Introduction to Artificial Intelligence

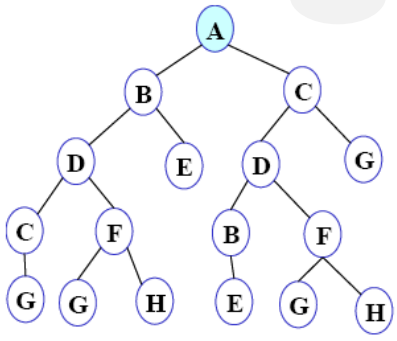
Unit # 3

Acknowledgement

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

Recap: BFS, DFS, IDS

- Start Node: A
- Goal Node: G
- Identify the path taken by each scheme
- Identify the number of nodes explored by each scheme



BFS and DFS Algorithms

BFS

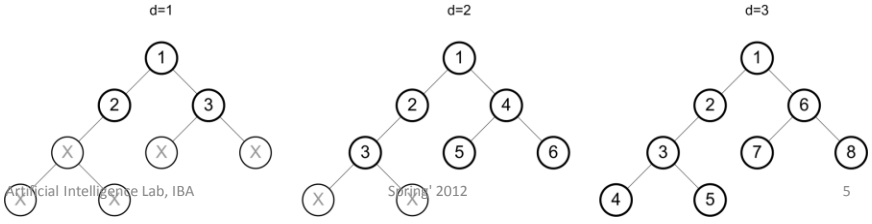
- Let L be a list containing the initial state
- Loop
  - If L is empty return failure
  - Node ← remove-first(L)
  - If Node is a goal
    - Return the path from initial state of Node
  - Else
    - Generate all successors of Node
    - Add generated nodes to the ***back*** of L
- End Loop

DFS

- Let L be a list containing the initial state
- Loop
  - If L is empty return failure
  - Node ← remove-first(L)
  - If Node is a goal
    - Return the path from initial state of Node
  - Else
    - Generate all successors of Node
    - Add generated nodes to the ***front*** of L
- End Loop

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

Search Methods

- Uninformed (Blind) Search
  - Breadth-first
  - Depth-first
  - Depth-limited
  - Iterative deepening depth-first
- Informed (or heuristic) Search
  - Best-first
  - A*
- Adversarial
  - Minimax
  - Alpha-beta Pruning

Problem Solving by Search

- Define search space
  - Initial, goal, and intermediate states
- Define operators for expanding a given state into its possible successor states
- Apply search algorithm to find path from initial to goal state, while avoiding a repeating state during the search.

The 8-puzzle

- The puzzle consists of a 3 x 3 grid, with the numbers 1 through 8 on tiles within the grid and one blank square.
- Tiles can be slid about within the grid, but a tile can only be moved into the empty square if it is adjacent to the empty square.
- The start state of the puzzle is a random configuration, and the goal state is as shown in the picture below.

7	6	
4	3	1
2	5	8

1	2	3
8		4
7	6	5

Goal State

Heuristic

- Depth-first and breadth-first search are described as brute-force search methods.
- They do not employ any special knowledge of the search trees they are examining but simply examine every node in order until they happen upon the goal.
- A heuristic is a rule of thumb that may help solve a given problem. Heuristics take problem knowledge into consideration to help guide the search within the domain.

Heuristics
(Source: Wikipedia)

- **Heuristic** refers to experience-based techniques for problem solving, learning, and discovery. Heuristic methods are used to speed up the process of finding a good enough solution, where an exhaustive search is impractical. Examples of this method include using a "rule of thumb", an educated guess, an intuitive judgment, or common sense.

Heuristic Function

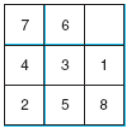
- A **heuristic evaluation function** is a function that when applied to a node gives a value that represents a good estimate of the distance of the node from the goal.
- For two nodes *m* and *n*, and a heuristic function *f*, if $f(m) < f(n)$, then it should be the case that *m* is more likely to be on an **optimal path** to the goal node than *n*.

Basic Idea Behind a Heuristic Search

- We assume that we have a heuristic (evaluation) function, f, to help decide which node is the best one to expand next. This function is based on information specific to the problem domain.
- Expand next that node, n, having the smallest value of f(n). Resolve ties arbitrarily.
- Terminate when the node to be expanded next is a goal node.

Heuristic # 1

- The first heuristic we consider is to count how many tiles are in the wrong place. We will call this heuristic, *h*1(node).
- In the case of the first state shown in Figure 4.5, *h*1 (node) = 8 because all the tiles are in the wrong place.
- However, this is misleading because we could imagine a state with a heuristic value of 8 but where each tile could be moved to its correct place in one move.



Heuristic # 2

- An improved heuristic, *h*2, takes into account how far each tile had to move to get to its correct state.
- This is achieved by summing the **Manhattan distances** of each tile from its correct position.
- Manhattan distance is the sum of the horizontal and vertical moves that need to be made to get from one position to another, named after the grid system of roads used in Manhattan.
- *h*2 (node) = 2 + 2 + 2 + 2 + 3 + 3 + 1 + 3 = 18
- It is worth noting that *h*2 (node) ≥ *h*1 (node) for any node. This means that *h*2 **dominates** *h*1, implying that a search method using heuristic *h*2 will always perform more efficiently than the same search method using *h*1.