3. ФИНАЛЬНЫЙ ЭТАП

Заключительный этап олимпиады состоит из двух частей: индивидуальное решение задач по предметам (математика, информатика) и командное решение инженерной задачи.

Задачи предметных туров составлены таким образом, чтобы обсудить некоторые ключевые моменты анализа данных на понятном языке школьной математики и информатики.

Так, например, темы задач **переобучение** и **разделяющая поверхность**, **скользящий контроль** и **стратификация** относятся к базовым понятиям машинного обучения. Темы задач**метод ближайшего соседа** и **линейный классификатор** относятся к основным алгоритмам.

Задачи индивидуального тура

На индивидуальное решение задач дается по 2 часа на один предмет. Для каждой из параллелей (9 класс или 10-11 класс) предлагается свой набор задач по математике, задачи по информатике — общие для всех участников.

Решение каждой задачи по математике дает определенное количество баллов (см. критерии оценки). При этом каждая задача делилась на 3 подзадачи таким образом, что решение каждой подзадачи подводило к решению следующей. За каждую подзадачу можно получить от 0 до указанного количества баллов.

Решение задач по информатике предполагало написание программ. Ограничения по используемым языкам программирования не было. Проверочные тесты для каждой задачи по информатике делились на несколько групп. Прохождение всех тестов в группе тестов дает определенное количество баллов за решение задачи.

Участники получают оценку за решение задач в совокупности по всем предметам данного профиля (математика и информатика) — суммарно от 0 до 200 баллов.

9.1. Задачи по математике (9 класс)

Задача 9.1.1. Переобучение (20 баллов)

На плоскости даны n точек. Плоскость необходимо разделить прямыми таким образом, чтобы никакие две точки не лежали в одной и той же части плоскости.

1. (4 балла) Всегда ли достаточно $n-1$ прямой, чтобы разделить плоскость таким образом?
2. (6 баллов) Приведите пример, когда $n-2$ прямых не достаточно.
3. (10 баллов) Существует ли расположение точек, при котором $[\sqrt{2n}]+1$ прямых будет достаточно?

Решение

1. Всегда: каждая прямая отделяет одну точку. Доказательство по индукции. База: для любых двух точек можно провести прямую таким образом, чтобы точки оказались в разных полуплоскостях. Шаг: пусть для любого количества t точек, такого, что $t \leqslant n$, можно провести $t-1$ прямую требуемым образом. Рассмотрим $n+1$ точку. Всегда можно провести прямую таким образом, чтобы не все точки оказались в одной полуплоскости. Пусть после проведения этой прямой в полуплоскостьях оказалось n_1 и n_2 точек. $n_1+n_2=n+1$, $0<n_1\leqslant n$, $0<n_2\leqslant n$. Воспользуемся предположением индукции, и проведем n_1-1 и n_2-1 прямую таким образом, чтобы никакие две точки не лежали в одной и той же части плоскости. Получим, что для этого было проведено $1+(n_1-1)+n_2-1=n$ прямых. Что и требовалось доказать. Вывод: для любого конечного количества точек n можно провести $n-1$ прямую таким образом, чтобы никакие две точки не лежали в одной и той же части плоскости.
2. Пример: n точек, лежащих на одной прямой, невозможно разделить $n-2$ прямыми. Доказательство: рассмотрим отрезки, соединяющие «соседние» точки. Таких отрезков $n-1$. Чтобы две соседние точки лежали в разных частях плоскости, необходимо, чтобы прямая пересекала соответствующий отрезок. Однако ни одна из прямых не пересекает более 1 отрезка. Следовательно, необходимо не менее $n-1$ прямой, чтобы никакие две точки не лежали в одной и той же части плоскости.
3. По индукции доказывается, что максимальное количество частей, на которых k прямых делит плоскость, равна $1+0,5(k(k+1))$. Действительно, пусть на плоскости уже имеется t прямых, которые делят плоскость на $1+0,5(t(t+1))$ части. Тогда $t+1$ прямая пересекает каждую из остальных не более одного раза, то есть остальные прямые делят ее не более чем на $t+1$ часть. Но тогда $t+1$ прямая добавляет не более $t+1$ части плоскости. В случае если $(t+1)$-я прямая пересекается со всеми остальными прямыми, добавляется ровно $t+1$ часть. Тогда максимальное возможное количество плоскостей $t+1+1+0,5(t(t+1))=1+0,5((t+1)(t+2))$, что и требовалось доказать. Итак, пусть дано $[\sqrt{2n}]+1$ прямых. Тогда максимальное количество частей, на которые они делят плоскость, равно $1+(([\sqrt{2n}]+1)([\sqrt{2n}]+2))/2>1+n$. То есть если поместить точки по одной в каждой части плоскости, получим

требуемое расположение

Ответ: 1) да, 3) да

Дополнительные критерии оценки

В пункте (2) полный балл ставился за правильный пример даже без обоснований. В пункте (3) за пример без обоснований ставился 1 балл.

Задача 9.1.2. Разделяющие поверхности и ГМТ (10 баллов)

Рассмотрим две функции:

$$\rho(A,B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

$$\rho^*(A,B) = |x_1 - x_2| + |y_1 - y_2|.$$

1. (2 балла) Существуют ли такие точки A и B на плоскости, что $\rho^*(A,B) > \rho(A,B)$? Если да, приведите пример.
2. (2 балла) Существуют ли такие точки A и B на плоскости, что $\rho^*(A,B) < \rho(A,B)$? Если да, приведите пример.
3. (2 балла) Дана точка $O(1,2)$. Найдите ГМ таких точек X, что $\rho^*(O,X) = 5$. Изобразите это множество на координатной плоскости.
4. (4 балла) Даны точки $A(3,5)$ и $B(7,10)$. Найдите ГМ таких точек X, что $\rho^*(A,X) = \rho^*(B,X)$. Изобразите это множество на координатной плоскости.

Решение

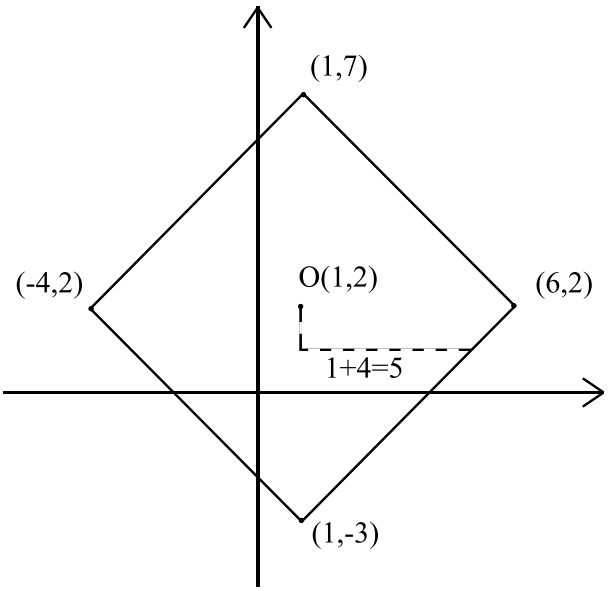
1. выполняется всегда, если обе координаты точек различны:

$$(\rho^*(A,B))^2 = (|x_1 - x_2| + |y_1 - y_2|)^2 > (x_1 - x_2)^2 + (y_1 - y_2)^2 = (\rho(A,B))^2$$

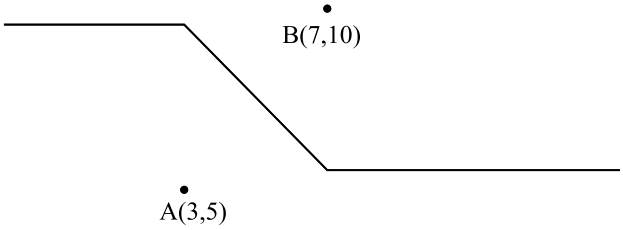
2. не выполняется никогда, если обе координаты точек различны:

$$(\rho^*(A,B))^2 = (|x_1 - x_2| + |y_1 - y_2|)^2 > (x_1 - x_2)^2 + (y_1 - y_2)^2 = (\rho(A,B))^2$$

3. Множество представляет собой границу квадрата с вершинами $(1;7)$, $(1;-3)$, $(-4;2)$, $(6;2)$.



4. решение представляет собой кусочно-линейную функцию:
   $x < 3$ горизонтальная прямая на высоте 9,5
   x от 3 до 7 отрезок соединяющий точки (3; 9,5) и (7; 5,5)
   $x > 7$ горизонтальная прямая на высоте 5,5



Ответ: 1) да, 2) нет

Задача 9.1.3. Профиль компактности и метод ближайшего соседа (20 баллов)

На плоскости расположены k черных и t белых точек. Все попарные расстояния между точками различны. Для каждой точки пронумеруем остальные по возрастанию расстояния. Первый сосед — ближайшая точка и т.д. Обозначим $P(m)$ долю точек, у которых m-й сосед другого цвета.

1. (2 балла) Может ли $P(1)$ быть равным 0?
2. (3 балла) Для каждого возможного m от 1 до $k+t-1$ подсчитаем $P(m)$. Найдите наибольшее возможное количество различных значений m, для которых $P(m) = 0$.
3. (5 баллов) Посчитайте $P(1) + P(2) + \ldots + P(k + t - 1)$.
4. (10 баллов) Пусть из выборки изъяли 1 объект, после чего значение $P(1)$ уменьшилось на некоторое число δ. Найдите максимальное возможное δ.

Решение

1. может: группы черных и белых точек расположены далеко друг от друга
2. каждая черная точка имеет ровно $k-1$ черного соседа и t белых; каждая белая точка имеет ровно $t-1$ белого соседа и k черных. Значит, количество различных нулевых профилей компактности не более $min(k-1, t-1)$. Пример: группы черных и белых точек расположены далеко друг от друга
3. каждая черная точка вносит в сумму вклад $k/(k+t)$, поскольку имеет ровно k соседей другого класса; аналогично белые точки вносят вклад $t/(k+t)$. И того $2tk/(k+t)$.
4. Точки расположены на плоскости и попарные расстояния различны. Следовательно, точка может быть ближайшей не более чем для 5 других точек. (Указание: предположим, что точка является ближайшей для 6 других; проводим окружность с центром в рассматриваемой точке и радиусом равным расстоянию до 6й (самой дальней) точки; рассматриваются секторы окружности, в которых не может быть точек.)

   Значит количество ближайших соседей чужого класса при таком удалении меняется не более чем на 1+5. Тогда получим $P(1) - \frac{(t+k)P(1)-6}{t+k-1} \leqslant \frac{6}{t+k-1}$. Равенство достигается когда группы черных и белых точек расположены далеко друг от друга кроме одной единственной точки, попавшей в «чужую» группу.

Ответ: 1) может, 2) min(k-1, t-1), 3) 2tk/(k+t), 4) 6/(t+k-1)

Дополнительные критерии оценки

В пункте (4) за идею решения ставилось 3 балла.

Задача 9.1.4. Скользящий контроль (30 баллов)

В мешке m черных и n белых шарика. Из мешка случайным образом извлекают k шариков.

1. (5 баллов) Пусть $k = 9$, $m = 100$, $n = 400$. Какова вероятность, того, что среди извлеченных шариков не меньше 8 белых.
2. (10 баллов) Пусть $k = 9$, $m = 100$, $n = 400$. Какова вероятность того, что среди k извлеченных шариков доля черных шариков больше чем в два раза превышает долю черных шариков в мешке.
3. (15 баллов) Пусть $m = 5$, $n = 20$. Какого наименьшего k достаточно, чтобы с вероятностью не менее 0,7 доля черных шариков среди выбранных k не превышала долю черных шариков в мешке более чем в 2 раза?

Решение

1. $(C_{100}^0 C_{400}^9 + C_{100}^1 C_{400}^8)/C_{500}^9$
2. $(C_{100}^0 C_{400}^9 + C_{100}^1 C_{400}^8 + C_{100}^2 C_{400}^7)/C_{500}^9$, поскольку данное условие означает, что белых шариков не менее 7.
3. подсчет аналогичным образом, ответ 1

Ответ: 1) $(C_{100}^0 C_{400}^9 + C_{100}^1 C_{400}^8)/C_{500}^9$, 2) $(C_{100}^0 C_{400}^9 + C_{100}^1 C_{400}^8 + C_{100}^2 C_{400}^7)/C_{500}^9$, 3) 1

Задача 9.1.5. Стратификация классов (20 баллов)

Дана стандартная колода из 36 игральных карт. Из колоды случайно равновероятно извлекают 8 карт. Порядок, в котором извлекались карты, не важен.

