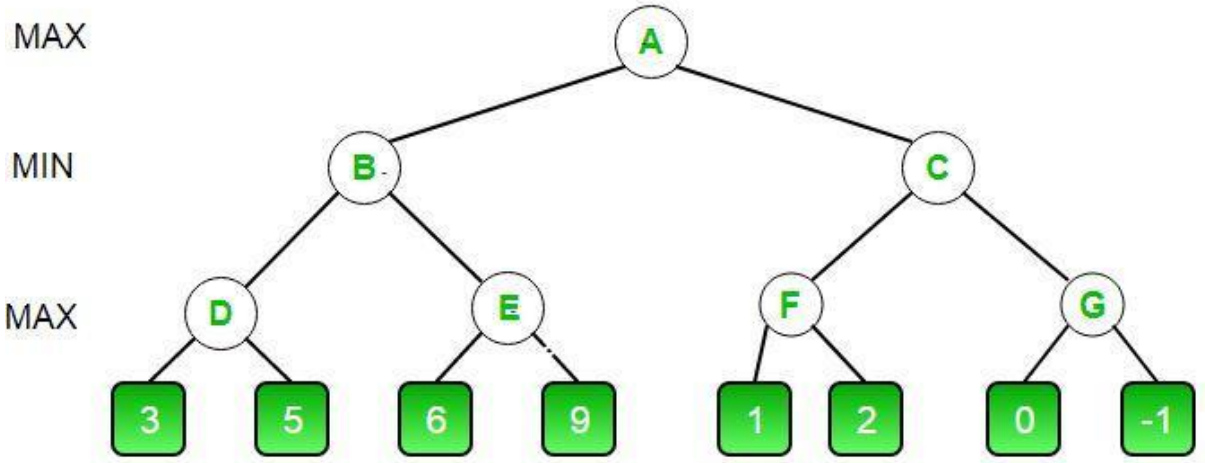
Minimax Algorithm in Game Theory (Alpha-Beta Pruning)

Alpha-Beta pruning is not actually a new algorithm, but rather an optimization technique for the minimax algorithm. It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available. It is called Alpha-Beta pruning because it passes 2 extra parameters in the minimax function, namely alpha and beta.
Let's define the parameters alpha and beta.
Alpha is the best value that the **maximizer** currently can guarantee at that level or above.
Beta is the best value that the **minimizer** currently can guarantee at that level or below.

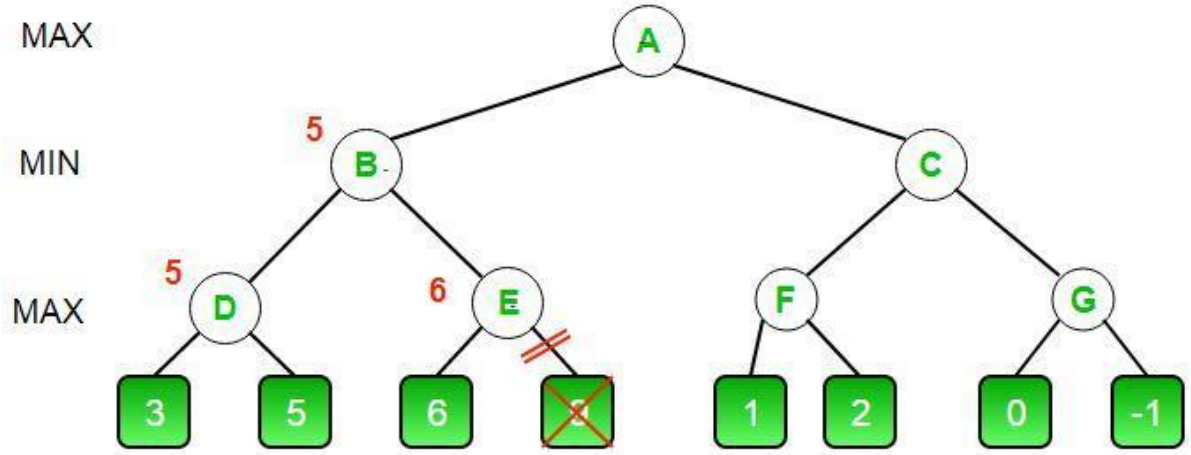
Pseudocode :

```
function minimax(node, depth, isMaximizingPlayer, alpha, beta):

    if node is a leaf node :
        return value of the node

    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each child node :
            value = minimax(node, depth+1, false, alpha, beta)
            bestVal = max( bestVal, value)
            alpha = max( alpha, bestVal)
            if beta <= alpha:
                break
        return bestVal

    else :
        bestVal = +INFINITY
        for each child node :
            value = minimax(node, depth+1, true, alpha, beta)
            bestVal = min( bestVal, value)
            beta = min( beta, bestVal)
            if beta <= alpha:
                break
        return bestVal

// Calling the function for the first time.
minimax(0, 0, true, -INFINITY, +INFINITY)
```



