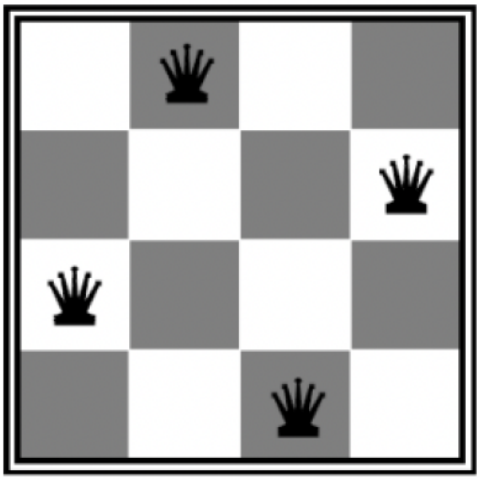
Constraint Satisfaction Problem (CSP) - Backtracking	start time:

Before you start, share this document with your team member(s) and then complete the form below to assign the role of speaker.

Team Role	Team Member
Speaker: shares your team's ideas with the class.	Kelii, Shreeya, Arogya

Part A. Backtracking	start time:



Q1. Recall the general idea of backtracking you have learned in 306.

Recursive function call,
Has a base case
State is stored in a variable and can be retrieved when back tracked
Stop backtracking when the list cannot be further explored, so backtrack to the next valid state

Q2. During the backtracking, does the order of assigning values to variables matter?

- Yes, because that is how backtracking is improved. Assigning one value may lead to finding a solution quickest than assigning another value.

Q3. What is the difference between the searching algorithm DFS and backtracking?

- In DFS, we keep moving forward/deeper in the nodes
- In backtracking, we can stop earlier and go back to previous nodes and move to a different child, we can cut earlier in the tree

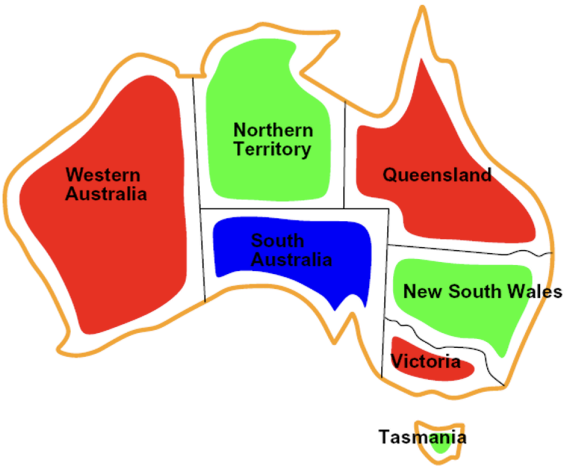
Q4. How could we use backtracking to solve CSP? Please describe the steps of solving CSP in backtracking using terms of CSP such as variable, domain, and constraint.

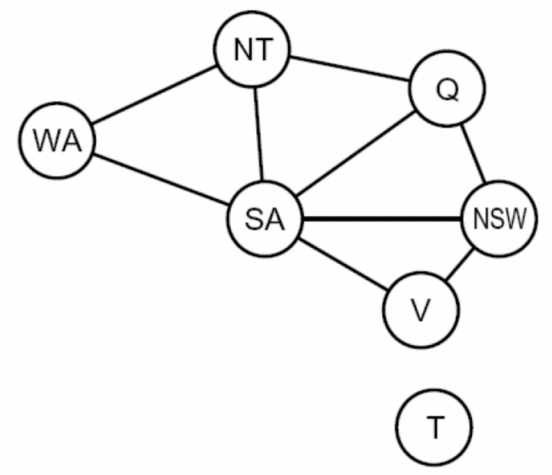
- Choose a variable
- Assign the value for this variable
- Figure out the constraints for that variable
- If it meets the constraint, select next variable else backtrack
- Check if it is the solution: when all variables have been assigned
- Possible backtrack: if the domain of a variable becomes empty

Q5. Write a pseudocode here to solve N-queen problem using backtracking. You need to formulate the N-queen problem into a CSP at first.

```
Func_N_queens:
    Func backtrack(assignment):
        If all assigned: return assignment
        Next_queen = select_unassigned_var()    # to optimize, choose variable in an
                                                  informed way to get to solution the quickest
        For value in domain:
            If is_consistent(Next_queen, value, assignment):
                Assignment[Next_queen] = value
                Result = backtrack(assignment)
                If result not failure: return result
                Assignment[Next_queen] = None #Backtrack
        Return failure

    Return backtrack({})
```



