National University of Computer and Emerging Sciences

CL 461 Artificial Intelligence

Lab Manual 05

Problem Solving by Searching
Uninformed/Blind Search Algorithms

PROBLEM-SOLVING AGENT

In Artificial Intelligence, Search techniques are universal problem-solving methods. **Rational agents** or **Problem-solving agents** in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result.

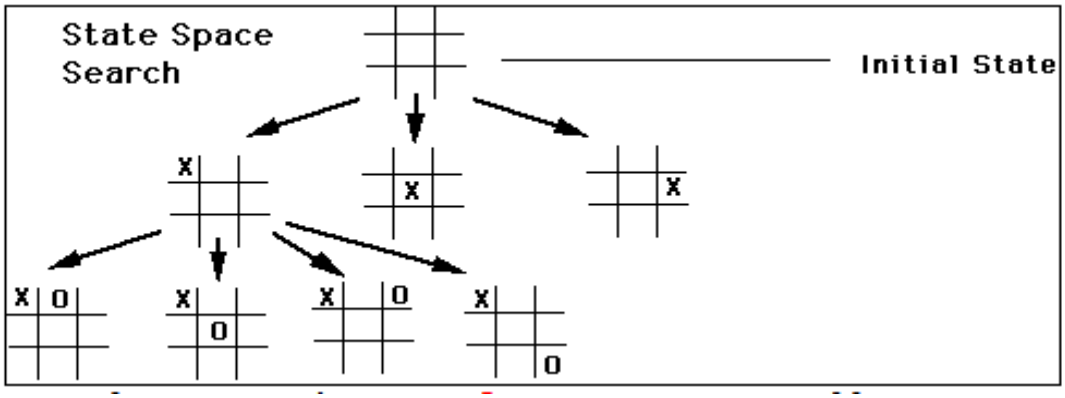
Problem-solving agents are the **goal-based agents** and use atomic representation. In this topic, we will learn various problem-solving search algorithms.

Steps performed by Problem-solving agent

1. **Goal Formulation:** It is the first and simplest step in problem-solving. It organizes the steps/sequence required to formulate one goal out of multiple goals as well as actions to achieve that goal. Goal formulation is based on the current situation and the agent's performance measure (discussed below).

2. **Problem Formulation:** It is the most important step of problem-solving which decides what actions should be taken to achieve the formulated goal. There are following five components involved in problem formulation:
    a. **Initial State:** It is the starting state or initial step of the agent towards its goal.
    b. **Actions:** It is the description of the possible actions available to the agent.
    c. **Transition Model:** It describes what each action does.
    d. **Goal Test:** It determines if the given state is a goal state.
    e. **Path cost:** It assigns a numeric cost to each path that follows the goal. The problem-solving agent selects a cost function, which reflects its performance measure. Remember, an optimal solution has the lowest path cost among all the solutions.

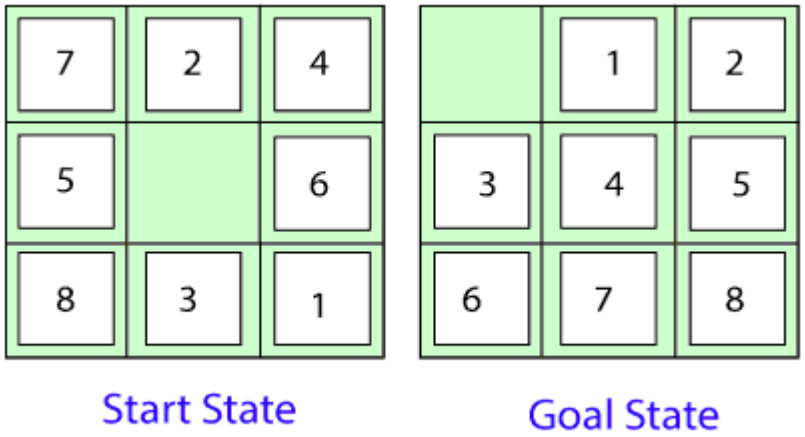
Note: **Initial state, actions,** and **transition model** together define the **state-space** of the problem implicitly.



