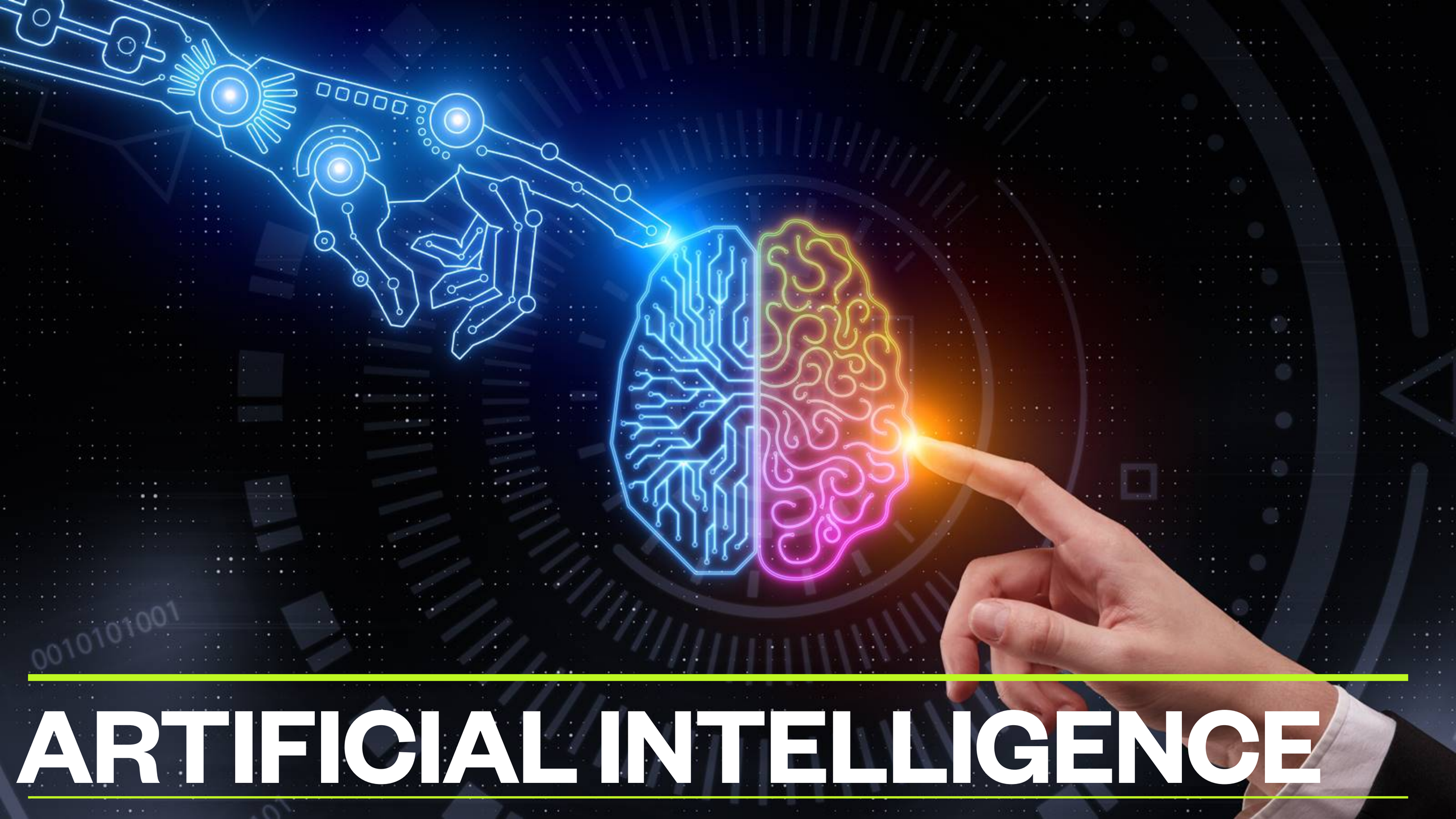


ARTIFICIAL INTELLIGENCE

Constraint Satisfaction

- **Constraint satisfaction is a technique where a problem is solved when its values satisfy certain constraints or rules of the problem.** Such type of technique leads to a deeper understanding of the problem structure as well as its complexity.
- Constraint satisfaction depends on three components, namely:

    1. **V:** It is a set of variables.
    2. **D:** It is a set of domains where the variables reside. There is a specific domain for each variable.
    3. **C:** It is a set of constraints which are followed by the set of variables.