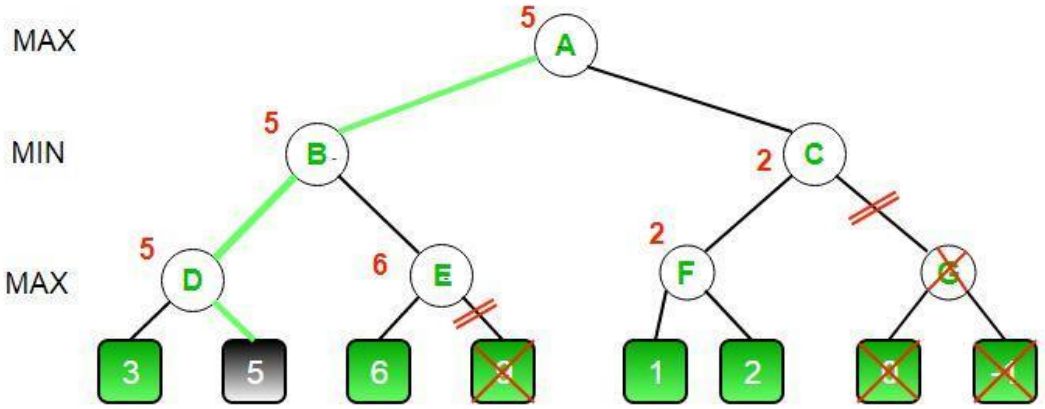
- The initial call starts from **A**. The value of alpha here is **-INFINITY** and the value of beta is **+INFINITY**. These values are passed down to subsequent nodes in the tree. At **A** the maximizer must choose max of **B** and **C**, so **A** calls **B** first
- At **B** it the minimizer must choose min of **D** and **E** and hence calls **D** first.
- At **D**, it looks at its left child which is a leaf node. This node returns a value of 3. Now the value of alpha at **D** is max(-INF, 3) which is 3.
- To decide whether its worth looking at its right node or not, it checks the condition beta<=alpha. This is false since beta = +INF and alpha = 3. So it continues the search.
- **D** now looks at its right child which returns a value of 5.At **D**, alpha = max(3, 5) which is 5. Now the value of node **D** is 5
- **D** returns a value of 5 to **B**. At **B**, beta = min(+INF, 5) which is 5. The minimizer is now guaranteed a value of 5 or lesser. **B** now calls **E** to see if he can get a lower value than 5.
- At **E** the values of alpha and beta is not -INF and +INF but instead -INF and 5 respectively, because the value of beta was changed at **B** and that is what **B** passed down to **E**
- Now **E** looks at its left child which is 6. At **E**, alpha = max(-INF, 6) which is 6. Here the condition becomes true. beta is 5 and alpha is 6. So beta<=alpha is true. Hence it breaks and **E** returns 6 to **B**
- Note how it did not matter what the value of **E**'s right child is. It could have been +INF or -INF, it still wouldn't matter, We never even had to look at it because the minimizer was guaranteed a value of 5 or lesser. So as soon as the maximizer saw the 6 he knew the minimizer would never come this way because he can get a 5 on the left side of **B**. This way we didn't have to look at that 9 and hence saved computation time.
- **E** returns a value of 6 to **B**. At **B**, beta = min(5, 6) which is 5.The value of node **B** is also 5

So far this is how our game tree looks. The 9 is crossed out because it was never computed.