Example Problems

1. ***8 Puzzle Problem:***

Here, we have a **3×3 matrix** with movable **tiles numbered from 1 to 8 with a blank space**. The tile adjacent to the blank space can slide into that space.
The objective is to reach a specified goal state similar to the goal state.
In the figure, our task is to convert the current state into goal state by sliding digits into the blank space.



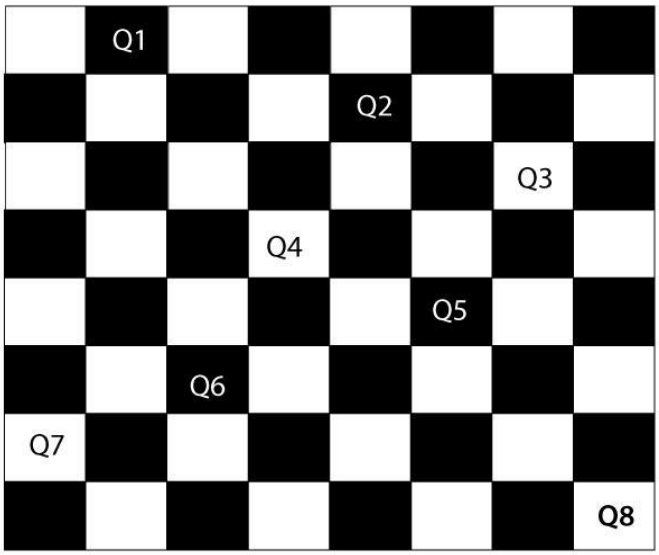
Lab Task 1:

Describe the problem formulation for 8 puzzle problem

a. **Initial State**
b. **Actions**
c. **Transition Model**
d. **Goal Test**
e. **Path cost**

2. ***8-queens problem:***

The aim of this problem is to place eight queens on a chessboard in an order where no queen may attack another. A queen can attack other queens either diagonally or in the same row and column.
From the following figure, we can understand the problem as well as its correct solution.
For this problem, there are two main kinds of formulation:



Incremental formulation:

It starts from an empty state where the operator augments a queen at each step.
In this formulation, there is approximately 1.8 x 10^{14} possible sequence to investigate.

Complete-state formulation:

It starts with all the 8-queens on the chessboard and moves them around, saving from the attacks.
This formulation is better than the incremental formulation as it reduces the state space from 1.8 x 10^{14} to 2057, and it is easy to find the solutions.

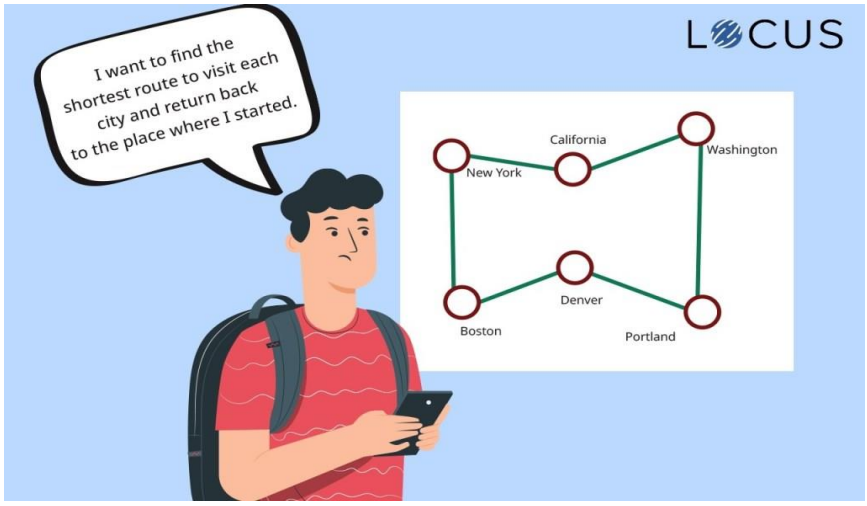
Lab Task 2:

Describe the problem formulation for 8 Queens Problem for both cases

a. **Initial State**
b. **Actions**
c. **Transition Model**
d. **Goal Test**
e. **Path cost**

