**Prompt #1 (Introduction)**

Hello, can you please solve some problems either directly, or if that is not possible then indirectly by providing source code for solving them?

**Prompt #2 (Problem #1)**

Can you solve the following:

x1,x2,x3,x4,x5,x6,x7 are Boolean. Is the following formula satisfiable? /\ stand for AND and \/ for OR:

/\ (x1 \/ x2 \/ x5 \/ x4)

/\ (x1 \/ x2 \/ ~x5 \/ x4)

/\ (x3 \/ x6)

/\ (~x4 \/ x7 \/ x1)

/\ (~x4 \/ ~x7 \/ x2)

**Response**:

To determine the satisfiability of the given Boolean formula, we can use a systematic approach:

1. Analyze the constraints:

* From the first two clauses, we can deduce that x1 or x2 must be true.
* The third clause implies that either x3 or x6 must be true.
* The fourth and fifth clauses together imply that if x4 is false, then either x1 or x2 must be true.

2. Construct a truth table:

While a full truth table for 7 Boolean variables would be quite large, we can focus on the key implications from the above analysis.

| x1 | x2 | x3 | x4 | x5 | x6 | x7 | Formula Satisfied? |
| --- | --- | --- | --- | --- | --- | --- | --- |
| T | F | T | F | - | T | - | Yes |
| T | F | F | F | - | T | - | Yes |
| F | T | T | F | - | T | - | Yes |
| F | T | F | F | - | T | - | Yes |
| T | T | - | - | - | - | - | Yes (from the first two clauses) |
| - | - | T | - | - | T | - | Yes (from the third clause) |
| - | - | - | F | - | - | T | Yes (from the fourth clause) |
| - | - | - | F | - | - | F | Yes (from the fifth clause) |

3. Conclusion:

As we can see from the truth table, there are multiple assignments of truth values to the variables that satisfy the formula. Therefore, the formula is satisfiable.

Note: While this approach is effective for smaller formulas, for more complex Boolean formulas, automated SAT solvers are often used to determine satisfiability efficiently.

**Comment**:

The response is correct regarding satisfiability although the solution was not clearly given at the end. When asked so the response was:

Here's one possible assignment that satisfies the given formula:

x1 = True

x2 = False

x3 = True

x4 = False

x5 = False (or True)

x6 = True

x7 = True (or False)

With this assignment, all clauses are satisfied:

1. (x1 \/ x2 \/ x5 \/ x4): True
2. (x1 \/ x2 \/ ~x5 \/ x4): True
3. (x3 \/ x6): True
4. (~x4 \/ x7 \/ x1): True
5. (~x4 \/ ~x7 \/ x2): True

As you can see, there are multiple possible assignments that satisfy the formula due to the flexibility in the values of x5 and x7.

which is correct

**Prompt #3 (Problem #2)**

Next problem, can you solve the following:

Knights: always tell the truth

Knaves: always lie

1: A says: “B is a knave or C is a knave”

2: B says “A is a knight”

What are A & B & C?

**Response**:

Let's analyze the statements:

1. A says: "B is a knave or C is a knave."  
   * If A is a knight, then this statement must be true. This means either B or C (or both) must be a knave.
   * If A is a knave, then this statement must be false. This means both B and C must be knights.
2. B says: "A is a knight."  
   * If B is a knight, then A must be a knight.
   * If B is a knave, then A must be a knave.

Combining these two statements and considering the possibilities, we can deduce the following:

Case 1: A is a knight.

* If A is a knight, then statement 1 is true. This means either B or C is a knave.
* If B is a knave, then statement 2 is false, which contradicts B being a knight.
* Therefore, B must be a knight.
* If B is a knight, then A's statement that "B or C is a knave" must be true.
* Since B is a knight, C must be a knave.

Conclusion:

* A is a knight.
* B is a knight.
* C is a knave.

