# AI Numerical

▼ **TOC**



# PEAS

PEAS stands for Performance measure, Environment, Actuators, and Sensors. It is a simple framework used in designing AI agents.

## ▼ Example

**Self Driving Car**

Performance measure: safety, time, legal drive, comfort.

Environment: roads, other vehicles, road sign, pedestrians.

Actuator: Steering, accelerator, brake, horn, etc.

Sensor: Camera, GPS, speedometer, accelerometer, SONAR, odometer.

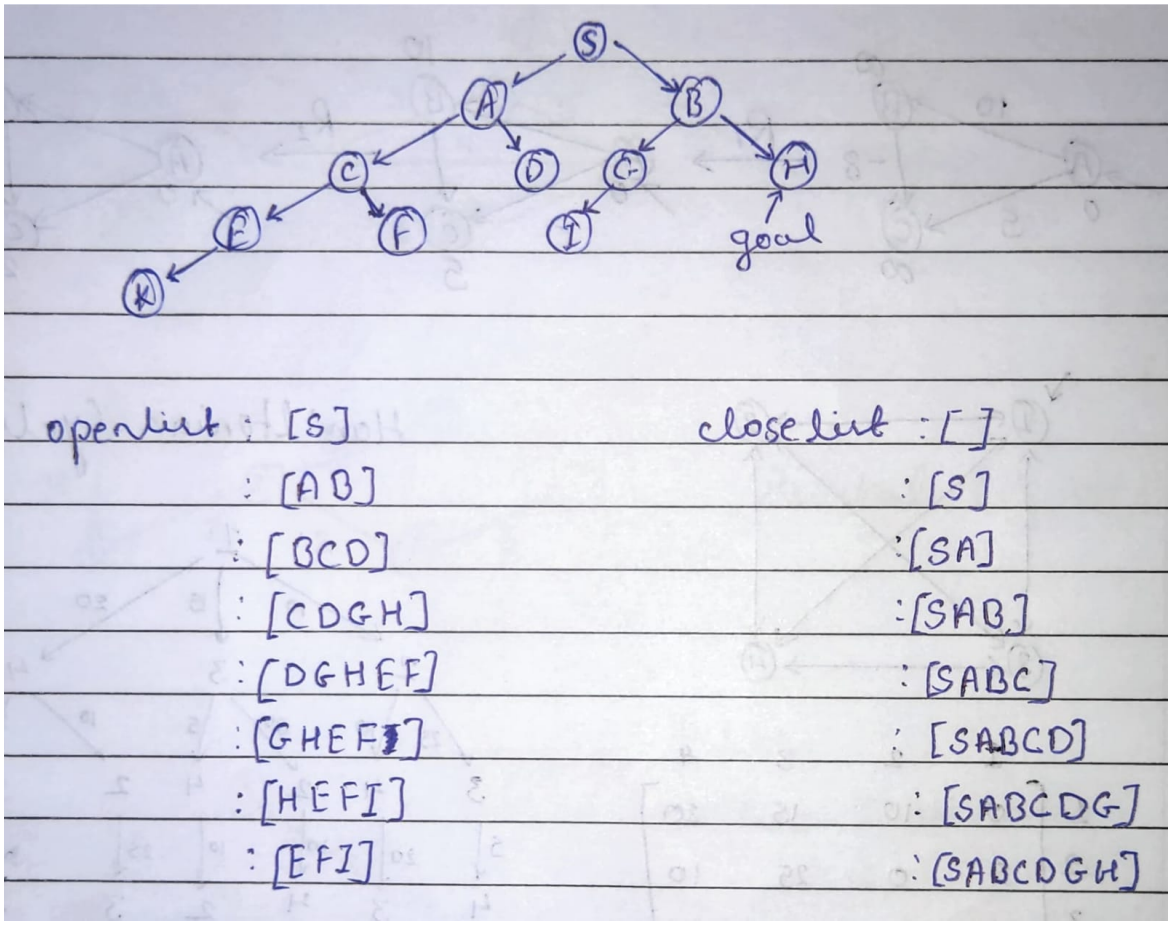
# Breadth first search

Breadth-first search is a graph traversal algorithm that explores all the neighboring vertices of a node before proceeding to the next level. It is a simple search algorithm that is often used in tree or graph data structures. The algorithm starts at the root node and visits all the nodes at the current depth level before moving to the next depth level.

## Algorithm

1. Initialize a queue and add the starting node to it.
2. Initialize the visited list as empty.
3. While the queue is not empty:
   a. Dequeue the first node from the queue.
   b. If the node is the goal node, return the solution.
   c. If the node has not been visited, mark it as visited and add it to the visited list.
   d. Generate the successors of the selected node and add them to the end of the queue.
4. If there is no solution, return failure.