3. **Traveling salesperson problem(TSP):**

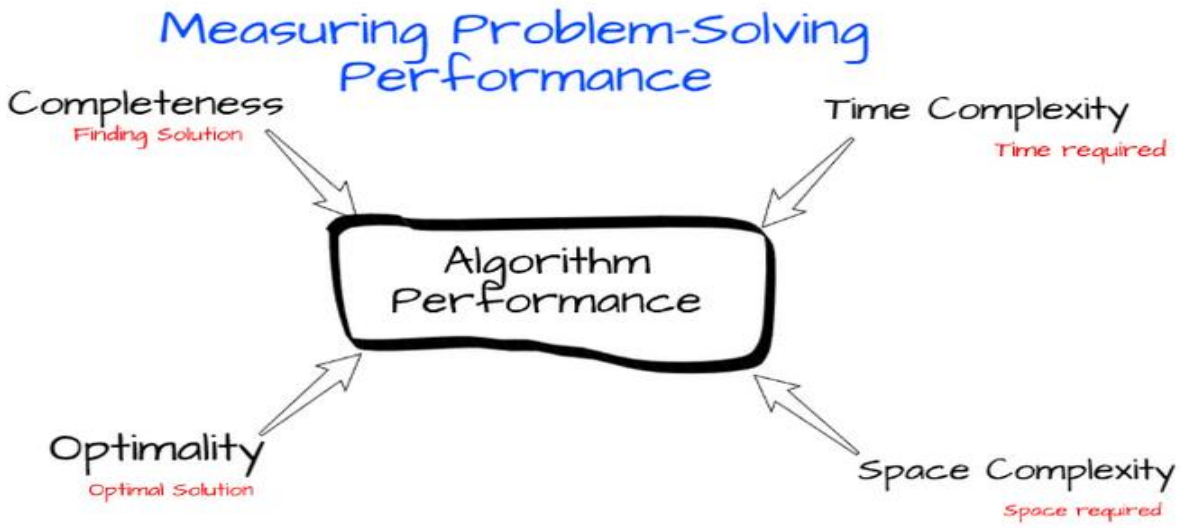
It is a touring problem where the salesman can visit each city only once. The objective is to find the shortest tour and sell-out the stuff in each city.



Lab Task 3:

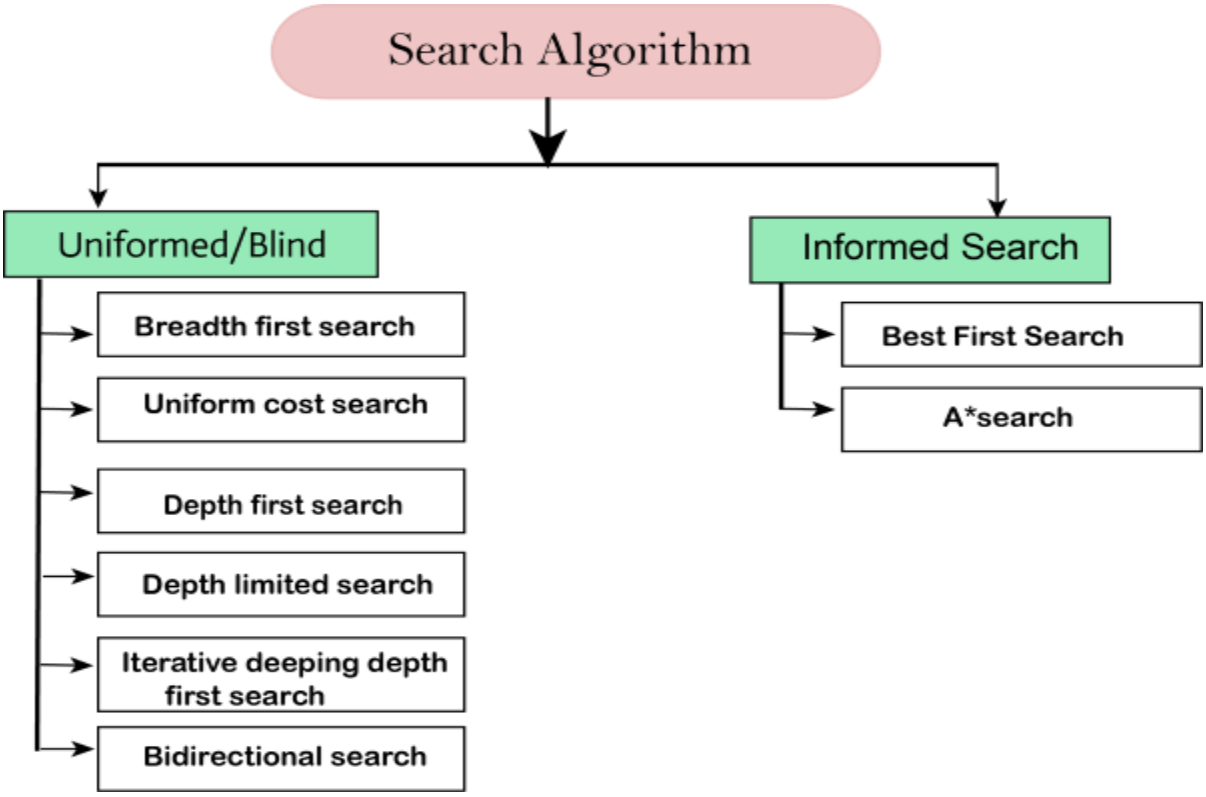
Describe the problem formulation for Travelling Salesman Problem