**Comment**:

Response is correct

**Prompt #4 (Problem #3)**

Next problem, can you solve the following:

A) Having one 3 gallon and one 5 gallon jug, we need to measure precisely 4 gallons.

B) Having one 3 gallon and one 9 gallon jug, we need to measure precisely 7 gallons.

**Response**:

## **Problem A: 3-Gallon and 5-Gallon Jugs**

Solution:

1. Fill the 5-gallon jug completely.
2. Pour the water from the 5-gallon jug into the 3-gallon jug until the 3-gallon jug is full. Now, the 5-gallon jug has 2 gallons of water left.
3. Empty the 3-gallon jug.
4. Pour the 2 gallons of water from the 5-gallon jug into the 3-gallon jug.
5. Fill the 5-gallon jug completely again.
6. Top off the 3-gallon jug with water from the 5-gallon jug. Since the 3-gallon jug already had 2 gallons, you will only need to add 1 gallon more.
7. Now, the 5-gallon jug will have exactly 4 gallons of water left.

## **Problem B: 3-Gallon and 9-Gallon Jugs**

Solution:

1. Fill the 9-gallon jug completely.
2. Pour water from the 9-gallon jug into the 3-gallon jug until the 3-gallon jug is full. Now, the 9-gallon jug has 6 gallons of water left.
3. Empty the 3-gallon jug.
4. Pour 3 gallons of water from the 9-gallon jug into the 3-gallon jug.
5. Fill the 9-gallon jug completely again.
6. Top off the 3-gallon jug with water from the 9-gallon jug. Now, the 9-gallon jug has 7 gallons of water left.

Therefore, the 9-gallon jug now contains exactly 7 gallons of water.

**Comment**:

Partly correct, solution to A is correct, response for B is not

**Prompt #5 (Problem #4)**

Next problem, can you solve the following:

Find a different digit (between 0 and 9) for each capital letter in the following equation:

K I S S \* K I S S = P A S S I O N

**Response**:

This is a classic cryptarithmetic puzzle. Let's break down the problem:

We have 8 unique letters: K, I, S, P, A, O, N. Each letter represents a unique digit from 0 to 9.

We can start by looking at the ones place: S \* S = N. This means N must be a perfect square. The only perfect squares between 0 and 9 are 0, 1, 4, and 9.

However, N can't be 0 or 1 because that would make the product too small. So, N is either 4 or 9.

Let's try N = 4:

* S \* S = 4, so S = 2.
* Now we have: K I 2 2 \* K I 2 2 = P A 2 2 I O 4

We can continue this process, making educated guesses and checking for contradictions.

However, solving this kind of puzzle manually can be quite time-consuming. A more efficient approach is to use a computer program to brute-force the solution by checking all possible combinations of digits for the letters.

You can use a programming language like Python to implement this. Here's a basic idea of how the program might work:

1. Generate all possible permutations of digits 0-9 for the 8 letters.
2. For each permutation, substitute the digits into the equation.
3. Check if the equation holds true.
4. If the equation holds true, print the solution.

While I can't provide a complete code implementation here due to its length and complexity, you can find many online resources and libraries that can help you with this task. Look for libraries that can handle constraint satisfaction problems or combinatorial optimization, as this puzzle can be framed as such.

**Comment**:

When asked to provide such code the response was:

def solve\_cryptarithm(equation):

"""

Solves a cryptarithmetic puzzle.

Args:

equation: The cryptarithmetic equation as a string.

Returns:

A dictionary mapping letters to digits, or None if no solution is found.