## ▼ Example



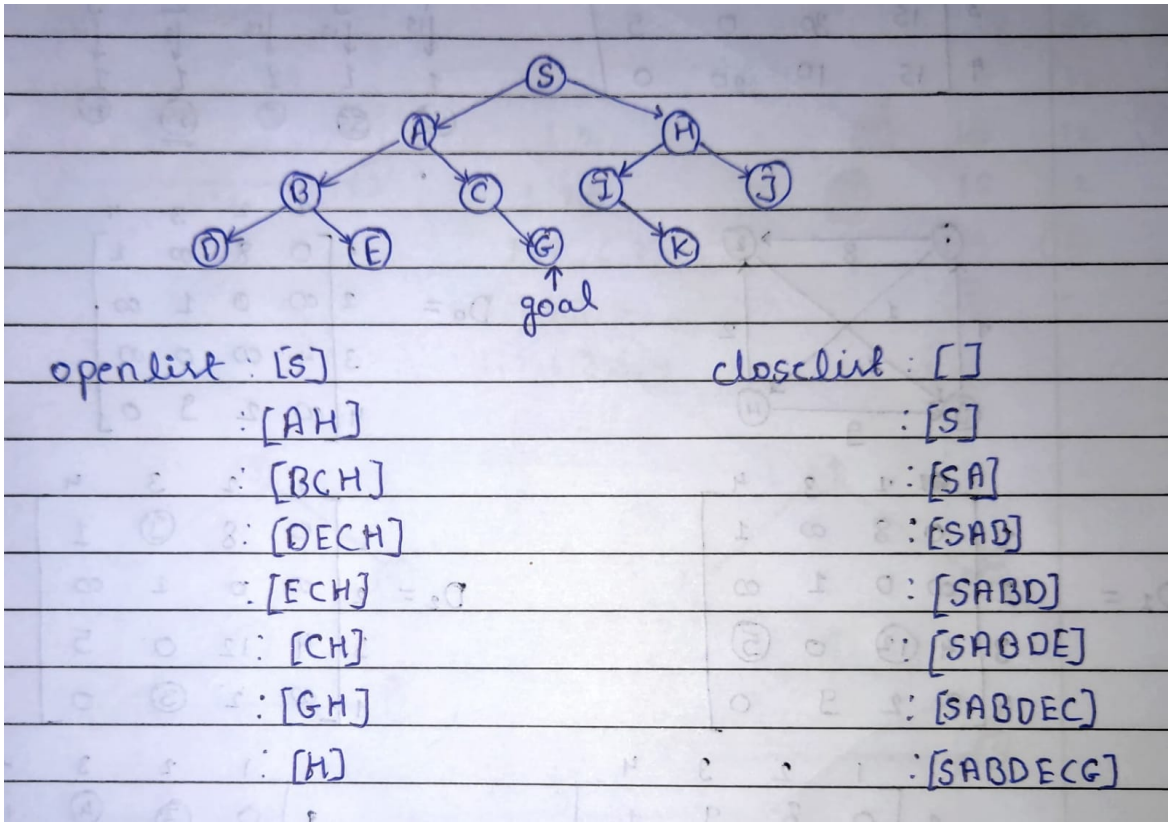

# Depth first search

Depth-first search is a graph traversal algorithm that explores the nodes along each branch as far as possible before backtracking. It is a simple search algorithm that is often used in tree or graph data structures. The algorithm starts at the root node and visits all the nodes along the current path before backtracking to the previous node.

## Algorithm

1. Initialize a stack and add the starting node to it.
2. Initialize the visited list as empty.
3. While the stack is not empty:
   a. Pop the first node from the stack.
   b. If the node is the goal node, return the solution.
   c. If the node has not been visited, mark it as visited and add it to the visited list.
   d. Generate the successors of the selected node and add them to the top of the stack.
4. If there is no solution, return failure.