1. (4 балла) Сколько всего существует способов извлечь таким образом 8 карт?
2. (6 баллов) Сколько существует способов извлечь 8 карт так, что среди этих 8 карт было ровно по 2 карты каждой масти.
3. (10 баллов) В скольких случаях, описанных в предыдущих двух пунктах, номиналы хотя бы двух карт из 8 совпадают.

Решение

1. C_{36}^8 по определению.
2. Каждая масть выбирается независимо, следовательно $(C_9^2)^4$.
3. Всего номиналов 9, а карт выбирается 8. Тогда в первом пункте количество способов выбрать карты так, чтобы номиналы НЕ совпадали равно: 8 способов – выбрать отсутствующий номинал, 4 способа выбрать масть каждого номинала, и того $8 \cdot 4^8$. Тогда для первого пункта $C_{36}^8 - (8 \cdot 4^8)$.
   Аналогично для второго пункта $(C_9^2)^4 - (8 * C_8^2 C_6^2 C_4^2)$.

Ответ: 1)C_{36}^8, 2)$(C_9^2)^4$, 3) $C_{36}^8 - (8 \cdot 4^8)$ и $(C_9^2)^4 - (8 * C_8^2 C_6^2 C_4^2)$ соответственно.

9.2. Задачи по математике (10-11 класс)

Задача 9.2.1. Сглаживание (30 баллов)

Значение функции $f(t)$ измеряется при целых значениях t. Введем операцию, которую мы будем называть r-усреднение: по функции $f(t)$ строим новую функцию

$$f_r(t) = \frac{f(t-r) + f(t-r+1) + \ldots + f(t) + f(t+1) + \ldots f(t+r)}{2r+1}$$

Число r натуральное.

1. (4 балла)
   $$f(t) = t + \sin(\frac{\pi}{10}t)$$
   При каком наименьшем значении r функция $f_r(t)$ будет совпадать с некоторой линейной функцией в целых точках t?
2. (6 баллов)
   $$f(t) = t + \sin(\pi\frac{m}{n}t),$$
   где m и n натуральные взаимнопростые числа. При каком наименьшем значении r функция $f_r(t)$ будет совпадать с некоторой линейной функцией в целых точках t?

3. (10 баллов)
$$f(t) = t + \sin(t)$$
Существует ли r, такое, что функция $f_r(t)$ будет совпадать с некоторой линейной функцией в целых точках t?

4. (10 баллов)
$$f(t) = t + \sin(t)$$
Пусть задано небольшое действительное число $\varepsilon > 0$. Существует ли r и линейная функция $l(t)$, такие, что $|f_r(t) - l(t)| < \varepsilon$ во всех целых точках t?

Решение

Условие слегка менялось: рассматривалось $r + 1$ слагаемое от $f(t)$ до $f(t + r)$.

Сделаем замену переменных $x = t + 0,5r$. Получим что x целое или полуцелое. При этом функция $f_r(x)$ будет симметричной относительно нового начала координат. Тогда поскольку модуль синуса не превышает 1, данная функция не отличается от x более чем на 1 ни при каких значениях x. Следовательно, данная линейная функция имеет вид $y = x+b$. Из симметрии относительно начала координат следует, что $b = 0$.

В пункте (1) и (2) достаточно умозрительных рассуждений о том, когда сумма синусов в двух соседних x не будет отличаться.

В пунктах (3) и (4) наиболее простое решение получается исходя из суммы синусов, получаемой по формуле сокращенного умножения:

$$\frac{1}{(r+1)sin(0,5)}\big(sin(0,5)sin(x-0,5r) + \ldots + sin(0,5)sin(x+0,5r)\big) =$$

$$= \frac{1}{(r+1)sin(0,5)}(cos(x-0,5r-0,5) - cos(x+0,5r+0,5))$$

модуль данного выражения легко оценить сверху, поскольку разность косинусов не превышает 2, а $sin(0,5)$ строго больше нуля. Таким образом получается решение пункта (4). Решение пукнта (3) следует из того, что равенство между $cos(x-0,5r-0,5)$ и $cos(x+0,5r+0,5)$ невозможно поскольку число π иррационально.

Ответ: 1) 20, 2) $2n$, 3) нет, 4) да.

Дополнительные критерии оценки

Отсутствуют.

Задача 9.2.2. Разделяющие поверхности и ГМТ (10 баллов)

Расстоянием между окружностью и точкой A, лежащей вне окружности, называется расстоянию от точки A до ближайшей точки окружности.

1. (2 балла) Рассмотрим координатную плоскость. Найдите расстояние от точки с координатами $(5, 8)$ до окружности, заданной уравнением $(x+2)^2+(y-1)^2 = 1$

2. (3 балла) Найдите геометрическое место точек, равноудаленных от прямой $y = 1$ и точки с координатами $(5, 6)$.
3. (5 баллов) На плоскости заданы две непересекающиеся окружности. ГМТ, равноудаленных от этих двух окружностей, является прямая. Радиус первой окружности равен 10. Найдите радиус второй окружности. Зависит ли радиус второй окружности от расстояния между центрами?

Решение

Расстояние от точки до окружности равно расстоянию от точки до центра окружности вычесть радиус. На этой идее основаны вычисления.

1. Центр окружности имеет координаты $x_0 = -2$, $y_0 = 1$. Радиус $R = 1$.
$$\sqrt{(x - x_0)^2 + (y - y_0)^2} - R = 7\sqrt{2} - 1$$
2. Метод координат. $(y - 1)^2 = (x - 5)^2 + (y - 6)^2$, что эквивалентно $y = 0,1(x - 5)2 + 3,5$. То есть кривая – парабола.
3. Координатный метод. Доказывается, что радиусы равны. Пусть радиус второй окружности R. Систему координат удобно выбрать так, что центр первой окружности в точке $(0, 0)$, а второй $(L, 0)$, где L расстояние между центрами. Тогда из соображений симметрии и поскольку такая прямая одна по условию – она вертикальная. То есть $x = a$, где a некоторое число. Тогда
$$(a^2 + y^2)^{0,5} - 10 = ((a - L)^2 + y^2)^{0.5} - R$$
после нескольких преобразований получим
$$(R - 10)^2 + 2(R - 10)(a^2 + y^2)^{0,5} = L^2 - 2aL$$
выражение должно быть справедливо при всех y, что возможно только при $R = 10$.

Ответ: 1) $7\sqrt{2} - 1$, 2) парабола $y = 0,1(x - 5)^2 + 3,5$, 3) 10.

Дополнительные критерии оценки

Отсутствуют.

Задача 9.2.3. Профиль компактности и метод ближайшего соседа (20 баллов)

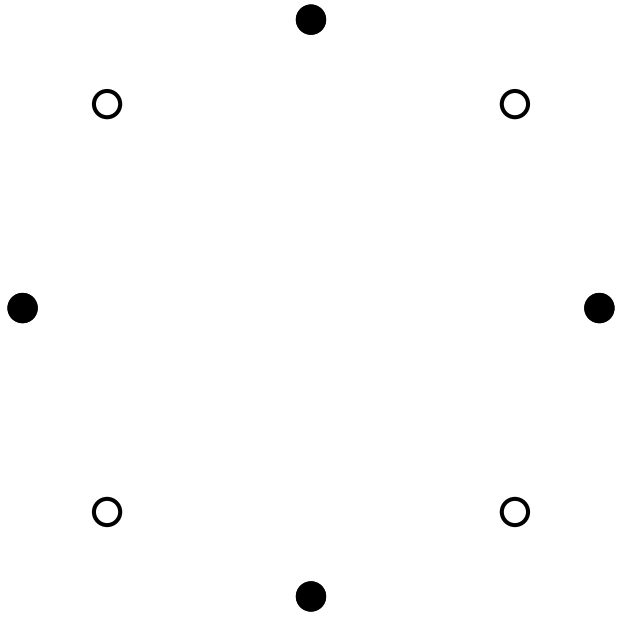
Дана плоскость, на которой мы можем нарисовать любое конечное количество черных и белых точек.

1. (4 балла) Возможно ли нарисовать черные и белые точки таким образом, чтобы для каждой точки две ближайшие к ней имели другой цвет? Если да, приведите пример.
2. (6 баллов) Возможно ли нарисовать черные и белые точки таким образом, чтобы для каждой точки три ближайшие к ней имели другой цвет? Если да, приведите пример.

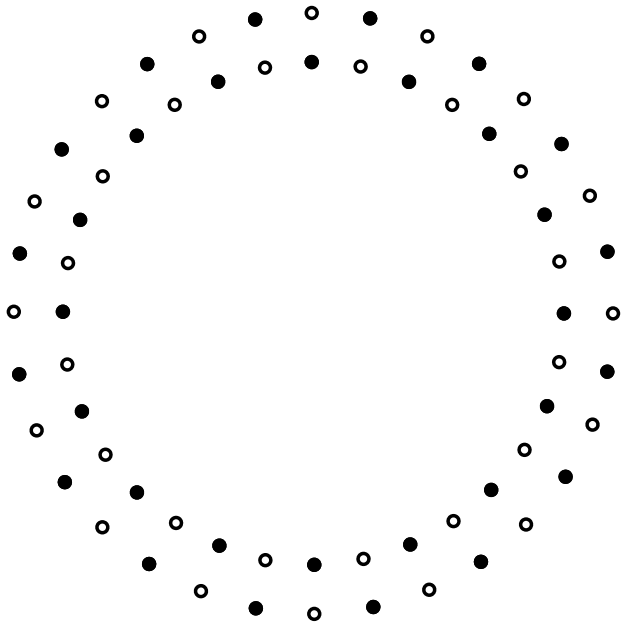
3. (10 баллов) Возможно ли нарисовать черные и белые точки таким образом, чтобы для каждой точки шесть ближайших к ней имели другой цвет? Если да, приведите пример.

Решение

1. Пример: 8 точек по окружности чередующегося цвета



2. Рассмотрим точки, расположенные по двум концентрическим окружностям с чередующимися цветами по 32 точки на каждой. Пусть радиус внешней окружности 6, а внутренней 5.



Для точек внутренней окружности ближайшие три точки чужого цвета находятся на расстояниях $2 \cdot 5 \cdot \sin(\frac{\pi}{32})$, $2 \cdot 5 \cdot \sin(\frac{\pi}{32})$, и 1. А ближайшие точки

своего цвета на расстоянии больше 1. Но $2 \cdot 5 \cdot \sin(\frac{\pi}{32})) < 2 \cdot 5 \cdot \frac{\pi}{32} < 1$. Следовательно, для точек внутренней окружности ближайшие три соседа имеют другой цвет.

Для точек внешней окружности три точки чужого цвета находятся на расстояниях $2 \cdot 6 \cdot \sin(\frac{\pi}{32})$, $2 \cdot 6 \cdot \sin(\frac{\pi}{32})$ и 1 .Ближайшие точки своего цвета находятся на расстоянии $\sqrt{(\cos^2 \frac{\pi}{32}2) + 121 \sin^2 \frac{\pi}{32}2)} = \sqrt{1 + 120 \sin^2(\frac{\pi}{32})}$. Обозначим $t = \sin(\frac{\pi}{32})$. Тогда сравним $\sqrt{1 + 120t^2}$ и $12t$; $1 + 120t^2$ и $144t^2$; 1 и $24t^2$. $24t^2 < 24 \cdot (\frac{\pi}{32})^2 < 24 \cdot (1/10)^2 < 1$. Следовательно, $\sqrt{1 + 120 \sin^2(\frac{\pi}{2})} > 2 \cdot 6 \cdot \sin(\frac{\pi}{32})$. А значит, ближайшие три точки имеют другой цвет, что и требовалось.

3. Возможны различные решения. Самое простое основывается на рассмотрении точки с минимальным расстоянием до 6-го соседа. Проводится окружность с центром в этой точке и проходящая через 6-го соседа. Рассматривая расстояния между этими 6 соседями получим, что среди них есть точка с меньшим расстоянием до 6 соседа, что противоречит условию.

