State Space Search in AI

Prof. (Dr.) K.R. Chowdhary
Email: kr.chowdhary@jietjodhpur.ac.in

Campus Director,
JIET College of Engineering, Jodhpur

Tuesday 22nd August, 2017

Outlines:

- Introduction
- Evaluation of search strategies
- AI search is graph search
- Uninformed Search
- Breadth-First Search
- Breadth-First Search Algorithm
- Depth-First Search
- Breadth-First Search Algorithm
- Analysis of BFS and DFS
- Depth-First Iterative Deepening (DFID) Search
- Bidirectional Search
- Problem formulation for Search

Introduction

The search space is generally represented as a directed graph (in fact a tree), with vertices as states, and edges of the graph as transitions for moves to explore for the goal state.

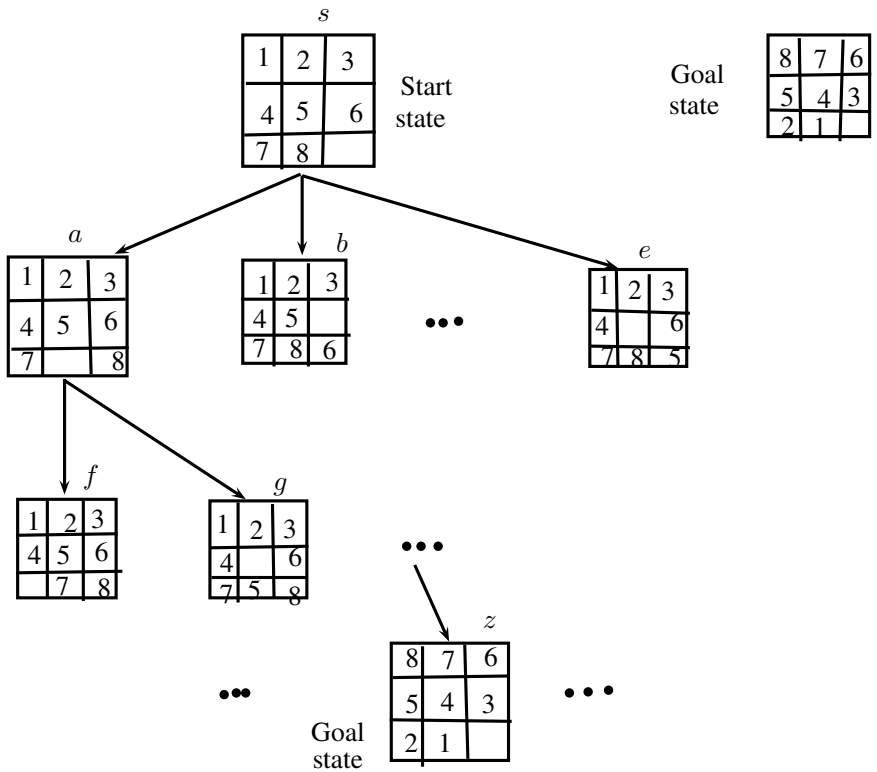


Figure 1: The 8-puzzle Game.

Evaluation of Search Strategies

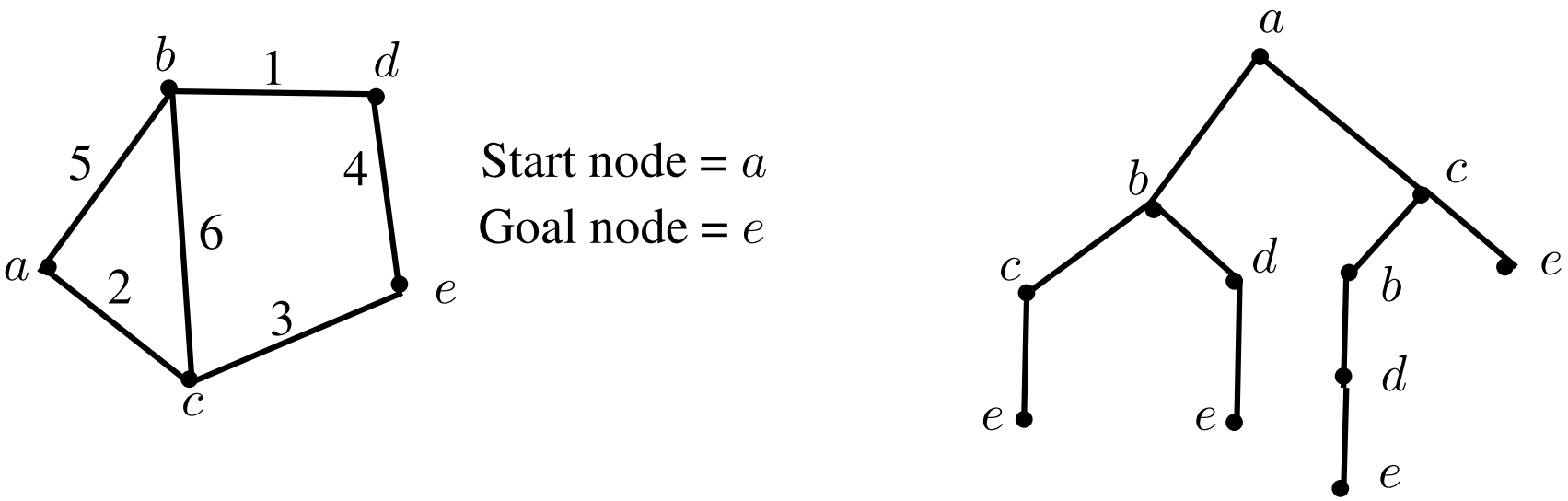
The Search Strategies are evaluated along the following dimensions:

- *Completeness:* Does it always find a solution if one exists?
- *Time complexity:* What is to be Number of nodes generated ? The duplicate nodes generated due to multiple paths, are also counted.
- *Space complexity:* What is maximum number of nodes in memory at any time?
- *Optimality:* Does it always find a least-cost solution?

AI search is graph search

Example

Consider the graph shown in figure 2 for a small size problem, where it is required to reach to goal state g from the start state a.



(a) Graph with states a-e. (b) Graph search is a tree.

Figure 2: Undirected graph.

Uninformed Search

- It is also called blind search, because we do not know, which direction move(s) ultimately will lead to goal faster.
- Since all the nodes are required to be searched - the search is called *exhaustive* search.
- To search the entire state space, the two important approaches are - *breadth-first search* (BFS) or *depth-first search* (DFS).

Breadth-First Search

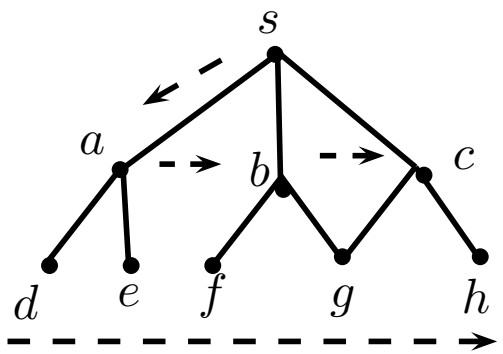


Figure 3: Breadth-first Search.

The *BFS* can be implemented using a *queue* type data structure, named here as **List**. The front node of the of the queue is represented by **List**.*Head*.

Breadth-First Search Algorithm

Algorithm 1 BFS(Input: **G**, **S**, **Goal**)

```
1:  List = [S]
2:  repeat
3:      if List.Head = Goal then
4:          return success
5:      end if
6:      generate children set C of List.Head
7:      append C to List
8:      delete List.Head
9:  until List = []
10: return fail
```

Depth-First Search

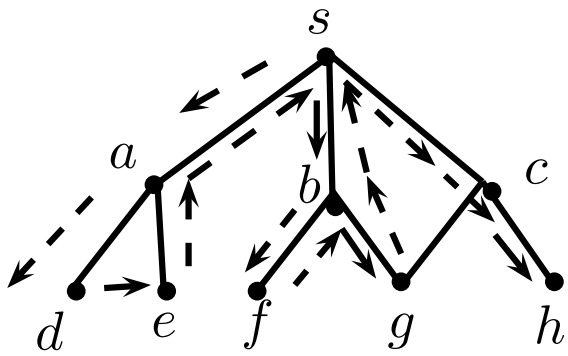


Figure 4: Depth-first Search.

To perform a DFS search, we generate all the next states for the root node, then pickup the left-most node, generate the children for this, check for goal, and repeat this, until we reach to goal or the dead end.

Depth-First Search Algorithm

Algorithm 2 DFS(Input: **G**, **S**, **Goal**)

```
1:  List = [S]
2:  repeat
3:      if List.Head = Goal then
4:          return success
5:      end if
6:      generate children set C of List.Head
7:      delete List.Head
8:      insert C at begin of List
9:  until List = []
10: return fail
```

Analysis of BFS and DFS

- **BFS:**
- Let us assume that in the tree constructed in a search has depth d and for each node there are b nodes that gets generated, called *branching factor*
- For a *BFS* search, total nodes visited for tree of depth d are:

$$1 + b + b^2 + \cdots + b^n = O(b^d), \tag{1}$$

- The maximum number of nodes in the 'Open-list' will be at the lowest level of the tree. Thus space-complexity is $O(b^d)$.

Analysis of BFS and DFS...

