State Space Representation and Search

Solving an AI Problem

- The problem is firstly represented as a state space.
- The state space is searched to find a solution to problem.
- Each state space takes the form of a tree or graph.
- Which search should be used?
  - Type of problem
  - How the problem can be represented

Components of a State Space

- Set of nodes representing each state of the problem.
- Arcs between nodes representing the legal moves from one state to another.
- An initial state.
- A goal state.

Search Techniques

- Uninformed
  - Depth First Search
  - Depth First Search with Iterative Deepening
  - Breadth First Search
- Informed/Heuristic
  - Best First Search
  - Hill Climbing
  - Branch and Bound Techniques
  - A* Algorithm
  - Simulated Annealing
  - Tabu Search

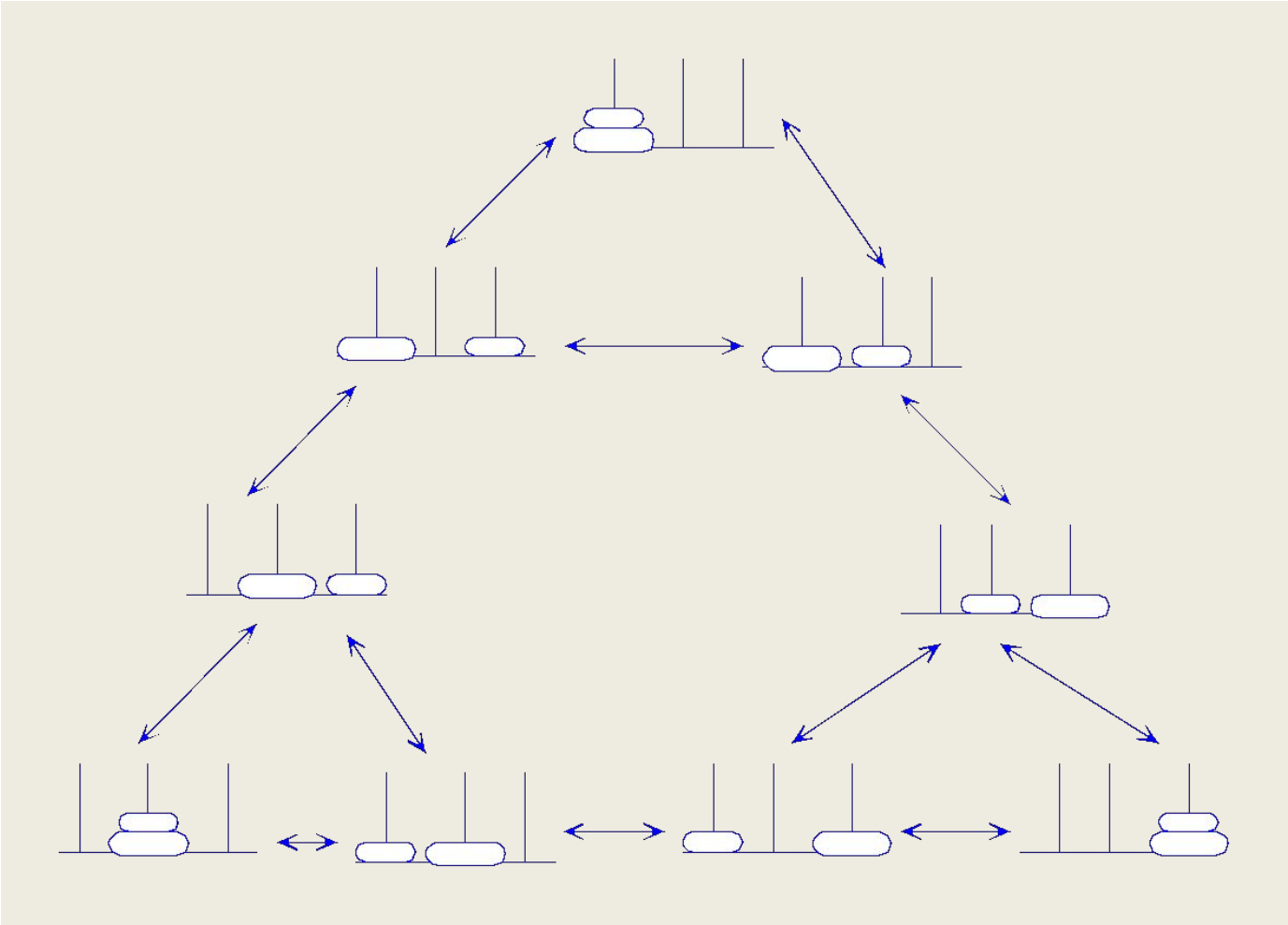
Classic AI Problems

- Traveling Salesman Problem
- Towers of Hanoi
- 8-Puzzle Problem

The Travelling Salesman Problem

- A salesman has a list of cities, each of which he must visit exactly once.
- There are direct roads between each pair of cities on the list.
- Find the route that the salesman should follow for the shortest trip that both starts and finishes at any one of the cities.

