Unit -3: Problem Solving by Searching

Compile By: Dharmendra Thapa

Ambition College, TU

July 7, 2024

Problem Solving

- Problem solving is a systematic process to solve a problem or to reach some predefined goal or solution.
- Searching is a commonly used method in AI for solving problems.
- Problem solving by searching is a systematic search through a range of possible actions in order to reach some predefined goal or solution.
- For problem solving, problem solving agents are used.
- These agents first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time.

Problem Solving

There are 4 steps in problem solving:

1. **Goal Formulation:** Goal is a state that the agent is trying to reach. t involves selecting the steps to formulate the perfect goal out of multiple goals and selecting actions to achieve the goal.
2. **Problem Formulation:** he most important step in problem-solving is choosing the action to be taken to achieve the goal formulated.
3. **Search:** Determine the possible sequence of actions that lead to the states of known values and then choosing the best sequence.
4. **Execute:** Perform the actions according to given solution.

Problem Formulation

- Problem formulation is the process of deciding what actions and states to consider, given a goal.
- Before it can do this, it needs to decide what sorts of actions and states it should consider.
- Therefore, the agent's task is to find out how to act, now and in the future, so that it reaches a goal state.

A problem is defined by:

- **Initail State:** The state in which agent satrt
- **Successor Function:** Description of possible actions available to the agent.
- **Goal Test:** Determine whether the given state is goal state or not
- **Path Cost:** Sum of cost of each path from initial state to the given state.

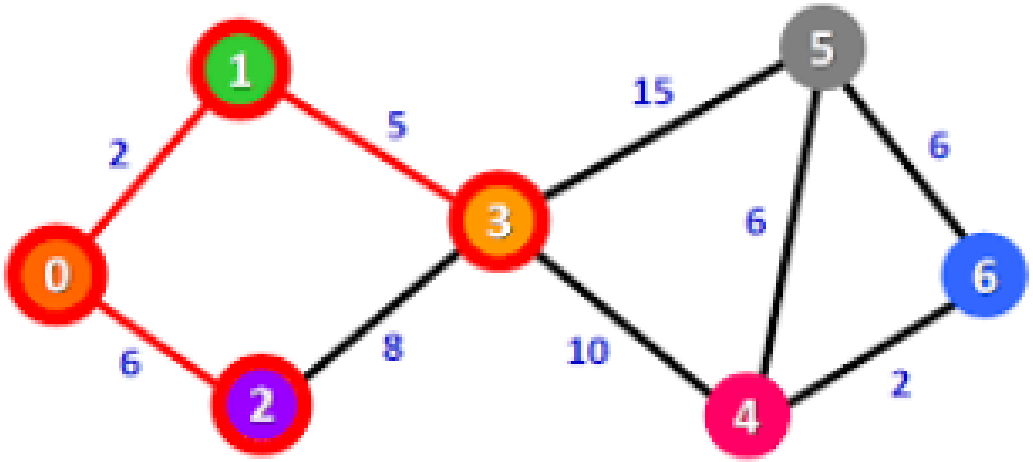
State Space Representation is used for formally define a problem.

State Space Representation

- The set of all states reachable from the initial state by any sequence of actions is called state space.
- The state space forms a directed graph in which nodes are states and the links between nodes are actions.
- A State space representation allows for the formal definition of a problem which makes the movement from initial state to goal state quite easy.
- A solution is a path from the initial state to a goal state

State Space Representation

For Example: Route Finding Problem
If the problem is to find route from 0 to 6



Problem Formulation may be as follows:

- **Initial State:** 0
- **Successor Function:** move form from one state to another state
- **Goal Test:** Wheter the arrived state is goal or not
- **Path Cost:** Total distance, money or time.

Well Defined Problem

- Well-defined problems provide a clear specification of the problem, allowing search algorithms to systematically explore the state space and find a solution.
- A well-defined problem in artificial intelligence (AI) is a problem where the initial state, allowable actions, transition model, goal test, and path cost are clearly specified.
- The key Components are
  - **Initial State:** The starting point or current state of the problem.
  - **Actions:** The set of possible actions the agent can take to transition from one state to another.
  - **Transition Model:** A function that describes what state results from taking a particular action in a given state.
  - **Goal Test:** A function that determines whether a given state is the desired goal state.
  - **Path Cost:** A numerical cost associated with each action, used to evaluate the overall cost of a solution path.

Solving Problem by Searching

- Having formulated some problems, we now need to solve them.
- To solve a problem we should perform a systematic search through a range of possible actions in order to reach some predefined goal or solution.
- A solution is an action sequence, so search algorithms works by considering various possible action sequences.
- Consider a problem represented with a state space graph. The nodes represent states and the links represent action from one state to another. The number associated to a link represents the cost of links.

Performance Evalutation of Serch Strategies

There are 4 ways to determines the performance of search techniques.