Ответ: 1) да, 2) да, 3) нет.

Дополнительные критерии оценки

В пунктах (1) и (2) полный балл ставился даже за пример без объяснения.

Задача 9.2.4. Стратификация классов (20 баллов)

Племя дикарей не имеет проблем со здоровьем кроме того, что некоторые не умеют правильно питаться, и страдают от недоедания и переедания. Старый шаман по каждому человеку может точно определить нормально ли тот кушает, недоедает или переедает. Вождь хочет выбрать нового шамана, и для этого проводит конкурс. В конкурсе участвует три ученика шамана. Победит тот, кто поставит больше правильных диагнозов. Ничья не считается победой.

Первый ученик так ничему и не научился, потому любому человеку говорит, что тот кушает нормально.

Второй ученик всегда правильно определяет недоедающих, но всем остальным говорит, что они питаются нормально.

Третий же ученик недоедающему говорит, что тот недоедает, переедающему - что тот переедает, а вот тех кто кушает нормально, ошибочно относит к недоедающим или переедающим случайным образом.

Старый шаман осмотрел 100 человек, результат осмотра для каждого записал в свой секретный свиток. Оказалось, что 10 человек из списка недоедают, 10 переедают, а остальные кушают нормально.

Вождь, не зная результатов осмотра, должен выбрать из этих 100 человек k, которых будут осматривать три конкурсанта. Предположим, что вождю стало известно содержание секретного свитка.

1. (2 балла) Пуст k=10. Сможет ли вождь так подобрать людей, чтобы выиграл первый ученик?

2. (3 балла) Пуст k=10. Сможет ли вождь так подобрать людей, чтобы гарантировать победу третьему ученику?
3. (5 баллов)Пуст k=10. Сможет ли вождь так подобрать людей, чтобы гарантировать победу второму ученику?
4. (10 баллов) Ответьте на вопросы предыдущих трех пунктов при k=30.

Решение

Первый ошибается на недоедающих и переедающих, второй – только на переедающих, а третий на нормально питающихся. Поэтому:

1. Второй ученик не может проиграть первому, то есть первый не может выиграть.
2. Если взять 10 переедающих, то первые два ученика сделают 10 ошибок, а третий не сделает ни одной.
3. Если взять 1 недоедающего и 9 нормально питающихся, первый ученик ошибется 1 раз, второй – 0 раз, третий – 9 раз. Таким образом выиграет второй ученик.
4. В случае k=30 нормально питающихся будет не меньше 10, то есть не меньше чем переедающих. Второй ученик не может проиграть первому, а значит первый выиграть не может. Третий ошибается на нормально питающихся, а значит, сделает не менее 10 ошибок. Второй ошибается только на переедающих, которых не более 10. Следовательно, второй сделает не более 10 ошибок. То есть второй ученик не может проиграть. Он выигрывает, например, если взять 10 недоедающих и 20 питающихся нормально. Третий не может выиграть, поскольку второй не может проиграть.

Ответ: 1) нет, 2) да, 3) да, 4) нет-нет-да

Дополнительные критерии оценки

В пункте (4) за каждый подпункт ставилось 3 балла. Ставилось 10 баллов только если задача сделана полностью.

Задача 9.2.5. Baseline (20 баллов)

В мешке 300 красных 200 синих и 100 зеленых шариков. Фокусник **знает** сколько шариков каждого из цветов находится в мешке. Из мешка собираются случайным образом извлечь 2 шарика. Фокусник должен попытаться заранее угадать цвета шариков, которые будут извлечены. Важно угадать оба цвета, но не важно в каком порядке их называть (синий+красный=красный+синий). Оцените выероятность ошибки перечисленных ниже стратегий:

1. (4 балла) Фокусник говорит, что выпадет 2 красных шарика.
2. (6 баллов) Фокусник 2 раза бросает симметричную игральную кость (с равной вероятностью выпадения каждой грани) раскрашенную так, что ровно 2 грани красные, 2 синие, 2 зеленые.

3. (10 баллов) Фокусник 2 раза бросает симметричную игральную кость (с равной вероятностью выпадения каждой грани) раскрашенную так, что ровно 3 грани красные, 2 синие, 1 зеленая.

Решение

Вероятность ошибки равна 1 вычесть вероятность угадать

1. 1) Вероятность вытащить первый шарик красным равна 300/600. Вероятность после этого вытащить второй шарик 299/599. Таким образом вероятность вытащить два красных шарика равна $P_1 = (300 \cdot 299)/(600 \cdot 599)$.
2. Аналогично предыдущему пункту, вероятность вытащить два красных шарика $(300 \cdot 299)/(600 \cdot 599)$, два синих $(200 \cdot 199)/(600 \cdot 599)$, два зеленых $(100{\cdot}99)/(600{\cdot}599)$, красный и синий $(2{\cdot}300{\cdot}200)/(600{\cdot}599)$ и т.д. А вероятность выкинуть кубиками 2 одинаковых $(2/6) \cdot (2/6) = 1/9$, два разных (например, красный и синий + синий и красный) $(2/6) \cdot (2/6) + (2/6) \cdot (2/6) = 2/9$. Вероятность угадать в таком случае равна сумме произведений соответствующих случаев.

$$P_2 = \frac{1}{9}\Big(\frac{300 \cdot 299}{600 \cdot 599} + \frac{200 \cdot 199}{600 \cdot 599} + \frac{100 \cdot 99}{600 \cdot 599}\Big) + \\ + \frac{1}{9}\Big(\frac{2 \cdot 300 \cdot 200}{600 \cdot 599} + \frac{2 \cdot 100 \cdot 200}{600 \cdot 599} + \frac{2 \cdot 300 \cdot 100}{600 \cdot 599}\Big)$$

3. Аналогично предыдущему пункту.

$$P_3 = \frac{3 \cdot 3}{6 \cdot 6}\frac{300 \cdot 299}{600 \cdot 599} + \frac{2 \cdot 3}{6 \cdot 6}\frac{200 \cdot 199}{600 \cdot 599} + \frac{1 \cdot 1}{6 \cdot 6}\frac{100 \cdot 99}{600 \cdot 599} + \\ \frac{3 \cdot 2}{6 \cdot 6}\frac{2 \cdot 300 \cdot 200}{600 \cdot 599} + \frac{3 \cdot 1}{6 \cdot 6}\frac{2 \cdot 300 \cdot 100}{600 \cdot 599} + \frac{1 \cdot 2}{6 \cdot 6}\frac{2 \cdot 100 \cdot 200}{600 \cdot 599}$$

В результате первая статегия оказалась наиболее выгодной. **Ответ:** 1) $1 - P_1$, 2) $1 - P_2$, 3) $1 - P_3$

Дополнительные критерии оценки

Отсутствуют.

9.3. Задачи по информатике

Задача 9.3.1. Кусочно постоянные функции - 1 (4 балла)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт

Пусть у нас есть функция $f(x)$ определённая на целых числах из $[1; n]$. Мы хотим приблизить её решающим деревом, т.е., в дереве в каждой внутренней вершине будет стоять неравенство вида $x < a$, а в листьях будут значиться некоторые числа.

Когда мы захотим вычислить $f(x)$, мы начнём с корня, а затем в каждой вершине мы будем смотреть, удовлетворяет ли x находящемуся в ней неравенству. Если да, то мы направимся в левое поддерево текущей вершины, а иначе — в правое. Когда мы окажемся в листе, мы скажем, что значение, которое в нём находится и есть приближенное значение $f(x)$.

Вам даны значения некоторой функции $f(x)$ в точках $1, \ldots, n$. Постройте дерево, которое позволит вычислить функцию $f(x)$ для любого $x \in \{1, \ldots, n\}$ такое что наибольшее число сравнений, требуемое для того, чтобы по нему вычислить некоторое значение $f(x)$ будет минимально.

Формат входных данных

В первой строке ввода задано число n ($1 \leqslant n \leqslant 10^4$).

В следующей строке задано n целых чисел $f(1), f(2), \ldots, f(n)$ ($1 \leqslant f(i) \leqslant 10^9$).

Формат выходных данных

Первая строка должна содержать единственное число m ($1 \leqslant m \leqslant 2n$) — количество вершин в искомом дереве.

Каждая из следующих m строк должна удовлетворять следующему формату:

1. k-я строка соответствует информации о вершине под номером k.
   При этом первая вершина обязана быть корнем дерева.
2. Если k-я вершина внутренняя, то k-я строка должна иметь вид `x < a ? left : right`
   Вместо a должно быть вставлено целое число от 1 до n, с которым происходит сравнение в данной вершине, вместо $left$ и $right$ — номера левого и правого потомков вершины соответственно (целые числа от 2 до m). Дерево должно быть полным, т.е. у каждой внутренней вершины должно быть ровно два потомка.
3. Если k-я вершина является листом, то k-я строка должна иметь вид `! y`
   Где y — целое число ($1 \leqslant y \leqslant 10^9$), которое дерево выдаст для x, которые придут в этот лист.

Если вариантов ответа несколько, выведите любой.

Примеры

Пример №1

Стандартный ввод
4 1 2 3 4

Стандартный вывод
7 x < 3 ? 2 : 3 x < 2 ? 4 : 5 x < 4 ? 6 : 7 ! 1 ! 2 ! 3 ! 4

Пример №2

Стандартный ввод
6 1 1 2 2 3 3

Стандартный вывод
5 x < 3 ? 2 : 3 ! 1 x < 5 ? 4 : 5 ! 2 ! 3

Решение

Для начала нужно «сжать» идущие подряд одинаковые значения, чтобы они в итоге соответствовали одному листу в дереве. Теперь мы знаем число листьев (меньше мы получить не сможем) и нам нужно построить над ними двоичное дерево наименьшей высоты.

Чтобы этого добиться, нужно строить дерево рекурсивно, при этом каждый раз разделяя множество листьев, для которых мы строим дерево на примерно равные половины – это выгодно делать, так как наименьшая высота растёт монотонно от числа листьев и нам всегда выгоднее иметь как можно меньше листьев в большей половине, чего мы и добиваемся делением поровну.

Ниже представлено решение на языке C++

```
#include <bits/stdc++.h>

using namespace std;

const int maxn = 2e4 + 42;

int sz = 2;
string ans[maxn];

template<typename iter>
void build(iter l, iter r, int v = 1) {
    if(r - l == 1) {
        ans[v] = "! " + to_string(l->second);
```

```
14          return;
15      }
16      iter m = l + (r - l) / 2;
17      int left = sz, right = sz + 1;
18      ans[v] = "x < " + to_string(m->first) + " ? " + to_string(left) + " : " + to_string(right);
19      sz += 2;
20      build(l, m, left);
21      build(m, r, right);
22  }
23
24  signed main() {
25      //freopen("input.txt", "r", stdin);
26      ios::sync_with_stdio(0);
27      cin.tie(0);
28      int n;
29      cin >> n;
30      int a[n];
31      for(int i = 0; i < n; i++) {
32          cin >> a[i];
33      }
34      vector<pair<int, int>> pos = {{1, a[0]}};
35      for(int i = 2; i <= n; i++) {
36          if(a[i - 2] != a[i - 1]) {
37              pos.emplace_back(i, a[i - 1]);
38          }
39      }
40      build(begin(pos), end(pos));
41      cout << sz - 1 << endl;
42      for(int i = 1; i < sz; i++) {
43          cout << ans[i] << endl;
44      }
45      return 0;
46  }
```

Задача 9.3.2. Кусочно постоянные функции - 2 (7 баллов)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт

Эта задача является второй из трёх задач про решающие деревья. Обратитесь к первой задаче за пояснением о том, что это такое. Вам даны значения некоторой функции $f(x)$ в точках $1, \ldots, n$ и нужно построить дерево, приближающее $f(x)$ для $x \in \{1, \ldots, n\}$. При этом число сравнений, которые будут произведены для любого такого x не должно превышать заданного числа k.

Так как в этой задаче может не быть возможным задать функцию деревом точно, вам нужно найти такое дерево, которое будет минимизировать сумму квадратов отклонений:

$$penalty = \sum_{i=1}^{n} (f(i) - f'(i))^2$$

Здесь через $f'(i)$ обозначены значения, которые на данном i выдаст дерево.

