подошел, то есть $k > 0$ после всех вычитаний, то надо вывести «`Overflow`», иначе мы достроим префикс до самого конца и получим искомую k-ю защищенную строку.

Задача H. Девятая планета

Автор идеи:	Алексей Плешаков
Разработка:	Алексей Плешаков
Разбор задачи:	Алексей Плешаков

Давайте для начала поймём, что операция «`+` x», применённая к числу a, не изменяет его остаток от деления на девять, а операция «`-` y» изменяет остаток от деления на девять ровно на y. Значит, чтобы получить b из a, нужно будет применить к a несколько операций «`-` y_i» таких, что $\sum_i y_i \equiv b - a \pmod 9$. Попутно мы должны как-то прибавлять девятки к a так, чтобы эти операции можно было выполнить.

Давайте прибавим к a столько девяток, чтобы первая цифра нового числа была равна единице. Удалим эту цифру. Будем повторять такие операции до тех пор, пока $b - a$ не будет делиться на девять, а после прибавим к a $\frac{b-a}{9}$.

Данное решение, очевидно, работает, если $|a| < |b|$, так как длина промежуточных чисел будет меньше либо равна b, а, если равна, то, значит, первая цифра — единица, и мы удалим её следующей операцией. Это работает быстро, потому что число операций не превосходит 17.

Пусть $|a| \geqslant |b|$. Существует число x, состоящее только из единиц, такое, что $x \equiv a \pmod 9$. Длина этого числа не превосходит 18, а, значит, мы можем получить из a число x, а затем из x — число 0. После этого воспользуемся вышеописанным алгоритмом.

Задача I. Даты

Автор идеи:	Андрей Станкевич
Разработка:	Дмитрий Гнатюк
Разбор задачи:	Дмитрий Гнатюк

Задача исключительно на реализацию, не требует особых алгоритмических идей.

Решим задачу для каждой строки.

После считывания пробегаемся по строке в поиске точки или слеша, чтобы определить в каком формате задана дата. Далее разделим строку по разделяющим символам, затем, если надо, запомним сколько нулей надо добавить в начало дня/месяца/года. Затем выводим ответ в двух форматах.

Задача J. На заводе

Разработка:	Дмитрий Саютин
Разбор задачи:	Андрей Станкевич

Решение задачи состоит из двух основных шагов.

Шаг 1: сведём задачу к немного другой. Рассмотрим путь Акане, заметим, что он разобьет плоскость на несколько частей. Если части назначить вершинами графа, и сказать, что вершины соединены ребром, если у них есть общая сторона, то граф частей получится двудольным. Раскрасим его в два цвета, чёрный и белый, пусть бесконечная часть раскрашена в белый цвет.

Заметим, что требования к пути означают, что:

1. все клетки, не являющиеся цехами, белые;
2. у каждого важного узла есть соседние клетки и белого и черного цвета;
3. множество чёрных клеток 8-связно (касающиеся диагонально части также являются соседними);

4. множество белых клеток 8-связно (касающиеся диагонально части также являются соседними).

Верно и обратное, если раскрасить плоскость, чтобы указанные условия выполнялись, то граница между чёрными и белыми частями будет корректным путём для Акане.

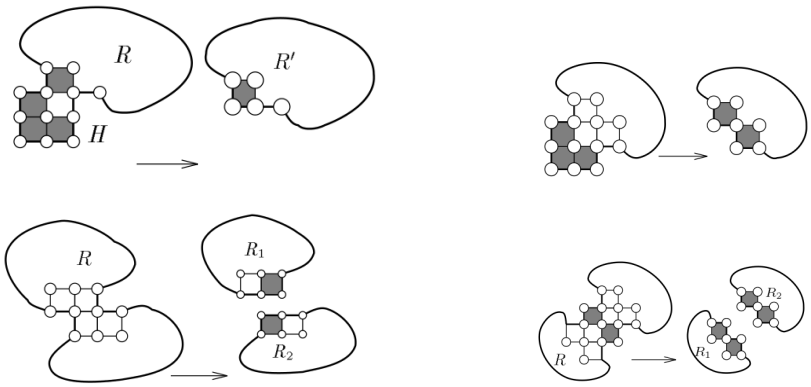
Шаг 2: построить искомое разбиение. Сформулируем без доказательства основные тезисы, которые позволяют построить разбиение. За формальным доказательством отправим читателя к статье Hwan-Gue Cho, Alexander Zelikovsky «Spanning closed trail and hamiltonian cycle in grid graphs», http://gg.gg/orspc-j.

Будем называть важный узел внутренним, если все четыре соседние с ним клетки являются территорией завода, и внешним в противном случае. Будем называть важный узел угловым, если от него отходят ровно 2 коридора.

Единственный случай, когда ответ отрицательный — когда завод представляет собой объединение квадратов 2×2, никакие два из которых не имет общей стороны длины 2.

Если у завода нет специального вида конструкций: корридора, который соединяет два узла на внешней границы завода, или внутреннего узла, у которого нет другого соседнего внутреннего узла, то жадный алгоритм строит корректное разбиение: объявить клетку чёрной, если она имеет соседний угловой узел, или сумма её координат чётна.

Если у завода есть описанные конструкции, то они, с точностью до поворота и отражения, сводятся к четырём ситуациям, показанным на рисунке ниже. Для первых двух ситуаций удалим крайний квадрат 2×2 и, решив задачу без него, вернём его, обойдя очевидным образом. Для остальных двух ситуаций разделим завод на два и, решив задачу для каждой части, объединим решения.



