Introduction to Artificial Intelligence and Search Algorithms

Introduction to Artificial Intelligence

- **What is Intelligence?**
- Intelligence is the ability to learn about, to learn from, to understand about, and interact with one's environment.
-
- **What is Artificial Intelligence (AI)?**
- **A.I:-** Is simply a way of making a computer think.
- **A.I:-** Is the part of computer science concerned with designing intelligent computer system that exhibit the characteristic associated with intelligent in human behavior.
- **This requires many processes:-**
- **1- Learning: -** acquiring the knowledge and rules that used these knowledge.
- **2- Reasoning:-** Used the previous rules to access to nearly reasoning or fixed reasoning.

Introduction to Artificial Intelligence

- **A.I Principles:-**
- 1- The data structures used in knowledge representation.
- 2- The algorithms needed to apply that knowledge.
- 3- The language and programming techniques used their implementation.

- **What are the goals of AI research?**
- The central problems (or goals) of AI research include reasoning, knowledge, planning, learning, natural language processing (communication), perception and the ability to move and manipulate objects.

- **What is problem reduction meaning?**
- Problem Reduction means that there is a hard problem may be one that can be reduced to a number of simple problems. Once each of the simple problems is solved, then the hard problem has been solved.
-

Applications of AI

- Game playing
- Speech recognition
- Understanding natural language
- Computer vision
- Expert systems
- Heuristic classification

Characteristics of AI

- • High societal impact (affect billions of people)
- • Diverse (language, vision, robotics)
- • Complex (really hard)

Search Algorithms

To successfully design and implement search algorithms, a programmer must be able to analyze and predict their behavior.

Many questions needed to be answered by the algorithm these include:

- Is the problem solver guaranteed to find a solution?
- Will the problem solver always terminate, or can it become caught in an infinite loop?
- When a solution is found, is it guaranteed to be optimal?
- What is the complexity of the search process in terms of time usage? Space search?
- How can the interpreter be designed to most effectively utilize a representation language?

State Space Search

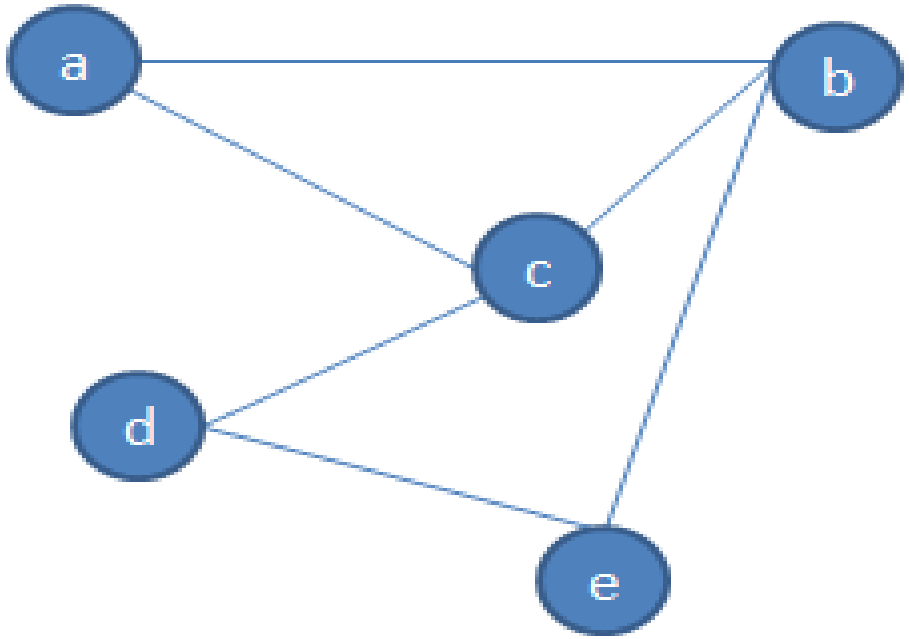
- The theory of state space search is our primary tool for answering these questions, by representing a problem as state space graph, we can use graph theory to analyze the structure and complexity of both the problem and procedures used to solve it.