Формат входных данных

В первой строке ввода заданы два числа n и k ($1 \leqslant n, k \leqslant 100$).

В следующей строке задано n целых чисел $f(1), f(2), \ldots, f(n)$ ($1 \leqslant f(i) \leqslant 10^5$).

Формат выходных данных

Первая строка должна содержать единственное число m ($1 \leqslant m \leqslant 2n$) — количество вершин в искомом дереве.

Каждая из следующих m строк должна удовлетворять следующему формату:

1. k-я строка соответствует информации о вершине под номером k.
   При этом первая вершина обязана быть корнем дерева.
2. Если k-я вершина внутренняя, то k-я строка должна иметь вид `x < a ? left : right`
   Вместо a должно быть вставлено целое число от 1 до n, с которым происходит сравнение в данной вершине, вместо $left$ и $right$ — номера левого и правого потомков вершины соответственно (целые числа от 2 до m). Дерево должно быть полным, т.е. у каждой внутренней вершины должно быть ровно два потомка.
3. Если k-я вершина является листом, то k-я строка должна иметь вид `! y`
   Где y — целое число ($1 \leqslant y \leqslant 10^5$), которое дерево выдаст для x, которые придут в этот лист.

Если вариантов ответа несколько, выведите любой.

Примеры

Пример №1

Стандартный ввод
4 1 1 2 3 4

Стандартный вывод
3 x < 2 ? 2 : 3 ! 1 ! 3

Пример №2

Стандартный ввод
6 1 1 1 2 2 3 3

Стандартный вывод
3 x < 3 ? 2 : 3 ! 1 ! 2

Решение

Для решения этой задачи нужно было воспользоваться либо динамическим программированием по подотрезкам, либо рекурсией с "мемоизацией". То есть, если в данный момент мы рассматриваем значения функции на отрезке $[l; r]$, можно перебрать число m, которое будет в вершине и рекурсивно запустится, чтобы узнать, какие будут отклонения на $[l; m)$ и $(m; r]$. Затем можно выбрать наилучший m. При этом, чтобы отработать за разумное время нужно хранить результат для каждого посчитанного $[l; r]$, и если мы пришли к нему повторно, выдавать посчитанный, а не перебирать m заново. Так мы получим полиномиальный алгоритм.

Ниже представлено решение на языке C++

```
#include <bits/stdc++.h>

using namespace std;

#define int int64_t

const int maxn = 200;

int a[maxn];
int dp[maxn][maxn][maxn];
int pre[maxn][maxn][maxn];

int sqr(int x) {
    return x * x;
}

//Возвращает десперсию (квадрат отклонения) всех чисел с отрезка a[l]...a[r] и числа t.
int dispersion(int l, int r, int t) {
    int ans = 0;
    for(int i = l; i < r; i++) {
        ans += sqr(t - a[i]);
    }
    return ans;
}

//Возвращает оптимальный центр отрезка в зависимости от значения дисперсии
int optimal(int l, int r) {
    int mid = accumulate(a + l, a + r, 0LL) / (r - l);
    int d1 = dispersion(l, r, mid);
    int d2 = dispersion(l, r, mid + 1);
    if(d1 <= d2) {
        return mid;
    } else {
        return mid + 1;
    }

}
/*Функция от отрезка l...r и количетсва шагов, которые остались:
Перебираем m рекурсивно(уменьшая k - кол-во шагов на 1) и находим наилучшую (если отрезок длины один
int pre_build(int l, int r, int k) {
    if(r - l == 1 || k == 0) {
        pre[l][r][k] = optimal(l, r);
        dp[l][r][k] = dispersion(l, r, pre[l][r][k]);
        return dp[l][r][k];
    }
    if(dp[l][r][k] == -1) {
```

```
        for(int m = l + 1; m < r; m++) {
            //if(a[m - 1] == a[m]) {
            //    continue;
            //}
            int A = pre_build(l, m, k - 1);
            int B = pre_build(m, r, k - 1);
            if(dp[l][r][k] == -1 || A + B < dp[l][r][k]) {
                dp[l][r][k] = A + B;
                pre[l][r][k] = m;
            }
        }
    }
    return dp[l][r][k];
}

int sz = 2;
string ans[maxn];

void build(int l, int r, int k, int v = 1) {
    //Если сжали отрезок до единицы или закончились шаги
    if(r - l == 1 || k == 0) {
        ans[v] = "! " + to_string(pre[l][r][k]);
    } else {
        //Иначе находим центр, выводим вопрос и запускаем build от двух половинок отрезка l...r
        int m = pre[l][r][k];
        int left = sz, right = sz + 1;
        ans[v] = "x < " + to_string(m + 1) + " ? " +
                          to_string(left) + " : " +
                          to_string(right);
        sz += 2;
        build(l, m, k - 1, left);
        build(m, r, k - 1, right);
    }
}

signed main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    memset(dp, -1, sizeof(dp));
    //считываем данные, запускаем функции, выводим ответ
    int n, k;
    cin >> n >> k;
    k = min(k, (int)10);
    for(int i = 0; i < n; i++) {
        cin >> a[i];
    }
    pre_build(0, n, k);
    build(0, n, k);
    cout << sz - 1 << endl;
    for(int i = 1; i < sz; i++) {
        cout << ans[i] << endl;
    }
    return 0;
}
```

Задача 9.3.3. Кусочно постоянные функции - 3 (10 баллов)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод

Ограничение по времени: 5 секунд
Ограничение по памяти: 256 мегабайт

Эта задача является последней из трёх задач про решающие деревья. Обратитесь к первой задаче за пояснением о том, что это такое. Вам даны значения некоторой функции $f(x)$ в точках $1, \ldots, n$ и нужно построить дерево, приближающее $f(x)$ для $x \in \{1, \ldots, n\}$. При этом число сравнений, которые будут произведены для любого такого x не должно превышать заданного числа k. Дополнительно, необходимо, чтобы ваше дерево было строго лучше дерева, получаемого *жадным алгоритмом*.

Назовём штрафом дерева сумму квадратов отклонений выдаваемых им значений от заданных:

$$penalty = \sum_{i=1}^{n} (f(i) - f'(i))^2$$

Здесь через $f'(i)$ обозначены значения, которые на данном i выдаст дерево.

Будем считать, что дерево T_1 *строго лучше* дерева T_2, если его штраф строго меньше штрафа T_2.

Определим функцию

$$optimal(l, r) = \min_{y \in \mathbb{Z}} \sum_{i=l}^{r} (f(i) - y)^2$$

Функция $optimal(l, r)$ равна минимальному штрафу, который можно получить выбором значения y в листе, в который попал отрезок координат $[l, r]$.

Опишем рекурсивный *жадный алгоритм* построения решающего дерева:

Вход алгоритма: отрезок координат $[l, r]$, над которыми нужно построить решающее дерево. Максимально допустимое число сравнений H, которое можно совершить над точками из $[l, r]$ (алгоритм возвращает поддерево, число сравнений в котором ни для какой точки не превышает H).

Выход алгоритма: решающее поддерево для отрезка $[l, r]$, максимальное число сравнений в котором для любой точки не превышает H. В частности, если $H = 0$, обязательно возвращается лист.

1. Если $l = r$ (отрезок состоит из единственной точки), или если $H = 0$, вернуть лист. Значение в листе y выбирается как y, на котором достигается минимум штрафа в листе, равный $optimal(l, r)$.

2. Иначе $r > l$. Выберем точку i для разбиения отрезка по условию $x < i$, как $i \in [l, r - 1]$, на котором достигается минимум выражения

$$optimal(l, i) + optimal(i + 1, r)$$

Иными словами, разбиение выбирается из предположения, что поддеревья не будут содержать сравнений. Если минимум выражения достигается на нескольких i, выберем минимальное (пытаемся разделить отрезок как можно левее).

3. Построим решающие поддеревья для отрезков $[l, i]$ и $[i + 1, r]$ с максимальным числом сравнений $H - 1$ рекурсивно. Объединим их в одно решающее дерево, сделав их сыновьями вершины со сравнением $x < i$.

Начальные значения: На вход жадному алгоритму подаётся $l = 1, r = n$, $H = k$.

В примечании к примеру входных и выходных данных приведён пример результата работы жадного алгоритма.

Постройте любое решающее дерево, *penalty* которого **строго меньше**, чем у дерева, которое строит *жадный алгоритм*. Значения функции $f(i)$ в тестах **случайные**, внимательно прочитайте формат ввода.

Формат входных данных

В первой строке ввода заданы два целых числа n и k. Во всех тестах, кроме теста из условия, $n = 4000$, а k может принимать одно из трёх значений: 5, 8 или 10. На тесте из условия любое дерево, подходящее под формат вывода, будет принято.

В следующей строке задано n целых чисел $f(1), f(2), \ldots, f(n)$ $(1 \leqslant f(i) \leqslant 1000)$.

В тестах этой задачи значения $f(1), \ldots, f(n)$ – незавимые случайные целые числа от 1 до 1000.

Формат выходных данных

Первая строка должна содержать единственное число $m (1 \leqslant m \leqslant 3n)$ — количество вершин в искомом дереве.

Каждая из следующих m строк должна удовлетворять следующему формату:

1. k-я строка соответствует информации о вершине под номером k.
   При этом первая вершина обязана быть корнем дерева.
2. Если k-я вершина внутренняя, то k-я строка должна иметь вид `x < a ? left : right`
   Вместо a должно быть вставлено целое число от 1 до n, с которым происходит сравнение в данной вершине, вместо $left$ и $right$ — номера левого и правого потомков вершины соответственно (целые числа от 2 до m). Дерево должно быть полным, т.е. у каждой внутренней вершины должно быть ровно два потомка.
3. Если k-я вершина является листом, то k-я строка должна иметь вид `! y`
   Где y — целое число $(1 \leqslant y \leqslant 2000)$, которое дерево выдаст для x, которые придут в этот лист.

Если вариантов ответа несколько, выведите любой.

Примеры

Пример №1

Стандартный ввод
6 2 10 2 3 5 6 1

Стандартный вывод
7 x < 4 ? 2 : 5 x < 2 ? 3 : 4 ! 10 ! 3 x < 6 ? 6 : 7 ! 6 ! 1

Решение

Несмотря на длинное и сложное условие, жюри было достаточно лояльно к участникам в данной задаче. Нужно было всего лишь добиться хоть какого-то улучшения по сравнению с описанной жадной стратегией. Для этого можно было, например, реализовать саму жадную стратегию, а затем, например, написать решение, которое пробует не только оптимальную точку из этого решения, но и небольшее число точек около неё. Также имело смысл решать задачу оптимальным алгоритмом из прошлой задачи когда размер массива в рекурсивном запуске становился достаточно маленьким. На случайных функциях $f(x)$ это гарантировало улучшение по сравнению с жадным алгоритмом.

