Задача A. Выборы в РИЛИ

Автор задачи: Глазков Илья
Разработал: Даминов Камиль

Давайте поймём, что значит набрать 50% голосов. Это означает, что первому кандидату нужно набрать голосов не меньше, чем двум других кандидатов в сумме. Когда только за первого кандидата голосуют дополнительно другие люди, то сумма голосов второго и третьего никак не меняется, значит нужно найти минимальное целое неотрицательное x, что $a + x \geqslant b + c$. Если $a \geqslant b + c$, то первый кандидат уже набрал хотя 50%, поэтому ответ будет равен нулю. Если же $a < b + c$, тогда понятно, что первому нужно набрать хотя бы $x = b+c-a$, чтобы сравняться с двумя другими. Если же за первого дополнительно проголосует меньше, чем $b+c-a$ людей, то тогда $a+x < a+(b+c-a)$, то есть $a + x < b + c$. Поэтому ответом на задачу в этом случае будет число $b + c - a$.

В итоге получаем такое решение на 100 баллов:

```
a = int(input())
b = int(input())
c = int(input())

if a >= b + c:
    print(0)
else:
    print(b + c - a)
```

Задача B. Магический квадрат

Разработчик и автор задачи: Нигматуллин Вадим Ринатович

Сумма в магическом квадрате равна 45. Тогда получается, что на каждой строке, столбце и диагонали сумма цифр должна быть равна 15. Вычислим неизвестные числа.

Посчитаем число в центре, оно равно 15 - (сумма двух чисел из второй строки / столбца). Вообще, если сделать небольшое исследование, то на самом деле в центре может быть единственное число - это 5.

Далее найдем первое число с первой строк: 15 - (сумма двух чисел из первого столбца).

Найдем число из последней строки: 15 - (сумма двух чисел последней строки).

Остается последнее число, второе неизвестное число из первой строки. Его также найдем, посчитав разницу между 15 и суммой известных чисел.

Задача C. Необычная шахматная доска

Автор задачи: Нигматуллин Вадим Ринатович
Разработал: Даминов Камиль

Дана доска размера $n \times m$, которая особым образом красится в три цвета — белый, черный и синий. Необходимо было найти, какого цвета больше всего.

Подзадача 1 (22 балла)

В этой подзадаче $n, m \leqslant 3$, поэтому можно нарисовать все возможные доски и посчитать количество каждого цвета. Можно заметить, что если $n = 2$ и $m = 2$, то черного цвета будет больше всех остальных, поэтому в этом случае ответом будет слово `«Black»`. В остальных случаях ответом будет `«White»`.

В итоге получаем такое решение:

```
n = int(input())
m = int(input())
```

```
if n == 2 and m == 2:
    print("Black")
else:
    print("White")
```

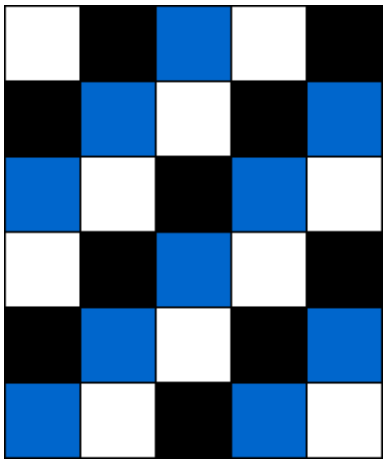
За такое решение вы могли получить 22 балла.

Подзадача 2 (48 баллов)

В этой подзадаче $n, m \leqslant 1000$, поэтому можно за $O(n^2)$ построить доску и покрасить её. После этого посчитать ответ будет не сложно. За такое решение вы могли получить 48 баллов.

