Introduction to Artificial Intelligence

Unit # 2

Acknowledgement

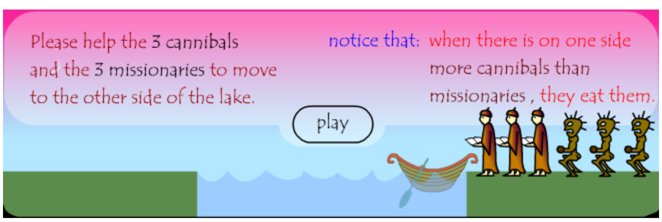
• Slides of this lecture have been taken from the lecture slides of CS307 – "Introduction to Artificial Intelligence" by Dr. Sajjad Haider.

Problem Solving as Search

- Problem solving is an important aspect of Artificial Intelligence.
- A problem can be considered to consist of a goal and a set of actions that can be taken to lead to the goal.
- Search can be defined as a problem solving technique that enumerates a problem space from an initial position in search of a goal position (or solution).
- At any given time, we consider the state of the search space to represent where we have reached as a result of the actions we have applied so far.

Missionaries and Cannibals

Three missionaries and three cannibals are on one side of a river, with a canoe. They all want to get to the other side of the river. The canoe can only hold one or two people at a time. At no time should there be more cannibals than missionaries on either side of the river, as this would probably result in the missionaries being eaten.

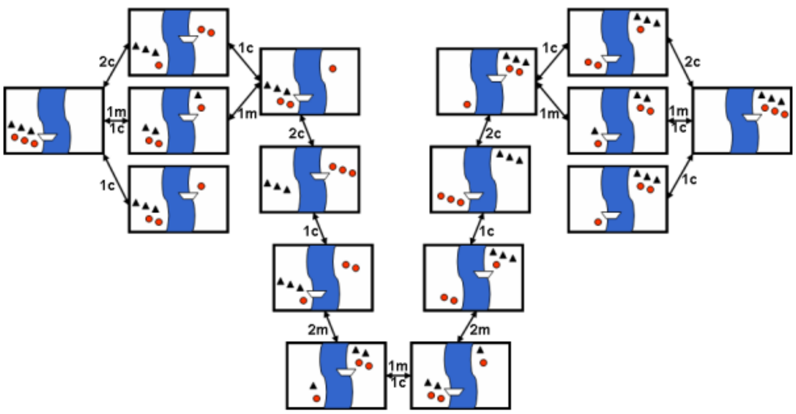


Missionaries and Cannibals (Cont’d)

- The following operators are available:
  1. Move one cannibal to the other side
  2. Move two cannibals to the other side
  3. Move one missionary to the other side
  4. Move two missionaries to the other side
  5. Move one cannibal and one missionary to the other side

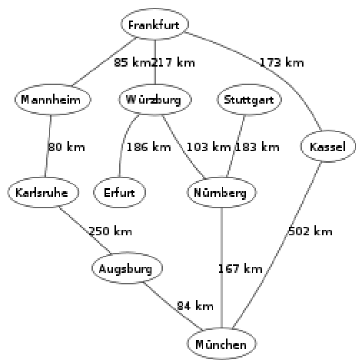
  – http://www.learn4good.com/games/puzzle/boat.htm

Missionaries and Cannibals (Cont’d)



Recap

- Graph
  - Nodes
  - Edges
  - Directed vs Undirected
  - Weighted Graph
- Tree



The Towers of Hanoi

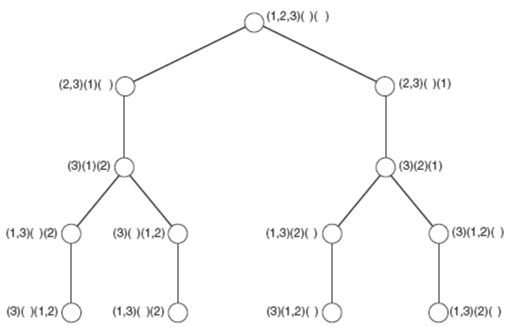
We have three pegs and a number of disks of different sizes. The aim is to move from the starting state where all the disks are on the first peg, in size order (smallest at the top) to the goal state where all the pegs are on the third peg, also in size order. We are allowed to move one disk at a time, as long as there are no disks on top of it, and as long as we do not move it on top of a peg that is smaller than it.