- **B** returns 5 to **A**. At **A**, alpha = max(-INF, 5) which is 5. Now the maximizer is guaranteed a value of 5 or greater. **A** now calls **C** to see if it can get a higher value than 5.
- At **C**, alpha = 5 and beta = +INF. **C** calls **F**
- At **F**, alpha = 5 and beta = +INF. **F** looks at its left child which is a 1. alpha = max(5, 1) which is still 5.
- **F** looks at its right child which is a 2. Hence the best value of this node is 2. Alpha still remains 5
- **F** returns a value of 2 to **C**. At **C**, beta = min(+INF, 2). The condition beta <= alpha becomes true as beta = 2 and alpha = 5. So it breaks and it does not even have to compute the entire sub-tree of **G**.
- The intuition behind this break-off is that, at **C** the minimizer was guaranteed a value of 2 or lesser. But the maximizer was already guaranteed a value of 5 if he choose **B**. So why would the maximizer ever choose **C** and get a value less than 2 ? Again you can see that it did not matter what those last 2 values were. We also saved a lot of computation by skipping a whole sub-tree.
- **C** now returns a value of 2 to **A**. Therefore the best value at **A** is max(5, 2) which is a 5.
- Hence the optimal value that the maximizer can get is 5

This is how our final game tree looks like. As you can see **G** has been crossed out as it was never computed.



Output

```
The optimal value is : 5
```

A constraint satisfaction problem (CSP) is **a problem that requires its solution to be within some limitations or conditions, also known as constraints, consisting of a finite variable set, a domain set and a finite constraint set**

http://www.sfu.ca/~tjd/310summer2019/chp6_csp.html

The term backtracking search is used for **a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign**.

What is Backtracking Algorithm?

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

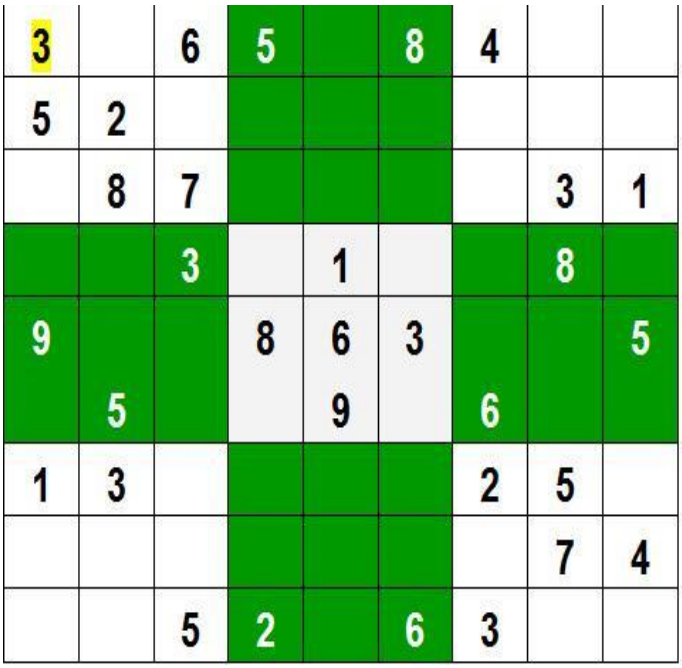
Types of Backtracking Algorithm

There are three types of problems in backtracking

1. Decision Problem – In this, we search for a feasible solution.
2. Optimization Problem – In this, we search for the best solution.
3. Enumeration Problem – In this, we find all feasible solutions.

When can be Backtracking Algorithm used?

For example, consider the SudoKo solving Problem, we try filling digits one by one. Whenever we find that current digit cannot lead to a solution, we remove it (backtrack) and try next digit. This is better than naive approach (generating all possible combinations of digits and then trying every combination one by one) as it drops a set of permutations whenever it backtracks.