Graph Theory:-

- A graph consists of a set of a nodes and a set of arcs or links connecting pairs of nodes. The domain of state space search, the nodes are interpreted to be stated in problem solving process, and the arcs are taken to be transitions between states.

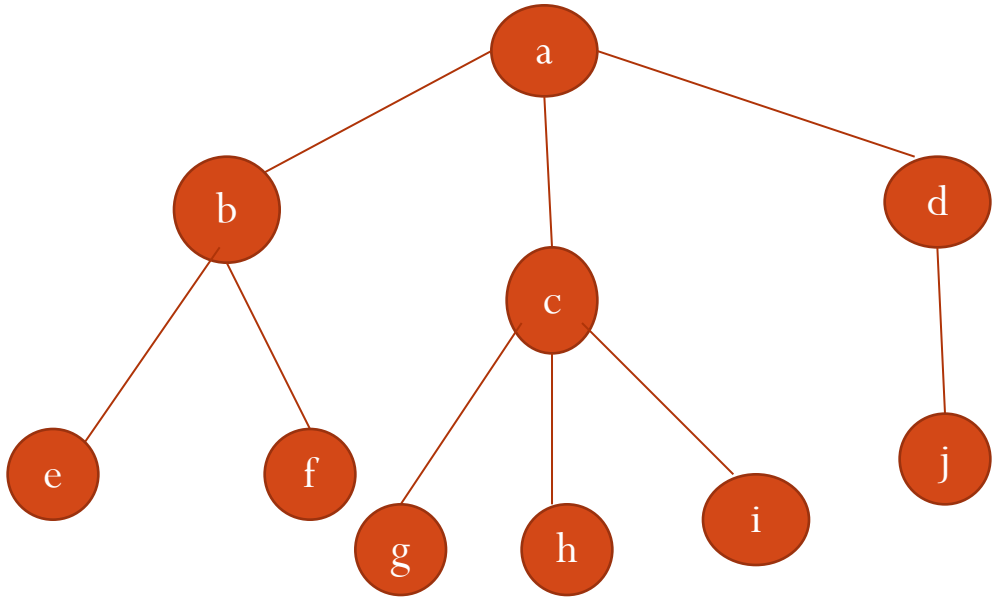
Graph theory is our best tool for reasoning about the structure of objects and relations.

