A **cryptarithmetic puzzle** is a math puzzle where: **Each letter stands for a unique digit** (0–9), and the goal is to make the arithmetic addition (or subtraction, etc.) correct.

**Step 1: Identify Letters**

**Step 2: Build Column Constraints**

**Step 3: Build the Constraint Graph**

1.

SEND  
+ MORE  
-------  
 MONEY

Each letter (S, E, N, D, M, O, R, Y) represents a digit.

* **Nodes**: letters
* **Edges**: between letters that are **in the same constraint**

## **Step 1: Column Constraints**

In puzzles like SEND + MORE = MONEY, **column constraints** come from **how addition works column by column**, from **right to left**, just like regular math.

|  |  |  |
| --- | --- | --- |
| **Column** | **Operation** | **Constraints per column** |
| 1 (units) | D + E = Y (+ possible carry) | D + E = Y + 10 \* C1 |
| 2 (tens) | N + R + carry = E | N + R + C1 = E + 10 \* C2 |
| 3 (hundreds) | E + O + carry = N | E + O + C2 = N + 10 \* C3 |
| 4 (thousands) | S + M + carry = O | S + M + C3 = O + 10 \* C4 |
| 5 (ten-thousands) | carry = M | C4 = M |

2.  **T W O**  
 **+ T W O**  
 **-------**  
 **F O U R**

**Step 1: Identify Letters**

### **Letters Involved:**

**L = {T, W, O, F, U, R}** → total **6 letters**

**Step 2: Build Column Constraints**

Let’s analyze the addition column by column (from right to left):

|  |  |  |
| --- | --- | --- |
| **Column** | **Expression** | **Addition Constraints** |
| 1 (units) | O + O = R (+ carry C1) | O + O = R + 10 \* X₁ |
| 2 (tens) | W + W + C1 = U (+ carry C2) | X₁ + W + W = U + 10 \* X₂ |
| 3 (hundreds) | T + T + C2 = O (+ carry C3) | X₂ + T + T = O + 10 \* X₃ |
| 4 (thousands) | C3 = F | X₃ = F |

**Also:** All letters must have unique digits (≠), and T and F ≠ 0 (leading digits)

### **Constraints:**

#### **1. Alldiff(F, T, U, W, R, O)**

* All these variables must take **unique digits**.