## ▼ Example



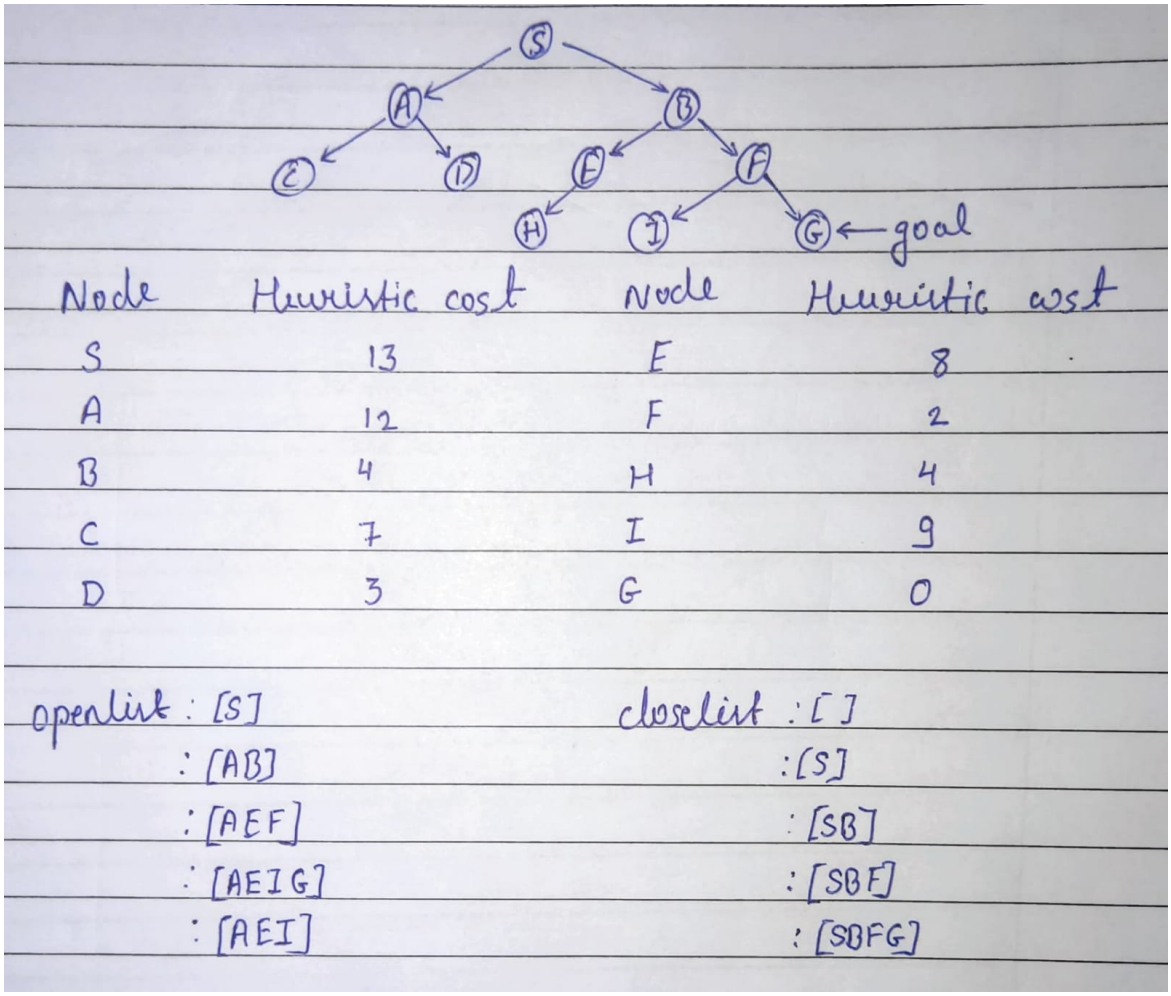

# Best first search

Best-first search is a heuristic search algorithm that is used to traverse a graph or tree data structure. It is an informed search algorithm that evaluates nodes based on a heuristic function and determines the best node to explore next. The algorithm is commonly used in various applications, including artificial intelligence, robotics, and natural language processing.

## Algorithm

1. Initialize the open list with the starting node.
2. Initialize the closed list as empty.
3. While the open list is not empty:
   a. Select the node with the highest priority from the open list.
   b. If the node is the goal node, return the solution.
   c. Generate the successors of the selected node.
   d. For each successor, compute the heuristic value and add it to the priority queue.
   e. Add the selected node to the closed list.
4. If there is no solution, return failure.

## ▼ Example




# A* Algorithm

A* is a search algorithm that combines the benefits of both breadth-first and depth-first search algorithms by using a heuristic function to determine the best node to explore next. The algorithm maintains two sets of nodes, the open set and the closed set, and evaluates nodes based on the sum of the actual cost from the start node and the estimated cost to the goal node.