Graph Theory



Nodes={a,b,c,d,e}

Arcs={(a,b), (a,c),(b,c),(b,e),(d,e),(d,c),(e,d)}



Nodes={a,b,c,d,e,f,g,h,i,j}

Arcs={(a,b),(a,c),(a,d),(b,e),(b,f),(c,g),(c,h),(c,i),(d,j)}

State Space Representation

A state space is represented by four tuple [N,A,S,GD], where:-

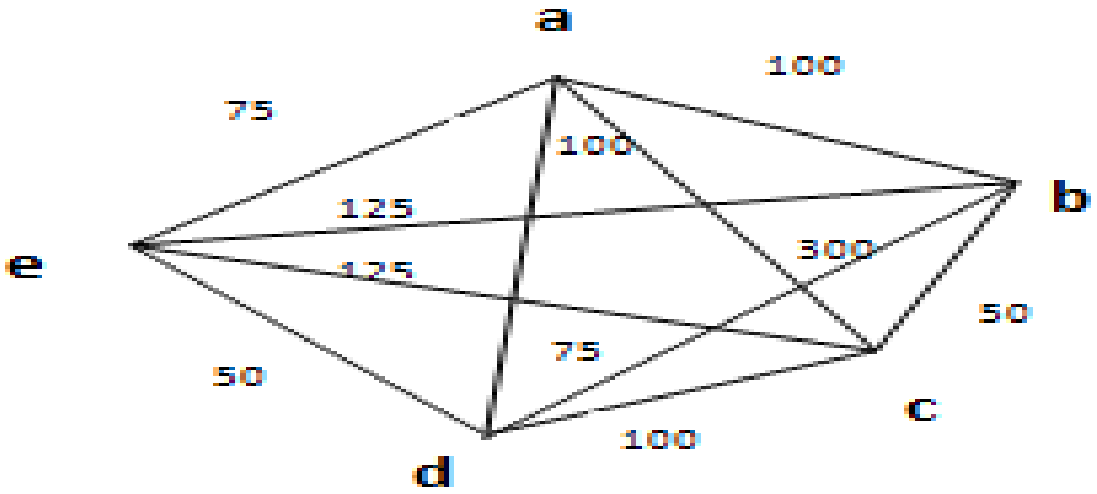
- **N** is a set of nodes or states of the graph. These correspond to the states in a problem –solving process.
- **A** is the set of arcs between the nodes. These correspond to the steps in a problem –solving process.
- **S** a nonempty subset of N , contains the start state of the problem.

- **G** a nonempty subset of N contains the goal state of the problem.

- **A solution path:- Is a path through this graph from a node S to a node in GD.**

Example:- Traveling Salesman Problem

- Starting at A , find the shortest path through all the cities , visiting each city exactly once returning to A.

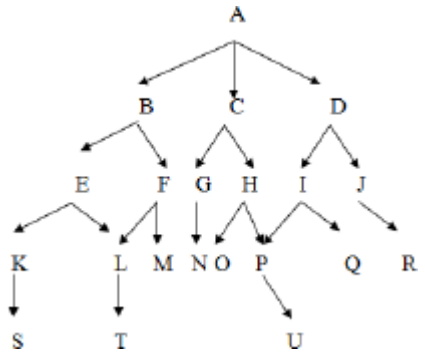


The complexity of exhaustive search in the traveling Salesman is **(N-1)!**, where N is the No. of cities in the graph. There are several technique that reduce the search complexity.

1-Blind search

This type of search takes all nodes of tree in a specific order until it reaches to goal. The order can be in the breath and the strategy will be called breadth–first–search, or strategy will be called depth first search

A- Depth First Searh

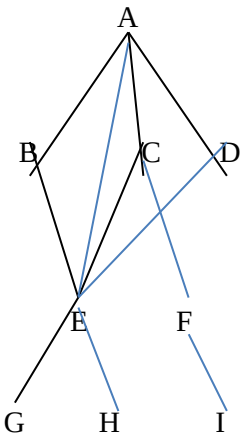


Suppose start is A and goal is N

Open	close
[A]	[]
[BCD]	[A]
[EFCD]	[AB]
[KLFCD]	[ABE]
[SLFCD]	[ABEK]
[LFCD]	[ABEKS]
[TFCD]	[ABEKSL]
[FCD]	[ABEKSLT]
[LMCD]	[ABEKSLT] DELET L
[CD]	[ABEKSLTM]
[GHD]	[ABEKSLTMC]
[NHD]	[ABEKSLTMCG]

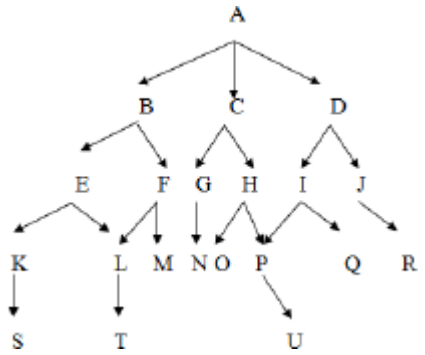
Depth first search use stack data structure.

