MODULE - II

SEARCH STRATEGIES

Blind Search Strategies

20MCA188 ARTIFICIAL INTELLIGENCE (Elective-2)

Selma Joseph A.

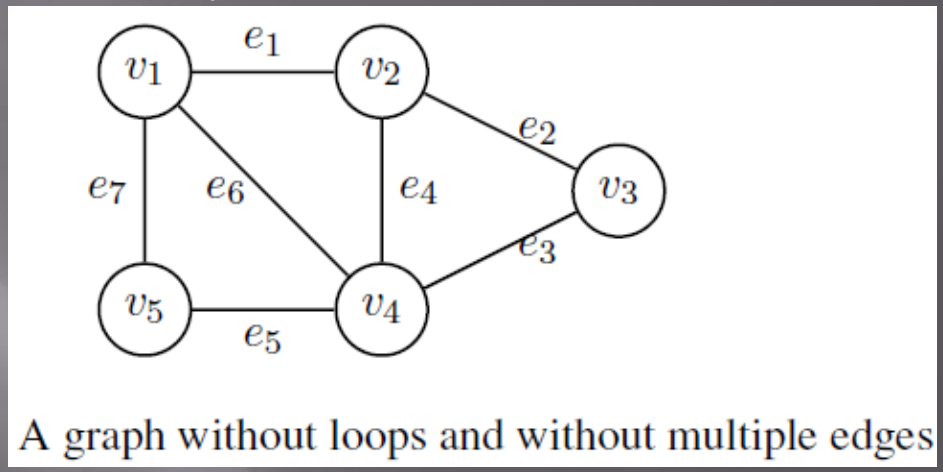
- Search algorithms are one of the most important areas of artificial intelligence.
- Combinatorial optimization, decision analysis, game playing, learning, planning, pattern recognition, robotics and theorem proving are some of the areas in which search algorithms play a key role.

Graphs

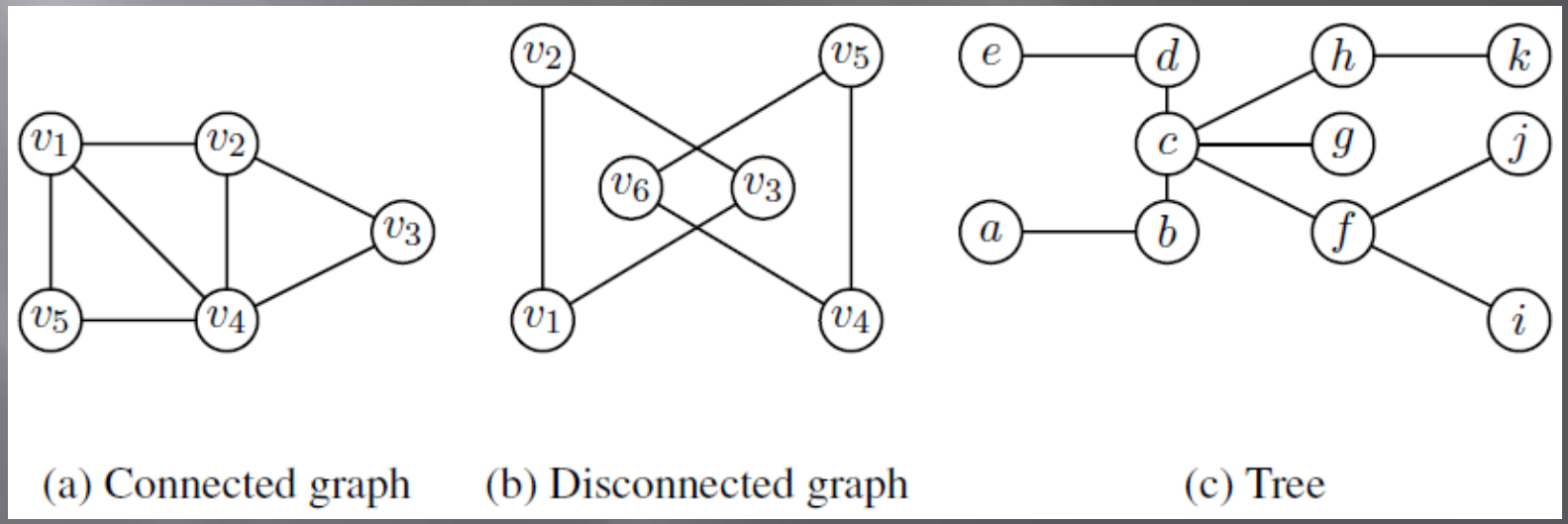
- A graph G is an ordered pair (V , E) where the elements of V are called vertices or nodes and the elements of E are called edges or sides. Each element of E can be thought of an arc joining two vertices in V.
- If the arcs may have specific directions in which case the graph is called a directed graph.
- If there are more than one edge joining two vertices, the graph is called a multigraph.
- There may also be loops in graphs, a loop being an edge joining a vertex with itself.