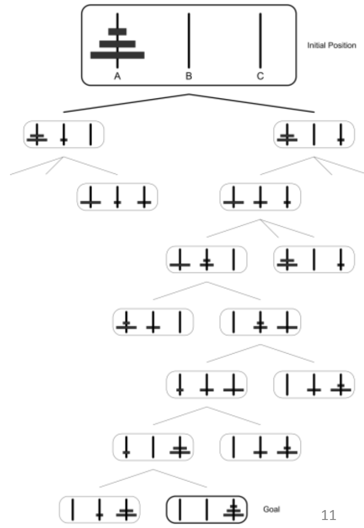
The Towers of Hanoi (From Wikipedia)

- The puzzle was invented by the French mathematician Édouard Lucas in 1883. There is a legend about an Indian temple which contains a large room with three time-worn posts in it surrounded by 64 golden disks. The priests of Brahma, acting out the command of an ancient prophecy, have been moving these disks, in accordance with the rules of the puzzle. According to the legend, when the last move of the puzzle is completed, the world will end. The puzzle is therefore also known as the Tower of Brahma puzzle.
- If the legend were true, and if the priests were able to move disks at a rate of one per second, using the smallest number of moves, it would take them 264−1 seconds or roughly 584.542 billion years [operation taking place is 2^64/60/60/24/365.25(to take into consideration leap years)/1000000000] .[1] (In context, the universe is currently about 13.7 billion years old.)

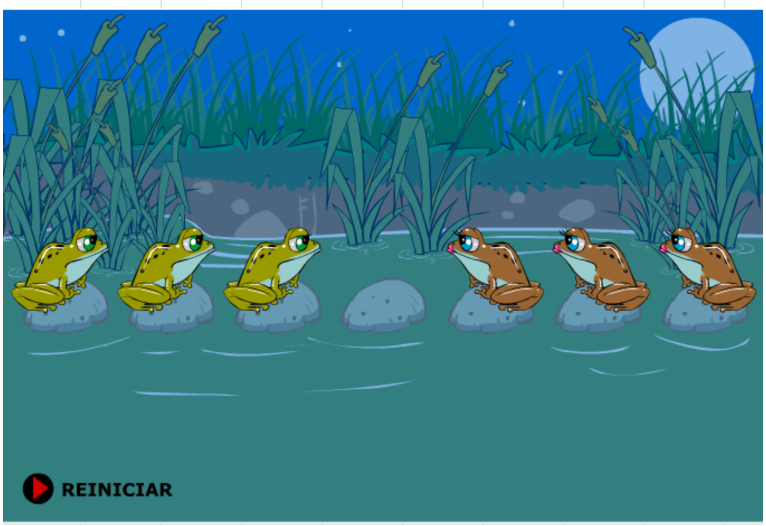
The Towers of Hanoi (Cont'd)

The Towers of Hanoi (Partial Search Tree)

