UNIT 2

Search Methodologies

Problem Solving as Search

- A problem consists of a **goal** and a set of actions that can be taken to lead to the goal.
- Consists of **initial state, goal state** and **intermediate states**
- **Search** is a method computers can use to examine a problem space to find a goal.
- A problem space is a search space because to solve the problem, we will search the space for a goal state.

Approaches

Two main approaches to searching a search tree

1. Data Driven and 2. Goal Driven

1) Data Driven:

- Starts from an initial state and uses actions that are allowed to move forward until a goal is reached.
- This approach is also known as **forward chaining**.

Ex: Analyzing astronomical data to make deductions about the nature of stars and planets (goal is not known here)

2) Goal Driven:

- Search starts at the goal and works back toward a start state, by seeing what moves could have led to the goal state.
- This approach is also known as **backward chaining**

Ex: Maze problem - easier to start from the end point and work back toward the start point. Because of many dead end paths set up from the start (data), and only one path set up to the end (goal) point

Generate and Test

- Involves generating each node in the search space and testing it to see if it is a goal node.
- Simplest form of **brute-force search** (also called **exhaustive/blind search**)
  - Assumes **no additional knowledge** but how to traverse the search tree and how to identify leaf nodes and goal nodes,
  - Examines every node in the tree until it finds a goal.
- Needs to have a suitable **Generator**, which should satisfy three properties:
  1. It must be **complete**: must generate every possible solution
  2. It must be **nonredundant**: not generate the same solution twice.
  3. It must be **well informed**: should not examine possible solutions that do not match the search space

DFS and BFS

Self Study

Properties of Search Methods

1. **Complexity**
   - The **time complexity** of a method is related to the length of time to find a goal state.
   - The **space complexity** is related to the amount of memory
   - BFS has a time complexity of $O(b^d)$ and DFS has $O(b^d)$
   - BFS has a space complexity of $O(b^d)$ and DFS has $O(d)$
2. **Completeness**
   - A search method is described as being **complete** if it is guaranteed to find a goal state if one exists
   - BFS is complete, but DFS is not because it may explore a path of infinite length and never find a goal node that exists on another path.

3. **Optimality**
   - A search method is **optimal** if it is guaranteed to find the best solution that exists.
   - BFS is optimal and DFS is not
4. **Admissibility**
   - An algorithm that finds a solution in the quickest possible time is called admissible
5. **Irrevocability**
   - Methods that do not use backtracking, and which therefore examine just one path, are irrevocable.
   - DFS is an example of revocable search and BFS is irrevocable

Depth-First Iterative Deepening

- Is an exhaustive search technique that **combines depth-first** with **breadth-first search.**
- The DFID algorithm involves repeatedly carrying out DFS on the tree, starting with a DFS limited to a depth of one, then a depth-first search of depth two, and so on, until a goal node is found.
- Combines the memory efficiency of DFS and time efficiency of BFS (finds the path that involves the fewest steps)

Calculation of Number of Nodes in a Tree

- For a tree of depth *d* and with a branching factor of *b*, the total number of nodes is

1 root node
b nodes in the first layer
b^2 nodes in the second layer
. . .
b^n nodes in the n^{th} layer

Hence, the total number of nodes is

$$1 + b + b^2 + b^3 + \ldots + b^d$$

which is a **geometric progression** equal to

$$\frac{1 - b^{d+1}}{1 - b}$$

- Using DFID, nodes must be examined more than once, resulting in the following progression:

$$(d+1) + b(d) + b^2(d-1) + b^3(d-2) + \ldots + b^d$$

Ex:

With a depth of 4 and a branching factor of 10, the tree has the following number of nodes:

$$\frac{1-10^5}{1-10} = 11{,}111 \text{ nodes}$$

DFID will examine the following number of nodes:

$$(4+1) + 10 \times 4 + 100 \times 3 + 1{,}000 \times 2 + 10{,}000 = 12{,}345 \text{ nodes}$$

Informed and Uninformed Methods

- A search method or heuristic is **informed** if it uses additional information about nodes to decide which nodes to examine next.
- If a method does not use additional information is **uninformed**, or **blind**.
- In other words, search methods that use heuristics are informed, and those that do not are blind.

Choosing a Good Heuristic

- Some heuristics are better than others, and the better (more informed) the heuristic is, the fewer nodes it needs to examine in the search tree to find a solution.

The 8-puzzle

- The puzzle consists of a 3X3 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.

Start State

7	6	
4	3	1
2	5	8

Goal State

1	2	3
8		4
7	6	5

- The first heuristic we consider is to count how many tiles are in the wrong place. We will call this heuristic, h_1(node).

  h1 = 8 (all are in wrong place)

- 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.

  h2 (node) = 2 + 2 + 2 + 2 + 3 + 3 + 1 + 3 = 18

Monotonicity

- A search method is described as **monotone** if it always reaches a given node by the shortest possible path.

Using Heuristics

- DFS and BFS are brute-force search methods. This is because they do not employ any special knowledge of the search trees
- Heuristic is defined as the additional information or knowledge that helps in searching the solution in Search trees

Best First Search

- **Best-first search** employs a heuristic
- It uses an **open queue** to maintain the unvisited nodes
- A **closed queue** is used to maintain the visited and closed nodes

Function of Best First Search

```
Function best ()
{
      queue = [];      // initialize an empty queue
      state = root_node;    // initialize the start state
      while (true)
      {
            if is_goal (state)
                  then return SUCCESS
            else
           {
                  add_to_front_of_queue (successors (state));
                  sort (queue);
           }
           if queue == []
                  then report FAILURE;
           state = queue [0]; // state = first item in queue
           remove_first_item_from (queue);
      }
}
```