Chapter 2 Problem Solving, Representation

Rapeeporn Chamchong

Department of CS, Faculty of Informatics, MSU

Content

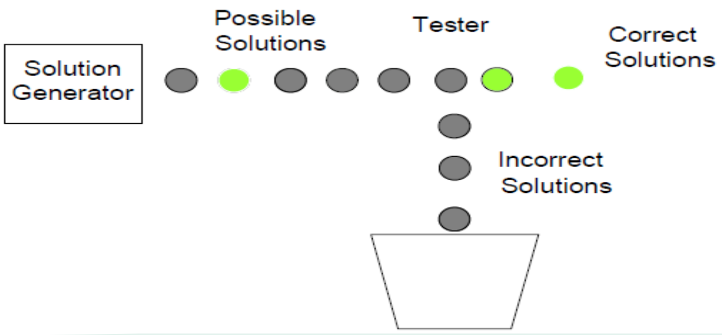
- Classical Approach/Generate and Test
- Systematic Problem Solving approach
  - Problem Representation
    - Graph and Tree Terminologies
  - Problem Solving
    - Components of problem solving
    - Searching

AI as Problem Solving

- Historically many people link intelligence with the capability to solve a problem.
- Problem solving is one of the factors that demonstrate intelligence.

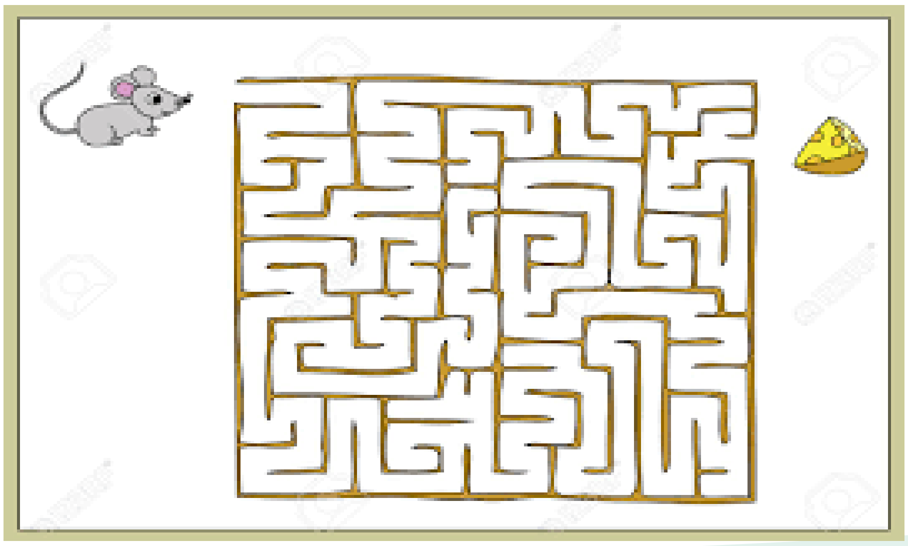
Classical Approach

- The classical approach that is used is "hit and trial" that check for various solutions to that problem.
- It is also known as 'Generate and Test' approach
- We use this approach to solve small problems.



Classical Approach

- Consider the maze searching problem.



Systematic Problem Solving Approach

- For a simple problem or smaller problem we can use hit and trial approach to solve the problem. However when the problem is complex then we need a systematic way to solve the problem
- The key to solving a complex problem is its representation.
- Normally, the problems are represented in the form of diagrams (such as graphs or trees).