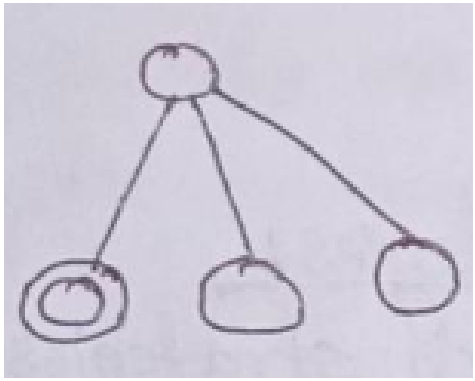
1. **Completencess:** An algorithm is said to be complete if it definitely finds solution to the problem, if exist.
2. T**ime Complexity:** How long (worst or average case) does it take to find a solution? Usually measured in terms of the number of nodes expanded.
3. **Space Complexity:** How much space is used by the algorithm? Usually measured in terms of the maximum number of nodes in memory at a time.
4. **Optimality/Admissibility:** If a solution is found, is it guaranteed to be an optimal one? For example, is it the one with minimum cost?

Performance Evalutation of Serch Strategies

Time and space complexity are measured in terms of

- **b:** maximum branching factor (number of successor of any node) of the search tree.
- **d:** depth of the least-cost solution.
- **m:** maximum length of any path in the space.

For Examples:



b = 3, d = 1, and m = 1

Searching Strategies

There are mainly 2 types of searching methodes.

1. Uninformed Search
2. Informed Search

Uninformed Search

- It is also knoe as blind search.
- This strategies are those search algorithms that explore a problem space without using any specific knowledge or heuristics about the problem domain.
- They operate in a brute-force manner, trying out every part of the search space systematically to find a solution.

There are following types of Uninformed searching algorithms:

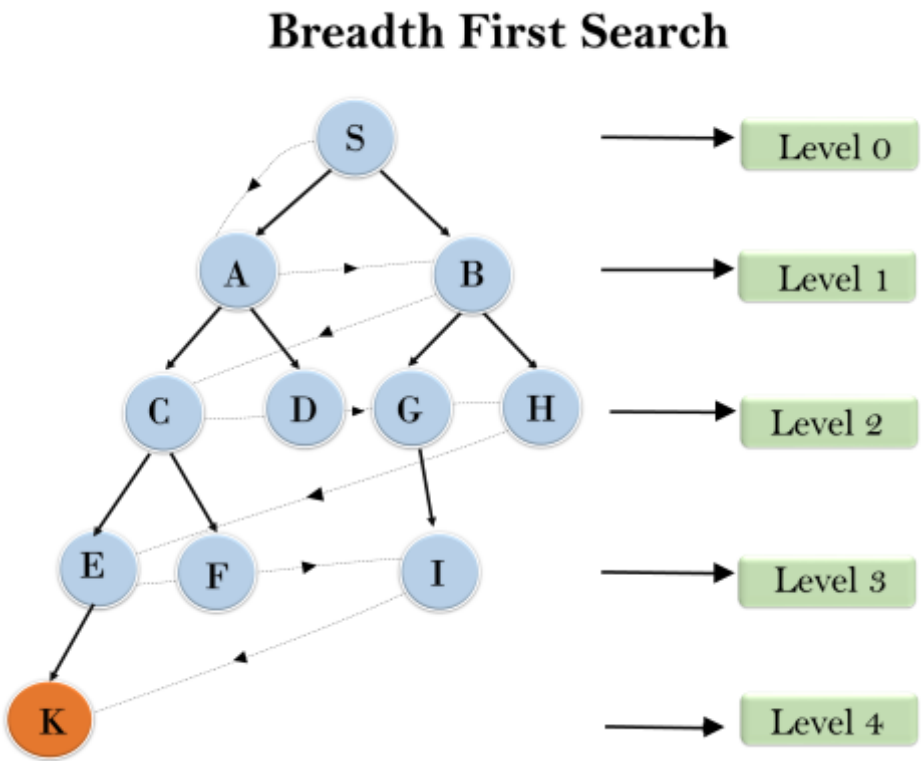
1. Breadth-First Search (BFS)
2. Depth-First Search (DFS)
3. Depth-Limited Search
4. Iterative Deepening Search
5. Bidirectional Search

Breadth-First Search (BFS)

- BFS explores the search space in breadth-first order, visiting all the neighboring nodes at the current depth before moving on to the nodes at the next depth level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.
- It uses a queue(FIFO) data structure to keep track of the nodes to visit.

Breadth-First Search (BFS)

For Example: Here the traversing of the tree using BFS algorithm from the root node S to goal node K.



The traversed path will be

$S \rightarrow A \rightarrow B \rightarrow C \rightarrow D \rightarrow G \rightarrow H \rightarrow E \rightarrow F \rightarrow I \rightarrow K$

Breadth-First Search (BFS)

BFS Evalutation

- **Completeness:** BFS is complete, which means if the shallowest goal node is at some finite depth(d), then BFS will find a solution.
- **Time Complexity:** Assume a state space where every state has b successors and solution is at d depth