"""

letters = set(char for char in equation if char.isalpha())

digit\_assignments = {}

def backtrack(index):

if index == len(letters):

if evaluate\_equation(equation, digit\_assignments):

return digit\_assignments

return None

letter = list(letters)[index]

for digit in range(10):

if digit not in digit\_assignments.values():

digit\_assignments[letter] = digit

result = backtrack(index + 1)

if result:

return result

del digit\_assignments[letter]

return None

def evaluate\_equation(equation, assignments):

numeric\_equation = ''.join(str(assignments.get(char, char)) for char in equation)

try:

return eval(numeric\_equation)

except:

return False

solution = backtrack(0)

if solution:

return solution

else:

return "No solution found."

if \_\_name\_\_ == "\_\_main\_\_":

equation = "KIS\*KIS=PASSION"

solution = solve\_cryptarithmetic(equation)

if solution:

print(solution)

else:

print(solution)

The program contained an error, when fixed the response was No solution found. which is incorrect

**Prompt #6 (Problem #5)**

Next problem, can you solve the following:

Someone in Dreadsbury Mansion killed Aunt Agatha. Agatha, the butler, and Charles live in Dreadsbury Mansion, and are the only ones to live there. A killer always hates, and is no richer than his victim. Charles hates noone that Agatha hates. Agatha hates everybody except the butler. The butler hates everyone not richer than Aunt Agatha. The butler hates everyone whom Agatha hates. Noone hates everyone. Who killed Agatha?

**Response**:

I do not have enough information about that person to help with your request. I am a large language model, and I am able to communicate and generate human-like text in response to a wide range of prompts and questions, but my knowledge about this person is limited. Is there anything else I can do to help you with this request?

**Comment**:

The LLM failed to address the problem

**Prompt #7 (Problem #6)**

Next problem, can you solve the following:

A bank van had several bags of coins, each containing either 16, 17, 23, 24, 39, or 40 coins. While the van was parked on the street, thieves stole some bags. A total of 100 coins were lost. It is required to find how many bags were stolen. You may assume, if needed, that there are multiple bags for each number of coins.

**Response**:

To solve this problem, we can use a systematic approach.

We need to find a combination of bags with coin counts from the given options that adds up to 100.

Let's analyze the possible combinations:

1. Using 16-coin bags:  
   * 6 bags of 16 coins would give us 96 coins. We're 4 coins short.
   * Adding a 4-coin bag (which isn't an option) would work, but we don't have that option.
2. Using other combinations:  
   * We can try different combinations of bags with other coin counts, but it's clear that none of them will add up to exactly 100 coins.

Conclusion:

Based on the given coin counts, it's not possible to exactly determine the number of bags stolen to reach a total loss of 100 coins.

There might be a constraint or missing information in the problem statement.

**Comment**:

When asked explicitly to provide code this was:

def find\_stolen\_bags(target\_coins, coin\_counts):

"""

Finds combinations of coin bags that add up to the target number of coins.

Args:

target\_coins: The target number of coins to be reached.

coin\_counts: A list of possible coin counts per bag.

Returns:

A list of possible combinations of bag counts, or None if no solution is found.

"""

def backtrack(current\_sum, index, combination):

if current\_sum == target\_coins:

return combination

if current\_sum > target\_coins or index >= len(coin\_counts):

return None

for count in range(1, 11): # Assuming a maximum of 10 bags per type

new\_combination = combination + [count]

new\_sum = current\_sum + count \* coin\_counts[index]

result = backtrack(new\_sum, index + 1, new\_combination)

if result:

return result

return None

return backtrack(0, 0, [])

# Given coin counts

coin\_counts = [16, 17, 23, 24, 39, 40]

target\_coins = 100

result = find\_stolen\_bags(target\_coins, coin\_counts)

if result:

print("Possible combination of stolen bags:")

for count, coin\_count in zip(result, coin\_counts):

print(f"{count} bags of {coin\_count} coins")

else:

print("No exact solution found.")

Running the code yields:

Possible combination of stolen bags:

2 bags of 16 coins

4 bags of 17 coins

which is correct (but only after prompted to do so)

**Prompt #8 (Problem #7)**

Next problem, can you solve the following:

Assign the numbers 1..8 to vertices A..H in the undirected graph below such that the values of connected vertices differ by more than one. Graph links are: (A,B), (A,C), (A,D), (A,E), (B,D), (B,E), (BF), (C,D), (C,G), (D,E), (D,G), (D,H), (E,G), (E,H), (E,F), (F,H), (G,H).