Tree and Graph Terminologies

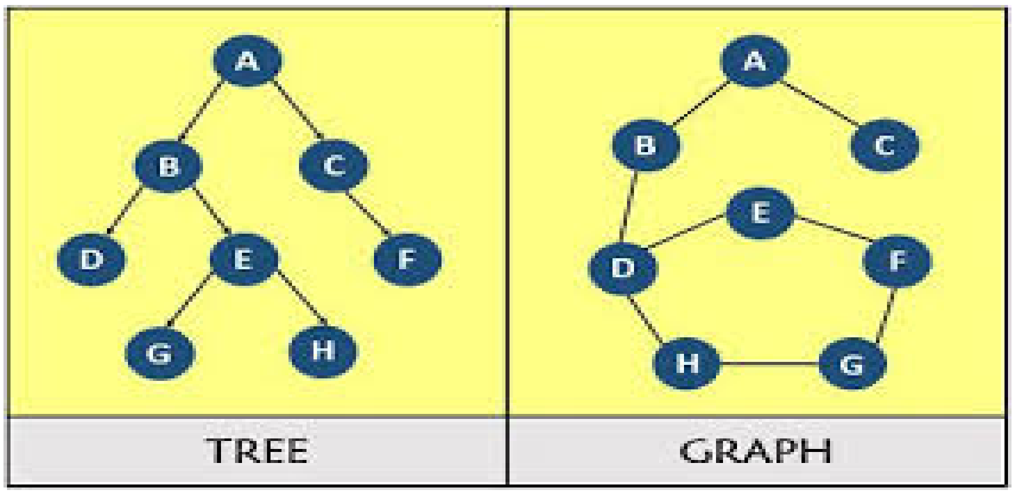
Tree

- A tree is a graph in which two nodes have at most one path between them.
- Trees often have roots that is why it is also known as rooted graph.
- In tree cycles are not allowed which means that it is impossible for a path to loop or cycle continuously through a sequence of nodes.

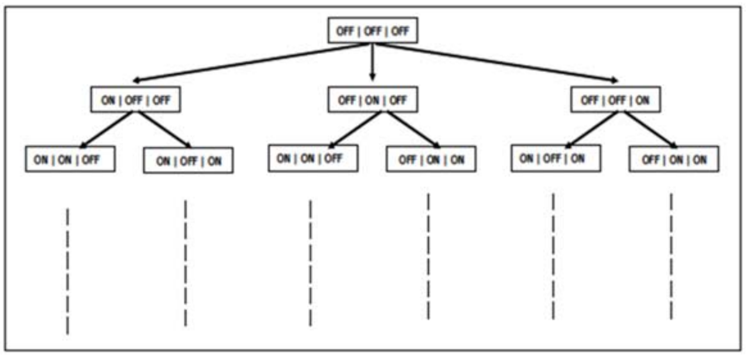
Graph

- A graph is a set of nodes and arc/edges that connect them.
- A graph is said to be directed if its arcs contain directionality using arrows.
- If arcs do not contain an arrow, then this graph is known as an undirected graph.

Tree and Graph Terminologies

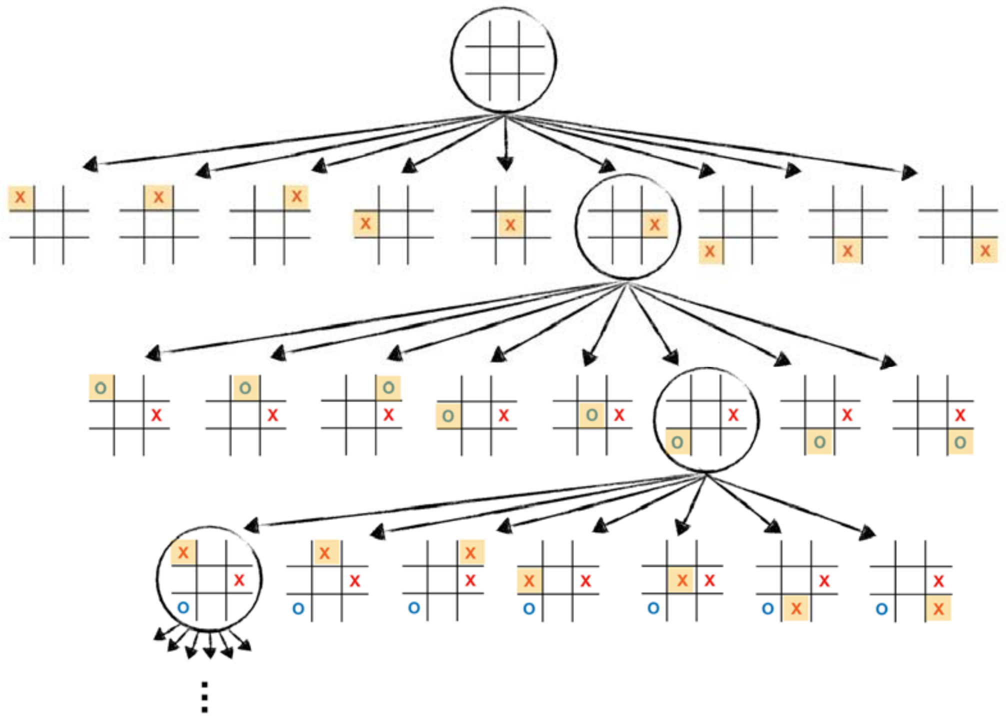


Problem Representation:

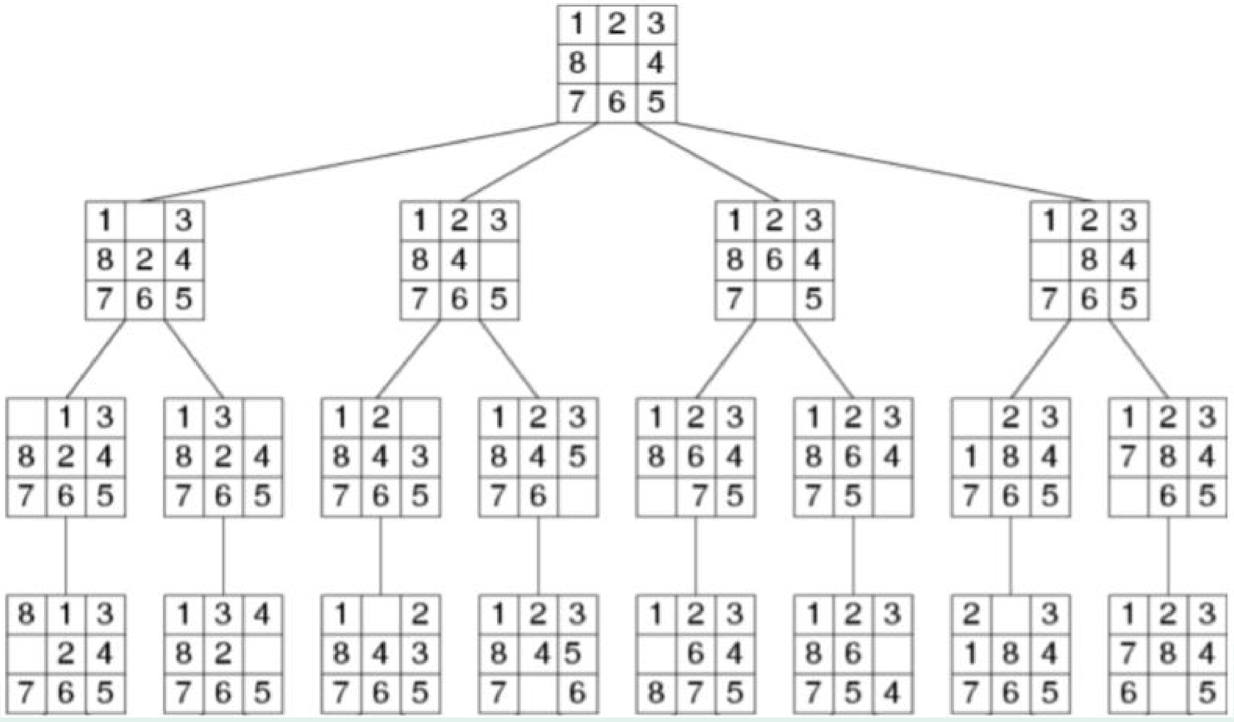


- It shows the problem of switching on the light in a graphical form.
- Each rectangle represents the state of the switchboard.
- OFF | OFF| OFF means that all three switches are OFF.
- OFF| ON | OFF means that the first and the last switch is OFF and the middle one is ON.
- Starting from the state when all the switches are OFF, the person can proceed in any of the three ways by switching either one of the switches ON.
- This brings the person to the next level in the tree. Now, from here, he can explore the other options till he gets to a state where the switch corresponding to the light is ON.

Problem Representation-Tic Tac Toe



Problem Representation- 8-Puzzle Game:



Problem Representation- The Traveling Salesperson

- Suppose a salesperson has five cities to visit and then must return home. The goal of the problem is to find the shortest path for the salesperson to travel, visiting each city, and them returning to the starting city (A)

