ARTIFICIAL INTELLIGENCE

UNIT 4: CONSTRAINT SATISFACTION PROBLEM (CSP)

PROF. JESLEENA GONSALVES

TOPICS

- Constraint satisfaction problems
- Backtracking search for CSPs
- Variables and value ordering
- Propagating information through constraints
- Intelligent backtracking
- Local search for CSP.
- Case study on CSP.

OVERVIEW

- Constraint Satisfaction Problem (CSP) is a fundamental topic in artificial intelligence that deals with solving problems by identifying constraints and finding solutions that satisfy those constraints.

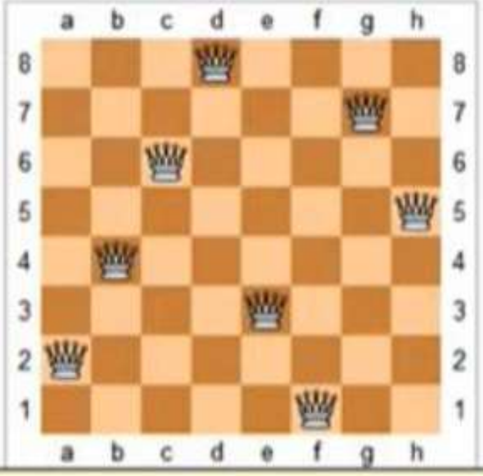
Introduction

1. The goal of AI is to create intelligent machines that can perform tasks that usually require human intelligence, such as reasoning, learning, and problem solving.
2. One of the key approaches in AI is the use of constraint satisfaction techniques to solve complex problems
3. CSP is a specific type of problem solving approach that involves identifying constraints that must be satisfied and finding a solution that satisfies all the constraints
4. CSP has been used in a variety of applications, including scheduling, planning, resource allocation, and automated reasoning

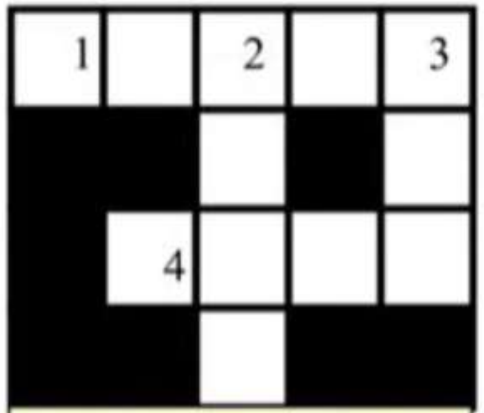
Constraint Satisfaction Problems (CSPs)

```
      B O B
    × B O B
    M E O Y
  M I L O
M E O Y
M A R L E Y
```

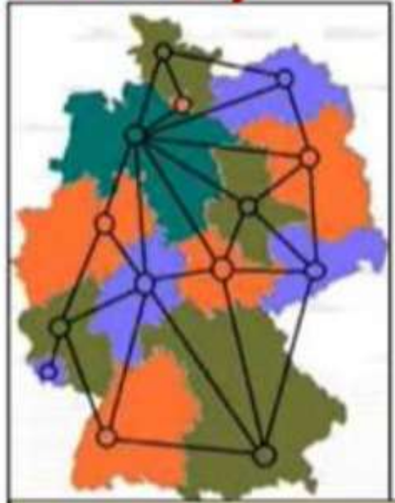
CRYPTARITHMETIC PUZZLE



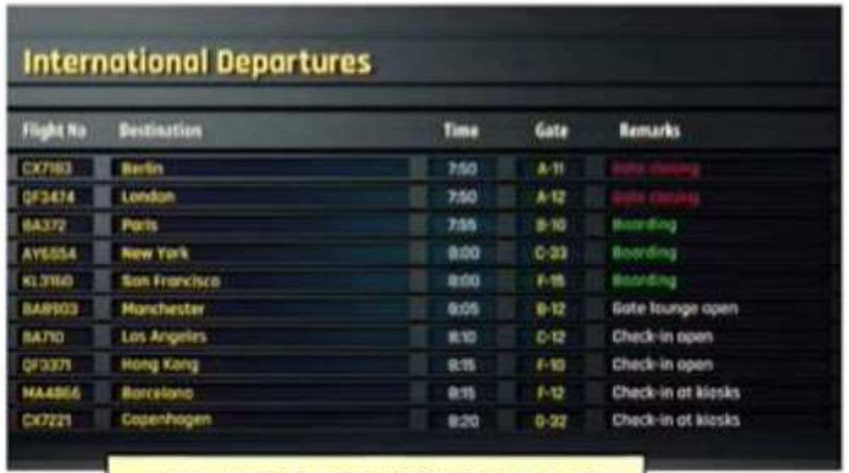
N-QUEENs



CROSSWORD PUZZLE



MAP COLOURING



International Departures

Flight No	Destination	Time	Gate	Remarks
CX7183	Berlin	7:50	A-11	Gate closing
QF3474	London	7:50	A-12	Gate closing
BA372	Paris	7:55	B-10	Boarding
AY6554	New York	8:00	C-33	Boarding
KL3160	San Francisco	8:00	F-15	Boarding
BA8903	Manchester	8:05	B-12	Gate lounge open
BA710	Los Angeles	8:10	C-12	Check-in open
QF3371	Hong Kong	8:15	F-10	Check-in open
MA4866	Barcelona	8:15	F-12	Check-in at kiosks
CX7221	Copenhagen	8:20	D-22	Check-in at kiosks

AIRLINE GATE SCHEDULING

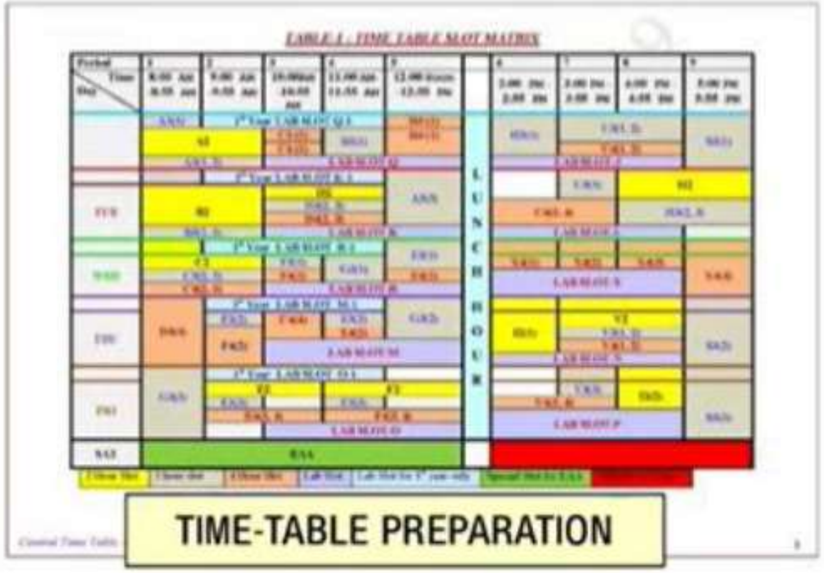
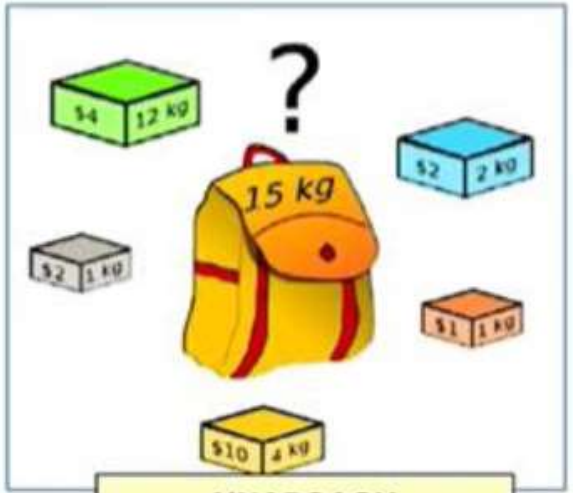


TABLE 1 : TIME TABLE SLOT MATRIX
LUNCH HOUR

TIME-TABLE PREPARATION



KNAPSACK

What is the Constraint Satisfaction Problem in AI

- In artificial intelligence and operations research, constraint satisfaction is the process of finding a solution to a set of constraints that impose conditions that the variables must satisfy
- CSP is defined by a set of variables X_1 , X_2 ….. $X_n$and set of conditions C_1, C_2,……. C_n such that each of the variables X_i has a non empty domain D_i of possible values
- Variables : A finite set of Variables X_1 , X_2 ….. X_n
- Domain: Each Variable has a domain D_1, D_2….. D_n from which it can take a value
- Constraints: A finite set of satisfaction constraints C_1, C_2,……. C_n
- For example, We need to color each country as Red, Green, Blue in such a way that no neighboring country has the same colors

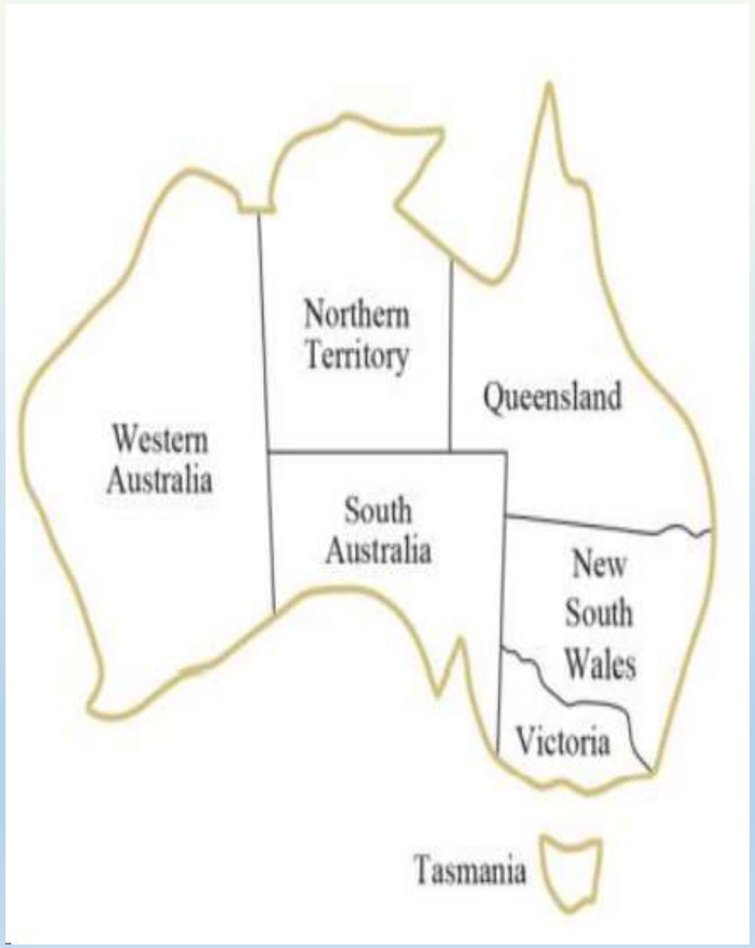
Types of Constraints in CSP

- Several types of constraints can be used in a Constraint satisfaction problem in artificial intelligence, including:

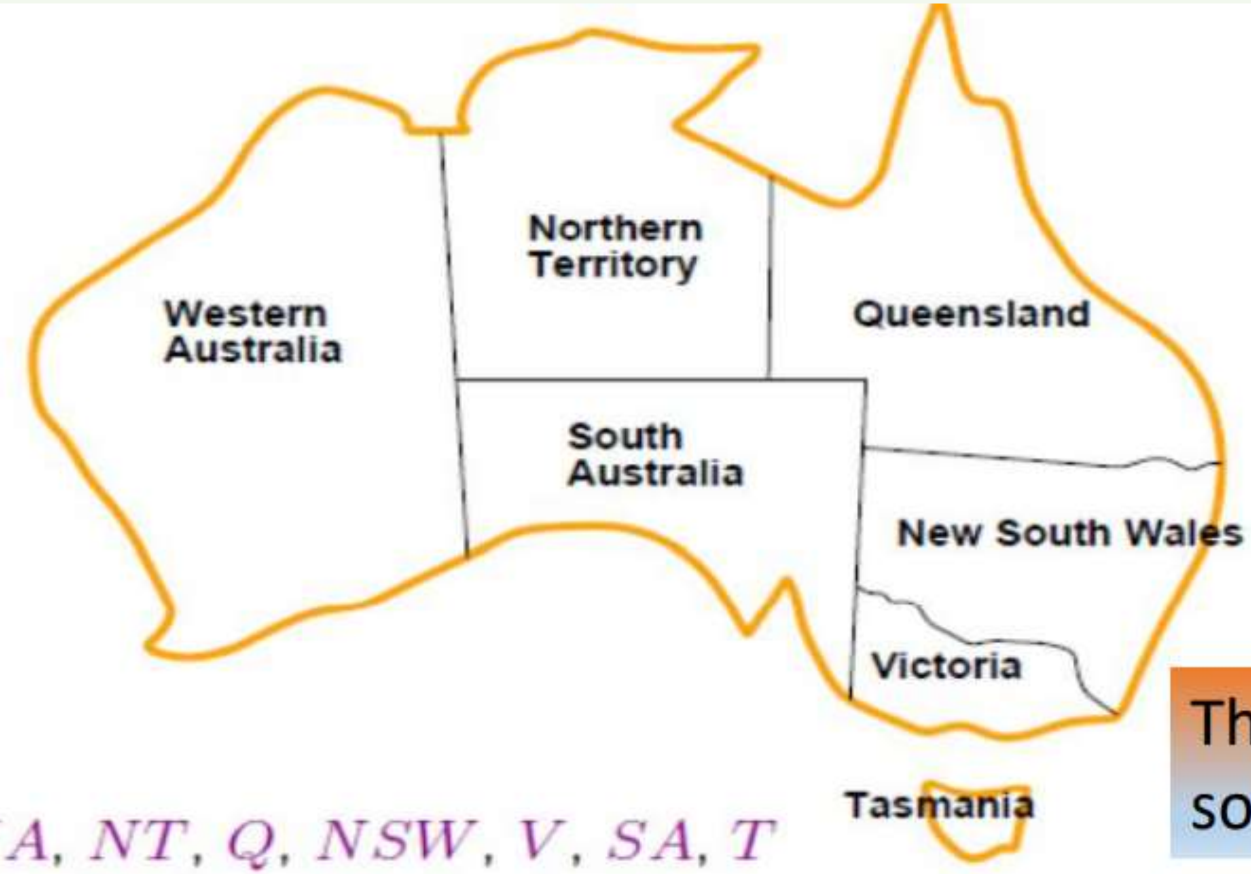
1. **Unary Constraints:** A unary constraint is a constraint on a single variable. For example, Variable A not equal to “Red”.
2. **Binary Constraints:** A binary constraint involves two variables and specifies a constraint on their values. For example, a constraint that two tasks cannot be scheduled at the same time would be a binary constraint.
3. **Global Constraints:** Global constraints involve more than two variables and specify complex relationships between them. For example, a constraint that no two tasks can be scheduled at the same time if they require the same resource would be a global constraint

CSP Example- Map Coloring

- Map of Australia showing each of its states and territories
- Task - To color each region either red, green, or blue in such a way that no two neighboring regions have the same color
- To formulate this as a CSP
- Identify variables, domain and constraints



CSP Example- Map Coloring

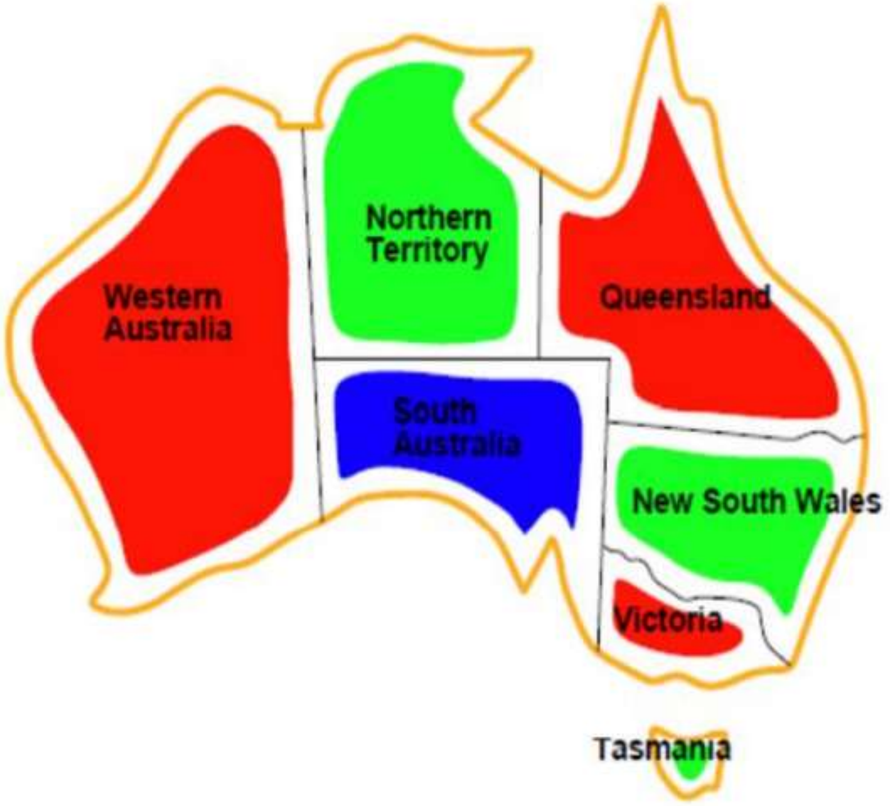


There are many possible solutions to this problem

Variables WA, NT, Q, NSW, V, SA, T
Domains $D_i = \{red, green, blue\}$
Constraints: adjacent regions must have different colors
e.g., $WA \neq NT$ (if the language allows this), or
$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

CSP Example- Map Coloring

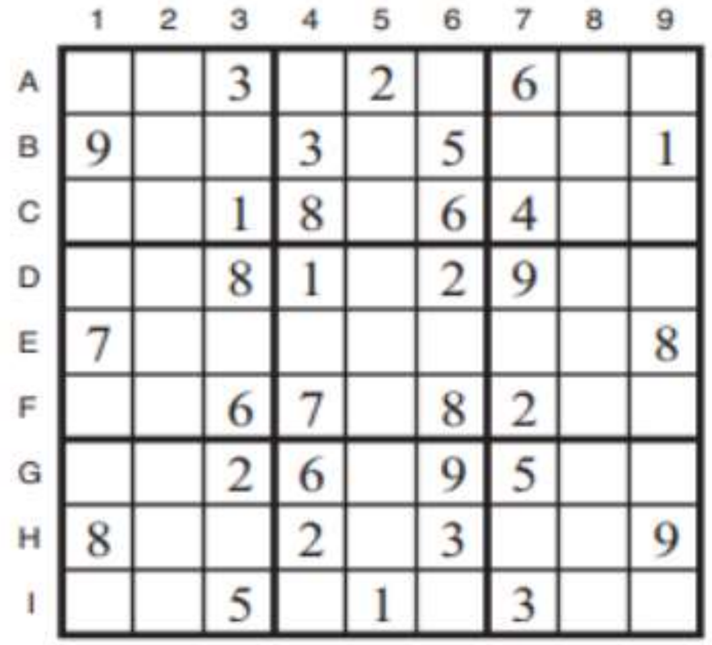
Map Coloring-One possible solution



Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

Formulating CSPs: Sudoku

- Identify Variables,Domain,Constraints in Sudoku problem



	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Variables: Cells

Domain: each variable is {1,2,3,4,5,6,7,8,9}

Constraints: rows,columns,boxes contain all different numbers

What Kinds of Algorithms are used for CSP?

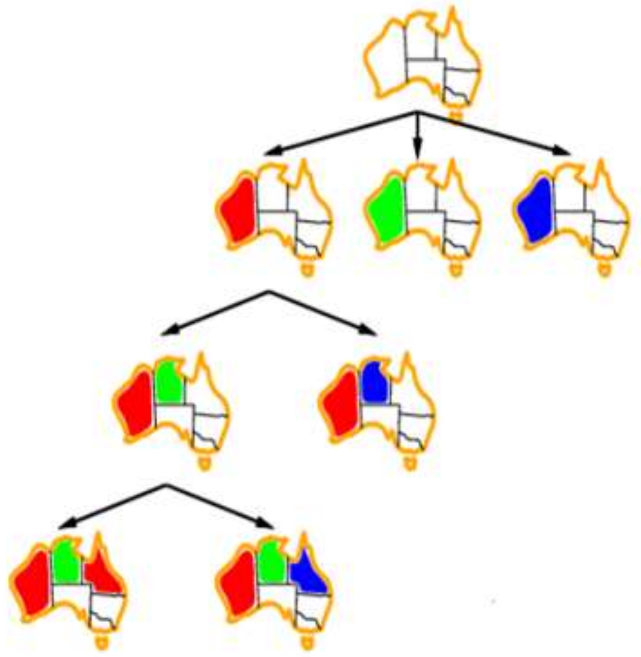
1. Backtracking Tree Search
2. Tree Search with Forward Checking
3. Tree Search with Discrete Relaxation (arc consistency)
4. Local Search using Complete State Formulation

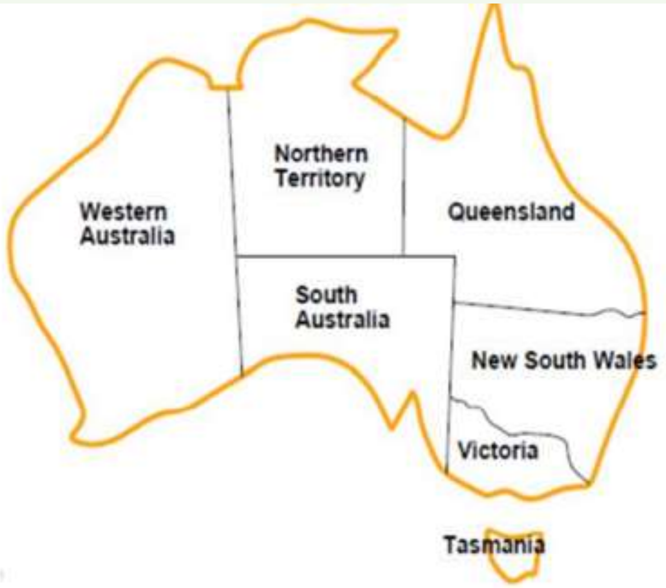
Backtracking Search in CSPs

- It chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.
- Variable assignments are commutative, i.e., [WA = red then NT = green] same as [NT = green then WA = red]
- Only need to consider assignments to a single variable at each node.
- Depth-first search for CSPs with single-variable assignments is called backtracking search.

Backtracking Search in CSPs

Backtracking search example





Backtracking Search in CSPs

How to improve backtracking?

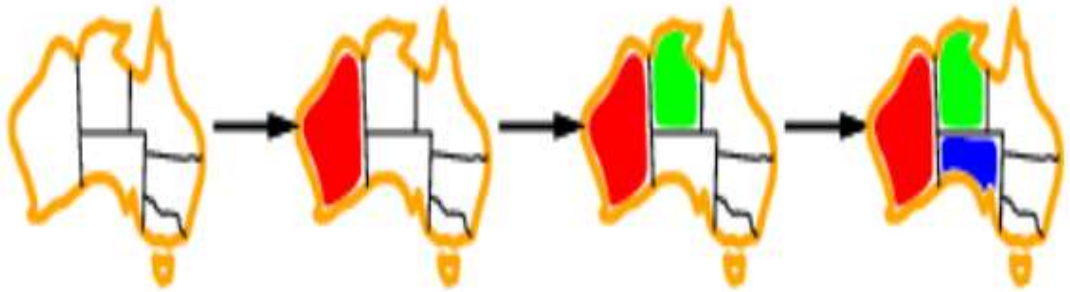
- Backtracking can be improved by using heuristics, such as choosing the most constrained variable or the least constraining value, to guide the search and reduce the number of backtracks.