$$T\ (b) = 1 + b^2 + b^3 + ... + b^d = O(b^{d+1})$$

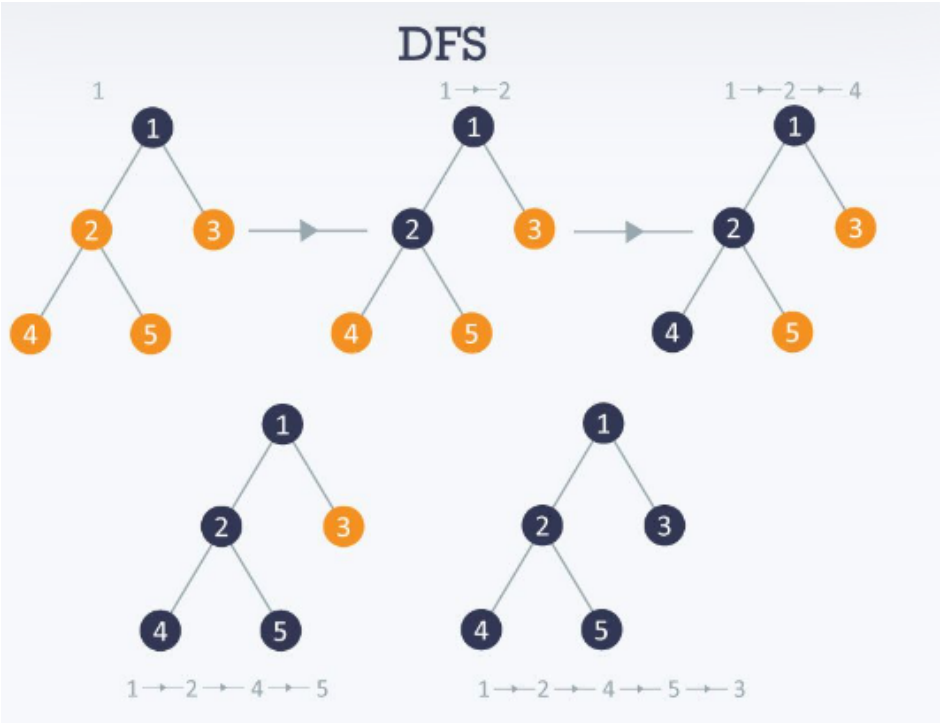
- **Space Complexity:** Space complexity of BFS algorithm is given by the memory size of problem which is $O(b^{d+1})$
- **Optimality:** BFS is Optimal if all paths have the same cost.Otherwise, not optimal but finds solution with shortest path length (shallowest solution).

Depth-First Search (DFS)

- It starts from the root node and follows each path to its greatest depth node before moving to the next path .
- DFS explores the search space in depth-first order, fully exploring one branch of the search tree before backtracking and exploring the next branch.
- DFS is a recursive algorithm for traversing a tree or graph data structure.
- DFS uses a stack data structure for its implementation.

Depth-First Search (DFS)

For Example: Here the traversing of the tree using DFS algorithm from the root node 1 to goal node 3.



The traversed path will be $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 3$

Depth-First Search (DFS)

DFS Evalutation

- **Completeness:** DFS is complete, which means if the shallowest goal node is at some finite depth(d), then DFS will find a solution.
- **Time Complexity:** Let, m is the maximum depth of the search tree.

$$T\ (b) = 1 + b^2 + b^3 + ... + b^m = O(b^m)$$

- **Space Complexity:** DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is O(bm).
- **Optimality:** DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

Depth-Limited Search (DLS)

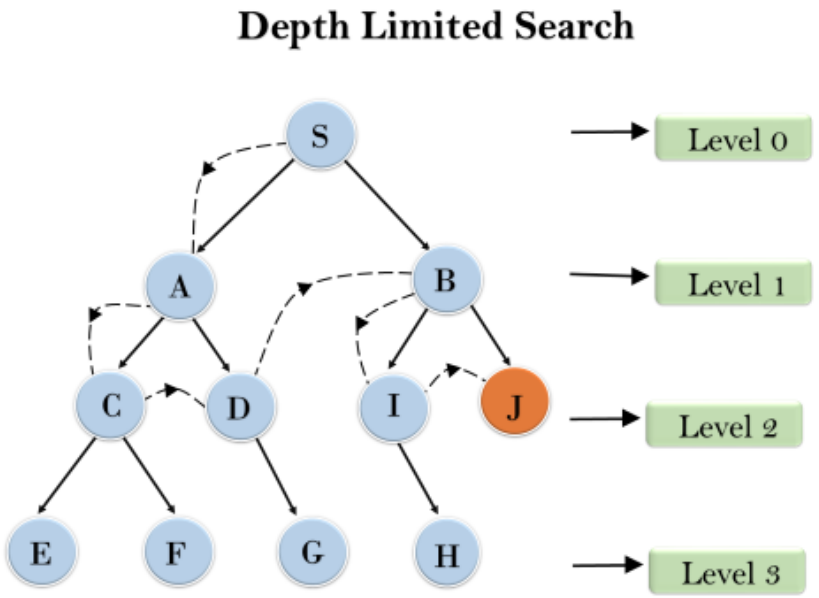
- It is a modification of depth-first search with a predetermined limit(l).
- In this algorithm, the node at the depth limit will treat as it has no successor nodes further.
- It solve the infinite path problem in DFS.

Termination Conditions:

- If there is no solution.
- Reaching on predetermined depth(limit)

Depth-Limited Search (DLS)

For Example: Here the traversing of the tree using DLS algorithm from the root node S to goal node J with depth limit 2.



The traversed path will be $S \rightarrow A \rightarrow C \rightarrow D \rightarrow B \rightarrow I \rightarrow J$

If depth limit is 1 then there is no solution.

