**AI Project 2**

**Name: Samrudhi Katariya**

**Class: TY F**

**Roll No.: 14**

**SRN No.: 202202074**

**Title: Cryptarithmetic Solver using Best-First Search Algorithm**

1. **Problem Statement and Objectives:**

Cryptarithmetic puzzles are mathematical equations in which digits are replaced by letters. Each letter represents a unique digit from 0 to 9. The objective of this project is to implement an intelligent algorithm that solves such puzzles using **Best-First Search** guided by a **heuristic function**. The solution must:

* Explore the search space of possible digit-letter assignments.
* Use a heuristic to prioritize promising assignments.
* Validate and output correct digit-letter mappings.
* Provide a user-friendly interface to input equations and visualize results.

1. **Methodology details:**

**1. Identify Dataset**

No dataset is required, as the user manually inputs the cryptarithmetic equation (e.g., SEND + MORE = MONEY).

**2. Preprocess Input**

Extract unique characters.

Validate equation format.

Sanitize input by removing whitespace and converting to uppercase.

**3. Implement Algorithm**

Best-First Search is used with a heuristic function.

A priority queue stores states based on how close their evaluation is to a valid result.

Each state is a partial assignment of digits to letters.

**4. Heuristic Function**

The heuristic evaluates how close a current assignment is to satisfying the equation.

It returns the absolute difference between the evaluated left-hand and right-hand sides.

**5. GUI Implementation**

A command-line interface is provided where the user can input the equation.

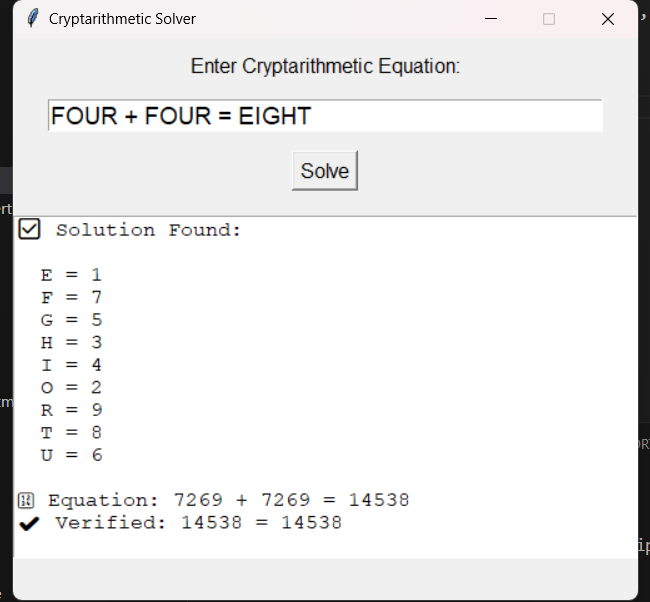
Optionally, a GUI can be developed using Streamlit or Tkinter for enhanced UX.

**6. Output Validation**

The solution mapping is verified by substituting digits back into the equation.