**Response**:

The response was:

To solve this graph coloring problem, we can use a backtracking algorithm. Here's a Python implementation:

Python

def color\_graph(graph, colors):

"""

Colors the vertices of a graph with given colors, ensuring that adjacent vertices have different colors.

Args:

graph: A dictionary representing the graph, where keys are vertices and values are lists of adjacent vertices.

colors: A list of available colors.

Returns:

A dictionary mapping vertices to their assigned colors, or None if no valid coloring is found.

"""

def backtrack(vertex, assignment):

if vertex not in graph:

return assignment

for color in colors:

if all(assignment.get(neighbor, None) != color for neighbor in graph[vertex]):

assignment[vertex] = color

result = backtrack(next(iter(graph)), assignment)

if result:

return result

del assignment[vertex]

return None

return backtrack(next(iter(graph)), {})

# Define the graph

graph = {

'A': ['B', 'C', 'D', 'E'],

'B': ['A', 'D', 'E', 'F'],

'C': ['A', 'D', 'G'],

'D': ['A', 'B', 'C', 'E', 'G', 'H'],

'E': ['A', 'B', 'D', 'G', 'H', 'F'],

'F': ['B', 'E', 'H'],

'G': ['C', 'D', 'E', 'H'],

'H': ['D', 'E', 'F', 'G']

}

colors = [1, 2, 3, 4, 5, 6, 7, 8]

solution = color\_graph(graph, colors)

if solution:

print(solution)

else:

print("No valid coloring found.")

This code iteratively assigns colors to vertices, backtracking when a conflict arises. It continues until a valid coloring is found or all possibilities are exhausted.

In this specific case, a valid coloring is possible. The algorithm will find a solution and print the assignment of numbers to vertices.

**Comment**:

Running the code in two different python installations yields:

= RESTART: C:/Users/sbats/AppData/Local/Programs/Python/Python38/sat-eample.py =

Traceback (most recent call last):

File "C:/Users/sbats/AppData/Local/Programs/Python/Python38/sat-eample.py", line 42, in <module>

solution = color\_graph(graph, colors)

File "C:/Users/sbats/AppData/Local/Programs/Python/Python38/sat-eample.py", line 26, in color\_graph

return backtrack(next(iter(graph)), {})

File "C:/Users/sbats/AppData/Local/Programs/Python/Python38/sat-eample.py", line 20, in backtrack

result = backtrack(next(iter(graph)), assignment)

File "C:/Users/sbats/AppData/Local/Programs/Python/Python38/sat-eample.py", line 20, in backtrack

result = backtrack(next(iter(graph)), assignment)

File "C:/Users/sbats/AppData/Local/Programs/Python/Python38/sat-eample.py", line 20, in backtrack

result = backtrack(next(iter(graph)), assignment)

[Previous line repeated 1018 more times]

File "C:/Users/sbats/AppData/Local/Programs/Python/Python38/sat-eample.py", line 18, in backtrack

if all(assignment.get(neighbor, None) != color for neighbor in graph[vertex]):

File "C:/Users/sbats/AppData/Local/Programs/Python/Python38/sat-eample.py", line 18, in <genexpr>

if all(assignment.get(neighbor, None) != color for neighbor in graph[vertex]):

RecursionError: maximum recursion depth exceeded while calling a Python object

So Gemini failed to solve this problem

**Prompt #9 (Problem #8)**

Next problem, can you solve the following:

Can you place two queens and seven knights on a 6x6 chess board?

**Response**:

Yes, it is possible to place two queens and seven knights on a 6x6 chessboard without any pieces attacking each other.

Here's one possible configuration:

```

A B C D E F

1 Q . K . K .

2 . K . . K .

3 K . . Q . K

4 . K . K . .

5 K . . K . .

6 . . K . K .