Depth-Limited Search (DLS)

DLS Evalutation

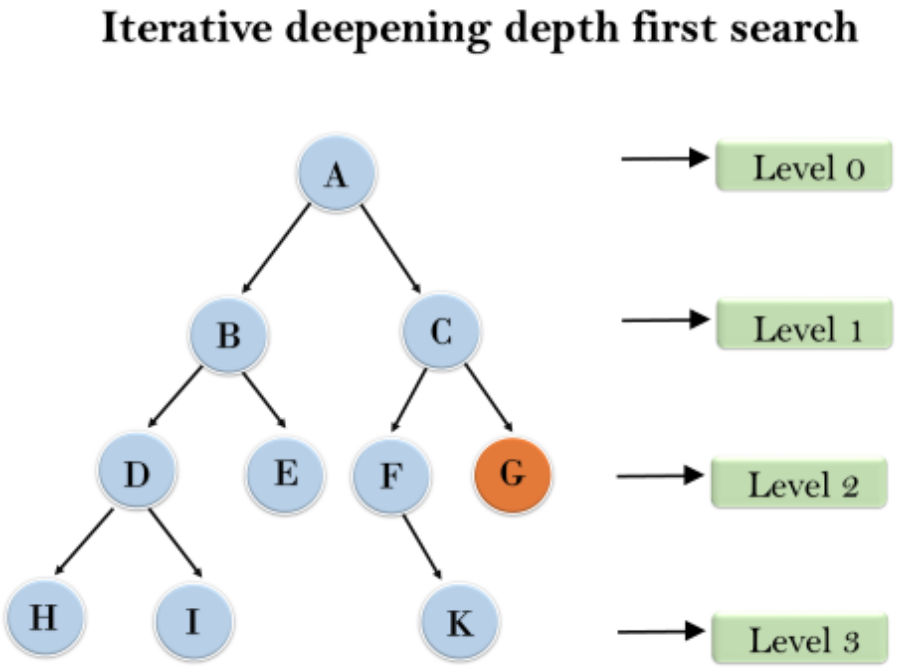
- **Completeness:** If $l < d$ then incompleteness results.
- **Time Complexity:** O(b^l)
- **Space Complexity:** O(bl).
- **Optimality:** If $l > d$ then not optimal.

Iterative Deepening Depth First Search (IDDFS)

- In this strategy, depth-limited search is run repeatedly, increasing the depth limit with each iteration until it reaches d, the depth of the shallowest goal state.
- It does this by gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found.
- It Combine both BFS and DFS.

Iterative Deepening Depth First Search (IDDFS)

For Example: Start node is 'A' and Goal node 'G'



The traversed path will be

Iteration 0: A

Iteration 1: $A \rightarrow B \rightarrow C$

Iteration 2: $A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F \rightarrow G$

Iterative Deepening Depth First Search (IDDFS)

IDDFS Evalutation

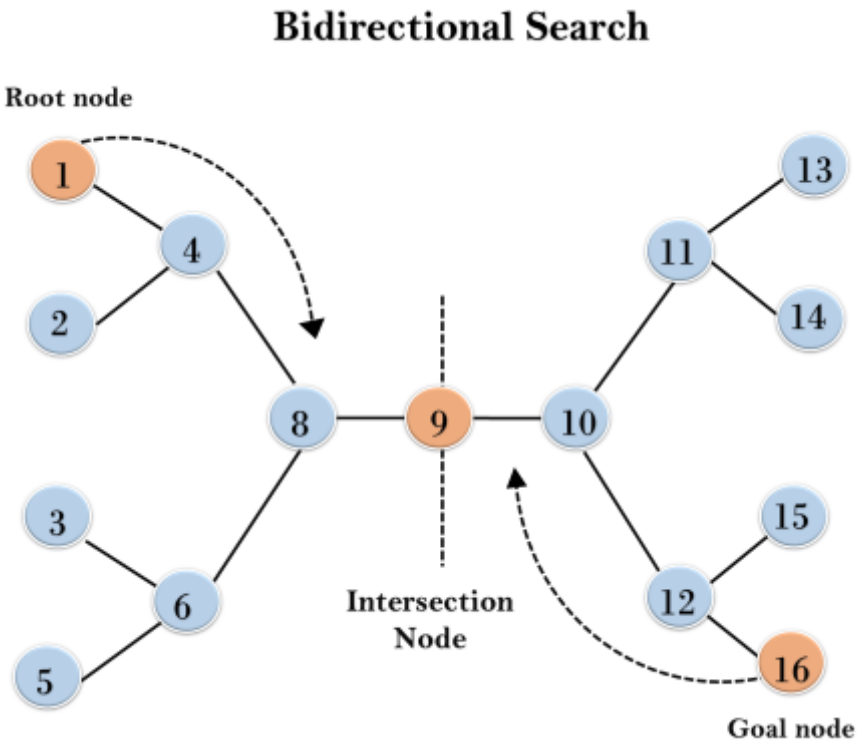
- **Completeness:** This algorithm is complete is if the branching factor is finite.
- **Time Complexity:** O(b^d)
- **Space Complexity:** O(bd).
- **Optimality:** IDDFS algorithm is optimal if path cost is a non-decreasing function of the depth of the node.

Bidirectional Search

- This is a search algorithm which replaces a single search graph.
- It two smaller graphs one starting from the initial state and one starting from the goal state.
- Check at each stage if the nodes are intersect. If so, the path concatenation is the solution.