Подзадача 3 (14 баллов)

В этой подзадаче n кратно трем, либо m кратно трем. Давайте будем считать, что именно n кратно трем (если m будет кратно трем, то решение будет точно таким же). Пусть, например, $n = 6$ и $m = 5$, тогда доска будет выглядеть так:

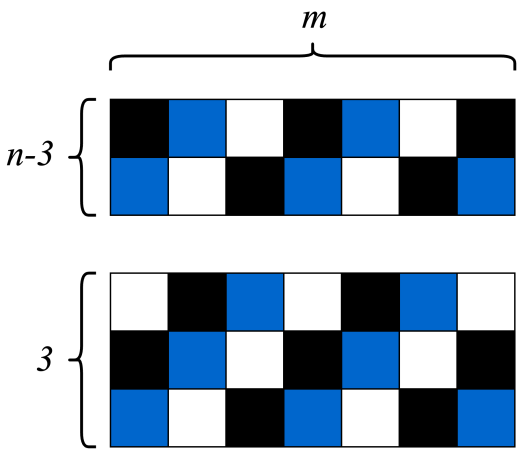


Так как n кратно трем, то в каждом столбце количество клеток кратно трем. Заметим, что в каждом столбце, количество клеток каждого цвета равно $n \div 3$. Это также легко понять, так как в столбце цвета повторяются каждые три клетки.

Так как в каждом столбце количество клеток каждого цвета одинаково, то и на всей доске будет также. Поэтому ответом в этом случае может быть любой цвет. За такое решение вы могли получить 14 баллов.

Полное решение

Из прошлой подзадачи мы узнали, что, если n делится на 3 или m делится на 3, то количество клеток каждого цвета будет одинаково. Давайте возьмем нашу доску $n \times m$. Если $n > 3$, то давайте отломим от этой доски кусок размера $3 \times m$. Тогда мы от доски $n \times m$ перейдем к доске $(n-3) \times m$. Так как мы отломили кусок $3 \times m$, то в этом куске количества клеток каждого цвета равны между собой, поэтому, если на доске $n \times m$ какого-то цвета было больше всех, то и на доске $(n-3) \times m$ этого цвета будет больше всех. Приведем пример, когда мы от доски 5×7 отломили кусок 3×7:



Этот факт говорит, что у доски $n \times m$ и доски $(n-3) \times m$ ответы будут равны. Аналогично можно показать, что у доски $n \times m$ и доски $n \times (m-3)$ ответы будут одинаковыми. Давайте поэтому будем уменьшать n до тех пор, пока $n > 3$. Аналогично будем уменьшать m. В итоге мы получим доску $n \times m$, у которой $1 \leqslant n, m \leqslant 3$. То есть получим первую подзадачу, в которой мы выяснили, что у доски 2×2 ответ `«Black»`, а у остальных — `«White»`.

Если мы уменьшали n все время на 3 и получили в итоге 2, то это значит, что остаток от деления n на 3 равен 2. Аналогично с m. Поэтому получаем следующее решение, которое набирает 100 баллов:

```
n = int(input())
m = int(input())
if n % 3 == 2 and m % 3 == 2:
    print("Black")
else:
    print("White")
```

Альтернативное полное решение

Посмотрим, в каком из четырёх углов находится диагональ под номером 1 (в примерах из условия это всегда левая верхняя клетка). Тогда утверждается, что ответом всегда является цвет клетки в соседнем углу. То есть, если таблица красится, например, начиная с левой верхней клетки, то соседний угол (например, верхняя правая клетка) всегда будет ответом. Доказательство этого оставим читателю в качестве упражнения.

Задача D. Пирамида

Автор задачи: Сулейманов Карам
Разработали: Сулейманов Карам и Даминов Камиль

Дано было N кирпичей с высотой 1, необходимо было построить пирамиду максимальной высоты.

Подзадача 1 (15 баллов)

В этой подзадаче $w_i = w_{i+1}$ и $h_i = h_{i+1}$, то есть все кирпичи одинакового размера. Поэтому нам не важно в каком порядке мы будем брать кирпичи. Так как нам необходимо иметь хотя бы k кирпичей размера не менее $(2k-1) \times (2k-1)$, чтобы можно было построить пирамиду высотой k, напишем неравенство. Нам необходимо максимальное k, которое удовлетворяет условиям: $k \leqslant n$ и $2k - 1 \leqslant min(w_1, h_1)$. Отсюда мы получаем, что $k \leqslant min(n, (min(w_1, h_1) + 1)/2)$. Далее задачу можно решить либо просто перебрав k и взяв максимальное, либо формулой.