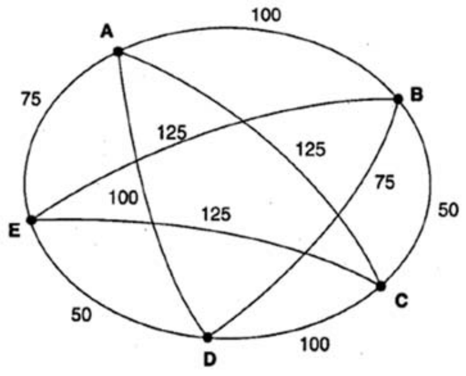
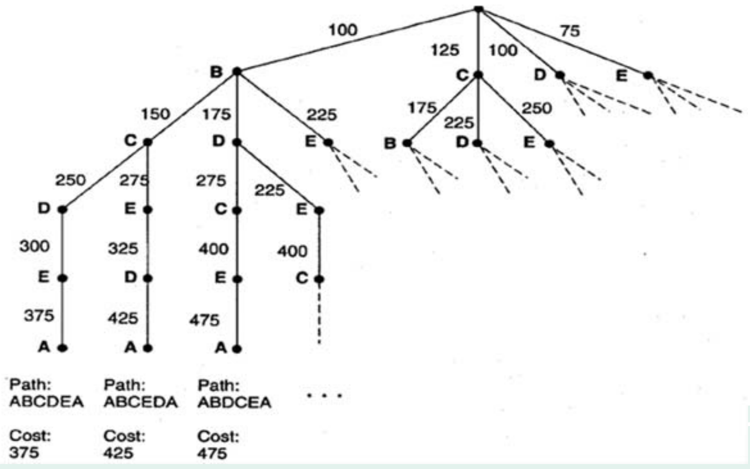
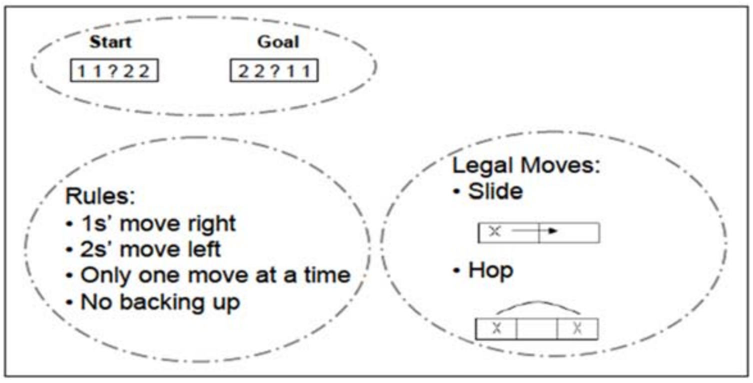


Figure An instance of the traveling salesperson problem.



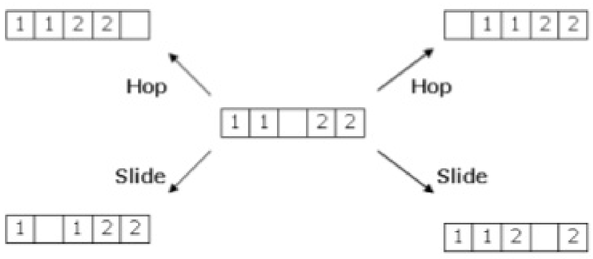
Problem Representation- The Two-One Problem

- The two-one problem is a game in which the starting state is 11?22, and the user needs to move these numbers' slots left/right by following the mentioned rules to obtain sequence 22?11.

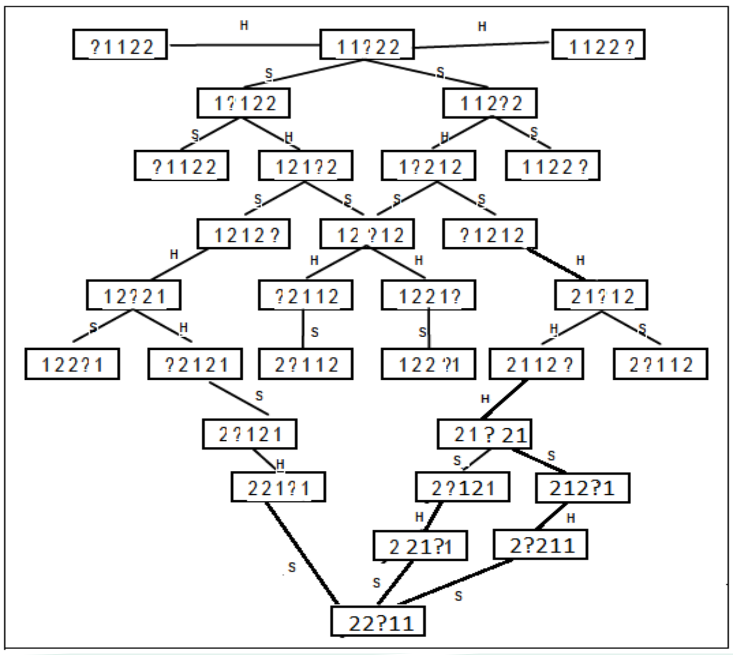


- The above game can be better represented as a tree to find the possible solutions. A tree sort of structure enumerates all the possible states and moves. Looking at this diagram we can easily figure out the solution to our problem.