Bidirectional Search

For Example: We can use BFS or DFS. Here we apply DFS



Bidirectional Search

Forward Search: $1 \rightarrow 4 \rightarrow 8 \rightarrow 9$
Backward Search: $16 \rightarrow 12 \rightarrow 10 \rightarrow 9$
The Finial traversed path will be the concatenation of Forward and Backward search: $1 \rightarrow 4 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 12 \rightarrow 16$

BS Evalutation

- **Completeness:** Yes (If done with correct strategy- e.g. BFS).
- **Time Complexity:** O($b^{d/2}$)
- **Space Complexity:** O($b^{d/2}$).
- **Optimality:** Yes (If done with correct strategy- e.g. BFS).

Informed Search

- Informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc.
- This knowledge help agents to explore less to the search space and find more efficiently the goal node.
- The informed search algorithm is more useful for large search space.
- Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

There are following types of Informed searching algorithms:

1. Greedy Best first search,
2. A^* *Search*,
3. Hill Climbing,
4. Simulated Annealing

Heuristic Search

- Heuristic Search Uses domain-dependent (heuristic) information in order to search the space more efficiently.
- It uses additional information, known as a heuristic function, guides the search process towards the most promising areas of the search space, allowing the algorithm to find good solutions faster than uninformed search methods.
- In general, it try to optimize a problem using heuristic function.

Heuristic Function

It is a function H(n) that gives an estimation on the cost of getting from node 'n' to the Goal state. It helps to selecting optimal node for expansion. Generally there are 2 types of Heuristic Function.

1. **Admissible Heuristic:** An admissible heuristic is a heuristic function h(n) that never overestimates(Underestimates) the actual cost to reach the goal from the current node n. i.e, heuristic function always less than or equal to actual cost of lowest cost path from node 'n' to goal.

Heuristic Search

$$H(n) \leq H^{'}(n)$$

1. **Non-admissible(Inadmissible) Heuristic:** An admissible heuristic is a heuristic function h(n) that overestimates the actual cost to reach the goal from the current node n. i.e, heuristic function always greater than or equal to actual cost of lowest cost path from node 'n' to goal.

$$H(n) > H^{'}(n)$$

Greedy Best first search

- The best-first search part of the name means that it uses an evaluation function to select which node is to be expanded next.
- The node with the lowest evaluation is selected for expansion because that is the best node,
- if the heuristic is good, then it supposedly has the closest path to the goal.

$$Evaluation\ function\ f(n) = h(n)(heuristic)$$

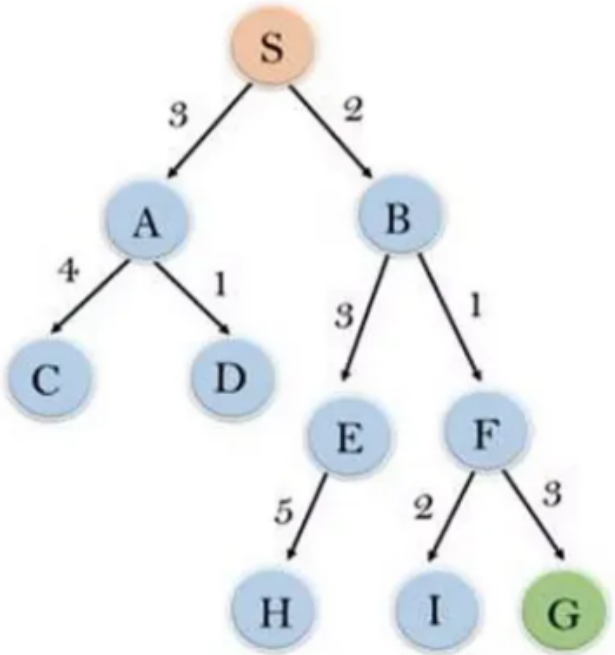
Greedy Best first search

Algorithm

1. Add start node in the OPEN LIST.
2. Expand current node and remove from OPEN LIST then, add the current node to the CLOSED LIST.
3. Remove the node with the lowest h(x) value from the OPEN LIST and mark as current node.
4. If current node is goal node then and into CLOSED list then return success. Else move to **Step 2**

Greedy Best first search

For example: Finding the path from S to G in the following graph:



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

Greedy Best first search

OPEN	CLOSED
S	
A,B	S
A	S,B
E,F,A	S,B
E,A	S,B,F
I,G,E,A	S,B,F
I,E,A	S,B,F,G

The Finial Path is $S \rightarrow B \rightarrow F \rightarrow G$ and total cost is 2+1+3=6

