

AI-DS ASSIGNMENT WORK

BY P Venkata Vishnu Vardhan Reddy

125156162

AIM: To solve a Crossword Puzzle using Constraint satisfaction Problem..

Problem Statement::

We are given with a Block that has some empty space and some filled spaces. We are provided with set of words in a file and our work is to fill empty space of given block with appropriate word of perfect length that satisfies the length of empty space.

Code ::

[Crossword.py:](#)

```
class Variable():
    ACROSS = "across"
    DOWN = "down"
    def __init__(self, i, j, direction, length):
        self.i = i
        self.j = j
        self.direction = direction
        self.length = length
        self.cells = []
        for k in range(self.length):
            self.cells.append(
                (self.i + (k if self.direction == Variable.DOWN else 0),
                 self.j + (k if self.direction == Variable.ACROSS else 0))
            )
    def __hash__(self):
        return hash((self.i, self.j, self.direction, self.length))
    def __eq__(self, other):
        return (
```

```

        (self.i == other.i) and
        (self.j == other.j) and
        (self.direction == other.direction) and
        (self.length == other.length)
    )
def __str__(self):
    return f"({self.i}, {self.j}) {self.direction} : {self.length}"
def __repr__(self):
    direction = repr(self.direction)
    return f"Variable({self.i}, {self.j}, {direction}, {self.length})"
class Crossword():
    def __init__(self, structure_file, words_file):
        with open(structure_file) as f:
            contents = f.read().splitlines()
            self.height = len(contents)
            self.width = max(len(line) for line in contents)
            self.structure = []
            for i in range(self.height):
                row = []
                for j in range(self.width):
                    if j >= len(contents[i]):
                        row.append(False)
                    elif contents[i][j] == "_":
                        row.append(True)
                    else:
                        row.append(False)
                self.structure.append(row)
        with open(words_file) as f:
            self.words = set(f.read().upper().splitlines())
        self.variables = set()
        for i in range(self.height):
            for j in range(self.width):
                starts_word = (
                    self.structure[i][j]
                    and (i == 0 or not self.structure[i - 1][j])
                )
                if starts_word:

```

```

length = 1
for k in range(i + 1, self.height):
    if self.structure[k][j]:
        length += 1
    else:
        break
if length > 1:
    self.variables.add(Variable(
        i=i, j=j,
        direction=Variable.DOWN,
        length=length
    ))
starts_word = (
    self.structure[i][j]
    and (j == 0 or not self.structure[i][j - 1])
)
if starts_word:
    length = 1
    for k in range(j + 1, self.width):
        if self.structure[i][k]:
            length += 1
        else:
            break
    if length > 1:
        self.variables.add(Variable(
            i=i, j=j,
            direction=Variable.ACROSS,
            length=length
        ))
self.overlaps = dict()
for v1 in self.variables:
    for v2 in self.variables:
        if v1 == v2:
            continue
        cells1 = v1.cells
        cells2 = v2.cells
        intersection = set(cells1).intersection(cells2)

```

```

        if not intersection:
            self.overlaps[v1, v2] = None
        else:
            intersection = intersection.pop()
            self.overlaps[v1, v2] = (
                cells1.index(intersection),
                cells2.index(intersection)
            )
    def neighbors(self, var):
        return set(
            v for v in self.variables
            if v != var and self.overlaps[v, var]
        )

```

Generate.py::

```

import sys
from crossword import *
class CrosswordCreator():
    def __init__(self, crossword):
        self.crossword = crossword
        self.domains = {
            var: self.crossword.words.copy()
            for var in self.crossword.variables
        }
    def letter_grid(self, assignment):
        letters = [
            [None for _ in range(self.crossword.width)]
            for _ in range(self.crossword.height)
        ]
        for variable, word in assignment.items():
            direction = variable.direction
            for k in range(len(word)):
                i = variable.i + (k if direction == Variable.DOWN else 0)
                j = variable.j + (k if direction == Variable.ACROSS else 0)

```

```

        letters[i][j] = word[k]
    return letters

def print(self, assignment):
    letters = self.letter_grid(assignment)
    for i in range(self.crossword.height):
        for j in range(self.crossword.width):
            if self.crossword.structure[i][j]:
                print(letters[i][j] or " ", end="")
            else:
                print("■", end="")
        print()

def save(self, assignment, filename):
    from PIL import Image, ImageDraw, ImageFont
    cell_size = 100
    cell_border = 2
    interior_size = cell_size - 2 * cell_border
    letters = self.letter_grid(assignment)
    img = Image.new(
        "RGBA",
        (self.crossword.width * cell_size,
         self.crossword.height * cell_size),
        "black"
    )
    font = ImageFont.truetype("assets/fonts/OpenSans-Regular.ttf", 80)
    draw = ImageDraw.Draw(img)
    for i in range(self.crossword.height):
        for j in range(self.crossword.width):
            rect = [
                (j * cell_size + cell_border,
                 i * cell_size + cell_border,
                 ((j + 1) * cell_size - cell_border,
                  (i + 1) * cell_size - cell_border)
            ]
            if self.crossword.structure[i][j]:
                draw.rectangle(rect, fill="white")
                if letters[i][j]:
                    _, _, w, h = draw.textbbox((0, 0), letters[i][j], font=font)