Problem Representation- The Two-One Problem



All possibilities of first state



Components of Problem Solving:

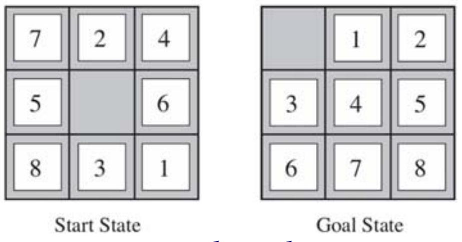
- Problem Statement: Problem at hand
- Goal State: The destination
- Search Space: The entire tree/graph
- Operators: Rules/actions to move in solution space

- Once the problem is represented, the goal state can be searched in search space using different search tactics

The Standard Formulation

- States
- Initial state
- Goal state
- Actions
- Transition model
- Goal test

Example-Standard Formulation



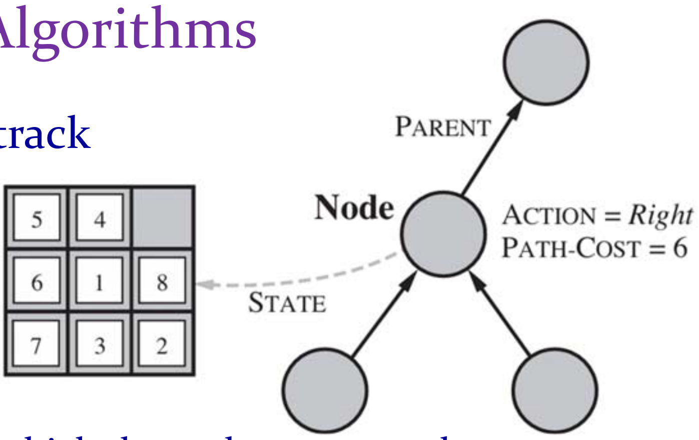
- **States:** A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
- **Initial state:** Any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states.
- **Goal state:** The objective of the activity/event.
- **Actions:** The simplest formulation defines the actions as movements of the blank space Left, Right, Up, or Down. Different subsets of these are possible depending on where the blank is.
- **Transition model:** Given a state and action, this returns the resulting state; for example, if we apply Left to the start state in Figure, the resulting state has the 5, and the blank switched.
- **Goal test:** This checks whether the state matches the goal configuration shown in Figure. (Other goal configurations are possible.)

Searching

- All the problems that we have looked at can be converted to a form where we have to start from a start state and search for a goal state by traveling through a search space.
- Searching is a formal mechanism to explore alternatives.
- If we can get our grips on algorithms that deal with searching techniques in graphs and trees, we'll be all set to perform problem solving in an efficient manner.

Infrastructure for Search Algorithms

- A data structure is required to keep track of the search tree that is being constructed. For each node n of the tree that contains 4 components:



  - n.STATE: the state in the state space to which the node corresponds;
  - n.PARENT: the node in the search tree that generated this node;
  - n.ACTION: the action that was applied to the parent to generate the node;
  - n.PATH-COST: the cost, traditionally denoted by $g(n)$, of the path from the initial state to the node, as indicated by the parent pointers.

- Given the components for a parent node, it is easy to see how to compute the necessary components for a child node. The function CHILD-NODE takes a parent node and an action and returns the resulting child node:

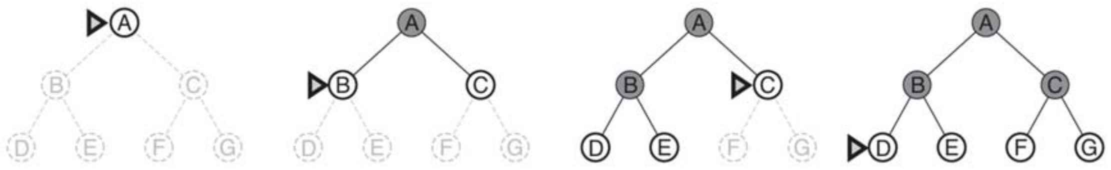
```
function CHILD-NODE(problem, parent, action) returns a node
  return a node with
      STATE = problem.RESULT(parent.STATE, action),
      PARENT = parent, ACTION = action,
      PATH-COST = parent.PATH-COST + problem.STEP-COST(parent.STATE, action)
```