Standard Problems on Backtracking:

1. The Knight's tour problem
2. Rat in a Maze
3. N Queen Problem | Backtracking-3
4. Subset Sum problem
5. m Coloring Problem
6. Hamiltonian Cycle
7. Sudoku | Backtracking-7
8. Magnet Puzzle
9. Remove Invalid Parentheses
10. A backtracking approach to generate n bit Gray Codes
11. Write a program to print all permutations of a given string

Given a string of digits, determine whether it is a 'sum-string'. A string S is called a sum-string if the rightmost substring can be written as the sum of two substrings before it and the same is recursively true for substrings before it.

```
"12243660" is a sum string.
Explanation : 24 + 36 = 60, 12 + 24 = 36

"1111112223" is a sum string.
Explanation: 111+112 = 223, 1+111 = 112

"2368" is not a sum string
```

In general, a string S is called sum-string if it satisfies the following properties:

```
sub-string(i, x) + sub-string(x+1, j)
 = sub-string(j+1, l)
```

```
and
sub-string(x+1, j)+sub-string(j+1, l)
 = sub-string(l+1, m)
and so on till end.
```

From the examples, we can see that our decision depends on the first two chosen numbers. So we choose all possible first two numbers for a given string. Then for every chosen two numbers, we check whether it is sum-string or not? So the approach is very simple. We generate all possible first two numbers using two substrings s1 and s2 using two loops. then we check whether it is possible to make the number s3 = (s1 + s2) or not. If we can make s3 then we recursively check for s2 + s3 and so on.

```
// C++ program to check if a given string
// is sum-string or not
#include <bits/stdc++.h>
using namespace std;

// this is function for finding sum of two
// numbers as string
string string_sum(string str1, string str2)
{
    if (str1.size() < str2.size())
        swap(str1, str2);

    int m = str1.size();
    int n = str2.size();
    string ans = "";

    // sum the str2 with str1
    int carry = 0;
    for (int i = 0; i < n; i++) {


        // Sum of current digits
        int ds = ((str1[m - 1 - i] - '0')
                  + (str2[n - 1 - i] - '0') + carry)
                 % 10;

        carry = ((str1[m - 1 - i] - '0')
                 + (str2[n - 1 - i] - '0') + carry)
                / 10;

        ans = char(ds + '0') + ans;
    }

    for (int i = n; i < m; i++) {
        int ds = (str1[m - 1 - i] - '0' + carry) % 10;
        carry = (str1[m - 1 - i] - '0' + carry) / 10;
        ans = char(ds + '0') + ans;
    }

    if (carry)
        ans = char(carry + '0') + ans;
    return ans;
```

```
}

// Returns true if two substrings of given
// lengths of str[beg..] can cause a positive
// result.
bool checkSumStrUtil(string str, int beg, int len1,
                     int len2)
{

    // Finding two substrings of given lengths
    // and their sum
    string s1 = str.substr(beg, len1);
    string s2 = str.substr(beg + len1, len2);
    string s3 = string_sum(s1, s2);

    int s3_len = s3.size();

    // if number of digits s3 is greater than
    // the available string size
    if (s3_len > str.size() - len1 - len2 - beg)
        return false;

    // we got s3 as next number in main string
    if (s3 == str.substr(beg + len1 + len2, s3_len)) {

        // if we reach at the end of the string
        if (beg + len1 + len2 + s3_len == str.size())
            return true;

        // otherwise call recursively for n2, s3
        return checkSumStrUtil(str, beg + len1, len2,
                               s3_len);
    }

    // we do not get s3 in main string
    return false;
}

// Returns true if str is sum string, else false.
bool isSumStr(string str)
{
    int n = str.size();

    // choosing first two numbers and checking
    // whether it is sum-string or not.
    for (int i = 1; i < n; i++)
        for (int j = 1; i + j < n; j++)
            if (checkSumStrUtil(str, 0, i, j))
                return true;

    return false;
}

// Driver code
```

```
int main()
{
    bool result;

    result = isSumStr("1212243660");
    cout << (result == 1 ? "True\n" : "False\n");

    result = isSumStr("123456787");
    cout << (result == 1 ? "True\n" : "False\n");
    return 0;
}
```