Start A goal I



Open	Close
[A]	[]
[BECD]	[A]
[EECD]	[AB]
[GHCD]	[ABE]
[HCD]	[ABEG]
[CD]	[ABEGH]
[EFD]	[ABEGH]
[ID]	[ABEGH]

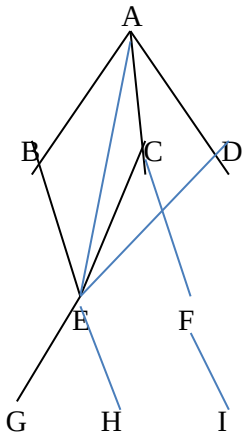
2-Breadth First search



Suppose start is A and goal is N

Open	close
[A]	[]
[BCD]	[A]
[CDEF]	[AB]
[DEFGH]	[ABC]
[EFGHIJ]	[ABCD]
[FGHIJKL]	[ABCDE]
[GHIJKLM]	[ABCDEF]
[HIJKLMN]	[ABCDEFG]
[IJKLMNOP]	[ABCDEFGH]
[JKLMNOPQ]	[ABCDEFGHI]
[KLMNOPQR]	[ABCDEFGHIJ]
[LMNOPQRS]	[ABCDEFGHIJK]
[MNOPQRST]	[ABCDEFGHIJKL]
[NOPQRST]	[ABCDEFGHIJKLM]

Breadth first use queue data structure



OPEN	CLOSE
[A]	[]
[BECD]	[A]
[ECDE]	[AB]
[CDGH]	[ABE]
[DGHEF]	[ABEC]
[GHFE]	[ABEC]
[HF]	[ABECG]
[F]	[ABECGH]
[I]	[ABECGH]

BFS	DFS
BFS starts traversal from the root node and visits nodes in a level by level manner (i.e., visiting the ones closest to the root first).	DFS starts the traversal from the root node and visits nodes as far as possible from the root node (i.e., depth wise).
Usually implemented using a queue data structure.	Usually implemented using a stack data structure.
Generally requires more memory than DFS.	Generally requires less memory than BFS.
Optimal for finding the shortest	Not optimal for finding the shortest

BFS	DFS
distance.	distance.
Used for finding the shortest path between two nodes, testing if a graph is bipartite, finding all connected components in a graph, etc.	Used for topological sorting, solving problems that require graph backtracking, detecting cycles in a graph, finding paths between two nodes, etc.

2-Heuristic search

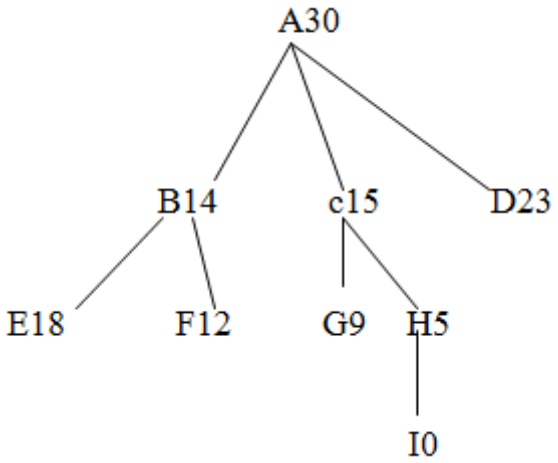
A-Best First Search

B- Hill Climbing

C- A*

A-Best First Search

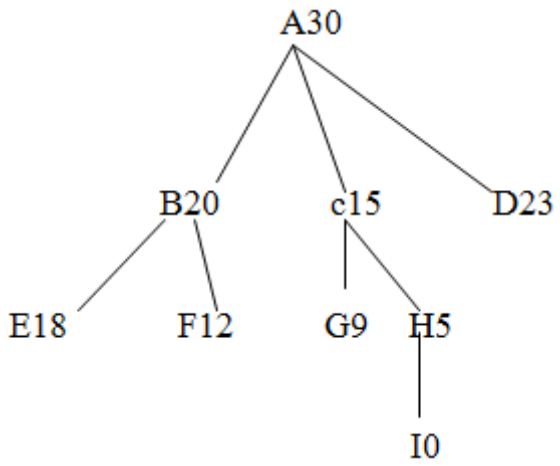
It is a search algorithm that works on a specific rule. The aim is to reach the goal from the initial state via the shortest path.