a. **Initial State**
b. **Actions**
c. **Transition Model**
d. **Goal Test**
e. **Path cost**



Types of search algorithms

Based on the search problems we can classify the search algorithms into uninformed (Blind search) search and informed search (Heuristic search) algorithms.



Criteria	BFS	Uniform-cost	DFS	Depth-limited	IDS	Bidirectional
Complete?	Yes[#]	Yes[#&]	No	No	Yes[#]	Yes[#+]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/e \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/e \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[$]	Yes	No	No	Yes[$]	Yes[$+]

b: Branching factor
d: Depth of the shallowest goal
l: Depth limit
m: Maximum depth of search tree
e: The lower bound of the step cost

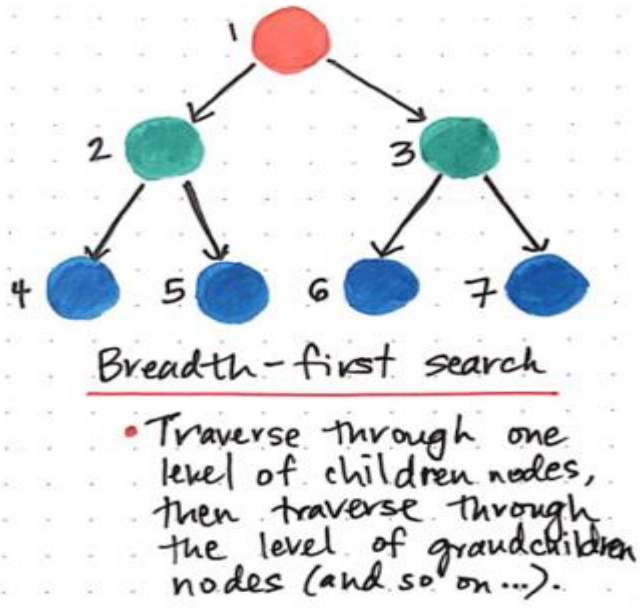
[#]: Complete if *b* is finite
[&]: Complete if step cost >= *e*
[$]: Optimal if all step costs are identical
[+]: If both direction use BFS

Uninformed/Blind Search:

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes. Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search. It examines each node of the tree until it achieves the goal node. Types of this type are already mentioned in the above diagram.

1. Breadth-first Search:

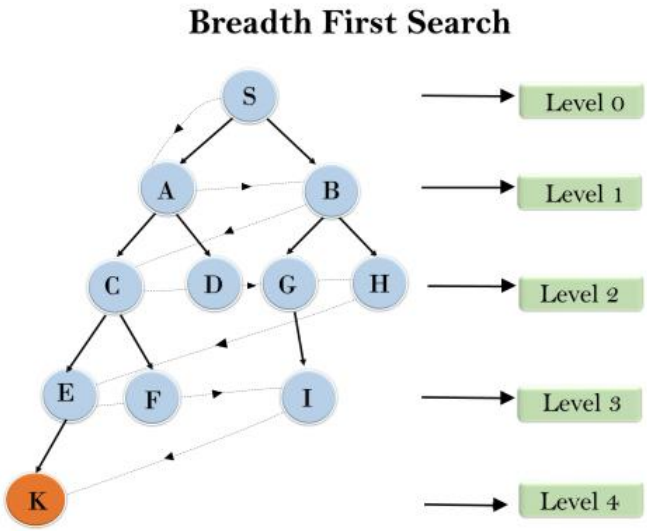
Breadth-first search implemented using FIFO queue data structure.