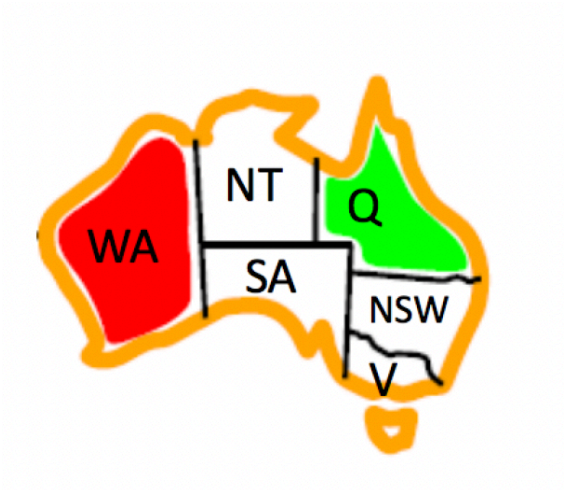


Consider map coloring the map of Australia using red, green, and blue. The constraints in this problem are simply that no two adjacent states can be the same color.
Q6. Please describe the steps of solving the map coloring problem using backtracking.

```
Func_map_coloring:
    Func backtrack(assignment):
        If all assigned: return assignment
        Next_color = select_unassigned_var()
        For value in domain:
            If is_consistent(Next_color, value, assignment):
                Assignment[Next_color] = value
                Result = backtrack(assignment)
                If result not failure: return result
                Assignment[Next_color] = None #Backtrack
        Return failure

    Return backtrack({})
```

Part B. Filtering - forward checking	start time:



Filtering - prune the domain of **unassigned** variables before we actually assign values for them later if we know ahead of time that certain values in their domain will result in backtracking.

Forward checking (a naive method of filtering) - whenever a value is assigned to Xi, removes values of domains from **unassigned** Xj if there is a conflict between the newly assigned value in Xi and the to-be-removed value in Xj. If the domain of Xj becomes empty after filtering, backtrack the assignment on Xi.

Note: prune not for all domain, but only for the unassigned neighbors' domain

Q7. Explain how and why forward checking can improve the performance of backtracking.

- By removing all the conflicting values (in the domain for a variable) using forward checking, we won't have to backtrack to values that are not fruitful, hence making the performance more efficient and faster. There will be less domains to explore for the next node and the performance gets faster.

Q8. Suppose red is assigned to WA, based on the forward checking, what values of variables will be removed from their domains?

- Red will be removed from NT's and SA's domain.

Q9. Suppose green is assigned to Q after Q8, based on the forward checking, what values of variables will be removed from their domains?

- Green will be removed from NT's, SA's, and NSW's domain

Q10. After Q9, assign a new value to one unassigned variable, repeat the forward checking, backtrack if necessary, until finding a solution.

- Assign Blue color to NT.
- SA will have an empty domain list, so backtrack to NT, then backtrack to Q…

Q11. Summarize a pseudocode of forward checking

```
this_function(chosen_variable, possible_values_domain):
    If possible_values_domain is empty:
        return False
    Possible_Assignment_Values = []
    Pruned_values = []
    Assign Possible_Assignment_Values[0] to the chosen_variable
    Loop through the unassigned successors of the variable
        Check their domain
        If the any value of the domain conflicts with the newly assigned value:
            Prune the value from the successor's domain
            Add the value to Pruned_values
            If no values in the domain:
                Possible_Assignment_Values.remove(0)
                Return this_function(variable, Possible_Assignment_Values)
    Return True
```

Part C. Filtering - arc consistency - AC3	start time:

The idea of forward checking can be generalized into the principle of arc consistency. For arc consistency, we interpret each undirected edge of the constraint graph for a CSP as two directed edges pointing in opposite directions. Each of these directed edges is called an arc. The arc consistency algorithm check all arcs to make sure that the graph is arc consistency on all arcs. If not, a backtracking occurs or no solution is found.