Trees

A graph is said to be connected if there is a path along the edges joining any two vertices in the graph. A path which returns to the starting vertex is called a cycle. A tree is a connected graph without cycles.



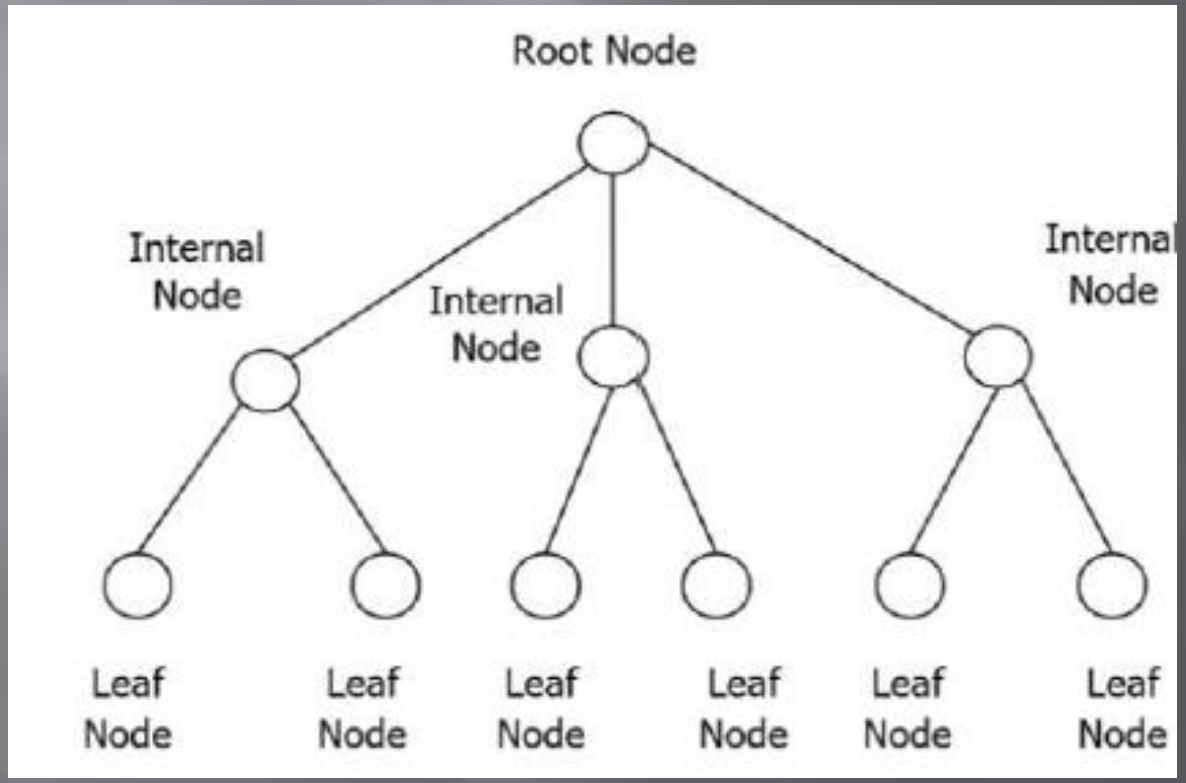
A graph without loops and without multiple edges



Different types of Graphs

Rooted trees

In search strategies, we generally consider rooted trees where one vertex is designated as the root and all edges are directed away from the root. The root vertex is positioned near the top of a page and the remaining vertices are positions below the root vertex at different levels.