Chapter 3
Search and Applications

Searching Strategies

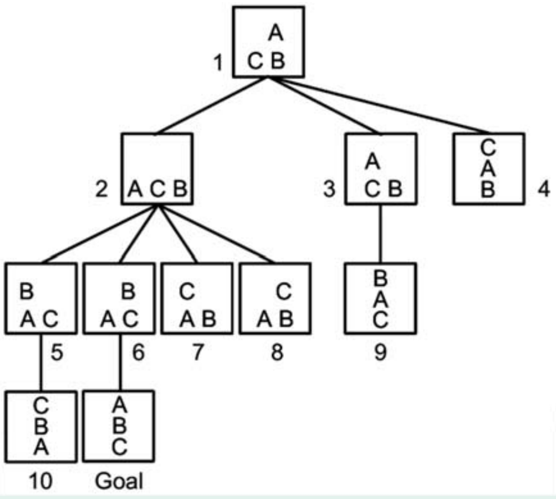
- Blind/uninformed
  - Breath-first Search (BFS)
  - Depth-first Search (DFS)
  - DFS with iterative deepening
- Informed/heuristic
  - Best First Search
    - Greedy Approach
    - A* search
- Any path/non-optimal
- Optimal path search algorithms

Breath First Search (BFS)



BFS on a simple binary tree. The node is expanded next by a marker.

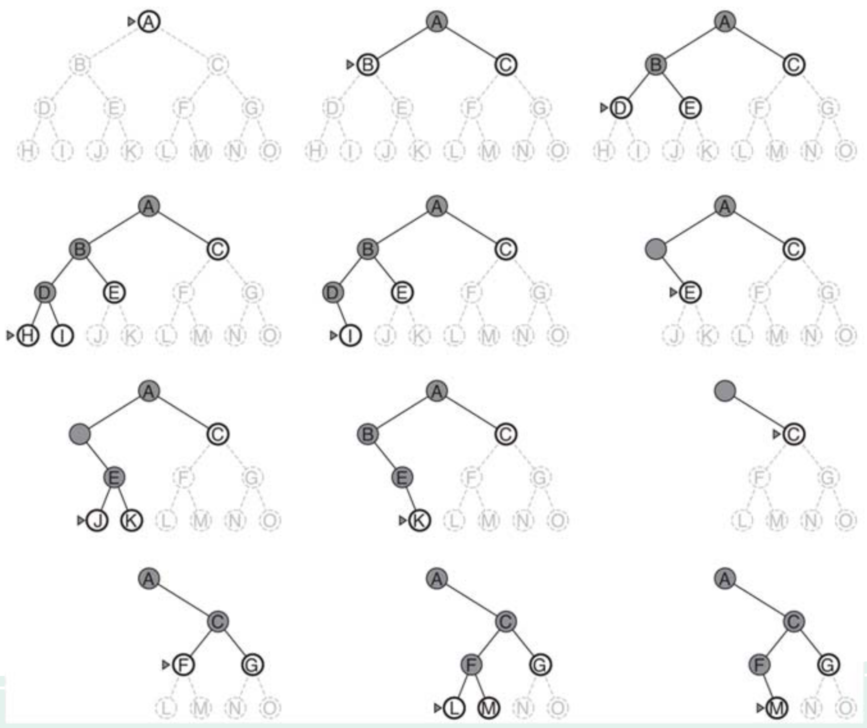
BFS on a tree of block problem.



BFS Algorithm

```
BFS(problem) returns a solution node or failure
    node ← NODE(problem.INITIAL)
    if problem.IS-GOAL(node.STATE) then return node
    frontier ← a FIFO queue, initialized with node
    reached ← {problem.INITIAL}
    while not IS-EMPTY(frontier) do
        node ← POP(frontier)
        for each child in EXPAND(problem, node) do
            s ← child.STATE
            if problem.IS-GOAL(s) then return child
            if s is not in reached then
                add s to reached
                add child to frontier
    return failure
```

Depth-first Search (DFS)-Backtracking search



DFS on a binary tree.