Here are the general steps of the AC-3 Algorithm to Maintain Arc Consistency:

1. Initialize the Constraint Queue
   Start with a queue of all arcs (variable pairs) in the CSP constraints.

2. Process the Queue Until Empty

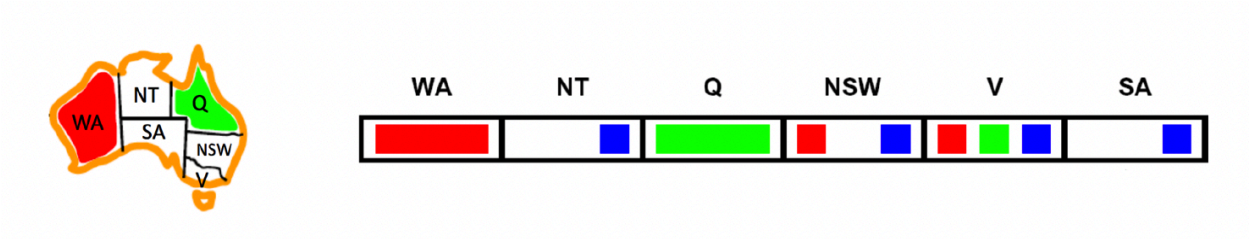
   While the queue is not empty:
       Remove an arc (X,Y) from the queue.

3. Revise the Domain of X Based on Y
   If there exists a value in X's domain that is inconsistent with all values in Y's domain (i.e., no value in Y satisfies the constraint), remove that value from X's domain.

4. Check for Domain Reduction
   If X's domain is reduced:
   - If X's domain becomes empty → backtrack (no solution possible).
   - Otherwise, add all arcs (Z,X) back to the queue for any neighbor Z of X (that are not already in the queue), ensuring consistency for affected variables.

5. Repeat Until No More Changes
   Continue processing arcs until the queue is empty, meaning the CSP is arc-consistent.

Suppose we are on the assignment below. Let's apply AC3 to filter values from variables before next assignment.



Q12. List out all arcs in a queue Q. Keep in mind that arc has direction.

WA -> NT
WA -> SA
NT -> WA
NT -> Q
NT -> SA
Q -> NT
Q -> SA
Q -> NSW
NSW -> Q
NSW -> V
NSW -> SA
V -> SA
V -> NSW
SA -> V
SA -> NSW
SA -> Q
SA -> NT
SA -> WA

18 total arcs.

Q13. Assume the first removed arc is SA -> V. Are there any values pruned?

SA (BLUE) -> V (BLUE, RED, GREEN)
No, there does not have to be because SA can be blue, as V can be either red or green.

Q14. Continue removing the second arc V-> SA from Q. Are there any values pruned?

The color blue is pruned from V's domain as SA cannot be anything other than blue.

Q15. Which arc is added to the Q after Q14?

The arcs to V are added to the Q since the domain of V changed.
So, SA -> V, NSW -> V (if not already in Q).

Q16. Continue this process until a solution is found or a domain of a variable becomes empty.

Skipped

QX. Comparing forward checking and AC3, which one is slower for filtering the domains?

AC3 is slower for filtering the domains because we add constraints back to the Queue and loop through. We also have to check for each domain twice for both ways.

QY. In what types of CSP problems would forward checking or AC-3 be considered better than the other?

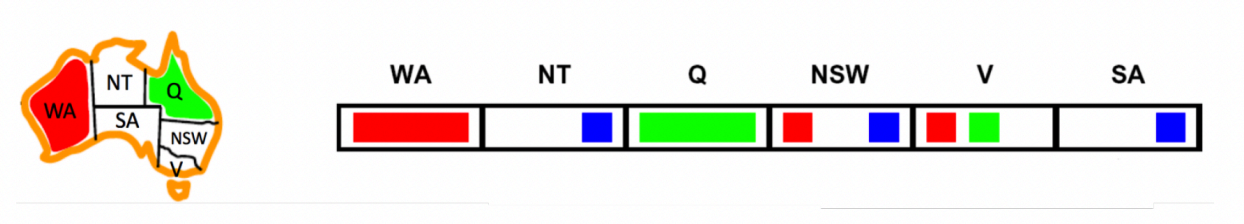
AC-3 would be better for Sudoku because there are numerous constraints.
Forward checking would be better for N-Queens.

Part D. Ordering - MRV	start time:

Ordering - choose order of variables or values to reach to the solution quickly with high possibility.

- MRV (Minimum Remaining Values) - better order of variables, choose the next best variable to assign value, which is the variable with the fewest remaining values.

- LCV (Least Constraining Value) - better order of values, choose the next best value for one variable - the one prunes fewest



Q17. Explain how MRV improves the backtracking.

> MRV checks the size of the domain and it is faster. It finds the variables that might fail/require backtracking earlier, which saves time.

Q18. Based on the MRV, which variable will be selected after assigning red for WA and green for Q.