A rooted tree

Search algorithm

- The process of looking for a sequence of actions that reaches the goal is called search.
- A search algorithm takes a problem as input and returns a solution in the form of an action sequence.
- Once a solution is found, the actions it recommends can be carried out. This is called the execution phase.

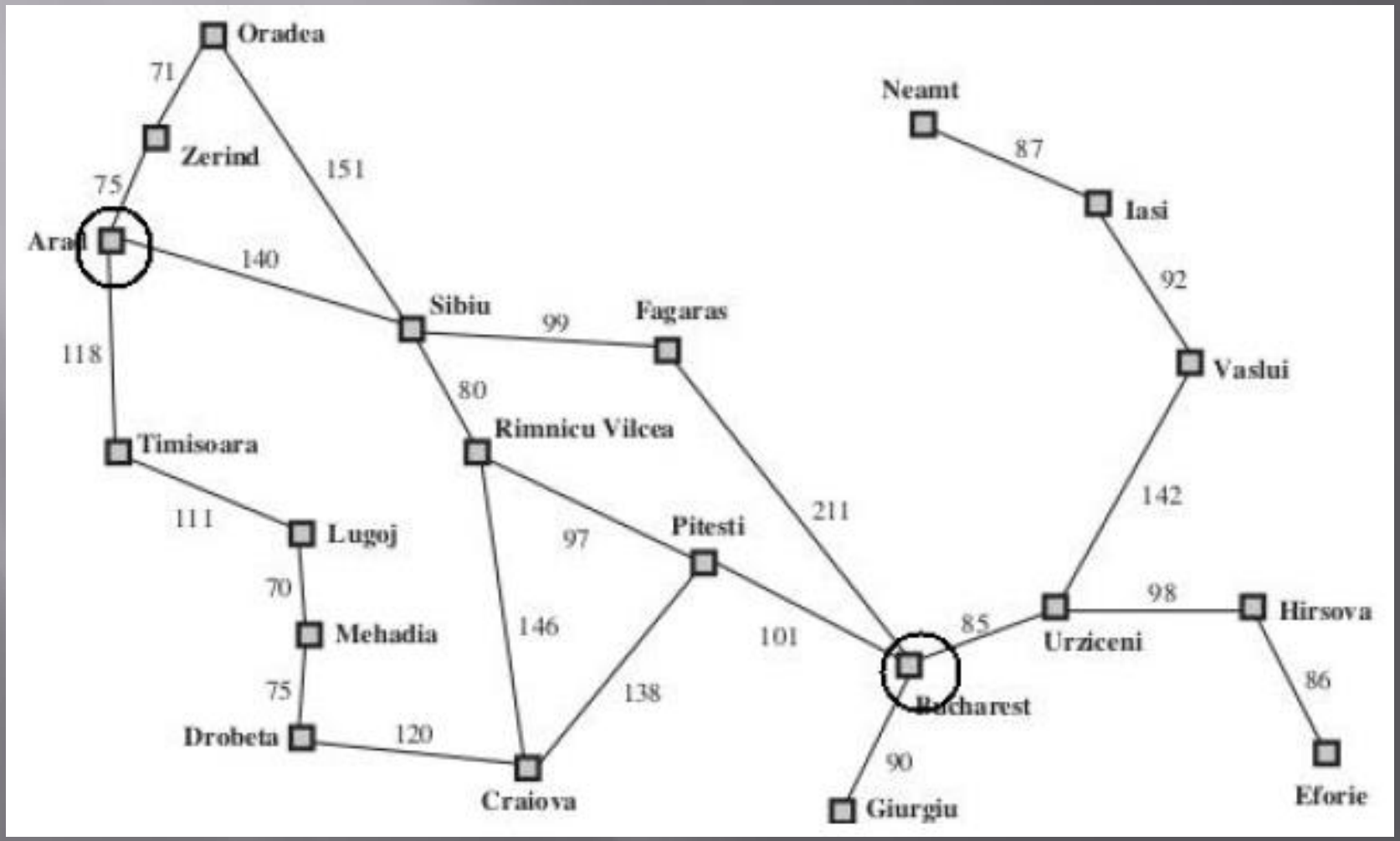
Search tree

- Having formulated some problems, we need to solve them. The possible action sequences starting at the initial state form a search tree with the initial state at the root; the branches are actions and the nodes correspond to states in the state space of the problem.
- The root node of the tree corresponds to the initial state.