A* search

- A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.
- Algorithm A* combines the advantages of two other search algorithms: Dijkstra's algorithm and Greedy Best-First Search.
- The main idea of A* is to evaluate each node based on two parameters:
  1. **g(n):** the actual cost to get from the initial node to node n. It represents the sum of the costs of node n outgoing edges.
  2. **h(n):** Heuristic cost (also known as "estimation cost") from node n to destination(Goal) node.
- The evaluation function of node n is defined as

$$f(n) = g(n) + h(n)$$

A* search

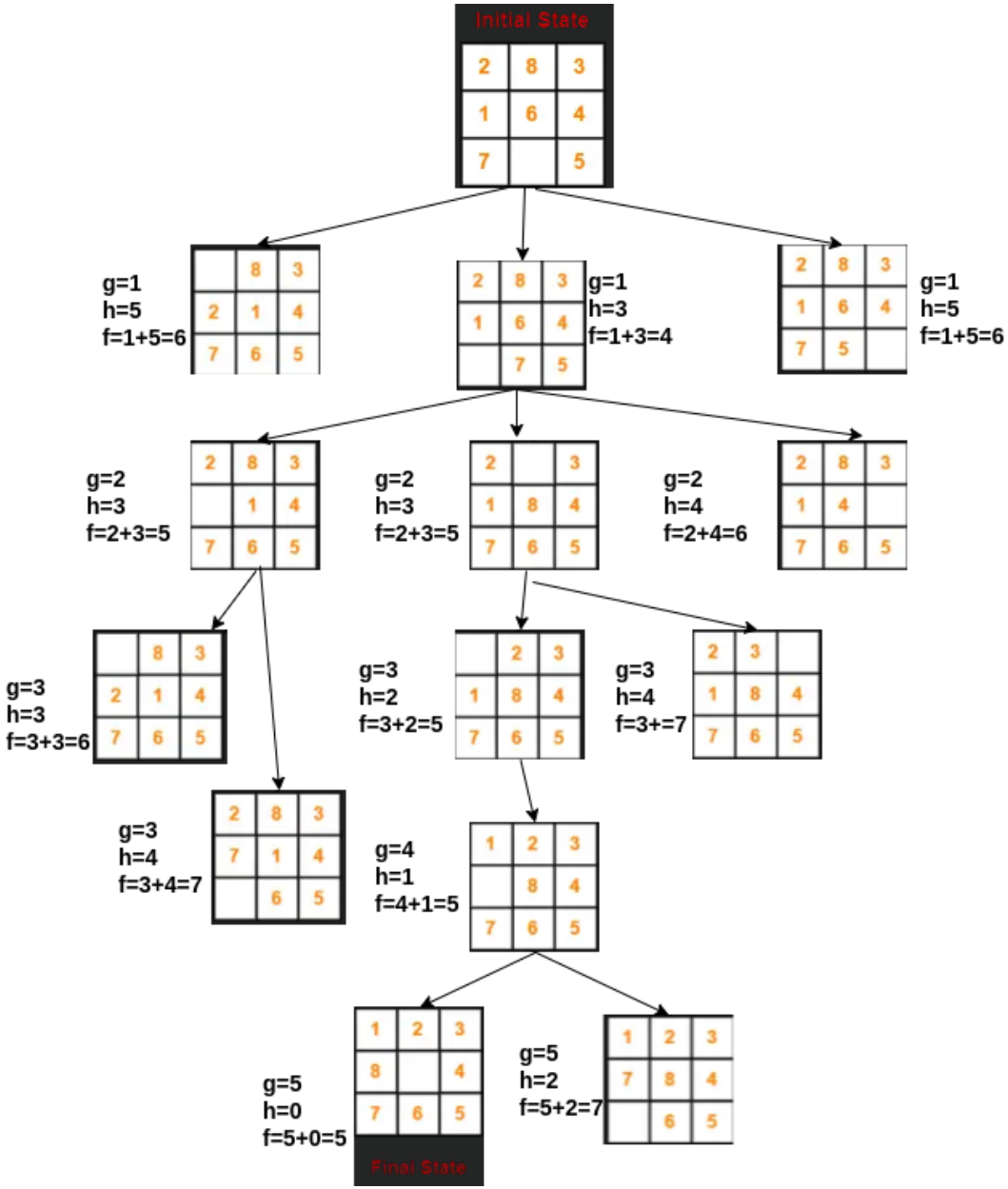
Algorithm

1. Add start node in the OPEN LIST.
2. Expand current node and remove from OPEN LIST then, add the current node to the CLOSED LIST.
3. Remove the node with the lowest $F(n)$ value from the OPEN LIST and mark as current node.
4. If current node is goal node then and into CLOSED list then return success. Else move to **Step 2**

A* search

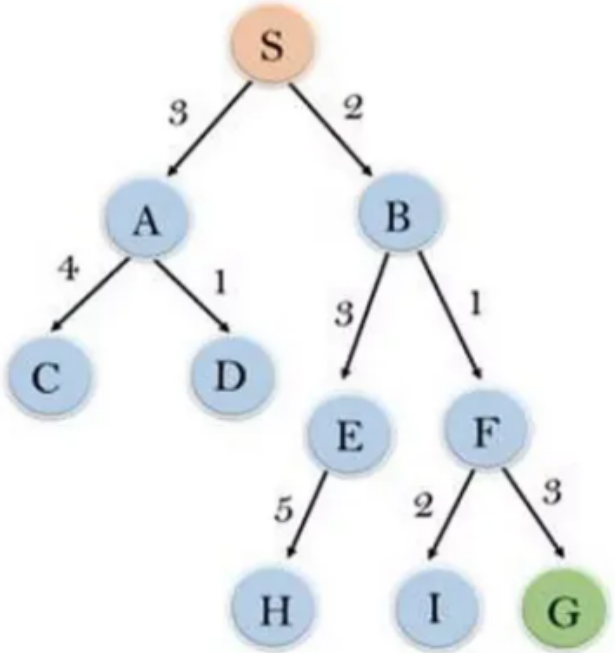
For example: Find the most cost-effective path to reach the final state from initial state using A* Algorithm. Consider g(n) = Depth of node and h(n) = Number of misplaced tiles.