Ниже представлено решение на языке C++

```
#include <bits/stdc++.h>

using namespace std;

using li = long long;

struct Quality {
    int n = 0;
    vector<li> sum, squareSum;
    Quality() {}

    void build(vector<int> a) {
        n = a.size();
        sum.resize(n);
        squareSum.resize(n);

        for (int i = 0; i < n; ++i) {
            sum[i] = a[i] + (i ? sum[i - 1] : 0);
            squareSum[i] = a[i] * (li)a[i] + (i ? squareSum[i - 1] : 0);
        }
    }
    // query sum (x-a[i])^2
    li query(int l, int r, int x) {
        int len = r - l + 1;
        assert(len >= 0);

        li D = len * x * (li)x + squareSum[r] - (l ? squareSum[l - 1] : 0);
        D -= 2 * (li)x * (sum[r] - (l ? sum[l - 1] : 0));

```

```
30          assert(D >= 0);
31          return D;
32      }
33      pair<li, int> optimal(int l, int r) {
34          assert(r >= l);
35          int avg = (sum[r] - (l ? sum[l - 1] : l)) / (r - l + 1);
36
37          li D1 = query(l, r, avg);
38          li D2 = query(l, r, avg + 1);
39
40          if (D1 < D2)
41              return make_pair(D1, avg);
42          else
43              return make_pair(D2, avg + 1);
44      }
45  };
46
47  int n, k;
48
49  struct GreedySolver {
50      Quality q;
51      vector<int> f;
52
53      int nVertices = 0;
54      vector<string> answer;
55
56      pair<li, int> solve(vector<int> f, int bf_depth) {
57          this->f = f;
58          q.build(f);
59          return solve(0, n - 1, 0, bf_depth);
60      }
61
62      pair<li, int> solve(int l, int r, int depth, int bf_depth = 0,
63                                                   bool build_answer = true) {
64          if (depth == k || l == r) {
65              auto opt = q.optimal(l, r);
66              li ans = opt.first;
67              int cur = nVertices + 1;
68              if (build_answer) {
69                  answer.push_back("");
70                  ++nVertices;
71                  answer[cur - 1] = "! " + to_string(opt.second);
72              }
73              return make_pair(ans, cur);
74          }
75
76          li minCan = -1;
77          int bestSplit = 0;
78
79          for (int i = l; i < r; ++i) {
80              li can = q.optimal(l, i).first + q.optimal(i + 1, r).first;
81              if (minCan < 0 || can < minCan) {
82                  minCan = can;
83                  bestSplit = i;
84              }
85          }
86
87          assert(minCan != -1);
88
89          int cur = nVertices + 1;
```

```
90          if (build_answer) {
91              answer.push_back("");
92              ++nVertices;
93          }
94
95          if (bf_depth > 0) {
96              li bestCan = -1;
97
98              const int OFFSET = bf_depth == 3 ? 3 : 1;
99              for (int t = max(l, bestSplit - OFFSET); t <= min(r - 1, bestSplit +
100                                                                  OFFSET); ++t) {
101                 auto left = solve(l, t, depth + 1, bf_depth - 1, false);
102                 auto right = solve(t + 1, r, depth + 1, bf_depth - 1, false);
103                 if (bestCan == -1 || left.first + right.first < bestCan) {
104                     bestCan = left.first + right.first;
105                     bestSplit = t;
106                 }
107             }
108         }
109
110         if (build_answer) {
111             nVertices = cur;
112             answer.resize(nVertices);
113         }
114         auto left = solve(l, bestSplit, depth + 1, max(bf_depth - 1,
115                                                        0), build_answer);
116         auto right = solve(bestSplit + 1, r, depth + 1, max(bf_depth - 1,
117                                                             0), build_answer);
118         if (build_answer)
119             answer[cur - 1] = "x < " + to_string(bestSplit + 1 + 1) + " ? " +
120                                       to_string(left.second) + " : " +
121                                       to_string(right.second);
122
123         li sub = left.first + right.first;
124         return make_pair(sub, cur);
125     }
126
127     void print() {
128         cout << nVertices << "\n";
129         assert(answer.size() == nVertices);
130         for (const string& s: answer)
131             cout << s << "\n";
132     }
133 };
134
135
136 int main() {
137     cin >> n >> k;
138
139     vector<int> f(n);
140     for (int i = 0; i < n; ++i)
141         cin >> f[i];
142
143     GreedySolver gs;
144     auto p = gs.solve(f, 3);
145     gs.print();
146
147     return 0;
148 }
```

Задача 9.3.4. Метод ближайшего соседа - 1 (5 баллов)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт

На равнине находится маленькая деревня, состоящая из n домиков. Известно, что попарные расстояния между всеми домиками различны. Обитатели каждого домика прокладывают дорожку к k ближайшим домикам. Других дорожек в деревне нет. Найдите наименьшее k такое, что для любого расположения домиков между любыми двумя будет существовать путь по дорожкам.

Формат входных данных

На первой строке входного файла расположено целое число n ($1 \leqslant n \leqslant 10^9$) — число домиков в деревне.

Формат выходных данных

Выведите наименьшее k такое, что для любого расположения домиков между любыми двумя будет существовать путь по дорожкам.

Примеры

Пример №1

Стандартный ввод
2
Стандартный вывод
1

Пример №2

Стандартный ввод
3
Стандартный вывод
1

Решение

В данной задаче можно легко показать, что ответ равен $k = \lfloor n/2 \rfloor$. При таком k действительно гарантирована связность, а при меньшем k можно взять группу из $k + 1$ людей и условно унести её на бесконечность, чтобы получившиеся группы из $k + 1$ и $n - (k + 1)$ людей образовали связи только внутри групп.

Ниже представлено решение на языке C++

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

mt19937 rnd(228);

int main() {
    int n;
    cin >> n;
    cout << n / 2 << '\n';
}
```

Задача 9.3.5. Метод ближайшего соседа - 2 (7 баллов)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт

На равнине находится маленькая деревня, состоящая из n домиков. Известны координаты каждого домика. Гарантируется, что попарные расстояния между всеми домиками различны. Обитатели каждого домика прокладывают дорожку к k ближайшим домикам. Других дорожек в деревне нет. Найдите наименьшее k такое, что между любыми двумя домиками будет существовать путь по дорожкам.

Формат входных данных

На первой строке входного файла расположено целое число n ($2 \leqslant n \leqslant 10^3$) — число домиков в деревне.

В следующих n строках расположены два целых числа x_i, y_i ($0 \leqslant x_i, y_i \leqslant 10^9$) — координаты i-го домика

Гарантируется, что попарные расстояния между всеми домиками различны и не существует двух домиков с равными координатами.

Формат выходных данных

Выведите наименьшее k такое, что между любыми двумя домиками будет существовать путь по дорожкам.

Примеры

Пример №1

Стандартный ввод
2 0 0 1 1

Стандартный вывод
2

Решение

Данная задача решается за $O(n^2 \log n)$ если заранее отсортировать для каждого домика остальные по удалению от него, а затем воспользоваться бинарным поиском по величине k.

Ниже представлено решение на языке C++

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
mt19937 rnd(228);

//Возвращает расстояние между домиками a и b
ll dist(pair <int, int> a, pair <int, int> b) {
    return (a.first - b.first) * (ll) (a.first - b.first) +
           (a.second - b.second) * (ll) (a.second - b.second);
}
const int N = 1000 + 1;
int pr[N];
//Здесь находим "главную" (первая которая появилась) вершину в компоненте связности
int get(int v) {
    if (v == pr[v]) {
        return v;
    } else {
        return pr[v] = get(pr[v]);
    }
}
//Здесь объединяем две вершину в одну компонету
void uni(int u, int v) {
    pr[get(u)] = get(v);
}

int main() {
    int n;
    cin >> n;
    vector <pair <int, int> > e(n);
    for (int i = 0; i < n; i++) {
        cin >> e[i].first >> e[i].second;
    }
    vector <pair <int, pair <int, int> > > edge;
    for (int i = 0; i < n; i++) {
        vector <pair <ll, int> > f;
        //Считаем расстояние от i домика до j домика и сортируем
        for (int j = 0; j < n; j++) {
            f.push_back({dist(e[i], e[j]), j});
        }
        sort(f.begin(), f.end());
        /*Массив edge формируем так:
        {n,{i,j}} - пая по длине дорога от i пойдет к j.*/
        for (int j = 0; j < n; j++) {
            edge.push_back({j, {i, f[j].second}});
        }
    }
    /*Сортируем edge, как раз получим сначала 0, потом 1 дороги, и т.д. Останется только проверить,
    sort(edge.begin(), edge.end());
    //Изначально домики не соединены и каждый в своей компоненте
    for (int i = 0; i < n; i++) {
```

```
53          pr[i] = i;
54      }
55      int ans = 0;
56      //Делаем перебор по k
57      for (auto c : edge) {
58          int u = c.second.first, v = c.second.second;
59          if (get(u) != get(v)) {
60              uni(u, v);
61              ans = c.first;
62          }
63      }
64      cout << ans << '\n';
65  }
```

Задача 9.3.6. Метод ближайшего соседа - 3 (8 баллов)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт

На равнине находится маленькая деревня, состоящая из n домиков. Домики пронумерованы натуральными числами от 1 до n. Домики покрашены в два цвета — в черный и в белый. Известны координаты каждого домика. Гарантируется, что попарные расстояния между всеми домиками различны. Для каждого домика пронумеруем остальные по возрастанию расстояния. Первый сосед — ближайший домик и т.д. Обозначим как $P(m)$ долю домиков, у которых m-й сосед другого цвета.

Найдите домик, при удалении которого $P(1)$ будет наибольшим.

Формат входных данных

На первой строке входного файла расположено целое число n ($3 \leqslant n \leqslant 10^3$) — число домиков.

В следующих n строках расположены три целых числа x_i, y_i, $color_i$ ($0 \leqslant x_i, y_i \leqslant 10^9$), ($0 \leqslant color_i \leqslant 1$) — координаты i-го домика и цвет (0 означает, что домик белый и 1 что домик черный).

Гарантируется, что попарные расстояния между всеми домиками различны и не существует двух домиков с равными координатами.

Формат выходных данных

Выведите номер домика, при удалении которого $P(1)$ будет наибольшим, в случае, если возможных ответов несколько, выведите минимальный.

Примеры

Пример №1

Стандартный ввод
3 0 0 1 1 1 0 1 2 0

Стандартный вывод
1

Решение

Мы можем заранее найти для каждой точки два ближайших соседа. Далее можно перебрать удаляемую точку и пройтись по всем точкам, чтобы узнать, какая точка станет для них ближайшей после удаления. Этого будет достаточно, чтобы решить задачу. Итоговое время работы $O(n^2)$.

Ниже представлено решение на языке C++

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
mt19937 rnd(228);

//Возвращает расстояние между 2 домиками
ll dist(pair <int, int> a, pair <int, int> b) {
    return (a.first - b.first) * (ll) (a.first - b.first) +
           (a.second - b.second) * (ll) (a.second - b.second);
}
const int N = 1000 + 1;
int pr[N];

//Здесь находим "главную" (первая которая появилась) вершину в компоненте связности
int get(int v) {
    if (v == pr[v]) {
        return v;
    } else {
        return pr[v] = get(pr[v]);
    }
}
//Здесь объединяем две вершину в одну компонету
void uni(int u, int v) {
    pr[get(u)] = get(v);
}

int main() {
    int n;
    cin >> n;
    vector <pair <int, int> > e(n);
    vector <int> color(n);
    for (int i = 0; i < n; i++) {
        cin >> e[i].first >> e[i].second >> color[i];
    }
    vector <int> res(n);
    vector <pair <int, pair <int, int> > > edge;