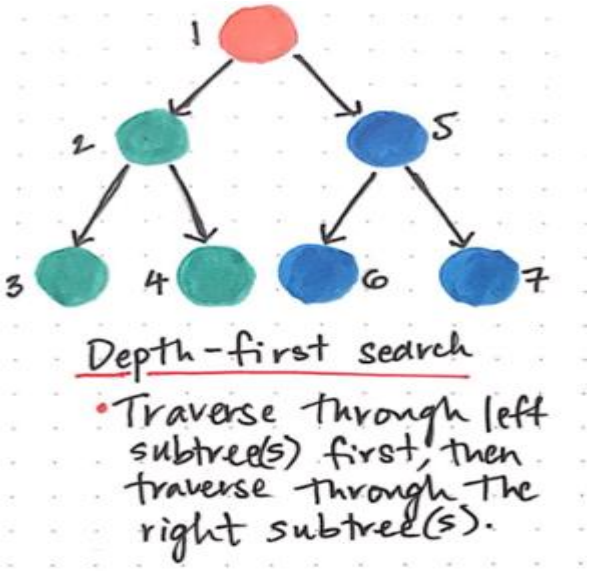
Example:

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K



2. Depth-first Search:



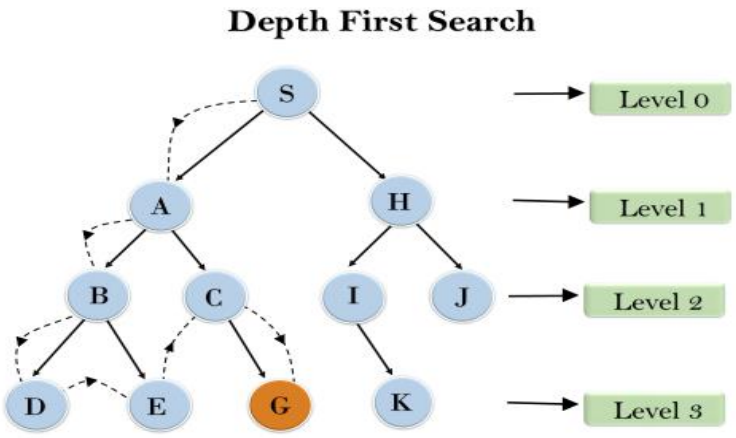
DFS uses a stack data structure for its implementation.

Note: Backtracking is an algorithm technique for finding all possible solutions using recursion.

Example:

In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

S---> A--->B---->D--->E---->C--->G



3. Depth-Limited Search Algorithm:

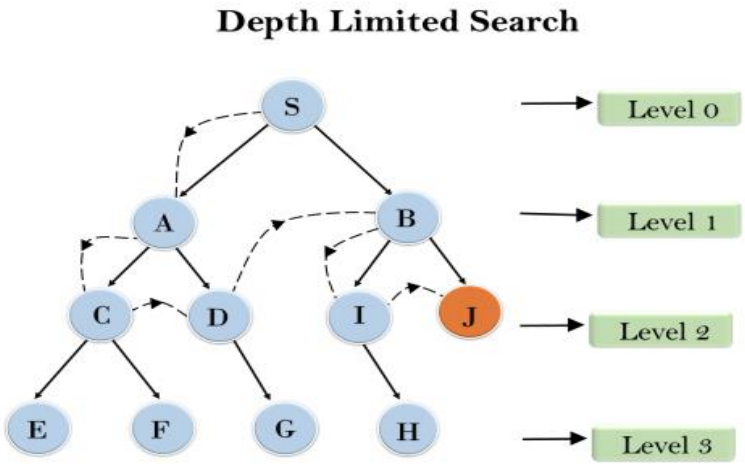
A depth-limited search algorithm is similar to depth-first search with a pre-determined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

- Standard failure value: It indicates that the problem does not have any solution.
- Cutoff failure value: It defines no solution for the problem within a given depth limit