- The constraint value consists of a pair of **{scope, rel}**. The **scope** is a tuple of variables which participate in the constraint and **rel** is a relation which includes a list of values which the variables can take to satisfy the constraints of the problem.

Solving Constraint Satisfaction Problems

The requirements to solve a constraint satisfaction problem (CSP) is:

- A state-space
- The notion of the solution.

A state in state-space is defined by assigning values to some or all variables such as

{X1=v1, X2=v2, and so on…}

An assignment of values to a variable can be done in three ways:

1. **Consistent or Legal Assignment:** An assignment which does not violate any constraint or rule is called Consistent or legal assignment.
2. **Complete Assignment:** An assignment where every variable is assigned with a value, and the solution to the CSP remains consistent. Such assignment is known as Complete assignment.
3. **Partial Assignment:** An assignment which assigns values to some of the variables only. Such type of assignments are called Partial assignments.

Types of Domains in CSP

- There are following two types of domains which are used by the variables :

1. **Discrete Domain:** It is an infinite domain which can have one state for multiple variables. **For example,** a start state can be allocated infinite times for each variable.

2. **Finite Domain:** It is a finite domain which can have continuous states describing one domain for one specific variable. It is also called a continuous domain.

Constraint Types in CSP

With respect to the variables, basically there are following types of constraints:

1. **Unary Constraints:** It is the simplest type of constraints that restricts the value of a single variable.

2. **Binary Constraints:** It is the constraint type which relates two variables. A value **x2** will contain a value which lies between **x1** and **x3**.

3. **Global Constraints:** It is the constraint type which involves an arbitrary number of variables.

Some special types of solution algorithms are used to solve the following types of constraints:

- **Linear Constraints:** These type of constraints are commonly used in linear programming where each variable containing an integer value exists in linear form only.

- **Non-linear Constraints:** These type of constraints are used in non-linear programming where each variable (an integer value) exists in a non-linear form.

- **Note:** A special constraint which works in real-world is known as **Preference constraint.**

Constraint Propagation

In local state-spaces, the choice is only one, i.e., to search for a solution. But in CSP, we have two choices either:

- We can search for a solution or
- We can perform a special type of inference called **constraint propagation**.
- Constraint propagation is a special type of inference which helps in reducing the legal number of values for the variables. The idea behind constraint propagation is **local consistency.**
- In local consistency, variables are treated as nodes, and each binary constraint is treated as an arc in the given problem. There are following local consistencies which are discussed below:

1. **Node Consistency:** A single variable is said to be node consistent if all the values in the variable's domain satisfy the unary constraints on the variables.

2. **Arc Consistency:** A variable is arc consistent if every value in its domain satisfies the binary constraints of the variables.

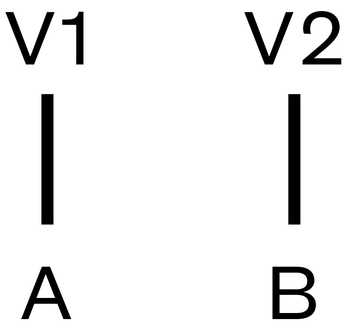
3. **Path Consistency:** When the evaluation of a set of two variable with respect to a third variable can be extended over another variable, satisfying all the binary constraints. It is similar to arc consistency.

4. **k-consistency:** This type of consistency is used to define the notion of stronger forms of propagation. Here, we examine the k-consistency of the variables.