A* search


Initial State
2 8 3
1 6 4
7 5
g=1
h=5
f=1+5=6
g=1
h=3
f=1+3=4
g=1
h=5
f=1+5=6
g=2
h=3
f=2+3=5
g=2
h=3
f=2+3=5
g=2
h=4
f=2+4=6
g=3
h=3
f=3+3=6
g=3
h=2
f=3+2=5
g=3
h=4
f=3+=7
g=3
h=4
f=3+4=7
g=4
h=1
f=4+1=5
g=5
h=0
f=5+0=5
Final State
g=5
h=2
f=5+2=7

A* search

Example 2: Find the most cost-effective path to reach from start state "S" to final state "G" using A* Algorithm.



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

Hill Climbing Search

- Hill climbing search is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem.
- It terminates when it reaches a peak value where no neighbor has a higher value.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- Hill Climbing can get stuck in local optima, meaning that it may not find the global optimum of the problem.
- This algorithm doesn't guarantee to find a solution(or optimal solution) of givrn problem because it only move one direction (increasing) and never backtrack.

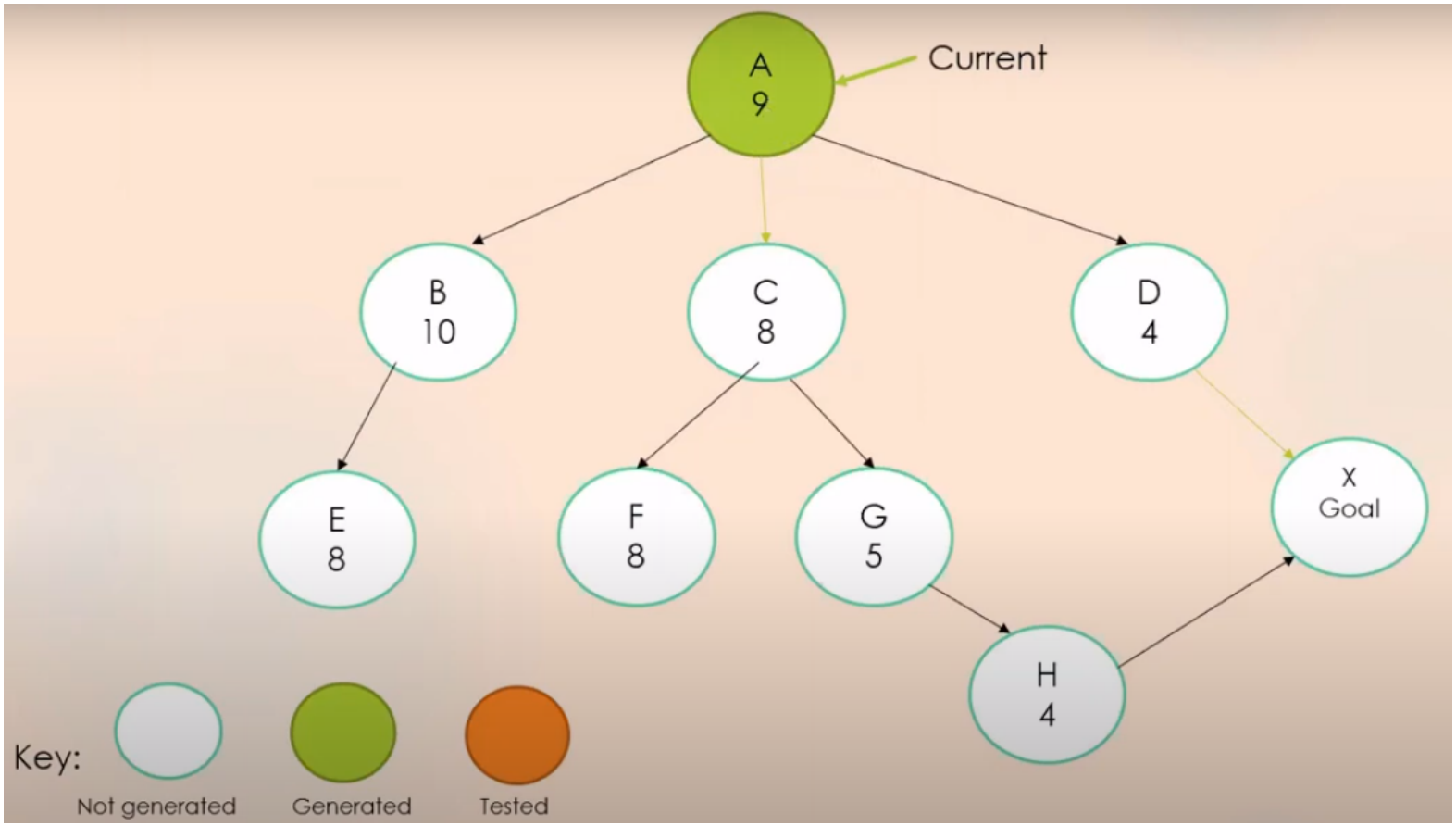
Hill Climbing Search

Algorithm

1. Evaluate the start state if goal then return (success).
   else continue with start state as the current state.
2. Loop until a solution is found or until there are no new operator to apply to current node :
   a. select a new operator and apply current state to produce a new state.
   b. evaluate the new state :
      - if it is a goal then return (success).
      - if not goal but better than current state then make it the current state.
      - if it is not better than current state then continue the loop