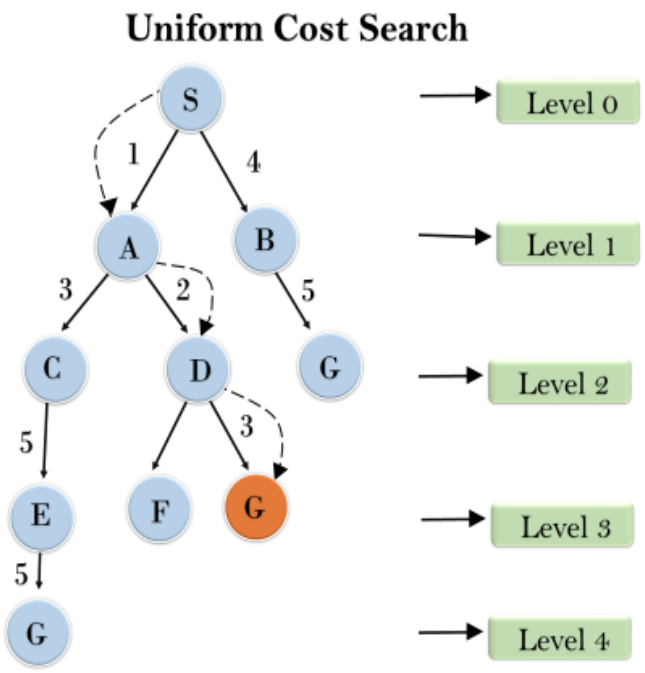
Example:

S---> A--->C---->D--->B---->I--->J



4. Uniform-cost Search Algorithm

Uniform-cost search is a searching algorithm used for **traversing a weighted tree or graph**. This algorithm comes into play when a different cost is available for each edge. The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost. Uniform-cost search expands nodes according to their path costs from the root node. It can be used to solve any graph/tree where the optimal cost is in demand. A uniform-cost search algorithm is implemented by the **priority queue**. It gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.



Example:

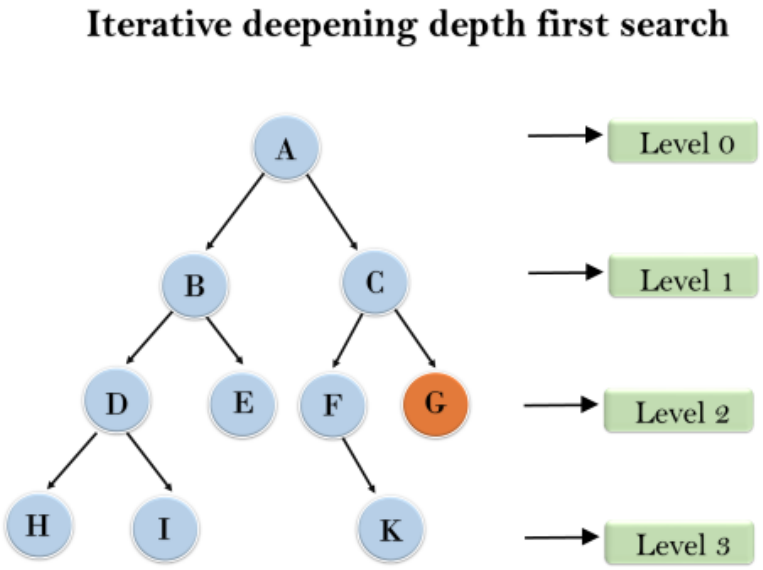
S---> A--->D---->G

5. Iterative deepening depth-first Search

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.
This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.
This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.
The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Example:

Following tree structure is showing the iterative deepening depth-first search. The IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:



1'st Iteration-----> A
2'nd Iteration----> A, B, C
3'rd Iteration------>A, B, D, E, C, F, G
4'th Iteration------>A, B, D, H, I, E, C, F, K, G
In the fourth iteration, the algorithm will find the goal node.