CSPs can be used to solve problems such as

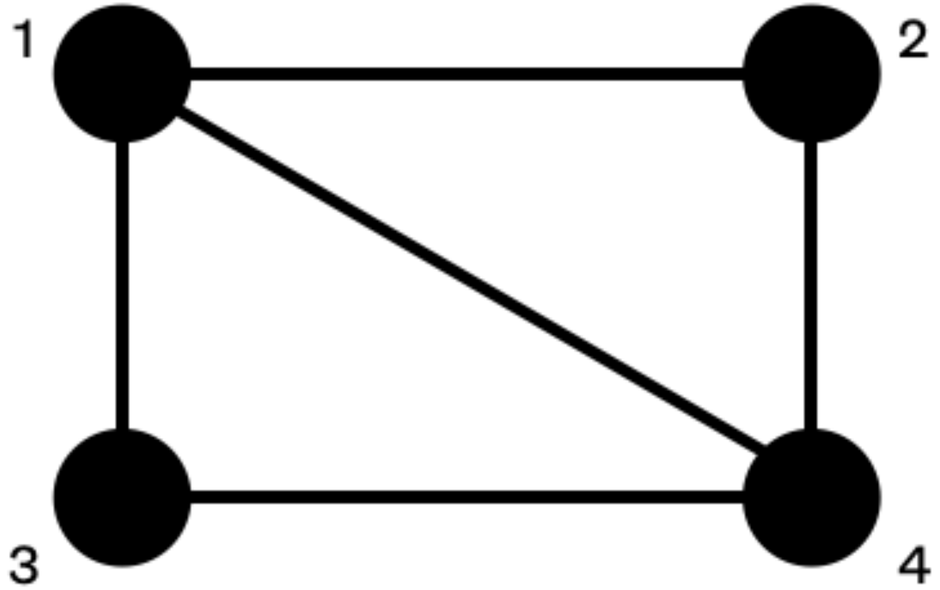
- **graph-colouring**: given a graph, the a colouring of the graph means assigning each of its vertices a colour such that no pair of vertices connected by an edge have the same color
    - in general, this is a very hard problem, e.g. determining if a graph can be colored with 3 colours is NP-hard
    - many problems boil down to graph coloring, or related problems
- **job shop scheduling**: e.g. suppose you need to complete a set of tasks, each of which have a duration, and constraints upon when they start and stop (e.g. task c can't start until both task a and task b are finished)
    - CSPs are a natural way to express such problems
- **crypt-arithmetic puzzles**: e.g. suppose you are told that TWO + TWO = FOUR, and each of the letters corresponds to a different digit from 0 to 9, and that a number can't start with 0 (so T and F are not 0); what, if any, are the possible values for the letters?
    - while these are not directly useful problems, they are a simple test case for CSP solvers

- Values of V1 and V2 cant be same.
- C={(V1,V2),(V1≠V2)}



Example :

- Set of variables : V={1,2,3,4}
- D={R,G,B}
- C={1 ≠ 2,1 ≠ 3,1 ≠ 4,2 ≠ 4,3 ≠ 4}



	1	2	3	4
Initial Domain	r,g,b	r,g,b	r,g,b	r,g,b
1=r	r	g,b	g,b	g,b
2=g	r	g	g,b	b
3=b	r	g	b(after backtracking G)	b

CRYPT ARITHMETIC

- Type of CSF
- **Constraints :**
    1. Digits that can be assigned to a word/alphabet can be from 0 to 9 (range).
    2. No two letters have same value.
    3. Sum of digits must be as shown in the problem.
    4. There should be only one carry forward.

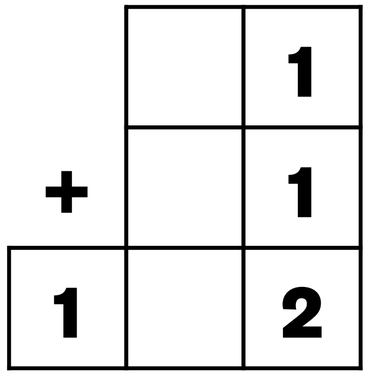
Example 1

- Start with left most digit , always = 1 (can never be 0)
- Hence, value of O =1

	T	O	
+	G	O	
O	U	T	

		1
+		1
1		

- O+O = 1+1 = 2 , Therefore value of T=2



Letter	Digit
T	2
O	1
G	
U	

2+G = U

Here, we have a carry thats why the value of O is 1

2+G=U+10 (either the value of G will be 8 or 9)

2+9=11 (not possible because the value 1 is already assigned to O)