**Step 3: Build the Constraint Graph**

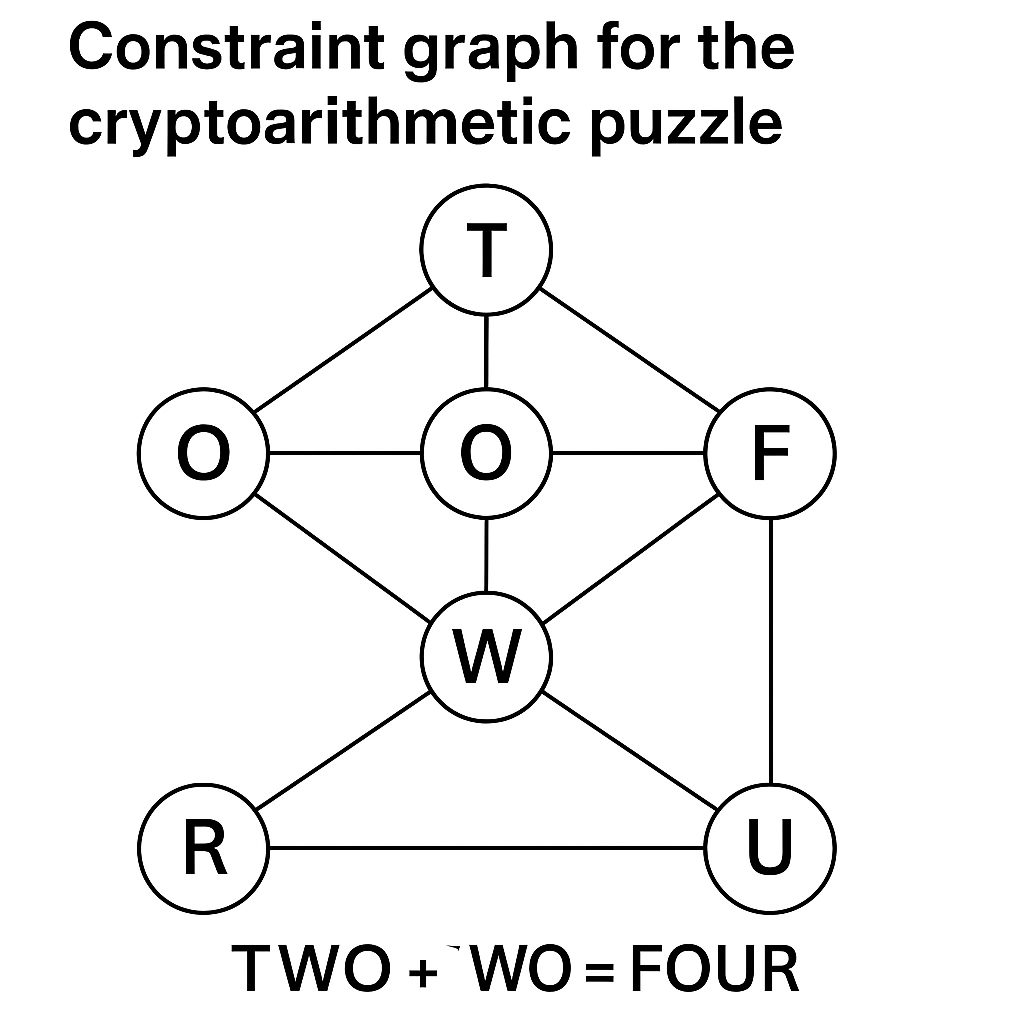
* **Each node** = one of the letters: {T, W, O, F, U, R}
* **Edges** connect letters that appear together in a **constraint equation**

### **Constraints (edges):**

1. O + O = R → edge between **O–R**
2. W + W + C1 = U → edges: **W–U**, **O–W** (because C1 depends on O)
3. T + T + C2 = O → edges: **T–O**, **W–T** (W affects C2)
4. C3 = F → edges: **T–F**, **W–F**, **O–F** (F indirectly depends on all others)

### **Final Set of Edges:**

* T–O
* T–F
* T–W
* W–U
* W–F
* O–R
* O–F
* O–W
* U–F
* R–F



3. **Map Coloring CSP problem:**

|  |  |
| --- | --- |
| **Step 1: Represent as a CSP** WA, NT, SA, Q, NSW, V, T  **1. Variables:**  X = {WA, NT, SA, Q, NSW, V, T}  **2. Domains:**  D = {Red, Green, Blue} for each variable  **3. Constraints (no same color for adjacent regions):**  WA ≠ NT  WA ≠ SA  NT ≠ SA  NT ≠ Q  SA ≠ Q  SA ≠ NSW  SA ≠ V  Q ≠ NSW  NSW ≠ V  ->T has no constraints since it's not adjacent to any other region. | A map of australia with states  AI-generated content may be incorrect. |

|  |  |
| --- | --- |
| **Step 2: Draw the Constraint Graph** **Nodes**: WA, NT, SA, Q, NSW, V, T  **Edges**: Between each pair of **adjacent** regions listed above |  |

## **Step 3: Apply Degree Heuristic**

**Degree Heuristic** = Choose the variable with the most constraints (most neighbors)

### **Number of neighbors:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Region** | **Neighbors** | **Degree** |  |
| WA | NT, SA | 2 |
| NT | WA, SA, Q | 3 |
| SA | WA, NT, Q, NSW, V | 5 ✅ (most) |
| Q | NT, SA, NSW | 3 |
| NSW | SA, Q, V | 3 |
| V | SA, NSW | 2 |
| T | — | 0 |