Задача K. Почти сортировка вращением

Автор идеи:	Михаил Дворкин
Разработка:	Михаил Дворкин
Разбор задачи:	Михаил Дворкин

Покажем, как в прямоугольнике ширины n и высоты $3 \leqslant h \leqslant n$ передвинуть n самых больших чисел в нижнюю строку в желаемом порядке. Такую операцию надо совершить $n-2$ раза, и задача будет решена.

Введём удобную процедуру: «поверни такой-то квадрат 2×2 так, чтобы максимум из его четырёх элементов оказался в такой-то его клетке». Такого результата можно добиться, выполнив три команды, поддерживаемых роботом. Вот пример кода, добивающегося, чтобы в квадрате с левым верхним углом `a1` максимум оказался в клетке `a2`:

```
b2 > a2 ? a1
b1 > a2 ? a1
a1 > a2 ? a1
```

Действительно, после первой команды выполнено, что максимум точно находится в множестве клеток {b1, a1, a2}, после второй — что точно в {a1, a2}, после третьей — что точно в клетке a2.

Следующая удобная процедура: «добейся, чтобы в таком-то прямоугольнике $2 \times s$ ($s \geqslant 2$) максимум оказался в таком-то его углу». Этого можно добиться, запуская предыдущую процедуру для всех вмещающихся в него квадратов 2×2 в естественном порядке — «пригоняя» максимум в нужный угол.

Теперь научимся ставить максимум всего прямоугольника $h \times n$ в правый нижний угол. В прямоугольнике, состоящем из двух верхних строк, «пригоним» максимум в нижний левый угол. Затем сделаем то же для строк 2 и 3, затем для 3 и 4, и так далее, до строк $h-2$ и $h-1$. Мы добились, что максимум просмотренной области оказался во второй снизу строке, самой левой её клетке. А теперь в прямоугольнике из двух нижних строк отправим максимум в правый нижний угол.

Аналогичная процедура позволит привести на положенное место и второй по величине элемент, и третий и так далее. Просто последний рассматриваемый прямоугольник из двух нижних строк надо брать ширины не n, а каждый раз на единицу меньшей.

Так мы сможем отправить на свои места $n-1$ наибольших чисел таблицы. Только с последним числом возникает техническая сложность, ведь прямоугольник 2×1 мы обрабатывать не умеем. Рассмотрим этот момент подробнее, например, при $n = 9$. После прохода по всем парам соседних строчек верно, что искомый максимум либо в a8 (приведённый туда проходом по всем парам строчек), либо в a9. А большее число уже поставлено в правильную позицию b9, и это нельзя испортить. Повернём угловой квадрат 2×2 так, чтобы уже поставленное число из b9 переехало в a9. Тогда два описанных выше кандидата переместятся в клетки b8 и a8. Повернём квадрат с левым верхним углом в a7 так, чтобы максимум из кандидатов оказался в a8. А затем повернём угловой квадрат, чтобы значения из a8 и a9 переехали соответственно в a9 и b9, чего и необходимо было достичь.

Задача L. Путешествия во времени

Автор идеи:	Иван Сафонов
Разработка:	Иван Сафонов
Разбор задачи:	Иван Сафонов

Можно заметить, что в каждый год дорожная система Берляндии имеет вид дерева, состоящего из n вершин. Задача заключается в том, чтобы для всех пар $1 \leqslant s, f \leqslant n$ найти количество таких $1 \leqslant p \leqslant n$, что в каждом из данных k деревьев вершина с номером p лежит на пути между вершинами с номерами s и f.

Обозначим за $dist_i[s][f]$ количество ребер на простом пути между вершинами с номерами s и f в дереве с номером i. Тогда, заметим что:

- $dist_i[s][f] = dist_i[s][p] + dist_i[p][f]$, если вершина p лежит на пути между вершинами s и f в i-м дереве;
- $dist_i[s][f] < dist_i[s][p] + dist_i[p][f]$, иначе.

Обозначим за $sum[s][f] = \sum_{i=1}^{k} dist_i[s][f]$. Из полученного легко получить, что:

- $sum[s][f] = sum[s][p] + sum[p][f]$, если вершина p лежит на пути между вершинами s и f во всех деревьях;
- $sum[s][f] < sum[s][p] + sum[p][f]$, иначе.

Тогда давайте посчитаем $sum[s][f]$ для всех $1 \leqslant s, f \leqslant n$. Для этого переберем все деревья, в каждом из них за $O(n^2)$ посчитаем расстояния между всеми парами вершин и прибавим их к значениям $sum[s][f]$. Эта часть решения работает за $O(n^2 k)$.

После того, как значения $sum[s][f]$ получены, для того чтобы для всех $1 \leqslant s, f \leqslant n$ получить ответ, просто переберем все возможные $1 \leqslant p \leqslant n$ и увеличим ответ на 1, если $sum[s][f] = sum[s][p] + sum[p][f]$. Эта часть решения работает за $O(n^3)$.

Получаем решение с временем работы $O(n^2(n+k))$.

Примечание: если считать расстояния между всеми парами вершин в каждом дереве, запуская `dfs` из каждой вершины в каждом дереве, то суммарное количество вызовов `dfs` будет $O(n^2k)$, что может быть долго из-за рекурсивности `dfs`. Такое решение «вписывается» в ограничение по времени почти вплотную. Для того, чтобы больше чем в три раза по времени ускорить решение, можно заметить, что для одного дерева можно посчитать попарные расстояния между всеми вершинами с помощью одного запуска `dfs`. Для этого будем считать глубины всех вершин, и, выходя из вершины p, можно посчитать расстояния для всех пар вершин, lca которых равен p. Чтобы перебрать все такие пары вершин, можно использовать эйлеров обход дерева.