Output
Is True or False

Knowlegde Based Agents

- Intelligent agents need knowledge about the world for making good decisions.
- The knowledge of an agent is stored in a knowledge base in the form of **sentences** in a knowledge representation language.
- A knowledge-based agent needs a **knowledge base** and an **inference mechanism**. It operates by storing sentences in its knowledge base, inferring new sentences with the inference mechanism, and using them to deduce which actions to take.
- A **representation language** is defined by its syntax and semantics, which specify the structure of sentences and how they relate to the facts of the world.
- The **interpretation** of a sentence is the fact to which it refers. If this fact is part of the actual world, then the sentence is true.

https://www.slideshare.net/anniyappa/knowledge-based-agents-250261883

What is knowledge representation?

Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world.

But how machines do all these things comes under knowledge representation and reasoning. Hence we can describe Knowledge representation as following:

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.
- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.

- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

What to Represent:

Following are the kind of knowledge which needs to be represented in AI systems:

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language)

Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

Types of knowledge

1. Declarative Knowledge:

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.

- It is also called descriptive knowledge and expressed in declarativesentences.
- It is simpler than procedural language.

2. Procedural Knowledge

- It is also known as imperative knowledge.
- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.

4. Heuristic knowledge:

- Heuristic knowledge is representing knowledge of some experts in a filed or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

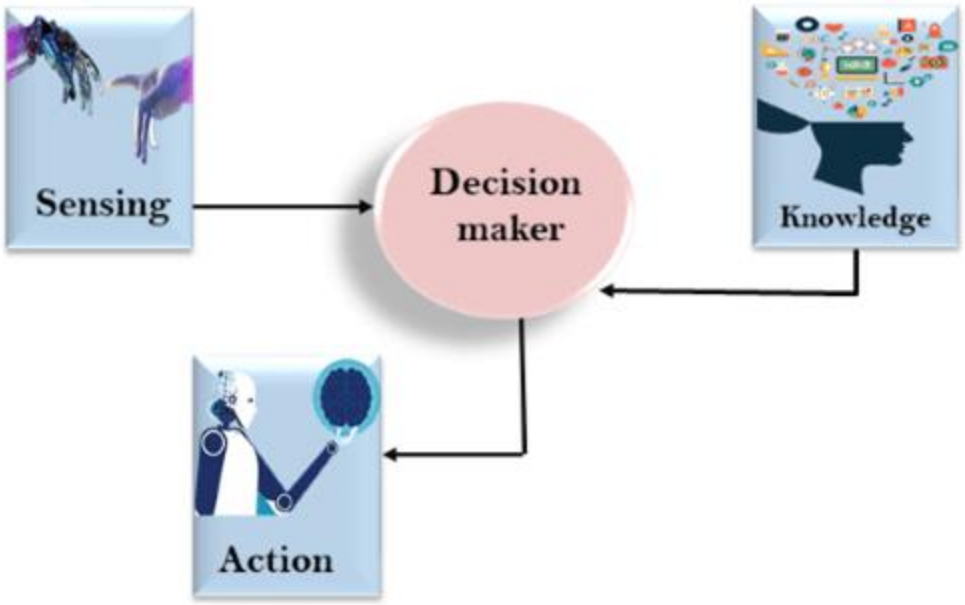
5. Structural knowledge:

- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

The relation between knowledge and intelligence:

Knowledge of real-worlds plays a vital role in intelligence and same for creating artificial intelligence. Knowledge plays an important role in demonstrating intelligent behavior in AI agents. An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