Example: Towers of Hanoi



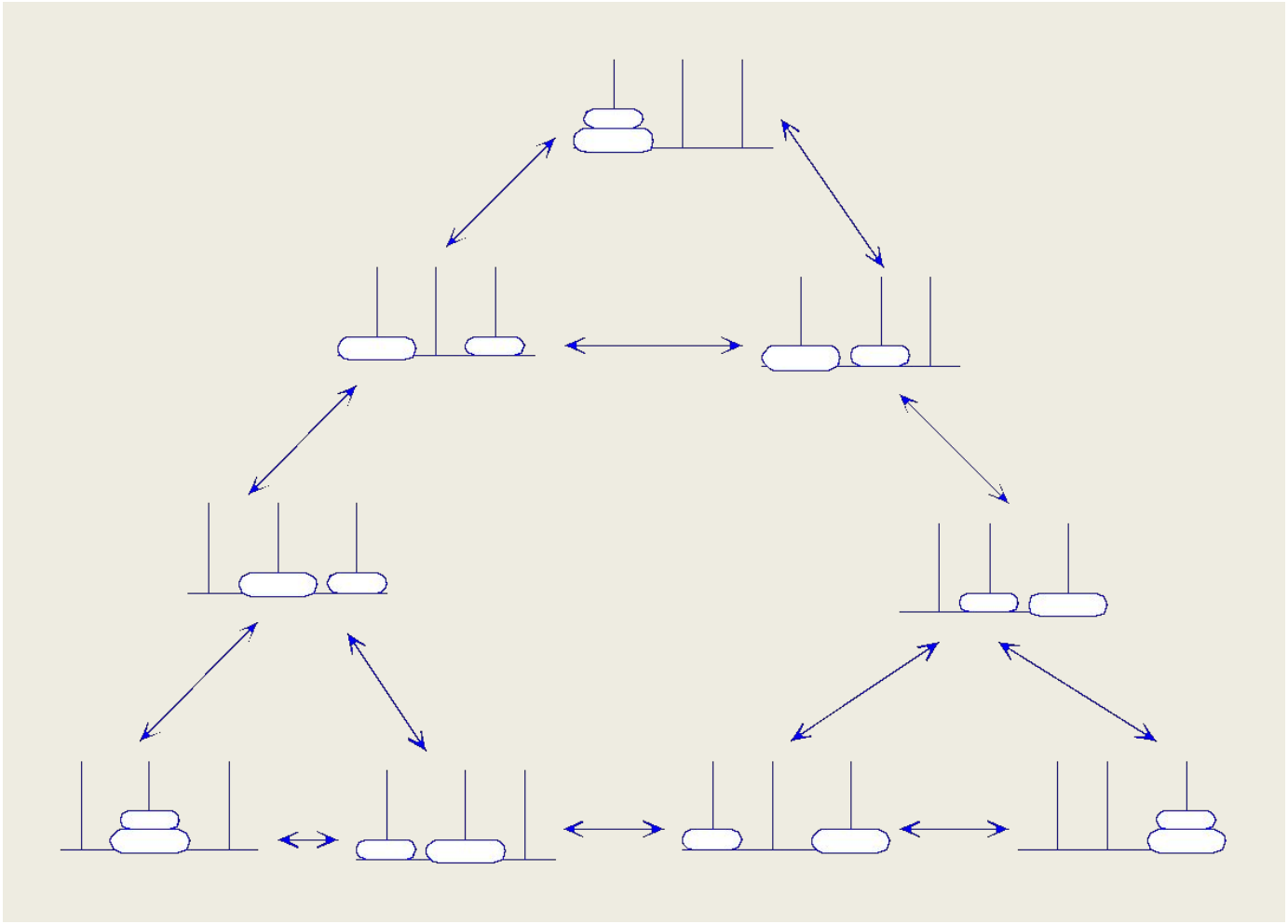
Towers of Hanoi

- In a monastery in the deepest Tibet there are three crystal columns and 64 golden rings.
- The rings are different sizes and rest over the columns.
- At the beginning of time all the rings rested on the leftmost column.
- All the rings must be moved to the last column without the smaller rings on top of larger rings.

Towers of Hanoi

- In moving the rings a larger ring must not be placed on a smaller ring.
- Furthermore, only one ring at a time can be moved from one column to the next.
- A simplified version of this problem which will consider involves only 2 or 3 rings instead of 64.

Example: Towers of Hanoi



8-Puzzle Example

Initial State

1	2	3
8		4
7	6	5

Goal State

1	8	3
2	6	4
7		5

8-Puzzle Problem

- The 8-Puzzle involves moving the tiles on the board above into a particular configuration.
- The blank square on the board represents a space.
- The player can move a tile into the space, freeing that position for another tile to be moved into and so on.