    for (int i = 0; i < n; i++) {
        vector <pair <ll, int> > f;
```

```
41          //Считаем расстояние от i домика до остальных
42          for (int j = 0; j < n; j++) {
43              f.push_back({dist(e[i], e[j]), j});
44          }
45          //Сортируем эти расстояния
46          sort(f.begin(), f.end());
47          //Убираем
48          f.erase(f.begin());
49          //Смотрим теперь на нового соседа
50          if (color[f[0].second] == color[i]) {
51              //Если условие для P выполнено, то увеличиваем res
52              if (color[f[1].second] != color[i]) res[f[0].second]++;
53          } else {
54              //Если условие для P невыполнено, то уменьшаем res
55              if (color[f[1].second] == color[i]) res[f[0].second]--;
56          }
57      }
58      //Вычисляем максимум в res и выводим ответ
59      int x = 0;
60      for (int i = 0; i < n; i++) {
61          if (res[i] > res[x]) {
62              x = i;
63          }
64      }
65      cout << x + 1 << '\n';
66  }
```

Задача 9.3.7. Монотонные классификаторы - 1 (4 балла)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт

Двудольным называется неориентированный граф $\langle V, E\rangle$, вершины которого можно разбить на два множества L и R, так что $L \cap R = \emptyset$, $L \cup R = V$ и для любого ребра $(u, v) \in E$ либо $u \in L, v \in R$, либо $v \in L, u \in R$.

Дан неориентированный граф. Требуется проверить, является ли он двудольным.

Формат входных данных

Первая строка входного файла содержит два натуральных числа n и m — количество вершин и ребер графа соответственно ($1 \leqslant n \leqslant 100\,000$, $0 \leqslant m \leqslant 200\,000$).

Следующие m строк содержат описание ребер по одному на строке. Ребро номер i описывается двумя натуральными числами b_i, e_i — номерами концов ребра ($1 \leqslant b_i, e_i \leqslant n$). Допускаются петли и параллельные ребра.

Формат выходных данных

В единственной строке выходного файла выведите «YES», если граф является двудольным и «NO» в противном случае.

Если граф двудольный, выведите правильное разбиение на два множества: для каждой вершины выведите 0, если она входит в множество L, и 1 иначе.

Примеры

Пример №1

Стандартный ввод
4 4 1 2 1 3 2 4 4 2
Стандартный вывод
NO

Решение

Проверка графа на двудольность – классическая задача. Каждая компонента связности имеет либо 0, либо 2 варианта корректной раскраски, то есть, если мы раскрасим одну вершину, мы сможем узнать цвета всех достижимых из неё. Поэтому можно в каждой компоненте связности покрасить одну вершину в белый цвет, потом попытаться раскрасить остальные непротиворечивым образом. Если это не получится сделать, то граф не двудолен, иначё мы нашли раскраску.

Ниже представлено решение на языке C++

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
mt19937 rnd(228);
const int N = 1e5 + 7;
vector <int> g[N];
int col[N];
//Запускаем dfs (параметры: вершина и цвет, в который мы ее хотим покрасить)
void dfs(int v, int c) {
    //Если раскрашена, то смотрим в нужный ли нам цвет.
    if (col[v] != -1) {
    //Если цвет противоположный, то выдаем NO.
        if (col[v] != c) {
            cout << "NO\n";
            exit(0);
        }
        return;
    }
    //Если была без цвета, то красим в нужный цвет.
    col[v] = c;
    //Запускаем dfs от всех смежных вершин, но с противоположным цветом.
    for (int to : g[v]) {
        dfs(to, c ^ 1);
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n, m;
    cin >> n >> m;
```

```
33      //Считываем список смежности
34      for (int i = 0; i < m; i++) {
35          int a, b;
36          cin >> a >> b;
37          a--, b--;
38          g[a].push_back(b);
39          g[b].push_back(a);
40      }
41      //Изначально все вершины без цвета.
42      for (int i = 0; i < n; i++) {
43          col[i] = -1;
44      }
45      //В цикле пытаемся красить нераскрашенные вершины
46      for (int i = 0; i < n; i++) {
47          if (col[i] == -1) {
48              dfs(i, 0);
49          }
50      }
51      //Если цикл заавершился, то все раскрашены без ошибок
52      cout << "Yes\n";
53      for (int i = 0; i < n; i++) {
54          cout << col[i] << ' ';
55      }
56      cout << '\n';
57  }
```

Задача 9.3.8. Монотонные классификаторы - 2 (7 баллов)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод
Ограничение по времени: 2 секунда
Ограничение по памяти: 256 мегабайт

Двудольным графом называется неориентированный граф (V, E), $E \subseteq V \times V$ такой, что его множество вершин V можно разбить на два множества A и B, для которых $\forall (e_1, e_2) \in E$ $e_1 \in A$, $e_2 \in B$ и $A \cup B = V$, $A \cap B = \varnothing$.

Требуется найти две (возможно, совпадающие) вершины левой доли такие, что в объединении множеств вершин, связанных с ними, содержится наибольшее количество элементов.

Формат входных данных

В первой строке записаны два целых числа n и m ($1 \leqslant n, m \leqslant 250$), где n — число вершин в множестве A, а m — число вершин в B.

Далее следуют n строк с описаниями рёбер — i-я вершина из A описана в $(i+1)$-й строке файла. Каждая из этих строк содержит номера вершин из B, соединённых с i-й вершиной A. Гарантируется, что в графе нет кратных ребер. Вершины в A и B нумеруются независимо (с единицы). Список завершается числом 0.

Формат выходных данных

Выведите максимальное число вершин в объединении множеств вершин, связанных с какими-то двумя (возможно, совпадающими) вершинами левой доли.

Примеры

Пример №1

Стандартный ввод
2 2 1 2 0 2 0

Стандартный вывод
2

Решение

Переберём вершину второй доли и пары вершин из первой доли, которые обе соединены с ней. Так мы посчитаем для каждой пары вершин первой доли, сколько у них общих соседей. Это в итоге решит задачу за $O(V^3)$.

Ниже представлено решение на языке C++

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

mt19937 rnd(228);

const int N = 1e5 + 7;

vector <int> gr[N];
//в cnt будем хранить, сколько вершин из B связано с этой парой из A
int cnt[300][300];

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n, m;
    cin >> n >> m;
    //Считываем список смежности
    for (int i = 0; i < n; i++) {
        int x;
        while (cin >> x) {
            if (x == 0) {
                break;
            }
            gr[x - 1].push_back(i);
        }
    }
    //Делаем полный перебор по m вершинам второй доли
    for (int i = 0; i < m; i++) {
        //Смотрим вершины с которыми связана i вершина 2 доли, для каждой пары +1
        for (int a : gr[i]) {
            for (int b : gr[i]) {
                cnt[a][b]++;
            }
        }
    }
```

```
39      int ans = 0;
40      //Находим максимум в массиве cnt
41      for (int i = 0; i < n; i++) {
42          for (int j = 0; j < n; j++) {
43              ans = max(ans, cnt[i][j]);
44          }
45      }
46      cout << ans << '\n';
47  }
```

Задача 9.3.9. Монотонные классификаторы - 3 (12 баллов)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод
Ограничение по времени: 2 секунда
Ограничение по памяти: 256 мегабайт

Двудольным графом называется неориентированный граф (V, E), $E \subseteq V \times V$ такой, что его множество вершин V можно разбить на два множества A и B, для которых $\forall (e_1, e_2) \in E$ $e_1 \in A$, $e_2 \in B$ и $A \cup B = V$, $A \cap B = \varnothing$.

Паросочетанием в двудольном графе называется любой набор его несмежных рёбер, то есть такой набор $S \subseteq E$, что для любых двух рёбер $e_1 = (u_1, v_1)$, $e_2 = (u_2, v_2)$ из S $u_1 \neq u_2$ и $v_1 \neq v_2$.

Дан двудольный граф, с n вершинами в множестве A и m вершинами в множестве B.

Требуется посчитать количество паросочетаний, состоящих из n ребер. Так как ответ может быть очень большим, посчитайте остаток от деления ответа на 998244353.

Формат входных данных

В первой строке записаны два целых числа n и m ($1 \leqslant n, m \leqslant 18$), где n — число вершин в множестве A, а m — число вершин в B.

Далее следуют n строк с описаниями рёбер — i-я вершина из A описана в $(i+1)$-й строке файла. Каждая из этих строк содержит номера вершин из B, соединённых с i-й вершиной A. Гарантируется, что в графе нет кратных ребер. Вершины в A и B нумеруются независимо (с единицы). Список завершается числом 0.

Формат выходных данных

Выведите остаток от деления числа паросочетаний из n ребер на 998244353.

Примеры

Пример №1

Стандартный ввод
2 2 1 2 0 2 0

Стандартный вывод
60

Решение

Данная задача решается динамическим программированием по подмаскам. Мы поддерживаем маску вершин второй доли, которым кого-то уже назначили и пробуем новой вершине первой доли назначить свободную соседнюю вершину второй доли. Это можно сделать за $O(n2^n)$.

Ниже представлено решение на языке C++

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
mt19937 rnd(228);
const int N = 19;
const int M = 998244353;

//Сложение по модулю M
inline int add(int a, int b) {
    a += b;
    if (a >= M) a -= M;
    return a;
}

//Возвращает произведение a и b по модулю M
inline int mul(int a, int b) {
    return (a * (ll) b) % M;
}

vector <int> g[N];
int dp[N][(1 << N)];

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n, m;
    cin >> n >> m;
    //Создаем список смежности
    for (int i = 0; i < n; i++) {
        int x;
        while (cin >> x) {
            if (x == 0) {
                break;
            }
            x--;
            g[i].push_back(x);
        }
    }
    int lim = (1 << m);
    dp[0][0] = 1;
    for (int i = 0; i < n; i++) {
        for (int mask = 0; mask < lim; mask++) {
            for (int to : g[i]) {
                if (!((mask >> to) & 1)) {
                    dp[i + 1][mask | (1 << to)] = add(dp[i + 1][mask | (1 << to)],
                                                      dp[i][mask]);
                }
            }
        }
```

```
52      }
53      int s = 0;
54      for (int mask = 0; mask < lim; mask++) {
55          s = add(s, dp[n][mask]);
56      }
57      cout << s << '\n';
58  }
```

Задача 9.3.10. Классификация и кластеризация - 1 (12 баллов)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт

Рассмотрим несколько вариантов того, что можно считать проведением случайной хорды в круге:

1. Метод случайных концов – выбираем две точки на окружности и проводим через них хорду.

2. Метод случайного радиуса – выбираем радиус, выбираем на нём точку и строим хорду, перпендикулярную радиусу, проходящую через выбранную точку.

3. Метод случайного центра – выбераем случайную точку и строим хорду с центром в этой точке.

Широко известный парадокс Бертрана заключается в том, что эти три способа приводят к разным распределениям хорд. Задан набор из достаточно большого числа хорд, полученных одним и тем же способом. Нужно узнать, какой из способов использовался.

Радиус круга равен единице.

Формат входных данных

В первой строке входного файла задано целое число N – количество хорд.

В каждой из последующих N строк заданы координаты концов хорды – вещественные числа x_1, y_1, x_2, y_2 с 5 знаками после десятичной точки. Гарантируется, что $|x_i^2 + y_i^2 - 1| < 10^{-5}$ для $i = 1, 2$, то есть что концы хорды (x_1, y_1) и (x_2, y_2) лежат на единичной окружности.

Во всех тестах, кроме теста из условия, N равно 2000. В тесте из условия любой ответ, подходящий под формат вывода, будет считаться правильным. Число тестов не из условия равно 40.

Формат выходных данных

Выведите одно число: 1, 2 или 3, соответствующее способу, с помощью которого был получен этот набор хорд.

Примеры

Пример №1

Стандартный ввод
2 0.17910 -0.98383 -0.95483 0.29715 0.94569 -0.32506 -0.63183 -0.77511
Стандартный вывод
3

Решение

Нужно посмотреть на долю хорд, которые длиннее, чем сторона правильного треугольника, вписанного в круг. В первом случае эта величина близка к $1/3$, во втором – к $1/2$, в третьем – к $1/4$.

Ниже представлено решение на языке C++

```
#include <bits/stdc++.h>

using namespace std;
using ld = double;
/*Создаем структуру и подгружаем операторы для нее: длина до
центра, расположение на круге, оператор разности (вектор от другой
точки до нашей/ через него можно считать расстояние между 2 точками).*/
struct point {
    ld x, y;
    point(ld x = 0, ld y = 0): x(x), y(y) {}
    point operator-(const point& p) const { return point(x - p.x, y - p.y); }
    ld length() const { return hypot(x, y); }
    bool onCircle() {
        return abs(length() - 1) < 1e-5;
    }
};
//Структура хорды и оператор длин для нее.
struct chord {
    point a, b;
    ld length() const {
        return (a - b).length();
    }
};
void solve(int numChords) {
    vector<chord> chords(numChords);
    //Считываем хорды, проверяем что точки лежат на окружности
    for (int i = 0; i < numChords; ++i) {
        cin >> chords[i].a.x >> chords[i].a.y >> chords[i].b.x >> chords[i].b.y;
        assert(chords[i].a.onCircle());
        assert(chords[i].b.onCircle());
    }
    //Длина стороны правильного треугольника
    ld equilateralTriangleSide = sqrt(3);
    int numLong = 0;
    //Для каждой хорды делаем сравнение ее длины и константы стороны треугольника
    for (int i = 0; i < numChords; ++i) {
        if (chords[i].length() > equilateralTriangleSide)
            ++numLong;
    }
    ld longChordsRatio = numLong / (ld)numChords;
    //Определяем к чему ближе лежит доля хорд больших стороны треугольника
    if (longChordsRatio < (1 / 4.0 + 1 / 3.0) * 0.5)
        cout << "3\n";
```

```
    else if (longChordsRatio < (1 / 3.0 + 1 / 2.0) * 0.5)
        cout << "1\n";
    else
        cout << "2\n";
}
int main() {
    int numChords;
    cin >> numChords;
    solve(numChords);
    return 0;
}
```

Задача 9.3.11. Классификация и кластеризация - 2 (15 баллов)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт

Дано чёрно-белое изображение, которое содержит одну из трёх фигур: круг, квадрат или треугольник. Определите, какая фигура изображена.