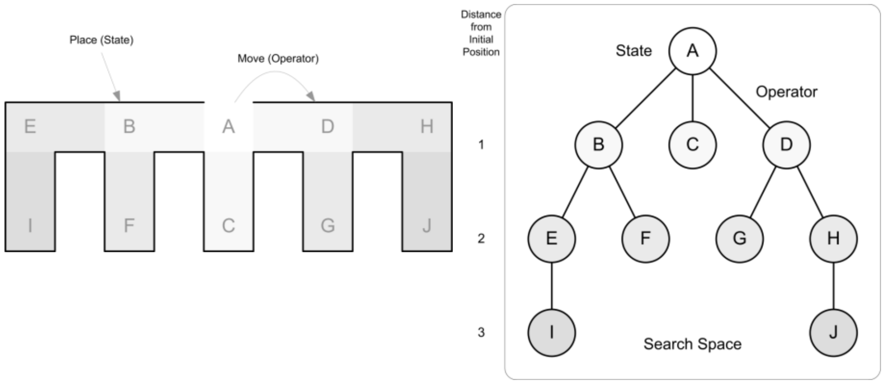




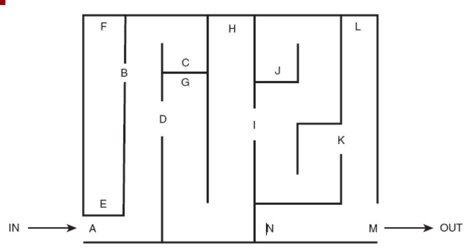
Frog Problem

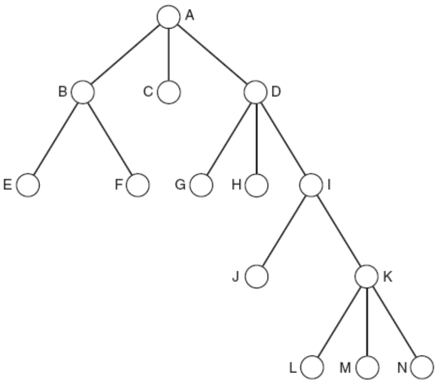


Representing Physical Space as Tree

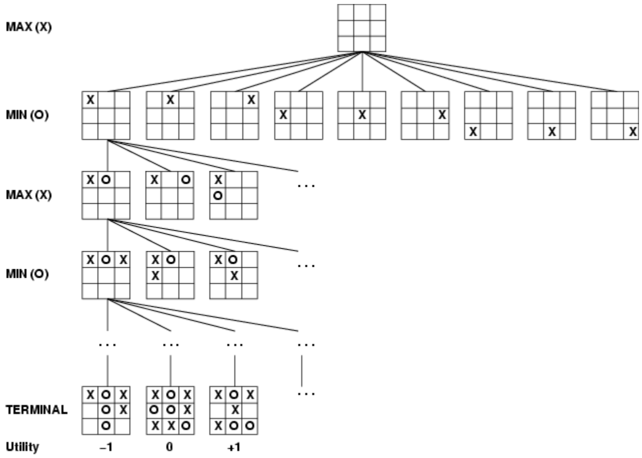


Traversing a Maze





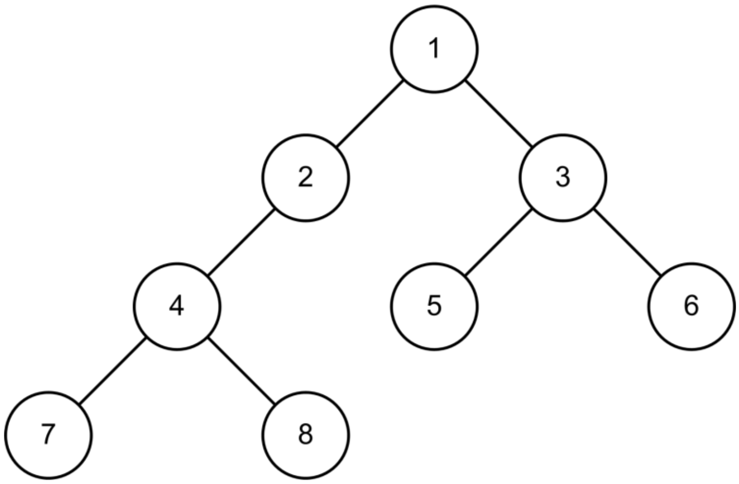
Search Trees for Game



Breadth-first Search

- In breadth-first search (BFS), we search the graph from the root node in order of the distance from the root.
- Rather than digging deep down into the graph, progressing further and further from the root (as is the case with DFS), BFS checks each node nearest the root before descending to the next level.
- The implementation of BFS uses a FIFO (first-in first-out) queue.