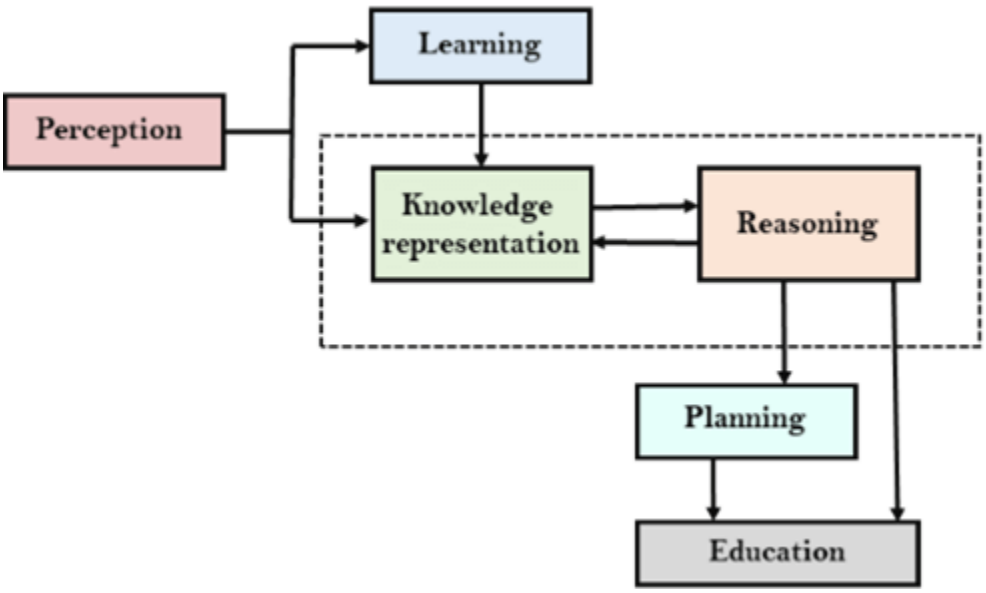
Let's suppose if you met some person who is speaking in a language which you don't know, then how you will able to act on that. The same thing applies to the intelligent behavior of the agents. As we can see in below diagram, there is one decision maker which act by sensing the environment and using knowledge. But if the knowledge part will not present then, it cannot display intelligent behavior.



AI knowledge cycle:

An Artificial intelligence system has the following components for displaying intelligent behavior:

- Perception
- Learning
- Knowledge Representation and Reasoning
- Planning
- Execution



The above diagram is showing how an AI system can interact with the real world and what components help it to show intelligence. AI system has Perception component by which it retrieves information from its environment. It can be visual, audio or another form of sensory input. The learning component is responsible for learning from data captured by Perception comportment. In the complete cycle, the main components are knowledge representation and Reasoning. These two components are involved in showing the intelligence in machine-like humans. These two components are independent with each other but also coupled together. The planning and execution depend on analysis of Knowledge representation and reasoning.

Approaches to knowledge representation:

There are mainly four approaches to knowledge representation, which are givenbelow:

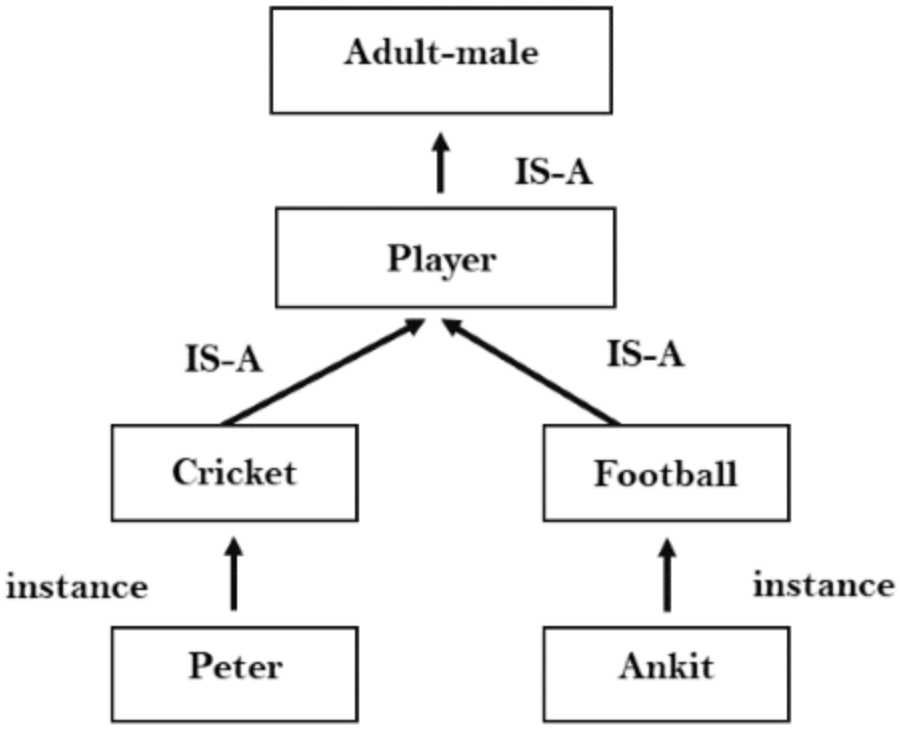
1. Simple relational knowledge:

- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This approach has little opportunity for inference.

2. Inheritable knowledge:

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- All classes should be arranged in a generalized form or a hierarchal manner.
- In this approach, we apply inheritance property.
- Elements inherit values from other members of a class.

- This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.
- Every individual frame can represent the collection of attributes and its value.
- In this approach, objects and values are represented in Boxed nodes.
- We use Arrows which point from objects to their values.
- **Example:**



3. Inferential knowledge:

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.
- **Example:** Let's suppose there are two statements:
    1. Marcus is a man
    2. All men are mortal
       Then it can represent as;

       man(Marcus)
       ∀x = man (x) ----------> mortal (x)s

4. Procedural knowledge:

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

Requirements for knowledge Representation system:

A good knowledge representation system must possess the following properties.

1. **1. Representational Accuracy:**
   KR system should have the ability to represent all kind of required knowledge.
2. **2. Inferential Adequacy:**
   KR system should have ability to manipulate the representational structures to produce new knowledge corresponding to existing structure.
3. **3. Inferential Efficiency:**
   The ability to direct the inferential knowledge mechanism into the most productive directions by storing appropriate guides.
4. **4. Acquisitional efficiency-** The ability to acquire the new knowledge easily using automatic methods.

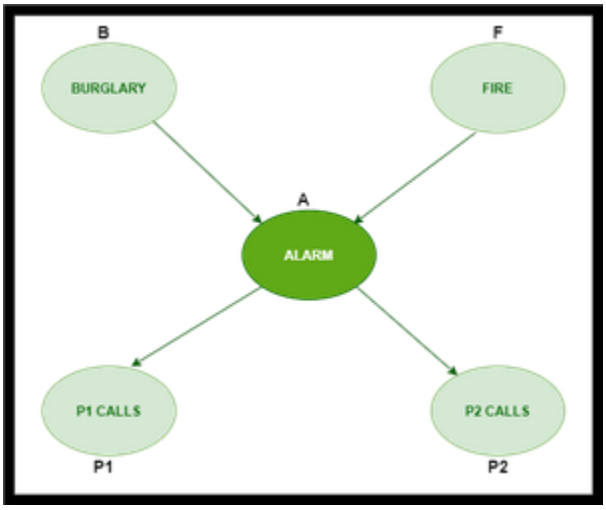
A Bayesian network (BN) is a probabilistic graphical model for representing knowledge about an uncertain domain where **each node corresponds to a random variable and each edge represents the conditional probability for the corresponding random variables** [9]. BNs are also called belief networks or Bayes nets.

How to handle uncertain knowledge in artificial intelligence?
There are four methods of manage uncertainty in expert systems and artificial intelligence [23] [24]. They are: 1) **default or non-monotonic logic, 2) probability, 3) fuzzy logic, 4) truth-value as evidential support, Bayesian theory, and 6) probability reasoning**.

Basic Understanding of Bayesian Belief Networks

Bayesian Belief Network is a graphical representation of different probabilistic relationships among random variables in a particular set. It is a classifier with no dependency on attributes i.e it is condition independent. Due to its feature of joint probability, the probability in Bayesian Belief Network is derived, based on a condition — P(attribute/parent) i.e probability of an attribute, true over parent attribute.