- **DFS:**
- For a *DFS* search, in the worst-case all the nodes are visited. Hence, time-complexity is same as for *BFS*, and equal to $O(b^d)$.
- Since, *DFS* needs to store only b nodes per level for a depth of d, the total nodes to be stored are $b \times d$, and space-complexity is $O(bd)$.

Depth-First Iterative Deepening (DFID) Search

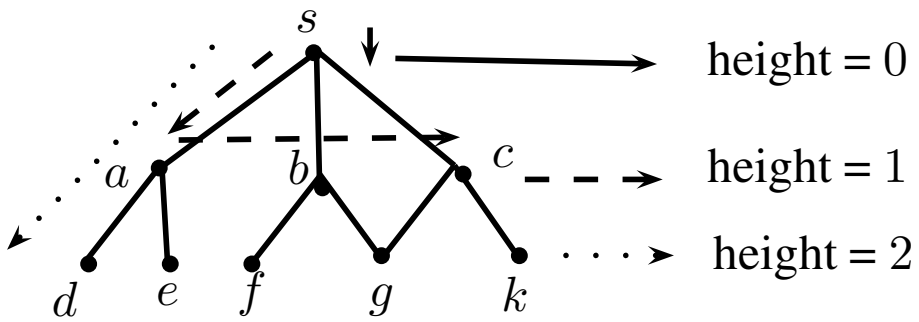


Figure 5: Search tree of Iterative deepening DFS.

Bidirectional Search

- If the search is carried out in both the directions, it can be speeded up.
- Consider that a bidirectional search is carried out with the depth as d and branching factor b.
- If each side progresses with same depth, the search required from each, for example, for iterative deepening *DFS*, has $O(b^{\frac{d}{2}})$ for time, and $O(b\frac{d}{2})$ for for space.
- When combined from both opposite sides, it becomes $2 \times O(b^{\frac{d}{2}}) = O(b^{\frac{d}{2}})$ for *time*, and $2 \times O(b \times \frac{d}{2}) = O(bd)$ for *space*.

Problem formulation for Search

Example

Farmer, goose, grains, fox problem.

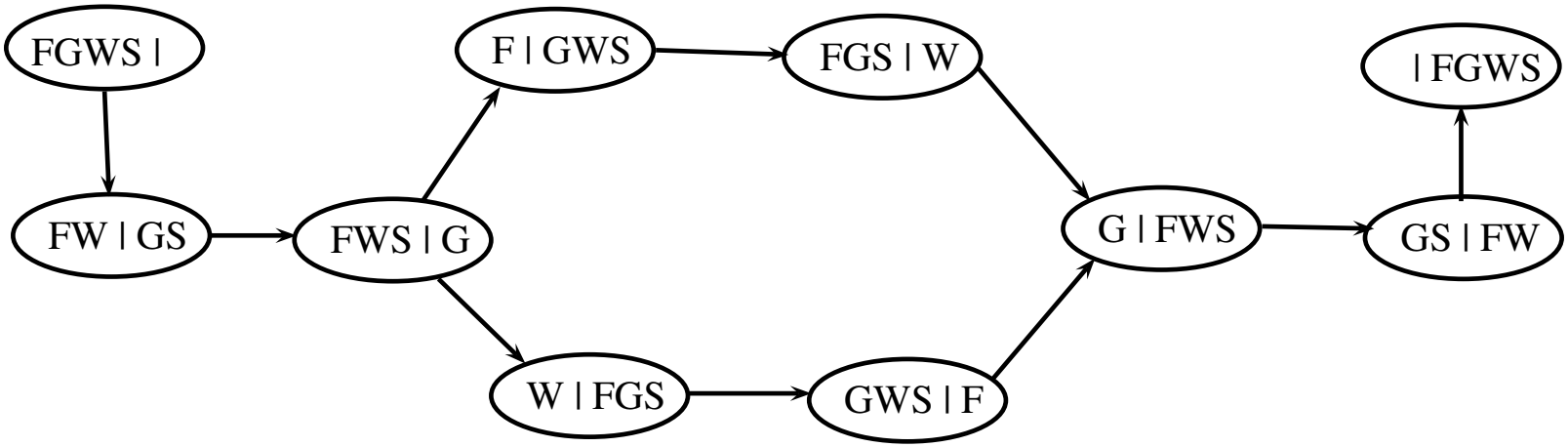


Figure 6: Farmer-goose-grains-fox Problem as Graph search.

How much you have understood?

1. The BFS uses what data structure?
2. The DFS uses what data structure?
3. Time complexity of DFS is ....
4. Time complexity of BFS is ....
5. Space complexity of DFS is ...
6. Space complexity is ....
7. Assumimg that there are 100 moves in a game of chess, and there are 20 possible alternatives per move, what is total possible number of chess games?
8. The answer of above concludes what?