Подзадача 2 (25 баллов)

В этой подзадаче $N \leqslant 100$. Для решения данной подзадачи, можно предположить, что исходя из ограничений решение будет за асимптотику похожую на $O(N^3)$. Тогда, хочется перебирать k (то есть ответ) и проверять за $O(N^2)$. Чтобы это сделать, необходимо заметить одну жадность, а именно то, что если мы хотим проверить какой-то c, то давайте будем пытаться собрать пирамидку снизу верх. То есть сначала нижний слой с размерами $(2c-1) \times (2c-1)$, затем лежащий на нем и так далее. Когда мы хотим взять кирпич на i-й слой, достаточно проверить, что есть кирпич размеры которого не меньше необходимых, а затем удалить любой такой, так как любой, подходящий на текущем слое, будет подходить на следующем. Делать это можно просто проходясь по всему массиву за $O(N)$.

Подзадача 3 (30 баллов)

В этой подзадаче $w_i = h_i$. Чтобы решить данную подзадачу, необходимо отсортировать все кирпичи, а потом заметить одну из предложенных жадностей.

1. Пусть мы зафиксировали какое-то k и теперь хотим его проверить. Заметим, что в силу сортировки, лучшие кирпичи будут последние k в порядке сортировки. Более формально, если взяв k последних кирпичей нельзя собрать пирамиду высотой k, то ее нельзя построить никак, иначе можно просто взяв эти k кирпичей. Этот факт следует просто из сортировки, если i-й нельзя взять на какой-то слой, то $(i-1)$-й тоже нельзя, так как $w_{i-1} \leqslant w_i$ и $h_{i-1} \leqslant h_i$. Заметим, что можно сделать бинпоиск по ответу, ведь если можно построить пирамиды высоты k, то можно и $k-1$ (просто взяв $k-1$ последних кирпичей). Получаем решение за $O(N * log(N))$.

2. Альтернативное решение, использует те же факты, однако хочет набирать кирпичи от маленьких к большим. Давайте заметим, что если с помощью кирпичей на префиксе до $i-1$ можно построить пирамиду с максимальной высотой k и i-й кирпич нельзя использовать на $k+1$ слое, то на префиксе до i максимальная пирамида будет иметь высоту k и $k+1$ иначе (это можно понять из рассуждение в первом пункте). Теперь просто будем идти по кирпичам слева направо и поддерживать k — максимальную высоту которую можно достичь используя первые i кирпичей. Получаем асимптотику $O(N * \log(N))$ за сортировку и $O(N)$ за остальное решение.

Полное решение

На самом деле, после решения предыдущей подзадачи хочется как-то отсортировать кирпичи, чтобы рассуждения выполняли. Оказывается это возможно, давай просто заметим, что мы требовали, чтобы выполнялись неравенства $w_i \leqslant w_{i+1}$ и $h_i \leqslant h_{i+1}$. Так давайте просто отсортируем кирпичи в таком порядке, что $min(w_i, h_i) \leqslant min(w_{i+1}, h_{i+1})$, тогда все наши рассуждения будут верны. Далее просто возьмем любое из решений из 3-й подзадачи с такой сортировкой и получим 100 баллов. Для этого можно написать компаратор для встроенной сортировки.

На C++ это будет выглядеть так (если все кирпичи лежат в `vector<pair<int, int>> mas`):

```
bool comp(pair<int, int> i, pair<int, int> j) {
        return min(i.first, i.second) < min(j.first, j.second);
}

int main() {
    ...
    sort(mas.begin(), mas.end(), comp);
    ...
}
```

Либо так:

```
int main() {
    ...
    sort(mas.begin(), mas.end(), [&] (auto i, auto j) {
        return min(i.first, i.second) < min(j.first, j.second);
```

```
    });
    ...
}
```

На Python это будет выглядеть так (если все кирпичи лежат в `list of tuples mas`):

```
def comp(i):
    return min(i[0], i[1])
...
mas.sort(key=comp)
...
```

Либо так:

```
...
mas.sort(key=lambda x: min(x[0], x[1]))
...
```

Задача E. Сложная игра

Автор задачи: Глазков Илья
Разработали: Глазков Илья и Даминов Камиль

Подзадача 1 (7 баллов)

В этой подзадаче $n = 1$. В силу того, что имеется только 1 ряд, у нас на каждом шагу только 1 переход вправо. Отсюда очевидно, что у нас всегда один маршрут, следовательно, есть только 1 способ набрать конфеты.