(Note: A classifier assigns data in a collection to desired categories.)



- In the above figure, we have an alarm ‘A’ – a node, say installed in a house of a person ‘gfg’, which rings upon two probabilities i.e burglary ‘B’ and fire ‘F’, which are – parent nodes of the alarm node. The alarm is the parent node of two probabilities P1 calls ‘P1’ & P2 calls ‘P2’ person nodes.
- Upon the instance of burglary and fire, ‘P1’ and ‘P2’ call person ‘gfg’, respectively. But, there are few drawbacks in this case, as sometimes ‘P1’ may forget to call the person ‘gfg’, even after hearing the alarm, as he has a tendency to forget things, quick. Similarly, ‘P2’, sometimes fails to call the person ‘gfg’, as he is only able to hear the alarm, from a certain distance.

Q) Find the probability that ‘P1’ is true (P1 has called ‘gfg’), ‘P2’ is true (P2 has called ‘gfg’) when the alarm ‘A’ rang, but no burglary ‘B’ and fire ‘F’ has occurred.

=> **P (P1, P2, A, ~B, ~F)** [where- P1, P2 & A are ‘true’ events and ‘~B’ & ‘~F’ are ‘false’ events]

[**Note:** The values mentioned below are neither calculated nor computed. They have observed values]

Burglary ‘B’ –

- **P (B=T) = 0.001** (‘B’ is true i.e burglary has occurred)
- **P (B=F) = 0.999** (‘B’ is false i.e burglary has not occurred)

Fire ‘F’ –

- **P (F=T) = 0.002** (‘F’ is true i.e fire has occurred)
- **P (F=F) = 0.998** (‘F’ is false i.e fire has not occurred)

B	**F**	**P (A=T)**	**P (A=F)**
T	T	0.95	0.05
T	F	0.94	0.06
F	T	0.29	0.71
F	F	0.001	**0.999**

The alarm ‘A’ node can be ‘true’ or ‘false’ (i.e may have rung or may not have rung). It has two parent nodes burglary ‘B’ and fire ‘F’ which can be ‘true’ or ‘false’ (i.e may have occurred or may not have occurred) depending upon different conditions.

A	**P (P1=T)**	**P (P1=F)**
T	**0.95**	0.05
F	0.05	0.95

The person ‘P1’ node can be ‘true’ or ‘false’ (i.e may have called the person ‘gfg’ or not) . It has a parent node, the alarm ‘A’, which can be ‘true’ or ‘false’ (i.e may have rung or may not have rung ,upon burglary ‘B’ or fire ‘F’).

Person ‘P2’ –

A	**P (P2=T)**	**P (P2=F)**
T	**0.80**	0.20
F	0.01	0.99

- The person ‘P2’ node can be ‘true’ or false’ (i.e may have called the person ‘gfg’ or not). It has a parent node, the alarm ‘A’, which can be ‘true’ or ‘false’ (i.e may have rung or may not have rung, upon burglary ‘B’ or fire ‘F’).

Solution: Considering the observed probabilistic scan –

With respect to the question — **P (P1, P2, A, ~B, ~F)** , we need to get the probability of ‘P1’. We find it with regard to its parent node – alarm ‘A’. To get the probability of ‘P2’, we find it with regard to its parent node — alarm ‘A’.

We find the probability of alarm ‘A’ node with regard to ‘~B’ & ‘~F’ since burglary ‘B’ and fire ‘F’ are parent nodes of alarm ‘A’.

From the observed probabilistic scan, we can deduce –

P (P1, P2, A, ~B, ~F)

= P (P1/A) * P (P2/A) * P (A/~B~F) * P (~B) * P (~F)

= 0.95 * 0.80 * 0.001 * 0.999 * 0.998

= 0.00075

Conditional independence relations in Bayesian networks

Before
– “numerical (global) semantics with probability distribution
– from this derive conditional independencies

Idea now
– opposite direction: topological (local) semantics

– specifies conditional independencies
– from this derive numerical semantics