

Надпись на табло

В этой задаче набранные баллы зависят от того, насколько аккуратно были разобраны разные случаи входных данных. Сама задача требует значительных навыков программирования.

Рассмотрим возможное решение.

```
import sys

def quit(c):
    print(c)
    sys.exit(0)

n = int(input())

a = [[] for i in range(n)]

for i in range(n):
    a[n - 1 - i] = list(input())

x1 = n + 1
x2 = -1
y1 = n + 1
y2 = -1

for y in range(n):
    for x in range(n):
        if a[y][x] == '#':
            x1 = min(x1, x)
            x2 = max(x2, x)
            y1 = min(y1, y)
            y2 = max(y2, y)

if x1 > x2:
    quit("X")

def find_rect():
    for y in range(y1, y2 + 1):
        for x in range(x1, x2 + 1):
            if a[y][x] == '.':
                y3 = y
                x3 = x
                y4 = y
                x4 = x
                while y4 + 1 <= y2 and a[y4 + 1][x] == '.':
                    y4 += 1
                while x4 + 1 <= x2 and a[y][x4 + 1] == '.':
                    x4 += 1
                for y in range(y3, y4 + 1):
                    for x in range(x3, x4 + 1):
                        if a[y][x] == '#':
                            quit("X")
                        else:
                            a[y][x] = '#'
                return (x3, x4, y3, y4)

return None
```

```

res = find_rect()

if res is None:
    quit("I")
else:
    x3, x4, y3, y4 = res

res = find_rect()
if res is None:
    if x1 < x3 <= x4 < x2 and y1 < y3 <= y4 < y2:
        quit("O")
    elif x1 < x3 <= x4 == x2 and y1 < y3 <= y4 < y2:
        quit("C")
    elif x1 < x3 <= x4 == x2 and y1 < y3 <= y4 == y2:
        quit("L")
    else:
        quit("X")
else:
    x5, x6, y5, y6 = res

res = find_rect()
if res is None:
    if x1 < x3 == x5 <= x4 == x6 < x2 and y1 == y3 <= y4 < y5 <= y6 == y2:
        quit("H")
    elif x1 < x3 == x5 <= x6 < x4 == x2 and y1 == y3 <= y4 < y5 <= y6 < y2:
        quit("P")
    else:
        quit("X")
else:
    quit("X")

```

В этом решении функция `quit` используется для вывода ответа с последующим завершением работы программы.

После чтения входных данных найдём значения x_1 , x_2 , y_1 , y_2 — минимальное и максимальное значения координаты x и координаты y клетки, в которой стоит символ «#». Это — предполагаемые границы внешнего прямоугольника.

Функция `find_rect` используется для выделения внутренних прямоугольников, составленных из символов «.». Она находит самый нижний символ «.», затем определяет высоту и ширину прямоугольника циклами по оси OY и по оси OX , затем во вложенном прямоугольнике заменяет все символы «#» на «.». Если при этом символ «#» будет найден там, где предполагается наличие внутренней части вложенного прямоугольника, сразу же вызываем `quit('X')`. Функция возвращает координаты прямоугольника, а если такого нет, то функция возвращает специальное значение `None`.

Будем вызывать эту функцию для выделения прямоугольников. Если первый же вызов функции вернул `None`, значит, внутри большого прямоугольника нет символов «.», ответом является «I». Если же нам удалось выделить прямоугольник, сохраним его координаты и вызовем функцию `find_rect` повторно. Если второй вызов вернул `None`, значит, внутри есть только один прямоугольник. Проверив условия на его границы, отделим ответы «O», «C», «L» или «X», если прямоугольник один, но условия не выполнены.

Если был найден второй прямоугольник, то убедимся, что больше символов «.» не осталось, вызвав функцию `find_rect` в третий раз. Для двух найденных прямоугольников проверим условия букв «H» и «P», наконец, если ничего не подошло, то выведем «X».

Но у задачи есть и более простое решение. Сначала удалим все нижние и верхние строки, которые не содержат символов «#», а также выключенные столбцы слева и справа.

Теперь «упростим» картинку: будем удалять соседние строки и соседние столбцы, если они совпадают. В результате буква «I» превратится всего лишь в один символ «#», а другие буквы превратятся в небольшие картинки, содержащие не более четырёх строк и трёх столбцов:

```
О   С   L   H   P
###  ##  #.  #.#  ###
#.#  #.  ##  ###  #.#
###  ##          #.#  ###
                        #..
```

После этого просто проверим совпадение полученного массива с одним из возможных вариантов. Пример такого решения.

```
n = int(input())
lines = [list(input()) for _ in range(n)]

i = 0
while i + 1 < len(lines):
    if lines[i] == lines[i + 1]:
        lines.pop(i)
    else:
        i += 1

i = 0
while i + 1 < len(lines[0]):
    if [lines[j][i] for j in range(len(lines))] == \
        [lines[j][i + 1] for j in range(len(lines))]:
        for j in lines:
            j.pop(i)
    else:
        i += 1

while lines and set(lines[0]) == {". "}:
    lines.pop(0)
while lines and set(lines[-1]) == {". "}:
    lines.pop()

while lines and lines[0] and \
    set(lines[j][0] for j in range(len(lines))) == {". "}:
    for j in lines:
        j.pop(0)
while lines and lines[-1] and \
    set(lines[j][-1] for j in range(len(lines))) == {". "}:
    for j in lines:
        j.pop()

for i in range(len(lines)):
    lines[i] = "".join(lines[i])

if lines == ["#"]:
    print("I")
elif lines == ['###', '#.#', '###']:
    print("O")
elif lines == ["##", "#.", "##"]:
```

```
    print("C")
elif lines == ['#.', '##']:
    print("L")
elif lines == ['#.#', '####', '#.#']:
    print("H")
elif lines == ['####', '#.#', '####', '#..']:
    print("P")
else:
    print("X")
```