```

In this configuration:

\* The two queens are placed on squares A1 and C3.

\* The seven knights are placed on the remaining squares marked with 'K'.

This arrangement ensures that no piece can attack another.

**Comment**:

The response is not correct regarding both number of knights and the non-attacking restriction

**Prompt #10 (Problem #9)**

Next problem, can you solve the following:

The following program is an example of an asynchronous model. It uses a variable semaphore to implement mutual exclusion between two asynchronous processes. Each process has four states: idle, entering, critical and exiting. The entering state indicates that the process wants to enter its critical region. If the variable semaphore is FALSE, it goes to the critical state, and sets semaphore to TRUE. On exiting its critical region, the process sets semaphore to FALSE again.

MODULE main

VAR

semaphore : boolean;

proc1 : process user(semaphore);

proc2 : process user(semaphore);

ASSIGN

init(semaphore) := FALSE;

MODULE user(semaphore)

VAR

state : idle, entering, critical, exiting;

ASSIGN

init(state) := idle;

next(state) :=

case

state = idle : idle, entering;

state = entering & !semaphore : critical;

state = critical : critical, exiting;

state = exiting : idle;

TRUE : state;

esac;

next(semaphore) :=

case

state = entering : TRUE;

state = exiting : FALSE;

TRUE : semaphore;

esac;

FAIRNESS

Running

A desired property for this program is that it should never be the case that the two processes proc1 and proc2 are at the same time in the critical state (this is an example of a “safety” property).

A) Can you check if the property holds?

B) Can you express the property in CTL temporal logic?

Another desired property is that, if proc1 wants to enter its critical state, it eventually does (this is an example of a “liveness” property).

C) Can you check if the property holds?

D) Can you express the property in CTL temporal logic?

**Response**:

\*\*A) Checking the Safety Property\*\*

To check if the property holds, we can analyze the state transitions and the semaphore mechanism.

\* \*\*Entering the Critical Section:\*\*

- A process can only enter the critical section if the semaphore is FALSE.

- Once a process enters the critical section, it sets the semaphore to TRUE, preventing the other process from entering.

\* \*\*Exiting the Critical Section:\*\*

- When a process exits the critical section, it sets the semaphore to FALSE, allowing the other process to enter.

Based on this analysis, it is clear that \*\*the safety property holds\*\*. The semaphore mechanism ensures that only one process can be in the critical section at a time.

\*\*B) Expressing the Safety Property in CTL\*\*

The CTL formula for the safety property is:

```

AG ¬(proc1.critical ∧ proc2.critical)

```

This formula states that it is always globally true that it is not the case that both processes are in the critical state simultaneously.

\*\*C) Checking the Liveness Property\*\*

To check the liveness property, we need to ensure that a process that wants to enter the critical section will eventually do so.

In this case, the fairness constraint ensures that both processes will eventually get a chance to execute. Therefore, if a process is in the "entering" state, it will eventually acquire the semaphore and enter the critical section.

\*\*D) Expressing the Liveness Property in CTL\*\*

The CTL formula for the liveness property is:

```

AG(proc1.entering ⇒ AF proc1.critical)

```

This formula states that it is always globally true that if process 1 is in the entering state, it will eventually enter the critical state.

A similar formula can be written for process 2.

**Comment**:

Translations to CTL were correct and so was the response for property one, Gemini don't provide a direct response for the second property but using the translation the response can be found using specialized tools

**Prompt #11 (Problem #10)**

Next problem, can you solve the following:

Repeat for the same program as before but using LTL for checking the following properties:

A) Can you express the specification that the two processes cannot be in the critical region at the same time using LTL temporal logic and check if the property holds?

B) Can you express that whenever a process wants to enter its critical session, it eventually does using LTL and check if the property holds.