Search tree

Example

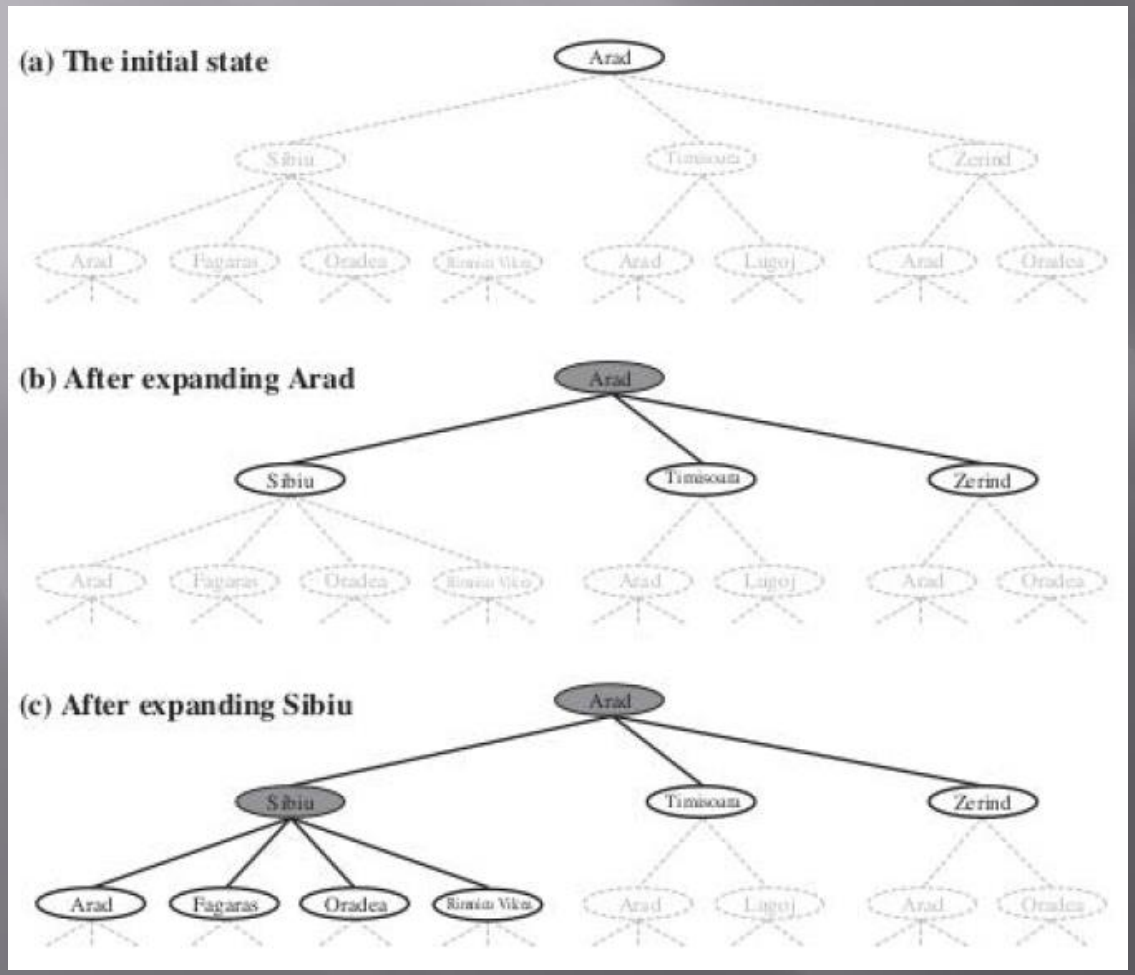
Consider a person intending to travel from Arad to Bucharest in Romania.



A road map of Romania

Search strategy

The first step is to test whether this is a goal state. Then we need to consider taking various actions. We do this by expanding the current state; that is, applying each legal action to the current state, thereby generating a new set of states. A search strategy specifies how to choose which state to expand next.



