

AIML LAB 7

U18CO021: SAHIL BONDRE

Question

Consider the problem of solving crossword puzzles: fitting words into a rectangular grid. The grid, which is given as part of the problem, specifies which squares are blank and which are shaded. For each word starting square you have a list of words that can be fitted (vertical and/or across).

The task is to fill in the blank squares using any subset of the given lists.

Hints:

word lists to be used

- 1A: hoses, laser, sails, sheet, steer
- 4A: heel, hike, keel, knot, line
- 7A: aft, ale, eel, lee, tie
- 8A: hoses, laser, sails, sheet, steer
- 2V: hoses, laser, sails, sheet, steer
- 3V: hoses, laser, sails, sheet, steer
- 5V: heel, hike, keel, knot, line
- 6V: aft, ale, eel, lee, tie

note that A stands for across (horizontal) and V for vertical. use the String domain and the crossword constraints

Here it is the grid definition:

1		2		3
	4		5	
6		7		
8				

Solution

We are going to solve this problem by using Constraint Satisfaction:

cell.py

```
superscript_map = {  
    "0": "⁰", "1": "¹", "2": "²", "3": "³", "4": "⁴", "5": "⁵", "6": "⁶",  
    "7": "⁷", "8": "⁸", "9": "⁹"}  

```

```
class Cell:  
    def __init__(self, is_black: bool = False, is_annoted: bool = False,  
annotation: int = 0, ):  
        self.is_black = is_black  
        # Character in Cell, "0" => black cell, " " => empty cell  
        self.value = " "  
        # annotation is the number if any that the cell carries  
        self.is_annoted = is_annoted  
        self.annotation = annotation  

```

```
@property  
def is_empty(self):
```

```

        return self.value == " " and not self.is_black

    def clone(self):
        cell = Cell(self.is_black, self.is_annotated, self.annotation)
        cell.value = self.value
        return cell

    def __str__(self):
        value = self.value if self.value != "0" else " "
        if self.is_black:
            return "|" + 3 * "\u2588" + "|"
        if self.is_annotated:
            return f"|{superscript_map[str(self.annotation)]} {value}|"
        return f"| {value}|"

    def fill(self, character: str):
        if len(character) != 1:
            raise Exception("Length of character should be 1")
        self.value = character

```

crossword.py

```

from cell import Cell
from enum import Enum

class Direction(Enum):
    HORIZONTAL = 0
    VERTICAL = 1

class Crossword:
    def __init__(self, grid):
        """
        grid specs:
            'X': Black
            '0': Empty
            '1': annotation
        """
        self.grid: list[list[Cell]] = []
        for row in grid:
            new_row = []
            for cell in row:
                if cell == "X":

```

```

        new_row.append(Cell(True))
    elif cell == "0":
        new_row.append(Cell())
    else:
        new_row.append(Cell(False, True, int(cell)))
    self.grid.append(new_row)

def print(self):
    for row in self.grid:
        for cell in row:
            print(cell, end=" ")
        print()
    print()

def clone(self):
    res = Crossword([])
    grid = []
    for row in self.grid:
        new_row = []
        for cell in row:
            new_row.append(cell.clone())
        grid.append(new_row)
    res.grid = grid
    return res

def _in_range(self, r: int, c: int) -> bool:
    return len(self.grid) > r and len(self.grid[r]) > c

def can_fit(self, word: str, r: int, c: int, dir: Direction) ->
bool:
    if not self._in_range(r, c):
        raise Exception("Coordinates out of bound")

    l = len(word)
    if dir == Direction.HORIZONTAL:
        for i in range(l):
            if not self._in_range(r, c + i):
                return False
            cell: Cell = self.grid[r][c + i]
            if cell.is_black or (cell.value != " " and cell.value !=
word[i]):
                return False

    if dir == Direction.VERTICAL:
        for i in range(l):
            if not self._in_range(r + i, c):

```

```

        return False
        cell: Cell = self.grid[r + i][c]
        if cell.is_black or (cell.value != " " and cell.value !=
word[i]):
            return False

    return True

def is_filled(self):
    for row in self.grid:
        for cell in row:
            if cell.is_empty:
                return False
    return True

def fit(self, word: str, r: int, c: int, dir: Direction):
    if not self.can_fit(word, r, c, dir):
        raise Exception("Cannot fit word")

    l = len(word)
    if dir == Direction.HORIZONTAL:
        for i in range(l):
            cell: Cell = self.grid[r][c + i]
            cell.fill(word[i])

    if dir == Direction.VERTICAL:
        for i in range(l):
            cell: Cell = self.grid[r + i][c]
            cell.fill(word[i])

```

index.py

```

from crossword import Crossword, Direction

grid = [
    "10203",
    "XX0X0",
    "X4050",
    "6X700",
    "80000",
    "0XX0X"
]
c = Crossword(grid)

```

```

def solve(cwd: Crossword, words: list) -> True:
    if cwd.is_filled():
        print("Solved!")
        cwd.print()
        print(f"Unused words: {words}")
        return True

    if not words:
        return False

    words_copy = list.copy(words)
    word = words_copy.pop(0)
    r = len(cwd.grid)
    if r == 0:
        return False
    c = len(cwd.grid[0])

    for x in range(r):
        for y in range(c):
            if cwd.grid[x][y].is_black:
                next
            clone = cwd.clone()
            if clone.can_fit(word, x, y, Direction.HORIZONTAL):
                clone.fit(word, x, y, Direction.HORIZONTAL)
                if solve(clone, words_copy):
                    return True
            clone = cwd.clone()
            if clone.can_fit(word, x, y, Direction.VERTICAL):
                clone.fit(word, x, y, Direction.VERTICAL)
                if solve(clone, words_copy):
                    return True
            clone = cwd.clone()

    return False

words = ["hoses", "hike", "steer", "laser",
         "sails", "keel", "ale", "eel", "sheet", "heal", "line", "aft",
         "tie", "knot"]

if not solve(c, words):
    print("No Solution Found :(")

```