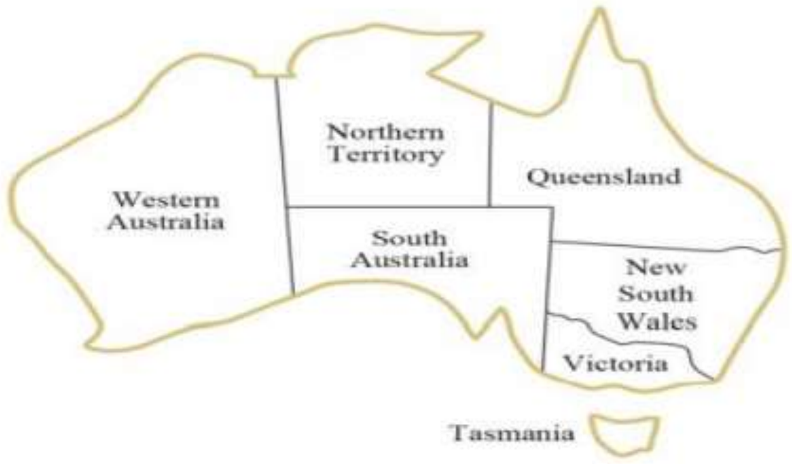
Backtracking improved by

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?

Most Constrained Variable

- Choose variables with fewest legal values



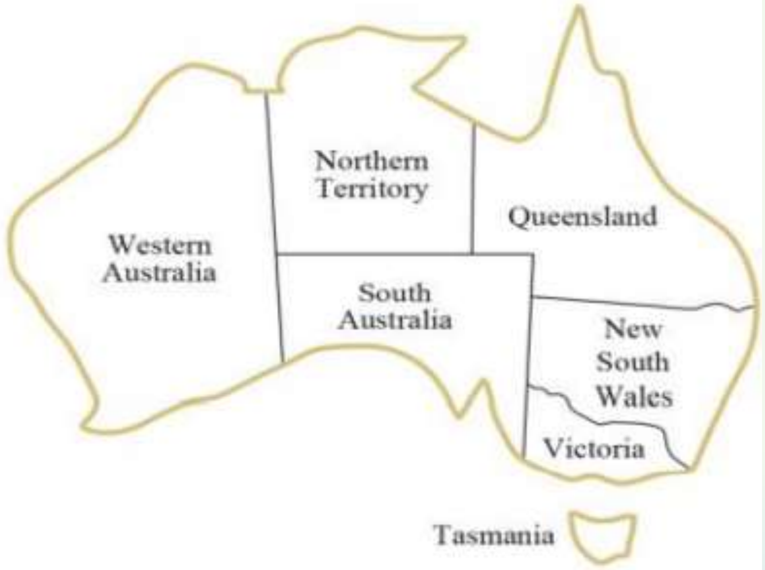


For example

- ❖ After the assignments for WA = red and NT = there is only one possible value for SA
- ❖ So it makes sense to assign SA = blue next rather than assigning Q
- ❖ In fact, after SA is assigned, the choices for Q, NSW , and V are all forced

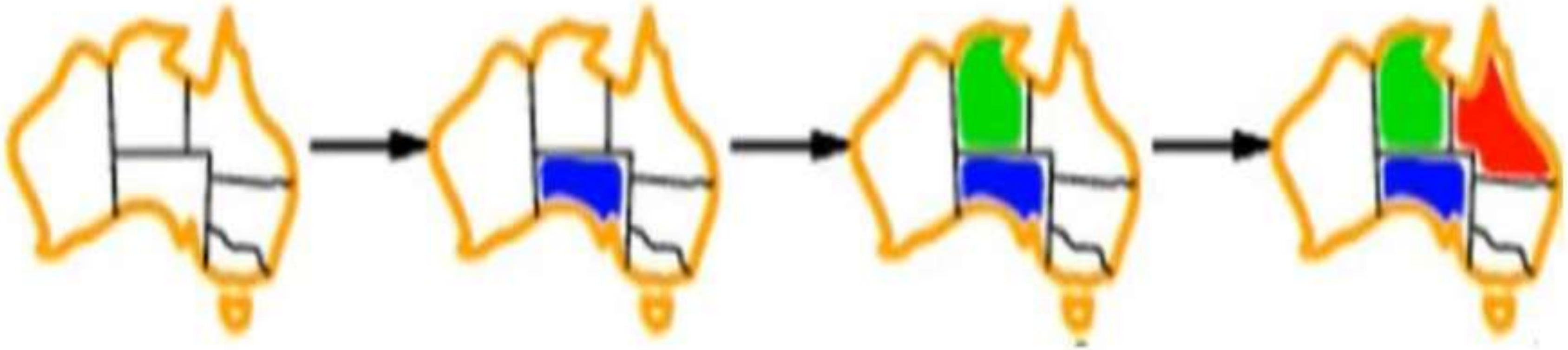
- ❑ **This intuitive idea of choosing the variable with the fewest “legal” values is called the Minimum Remaining Values (MRV) heuristic**
- ❑ **It also has been called the “most constrained variable” or “fail-first” heuristic**

Most constrained variable



❑A good idea is to use it as a tie-breaker among most constrained variables.

- **Most constrained variable:** **Choose the variable with the fewest legal values.**

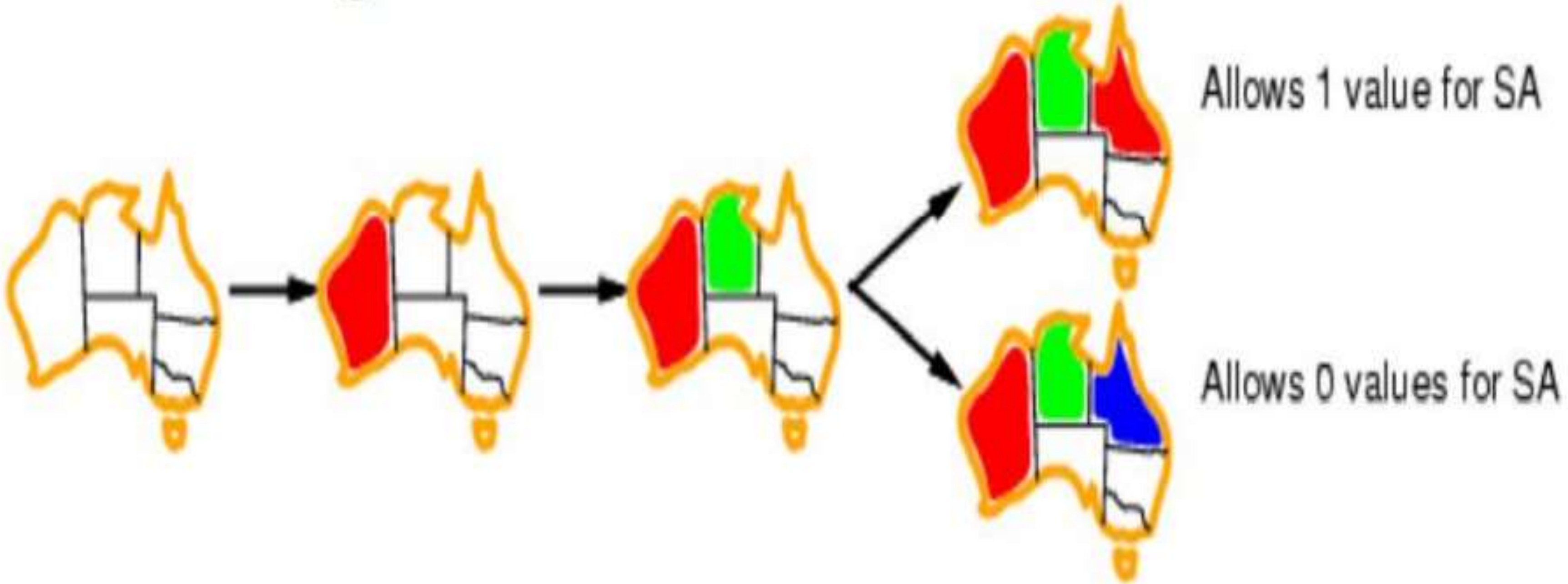


Least constraining value

Northern Territory
Queensland
Western Australia
South Australia
New South Wales
Victoria
Tasmania

- **Given a variable to assign the least constraining value:**

-The one that rules out the fewest values in the remaining variables.

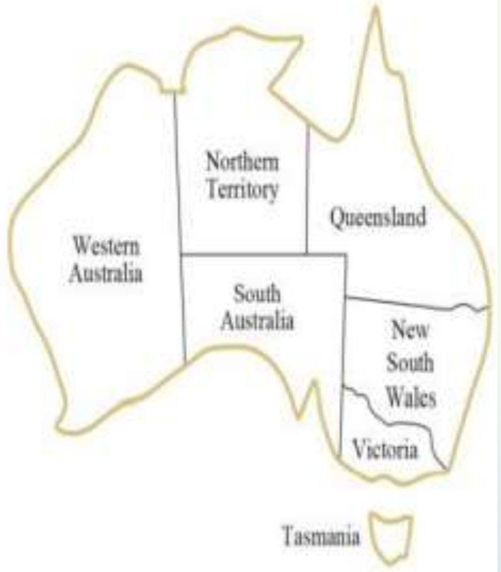


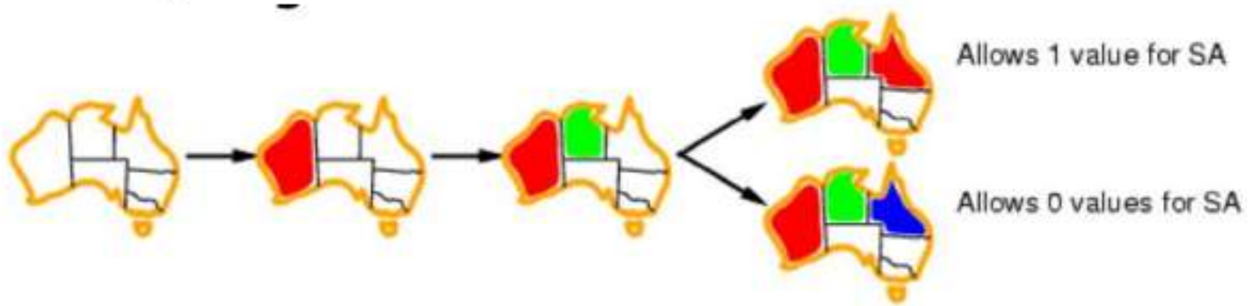
Least constraining value

- Once a variable has been selected, the algorithm must decide on the order in which to examine its values
- For this, the least-constraining-value heuristic can be effective in some cases.
- **It prefers the value that rules out the fewest choices for the neighboring variables in the constraint graph**

For example:

- We have generated the partial assignment with WA = red and NT = green and that our next choice is for Q
- Blue would be a bad choice because it eliminates the last legal value left for Q's neighbor, SA
- Least-constraining-value heuristic therefore prefers red to blue





Propagating information through constraints

- So far our search algorithm considers the constraints on a variable only at the time that the variable is chosen by SELECT-UNASSIGNED-VARIABLE.
- But by looking at some of the constraints earlier in the search, or even before the search has started, we can drastically reduce the search space.

