**Name:** Mohamed Nagy Elkholy

**HW6**

**1. Dynamic Variable Ordering:**

1. Dynamic variable ordering in Backtrack Search refers to a method that strategically chooses variables during the search process. This technique involves adapting the selection criteria based on the changing state. It aims to improve the efficiency of the backtracking algorithm by smartly determining the order in which variables are assigned values.

a. Degree Heuristic: This heuristic selects the variable that is involved in the largest number of constraints with other unassigned variables. The idea is to prioritize the variable with the highest number of unassigned neighbors, as it might have more impact on reducing the domain of other variables.

b. Minimum Remaining Values (MRV): This heuristic prioritizes selecting variables with the fewest remaining legal values. By focusing on variables with the smallest domain size, it tends to reduce the search space more quickly, which may lead to faster problem-solving.

c. Least Constraining Value (LCV): After selecting a variable, this heuristic guides the selection of values for that variable by considering the potential impact on other variables. It aims to choose the value that constrains the fewest values in the remaining unassigned variables, which might prevent dead-ends and increase the chances of finding a solution.

**2. Constraint propagation:**

1. CSP I:

It is arc consistent because for each value a of each variable, and each neighbor, there is some value of b in the domain of the neighbor which is consistent with a as it satisfies the constraint between the variable and its neighbor.

1. CSP II:

It is not arc consistent because there is a variable (2) in the domain of x that does not satisfy the constraint [(x+y) is odd]. So, the domain of x must be reduced using the constraint [(x+y) is odd] to eliminate (2), so the domain will be {1}.

**3. Simple scheduling problem:**

1. Variables:

B: The time when bulb construction starts.

P: The time when solar panel making starts.

W The time when the wiring starts.

H: The time when the assembling of the housing starts.

Domains:

B = P = W = H = {1,2,3,4}. The domain

Constraints

Unary Constraints: (Each task must be done before the total 4 hours)

B + 2 <= 5 >>> B <= 3

P + 1 <= 5 >>> P <= 4

W + 2 <= 5 >>> W <= 3

H + 1 <= 5 >>> H <= 4

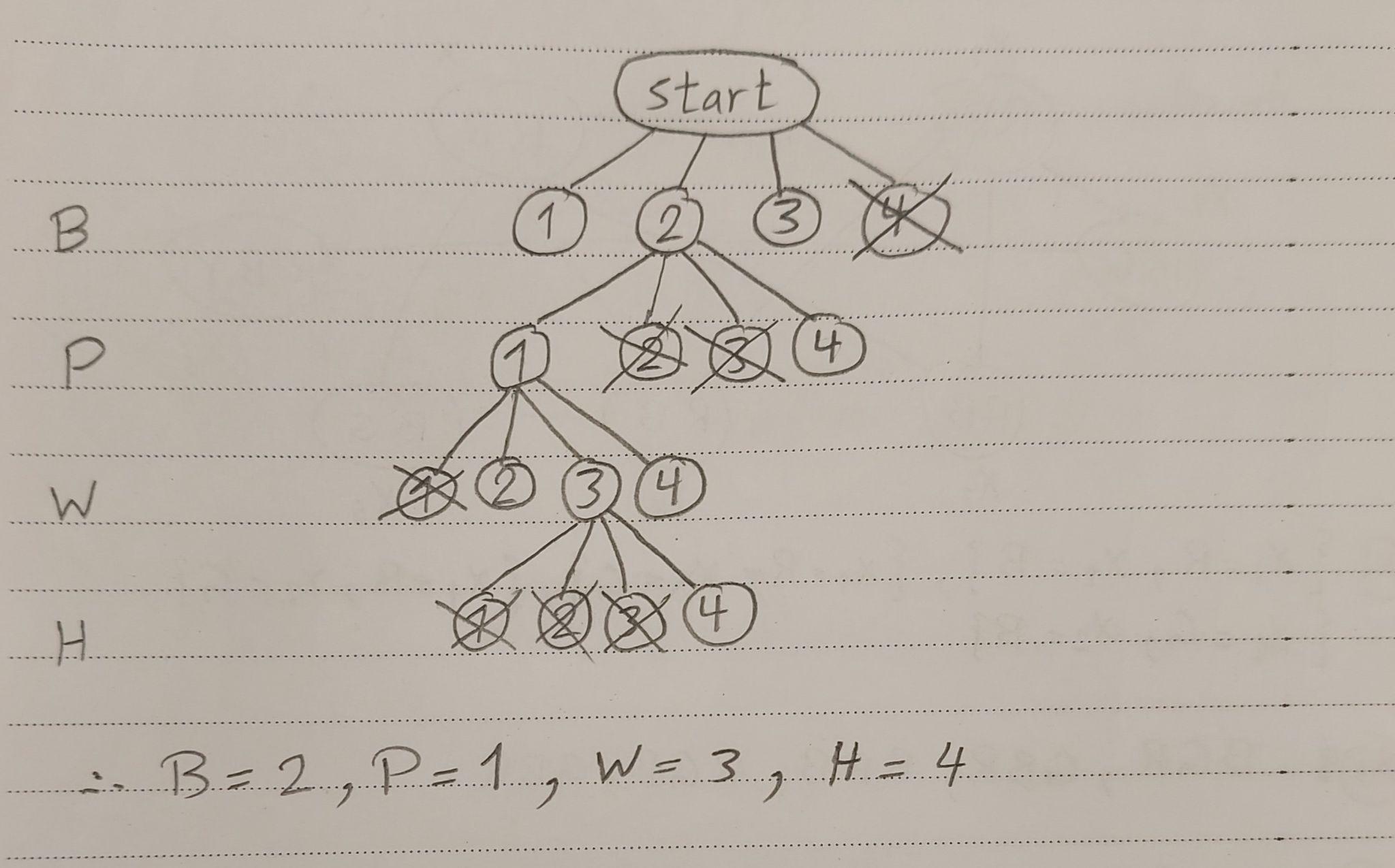
Binary Constraints: (Some tasks must be performed before the others)