Blind search

- A blind search (also called an uninformed search) is a search that has no information about its domain. The only thing that a blind search can do is distinguish a non-goal state from a goal state. The blind search algorithms have no domain knowledge of the problem state. The only information available to blind search algorithms is the state, the successor function, the goal test and the path cost.
- Blind search is useful in situations where there may not be any information available to us. We might just be looking for an answer and won't know we have found it until we see it. It is also useful to study these blind searches as they form the basis for some of the intelligent searches.

- Strategies that know whether one non-goal state is "more promising" than another are called informed search or heuristic search strategies.

Let us assume that we are currently at Arad and we want to get to Bucharest. To construct a search tree we start by designating the initial state as the root node. In the present problem, the initial state is the city of Arad and so we designate Arad as the root node. There are three possible actions that we can take while we are at Arad, namely, go to Sibiu, go to Timisoara and go to Zerind. These actions define three branches of the search tree emanating from the root node and ending at the nodes named as Sibiu, Timisoara and Zerind. These three branches form level 1 of the search tree. A blind search will have no preference as to which node it should explore first.

Suppose we arbitrarily move to Sibiu. Then the possible actions are: go to Arad, go to Rimnicu Vicea, go to Fagaras and go to Oradea. These actions produce four branches from the node.

The process is continued.

Remarks

The search tree shown in (c) includes the path from Arad to Sibiu and back to Arad again! We say that Arad is a repeated state in the search tree, generated in this case by a loopy path.

Breadth-first search

- Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

Breadth-first search

Algorithm

The algorithm returns the goal state or failure.

1. Create a variable called NODE-LIST and set it to the initial state.
2. Until a goal state is found or NODE-LIST is empty do:

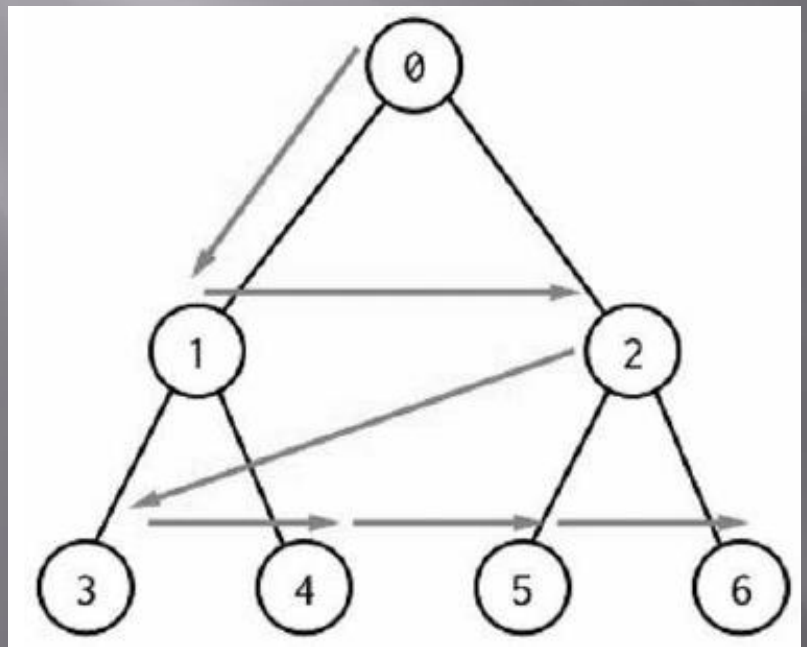
(a) Remove the first element from NODE-LIST and call it E. If NODE-LIST was empty, quit and return failure.

(b) For each way that each rule can match the state described in E do:

i. Apply the rule to generate a new state.

ii. If the new state is a goal state, quit and return this state.

iii. Otherwise, add the new state to the end of NODE-LIST.