Forward checking

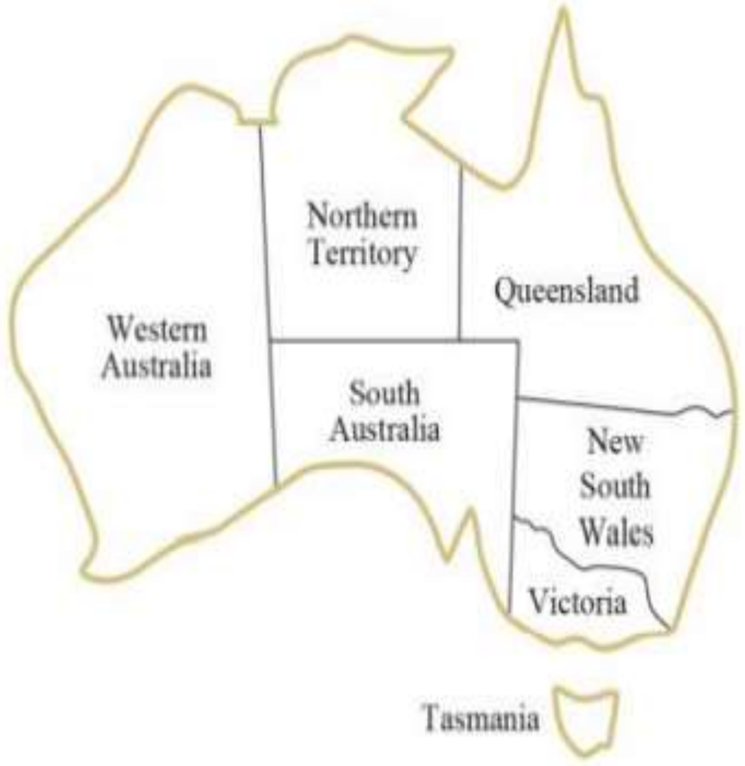
Idea:

- ❖ To assign a variable with some value and then check if it still allows remaining assignments to the future variables
- ❖ Keep a track of remaining legal values for unassigned variables
- ❖ Terminate search when any variable has no legal values

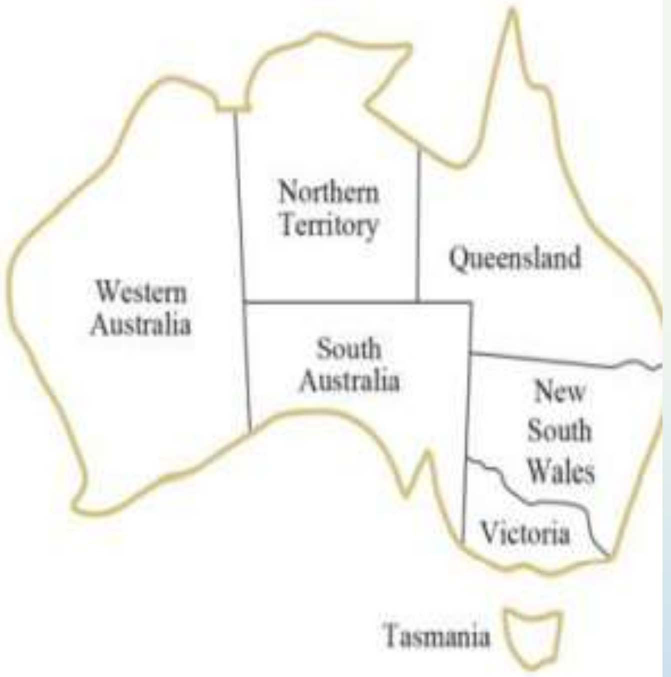
WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦

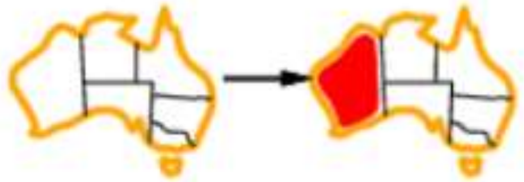
Forward checking example

	WA	NT	Q	NSW	V	SA
Initial domains	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦



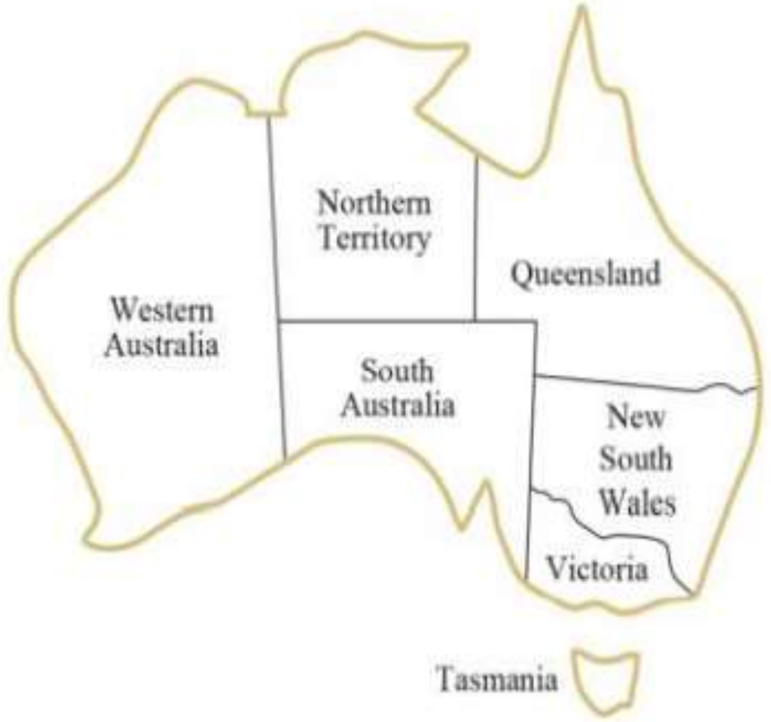
Forward checking example

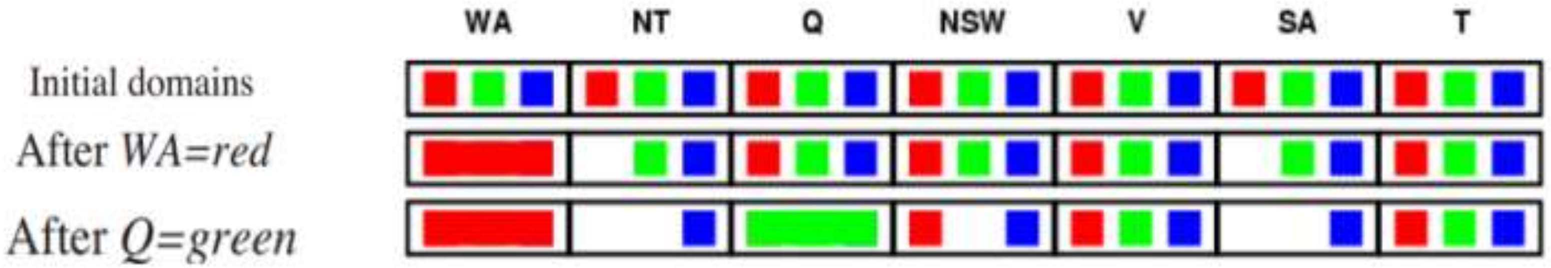




	WA	NT	Q	NSW	V	SA	T
Initial domains	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
After *WA=red*	🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦

Forward checking example

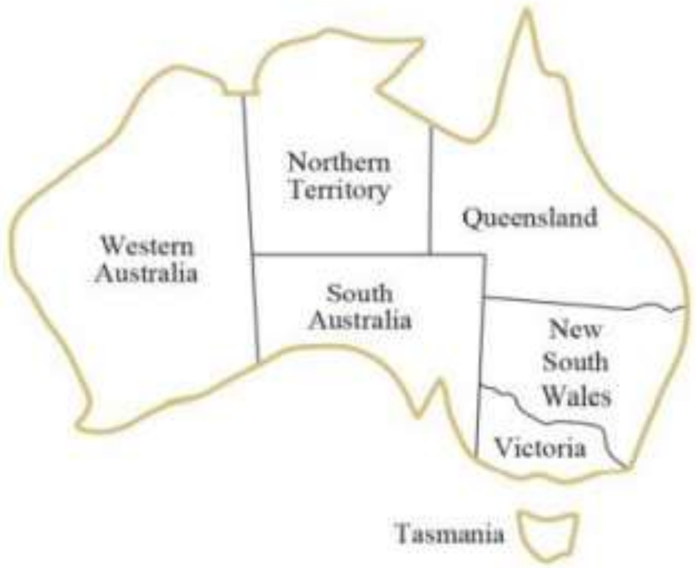


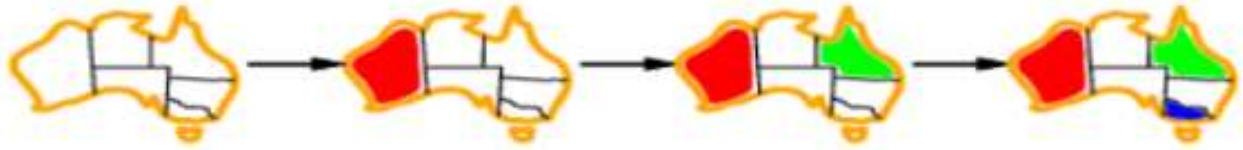


	WA	NT	Q	NSW	V	SA	T
Initial domains	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
After *WA=red*	🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
After *Q=green*	🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

Forward checking example

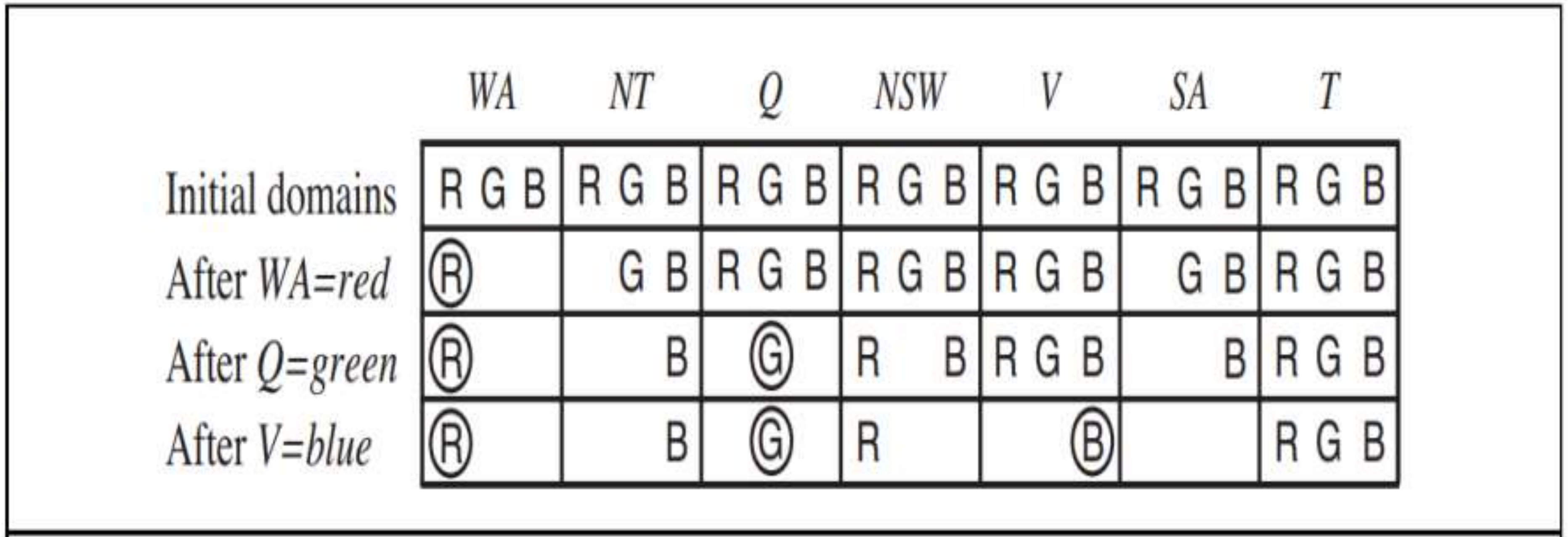
No assignment left for SA! Time to backtrack