8-Puzzle Example

Initial State

1	2	3
8		4
7	6	5

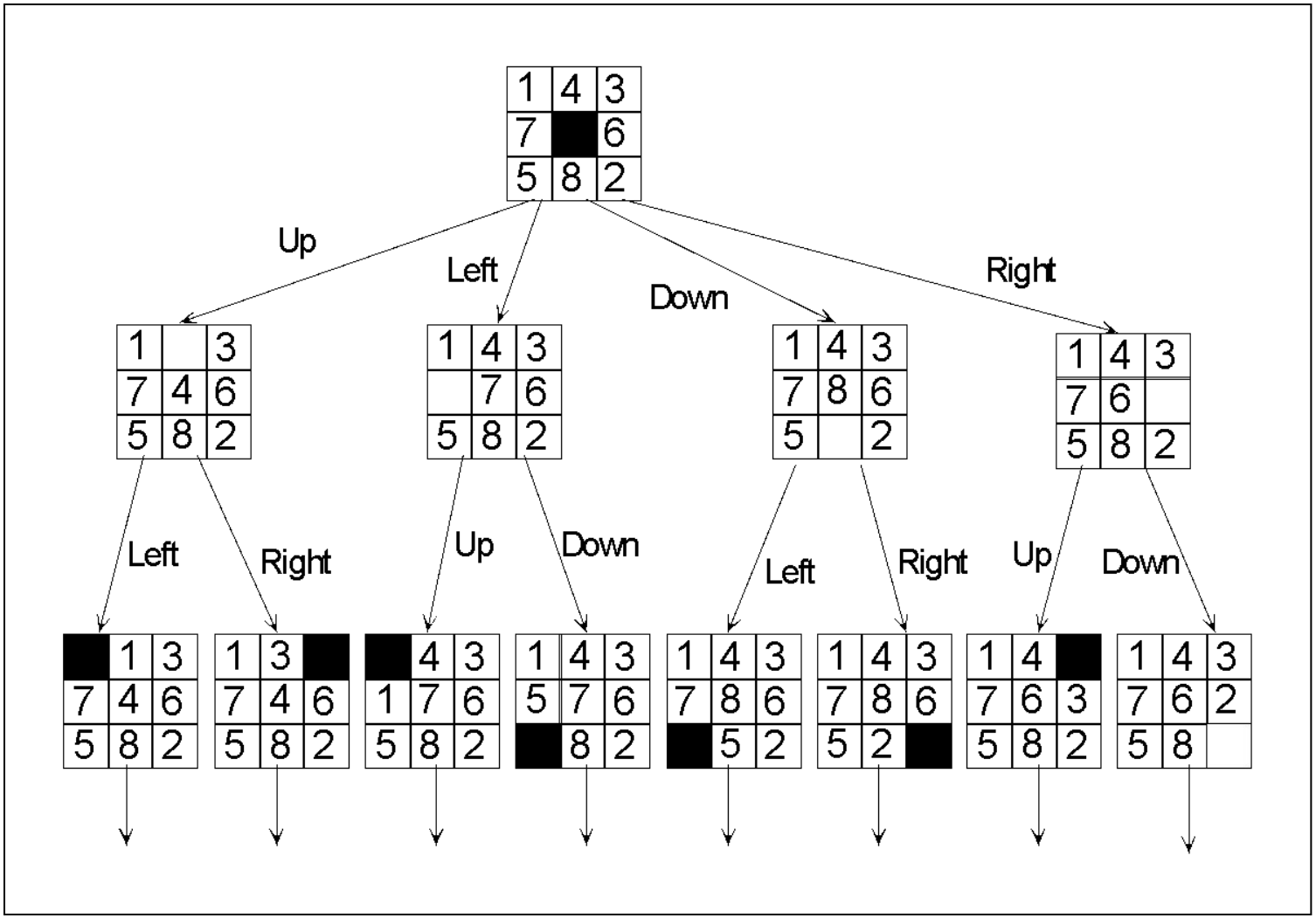
Goal State

1	8	3
2	6	4
7		5

Problem Representation

- What is the goal to be achieved?
- What are the legal moves or actions?
- What knowledge needs to be represented in the state description?
- Type of problem.
- Optimal solution vs. good enough solution

