Муниципальный этап
всероссийской олимпиады школьников по информатике.
2023 – 2024 учебный год 7-8 класс

Для олимпиады даются 4 задачи: 1 очень простая (бланковая форма, предполагается что все участники должны ее решить), 2 простых и 1 средней сложности.
В следующей таблице дана сравнительная характеристика всех 4 задач

Задание	**Тематика**	**Сложность алгоритма**	**Реализация**
1	Простая формула	Простая	Числовые данные
2	Условие	Простая	Файл
3	Составное условие, формула	Простая	Файл
4	Формула, составное условие, цикл	Средняя	Файл

Задача № 1. **Палиндромы (5 баллов)**
Ответом на это задание будут следующие числа.

54345
49094
113311
300003
1000001

Задача № 2. **Ковровая дорожка (10 баллов)**
Искомая площадь получится, если из прямоугольника размером a × b вычесть площадь внутреннего незаполненного прямоугольника размером (a - 2h) × (b - 2h).
Ответ: a * b - (a - 2 * h) * (b - 2 * h). Допустимы и другие формы записи ответа.

Задача № 3. **Урок (10 баллов).**

Для решения нужно просто промоделировать окончание каждого следующего урока. Первый урок заканчивается в 8 ч. 45 мин., а каждый следующий урок первой смены заканчивается через 55 мин. Затем перемена в 25 минут и каждый следующий урок второй смены заканчивается через 55 мин.

```
var
  K,t: integer;
begin
  read(K);
  if K=1 then writeln ('8 45') else
  if (K>1) and (k <=6) then
   begin
     t:= 525 + (k-1)*55;
     writeln (t div 60,' ', t mod 60 ) ;
   end
   else
   begin
     t:= 550 + (k-1)*55;
     writeln (t div 60,' ', t mod 60 ) ;
   end;
end.
```

Задача № 4. **Простой квадрат (10 баллов)**

Для решения этой задачи необходимо было организовать перебор возможных маршрутов, что можно слелать по-разному.

Заметим, что соблюдая правила игры на квадрате всегда можно нарисовать маршрут длины 9, поэтому ответ всегда будет 9-значным. Но поскольку для строк длины 9, составленных из цифр от 1 до 9, и для 9-значных чисел из этих же цифр лексикографический и числовой порядок сортировки совпадают, удобней работать со строками, а не с числами.

Будем хранить квадрат в виде квадратного двумерного массива `A` размера 3×3:

```
A[0][0]  A[0][1]  A[0][2]
A[1][0]  A[1][1]  A[1][2]
A[2][0]  A[2][1]  A[2][2]
```

В массиве будут храниться символы (в языке Питон это строки длины 1).

Ниже приведен вариант решения, в котором перебираются все возможные маршруты на квадрате при помощи “перебора с возвратом” (англ. backtracking).

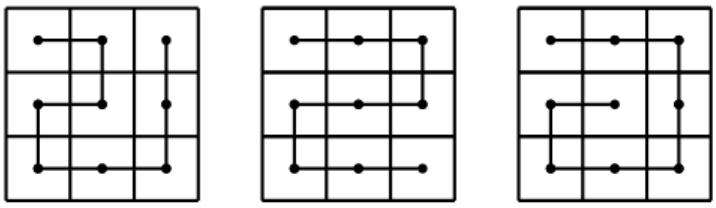
```
Answer = ''

A = [input().split() for i in range(3)]

Visited = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

def Backtracking(prefix, x, y):
    global Answer
    prefix += A[x][y]
    if len(prefix) == 9 and prefix > Answer:
        Answer = prefix
    Visited[x][y] = 1
    for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
        if 0 <= x + dx <= 2 and 0 <= y + dy <= 2 and Visited[x + dx][y + dy] == 0:
            Backtracking(prefix, x + dx, y + dy)
    Visited[x][y] = 0

for x in range(3):
    for y in range(3):
        Backtracking('', x, y)
print(Answer)
```

Смысл перебора с возвратом следующий. Заводится двумерный массив `Visited` в котором делается пометка 1, если данная клетка была посещена и заходить в нее больше нельзя. Если же клетка доступна для посещения, то в ней делается пометка 0. Алгоритм перебора реализован в виде рекурсивной процедуры `Backtracking`, которой в качестве параметров передается строка `prefix` — последовательность уже выписанных чисел и координаты `x` и `y` новой клетки, которая добавляется к маршруту. Данный алгоритм добавляет к строке `prefix` цифру, записанную в клетке `A[x][y]` и проверяет, не получился ли при этом ответ больший, чем сохранен в глобальной переменной `Answer`.

Затем клетка помечается, как посещенная (`Visited[x][y] = 1`). После этого перебираются все соседние клетки в цикле по двум переменным `dx` и `dy`, пара `(dx, dy)` задает направление перемещения, то есть следующая клетка перебираемого маршрута будет иметь координаты `(x + dx, y + dy)`. Проверяется условие невыхода за границы квадрата и если следующая клетка свободна, то функция `Backtracking` запускается из следующей клетки `(x + dx, y + dy)`. Когда функция `Backtracking` выходит из клетки `(x, y)`, нужно отметить ее доступной для посещения (`Visited[x][y] = 0`).

Далее перебираются все клетки квадрата и из каждой клетки запускается функция `Backtracking` для перебора всех маршрутов, начинающихся в этой клетке.