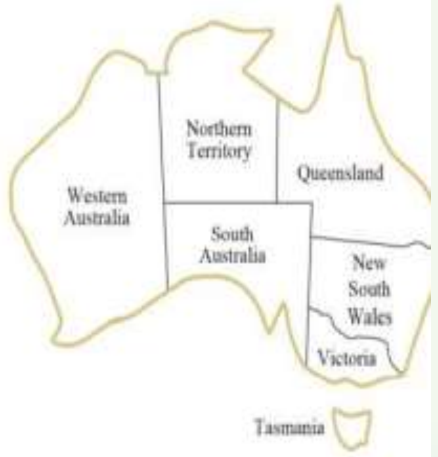
	WA	NT	Q	NSW	V	SA	T
Initial domains	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
After *WA=red*	🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
After *Q=green*	🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
After *V=blue*	🟥	🟦	🟩	🟥	🟦		🟥🟩🟦

Forward checking example



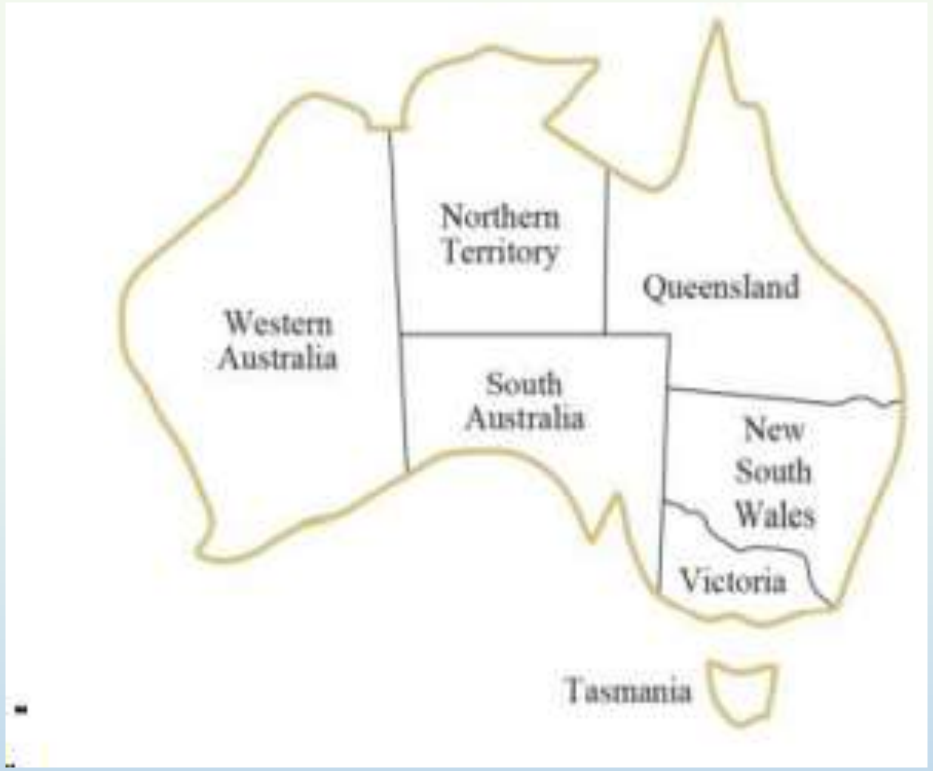
	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	Ⓡ	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	Ⓡ	B	Ⓖ	R B	R G B	B	R G B
After *V=blue*	Ⓡ	B	Ⓖ	R	Ⓑ		R G B

Figure 6.7 The progress of a map-coloring search with forward checking. $WA = red$ is assigned first; then forward checking deletes *red* from the domains of the neighboring variables NT and SA. After $Q = green$ is assigned, *green* is deleted from the domains of NT, SA, and NSW. After $V = blue$ is assigned, *blue* is deleted from the domains of NSW and SA, leaving SA with no legal values.



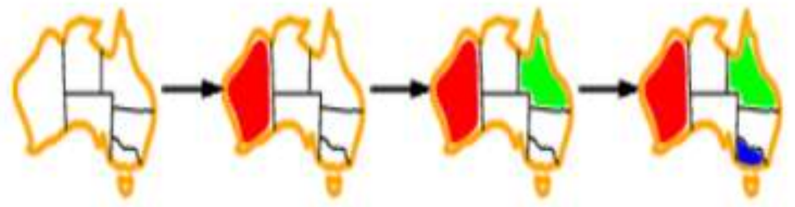
Forward Checking Example

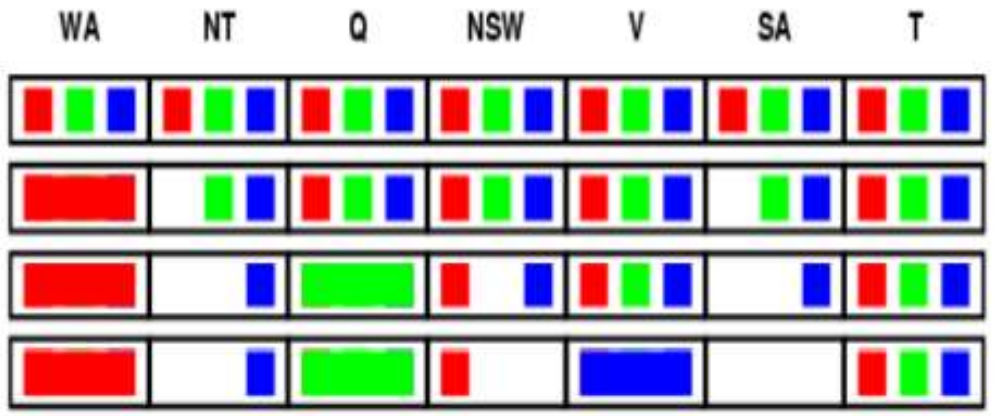
	WA	NT	SA	Q	NSW	V
Initial Domain	RGB	RGB	RGB	RGB	RGB	RGB
WA=B	B	RG	RG	RGB	RGB	RGB
NT=G	B	G	R	RB	RGB	RGB
SA=R	B	G	R	B	GB	GB
Q=R	B	G	R	B	G	GB
NSW=G	B	G	R	B	G	B
V=B	B	G	R	B	G	B



Summary of forward checking in the above example

- There are two important points to notice about this example
- First, notice that after WA = red and Q = green are assigned, the domains of NT and SA are reduced to a single value; we have eliminated branching on these variables altogether by propagating information from WA and Q
- A second point to notice is that after V = blue, the domain of SA is empty Hence, forward checking has detected that the partial assignment {WA = red, Q = green, V = blue} is inconsistent with the constraints of the problem, and the algorithm will therefore backtrack immediately

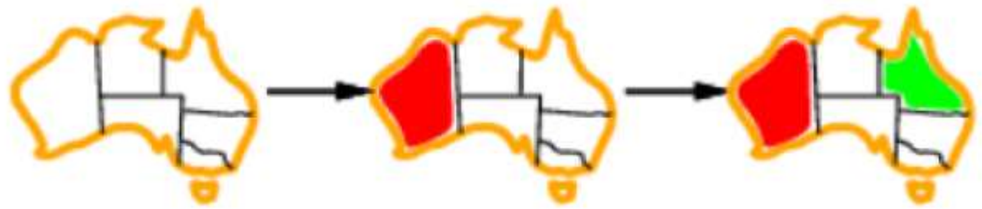




WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
🟥	🟦	🟩	🟥	🟦		🟥🟩🟦

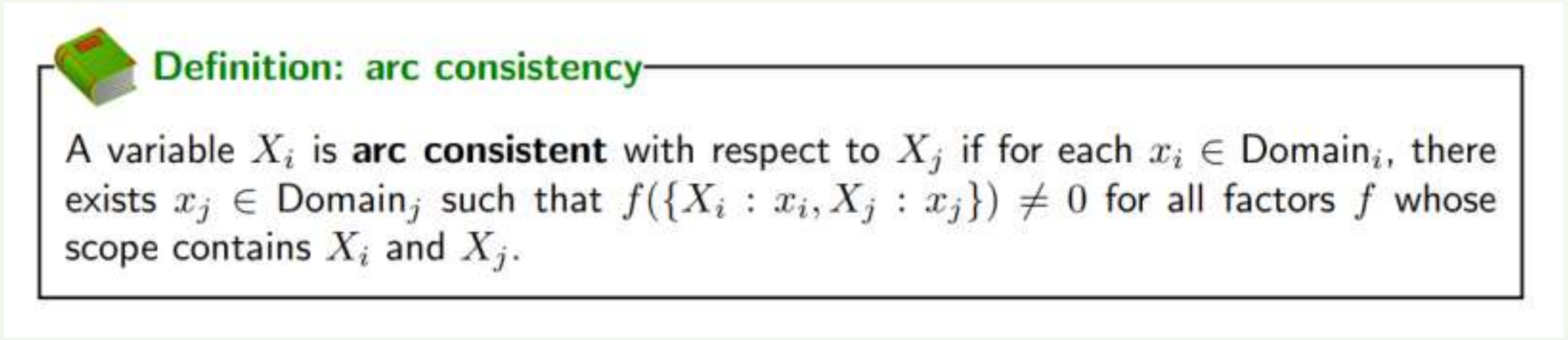
Problem with Forward checking

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures.
- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally.



WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

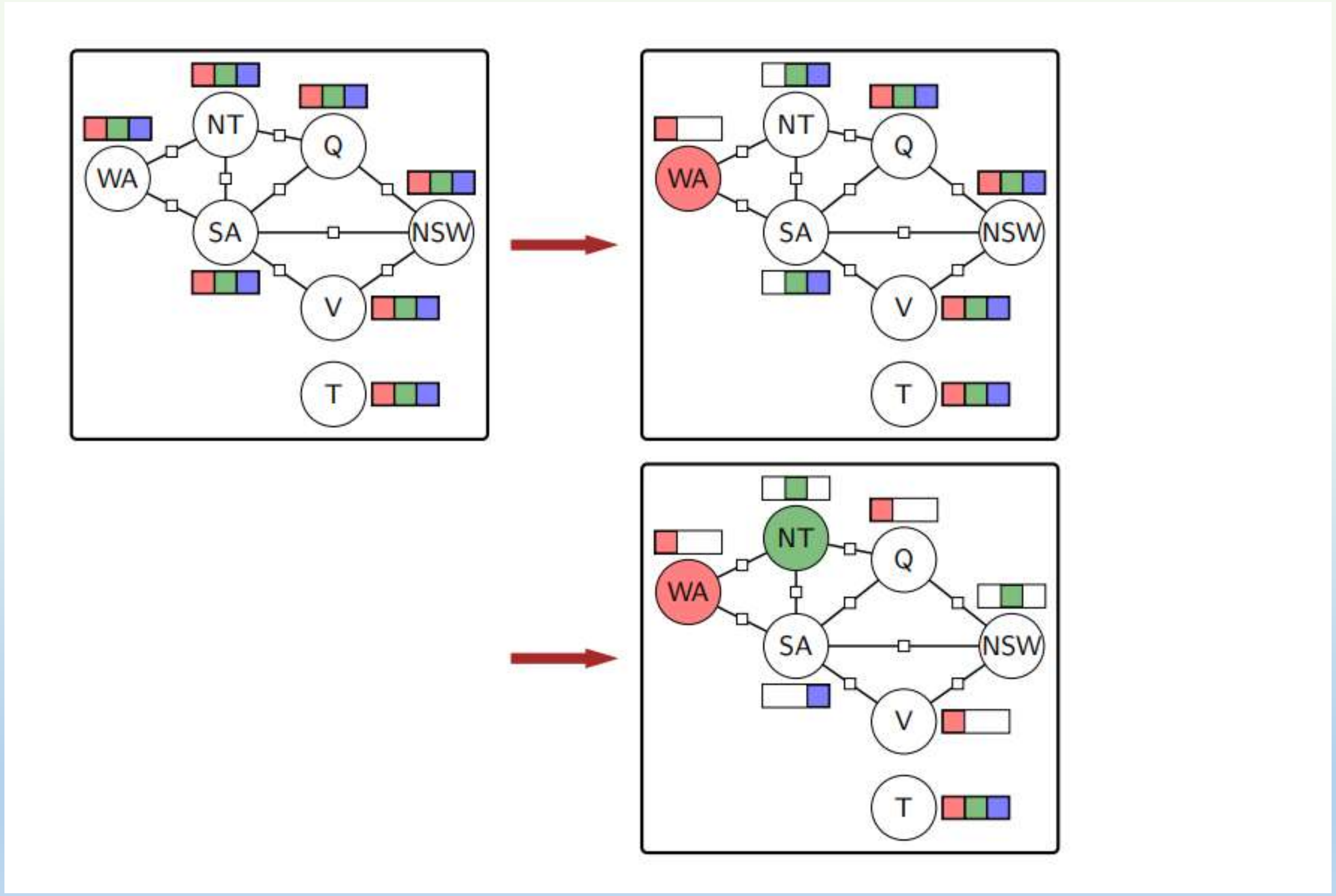
Arc Consistency


Definition: arc consistency

A variable X_i is **arc consistent** with respect to X_j if for each $x_i \in \text{Domain}_i$, there exists $x_j \in \text{Domain}_j$ such that $f(\{X_i : x_i, X_j : x_j\}) \neq 0$ for all factors f whose scope contains X_i and X_j.