2+8=10

so, the value of **G will be 8**

	2	**1**
+	**8**	**1**
1	**0**	**2**

Letter	Digit
T	2
O	1
G	8
U	0

Example 2

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

+	1			
1				

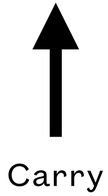
- If M=1 and there is carry, that means S+M ≥10 , Therefore **S=9 and O=0**
- E+O=N (O is without carry that means E≠N) which means carry =1
- Let E=5
- E+0+C2=N —> 5+1=**6 = N**

1	**c3**	**c2 =1**	**c1**	
	9	**5**	**6**	
+	**1**	**0**		**5**
1	**0**	**6**	**5**	

- N+R=E
- 6+R = 5
- R can be 9 — **NO** , why ?

1	c3	c2 =1	c1	
	9	5	6	
+	1	0		5
1	0	6	5	

- N+R=E
- 6+R = 5
- R can be 9 — **NO** , why ?(assigned to S already)
- So we can convert the value with a carry:
- 6 +**8** +1 =**1**5



Carry

1	c3	c2 =1	c1	
	9	5	6	
+	1	0	8	5
1	0	6	5	

- N+R=E
- 6+R = 5
- R can be 9 — **NO** , why ?(assigned to S already)
- So we can convert the value with a carry:
- 6 +**8** +1 =**1**5
- D+E=Y (providing the carry)
- **D =7**

1	c3	c2 =1	c1 =1	
	9	5	6	7
+	1	0	8	5
1	0	6	5	

1	c3	c2 =1	c1 =1	
	9	5	6	7
+	1	0	8	5
1	0	6	5	2

Alpha -Beta pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.

- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.

- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.

- The two-parameter can be defined as:

    1. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.

    2. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.

- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

- **Condition for Alpha-beta pruning:**

The main condition which required for alpha-beta pruning is:

$$\alpha >= \beta$$

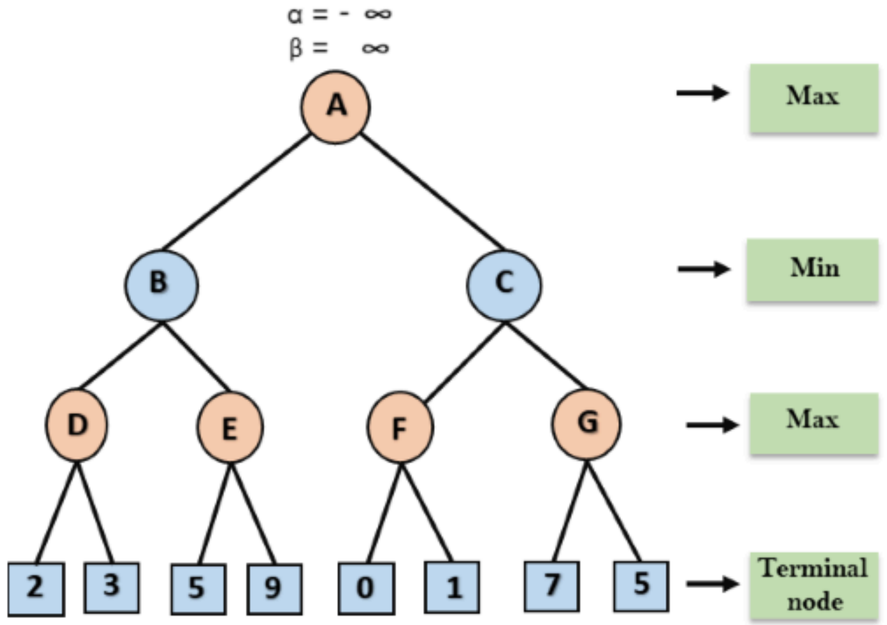
- **Key points about alpha-beta pruning:**

1. The Max player will only update the value of alpha.
2. The Min player will only update the value of beta.
3. While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
4. We will only pass the alpha, beta values to the child nodes.