```

```

        draw.text(
            (rect[0][0] + ((interior_size - w) / 2),
             rect[0][1] + ((interior_size - h) / 2) - 10),
            letters[i][j], fill="black", font=font
        )
    img.save(filename)
def solve(self):
    self.enforce_node_consistency()
    self.ac3()
    return self.backtrack(dict())
def enforce_node_consistency(self):
    for variable in self.crossword.variables:
        for word in self.crossword.words:
            if len(word) != variable.length:
                self.domains[variable].remove(word)
def enforce_node_consistency(self):
    for variable in self.crossword.variables:
        self.domains[variable] = [word for word in self.domains[variable] if
len(word) == variable.length]
def revise(self, x, y):
    var1, var2 = self.crossword.overlaps[x, y]
    revised = False
    for x_word in set(self.domains[x]):
        valid_overlap = any(x_word[var1] == y_word[var2] for y_word in
self.domains[y])
        if not valid_overlap:
            self.domains[x].remove(x_word)
            revised = True
    return revised
def ac3(self, arcs=None):
    if arcs is None:
        arcs = []
    for x in self.domains:
        for y in self.crossword.neighbors(x):
            arcs.append((x,y))
    while arcs:
        (x,y) = arcs.pop()

```

```

    if self.revise(x,y):
        if not self.domains[x]:
            return False
        for z in self.crossword.neighbors(x) - set(self.domains[y]):
            arcs.append((z,x))
    return True
def assignment_complete(self, assignment):
    for variable in self.crossword.variables:

        if variable not in assignment.keys():
            return False
        if not assignment.get(variable):
            return False
    return True
def consistent(self, assignment):
    for x_1 in assignment:
        word = assignment.get(x_1)
        if len(word) != x_1.length:
            return False
    for x_2 in assignment:
        word2 = assignment.get(x_2)
        if x_1 != x_2:
            if word == word2:
                return False
            overlap = self.crossword.overlaps[x_1, x_2]
            if overlap:
                a, b = overlap
                if word[a] != word2[b]:
                    return False
    return True
def order_domain_values(self, var, assignment):
    results = {}
    for i in self.domains[var]:
        results[i] = 0
        for neighbor in self.crossword.neighbors(var) - assignment.keys():
            if i in self.domains[neighbor]:
                results[i] += 1

```

```

        return sorted(results, key=results.get)
    def select_unassigned_variable(self, assignment):
        unassigned_vars = [var for var in self.crossword.variables if var not in
assignment]
        if not unassigned_vars:
            return None
        variable = min(unassigned_vars, key=lambda var:
(len(self.domains[var]), -len(self.crossword.neighbors(var))))
        return variable
    def backtrack(self, assignment):
        if self.assignment_complete(assignment):
            return assignment
        var = self.select_unassigned_variable(assignment)
        for value in self.order_domain_values(var, assignment):
            assignment[var] = value
            if self.consistent(assignment):
                result = self.backtrack(assignment)
                if result is not None:
                    return result
            assignment[var] = None
        return None
def main():
    if len(sys.argv) not in [2, 3]:
        sys.exit("Usage: python generate.py structure words [output]")
    structure = sys.argv[1]
    words = sys.argv[2]
    output = sys.argv[3] if len(sys.argv) == 4 else None
    crossword = Crossword(structure, words)
    creator = CrosswordCreator(crossword)
    assignment = creator.solve()
    if assignment is None:
        print("No solution.")
    else:
        creator.print(assignment)
        if output:
            creator.save(assignment, output)
if __name__ == "__main__":

```


main()

[“Text files containing words and structures files are attached along with this file.”](#)

Command to run generate.py code:::

Python generate.py structure0.txt words0.txt

OUTPUT::

For word0.txt and structure0.txt:

```
pvenkatavishnuvardhanreddy@macbookm2 desktop % python3 generate.py structure0.txt words0.txt
SIX
E F
V I
E V
NINE
```

For word1.txt and structure1.txt:

```
pvenkatavishnuvardhanreddy@macbookm2 desktop % python3 generate.py structure1.txt words1.txt
  M R
  | |
I N T E L L I G E N C E
N   N   S
F   N   O L V E
E   M   A
R   X   E
  | |
  | |
```

For word2.txt and structure2.txt:

```
pvenkatavishnuvardhanreddy@macbookm2 desktop % python3 generate.py structure2.txt words2.txt
  W
  |
F L O W
A   O N T O
I   R   D
L   R   E
  |   N
  |   N
```