Example: 8-Puzzle Problem



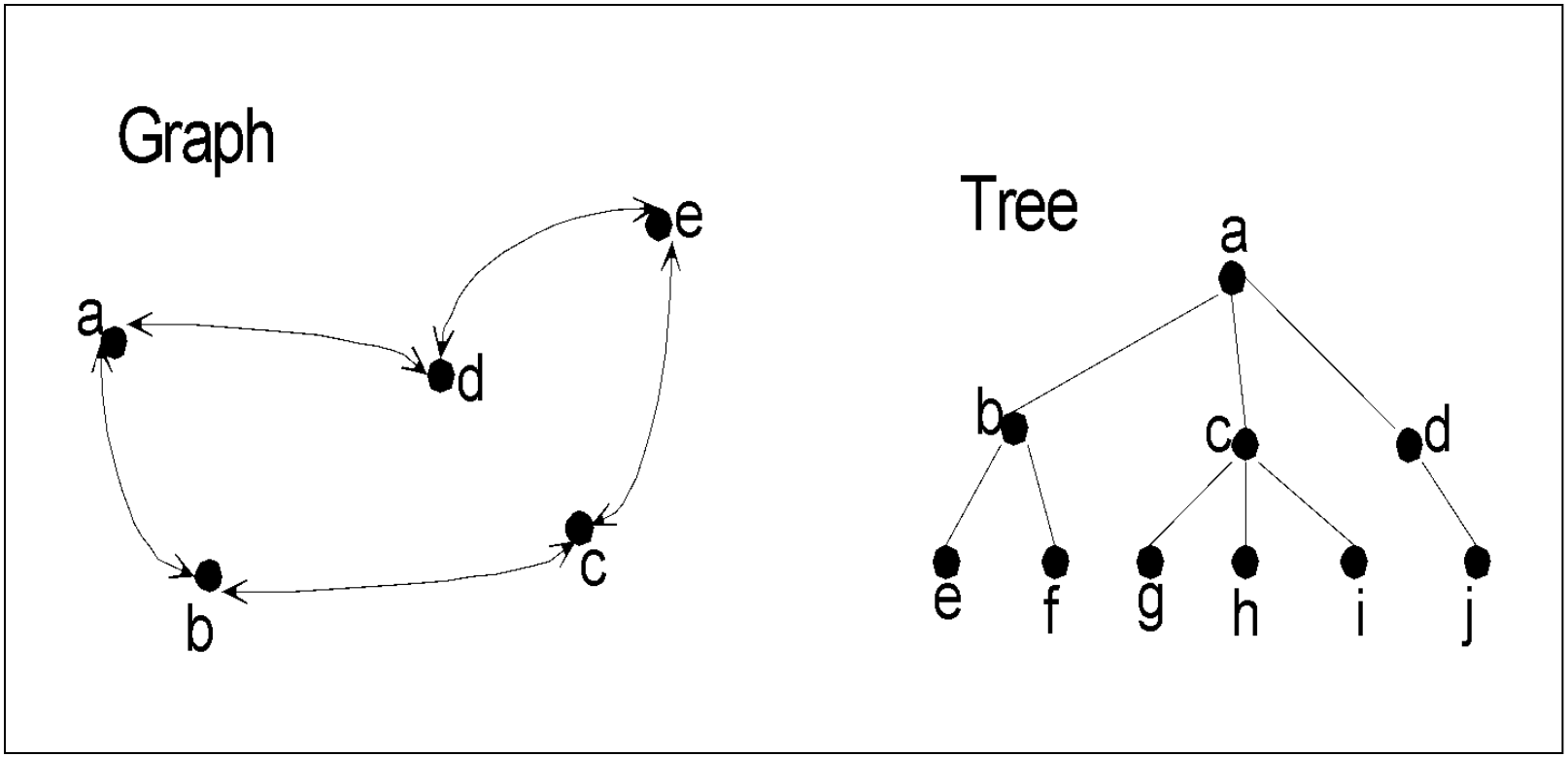
State Space Summary

- A state space is a set of descriptions or states.
- Composition of each problem:
  - One or more initial states
  - A set of legal moves
  - One or more goal states
- The number of operators are problem dependant and specific to a particular state space representation.
- The more operators the larger the branching factor of the state space.
- Why generate the state space at run-time, and not just have it built in advance?
- A search algorithm is applied to a state space representation to find a solution path.
- Each search algorithm applies a particular search strategy.

Graph vs. Tree

