National University of Computer
and Emerging Sciences

Department: Data Science

Name of Assignment: ASSIGNMENT # 1

Name: Muhammad Talal

Section: “DS-U”

Subject: AI

Date of Submission: 09/03/2024

Introduction to Cryptarithmetic Puzzles

Objective: Cryptarithmetic puzzles, also known as alphametics, are a type of mathematical game or puzzle. In these puzzles, digits are replaced by letters of the alphabet. The primary objective is to crack the code: each letter represents a unique digit (from 0 to 9), and the goal is to find a consistent substitution that makes the given arithmetic equation true. These puzzles serve not only as entertaining brain teasers but also as tools for enhancing logical reasoning, mathematical skills, and problem-solving capabilities.

Problem Example: One classic example of a Cryptarithmetic puzzle is "SEND + MORE = MONEY". This puzzle is intriguing because it involves finding distinct digits for each letter in a way that the addition of "SEND" and "MORE" accurately sums up to "MONEY".

Problem Formulation

Description: Cryptarithmetic puzzles are formulated with letters substituting for digits in arithmetic equations. The challenge lies in identifying which digit each letter represents. A solution is valid only if the substitution leads to a correct arithmetic sum, with the added constraint that no two letters can represent the same digit. In essence, each puzzle is a test of deducing numerical values assigned to alphabetical symbols to resolve an arithmetic statement.

Constraints:

1. **Uniqueness Constraint:** Each letter must represent a different digit. This constraint ensures a one-to-one mapping between letters and digits.
2. **Leading Digit Constraint:** No equation part (either operand or the sum) can start with the digit 0. This reflects standard numerical notation where leading zeros are omitted.
3. **Arithmetic Integrity:** The substitution of letters with their corresponding digits must result in a valid arithmetic equation. The sum, when calculated, should adhere strictly to the principles of addition.

Conversion into a Search Problem

Solving Cryptarithmetic puzzles, such as "SEND + MORE = MONEY", can be elegantly formulated as a search problem. This formulation involves defining the state space, initial state, goal state, operators, and path cost. Here's a detailed breakdown:

State Space

The state space for a Cryptarithmetic puzzle is defined as all possible mappings of letters to digits (0 through 9). Given that each letter must correspond to a unique digit and considering that most Cryptarithmetic puzzles involve a subset of 10 or fewer distinct letters, the state space is finite but can be vast. For instance, with 10 unique letters, there are
10! (3,628,800) possible mappings, forming a large but searchable space.

Initial State

The initial state of our search problem is a condition where no letters have been assigned digits yet. This means we have a blank slate: we know the letters involved in the puzzle, but we have not yet started mapping them to digits. The process of solving the puzzle involves transitioning from this initial state to a goal state through a series of actions.

Goal State

The goal state is reached when a complete and valid mapping of letters to digits satisfies the given arithmetic equation. For "SEND + MORE = MONEY", this means finding a digit for each letter such that the sum
SEND+MORE=MONEY holds true. Achieving the goal state means successfully solving the puzzle.

Operators

Operators in this context are actions that assign digits to letters. Each action takes the current state (a partial or complete mapping) to a new state. An operator is valid if it assigns a digit to a letter that has not yet been assigned, and the assignment does not violate the constraint that each letter must represent a unique digit. The application of these operators expands the search tree, exploring the state space towards the goal state.

Path Cost

In the context of Cryptarithmetic puzzles, the path cost can be considered uniform for all paths. Each step (operator application) costs the same, as the primary objective is not to minimize the number of steps but rather to find any path that leads from the initial state to the goal state. Therefore, the cost of a path is simply the number of steps (assignments) taken from the initial state to reach the goal state, and all solutions are equally preferable if they satisfy the puzzle's constraints.

Algorithm used:

DFS & CSP.

Output:

```
Solution found:
Mapping: {'R': 8, 'O': 0, 'M': 1, 'D': 7, 'E': 5, 'S': 9, 'Y': 2, 'N': 6}
SEND + MORE = MONEY
9567 + 1085 = 10652
```

CSP vs. DFS for Cryptarithmetic Puzzles:

Time Complexity: Both CSP and DFS have worst-case O(n!) complexity, exploring all possible mappings of letters to digits.

Space Complexity: Both have worst-case O(n) space complexity, driven by storing state and recursion/stack depth.

Comparison: While both explore the same search space, CSP may offer more efficient pruning techniques, but both methods have similar complexities.

Using DFS:

Step 1: Problem Breakdown

- **Objective:** Assign digits to letters (S, E, N, D, M, O, R, Y) so that the sum SEND + MORE = MONEY holds true.
- **Constraints:** Each letter represents a unique digit (0-9), and leading letters (S, M) cannot be 0.

Step 2: Search Tree Initialization

- **Root Node:** Represents the initial state with no assignments: **{}**.
- The tree expands by assigning digits to letters, creating branches for each possibility.

Step 3: Assigning Digits and Expanding the Tree

Node 1: Assign 'M'

- **First Decision:** Start with 'M' because it's a leading digit in "MONEY" and its assignment affects the most significant digit of the sum.
- **Assignment:** Let's choose **M = 1** as a starting point because it cannot be 0 and assigning 1 to 'M' simplifies initial exploration. This decision creates our first node: **{M: 1}**.