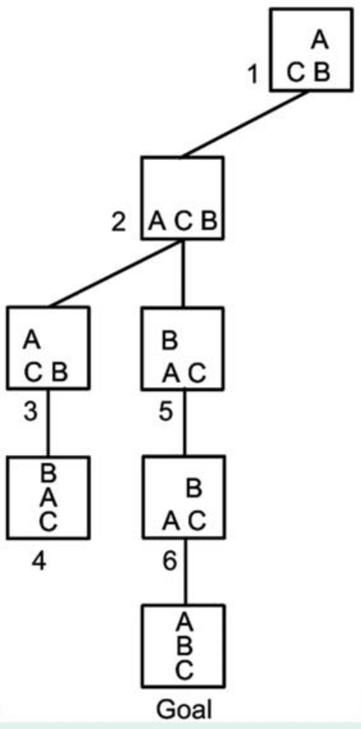
The unexplored region is shown in light gray.

Explored nodes with no descendants in the frontier are removed from memory.

Nodes at depth 3 have no successors and *M* is the only goal node.

DFS Algorithm

```
DFS(graph, start, goal):
    stack ← empty stack
    visited ← empty set
    stack.push(start)
    while not stack.isEmpty( ):
        current ← stack.pop( )
        if current == goal:
            return “Success"
        visited.add(current)
        neighbors ← graph.getNeighbors(current)
        for neighbor in neighbors:
            if neighbor not in visted:
                stack.push(neighbor)
    return “Failure"
```

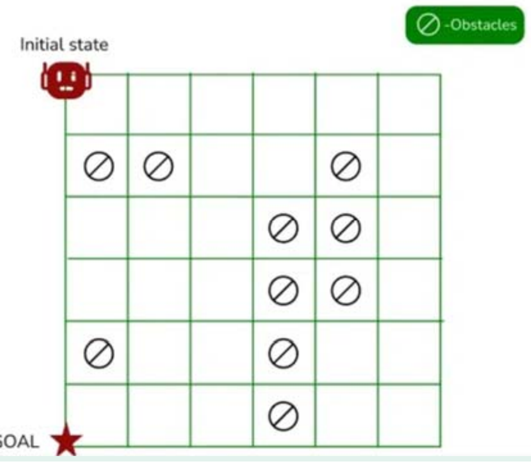


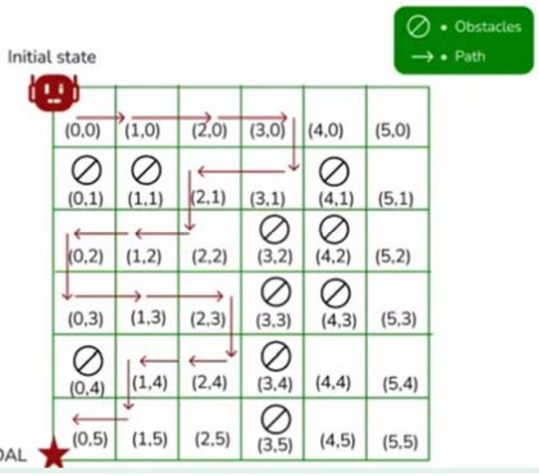
Assignment 2

- Write the function BFS
- Write the function DFS
- Use both algorithms to solve the binary tree and block problem on pages 50, 52, 53

Assignment 3

- Solve the pathfinding problem of robotics.
  - States: size=(6x6)
  - Initial state: (0,0)
  - Goal state: (0,5)
  - Obstacles: (0,1),(1,1),(3,2),(3,3),(3,4),(3,5), (0,4),(4,1),(4,2),(4,3)


Initial state
-Obstacles
GOAL


Initial state
• Obstacles
• Path
(0,0)
(1,0)
(2,0)
(3,0)
(4,0)
(5,0)
(0,1)
(1,1)
(2,1)
(3,1)
(4,1)
(5,1)
(0,2)
(1,2)
(2,2)
(3,2)
(4,2)
(5,2)
(0,3)
(1,3)
(2,3)
(3,3)
(4,3)
(5,3)
(0,4)
(1,4)
(2,4)
(3,4)
(4,4)
(5,4)
(0,5)
(1,5)
(2,5)
(3,5)
(4,5)
(5,5)
GOAL