6. Bidirectional Search Algorithm:

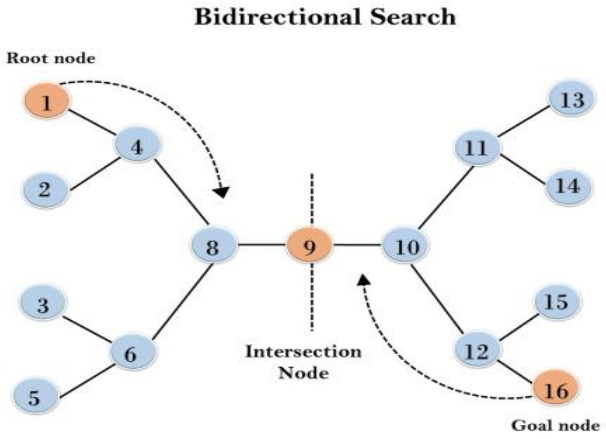
Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node. Bidirectional search replaces one single search graph with two small sub graphs in which one starts the search from an initial vertex and other starts from goal vertex. The search stops when these two graphs intersect each other.

Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

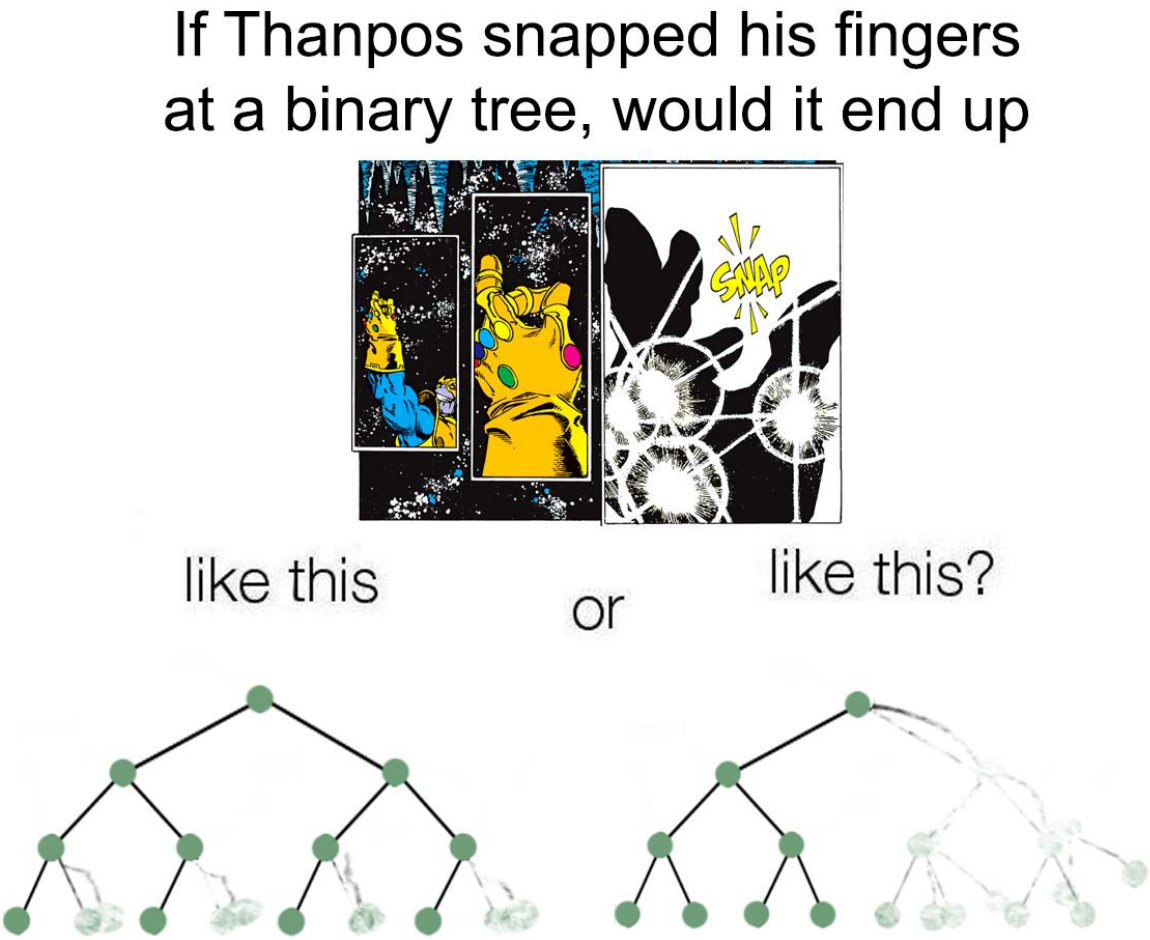
Example:

In the below search tree, a bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.