B + 2 <= H

P + 1 <= W

Binary Constraints: (Some tasks cannot be performed simultaneously)

(P + 1 <= B) OR (B + 2 <= P)



**4. Consistency checking:**

1. To achieve strong 3-consistency, we need to first ensure arc-consistency (2-consistency), and then proceed to path-consistency (3-consistency).

Arc-Consistency (2-Consistency):

Updated domains after arc-consistency:

- x1 = {R, G}

- x2 = {B}

- x3 = {R}

- x4 = {R}

- x5 = {G}

- x6 = {R, Y}

- x7 = {}

Path-Consistency (3-Consistency): We don't have any universal constraints with three variables involved. So, there's no need to further refine the constraints, and we have achieved strong 3-consistency.

1. Based on the strong 3-consistent CSP:

Start by selecting variables with only one remaining value in their domains (fixed variables):

- x2 = {B} (only one choice)

- x3 = {R} (only one choice)

- x4 = {R} (only one choice)

- x5 = {G} (only one choice)

The domains of the remaining variables are:

- x1 = {R, G}

- x6 = {R, Y}

- x7 = {}

Since there are no constraints involving x7, we can select any color for it. For example, B.

- x1: {R, G}

- x6: {R, Y}

- x7: {B}

For x1 and x6:

- x1: {R, G} (choose G)

- x6: {R, Y} (choose Y)

The domains are all filled, and the CSP is satisfied:

- x1 = G

- x2 = B

- x3 = R

- x4 = R

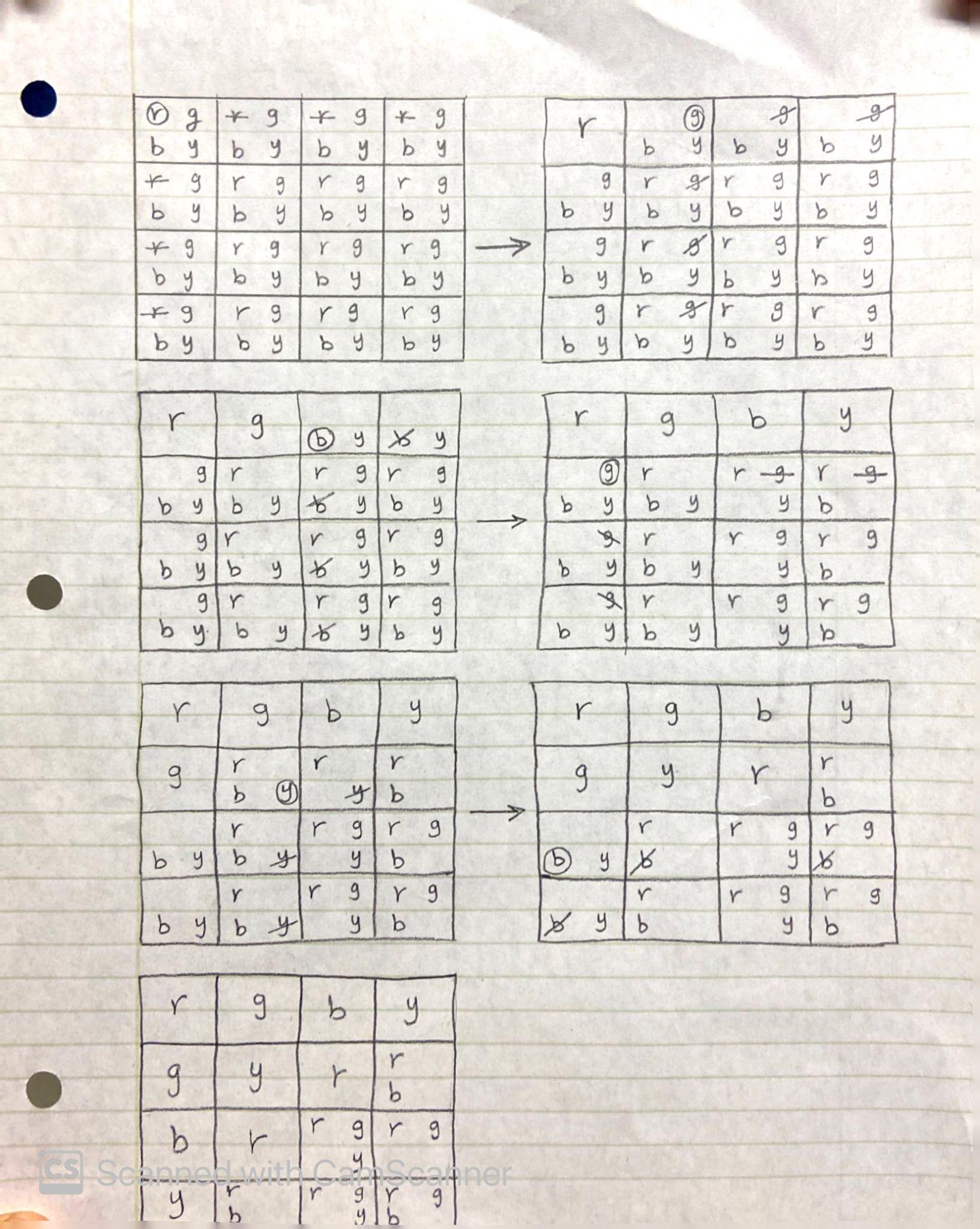
- x5 = G

- x6 = Y

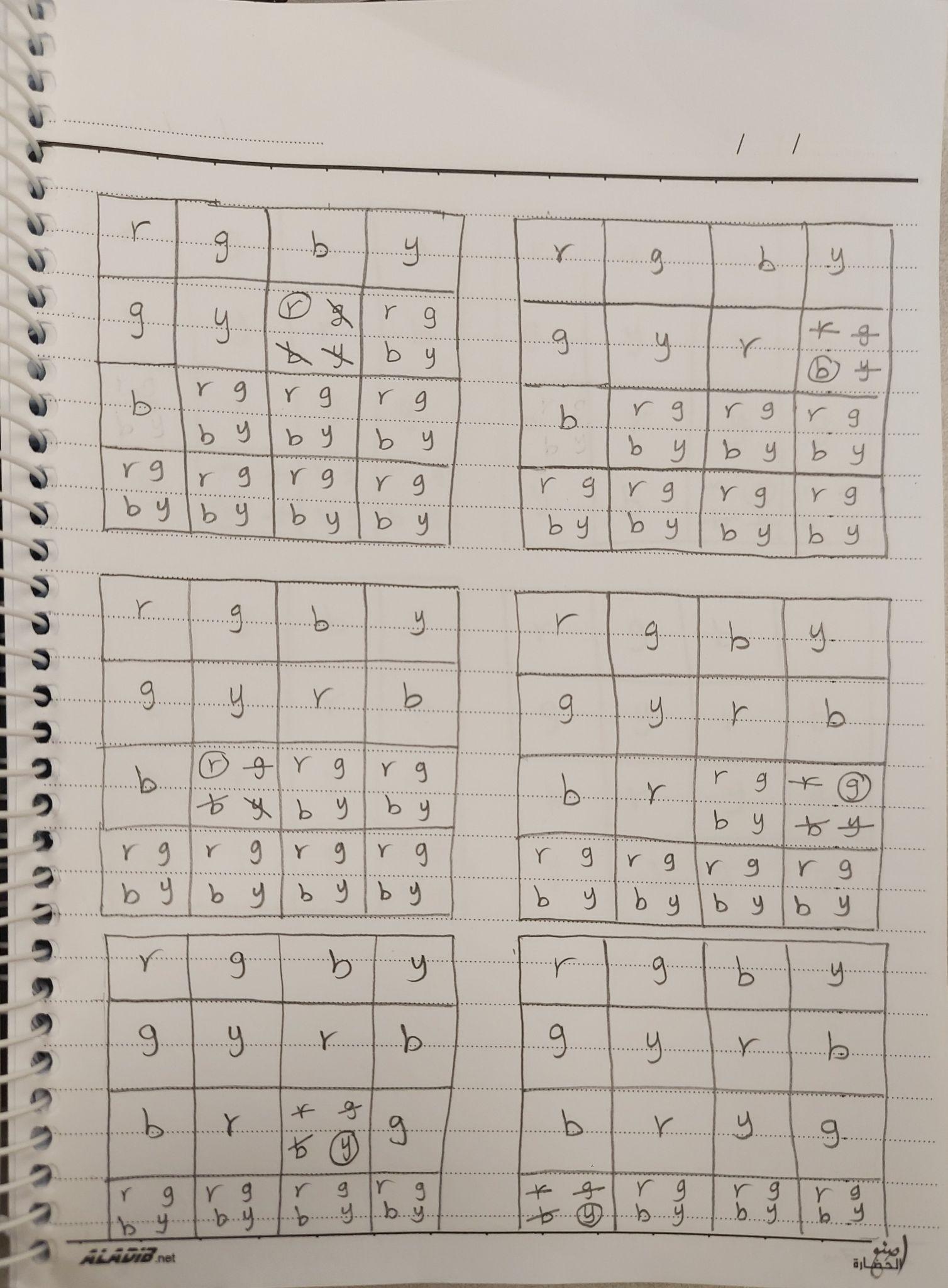
- x7 = B

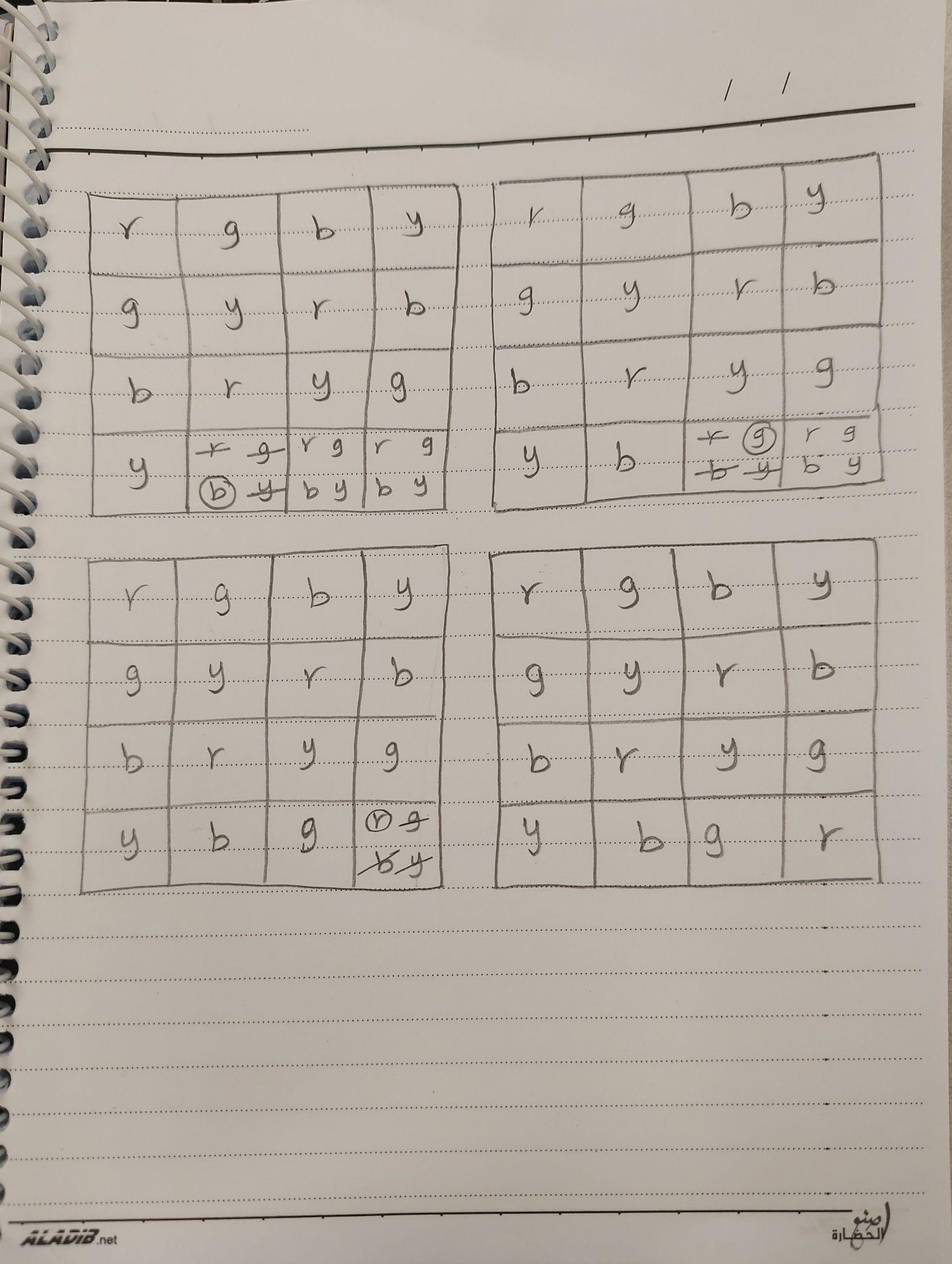
**5. Latin square:**

The partial look-ahead strategy, forward checking:



The full look-ahead strategy, maintaining arc consistency:





**6. Reduction of 3SAT into a CSP:**

* Variables of CSP: For the Boolean variables in 3SAT, let X represent the set of Boolean variables. Each variable can take two values: True (T) and False (F).
* Representation of a Clause in CSP: A clause in 3SAT is a disjunction of literals. To represent a clause in CSP, we create a constraint for each clause, allowing a disjunction of variables or their negations. A constraint is formed for each clause with three literals.