Working :

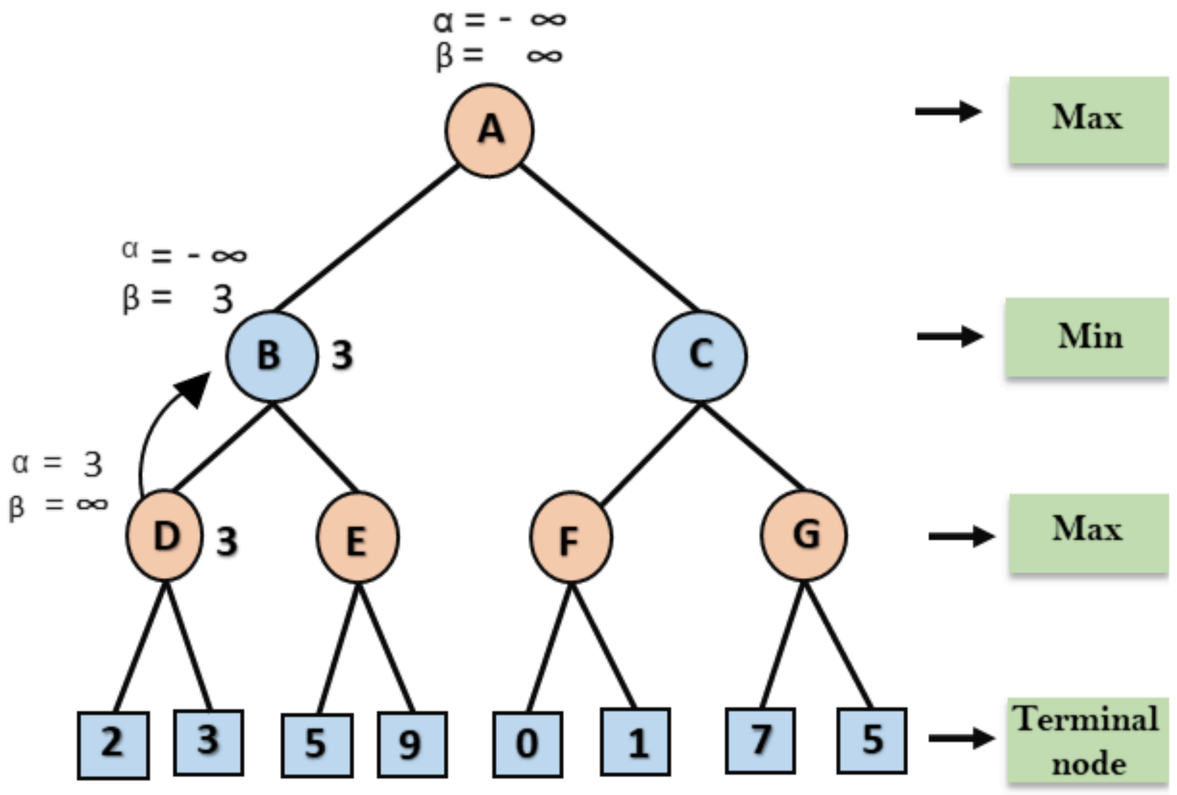
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

- **Step 1:** At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.