Стороны квадрата не обязательно параллельны осям координат!

Пиксель изображения (x, y) является чёрным, если соответствующая точка с целочисленными координатами принадлежит фигуре. Во всех тестах, кроме теста из условия, размер изображения равен 100, при этом радиус круга не меньше 10, ширина стороны квадрата не меньше 20, площадь треугольника составляет хотя бы 30% изображения, граница изображения является белой (то есть фигуры полностью содержатся на рисунке). Человек способен легко определить тип фигуры при таких ограничениях.

Формат входных данных

В первой строке входного файла задано число N – размер изображения. Далее следуют N строк, в каждой из которых находятся N символов, обозначающих цвета изображения. Белый пиксель обозначается символом '.' (точка, ASCII код 46), чёрный – символом '#' (решётка, ASCII код 35).

Во всех тестах, кроме теста из условия, N равно 100. В тесте из условия любой ответ, подходящий под формат вывода, будет считаться правильным.

Формат выходных данных

Выведите 1, если на рисунке изображён круг, 2, если квадрат, и 3, если треугольник.

Примеры

Пример №1

Стандартный ввод
15 ....#........... ...######...... ...###########. ...#########... ..########..... ..######....... ..####......... ..##........... .#............. ....

Стандартный вывод
3

Решение

Данная задача может быть решена аккуратным разбором случаев. Кроме этого есть элегантное решение, заключающееся в том, чтобы рассмотреть отношение центрального момента инерции к площади, для квадрата и круга это характерные величины, равные 6 и 2π соответственно. Величина для треугольника будет где-то между этими двумя значениями. Центральный момент инерции в данном случае это сумма по всем закрашенным точкам $\sum_i[(x_i-x_c)^2+(y_i-y_c)^2]$, где (x_c, y_c) – координаты центра масс закрашенных точек.

Ниже представлено решение на языке C++

```
#include <bits/stdc++.h>

using namespace std;
using ld = double;
const ld pi = acos(-1);

int solution(vector<string> input) {
    int n = input.size();
    int area = 0;
    //координаты центра масс.
    ld cm_x = 0, cm_y = 0;
    //Считаем суммарные координаты всех # и их количество.
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (input[i][j] == '#') {
                cm_x += i;
                cm_y += j;
                ++area;
            }
    //Находим координаты центра масс.
    cm_x /= area;
    cm_y /= area;
    ld I = 0;
```

```
    //Считаем момент по формуле из решения.
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (input[i][j] == '#') {
                I += (i - cm_x) * (i - cm_x) + (j - cm_y) * (j - cm_y);
            }

    ld eps = 0.2;
    ld ratio = area * area / I;

    //В зависимости от I даем ответ.
    if (abs(ratio - 6) < eps)
        return 2;
    else if (abs(ratio - 2 * pi) < eps)
        return 1;
    else
        return 3;
}

int main() {
    int n;
    cin >> n;
    vector<string> input(n);
    for (int i = 0; i < n; ++i)
        cin >> input[i];
    cout << solution(input) << "\n";
    return 0;
}
```

Задача 9.3.12. Классификация и кластеризация - 3 (18 баллов)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт Назовём *фигурой* множество точек в трёхмерном пространстве.

Будем считать две фигуры *эквивалентными*, если одну из них можно получить из другой, применив произвольную композицию поворотов, параллельных переносов и отражений относительно плоскости. Иными словами, фигуры эквивалентны, если существует движение, переводящее одну фигуру в другую.

Дан набор фигур, число точек во всех фигурах одинаковое. Необходимо разбить их на классы эквивалентости, то есть на непересекающиеся множества фигур такие, что фигуры в одном множестве эквивалентны, а в разных – нет.

Дополнительно, гарантируется, что каждая фигура представляет собой что-то из нижеперечисленного списка (размер фигуры и её сдвиг может быть произвольным, при условии, что вершины удовлетворяют ограничениям на входные данные – см. ниже):

1. Правильный многогранник (куб, тетраэдр, октаэдр, додекаэдр).
2. Случайное множество точек на сфере.
3. Случайное центрально-симметричное множество точек на сфере (вместе с точкой в множестве содержится диаметрально противоположная).

4. Случайное множество точек, лежащее в одной плоскости.
5. Случайное множество точек, лежащее в одной плоскости и на одной окружности.
6. Случайное множество точек, расстояние от которых до некоторого центра лежит в интервале от r до R, $R > r$ – "шаровой слой", разность большого и маленького шаров.

Формат входных данных

В первой строке входного файла заданы число фигур N ($2 \leqslant N \leqslant 200$) и число точек в каждой фигуре M ($2 \leqslant M \leqslant 16$). Следующие NM строк описывают фигуры: первые M строк описывают первую фигуру, следующие M – вторую, и так далее. В каждой строке записаны три вещественных числа – координаты вершин фигуры с 5 знаками после десятичной точки.

Координаты фигур не превышают по модулю 10^3. Расстояние между любыми двумя вершинами фигуры составляет хотя бы 10^{-3}. Диаметр любой фигуры (максимальное расстояние между точками) не превышает 20.

Формат выходных данных

В первой строке выведите число X классов эквивалентности. В последующих X строках выведите классы эквивалентности, по одному в строке: сначала размер класса, затем индексы фигур, принадлежащих этому классу, в произвольном порядке. Фигуры номеруются начиная с 1 в том порядке, в котором они заданы во входном файле.

Примеры

Пример №1

Стандартный ввод
4 2 5.00000 0.00000 0.00000 4.00000 0.00000 0.00000 3.00000 3.00000 3.00000 2.00000 3.00000 3.00000 2.00000 0.00000 2.00000 1.00000 1.00000 2.00000 0.00000 3.00000 2.00000 1.00000 2.00000 2.00000

Стандартный вывод
2 2 1 2 2 3 4

Решение

Как и в прошлой задаче, здесь от участников ожидалось, что они придумают какие-нибудь быстро вычислимые статистики, которые будут равны независимо от

параллельного переноса и вращений. В качестве такой статистики можно было взять какую-нибудь квадратичную форму от координат точек и рассмотреть её собственные значения. Однако, хватало и более простого подхода, например, посмотреть на вектор расстояний от каждой точки до центра масс.

Ниже представлено решение на языке C++

```
#include <bits/stdc++.h>

using namespace std;

using ld = double;

#define all(v) (v).begin(), (v).end()

struct point {
    ld x, y, z;

    point(ld x = 0, ld y = 0, ld z = 0): x(x), y(y), z(z) {}

    point operator-(const point& p) const {return point(x - p.x, y - p.y, z - p.z); }

    ld len() const { return sqrt(x * x + y * y + z * z); }
};

struct figure {
    vector<point> points;
};

//Возвращает вектор всех попарных расстояний
vector<ld> allDistances(const vector<point>& v) {
    vector<ld> res;
    for (int i = 0; i < v.size(); ++i)
        for (int j = i + 1; j < v.size(); ++j)
            res.push_back((v[i] - v[j]).len());
    sort(all(res));
    assert(res.back() <= 20 && res.front() > 1e-3);
    return res;
}

/*Возвращает true, если сумму модулей разностей i-ых координат
векторов a и b равна 0 в пределах погрешности. И false - иначе.*/
bool sortedVectorsClose(const vector<ld>& a, const vector<ld>& b) {
    assert(a.size() == b.size());

    ld error = 0;
    for (int i = 0; i < a.size(); ++i)
        error += abs(a[i] - b[i]);

    return error < 1e-3;
}

//Сравнивает фигуры на основе решения sortedVectorsClose
bool sameFigure(const figure& a, const figure& b) {
    return sortedVectorsClose(allDistances(a.points), allDistances(b.points));
}

vector<vector<int>> solve(const vector<figure>& figures) {
    int nFigures = figures.size();

```

```
54      vector<char> used(nFigures);
55      vector<vector<int>> clusters;
56      /*Попарно сравниваем фигуры с помощью sameFigure, если одинаковые
57      - помещаем в один кластер. Также в ходе проверки можем наткнуться на фигуру,
58      которую уже поместили в какой-то кластер, для этого массив used.
59      Ответ - кол-во кластеров.*/
60      for (int i = 0; i < nFigures; ++i) {
61          if (used[i]) continue;
62
63          vector<int> cluster;
64          for (int j = i; j < nFigures; ++j)
65              if (sameFigure(figures[i], figures[j])) {
66                  cluster.push_back(j);
67                  used[j] = true;
68              }
69
70          assert(!cluster.empty() && cluster[0] == i);
71          clusters.push_back(cluster);
72      }
73      sort(all(clusters));
74
75      return clusters;
76  }
77
78  int main() {
79      int nFigures, nPoints;
80      cin >> nFigures >> nPoints;
81
82      //Считываем фигуры
83      vector<figure> figures(nFigures);
84      for (int i = 0; i < nFigures; ++i) {
85          figures[i].points.resize(nPoints);
86          for (int j = 0; j < nPoints; ++j) {
87              cin >> figures[i].points[j].x >>
88                     figures[i].points[j].y >>
89                     figures[i].points[j].z;
90              auto& pt = figures[i].points[j];
91              assert(abs(pt.x) < 1e3);
92              assert(abs(pt.y) < 1e3);
93              assert(abs(pt.z) < 1e3);
94          }
95      }
96      //Запускаем сравнение и выводим ответ
97      auto answer = solve(figures);
98      cout << answer.size() << "\n";
99      for (auto& it: answer) {
100         cout << it.size() << " ";
101         for (int index: it)
102             cout << index + 1 << " ";
103         cout << endl;
104     }
105
106     return 0;
107 }
```

Задача 9.3.13. Различные методы - 1 (8 баллов)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод

Ограничение по времени: 2 секунда
Ограничение по памяти: 256 мегабайт

Дано N точек на плоскости, каждая из которых принадлежит одному из двух классов: 1 или -1. Также дана точка P с координатами (P_x, P_y). Постройте линейный классификатор, то есть функцию вида,

$$f(x,y) = \begin{cases} 1, & Ax + By + C > 0, \\ 0, & Ax + By + C = 0, \\ -1 & Ax + By + C < 0 \end{cases}$$

разделяющая прямая которого проходит через точку P ($f(P_x, P_y) = 0$), который достигает максимальной точности предсказания, то есть, число точек (x, y), класс которых совпадает с $f(x, y)$, максимально возможное.

Координаты всех заданных точек целые. Коэффициенты A, B, C должны быть **32-битными целыми числами**, в частности, должны принадлежать отрезку $[-2^{31}, 2^{31}-1]$. Гарантируется, что при данных ограничениях на координаты, такой классификатор существует.

Формат входных данных

В первой строке находится число N объектов ($1 \leqslant N \leqslant 10^5$).

Во второй строке через пробел находятся два целых числа P_x и P_y ($-10^4 \leqslant P_x, P_y \leqslant 10^4$) – координаты точки P.

В N последующих строках находятся тройки целых чисел x, y, c ($-10^4 \leqslant x, y \leqslant 10^4, c \in \{-1, 1\}$), задающих точку (x, y) класса c.

Все точки (в том числе точка P), заданные во входном файле, различны.

Формат выходных данных

Выведите через пробел 3 **целых** числа – коэффициенты A, B, C ($-2^{31} \leqslant A, B, C \leqslant 2^{31} - 1$) построенного линейного классификатора максимальной точности, проходящего через точку P. Если подходящих классификаторов несколько, выведите любой.

Примеры

Пример №1

Стандартный ввод
4 0 1 0 2 1 2 0 1 1 -1 -1 -2 2 -1

Стандартный вывод
3 3 -3

Решение

Отсортируем все точки вокруг A по полярному углу. Затем пройдя по ним двумя указателями, можно в любой момент знать, сколько белых точек попадут в первую половину и сколько чёрных в другую. Сложность такого решения будет $O(n \log n)$.