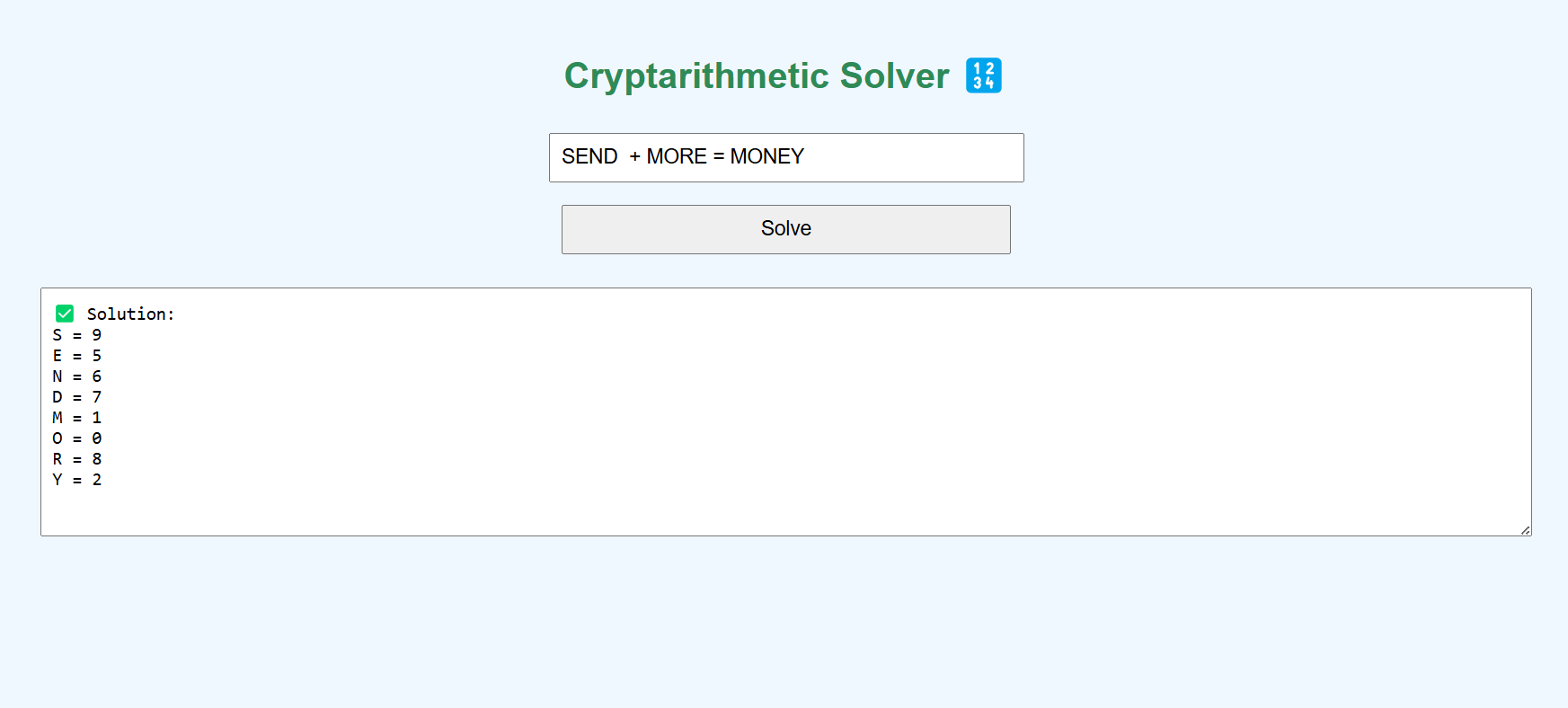
A correct mapping produces a valid arithmetic expression with no leading zeros.