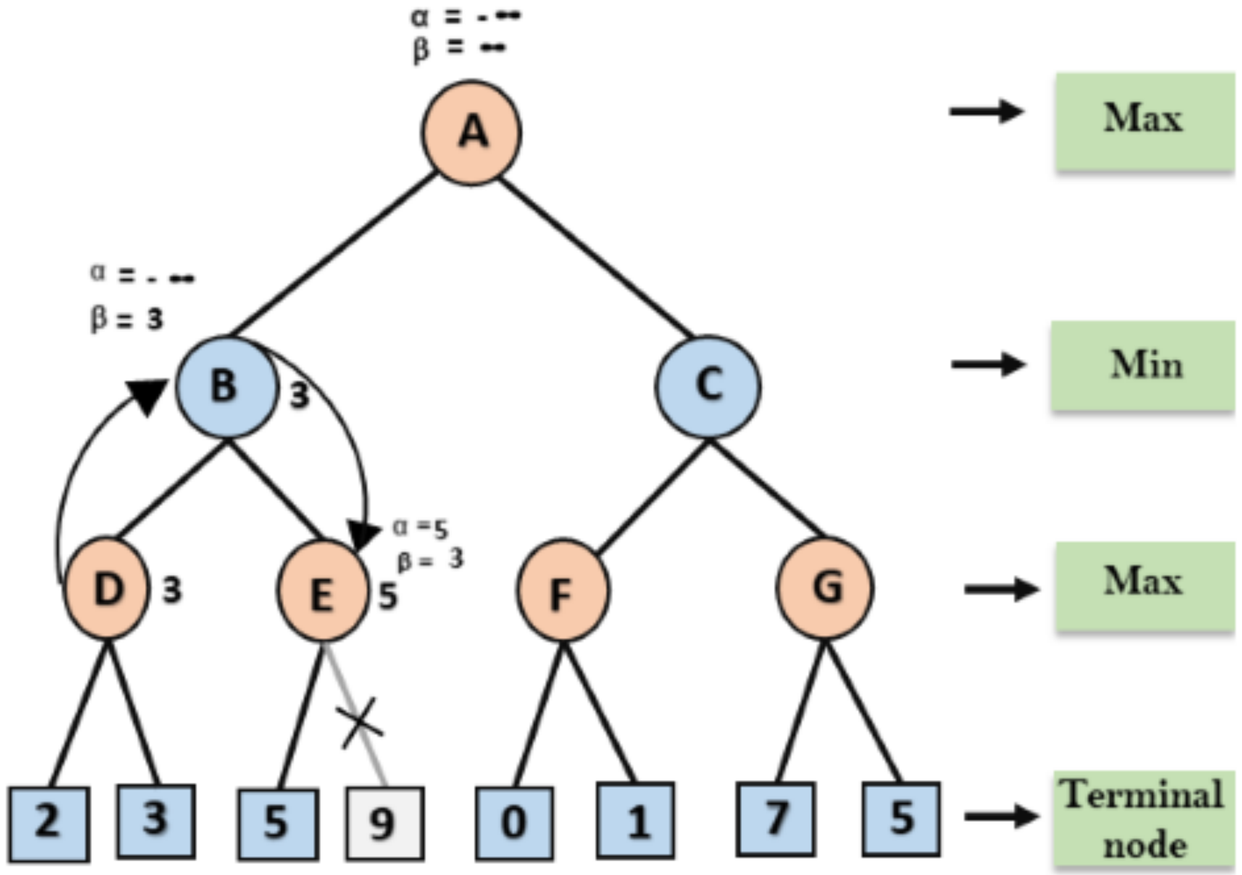


- **Step 2:** At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

- **Step 3:** Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.