Подзадача 2 (16 баллов)

В этой подзадаче $n \cdot m \leqslant 10$. Из ограничений так же следует, что $n, m \leqslant 10$. Так как длина каждого пути $n + m$, а каждый шаг может принимать 2 различных значения, всего различных путей $O(2^{n+m})$. Значит, можно перебрать все возможные пути.

Подзадача 3 (21 баллов)

В этой подзадаче $n = 2$. У нас имеется всего два ряда, и при этом мы начинаем в первом, а заканчиваем во втором, поэтому каждый путь будет сначала идти на первом ряду, а затем спускаться на второй и доходить по нему до конца. Давайте создадим массивы p и s, где p_i — это количество конфет на первом ряду и столбцах с номерами от 1 до i, а s_i — это количество конфет на втором ряду и столбцах с номерами от i до m. Теперь путь, который в i-м столбце переходит с первого ряда на второй, будет иметь $p_i + s_i$ конфет. Перебрав i от 1 до m, можно получить все достижимые значения.

Подзадача 4 (14 баллов)

В этой подзадаче в таблице **ровно** две конфеты и нам нужно проверить, можем ли мы собрать ровно две конфеты на пути от начальной до конечной клетки. Конфету b можно собрать после конфеты a, только если $x_b \leqslant x_a$ и $y_b \leqslant y_a$, где x_a, x_b, y_a, y_b — координаты клеток. Проверим это и выведем «YES» или «NO».

Подзадача 5 (25 баллов)

В этой подзадаче $q = 1,\ s_1 = 0$. Для решения данной подзадачи можно использовать множество подходов, однако, в рамках разбора мы ограничимся лишь решением методом динамического программирования. Пусть $dp_{i,j}$ — можно ли дойти до клетки в i ряду и j столбце (клетка (i, j) далее) не собрав ни одной конфеты. Тогда, `dp[i][j] = false`, если в клетке (i, j) есть конфета и `dp[i][j] = dp[i - 1][j] + dp[i][j - 1]` иначе. База динамики `dp[1][1] = true` если в клетке $(1, 1)$ нету конфеты, иначе ответ всегда «NO». Теперь, если `dp[n][m] == true`, то ответ «YES», иначе «NO».

Пример реализации на C++:

```
int main() {
    ...
    vector dp(n + 2, vector(m + 2, false));
    dp[0][1] = true;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            if (d[i][j] == '0')
                dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
    ...
}
```

Полное решение

Давайте будем обозначать клетку, которая находится на пересечении i-й строки и j-го столбца, как $(i,\ j)$. Получается, что в задаче нужно дойти от клетки $(1,\ 1)$ до клетки $(n,\ m)$. Также пусть $candy[i][j]$ — это количество конфет в клетке (i, j).

Как и в предыдущей подзадаче воспользуемся методом динамического программирования. Пусть $dp_{min}[i][j]$ — это минимальное количество конфет на пути от клетки $(1,\ 1)$ до клетки $(i,\ j)$. То есть рассмотрим все возможные правильные пути от клетки $(1,\ 1)$ до клетки $(i,\ j)$, посмотрим сколько конфет лежит на каждом пути, и минимальное из всех этих количеств — это $dp_{min}[i][j]$. Обратите внимание, что мы просто объяснили, что значит $dp_{min}[i][j]$. Не нужно искать все эти значения таким образом, так как это слишком долго. Аналогично пусть $dp_{max}[i][j]$ — это максимальное количество конфет на пути от клетки $(1,\ 1)$ до клетки $(i,\ j)$.

Давайте поймём, как правильно посчитать оба этих значения на примере dp_{min}, так как dp_{max} будет считаться точно также. Мы хотим посчитать $dp_{min}[i][j]$. Давайте поймём, как вообще может выглядеть путь из $(1,\ 1)$ в $(i,\ j)$. Мы знаем, что последняя клетка на этом пути — это $(i,\ j)$. А какой может быть предпоследняя? Каждый ход персонажа — это либо переместиться вниз, либо переместиться вправо, поэтому рассмотрим два случая:

- Персонаж из предпоследней клетки сделал шаг вниз и пришёл в клетку $(i,\ j)$. Получается, что в клетку $(i,\ j)$ игрок пришёл сверху, то есть из клетки $(i - 1,\ j)$. Чтобы путь от клетки $(1,\ 1)$ до $(i,\ j)$ был минимальным по количеству конфет нужно, чтобы путь от $(1,\ 1)$ до $(i - 1,\ j)$ тоже был минимальным. Знаем ли мы, какое минимальное количество конфет будет на пути от $(1,\ 1)$ до $(i - 1,\ j)$? Конечно, ведь это значение $dp_{min}[i - 1][j]$. Получаем, что в этом случае $dp_{min}[i][j] = dp_{min}[i - 1][j] + candy[i][j]$.
- Персонаж из предпоследней клетки сделал шаг вправо и пришёл в клетку $(i,\ j)$. Получается, что в клетку $(i,\ j)$ игрок пришёл сверху, то есть из клетки $(i,\ j - 1)$. Делая точно такие же умозаключения, как и в пункте выше, получаем, что в этом случае $dp_{min}[i][j] = dp_{min}[i][j - 1] + candy[i][j]$.

Получаем следующее:

- $dp_{min}[i][j] = candy[i][j]$, если $i = 1$ и $j = 1$, то есть точка $(i,\ j)$ — стартовая.

- $dp_{min}[i][j] = dp_{min}[i-1][j] + candy[i][j]$, если $i > 1$ и $j = 1$. Так как $j == 1$, то клетка $(i,\ j)$ находится в первом столбце, то есть в клетку $(i,\ j)$ мы можем прийти только сверху.
- $dp_{min}[i][j] = dp_{min}[i][j-1] + candy[i][j]$, если $i = 1$ и $j > 1$. Так как $i == 1$, то клетка $(i,\ j)$ находится в первой строке, то есть в клетку $(i,\ j)$ мы можем прийти только слева.
- $dp_{min}[i][j] = min(dp_{min}[i-1][j] + candy[i][j], dp_{min}[i][j-1] + candy[i][j])$, если $i > 1$ и $j > 1$. Мы берём минимальный из двух возможных вариантов.

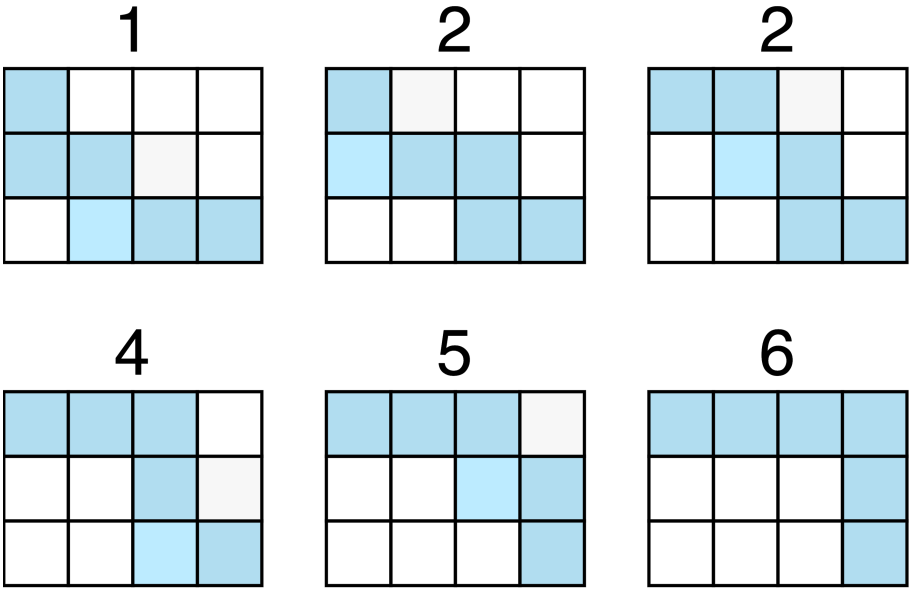
Если перебирать i и j в порядке возрастания, то все необходимые клетки для $(i,\ j)$ уже будут посчитаны. Точно также можно посчитать и dp_{max}, только нужно везде min поменять на max.

Как найти ответ на все запросы? Теперь мы знаем, что минимальный путь из $(1,\ 1)$ до $(n,\ m)$ равен $dp_{min}[n][m]$, а максимальный — $dp_{max}[n][m]$. Теперь, чтобы проверить, что можно получить сумму x на пути на начальной до конечной клетки, нужно проверить, что $dp_{min}[n][m] \leqslant x \leqslant dp_{max}[n][m]$. Докажем это дальше.