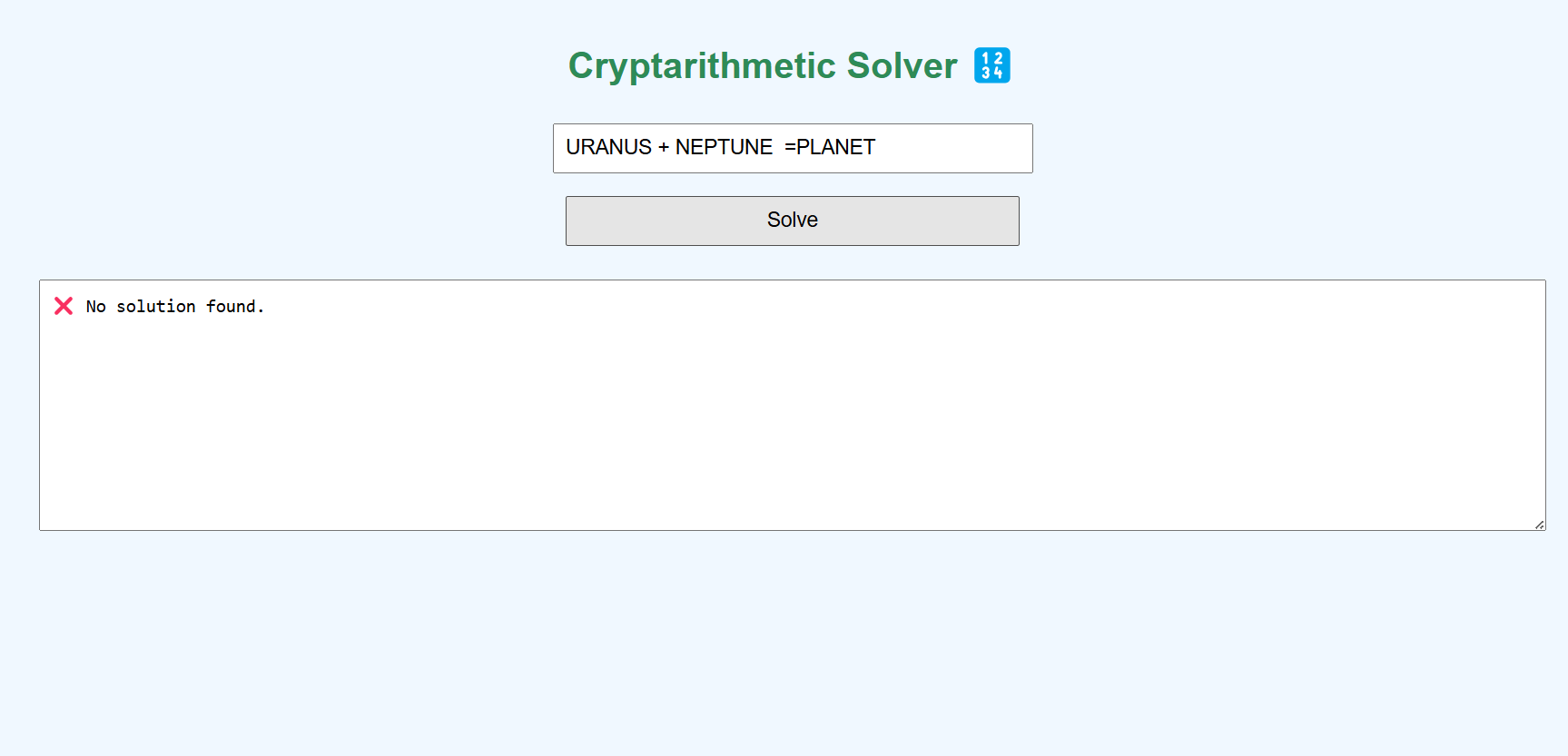
1. **Source code:**
2. import heapq
3. import re
4. class Node:
5. def \_\_init\_\_(self, assignment, score):
6. self.assignment = assignment
7. self.score = score
8. def \_\_lt\_\_(self, other):
9. return self.score < other.score
10. def extract\_letters(equation):
11. return sorted(set(re.findall(r'[A-Z]', equation)))
12. def is\_valid\_number(word, assignment):
13. if word[0] in assignment and assignment[word[0]] == 0:
14. return False  # Leading zeros not allowed
15. return True
16. def substitute(equation, assignment):
17. table = str.maketrans({k: str(v) for k, v in assignment.items()})
18. return equation.translate(table)
19. def compute\_score(equation, assignment):
20. if not all(letter in assignment for letter in re.findall(r'[A-Z]', equation)):
21. return float('inf')  # Incomplete assignment
22. if not all(is\_valid\_number(word, assignment) for word in re.findall(r'\b[A-Z]+\b', equation)):
23. return float('inf')  # Invalid due to leading zero
24. substituted = substitute(equation, assignment)
25. left, right = substituted.split('=')
26. try:
27. return abs(eval(left) - eval(right))
28. except:
29. return float('inf')
30. def best\_first\_search(equation):
31. equation = equation.replace(" ", "").upper()
32. letters = extract\_letters(equation)
33. frontier = []
34. heapq.heappush(frontier, Node({}, 0))
35. visited = set()
36. while frontier:
37. current = heapq.heappop(frontier)
38. if len(current.assignment) == len(letters):
39. if compute\_score(equation, current.assignment) == 0:
40. return current.assignment
41. for digit in range(10):
42. for letter in letters:
43. if letter not in current.assignment and digit not in current.assignment.values():
44. new\_assign = current.assignment.copy()
45. new\_assign[letter] = digit
46. key = tuple(sorted(new\_assign.items()))
47. if key not in visited:
48. visited.add(key)
49. score = compute\_score(equation, new\_assign)
50. heapq.heappush(frontier, Node(new\_assign, score))
51. return None
52. def main():
53. print("🔍 Cryptarithmetic Solver using Best-First Search")
54. equation = input("Enter an equation (e.g., SEND + MORE = MONEY): ").strip()
55. solution = best\_first\_search(equation)
57. if solution:
58. print("\n✅ Solution Found:")
59. for k, v in sorted(solution.items()):
60. print(f"  {k} = {v}")
61. translated = substitute(equation.upper(), solution)
62. print(f"\n🔢 Equation: {translated}")
63. print(f"✔️ Verified: {eval(translated.split('=')[0])} = {eval(translated.split('=')[1])}")
64. else:
65. print("❌ No valid solution found.")
66. if \_\_name\_\_ == "\_\_main\_\_":
67. main()
68. **Output screenshots:**