- Arc consistency eliminates values from domain of variable that can never be part of a consistent solution.
- We can achieve consistency on arc by deleting values form Di (domain of variable at tail of constraint arc) that fail this condition.
- Arc consistency checking can be applied either as a preprocessing step before the beginning of the search process, or as a propagation step (like forward checking) after every assignment during search.
- In either case, the process must be applied repeatedly until no more inconsistencies remain.

Arc Consistency



Arc Consistency

- The AC algorithm simply enforces arc consistency until no domains change.
- Let’s walk through this example.
- We start with the empty assignment.
- Suppose we assign WA to R. Then we enforce arc consistency on the neighbors of WA.
- Those domains change, so we enforce arc consistency on their neighbors, but nothing changes.
- Note that at this point AC produces the same output as forward checking.
- We recurse and suppose we now pick NT and assign it G. Then we enforce arc consistency on the neighbors of NT (which is forward checking), SA, Q, NSW.
- AC converges at this point, but note how much progress we’ve made: we have nailed down the assignment for all the variables (except T)!
- Importantly, though we’ve touched all the variables, we are still at the NT node in the search tree.
- We’ve just paved the way, so that when we recurse, there’s only one value to try for SA, Q, NSW, and V!

INTELLIGENT BACKTRACKING

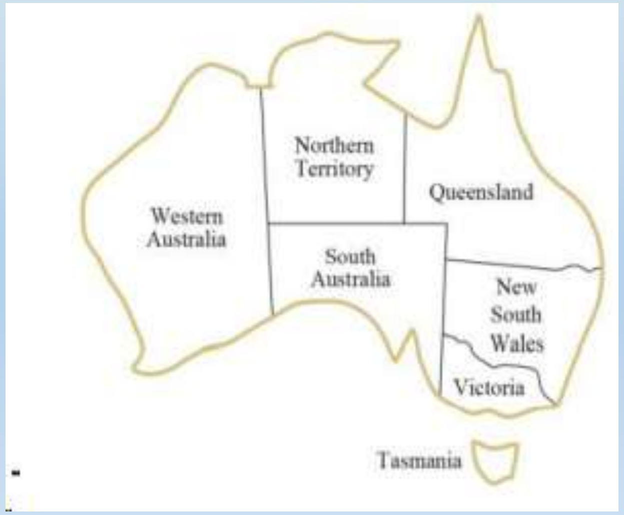
- The BACKTRACKING-SEARCH algorithm has a very simple policy for what to do when a branch of the search fails: back up to the preceding variable and try a different value for it.
- This is called chronological backtracking, because the *most recent* decision point is revisited.
- Consider what happens when we apply simple backtracking with a fixed variable ordering *Q*, NSW, *V*, *T*, SA, WA, NT.
- Suppose we have generated the partial assignment *{Q = red*, NSW = *green*, V = *blue*, *T = red)*.
- When we try the next variable, SA, we see that every value violates a constraint.
- We back up to T and try a new color for Tasmania! Obviously this is silly-recoloring Tasmania cannot resolve the problem with South Australia.
- A more intelligent approach to backtracking is to go all the way back to one of the set of variables that *caused the* ***failure.***
- This set is called the conflict set; here, the conflict set for SA is {*Q*, NSW, V).
- In general, the conflict set for variable X is the set of previously assigned

variables that are connected to *X* by constraints.

- The backjumping method backtracks to the *most recent* variable in the

conflict set; in this case, backjumping would jump over Tasmania and try a new

value for V.



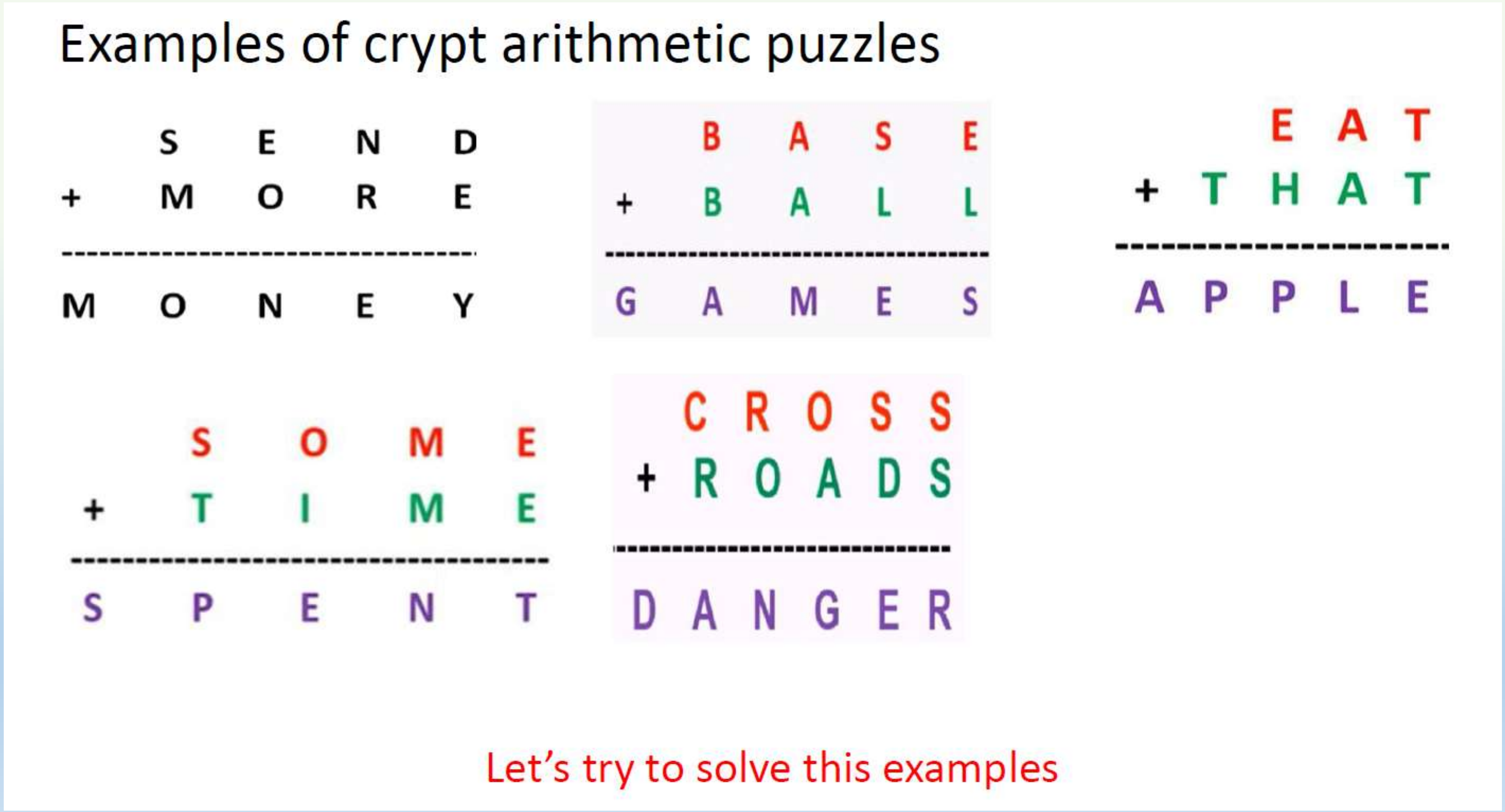
CRYPT ARITHMETIC

- Crypt arithmetic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacement either with alphabets or other symbols
- In crypt arithmetic problem, the digits 0 9 get substituted by some possible alphabets or symbols
- The task in crypt arithmetic problem is to substitute each digit with an alphabet to get the result arithmetically correct

CRYPT ARITHMETIC

- The rules or constraints on a crypt arithmetic problem are as follows
- There should be a unique digit to be replaced with a unique alphabet
- The result should satisfy the predefined arithmetic rules i e 2+2 = 4
- Digits should be from 0- 9 only
- There should be only one carry forward, while performing the addition operation on a problem
- The problem can be solved from both sides, i e left hand side (L H S), or right hand side (R H S)

CRYPT ARITHMETIC



Examples of crypt arithmetic puzzles

```
    S  E  N  D
+   M  O  R  E
--------------
 M  O  N  E  Y
```

```
    B  A  S  E
+   B  A  L  L
--------------
 G  A  M  E  S
```