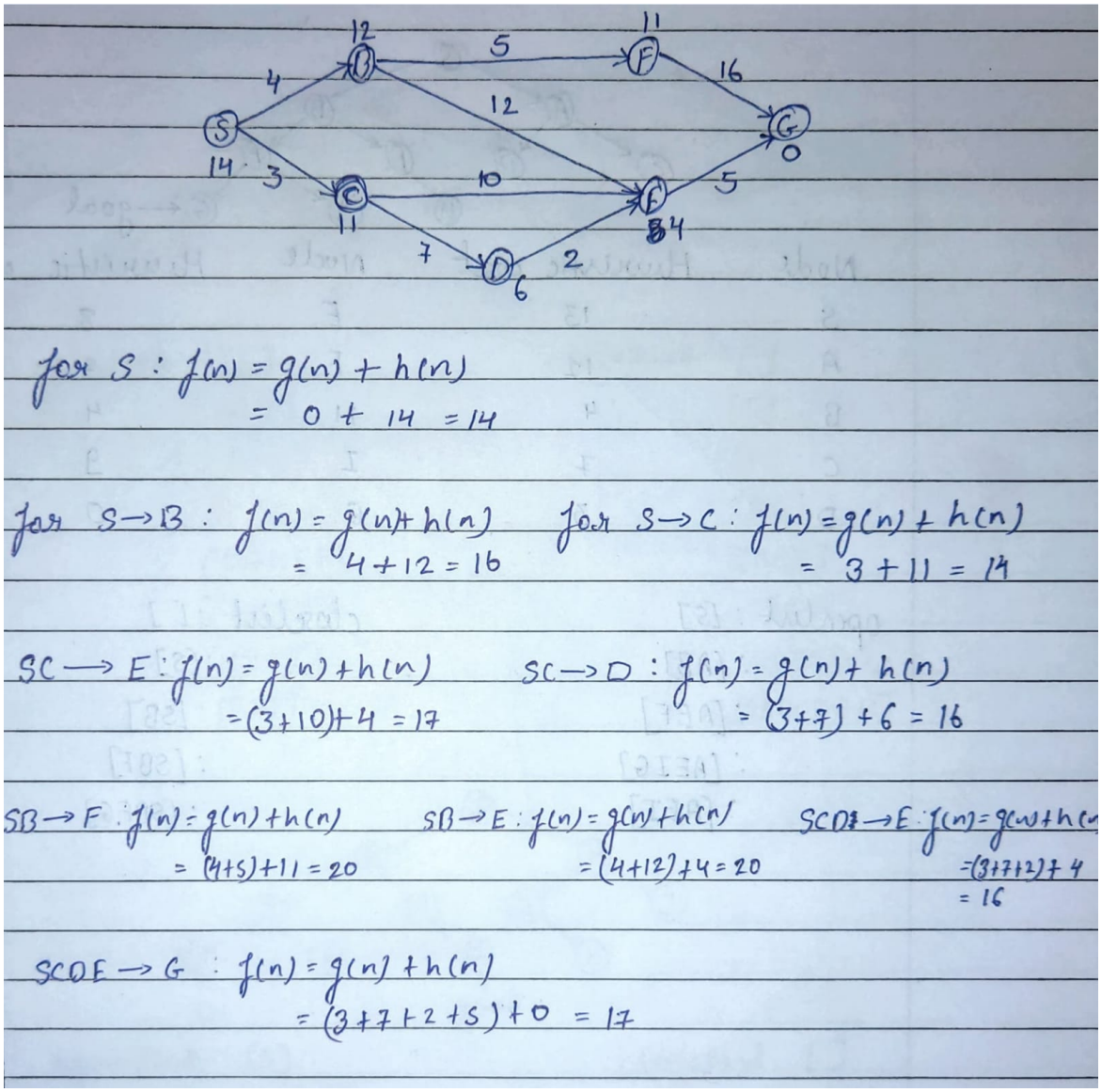
## Algorithm

1. Initialize the open set with the starting node and its cost to the goal node, f(start) = g(start) + h(start) = 0 + h(start).
2. Initialize the closed set as empty.
3. While the open set is not empty:
   a. Select the node with the lowest combined cost from the open set.
   b. If the node is the goal node, return the solution.
   c. Generate the successors of the selected node.
   d. For each successor, compute the actual cost from the start node to the successor, denoted by g(successor) = g(current) + cost(current, successor), where cost(current, successor) is the cost to move from the current node to the successor.

e. Compute the estimated cost from the successor to the goal node using a heuristic function, denoted by h(successor).
f. Calculate the combined cost of the successor, denoted by f(successor) = g(successor) + h(successor).
g. If the successor is not in the open set or its cost is lower than its previous cost, update the node with the new cost and add it to the open set.
h. Add the selected node to the closed set.

4. If there is no solution, return failure.

## ▼ Example




# A* Admissible

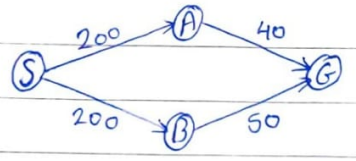
An admissible heuristic is one that never overestimates the cost to reach the goal node. In other words, the heuristic function must always be equal to or less than the actual cost to reach the goal

node. This property ensures that the A* algorithm always finds the optimal path from the start node to the goal node.

## Steps

1. Determine the actual cost to reach the goal node from the start node using a known algorithm or method.
2. Define a heuristic function that estimates the distance from a given node to the goal node.
3. For each node, calculate the actual cost to reach the goal node and compare it with the estimated cost given by the heuristic function.
4. If the estimated cost is always equal to or less than the actual cost for all nodes, then the heuristic function is admissible.
5. If the estimated cost is greater than the actual cost for at least one node, then the heuristic function is not admissible.

## ▼ Example




Cases:

$h(n) \leq h^*(n)$ - underestimation

$h(n) \geq h^*(n)$ - overestimation

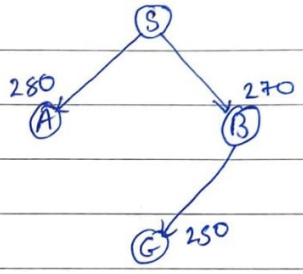
where:

$h(n)$ - estimated value

$h^*(n)$ - actual value

Case I: Overestimation

let $h(A) = 80$, $h(B) = 70$ $\} \geq h^*$



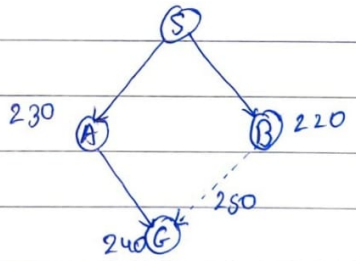

$S \rightarrow A : f(A) = 200 + 80 = 280$

$S \rightarrow B : f(B) = 200 + 70 = 270$

$S \rightarrow B \rightarrow G : f(G) = 250 + 0 = 250$

Case II: Underestimation

let $h(A) = 30$, $h(B) = 20$ $\} \leq h^*$




$S \rightarrow A : f(A) = 200 + 30 = 230$

$S \rightarrow B : f(B) = 200 + 20 = 220$

$S \rightarrow B \rightarrow G : f(G) = (200 + 50) + 0 = 250$

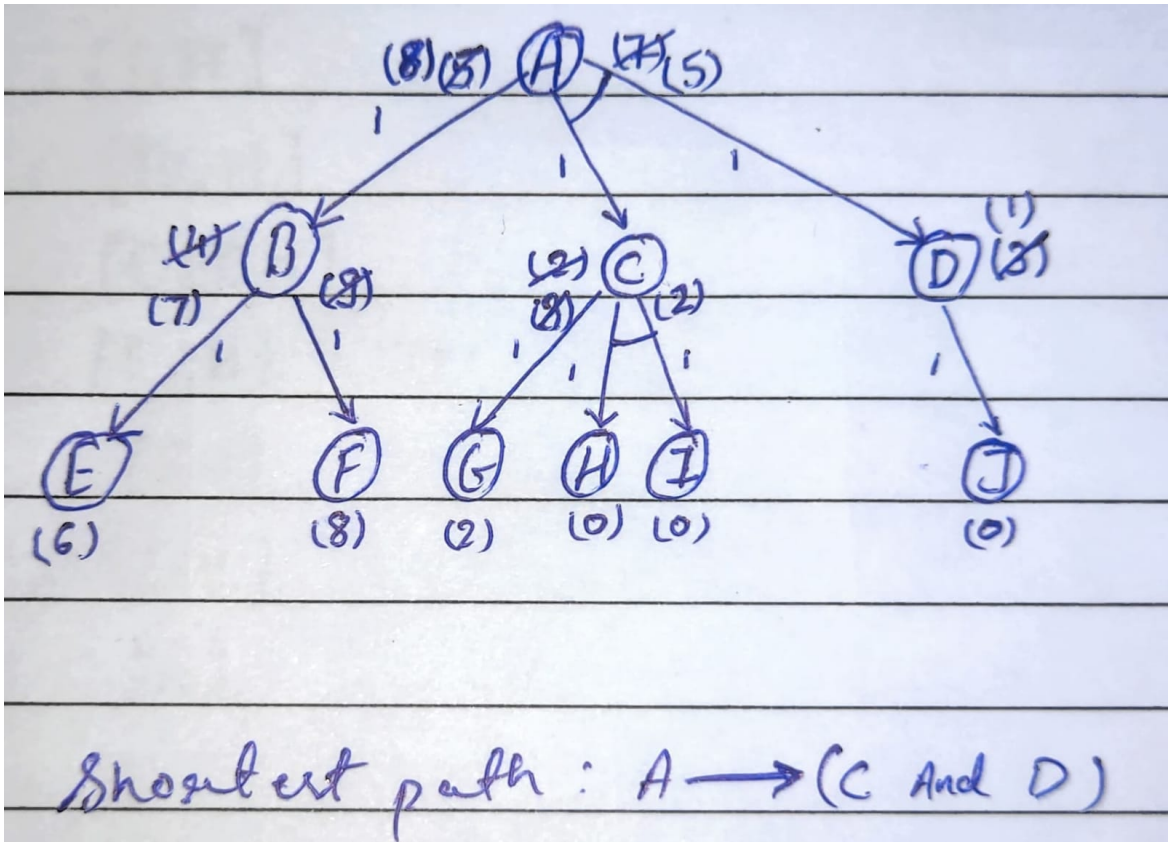
$S \rightarrow A \rightarrow G : f(G) = (200 + 40) + 0 = 240$

As path cost for S → A → G is lesser than any other path cost, it is the optimal solution.

If the estimated cost is less than actual cost than A* is said to be admissible, otherwise not.

# AO* Algorithm

## ▼ Example



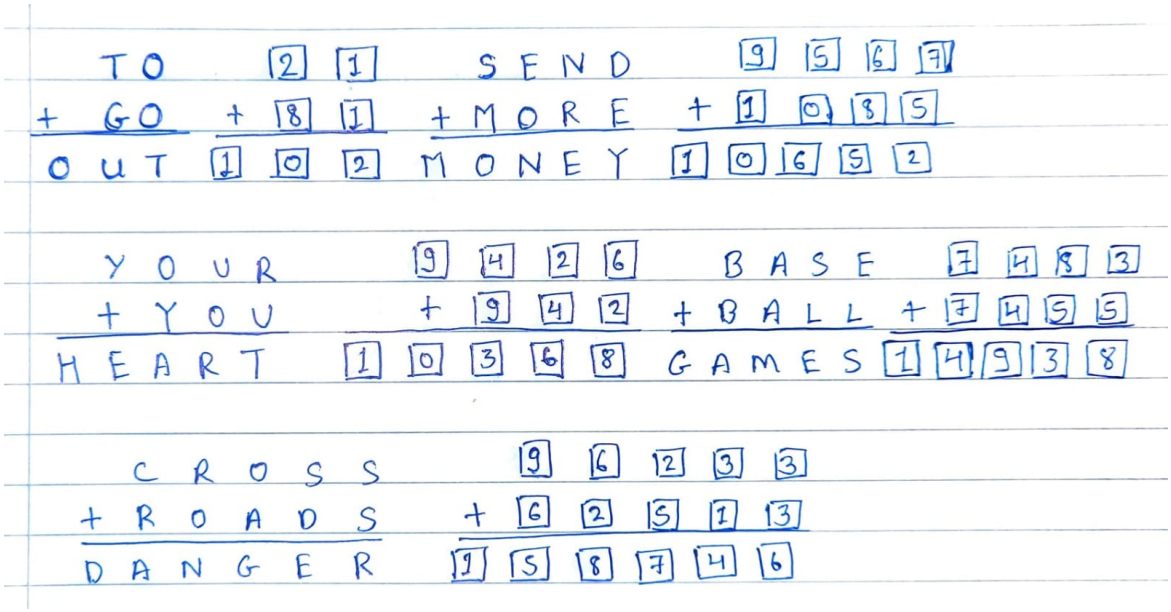

# Crypt-arithmetic problem

Cryptarithmetic is a type of mathematical puzzle in which a mathematical expression is written in a way that the digits are replaced with letters or symbols. The challenge of the puzzle is to decipher the values of each letter or symbol such that the expression is true.

