Shakufa Shendrik

CS472 Assignment 07: Computational Complexity

1. Write a program that a) Converts a Sudoku puzzle stored in a 9x9 array into a graph, per the construction above, b) uses your favorite graph library to 9-color the resulting graph, c) and then uses the coloring to complete the puzzle.

```
"""
   CS472 - Assignment 7, program 3, Computational Complexity
   Shakufa Shendrik
   Sudoku solver
"""

import networkx as nx
from networkx.algorithms.coloring import greedy_color

# Function to convert a Sudoku grid into a graph
def sudoku_to_graph(sudoku):
   G = nx.Graph()
   # Create nodes for each cell in the 9x9 grid
   for row in range(9):
       for col in range(9):
           G.add_node((row, col), value=sudoku[row][col])

   # Add edges between nodes that are in the same row, column, or block
   for row in range(9):
       for col in range(9):
           for r2 in range(row + 1, 9):
               G.add_edge((row, col), (r2, col))  # Same column
           for c2 in range(col + 1, 9):
               G.add_edge((row, col), (row, c2))  # Same row
           # Add edges for the 3x3 subgrid
           block_row_start = (row // 3) * 3
           block_col_start = (col // 3) * 3
           for r in range(block_row_start, block_row_start + 3):
               for c in range(block_col_start, block_col_start + 3):
                   if (r, c) != (row, col):
                       G.add_edge((row, col), (r, c))  # Same block

   return G
```

```
# Function to solve the Sudoku using the graph coloring
def solve_sudoku(sudoku):
    G = sudoku_to_graph(sudoku)

    # Use greedy coloring to assign numbers (1-9) to nodes (cells in Sudoku)
    coloring = greedy_color(G, strategy="largest_first")

    # Reconstruct the Sudoku grid from the coloring
    solution = [[0 for _ in range(9)] for _ in range(9)]
    for (row, col), color in coloring.items():
        solution[row][col] = color + 1  # Colors are zero-indexed, so add 1 to map to
Sudoku numbers

    return solution

# Function to print the Sudoku grid
def print_sudoku(sudoku):
    for row in sudoku:
        print(" ".join(str(num) if num != 0 else "." for num in row))

# Example Sudoku puzzle (0 represents empty cells)
sudoku = [
        [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]
    ]
#sudoku = [
#     [0, 0, 3, 0, 2, 0, 6, 0, 0],
#     [9, 0, 0, 3, 0, 5, 0, 0, 1],
#     [0, 0, 1, 8, 0, 6, 4, 0, 0],
#     [0, 0, 8, 1, 0, 2, 9, 0, 0],
#     [7, 0, 0, 0, 0, 0, 0, 0, 8],
#     [0, 0, 6, 7, 0, 8, 2, 0, 0],
#     [0, 0, 2, 6, 0, 9, 5, 0, 0],
#     [8, 0, 0, 2, 0, 3, 0, 0, 9],
#     [0, 0, 5, 0, 1, 0, 3, 0, 0]
```

```
#]

# Solve the Sudoku
solution = solve_sudoku(sudoku)

# Print the solved Sudoku
print("Solved Sudoku:")
print_sudoku(solution)
```

```
(cs472) $ python prog3_sudoku.py
Solved Sudoku:
4 5 10 3 7 1 9 2 6
9 3 1 2 8 4 5 10 7
6 8 2 9 10 5 4 3 1
1 10 6 5 3 2 7 4 9
3 4 5 10 6 7 8 1 2
7 2 9 4 1 8 3 6 5
5 11 3 6 4 10 1 9 8
8 7 4 1 2 9 6 5 3
2 1 12 8 5 3 10 7 4
(cs472) $
```

2.Sorting *n* numbers can be reduced to convex hull computation by mapping so that the convex hull then yields points sorted by *x*. Since sorting requires Ω(*n* log *n*), convex hull (in this case) has a lower bound of Ω(*n* log *n*). Source: https://www.geeksforgeeks.org/convex-hull-algorithm/

3. This is a *reduction* algorithm that transforms the Max-Cut problem into another problem. The goal is to visit each edge at least once. To reduce Max-Cut in a planar graph G (where G represents graph) to a shortest tour in its dual G*, we follow these steps: 1) Embed G by drawing *G* planarly 2) Form G where *a)* faces of *G* → vertices of *G** and b) edges between faces in *G* → weighted edges in *G**. 3) Then we find the shortest tour visiting all edges of *G** 4) Tour in *G** → cut in *G* (partitioning vertices).5) Select the maximum-weight cut (Max-Cut step). So the complete algorithm is:

1. Embed *G*.
2. Form dual *G*.

3. Find shortest tour in *G*.
4. Tour → cut in *G*.
5. Select max-weight cut.

4. The complexity class P contains decision problems solvable in polynomial time. However, the given brute-force algorithm has a time complexity of O(n). Since the input size (b) is the number of bits to represent n, the algorithm's is actually exponential time complexity, meaning the algorithm does not run in polynomial time and therefore does not put the composite number problem into the class P. source: https://www.geeksforgeeks.org/composite-number/