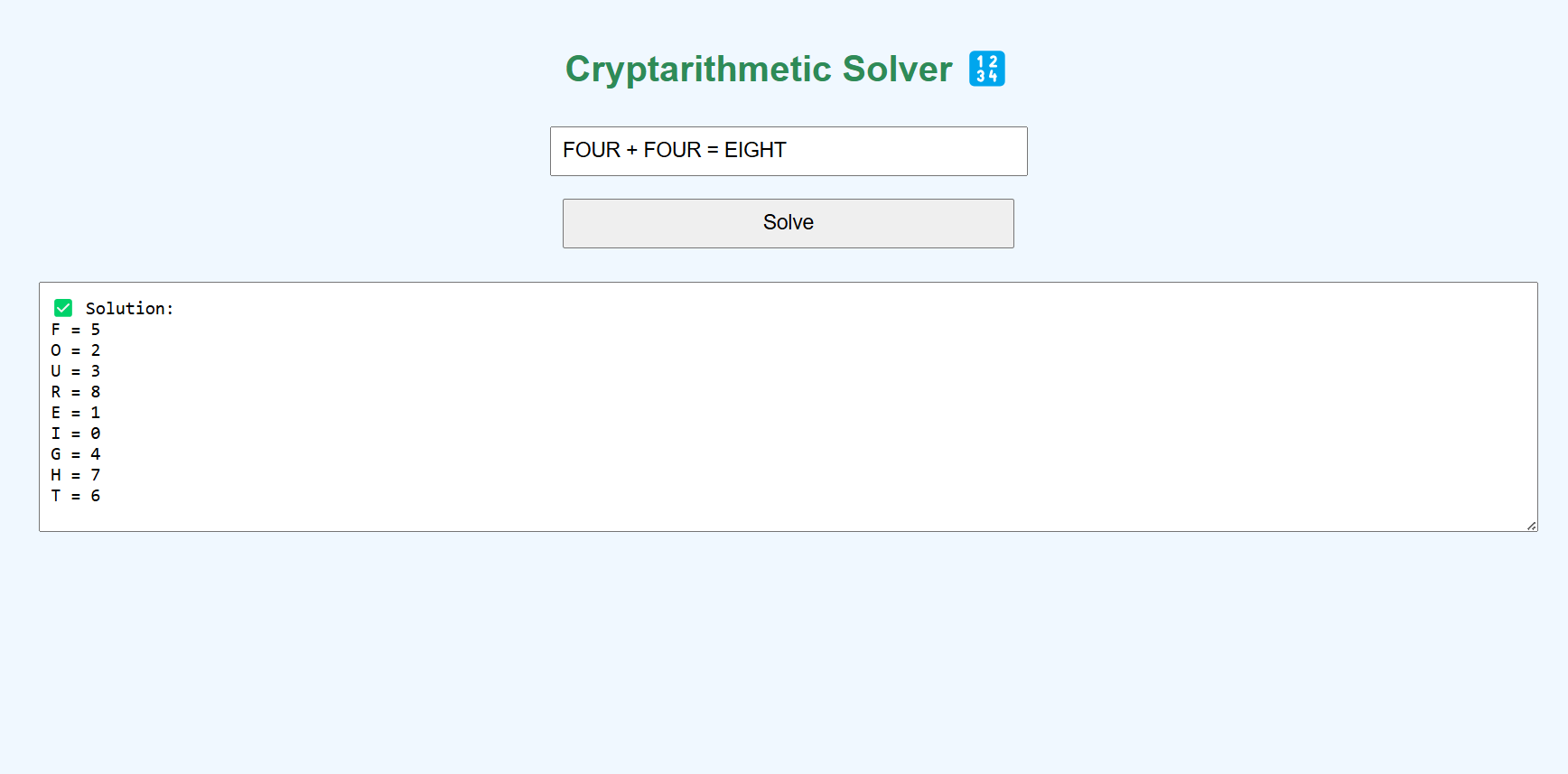


.

1. **Testing screenshots:**

****

****

****

**Observation and Conclusion**

* **The Best-First Search approach successfully solves cryptarithmetic problems with intelligent exploration of the search space.**
* **Heuristic guidance improves performance over brute-force methods.**
* **The solution is scalable and flexible for different equations.**
* **Further enhancement can include GUI for better usability and multiple-solution support.**