* Representation of a 3SAT Sentence in CSP: The 3SAT sentence, which is a conjunction of clauses, is represented by creating constraints for each clause. The entire sentence becomes a conjunction of these individual clause constraints.
* Reduction of 3SAT Question to CSP: The question of 3SAT is reduced to the CSP by creating constraints for each clause, reflecting the disjunction of literals. The solution to 3SAT exists if and only if a solution to the corresponding CSP exists.

1. The constraints in the resulting CSP have an arity of 3, as each constraint is associated with a clause in 3SAT, representing the disjunction of three literals.

* Variables: c1, c2, c3, c4, c5
* Domains: Each variable can take values True or False.
* Constraints: We introduce variables for each literal in the clauses:

Clause 1: L1, L2, L3 (from (c1 ∨ c2 ∨ c3))

Clause 2: L4, L5, L6 (from (c2 ∨ c3 ∨ c4))

Clause 3: L7, L8 (from (¬c1 ∨ c5))

Clause 4: L9, L10, L11 (from (c1 ∨ c4 ∨ c5))

We add constraints for each clause to ensure exactly one of the literals in the clause is True:

- {L1, L2, L3} = ExactlyOneTrue(L1, L2, L3)

- {L4, L5, L6} = ExactlyOneTrue(L4, L5, L6)

- {L7, L8} = ExactlyOneTrue(L7, L8)

- {L9, L10, L11} = ExactlyOneTrue(L9, L10, L11)

1. To represent a clause with an arbitrary number of literals in CSP, we can use the same approach but introduce a variable for each literal in the clause and add a constraint that ensures exactly one of these literals is True. If the clause has n literals, we would introduce n variables and use an "ExactlyOneTrue" constraint for those n variables. So, we can handle clauses with any number of literals.