The algorithm terminates at node 9 where two searches meet.



Search Algorithm	Advantages and disadvantages
1. Breadth-first search	A. Complete and optimal but has space complexity of b^d
2. Depth-first search	B. Has space complexity of bl but may not find a Solution if the depth is not deep enough
3. Depth-limited	C. Complete and Optimal If step-costs are identical but may take longer than breadth-first search
4. Uniform-cost	D. Much faster than the other searches but needs a reverse action
5. Iterative-deepening	E. Has a space complexity of bd but is not complete
6. Bi-directional Search	AB. Complete but has a space complexity of b^d

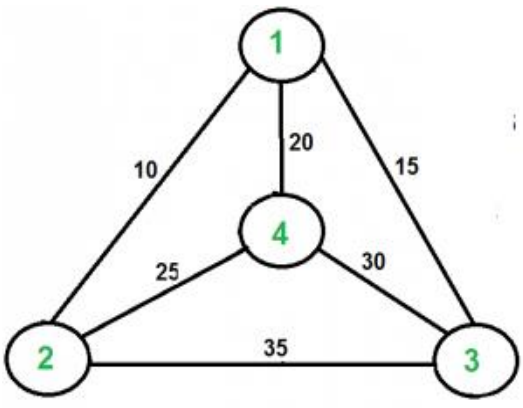


Lab Task 4:

Implement BFS and DFS on either graph or tree.

Lab Task 5:

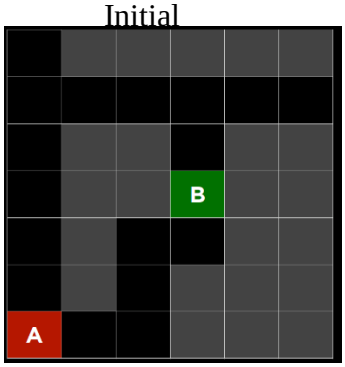
Implement travelling salesman problem on the following map of the city.



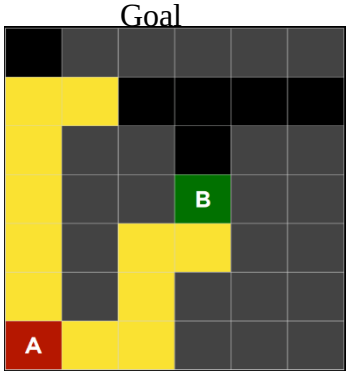
Lab Task 6:

Write a program to solve the Maze problem using the given image using bfs and dfs.
For bfs:

Initial

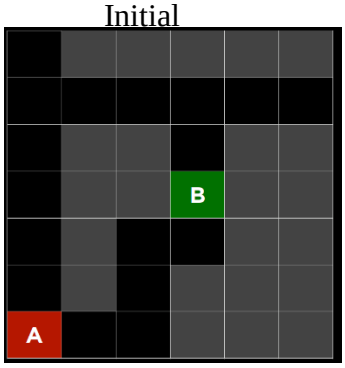


Goal

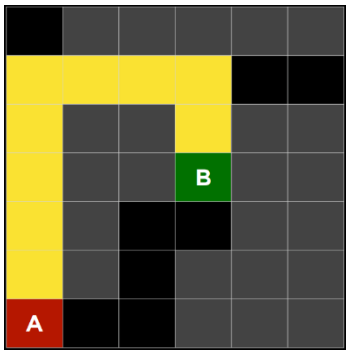


For dfs:

Initial

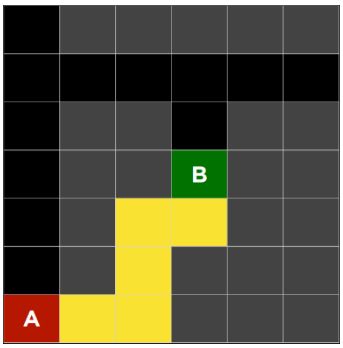


Goal



or

Goal



Lab Task 7:

Write a program to solve the 8-puzzle problem using the DFS and BFS search algorithm.