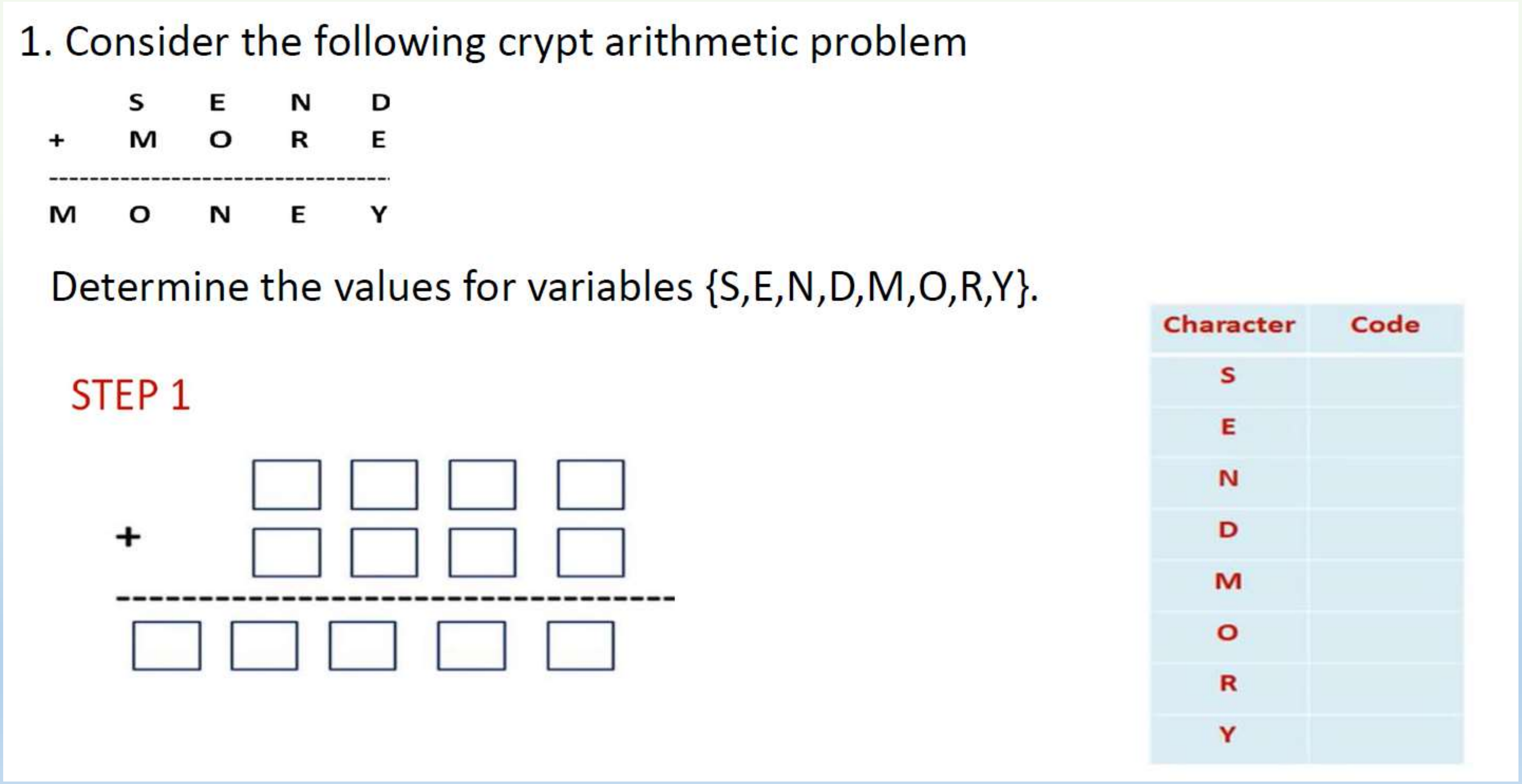
```
       E  A  T
+   T  H  A  T
--------------
 A  P  P  L  E
```

```
    S  O  M  E
+   T  I  M  E
--------------
 S  P  E  N  T
```

```
    C  R  O  S  S
+   R  O  A  D  S
-----------------
 D  A  N  G  E  R
```

Let's try to solve this examples

CRYPT ARITHMETIC



1. Consider the following crypt arithmetic problem

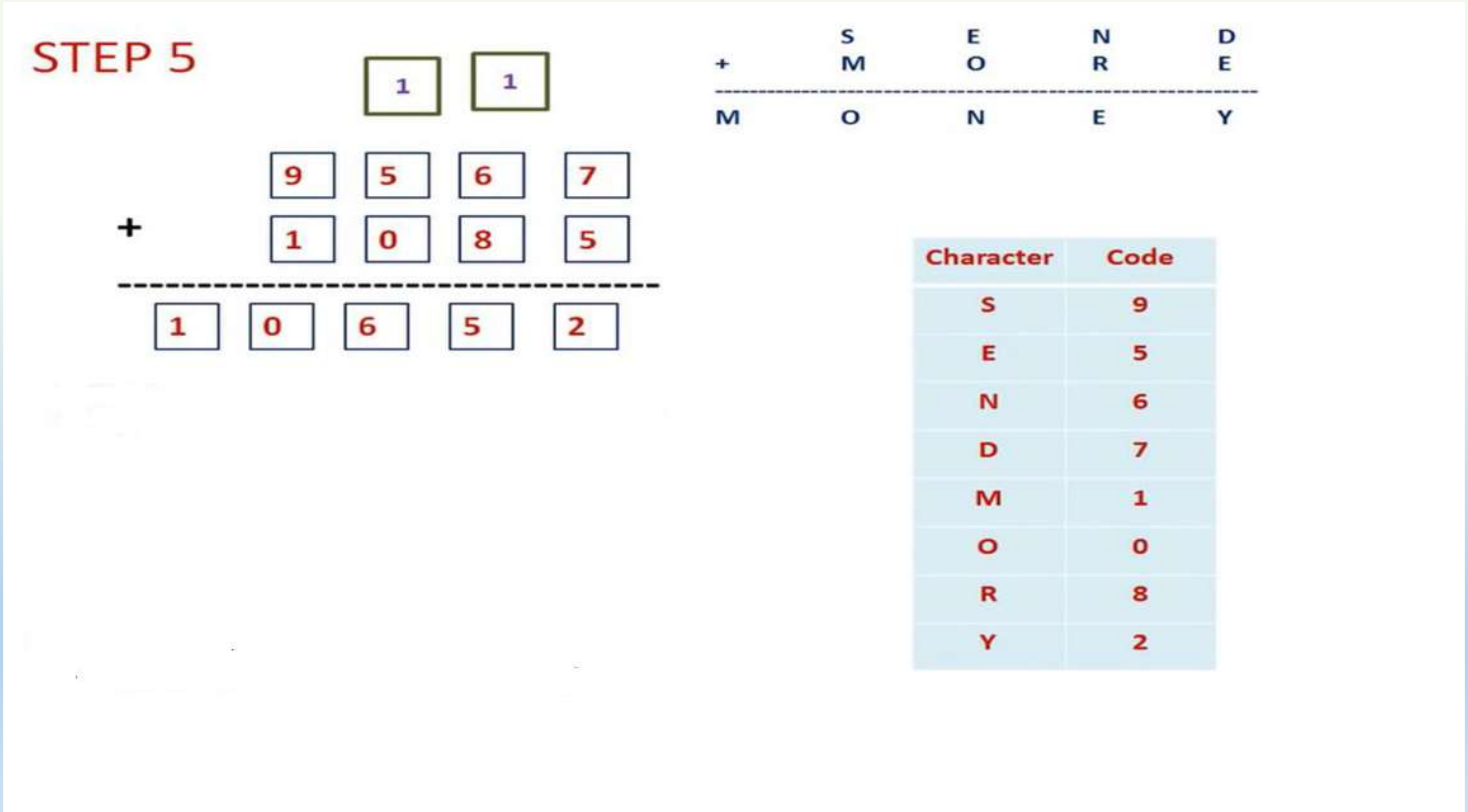
```
      S   E   N   D
+     M   O   R   E
-------------------
M     O   N   E   Y
```

Determine the values for variables {S,E,N,D,M,O,R,Y}.

STEP 1

Character	Code
S	
E	
N	
D	
M	
O	
R	
Y	

CRYPT ARITHMETIC



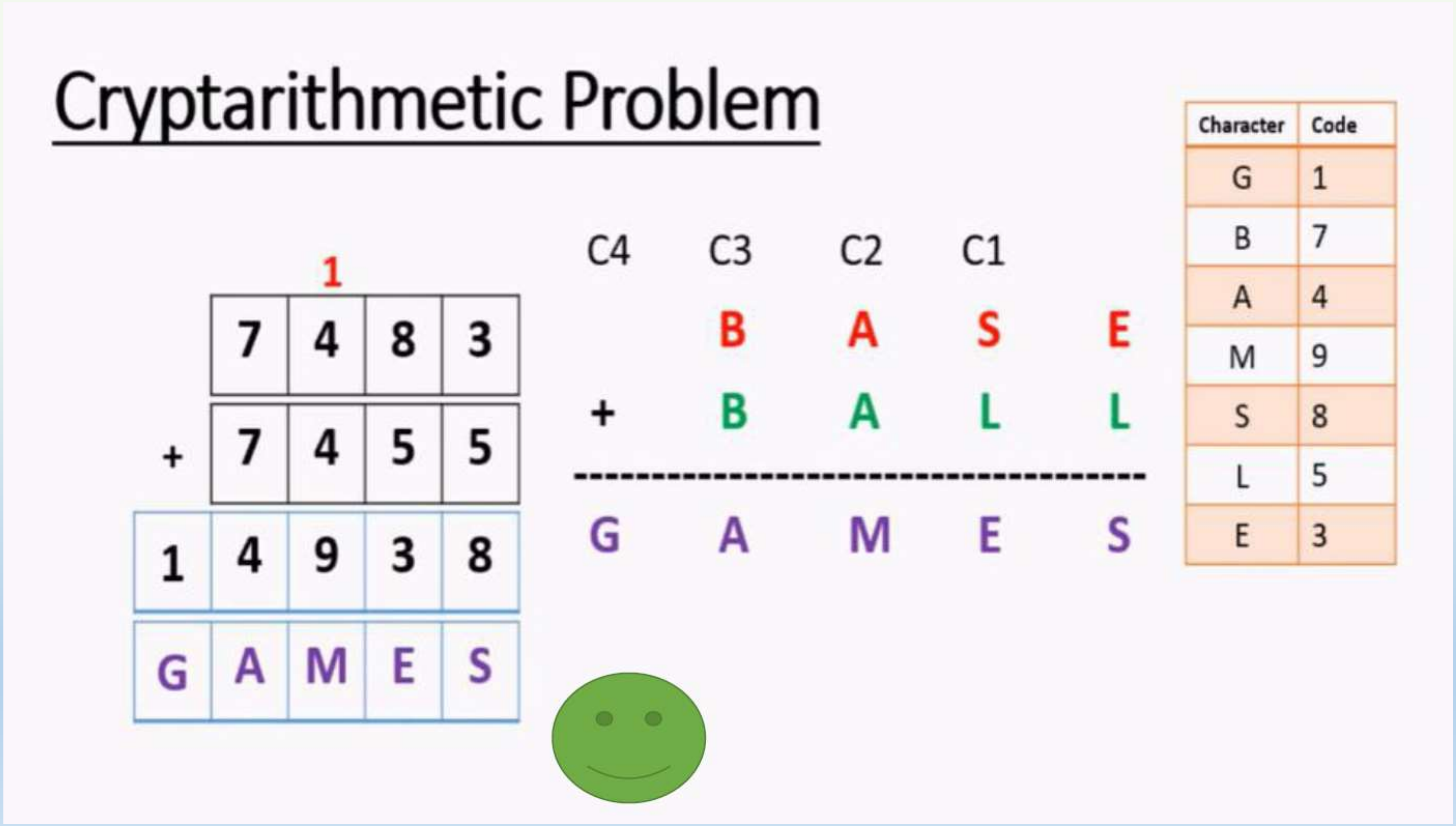
STEP 5

$$\begin{array}{cccccc} & & & \boxed{1} & \boxed{1} & \\ & & \boxed{9} & \boxed{5} & \boxed{6} & \boxed{7} \\ + & & \boxed{1} & \boxed{0} & \boxed{8} & \boxed{5} \\ \hline & \boxed{1} & \boxed{0} & \boxed{6} & \boxed{5} & \boxed{2} \end{array}$$

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline & M & O & N & E & Y \end{array}$$