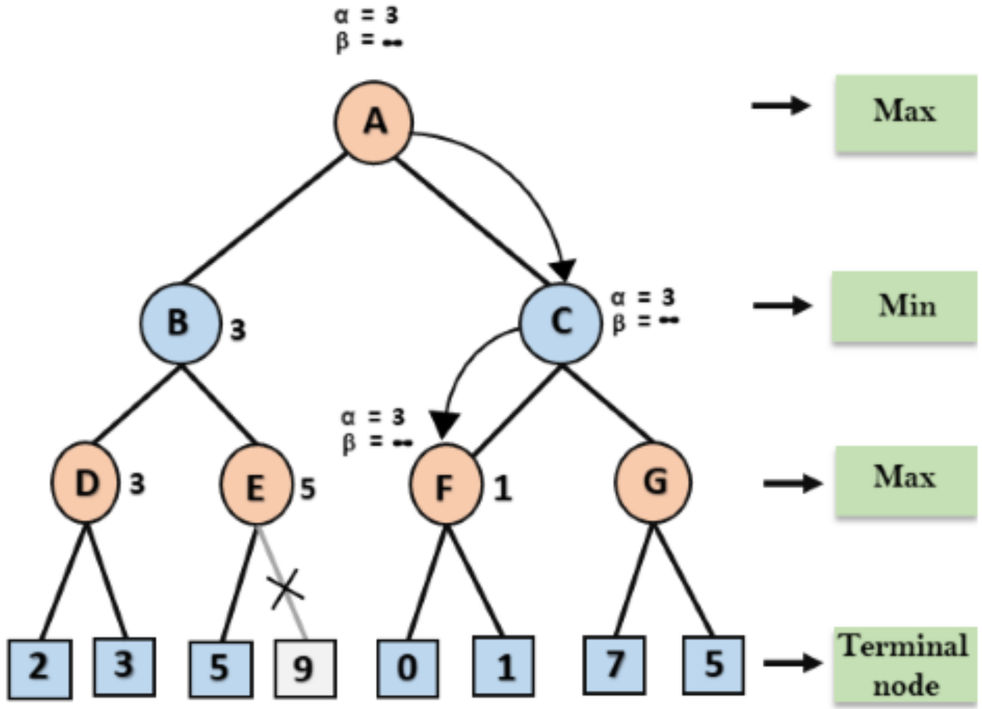


- **Step 4:** At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.

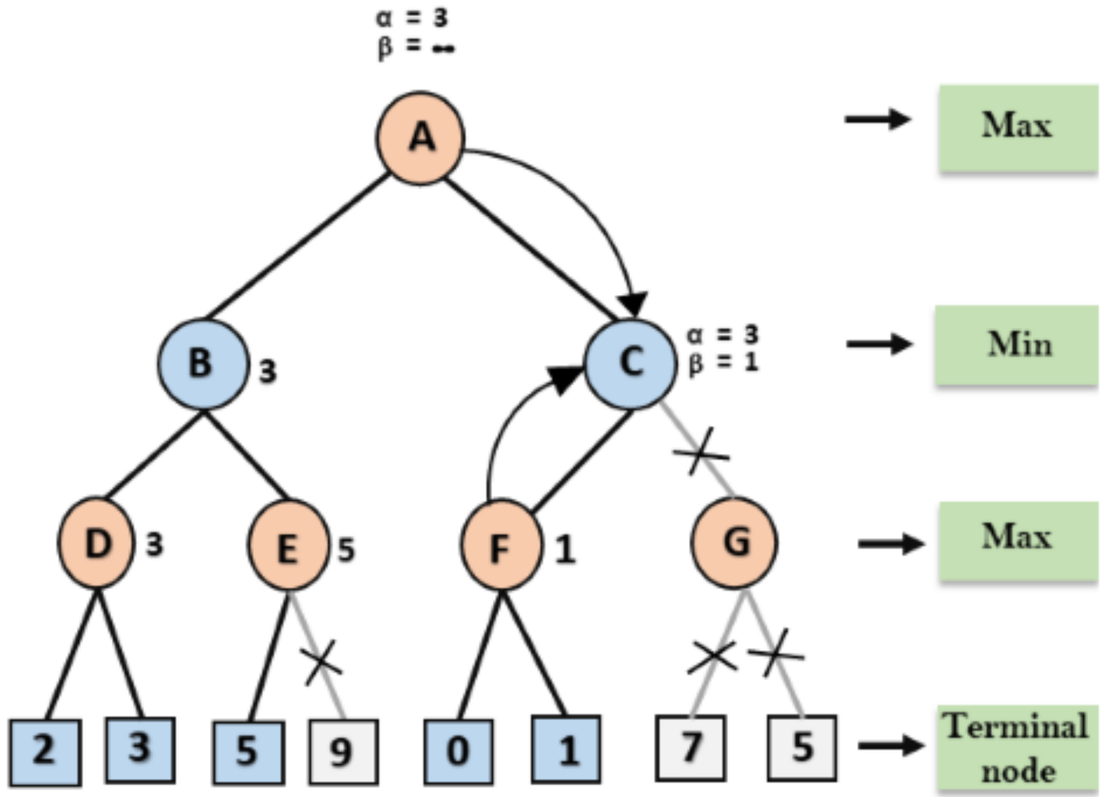


- **Step 5:** At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.At node C, α=3 and β= +∞, and the same values will be passed on to node F.