> NT or SA

Part E. Applications	start time:

Air Traffic Control

We have five planes: A, B, C, D, and E and two runways: international and domestic. We would like to schedule a time slot and runway for each aircraft to either land or take off. We have four time slots: {1, 2, 3, 4} for each runway, during which we can schedule a landing or take off of a plane.

Constraints:
• Plane B has lost an engine and must land in time slot 1.
• Plane D can only arrive at the airport to land during or after time slot 3.
• Plane A is running low on fuel but can last until at most time slot 2.
• Plane D must land before plane C takes off, because some passengers must transfer from D to C.
• Planes A, B, and C cater to international flights and can only use the international runway.
• Planes D and E cater to domestic flights and can only use the domestic runway.
• No two aircrafts can reserve the same time slot for the same runway.

Q19. Copy and paste the formulation of CSP from the previous activity.

Constraints:
B_runway = A_runway = C_runway = international
D_runway = E_runway = domestic

A_timeslot != B_timeslot != C_timeslot
D_timeslot != E_timeslot

B_timeslot = 1
A_timeslot <= 2
C_timeslot > D_timeslot
D_timeslot >= 3

Q20. Since unary constraints determine which runway each plane must use, for the next two parts we will look at just the timeslot. What are the domains of the variables after enforcing unary constraints and arc-consistency?

B = (1)
A = (2)
D = (3, 4)
C = (4)
E = (1, 2)

Q21. Arc-consistency can be expensive to enforce, and we may get faster solutions using only forward checking. Using MRV, perform backtracking search, breaking ties by picking lower values first. List the variable assignments in the order they occur (including assignments that are reverted).

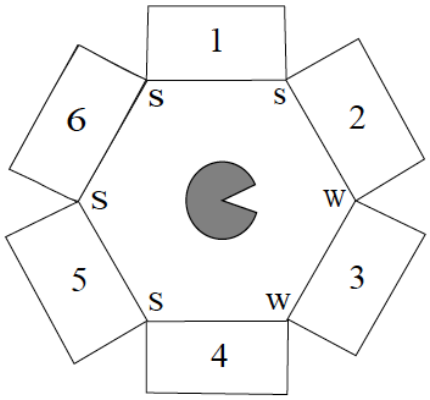
Order based on size of domain
B A C E D

Trapped Pacman

Pacman is trapped! He is surrounded by mysterious corridors, each of which leads to either a pit (P), a ghost (G), or an exit (E). In order to escape, he needs to figure out which corridors, if any, lead to an exit and freedom, rather than the certain doom of a pit or a ghost.

The one sign of what lies behind the corridors is the wind: a pit produces a strong breeze (S) and an exit produces a weak breeze (W), while a ghost doesn't produce any breeze at all. Unfortunately, Pacman cannot measure the strength of the breeze at a specific corridor. Instead, he can stand between two adjacent corridors and feel the max of the two breezes. For example, if he stands between a pit and an exit he will sense a strong(S) breeze, while if he stands between an exit and a ghost, he will sense a weak (W) breeze. The measurements for all intersections are shown in the figure below.

Also, while the total number of exits might be zero, one, or more, Pacman knows that two neighboring squares will not both be exits.



Pacman models this problem using variables Xi for each corridor i and domains P, G, and E.

Q22. Copy and paste the formulation of CSP from the previous activity.

S = strong breeze, W = weak breeze
P = pit, G = ghost, E = exit

E_total >= 0 and E_total <= i//2
Xi == E then X(i-1), X(i+1) != E
Xi != (S and W) == G

If S is felt, one of the adjacent doors is a pit

X2 & X3 & X4 != P
X2 == G or E
X1 = P or X6 = P

Q23. Suppose we assign X1 to E. Perform forward checking after this assignment. Also, enforce unary constraints.

X1 = E
X6 & X2 = not E (P, G)
X2 = G
X3 = E [not G (P, E)]
X4 = G [not E (P, G)]
X5 = not G (P, E)
X5 = P
X6 = G

Q24. Suppose forward checking returns the following set of possible assignments:

X1 P
X2 G E
X3 G E
X4 G E
X5 P
X6 P G E

According to MRV, which variable or variables could the solver assign first?

X1 or X5

Q25. Assume that Pacman knows that X6 = G. List all the solutions of this CSP or write none if no solutions exist.

X6 = G
Then
X1 = E [Not G, P]
Assuming X1 is P
X2 = E
X3 = G [not P]
X4 = E [Not P]
X5 = P [Not G]