Character	Code
S	9
E	5
N	6
D	7
M	1
O	0
R	8
Y	2

CRYPT ARITHMETIC



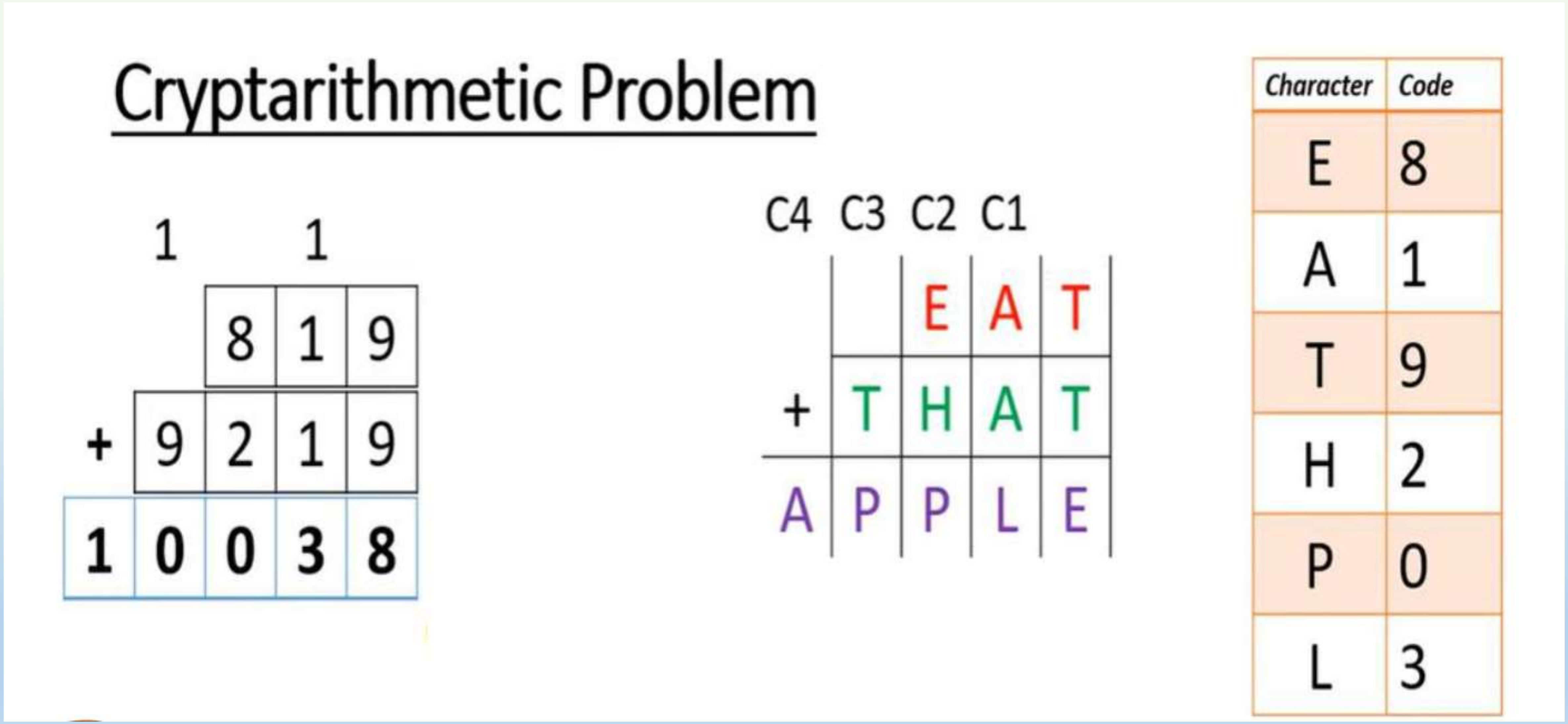
Cryptarithmetic Problem

$$\begin{array}{ccccc} & & 1 & & \\ & 7 & 4 & 8 & 3 \\ + & 7 & 4 & 5 & 5 \\ \hline 1 & 4 & 9 & 3 & 8 \\ G & A & M & E & S \end{array}$$

$$\begin{array}{ccccc} C4 & C3 & C2 & C1 & \\ & B & A & S & E \\ + & B & A & L & L \\ \hline G & A & M & E & S \end{array}$$

Character	Code
G	1
B	7
A	4
M	9
S	8
L	5
E	3

CRYPT ARITHMETIC



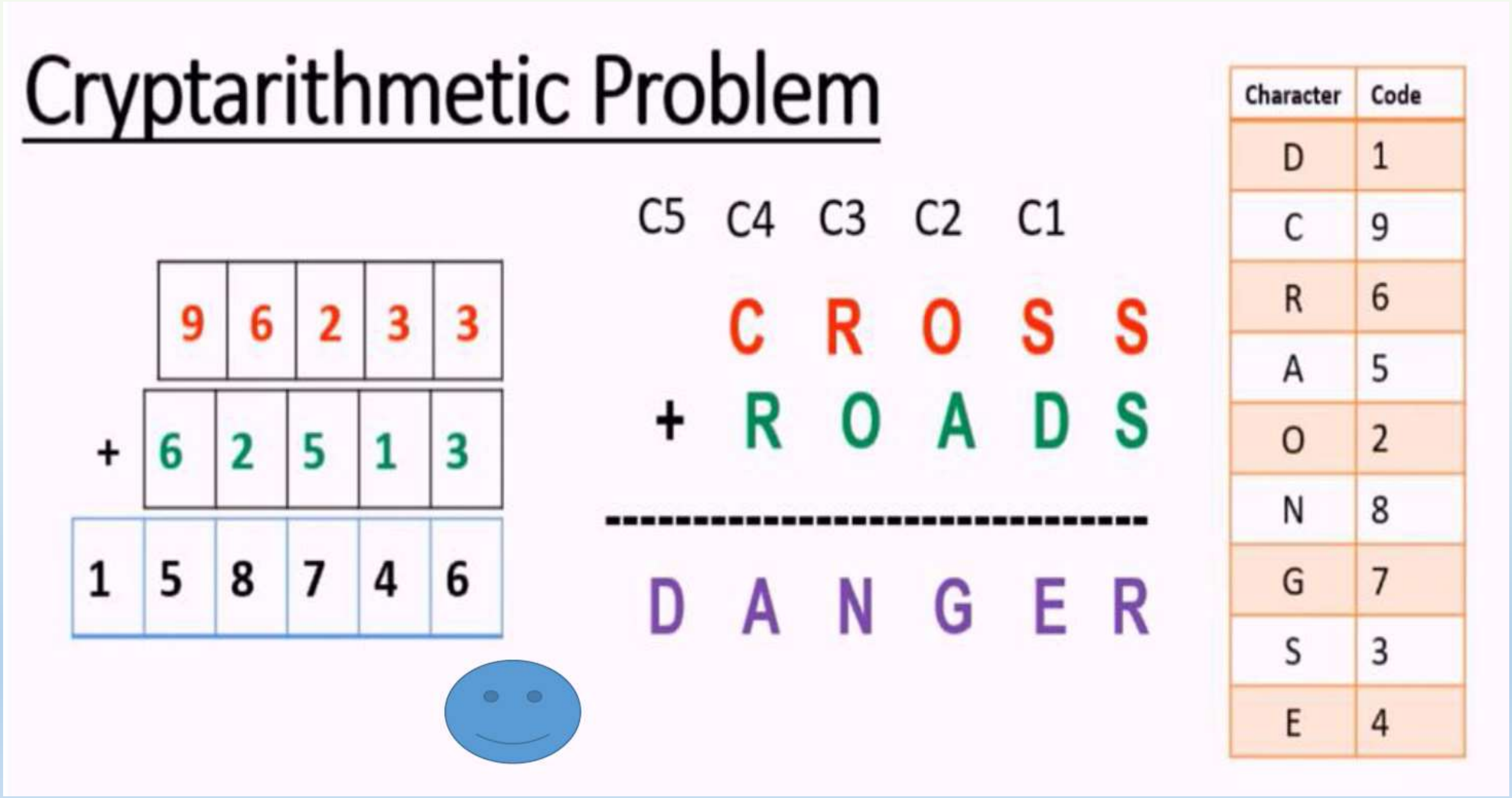
Cryptarithmetic Problem

$$\begin{array}{cccccc} & & 1 & & 1 & \\ & & & 8 & 1 & 9 \\ + & & 9 & 2 & 1 & 9 \\ \hline & 1 & 0 & 0 & 3 & 8 \end{array}$$

$$\begin{array}{c|c|c|c|c|} C4 & C3 & C2 & C1 & \\ & & E & A & T \\ \hline + & T & H & A & T \\ \hline A & P & P & L & E \end{array}$$

Character	Code
E	8
A	1
T	9
H	2
P	0
L	3

CRYPT ARITHMETIC



Cryptarithmetic Problem

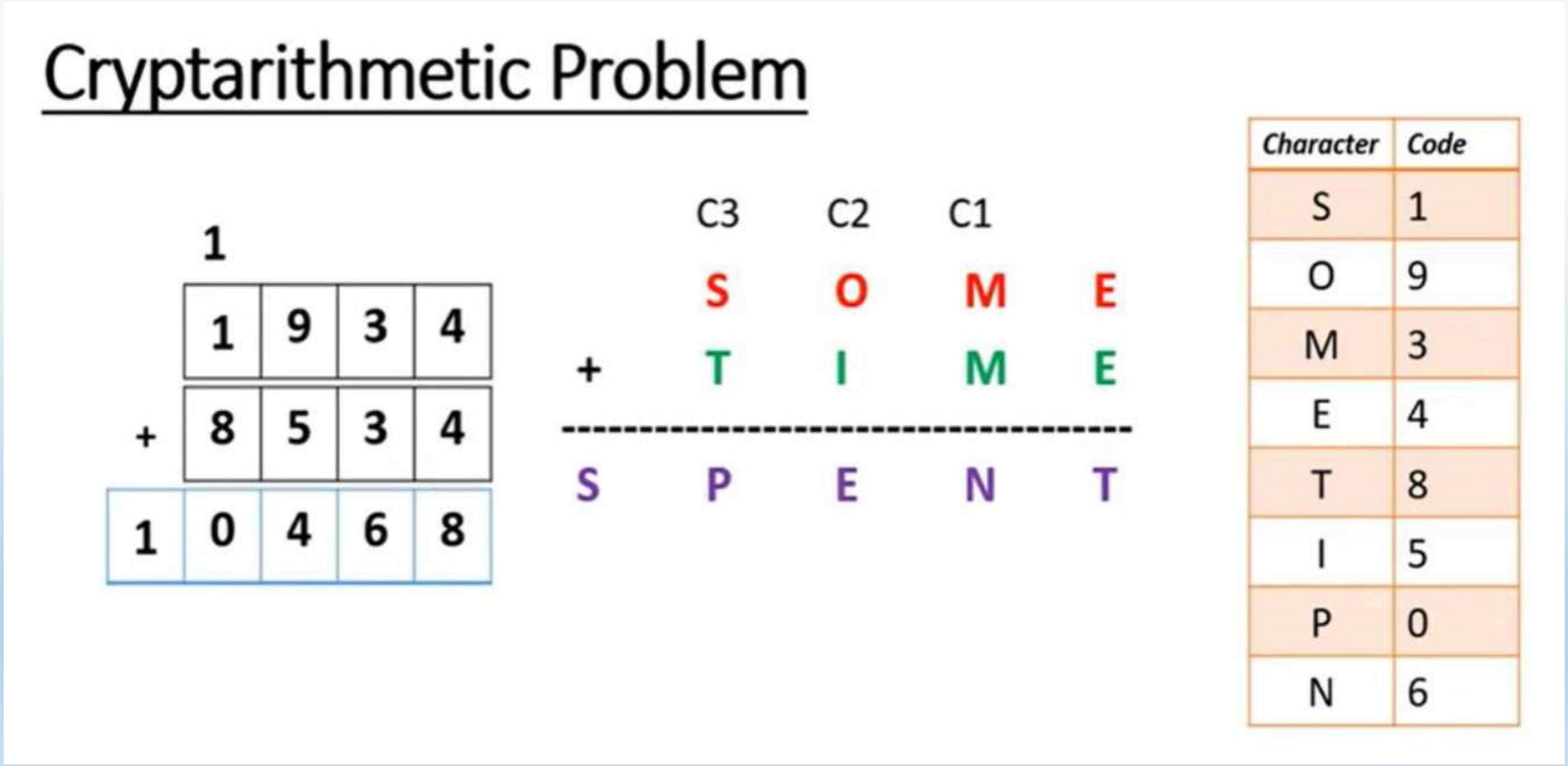
```
     9 6 2 3 3
+  6 2 5 1 3
-----------
 1 5 8 7 4 6
```

```
C5  C4  C3  C2  C1

      C  R  O  S  S
+  R  O  A  D  S
-----------------
 D  A  N  G  E  R
```

Character	Code
D	1
C	9
R	6
A	5
O	2
N	8
G	7
S	3
E	4

CRYPT ARITHMETIC



Cryptarithmetic Problem

Character	Code
S	1
O	9
M	3
E	4
T	8
I	5
P	0
N	6

THANK-YOU