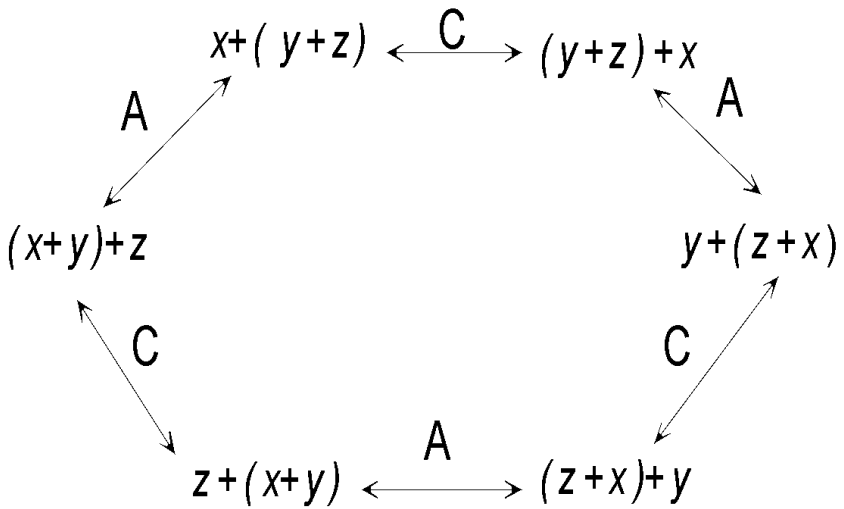
- If states in the solution space can be revisited more than once a directed graph is used to represent the solution space.
- In a graph more than one move sequence can be used to get from one state to another.
- Moves in a graph can be undone.
- In a graph there is more than one path to a goal whereas in a tree a path to a goal is more clearly distinguishable.
- A goal state may need to appear more than once in a tree. Search algorithms for graphs have to cater for possible loops and cycles in the graph.
- Trees may be more “efficient” for representing such problems as loops and cycles do not have to be catered for.
- The entire tree or graph will not be generated.