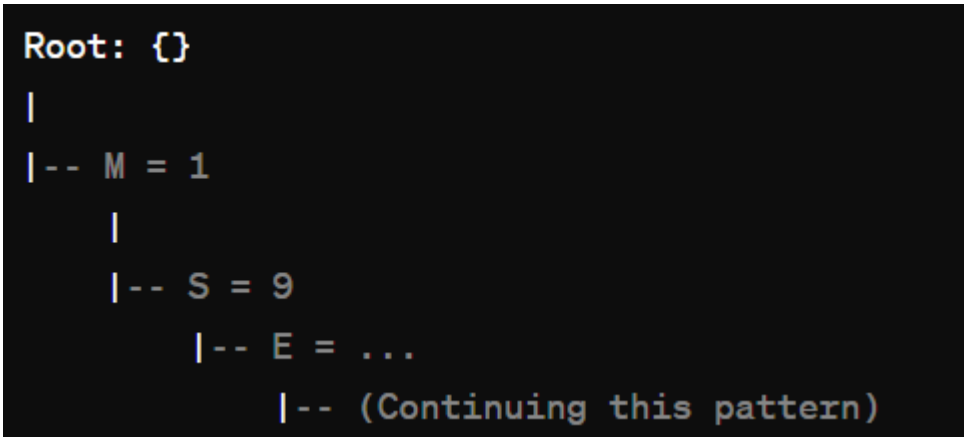
Node 2: Exploring 'S'

- **Next Letter:** Choose 'S' next, as it's also a leading digit and cannot be 0. We must choose a different digit from 'M'.
- **Branching:** For **{M: 1}**, we create branches for 'S' = 2 to 9 (since 'S' ≠ 0 and 'S' ≠ 'M').
- **Example Assignment:** Let's take one branch to explore, say **S = 9**, leading to the next node: **{M: 1, S: 9}**.

Further Expansion

- **Continue with Other Letters:** Following the same pattern, we would assign digits to 'E', 'N', 'D', 'O', 'R', and 'Y', each time creating branches for all possible digits not already assigned to other letters.
- **Depth-First Exploration:** At each level, choose a branch, dive deeper by assigning a digit to the next letter, and create nodes representing these decisions.

Visualization of Initial Branches (Simplified)



Step 4: Checking and Pruning

- **Arithmetic Check:** After several assignments, perform arithmetic checks to see if the partial sums align with the puzzle's requirements.
- **Pruning:** If a branch violates arithmetic rules or constraints, it's pruned. For example, if **SEND + MORE** does not partially match **MONEY** with the current assignments, backtrack and try a different digit for the last letter assigned.

Example Partial Solution Check

Let's assume we've progressed to **{M: 1, S: 9, E: 5, N: 6, D: 7}** and we're exploring options for 'O', 'R', and 'Y'. If we find that the current assignments lead to an incorrect partial sum (e.g., the units or tens place doesn't add up correctly when we try a certain 'O' or 'R' value), that indicates a need to backtrack and try different assignments.

Using CSP:

Initial Setup

1. **Variables:** S, E, N, D, M, O, R, Y
2. **Domains:** Each variable can initially take any digit from 0 to 9, with constraints:
    - $S \neq 0$, $M \neq 0$ (because they are leading digits)
    - All digits must be unique.
3. **Constraints:** The sum SEND + MORE = MONEY must hold true.

Step 1: Assign 'M'

- Assign **M = 1** to reduce the complexity right off the bat, respecting the constraint that M cannot be 0.
- **State:** M = 1
- **Domains Update:** S = {2-9}, E = {0-9}, N = {0-9}, D = {0-9}, O = {0-9}, R = {0-9}, Y = {0-9} (excluding 1 from all domains)

Step 2: Choose Next Variable and Assign

- Using the "Minimum Remaining Values" heuristic, we choose the next variable. Since 'S' and 'O' have reduced domains due to leading digit constraints, let's focus on 'S'.
- **Assignment:** Suppose we try **S = 9**.
- **State Update:** M = 1, S = 9
- **Domains Update:** Exclude 9 from the domains of other variables.

Step 3: Propagate Constraints and Further Assignments

- **Forward Checking:** For each assignment, we check if it leads to a domain of another variable being empty, which would indicate a conflict.
- Let's continue with arbitrary assignments for illustration, like **E = 5**, **N = 6**, considering constraints and updated domains.
- **State Update:** M = 1, S = 9, E = 5, N = 6

Step 4: Detecting Conflict and Backtracking

- **Conflict Example:** Assume we proceed to assign **D = 2** and **O = 0**, and later find that no valid assignment for 'R' and 'Y' can satisfy the puzzle (perhaps because the sum doesn't add up correctly or due to domain wipeout).
- **Action:** We backtrack. This might mean reconsidering the assignment to 'O' or 'D', or even going back further if necessary.

Step 5: Backtrack and Try Different Paths

- **Backtracking:** Suppose the conflict was with **O = 0**, we backtrack to try the next value in 'O's domain that hasn't been tried and doesn't cause immediate domain wipeout for other variables.
- **New Assignment:** Change **O** to another value, let's say **O = 8**, and then proceed to reassign 'R' and 'Y' based on the new state.

Step 6: Continue With Assignments and Constraint Propagation

- After adjusting 'O's value, we reassess 'R' and 'Y' under the new constraints, continually using forward checking and constraint propagation to guide our assignments.
- For example, now we might find that **R = 7** and **Y = 2** fit without causing any conflicts, satisfying all constraints including the arithmetic sum.

Final Checks and Solution Verification

- **Verification:** With the new set of assignments, verify the arithmetic operation SEND + MORE = MONEY holds true.
- **Solution Found:** If the sum matches, we've found a solution. If not, or if another domain wipeout occurs, continue backtracking and trying different assignments.