Доказательство. Посмотрим, как может выглядеть минимальный путь на примере таблицы 3×4:

1	0	1	0
0	1	1	1
1	0	0	0

Давайте приведем этот путь к пути, который проходит через точку $(1,\ m)$ (правый верхний угол), изменяя ровно одну клетку пути так, чтобы путь оставался правильным. Например это можно сделать так:



За каждое действие мы удаляем одну клетку (в которой максимум одна конфета), и добавляем одну клетку (в которой максимум одна конфета). Получается, что за один ход, мы изменяем количество конфет на пути не более чем на 1.

Заметим, что максимальный путь можно аналогично привести к пути, который проходит через $(1,\ m)$. Так как и минимальный, и максимальный путь можно привести к пути $(1,\ m)$, то и просто минимальный путь можно привести к максимальному, изменяя каждый раз сумму не более чем на 1. Что произошло? Мы взяли путь, на котором ровно $dp_{min}[n][m]$ конфет. Каждый раз мы за одну операцию переходили к другому пути, количество конфет на котором отличается от предыдущего

не более чем на 1. В итоге мы пришли к пути, на котором $dp_{max}[n][m]$ конфет. Из этого можно легко получить, что для любого целого x ($dp_{min}[n][m] \leqslant x \leqslant dp_{max}[n][m]$) существует путь, на котором ровно x конфет. Если это не так, то тогда, чтобы перейти из пути с $\leqslant (x-1)$ конфетами, к пути с $\geqslant (x+1)$ конфетами, нужно изменить путь на $\geqslant 2$ конфеты, чего мы не делаем.

Задача F. Искусственный интеллект

Автор задачи: Даминов Камиль
Разработал: Даминов Камиль

Подзадача 1 (4 балла)

В этой подзадаче все слова состоят только из буквы «a», включая строки запросов, поэтому у любой пары слов будет общая буква. Значит на все запросы ответ будет -1.

Подзадача 2, 3 и 5 (8-28 баллов)

В этой подзадаче сумма длин строк небольшое, поэтому можно реализовать решение за $O(ST)$. Для каждого запроса будем перебирать все возможные варианты строк из коллекции. И среди всех таких, у которых нет общих символов со строкой запросом, будем выбирать лексикографически минимальную. В зависимости от языка программирования и реализации это решение будет набирать от 8 до 28 баллов.

Подзадача 4 (17 баллов)

В этой подзадаче все слова состоят только из букв «a» и «b». Найдем лексикографически минимальную (в этом случае самую короткую) строку, состоящую только из буквы «a» и запишем её номер в переменную A (если такой нет, то $A = -1$). Аналогично найдем лексикографически минимальную (в этом случае самую короткую) строку, состоящую только из буквы «b» и запишем её номер в переменную B (если такой нет, то $B = -1$).

Будем обрабатывать запросы. Мы знаем, что j-й запрос представляет собой строку t_j. Если в строке t_j есть и буква «a», и буква «b», то ответ будет равен -1, потому что все строки состоят только из букв «a» и «b», поэтому с любой строкой s_i всегда будет общий символ. Если строка t_j состоит только из буквы «a», то ответом будет B, так как B — это номер лексикографически минимальной строки, состоящей только из буквы «b», поэтому у этой строки и строки t_j точно не будет общих символов. Аналогично, если строка t_j состоит только из символа «b», то ответом будет A.

В итоге получаем решение за $O(S+T)$.

Подзадача 6 (37 баллов)

В этой подзадаче $w \leqslant 10$, то есть количество символов, из которых состоят все строки, не больше 10. Давайте лексикографически отсортируем все строки и заменим их изначальные номера на номера в этом отсортированным массиве. Теперь наша задача свелась к тому, чтобы просто найти наименьший номер строки, такой, что её символы и символы строки-запроса не пересекаются.