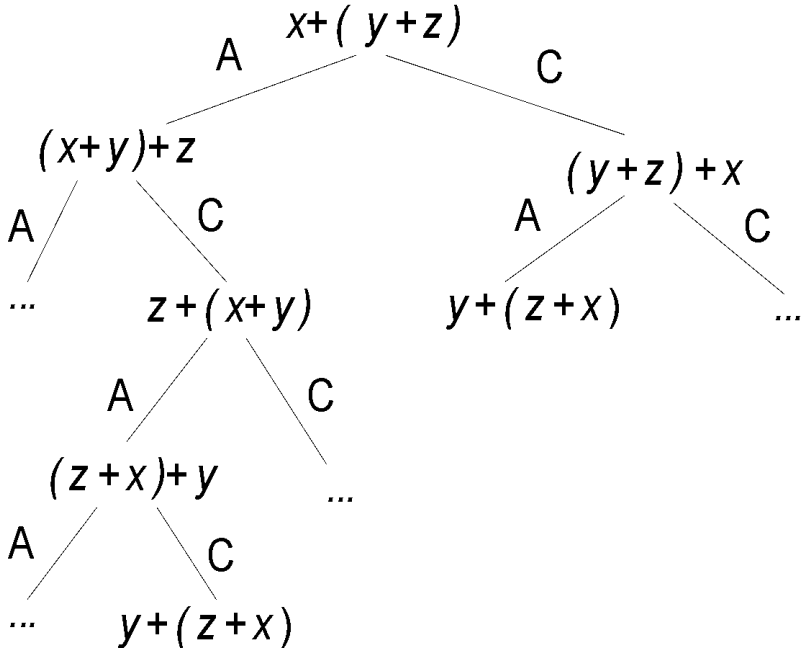
Example 1: Graph vs. Tree



Example 2: Graph vs. Tree

Prove x + (y +z) = y + (z+x) given
L+(M+N) = (L+M) + N .....(A)
M+N = N+M.....(C)



