

|  |  |
| --- | --- |
| **Student Name** | **Arun Adhikari** |
| **SRN No** | 202100406 |
| **Roll No** | 31 |
| **Program** | **Department of Computer Engineering** |
| **Year** | 2024-2025 |
| **Division** | G |
| **Subject** | Artificial Intelligence lab |
| **Assignment No** | 6 |



**Experiment Number - 06**

**Title /Problem Statement: Implement Constraint Satisfaction Algorithm for the following problems: a. Cryptarithmetic b. Crossword puzzle c. Map coloring problem**

# Description: Constraint Satisfaction Problems (CSPs) are mathematical questions defined by a set of objects whose state must satisfy several constraints. CSPs are widely used in AI applications such as scheduling, planning, and resource allocation. In this assignment:

# Cryptarithmetic solves alphabets-to-digits puzzles ensuring unique assignments.

# Crossword puzzle filling ensures that each word fits correctly into a given grid.

# Map coloring problem assigns colors to regions without adjacent regions sharing the same color.

**Theory: A Constraint Satisfaction Problem consists of:**

* **Variables (to be assigned)**
* **Domains (possible values for each variable)**
* **Constraints (conditions that must be satisfied)**

**CSP algorithms, like backtracking with forward checking and constraint propagation, systematically search for solutions while reducing unnecessary paths:**

* **Cryptarithmetic uses backtracking to assign digits to letters ensuring arithmetic correctness.**
* **Crossword puzzle filling applies backtracking to fit words without conflict.**
* **Map Coloring uses backtracking to assign different colors to neighboring regions with minimum conflicts.**

**Efficiency is improved with heuristics like:**

* **Minimum Remaining Values (MRV): Choose the variable with the fewest legal moves.**
* **Degree heuristic: Select the variable involved in the largest number of constraints.**
* **Forward Checking: Remove incompatible values ahead of time.**

Part A:

**Code:**

**from itertools import permutations**

**def solve\_cryptarithmetic():**

**letters = 'SENDMORY'**

**digits = range(10)**

**for perm in permutations(digits, len(letters)):**

**assign = dict(zip(letters, perm))**

**if assign['S'] == 0 or assign['M'] == 0:**

**continue**

**send = assign['S']\*1000 + assign['E']\*100 + assign['N']\*10 + assign['D']**

**more = assign['M']\*1000 + assign['O']\*100 + assign['R']\*10 + assign['E']**

**money = assign['M']\*10000 + assign['O']\*1000 + assign['N']\*100 + assign['E']\*10 + assign['Y']**

**if send + more == money:**

**return assign, send, more, money**

**solution, send, more, money = solve\_cryptarithmetic()**

**print(f"SEND: {send}, MORE: {more}, MONEY: {money}")**

**print("Assignment:", solution)**

**Part 2:**

**Code:**

**from itertools import product**

**def crossword\_solver():**

**words = ["HELLO", "HOUSE", "SOLAR", "SHORE", "LARGE", "LEARN", "ROSES"]**

**slot1 = slot2 = slot3 = words**

**for w1, w2, w3 in product(slot1, slot2, slot3):**

**if w1[0] == w2[0] and w2[2] == w3[2]:**

**print(f"Slot1 (across): {w1}")**

**print(f"Slot2 (down):   {w2}")**

**print(f"Slot3 (across): {w3}")**

**return**

**crossword\_solver()**

**Part 3:**

**Code:**

**def map\_coloring():**

**colors = ['Red', 'Green', 'Blue']**

**regions = {**

**'WA': ['NT', 'SA'],**

**'NT': ['WA', 'SA', 'Q'],**

**'SA': ['WA', 'NT', 'Q', 'NSW', 'V'],**

**'Q': ['NT', 'SA', 'NSW'],**

**'NSW': ['SA', 'Q', 'V'],**

**'V': ['SA', 'NSW'],**

**'T': []**

**}**

**def is\_valid(assign, region, color):**

**for neighbor in regions[region]:**

**if neighbor in assign and assign[neighbor] == color:**

**return False**

**return True**

**def backtrack(assign):**

**if len(assign) == len(regions):**

**return assign**

**for region in regions:**

**if region not in assign:**

**for color in colors:**

**if is\_valid(assign, region, color):**

**assign[region] = color**

**result = backtrack(assign)**

**if result:**

**return result**

**del assign[region]**

**return None**

**solution = backtrack({})**

**print("Map Coloring Solution:")**

**for region in sorted(solution):**

**print(f"{region}: {solution[region]}")**

**map\_coloring()**

**def map\_coloring():**

**colors = ['Red', 'Green', 'Blue']**

**regions = {**

**'WA': ['NT', 'SA'],**

**'NT': ['WA', 'SA', 'Q'],**

**'SA': ['WA', 'NT', 'Q', 'NSW', 'V'],**

**'Q': ['NT', 'SA', 'NSW'],**

**'NSW': ['SA', 'Q', 'V'],**

**'V': ['SA', 'NSW'],**

**'T': []**

**}**

**def is\_valid(assign, region, color):**

**for neighbor in regions[region]:**

**if neighbor in assign and assign[neighbor] == color:**

**return False**

**return True**

**def backtrack(assign):**

**if len(assign) == len(regions):**

**return assign**

**for region in regions:**

**if region not in assign:**

**for color in colors:**

**if is\_valid(assign, region, color):**

**assign[region] = color**

**result = backtrack(assign)**

**if result:**

**return result**

**del assign[region]**

**return None**

**solution = backtrack({})**

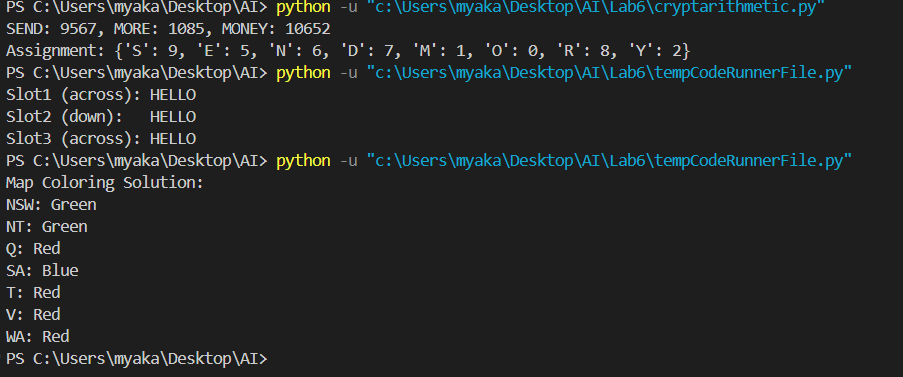
**print("Map Coloring Solution:")**

**for region in sorted(solution):**

**print(f"{region}: {solution[region]}")**

**map\_coloring()**

**Output Screenshot:**



**Conclusion:**

Constraint Satisfaction Problems (CSPs) provide an efficient framework for solving complex problems where a solution must satisfy a number of restrictions. In this experiment, we explored CSP techniques applied to Cryptarithmetic, Crossword Puzzle filling, and Map Coloring problems. Each problem demonstrated how systematic search strategies like backtracking and constraint propagation can find valid solutions efficiently. Cryptarithmetic showcased the assignment of unique digits to letters while satisfying arithmetic equations, crossword filling emphasized fitting words accurately within a defined structure, and map coloring ensured that no adjacent regions shared the same color using a limited set of colors. These problems highlight the importance of logical reasoning, domain reduction, and intelligent search in Artificial Intelligence. Overall, CSPs remain a foundational approach in AI for solving structured problems with clearly defined rules and constraints.