Алгоритм полного перебора с возвратом в данном случае позволяет полностью решить задачу, но во многих случаях он оказывается неэффективен из-за слишком большого объема перебора. В данной задаче можно сократить перебор, если заметить, что существует всего три принципально различные формы маршрута, проходящего через все клетки квадрата.



Следующее решение основано на идее перебора всех этих форм маршрутов. Маршрут может начинаться либо в угловой клетке, либо в центре квадрата. Если он начинается в угловой клетке, то наилучший из всех маршрутов будет начинаться в том углу, в котором записано наибольшее из четырех чисел, записанных во всех углах. Поэтому будем поворачивать квадрат до тех пор, пока наибольшая цифра из четырех угловых клеток не окажется в левом верхнем углу (клетке `A[0][0]`). Для поворота реализуем функцию `Rotate`.

Теперь если наилучший маршрут начинается в угловой клетке, то он начинается в клетке `A[0][0]`. Следующая клетка может быть одна из двух соседних с ней `A[1][0]` или `A[0][1]`. Сравним эти два значения, и если `A[1][0]` окажется больше, отразим квадрат относительно главной диагонали при помощи функции `Mirror`. Тем самым мы добьемся того, что если наилучший маршрут начинается в угловой клетке, то он начинается в клетке `A[0][0]`, затем идет в клету `A[0][1]`. Таких маршрутов всего четыре: три изображены на рисунке, а четвертый маршрут — это маршрут на первом рисунке, но записанный в обратном порядке. Числа вдоль всех этих четырех маршрутов выпишем в переменные `P1`, `P2`, `P3`, `P4` (переменные будут строковыми).

Теперь рассмотрим маршруты, начинающиеся в центре квадрата. Такой маршрут должен из центра перейти в одну из четырех соседних клеток, причем в ту, в которой записано наибольшее число. Опять используя фунцию `Rotate` добьемся того, что наибольшее из чисел, записанных в соседних с центром клетках, будет записано на верхней стороне квадрата, то есть в клетке `A[0][1]`. Маршрутов, начинающихся в `A[1][1]`, затем проходящих через `A[0][1]` всего два — это две спирали, закрученных в разных направления. Числа вдоль этих двух маршрутов выпишем в строковые переменные `P5` и `P6`.

Затем выведем наибольшую из всех строк `P1`, `P2`, `P3`, `P4`, `P5`, `P6`.

```
A = [input().split() for i in range(3)]

# Функция поворачивает квадрат против часовой стрелки
def Rotate():
    A[0][0], A[0][2], A[2][2], A[2][0] = A[0][2], A[2][2], A[2][0], A[0][0]
    A[0][1], A[1][2], A[2][1], A[1][0] = A[1][2], A[2][1], A[1][0], A[0][1]

# Функция отражает квадрат относительно главной диагонали
def Mirror():
    A[0][1], A[1][0] = A[1][0], A[0][1]

     A[0][2], A[2][0] = A[2][0], A[0][2]
     A[1][2], A[2][1] = A[2][1], A[1][2]

 # Ищем лучший путь из угла

 # Максимальное значение, записанное в углах
 MaxInAngles = max(A[0][0], A[0][2], A[2][2], A[2][0])
```

```
# Поворачиваем квадрат, пока наибольшее из значений в углах
# не окажется в левом верхнем углу
while A[0][0] != MaxInAngles:
    Rotate()

# Сравним соседние клетки в левом верхнем углу, при необходимости
# отразим квадрат так, чтомы следущая клетка маршрута была A[0][1]
if A[1][0] > A[0][1]:
    Mirror()

# Все 4 возможных маршрута, начинающихся в A[0][0], затем в A[0][1]
P1 = A[0][0]+A[0][1]+A[1][1]+A[1][0]+A[2][0]+A[2][1]+A[2][2]+A[1][2]+A[0][2]
P2 = A[0][0]+A[0][1]+A[0][2]+A[1][2]+A[2][2]+A[2][1]+A[1][1]+A[1][0]+A[2][0]
P3 = A[0][0]+A[0][1]+A[0][2]+A[1][2]+A[1][1]+A[1][0]+A[2][0]+A[2][1]+A[2][2]
P4 = A[0][0]+A[0][1]+A[0][2]+A[1][2]+A[2][2]+A[2][1]+A[2][0]+A[1][0]+A[1][1]

# Теперь будем искать лучший маршут из середины квадрата

# Максимальное значение, записанное на сторонах
MaxInSides = max(A[0][1], A[1][2], A[2][1], A[1][0])

# Поворачиваем квадрат, пока наибольшее из значений на сторонах
# не окажется на верхней стороне (A[0][1])
while A[0][1] != MaxInSides:
    Rotate()

# Теперь рассмотрим две спирали, начинающиеся в центре, затем
# идущих в клетку A[0][1]

P5 = A[1][1]+A[0][1]+A[0][2]+A[1][2]+A[2][2]+A[2][1]+A[2][0]+A[1][0]+A[0][0]
P6 = A[1][1]+A[0][1]+A[0][0]+A[1][0]+A[2][0]+A[2][1]+A[2][2]+A[1][2]+A[0][2]

# Выведем максимум из всех полученных маршрутов
print(max(P1, P2, P3, P4, P5, P6))
```

Подобный алгоритм перебора можно реализовать и без использования циклов и вспомогательных функций, и даже не используя двумерные массивы (можно хранить числа, записанные в клетках квадрата, в девяти различных переменных).