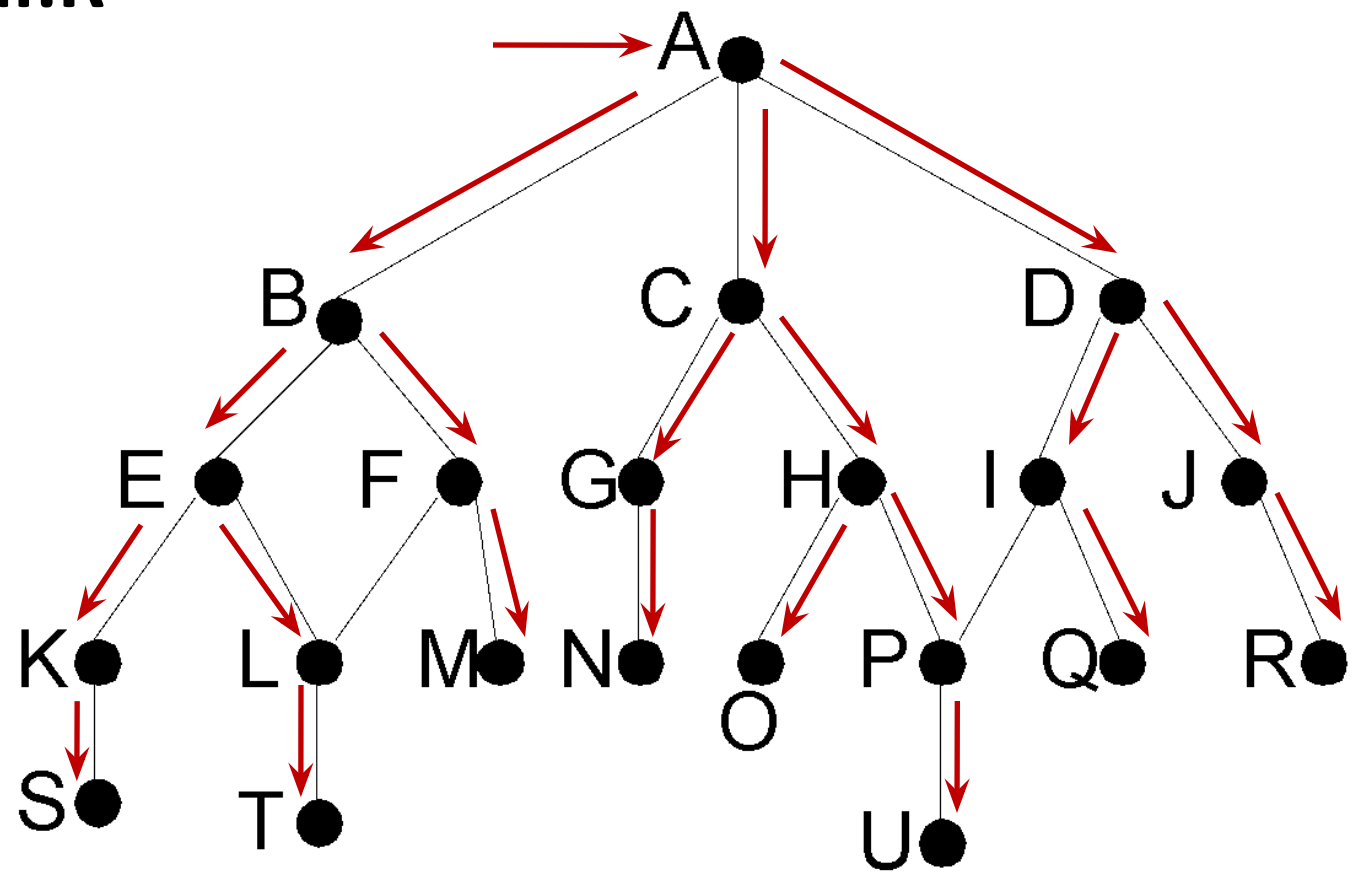


Type of Problem

- Some problems only need a representation, e.g. crossword puzzles.
- Other problems require a yes or no response indicating whether a solution can be found or not.
- The last type problem are those that require a solution path as an output, e.g. mathematical theorems, Towers of Hanoi. In these cases we know the goal state and we need to know how to attain this state.