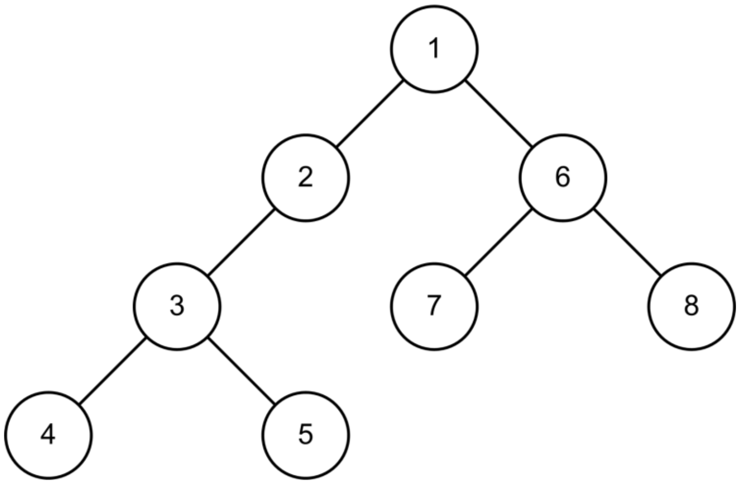
Search Order of the BFS



Depth-first Search

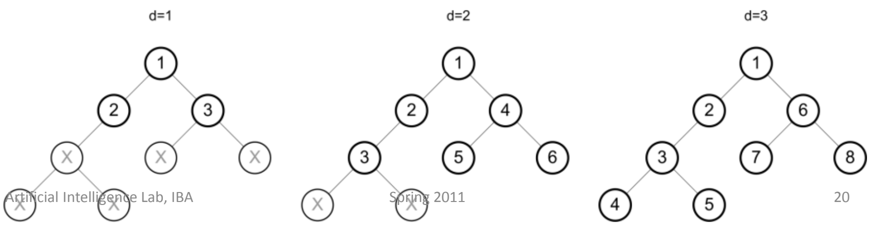
- The Depth-first search (DFS) algorithm is a technique for searching a graph that begins at the root node, and exhaustively searches each branch to its greatest depth before backtracking to previously unexplored branches.
- The implementation of DFS uses a LIFO (last-in first-out) queue.

Search Order of the DFS



Iterative Deepening Search (IDS)

- IDS combines the features of DFS with that of BFS.
- IDS operates by performing searches with increased depths until the goal is found.
- The depth begins at one, and increases until the goal is found, or not further nodes can be enumerated.
- It combines the efficiency of memory use of depth-first search with the advantage that branches of the search tree that are infinite or extremely large will not sidetrack the search.
- It also shares the advantage of breadth-first search that it will always find
- the path that involves the fewest steps through the tree



Properties of Search Methods

- Completeness
  - A search method is described as being **complete** if it is guaranteed to find a goal state if one exists.
  - Breadth-first search is complete, but depth-first search is not because it may explore a path of infinite length and never find a goal node that exists on another path.
  - Completeness is usually a desirable property because running a search method that never finds a solution is not often helpful.
  - A method that is not complete has the disadvantage that it cannot necessarily be believed if it reports that no solution exists.

Properties of Search Methods

- Optimality
  - A search method is **optimal** if it is guaranteed to find the best solution that exists.
  - This does not mean that the search method itself is efficient—it might take a great deal of time for an optimal search method to identify the optimal solution—but once it has found the solution, it is guaranteed to be the best one.
  - Breadth-first search is an optimal search method, but depth-first search is not. Depth-first search returns the first solution it happens to find, which may be the worst solution that exists. Because breadth-first search examines all nodes at a given depth before moving on to the next depth, if it finds a solution, there cannot be another solution before it in the search tree.

Depth-first vs. Breadth-first

Scenario	Depth first	Breadth first
Some paths are extremely long, or even infinite	Performs badly	Performs well
All paths are of similar length	Performs well	Performs well
All paths are of similar length, and all paths lead to a goal state	Performs well	Wasteful of time and memory
High branching factor	Performance depends on other factors	Performs poorly

Algorithm	Optimal	Complete
DFS	No	No
BFS	Yes	Yes
IDS	Yes	Yes