Давайте разберемся, как отвечать за запрос. В j-м запросе у нас есть строка t_j. Давайте посмотрим на те символы, которые используются в строке t_j. Теперь возьмём все возможные w символов, из которых в принципе состоят все символы, и удалим из них те символы, которые используются в t_j. Пусть множество этих символов называется X. Например, если $w = 4$ и $t_j = bc$, то $X = \{a, d\}$, то есть мы взяли все символы $\{a, b, c, d\}$ и убрали из них те, которые используются в t_j (это $\{b, c\}$). Наша задача свелась к тому, чтобы находить минимальный номер строки, которая состоит только из символов множества X.

Давайте будем хранить множество X, как битовую маску. То есть как число, где i-й бит числа равен одному, если в множестве X есть i-й символ. Например, если $X = \{a, d, e\}$, то есть в нем

участвуют 0-й символ («a»), 3-й символ («d») и 4-й символ («e»), тогда маска этого числа будет равна $11001_2 = 2^0 + 2^3 + 2^4 = 25$. Так как всего различных символов w, то и количество различных масок равно 2^w. Заведем массив dp размера 2^w, где $dp[mask]$ — это номер минимальной строки, состоящей только из символов маски $mask$. Если такой строки не будет, то тогда пусть $dp[mask] = n + 1$.

Изначально заполним массив dp числом $n+1$. Теперь переберем все строки коллекции, посчитаем для каждой строки маску множества символов, из который состоит эта строка. Пусть полученная маска равна $mask$, а номер этой строки в отсортированном массиве равен i, тогда нам нужно сделать $dp[mask] = min(dp[mask], i)$.

Но пока массив dp всё равно заполнен некорректно. В $dp[mask]$ учитываются только те строки, в которых есть каждый символ множества маски $mask$, но нам нужно учитывать и такие строки, у которых киких-то символов множества маски $mask$ может и не быть. Для этого давайте будем перебирать $mask$ от 0 до 2^w-1. Значение $dp[mask]$ мы хотим посчитать корректно. Будем перебирать во втором цикле вторую маску $mask2$. Проверим, что $mask2$ является подмножеством маски $mask$. Для этого во много языках программирования можно проверить так: `mask & mask2 == mask2`. Если $mask2$ является подмножеством $mask$, то тогда каждая строка, подходящая под множество $mask2$, будет подходить и под маску $mask$. Сделаем $dp[mask] = min(dp[mask], dp[mask2])$. В итоге эта часть кода будет работать за $O(2^w \cdot 2^w)$, то есть за $O(2^{2w})$. Пример кода на `Python`:

```
for mask in range(2 ** w):
    for mask2 in range(2 ** w):
        if (mask & mask2) == mask2:
            dp[mask] = min(dp[mask], dp[mask2])
```

Вернемся к исходной задаче. Для каждого запроса мы посчитали множества X. Давайте найдём маску этого множества, пусть она равна $mask$. Теперь, если $dp[mask] = n + 1$, то ответ равен -1, иначе ответ это строка под номером $dp[mask]$ в отсортиваронном массиве. Получается, что мы $O(|t_j|)$ посчитали ответ на запрос, то есть за линию.

Полное решение

В прошлом решение единственное место, которое долго работает — это подсчёт массива dp через подмножества. Далее опишем решение, которое посчитать его не за $O(2^{2w})$, а за $O(2^w \cdot w)$. Давайте также будем перебирать $mask$. Будем предполагать, что когда мы рассматриваем это значение, то $dp[mask]$ посчитан корректно. Давайте попробуем добавить символ, которого нет в множестве маски $mask$ и пересчитать значение этого множества через $dp[mask]$. Переберём этот символ, пусть он равен i, тогда маска с добавленным символом i будет равна `mask | (1 « i)`. Тогда давайте пересчитаем $dp[mask|(1 << i)]$ через $dp[mask]$. Если символ i есть в $mask$, то это действие ничего не поменят, а иначе пересчитает всё кореектно. Пример на `Python`:

```
for mask in range(2 ** w):
    for i in range(w):
        dp[mask | (1 << i)] = min(dp[mask | (1 << i)], dp[mask])
```

Не сложно понять, что для любого множетсва, значение подмножества правильно посчитаеся, так как мы перебираем $mask$ в порядке возрастания, а это значит, что все подмножества мы обрабатываем раньше самого множества. В итоге полное решение работает за $O(S \log S + T + 2^w)$, чего хватает для 100 баллов.