Ниже представлено решение на языке C++

```
#include <bits/stdc++.h>

using namespace std;

using li = long long;
#define int li

#define all(v) (v).begin(), (v).end()

struct point {
    int x, y;
    int sign;

    point(int x = 0, int y = 0, int s = 0): x(x), y(y), sign(s) {}

    point operator+(const point& p) const { return point(x + p.x, y + p.y, sign); }
    point operator-(const point& p) const { return point(x - p.x, y - p.y, sign); }

    int vprod(const point& p) const { return x * p.y - y * p.x; }

    int half() const {
        if (y)
            return y < 0;
        else
            return x < 0;
    }

    bool operator<(const point& p) const {
        if (half() != p.half())
            return half() < p.half();
        else
            return vprod(p) > 0;
    }
};

signed main() {
    int n;
    cin >> n;

    point A;
    cin >> A.x >> A.y;

    vector<point> p(n);
    for (int i = 0; i < n; ++i) {
        cin >> p[i].x >> p[i].y >> p[i].sign;
        p[i] = p[i] - A;
    }

    p.emplace_back(1, 0, 0);
    p.emplace_back(-1, 0, 0);
    p.emplace_back(0, 1, 0);
    p.emplace_back(0, -1, 0);
```

```
53
54      sort(all(p));
55
56      vector<point> new_p;
57
58      for (int i = 0; i < p.size();) {
59          int r = i, sum = 0;
60          while (r < p.size() && p[i].vprod(p[r]) == 0) {
61              sum += p[r].sign;
62              ++r;
63          }
64
65          new_p.push_back(point(p[i].x, p[i].y, sum));
66          i = r;
67      }
68
69      p = new_p;
70      n = p.size();
71
72      int r = 0;
73      int sum = 0;
74
75      int maxSum = -1e9;
76      point best = 0;
77
78      for (int l = 0; l < n; ++l) {
79          while (p[l].vprod(p[r]) >= 0) {
80              sum += p[r].sign;
81              r = (r + 1) % n;
82          }
83
84          sum -= p[l].sign;
85
86          if (sum > maxSum) {
87              maxSum = sum;
88              best = p[l] + p[(l + 1) % n];
89          }
90
91          sum -= p[l].sign;
92      }
93
94      cout << -best.y << " " << best.x << " " << -best.vprod(A) << "\n";
95
96      return 0;
97  }
```

Задача 9.3.14. Различные методы - 2 (10 баллов)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт

Дизъюнктивная нормальная форма (ДНФ) булевой функции – это её представление в виде дизъюнкции (логического ИЛИ) нескольких конъюнктов. Конъюнктом называется конъюнкция (логическое И) нескольких переменных или их отрицаний. Например,

$$f(x, y, z, t) = (x \wedge y) \vee \neg z$$

– это ДНФ, состоящая из конъюнктов $x \wedge y$ и $\neg z$. Заметим, что f не зависит от одного из своих аргументов, t.

Дана ДНФ, зависящая от N булевых переменных, в которой каждая переменная встречается не более одного раза (возможно, в виде отрицания). На скольких наборах значений аргументов (из 2^N возможных) она принимает истинное значение?

Формат входных данных

В первой строке содержится число аргументов N ($1 \leqslant N \leqslant 62$) и число конъюнктов K ($1 \leqslant K \leqslant N$).

В последующих K строках описаны конъюнкты, по одному в строке. Первое число в строке – число переменных или их отрицаний k_i ($k_i \geqslant 1$) в конъюнкте. Последующие k_i чисел задают переменные или их отрицания следующим образом: положительное число $x \geqslant 1$ означает переменную с номером x, отрицательное число $-x \leqslant -1$ означает отрицание переменной с номером x.

Гарантируется, что каждая переменная встречается в ДНФ максимум один раз.

Формат выходных данных

Выведите единственное число – ответ на задачу.

Примеры

Пример №1

Стандартный ввод
4 2 2 1 2 1 -3

Стандартный вывод
10

Решение

ДНФ истинна если и только если истинен хотя бы один конъюнкт. Посчитаем, сколько есть наборов переменных, на которых данная ДНФ ложна и вычтем из 2^N. В условии сказано, что каждая переменная встречается не больше одного раза, поэтому у нас есть $K + 1$ групп переменных – своя группа у каждого конъюнкта и группа переменных, которые нигде не встречаются. Обозначим размер последней группы как k_0. Чтобы ДНФ была ложной, мы можем выбрать элементы первой группы произвольным образом, а во всех остальных группах будет ровно один набор переменных, который не занулит ДНФ.

Учитывая, что наборы можно рассматривать независимо, получается, что ответом будет $2^N - 2^{k_0} \prod_{i=1}^{K} \left(2^{k_i} - 1\right)$.

Ниже представлено решение на языке C++

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    int nVars, nTerms;
    cin >> nVars >> nTerms;

    unsigned long long nFalse = 1;

    int meet = 0;

    for (int i = 0; i < nTerms; ++i) {
        int x;
        cin >> x;

        vector<int> a(x);
        for (int i = 0; i < x; ++i)
            cin >> a[i];

        //nFalse = nFalse * 2^{x} - 1.
        //кол-во встреченных переменных увеличиваем на x.
        nFalse *= (1ULL << x) - 1;
        meet += x;
    }
    //После работы цикла в nFalse окажется нужное произведение, только без 2^{k_0}.

    //nFalse = nFalse * 2^{k_0} (k_0 = nVars - meet)
    nFalse *= (1ULL << (nVars - meet));

    //nTrue = 2^N- nFalse;
    unsigned long long nTrue = (1ULL << nVars) - nFalse;
    cout << nTrue << endl;

    return 0;
}
```

Задача 9.3.15. Различные методы - 3 (18 баллов)

Имя входного файла: стандартный ввод
Имя выходного файла: стандартный вывод
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт

Дизъюнктивная нормальная форма (ДНФ) булевой функции – это её представление в виде дизъюнкции (логического ИЛИ) нескольких конъюнктов. Конъюнктом называется конъюнкция (логическое И) нескольких переменных или их отрицаний. Например,

$$f(x, y, z, t) = (x \wedge y) \vee \neg z$$

– это ДНФ, состоящая из конъюнктов $x \wedge y$ и $\neg z$. Заметим, что f не зависит от одного из своих аргументов, t.

Вам загадали булеву функцию от трёх переменных (x, y и z), а затем для каждого из $2^3 = 8$ возможных наборов аргументов посчитали её значение. Вам нужно построить минимальную по длине ДНФ, задающую эту функцию.

Формат входных данных

В первой строке дано число t — количество функций, для которых надо посчитать ответ ($1 \leqslant t \leqslant 256$).

В следующих t строках дано описание функций. В i-й строке записана строка длины 8, состоящая из символов 0 или 1 — таблица истинности i-й функции.

Символ на позиции j означает, что должна возвращать функция, если ей на вход подать $x = \lfloor j/4 \rfloor, y = \lfloor j/2 \rfloor \bmod 2, z = j \bmod 2$.

Формат выходных данных

Выведите t строк, в i-й должна содержаться ДНФ минимальной длины, равная i-й функции. Если таких выражений несколько, выведите любое.

Примеры

Пример №1

Стандартный ввод
4 00110011 00000111 11110000 00011111

Стандартный вывод
y x\&y|x\&z !x x|y\&z

Решение

Нужно составить контекстно-свободную грамматику, задающую ДНФ. По ней можно генерировать формулы и поддерживать значения, которые они принимают. Если достаточно соптимизировать перебор, окажется, что состояний нужно перебрать не очень много.

Ниже представлено решение на языке C++

```
#include <bits/stdc++.h>

using namespace std;

vector<pair<string, int>> gen_f(int n);

```

```
vector<pair<string, int>> ans_t[40], ans_e[40], ans_f[40];

vector<pair<string, int>> sparse(vector<pair<string, int>> g) {
    string gg[256];
    int used[256] = {0};
    vector<pair<string, int>> res;
    for(auto it: g) {
        if(!used[it.second]) {
            used[it.second] = true;
            gg[it.second] = it.first;
        } else {
            gg[it.second] = min(gg[it.second], it.first);
        }
    }
    for(int i = 0; i < 256; i++) {
        if(gg[i].size()) {
            res.push_back({gg[i], i});
        }
    }
    return res;
}

vector<pair<string, int>> gen_t(int n) {
    if(ans_t[n].size()) {
        return ans_t[n];
    }
    auto ret = gen_f(n);
    if(n > 2) {
        for(int i = 1; n - i - 1 > 0; i++) {
            auto x = gen_t(i);
            auto y = gen_f(n - i - 1);
            for(auto it: x) {
                for(auto jt: y) {
                    ret.push_back({it.first + "&" + jt.first,
                                   it.second & jt.second});
                }
            }
        }
    }
    return ans_t[n] = sparse(ret);
}

vector<pair<string, int>> gen_e(int n) {
    if(ans_e[n].size()) {
        return ans_e[n];
    }
    auto ret = gen_t(n);
    if(n > 2) {
        for(int i = 1; n - i - 1 > 0; i++) {
            auto x = gen_e(i);
            auto y = gen_t(n - i - 1);
            for(auto it: x) {
                for(auto jt: y) {
                    ret.push_back({it.first + "|" + jt.first,
                                   it.second | jt.second});
                }
            }
        }
    }
    return ans_e[n] = sparse(ret);
```

```
}

string get_tab(string f);

vector<pair<string, int>> gen_f(int n) {
    if(ans_f[n].size()) {
        return ans_f[n];
    }
    if(n == 1) {
        return {{"x", bitset<8>(get_tab("x")).to_ulong()},
                {"y", bitset<8>(get_tab("y")).to_ulong()},
                {"z", bitset<8>(get_tab("z")).to_ulong()}};
    }
    vector<pair<string, int>> ret;
    { // ! A
        auto gg = gen_f(n - 1);
        for(auto it: gg) {
            ret.push_back({"!" + it.first, ((1 << 8) - 1) ^ it.second});
        }
    }
    return ans_f[n] = sparse(ret);
}

int parse_f(string &f, int x, int y, int z);

int parse_t(string &f, int x, int y, int z) {
    int s = parse_f(f, x, y, z);
    if(!f.empty() && f.back() == '&') {
        f.pop_back();
        s &= parse_t(f, x, y, z);
    } else if(!f.empty() && f.back() != '|') {
        // Parse error
    }
    return s;
}

int parse_e(string &f, int x, int y, int z) {
    int s = parse_t(f, x, y, z);
    if(!f.empty() && f.back() == '|') {
        f.pop_back();
        s |= parse_e(f, x, y, z);
    } else if(!f.empty()) {
        // Parse error
    }
    return s;
}

int parse_f(string &f, int x, int y, int z) {
    if(f.back() == '!') {
        f.pop_back();
        return !parse_f(f, x, y, z);
    } else if(f.back() == 'x') {
        f.pop_back();
        return x;
    } else if(f.back() == 'y') {
        f.pop_back();
        return y;
    } else if(f.back() == 'z') {
        f.pop_back();
        return z;
```

```
    } else {
        // Parse error
    }
}

string get_tab(string f) {
    reverse(begin(f), end(f));
    string res;
    string t;
    t = f; res += char('0' + parse_e(t, 0, 0, 0));
    t = f; res += char('0' + parse_e(t, 0, 0, 1));
    t = f; res += char('0' + parse_e(t, 0, 1, 0));
    t = f; res += char('0' + parse_e(t, 0, 1, 1));
    t = f; res += char('0' + parse_e(t, 1, 0, 0));
    t = f; res += char('0' + parse_e(t, 1, 0, 1));
    t = f; res += char('0' + parse_e(t, 1, 1, 0));
    t = f; res += char('0' + parse_e(t, 1, 1, 1));
    return res;
}


signed main() {
    //freopen("input.txt", "r", stdin);
    ios::sync_with_stdio(0);
    cin.tie(0);
    int have = 0, need = 256;
    string ans[256];
    int cur = 1;
    while(have < need) {
        auto x = gen_e(cur);
        for(auto it: x) {
            if(ans[it.second].empty()) {
                ans[it.second] = it.first;
                have++;
            }
        }
        cur++;
    }
    for(int i = 0; i < 256; i++) {
        //cout << "\"" << ans[i] << "\"," << endl;
        if(ans[i].size() > 0) {
            assert(bitset<8>(get_tab(ans[i])).to_ulong() == i);
        }
    }

    int n;
    cin >> n;
    while(n--) {
        string t;
        cin >> t;
        cout << ans[bitset<8>(t).to_ulong()] << endl;
    }
    return 0;
}
```