Hill Climbing Search

Example: Find the most cost-effective path to reach from start state "A" to final state "X" using Hill Algorithm



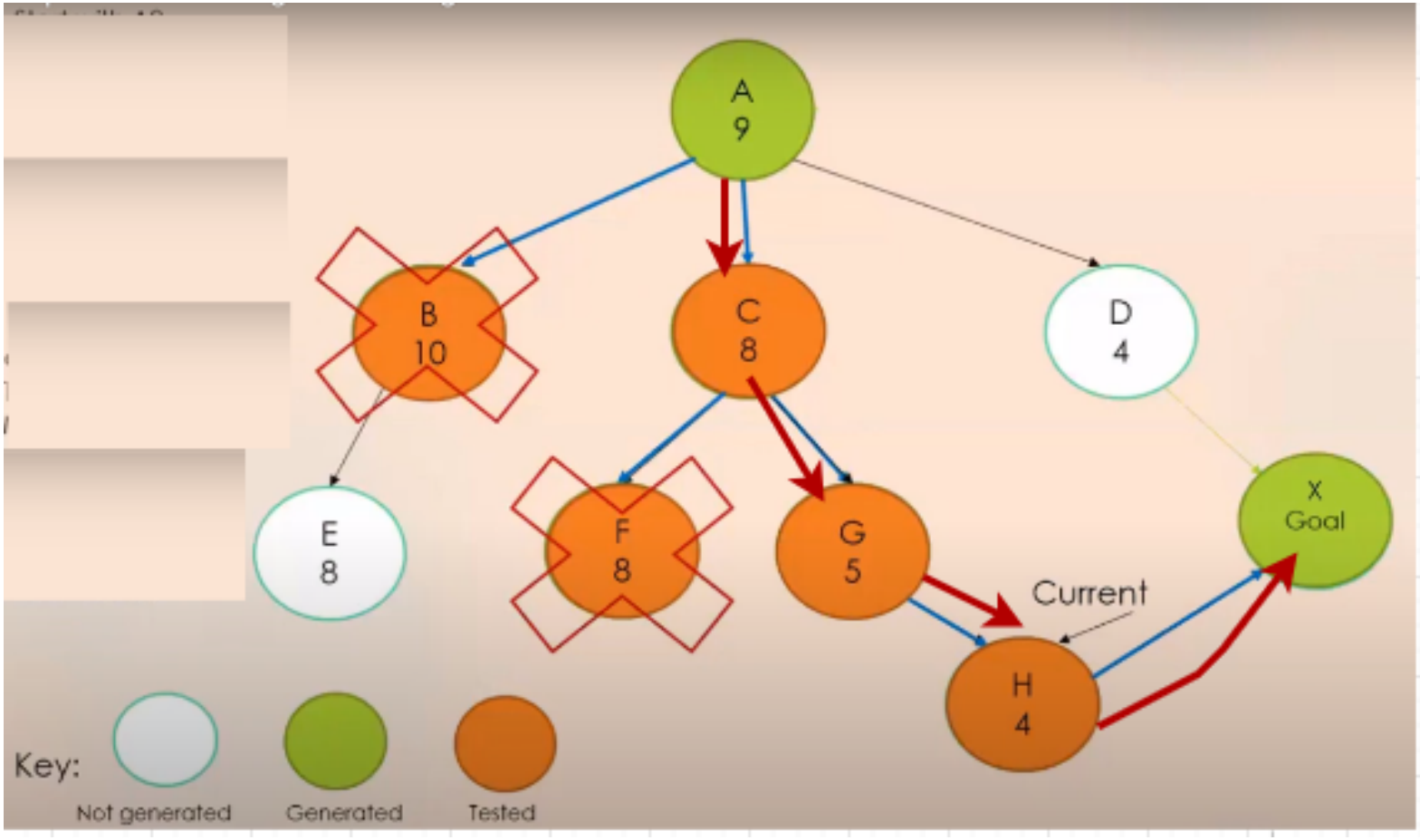
Hill Climbing Search

Solution

1. Start with A(9)
2. Generate B(10)
3. Test B(10)-not better
4. Discard B(10)
5. Generate C(8)
6. Test C(8) - Better
7. Make C(8)-Current
8. Generate F(8)
9. Test F(8)-not better
10. Discard F(8)
11. Generate G(5)
12. Test G(5)-better
13. Make G(5) current
14. Generate H(4)
15. Test H(4) - Better
16. Make H(4) current
17. Generate X
18. Test X - GOAL
19. EXIT

Hill Climbing Search

The Finial path $A \rightarrow C \rightarrow G \rightarrow H \rightarrow X$ which display in below graph.



Simulated Annealing Search

We already learn about it.

Game Playing

We already learn about TIC TAC TOE gaming

Adversarial Search Techniques

We already learned about **Mini-Max Search and Aplha-Beta Pruning**

Constraint Satisfaction Problem(CSP)

We already learned about **Coloring Maping Problem**