Rules:

1. Digits range from 0 to 9 only.
2. Each variable should have unique and distinct value.
3. Each letter or symbol represents only one digit throughout the problem.
4. You have to find value of each letter in the crip arithmetic problem.
5. The numerical base unless specifically stated is 10.
6. After replacing letters by their digits the resulting arithmetic operations must be correct. Carry over can only be 1.

## ▼ Example




# Tautology, Contradiction, and Contingency

| p | q | p and q | p or q | p → q | p <-> q |
|---|---|---|---|---|---|
| T | T | T | T | T | T |
| T | F | F | T | F | F |
| F | T | F | T | T | F |
| F | F | F | F | T | T |

De Morgan's Law:

1. $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$
2. $\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$

# Inference

Inference is deriving conclusions form evidences. Evidences are premises or assumptions, based on the assumptions we are trying to derive conclusions.

## Types of Inference rules

1. **Modus Ponens**: $[(p \implies q) \wedge p] \implies q$
2. **Modus Tollens**: $[(p \implies q) \wedge \neg q] \implies \neg p$
3. **Hypothetical syllogism**: $[(p \implies q) \wedge (q \implies r)] \implies (p \implies r)$
4. **Disjunctive syllogism**: $[(p \vee q) \wedge \neg p] \implies q$
5. **Addition**: $p \implies (p \vee q)$
6. **Simplification**: $(p \wedge q) \implies p$ or $(p \wedge q) \implies q$
7. **Conjunction**: $[(p) \wedge (q)] \implies (p \wedge q)$

8. **Resolution**: $[(p \vee q) \wedge (\neg p \vee r)] \implies (q \vee r)$

# FOL

FOL uses predicates and quantifiers to express statements about objects and their relationships. FOL is widely used in AI applications such as knowledge representation and automated reasoning. FOL enables us to represent complex relationships between objects and infer new knowledge.

### Steps to convert Natural language to FOL

1. Identify the subject and main verb: The subject is the entity that the sentence is talking about, and the main verb is the action or state of being being expressed.
2. Identify the logical connectives: Logical connectives such as "and", "or", and "if-then" are used to connect propositions together in a sentence.
3. Identify the quantifiers: Quantifiers, such as "all" and "some", are used to specify the scope of the subject in the sentence.
4. Identify the predicates: Predicates represent properties or relationships between objects in the sentence.
5. Assign variables: Assign variables to represent the objects or individuals being referred to in the sentence. Make sure to use different variables for different objects.
6. Write the FOL statement: Using the information gathered from the previous steps, write the FOL statement for the sentence. Make sure to include the quantifiers, logical connectives, predicates, and variables in the appropriate places.

## Example

All dogs are mammals, but not all mammals are dogs.

1. Subject and main verb: Dogs are mammals.
2. Logical connective: "But not" indicates a negation.
3. Quantifier: "All" specifies the scope of the subject.
4. Predicate: "are mammals" represents the relationship between dogs and mammals.
5. Variables: Assign "x" to represent dogs and "y" to represent mammals.
6. FOL statement: ∀x (D(x) → M(x)) ∧ ¬∀x (M(x) → D(x))

# FOL to CNF

FOL to CNF is a process of converting FOL statements into Conjunctive Normal Form (CNF) to make them easier to process using automated reasoning tools.

### Steps to convert FOL to CNF

1. **Eliminate bi-conditional and implications statements**

a. Replace any bi-conditional statements (A ↔ B) with their equivalent statement [(A → B) ∧ (B → A)].

b. Replace any implications (A → B) with their equivalent statement (¬A ∨ B).

2. **Move negations inwards** using De Morgan's laws and double negation elimination

   a. Apply De Morgan's laws by distributing negation (¬) over the logical connectives (∧, ∨):

      i. ¬(A ∧ B) = ¬A ∨ ¬B

      ii. ¬(A ∨ B) = ¬A ∧ ¬B

   b. Replace any double negation (¬¬A) with A.

3. **Standardize variables**: Rename any variables that have the same name in different parts of the FOL statement to avoid confusion.

4. **Move quantifiers outward**: Move all the quantifiers to the left side without disturbing their order (Prenex Normal Form).

4. **Skolemize** (if needed): If the FOL statement has existential quantifiers (∃), Skolemize it by replacing them with Skolem functions.

5. **Drop universal quantifiers**: Just remove all the universal quantifiers from the statement.

5. **Distribute disjunctions over conjunctions**

   a. Distributive law:

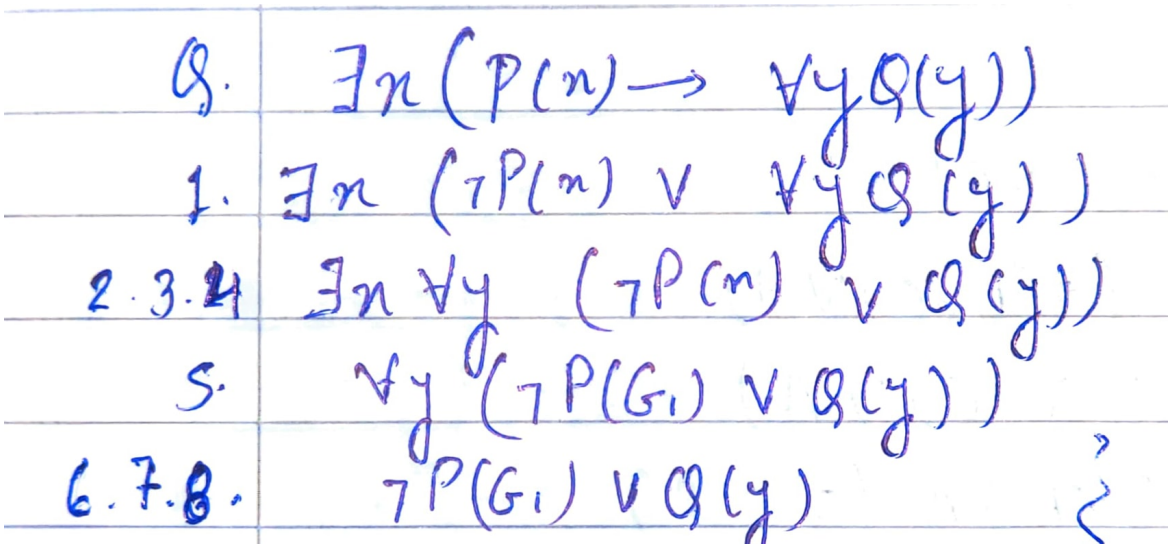
      - A ∧ (B ∨ C) ≡ (A ∧ B) ∨ (A ∧ C)
      - A ∨ (B ∧ C) ≡ (A ∨ B) ∧ (A ∨ C)

   b. Commutative law:

      - A ∧ B ≡ B ∧ A
      - A ∨ B ≡ B ∨ A

6. **Combine clauses**: Combine all the conjunctive clauses to get the CNF form.

## ▼ Example



# FOL Resolution

Resolution is a proof technique for propositional logic and first-order logic that allows us to determine the validity of a logical statement. It works by showing that a statement is true if and only if its negation leads to a contradiction.

It is also called proof by refutation. It is basically used to provide the satisfiability of a sentence.

### Steps to perform resolution

1. Convert the natural language into FOL (First Order Logic).
2. Convert FOL into CNF (Conjunction Normal Form).
3. Negate statement that is to be proved.
4. Draw resolution graph.

## Example

▼ Question 1

1. John likes all kind of food.
2. Apple and vegetable are food.
3. Anything anyone eats and not killed is food.
4. Anil eats peanuts and still alive.
5. Harry eats everything that Anil eats.
6. Goal: John likes peanuts.

▼ FOL:

a. $\forall x(Food(x) \rightarrow Likes(john, x))$

b. $Food(apple) \wedge Food(vegetable)$

c. $\forall x \forall y((Eats(x, y) \wedge \neg Killed(y)) \rightarrow Food(y))$

d. $Eats(anil, peanuts) \wedge Alive(anil)$

e. $\forall x(Eats(anil, x) \rightarrow Eats(harry, x))$

f. $\neg Killed(x) \rightarrow Alive(x)$

g. $Alive(x) \rightarrow \neg Killed(x)$

h. $Likes(john, peanuts)$

▼ CNF:

▼ Step 1: Remove bi-conditional and implications:

1. $\forall x (\neg Food(x) \vee Likes(john, x))$
2. $Food(apple) \wedge Food(vegetable)$
3. $\forall x \forall y \neg((Eats(x, y) \wedge \neg Killed(y)) \vee Food(y)$

4. Eats(anil, peanuts) ∧ Alive(anil)
5. ∀x ¬(Eats(anil, x) ∨ Eats(harry, x))
6. **¬(**¬Killed(x)) ∨ Alive(x)
7. ¬Alive(x) ∨ ¬Killed(x)
8. Likes(john, peanuts)

▼ Step 2: Move ¬ inwards:

1. ∀x (¬Food(x) ∨ Likes(john, x))
2. Food(apple) ∧ Food(vegetable)
3. ∀x∀y ((¬Eats(x, y) ∨ Killed(y)) ∨ Food(y)
4. Eats(anil, peanuts) ∧ Alive(anil)
5. ∀x (¬Eats(anil, x) ∨ Eats(harry, x))
6. Killed(x) ∨ Alive(x)
7. ¬Alive(x) ∨ ¬Killed(x)
8. Likes(john, peanuts)

▼ Step 3: Standardize variables:

1. ∀x (¬Food(x) ∨ Likes(john, x))
2. Food(apple) ∧ Food(vegetable)
3. ∀y∀z ((¬Eats(y, z) ∨ Killed(y)) ∨ Food(z)
4. Eats(anil, peanuts) ∧ Alive(anil)
5. ∀w (¬Eats(anil, w) ∨ Eats(harry, w))
6. Killed(g) ∨ Alive(g)
7. ¬Alive(k) ∨ ¬Killed(k)
8. Likes(john, peanuts)

▼ Step 4 and 5 are skipped

▼ Step 6: Drop universal quantifiers:

1. ¬Food(x) ∨ Likes(john, x)
2. Food(apple)
3. Food(vegetable)
4. ¬Eats(y, z) ∨ Killed(y) ∨ Food(z)
5. Eats(anil, peanuts)
6. Alive(anil)
7. ¬Eats(anil, w) ∨ Eats(harry, w)
8. Killed(g) ∨ Alive(g)