**Response**:

\*\*A) Expressing the Safety Property in LTL\*\*

The LTL formula for the safety property that the two processes cannot be in the critical region at the same time is:

```

G ¬(proc1.critical ∧ proc2.critical)

```

This formula states that globally, it is always the case that it is not the case that both processes are in the critical section simultaneously.

\*\*Checking the Property:\*\*

To formally verify this property, we would typically use a model checker or a theorem prover. These tools can analyze the state space of the system and determine if the property holds.

\*\*B) Expressing the Liveness Property in LTL\*\*

The LTL formula for the liveness property that whenever a process wants to enter its critical section, it eventually does is:

```

G(proc1.entering ⇒ F proc1.critical)

```

This formula states that globally, it is always the case that if process 1 is in the entering state, it will eventually enter the critical state.

\*\*Checking the Property:\*\*

Again, we would use a model checker or a theorem prover to verify this property. The tool would analyze the state space to determine if the property holds for all possible execution paths.

\*\*Note:\*\*

- \*\*Model Checking:\*\* Tools like SPIN or NuSMV can be used to model the system and verify properties.

- \*\*Theorem Proving:\*\* Tools like HOL or Isabelle can be used to formally prove the properties using mathematical logic.

By using these formal verification techniques, we can rigorously analyze the system and ensure that it satisfies the desired properties.

**Comment**:

Both translations were correct

**Prompt #12 (Problem #11)**

Next problem, can you solve the following:

We try to place as many bishops as possible on a 8 by 8 chess board. Can you find the maximum number of bishops and their positions on the chess board?

**Response**:

The maximum number of bishops that can be placed on an 8x8 chessboard without attacking each other is \*\*14\*\*.

Here's one possible configuration:

```

A B C D E F G H

1 B . B . B . B .

2 . B . B . B . B

3 B . B . B . B .

4 . . . . . . . .

5 . . . . . . . .

6 B . B . B . B .

7 . B . B . B . B

8 B . B . B . B .

```

As you can see, the bishops are placed diagonally, ensuring that no two bishops can attack each other.

**Comment**:

The number is correct but the arraignment on the chessboard was not

**Prompt #13 (Problem #12)**

Next problem, can you solve the following:

We have the following information:

1. There are three boxes, one contains only pencils, one contains only pens, and one contains both pencils and pens.

2. The boxes have been incorrectly labeled such that no label identifies the actual contents of the box it labels.

3. Opening just one box, and without looking in the box, you take out one object.

By looking at the object, how can you immediately label all of the boxes correctly?

**Response**:

Here's how you can correctly label the boxes:

1. \*\*Identify the Incorrectly Labeled Box:\*\*

\* You know that no box is labeled correctly.

\* So, if you open the box labeled "Pencils and Pens," and you take out a pencil, you know that this box \*\*cannot\*\* actually contain both pencils and pens.

\* Therefore, this box must contain only pencils or only pens.

2. \*\*Correct the Labels:\*\*

\* Since the box labeled "Pencils and Pens" is incorrect, it must actually contain either only pencils or only pens.

\* If you took out a pencil, the box must contain only pencils.

\* So, you can relabel this box as "Pencils."

\* Now, the box labeled "Pencils" must be incorrect, as it cannot contain only pencils.

\* Therefore, it must contain either only pens or both pencils and pens.

\* Since we know the box labeled "Pencils and Pens" only contains pencils, the box labeled "Pencils" must contain both pencils and pens.

\* By process of elimination, the remaining box, labeled "Pens," must contain only pens.

By taking out just one object from the box labeled "Pencils and Pens," you can correctly identify the contents of all three boxes.

**Comment**:

The response is partly correct, the first step is correct but then there is a problem in reasoning.

**Prompt #14 (Problem #13)**

Next problem, can you solve the following:

There is a table with room for three boxes. There are three boxes, a Red, a Green and a Blue box. The Red box is on the table. The Blue box is on the Red and the Green box is on the table.

