دانشگاه شهید بهشتی
دانشکده مهندسی برق و کامپیوتر

عنوان:

حل جدول سودوکو با استفاده از بهینه سازی محدب

استاد راهنما:
جناب آقای دکتر مهرشاهی

نگارنده:
نسرین کریمی

شماره دانشجویی:
۴۰۱۴۴۸۱۴۷

خرداد ۱۴۰۲

بسم الله الرحمن الرحیم

Code:

```
import numpy as np

def solve_sudoku_backtracking(sudoku):
    if is_complete(sudoku):
        return sudoku

    i, j = find_next_empty_cell(sudoku)
    possible_values = get_possible_values(sudoku, i, j)

    for value in possible_values:
        sudoku[i, j] = value
        if solve_sudoku_backtracking(sudoku) is not None:
            return sudoku
        sudoku[i, j] = 0

    return None

def is_complete(sudoku):
    return np.all(sudoku != 0)

def find_next_empty_cell(sudoku):
    for i in range(9):
        for j in range(9):
            if sudoku[i, j] == 0:
                return i, j
    return None, None

def get_possible_values(sudoku, i, j):
    row_values = set(sudoku[i, :])
    column_values = set(sudoku[:, j])
    block_values = set(sudoku[(i//3)*3:(i//3)*3+3,
(j//3)*3:(j//3)*3+3].flatten())
    return set(range(1, 10)) - (row_values | column_values |
block_values)

# Example Sudoku puzzle
sudoku_puzzle = np.array([
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
```

```
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
])

print("Sudoku Puzzle:")
print(sudoku_puzzle)

print("\nSolving using backtracking algorithm...\n")
solution = solve_sudoku_backtracking(sudoku_puzzle)

if solution is not None:
    print("Solution:")
    print(solution)
else:
    print("No solution exists.")
```

Result:

```
Sudoku Puzzle:
[[5 3 0 0 7 0 0 0 0]
 [6 0 0 1 9 5 0 0 0]
 [0 9 8 0 0 0 0 6 0]
 [8 0 0 0 6 0 0 0 3]
 [4 0 0 8 0 3 0 0 1]
 [7 0 0 0 2 0 0 0 6]
 [0 6 0 0 0 0 2 8 0]
 [0 0 0 4 1 9 0 0 5]
 [0 0 0 0 8 0 0 7 9]]

Solving using backtracking algorithm...

Solution:
[[5 3 4 6 7 8 9 1 2]
 [6 7 2 1 9 5 3 4 8]
 [1 9 8 3 4 2 5 6 7]
 [8 5 9 7 6 1 4 2 3]
 [4 2 6 8 5 3 7 9 1]
 [7 1 3 9 2 4 8 5 6]
 [9 6 1 5 3 7 2 8 4]
 [2 8 7 4 1 9 6 3 5]
 [3 4 5 2 8 6 1 7 9]]
```

توضیحات مربوط به کد:

یکی از روش‌های ممکن استفاده از بک‌ترکینگ است که یک تکنیک محبوب برای حل مسائل است.
در ابتدا باید تمامی کتابخانه های مربوط را نصب کنیم.

```
import numpy as np
```

این خط کتابخانه NumPy را وارد می کند که از آرایه های چند بعدی و عملیات ریاضی روی آنها پشتیبانی می کند.

```
def solve_sudoku_backtracking(sudoku):
```

این خط تابعی به نام solve_sudoku_backtracking را تعریف می کند که یک پازل سودوکو را به عنوان ورودی می گیرد و سعی می کند آن را با استفاده از الگوریتم backtracking حل کند.

```
    if is_complete(sudoku):
        return sudoku
```

این خط بررسی می‌کند که آیا پازل سودوکو از قبل کامل شده است (یعنی تمام سلول‌ها پر شده‌اند)، و اگر چنین است، پازل حل‌شده را به عنوان راه‌حل برمی‌گرداند.

```
    i, j = find_next_empty_cell(sudoku)
```

این خط تابع find_next_empty_cell را برای بدست آوردن شاخص های سلول خالی بعدی در پازل سودوکو فراخوانی می کند.

```
    possible_values = get_possible_values(sudoku, i, j)
```

این خط تابع get_possible_values را فراخوانی می‌کند تا مقادیر ممکنی را که می‌توان در سلول خالی در شاخص‌های (i، j) قرار داد، تعیین کرد.

```
    for value in possible_values:
        sudoku[i, j] = value
        if solve_sudoku_backtracking(sudoku) is not None:
            return sudoku
        sudoku[i, j] = 0
```

این بلوک کد روی هر مقدار ممکن برای سلول خالی تکرار می شود و سعی می کند پازل سودوکو را به صورت بازگشتی حل کند. یک مقدار را در سلول خالی قرار می دهد، Sol_sudoku_backtracking را به صورت بازگشتی

فراخوانی می کند، و اگر راه حلی پیدا شد، پازل حل شده را برمی گرداند. اگر راه حلی پیدا نشد، با تنظیم مجدد مقدار سلول به ۰، به عقب برمی گردد.

```
    return None
```

اگر راه حلی برای معما سودوکو پیدا نشود، به این خط می رسد که نشان می دهد معما غیر قابل حل است.

```
def is_complete(sudoku):
    return np.all(sudoku != 0)
```

این تابع بررسی می‌کند که آیا پازل سودوکو کامل است یا خیر، اگر تمام سلول‌ها پر شده باشند (غیر صفر) True و در غیر این صورت False را برگرداند.

```
def find_next_empty_cell(sudoku):
    for i in range(9):
        for j in range(9):
            if sudoku[i, j] == 0:
                return i, j
    return None, None
```

این تابع با تکرار بر روی هر سلول و برگرداندن شاخص های اولین سلول خالی یافت شده، سلول خالی بعدی را در پازل سودوکو جستجو می کند. اگر سلول خالی پیدا نشد، None، None را برمی گرداند.

```
def get_possible_values(sudoku, i, j):
    row_values = set(sudoku[i, :])
    column_values = set(sudoku[:, j])
    block_values = set(sudoku[(i//3)*3:(i//3)*3+3, (j//3)*3:(j//3)*3+3].flatten())
    return set(range(1, 10)) - (row_values | column_values | block_values)
```

این تابع مقادیر ممکنی را که می توان در یک سلول خاص (i، j) از پازل سودوکو قرار داد، تعیین می کند. مقادیر موجود در سطر، ستون و بلوک 3 × 3 مربوطه را بررسی می‌کند و مجموعه مقادیری را از ۱ به ۹ برمی‌گرداند که قبلاً وجود ندارند.

بخش باقی مانده از کد وظیفه تعریف مثال پازل سودوکو، فراخوانی تابع solve_sudoku_backtracking برای حل آن، چاپ راه حل یا نشان دادن عدم وجود راه حل است.

```
# Example Sudoku puzzle
sudoku_puzzle = np.array([
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
```

```
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
])

print("Sudoku Puzzle:")
print(sudoku_puzzle)

print("\nSolving using backtracking algorithm...\n")
solution = solve_sudoku_backtracking(sudoku_puzzle)

if solution is not None:
    print("Solution:")
    print(solution)
else:
    print("No solution exists.")
```