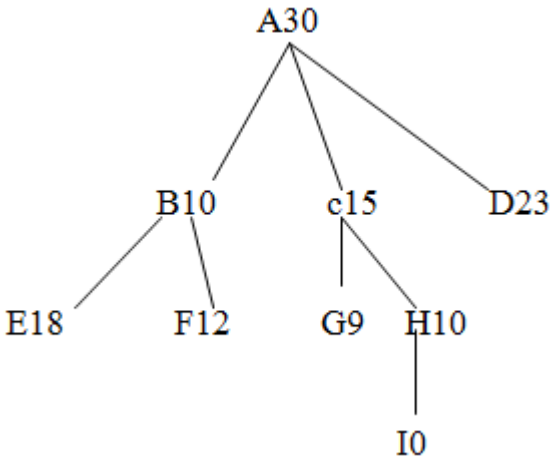
Open	Close
[A30]	[]
[B14 C15 D23]	[A30]
[F12 C15 E18 D23]	[A30 B14]
[C15 E18 D23]	[A30 B14 F12]
[H5 G9 E18 D23]	[A30 B14 F12 C15]
[I0 G9 E18 D23]	[A30 B14 F12 C15 H5]

B-Hill Climbing

search used for mathematical optimization problems in the field of Artificial Intelligence.Given a large set of inputs and a good heuristic function, it tries to find a sufficient good solution to the problem. This solution may not be the global optimal maximum.



Open	Close
[A30]	[]
[C15 B20 D23]	[A30]
[H5 G9]	[A30 C15]
[I0]	[A30 C15 H5]

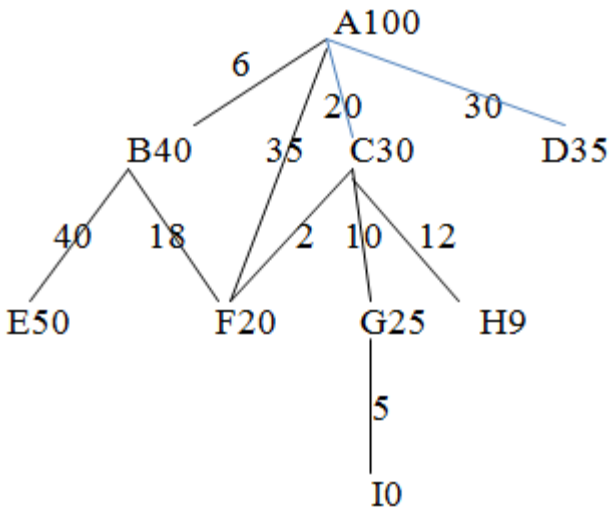


Open	Close
[A30]	[]
[B10 c15 D23]	[A30]
[F12 E18]	[A30 B10]
[]	[A30 B10 F12]

Problems of Hill Climbing

1. **Local maximum:** It is a state which is better than its neighboring state however there exists a state which is better than it(global maximum). This state is better because here the value of the objective function is higher than its neighbors.
2. **Global maximum :** It is the best possible state in the state space diagram. This because at this state, objective function has highest value.
3. **Plateua/flat local maximum :** It is a flat region of state space where neighboring states have the same value.

C. A*



Open	Close
[A100]	[]
[B46 C50 F55 D65]	[A100]
[F44 C50 F55 D65 E96]	[A100 B46]
[C50 D65 E96]	[A100 B46 F44]
[H41 F42 G55 D65 E96]	[A100 B46 F44 C50]
[F42 G55 D65 E96]	[A100 B46 C50 H41]
[G55 D65 E96]	[A100 B46 C50 H41 F42]
[I35 D65 E96]	[A100 B46 C50 H41 F42 G55]