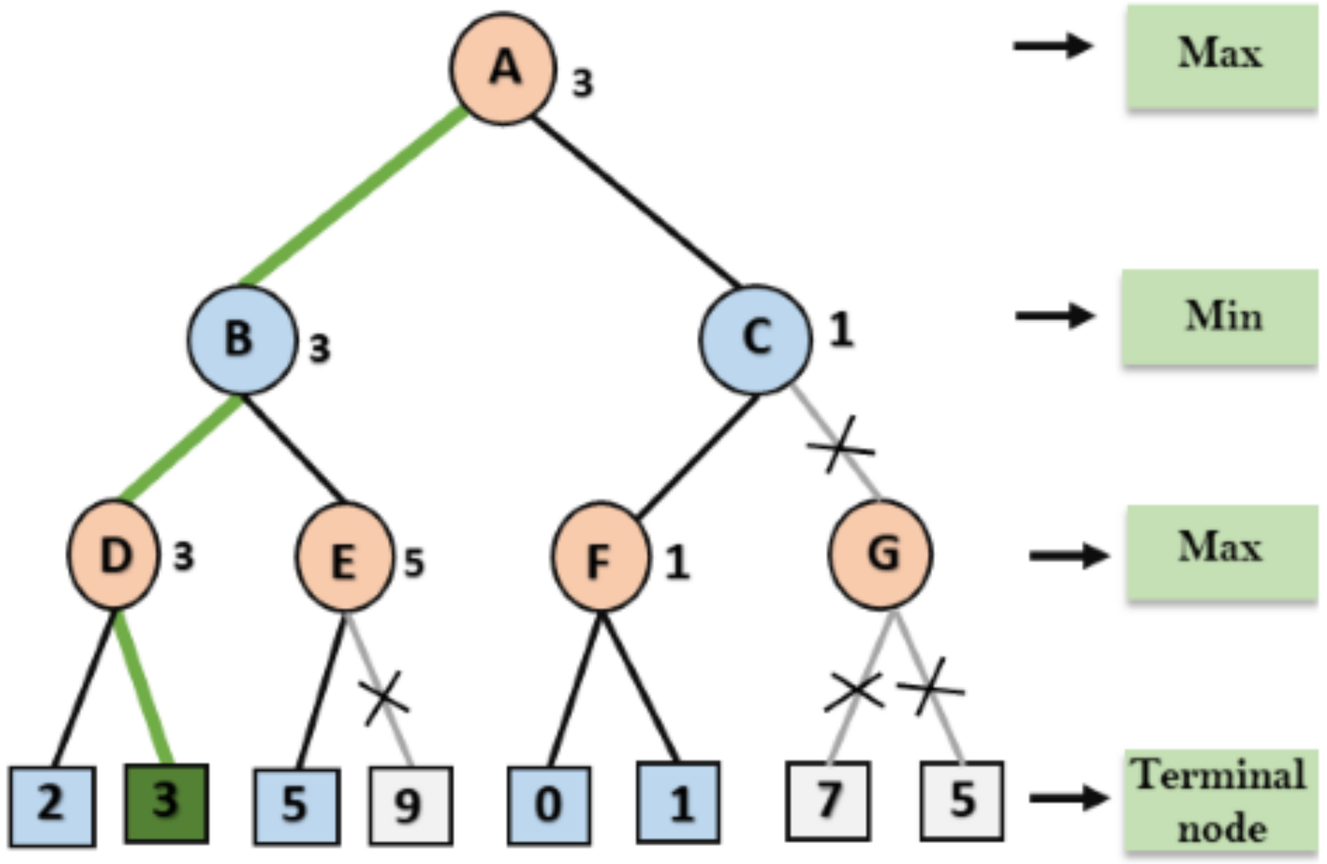
- **Step 6:** At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1



- **Step 7:** Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



- **Step 8:** C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



Move Ordering in Alpha-Beta pruning:

- The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.
- It can be of two types:

1. **Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.
2. **Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.

Rules to find good ordering:

- Following are some rules to find good ordering in alpha-beta pruning:

    1. Occur the best move from the shallowest node.
    2. Order the nodes in the tree such that the best nodes are checked first.
    3. Use domain knowledge while finding the best move. Ex: for Chess, try order: captures first, then threats, then forward moves, backward moves.
    4. We can bookkeep the states, as there is a possibility that states may repeat.