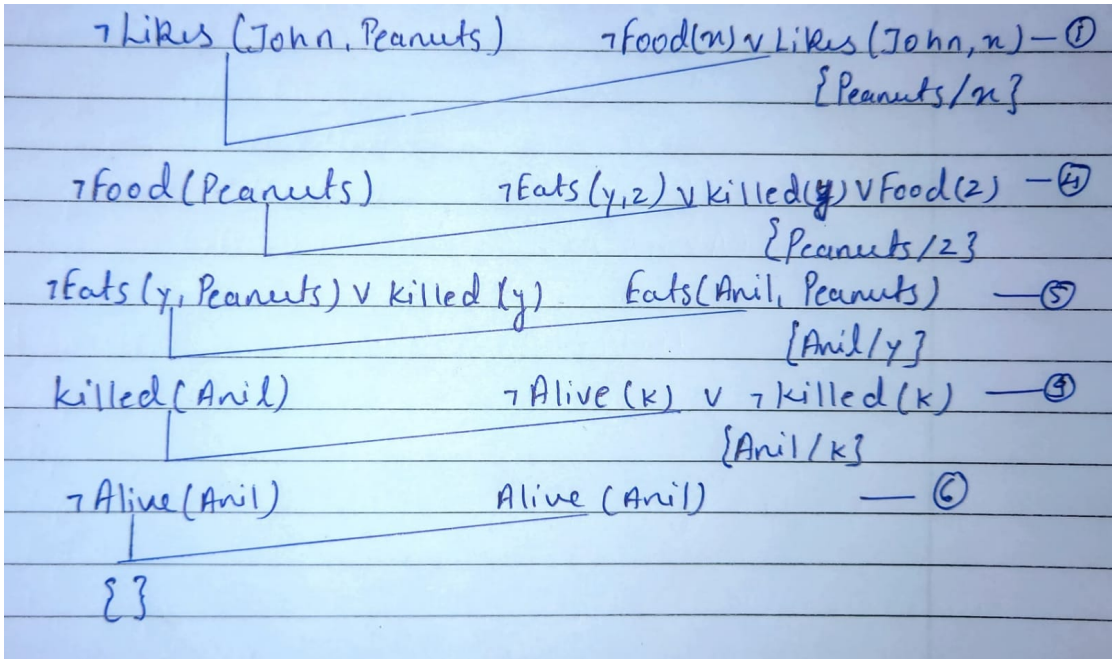
9. ¬Alive(k) ∨ ¬Killed(k)
10. Likes(john, peanuts)

▼ Step 7: is skipped

▼ Negate statement to be proved:

¬Likes(john, peanuts)

▼ Draw resolution graph:




Hence proved.

▼ Question 2

1. If it is sunny and warm day you will enjoy.
2. If it is raining you will get wet.
3. It is warm day.
4. It is raining.
5. It is sunny.
6. Goal: You will enjoy.

▼ FOL:

1. (Sunny ∧ Warm) → Enjoy
2. Raining → Wet
3. Warm
4. Raining
5. Sunny

6. Enjoy

▼ CNF:

▼ Step 1: Eliminate bi-conditional and implications

1. ¬(Sunny ∧ Warm) ∨ Enjoy
2. ¬Raining ∨ Wet
3. Warm
4. Raining
5. Sunny
6. Enjoy

▼ Step 2: Negation inward

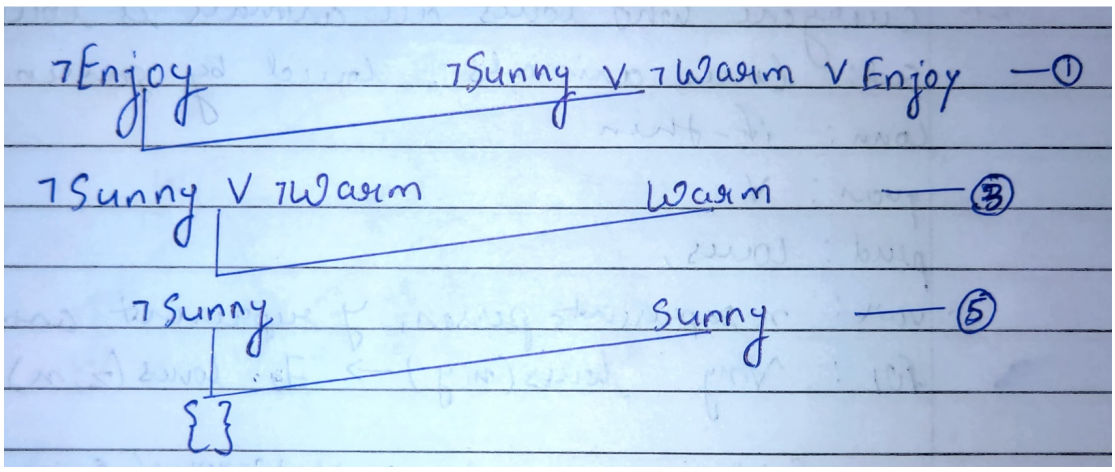
1. ¬Sunny ∨ ¬Warm ∨ Enjoy
2. ¬Raining ∨ Wet
3. Warm
4. Raining
5. Sunny
6. Enjoy

▼ Step 3, 4, 5, 6, and 7 are skipped

▼ Negate

¬Enjoy

▼ Draw Resolution graph




Hence proved.