So the **first variable chosen = SA** (highest degree = 5)

## **Step 4: Solve Using Backtracking + MRV (Minimum Remaining Values)**

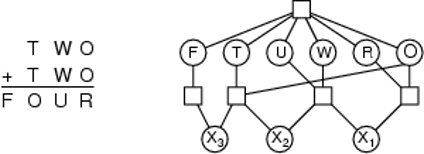
apply **backtracking** with **MRV heuristic**, which always selects the variable with the **fewest legal values** left in its domain.

|  |  |
| --- | --- |
| **1. Choose SA** (initial step by degree heuristic)   * Domain = {Red, Green, Blue} * Assign: **SA = Red**   **2. Choose MRV among neighbors**: **WA**, NT, Q, NSW, V  → All have 3 values initially, pick **WA**   * SA = Red → WA ≠ Red → Domain = **{Green, Blue}** * Assign: **WA = Green**   **3. Next MRV: NT**   * Adjacent to WA (Green) and SA (Red) → NT ≠ Green, ≠ Red → Domain = {Blue} * Assign: **NT = Blue**   **4. Next: Q**   * Adjacent to NT (Blue), SA (Red) → Q ≠ Blue, ≠ Red → Domain = {Green} * Assign: **Q = Green**   **5. Next: NSW**   * Adjacent to SA (Red), Q (Green) → NSW ≠ Red, ≠ Green → Domain = {Blue} * Assign: **NSW = Blue**   **6. Next: V**   * Adjacent to SA (Red), NSW (Blue) → V ≠ Red, ≠ Blue → Domain = {Green} * Assign: **V = Green**   **7. T has no constraints → any color**   * Assign: **T = Red** | A map of australia with states  AI-generated content may be incorrect.  A map of australia with states  AI-generated content may be incorrect. |

|  |  |
| --- | --- |
| **Region** | **Color** |
| SA | Red |
| WA | Green |
| NT | Blue |
| Q | Green |
| NSW | Blue |
| V | Green |
| T | Red |

-> **All constraints are satisfied!**

**4.**

  
 **Variables: T, W, O, F, U, R**

* **F, T, U, W, R, O** are the letters to assign digits to.
* **X₁, X₂, X₃** are **carry digits** from column addition (from right to left).

### **Domains:**

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

* Every letter and carry digit can take values from 0 to 9.
* Except: **F ≠ 0**, **T ≠ 0** (because they are leading digits)

### **Constraints:**

#### **1. Alldiff(F, T, U, W, R, O)**

* All these variables must take **unique digits**.

### **Column-wise Addition Constraints (Right to Left):**

#### **Column 1 (Units):**

O + O = R + 10 \* X₁

* Two O’s being added must equal digit R plus any carry to next column.

#### **Column 2 (Tens):**

X₁ + W + W = U + 10 \* X₂

* Add carry from previous (X₁) and both W’s → should equal U + new carry X₂

#### **Column 3 (Hundreds):**

X₂ + T + T = O + 10 \* X₃

* Add carry from previous (X₂) and both T’s → should equal O + carry X₃

#### **Column 4 (Thousands):**

X₃ = F  
 The carry from the last column becomes the first digit F in the result.

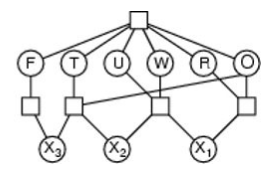
So **F = 1 or 2** (depending on carry)

**F ≠ 0**

### **Constraint:**

T ≠ 0, F ≠ 0

Because **T** and **F** are the **leading digits** of numbers (TWO and FOUR), and numbers don’t start with 0.



The graph shows:

* **Nodes** = variables (letters)
* **Edges** = constraints between them
* **Rectangles** = constraint functions (like addition or alldiff)
* Shows how X₁, X₂, and X₃ propagate through the structure of the addition