Depth-First Search

Goal:R



A B E K S L T F M C G N H O P U D I Q J R

Exercise: DFS

A to B and C

B to D and E

C to F and G

D to I and J

I to K and L

Start state: A

Goal state: E , J

Exercise: DFS

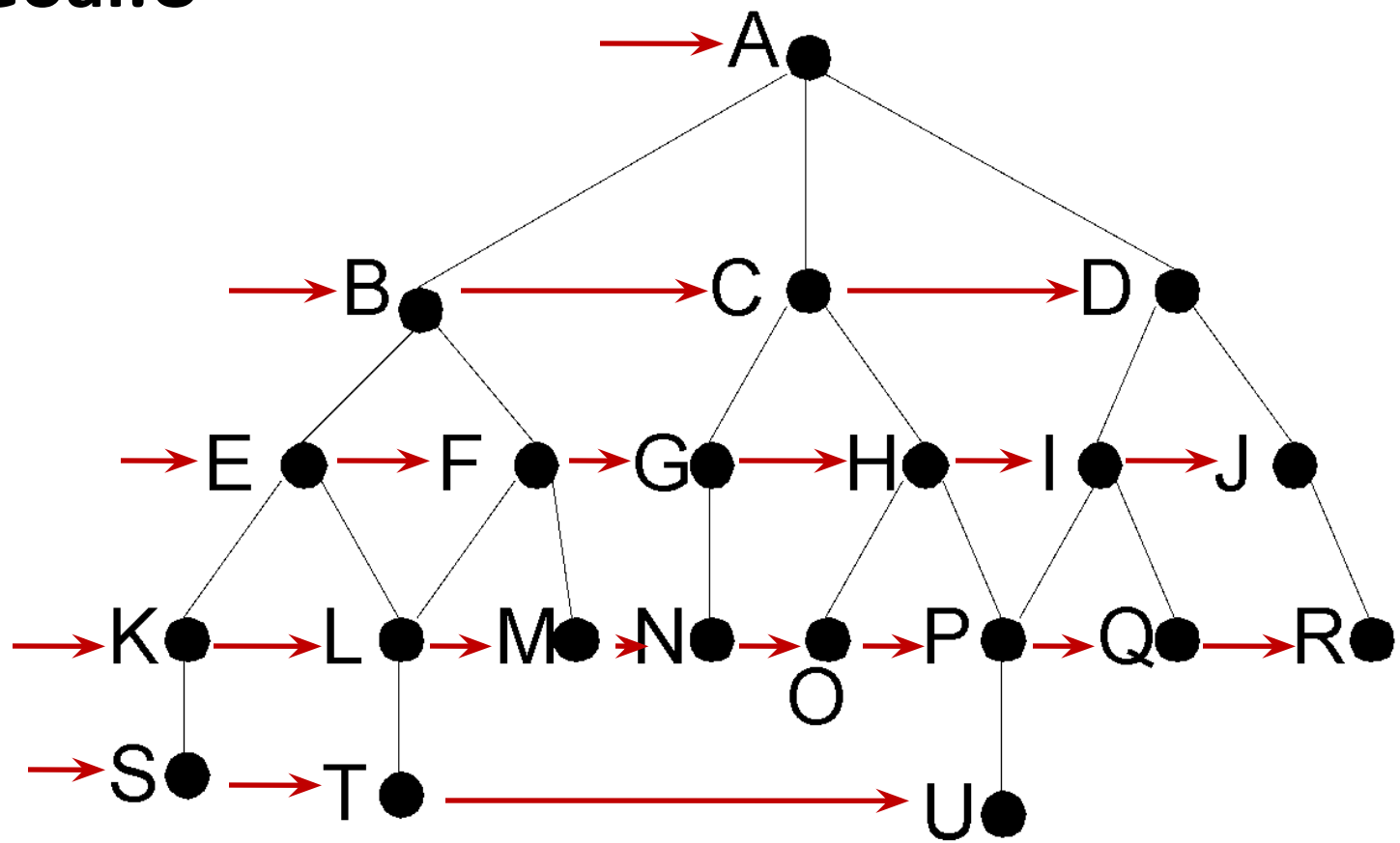
Start state: A Goal state: E , J

Let us now conduct a depth first search of the space. Remember that tree will not be generated before hand but is generated as part of the search. The search and hence generation of the tree will stop as soon as success is encountered or the open list is empty.

	OPEN	CLOSED	X	X's Children	State
1.	A	-			failure
2.			A	B, C	failure
3.	B, C	A			failure
4.	C	A	B	D, E	failure
5.	D, E, C	A, B			failure
6.	E, C	A, B	D	I, J	failure
7.	I, J, E, C	A, B, D			failure
8.	J, E, C	A, B, D	I	K, L	failure
9.	K, L, J, E, C	A, B, D, I			failure
10.	L, J, E, C	A, B, D, I,	K	none	failure
11.	J, E, C	A, B, D, I	L	none	failure
12.	E, C	A, B, D, I	J		success

Breadth-First Search

Goal:U



A B C D E F G H I J K L M N O P Q R S T U

Exercise: BFS

A to B and C

B to D and E

C to F and G

D to I and J

I to K and L

Start state: A

Goal state: E , J

DFS with Iterative Deepening

```
procedure DFID (intial_state, goal_states)
begin
    search_depth=1
    while (solution path is not found)
    begin
        dfs(initial_state, goals_states) with a
        depth bound of search_depth
        increment search_depth by 1
    endwhile
end
```

Difference Between Depth First and Breadth First

- Breadth first is guaranteed to find the shortest path from the start to the goal.
- DFS may get “lost” in search and hence a depth-bound may have to be imposed on a depth first search.
- BFS has a bad branching factor which can lead to combinatorial explosion.
- The solution path found by the DFS may be long and not optimal.
- DFS more efficient for search spaces with many branches, i.e. the branching factor is high.

Deciding on which Search to Use

- The importance of finding the shortest path to the goal.
- The branching of the state space.
- The available time and resources.
- The average length of paths to a goal node.
- All solutions or only the first solution.

QUESTIONS