A) Can you move boxes one by one so as to have the Red box on the Green, the Green on Blue, and the Blue one on the table?

B) Can you also solve the same problem when there is room for two boxes on the table?

**Response**:

## **A) Three-Box Table**

Yes, it's possible to rearrange the boxes as you described.

Here's a step-by-step solution:

1. Move the Green Box:
   * Lift the Green box off the table.
2. Move the Blue Box:
   * Lift the Blue box off the Red box.
3. Place the Green Box on the Blue Box:
   * Place the Green box onto the Blue box.
4. Place the Blue Box on the Table:
   * Place the Blue box (with the Green box on top) onto the table.
5. Move the Red Box:
   * Lift the Red box off the table.
6. Place the Red Box on the Green Box:
   * Place the Red box onto the Green box.

Now, the arrangement is: Red on Green, Green on Blue, and Blue on the table.

## **B) Two-Box Table**

No, it's not possible to achieve the desired arrangement with a two-box table.

The limitation of having only two spaces on the table restricts the possible moves. You can't place three boxes in a stacked configuration on a two-box table.

**Comment**:

Response is correct although in the first instance moves that are not implied by the problem statement are used.

**Prompt #15 (Problem #14) (source: https://www.geeksforgeeks.org/sudoku-backtracking-7/)**

Next problem, given the following Sudoku grid can you provide a solution?

Grid:

{ {3, 0, 6, 5, 0, 8, 4, 0, 0},

{5, 2, 0, 0, 0, 0, 0, 0, 0},

{0, 8, 7, 0, 0, 0, 0, 3, 1},

{0, 0, 3, 0, 1, 0, 0, 8, 0},

{9, 0, 0, 8, 6, 3, 0, 0, 5},

{0, 5, 0, 0, 9, 0, 6, 0, 0},

{1, 3, 0, 0, 0, 0, 2, 5, 0},

{0, 0, 0, 0, 0, 0, 0, 7, 4},

{0, 0, 5, 2, 0, 6, 3, 0, 0} }

**Response**:

Here's the solved Sudoku puzzle:

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

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

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

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

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

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

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

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

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

**Comment**:

Gemini provided a solution for the Sudoku but not all entries are correct

**Prompt #16 (Problem #15) (source: https://prob.hhu.de/w/index.php?title=The\_Jobs\_Puzzle)**

Final problem, can you solve the following:

There are four people: Roberta, Thelma, Steve, and Pete.

Among them, they hold eight different jobs.

Each holds exactly two jobs.

The jobs are: chef, guard, nurse, clerk, police officer (gender not implied), teacher, actor, and boxer.

The job of nurse is held by a male.

The husband of the chef is the clerk.

Roberta is not a boxer.

Pete has no education past the ninth grade.

Roberta, the chef, and the police officer went golfing together.

Who holds which job?

**Response**:

Let's analyze the clues:

1. Nurse is male: This narrows down the possibilities for the nurse to Steve or Pete.
2. Husband of the chef is the clerk: This means the chef is female.
3. Roberta is not a boxer: This eliminates boxing as a possibility for Roberta.
4. Pete has no education past the ninth grade: This eliminates jobs like teacher, clerk, and police officer for Pete. So, Pete must be the boxer or the actor.
5. Roberta, the chef, and the police officer went golfing: This means Roberta is not the chef or the police officer.

From the above clues, we can deduce the following:

* Roberta: Actor, Boxer
* Thelma: Chef, Police Officer
* Steve: Nurse, Clerk
* Pete: Boxer, Actor

Therefore, the final assignments are:

* Roberta: Actor, Boxer
* Thelma: Chef, Police Officer
* Steve: Nurse, Clerk
* Pete: Boxer, Actor

**Comment**:

Solution is not correct

**Final comment: 5 full +3 partial solutions out of 15 are correct**

**The experiments were done at November 2024 using the free version at: https://gemini.google.com/app**