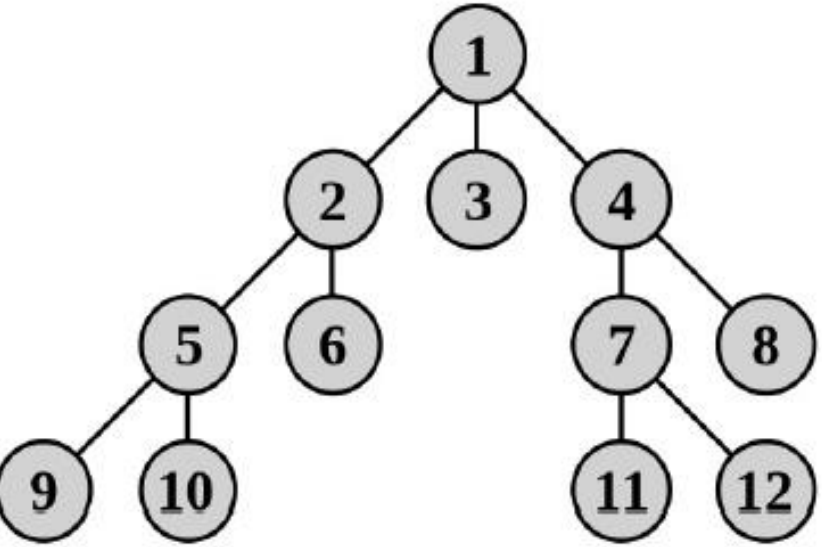
The order in which the nodes are explored while applying the BFS algorithm:

0 ➡ 1 ➡ 2 ➡ 3 ➡ 4 ➡ 5 ➡ 6

Examples

Example 1

Let the search tree be as in Figure the initial and goal states being 1 and 8 respectively. Apply the breadth-first search algorithm to find the goal state.



ble called NODE-LIST and set it to the initial state.
ate is found or NODE-LIST is empty do:
irst element from NODE-LIST and call it E. If NODE-
pty, quit and return failure.
that each rule can match the state described in E do:
the rule to generate a new state.
new state is a goal state, quit and return this state.
rwise, add the new state to the end of NODE-LIST.

Step no.	*NODE-LIST*	Node E	Node generated from E	Comment
1	1	-	-	Initial state
2	-	1	-	-
3	-	1	2	-
4	2	1	-	-
5	2	1	3	-
6	2, 3	1	-	-
7	2, 3	1	4	-
8	2, 3, 4	1	-	-
9	3, 4,	2	-	-
10	3, 4	2	5	-
11	3, 4, 5	2	-	-
12	3, 4, 5	2	6	-
13	3, 4, 5, 6	2	-	-
13	4, 5, 6	3	-	No node generated from E
14	5, 6	4	-	-
15	5, 6	4	7	-
16	5, 6, 7	4	-	-
17	5, 6, 7	4	8	Goal state. Return 8 and quit.

Examples

Example 2

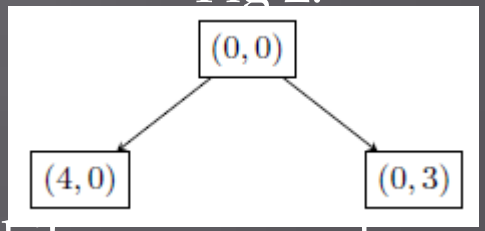
Consider the water jug problem and the associated production rules. We now construct the search tree using the breadth-first search algorithm.

Stage 1. Construct a tree with the initial state (0, 0) as its root.

Fig 1: (0,0)

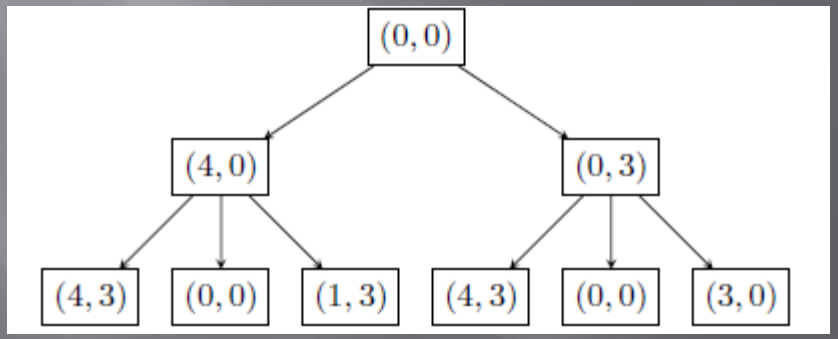
Stage 2. Construct all the children of the root by applying each of the applicable rules to the initial state. Then we get a tree.

Fig 2:



Stage 3. Now for each leaf node in Figure 3.9, generate all its successors by applying all the rules that are appropriate. The tree at this point is

Fig 3:

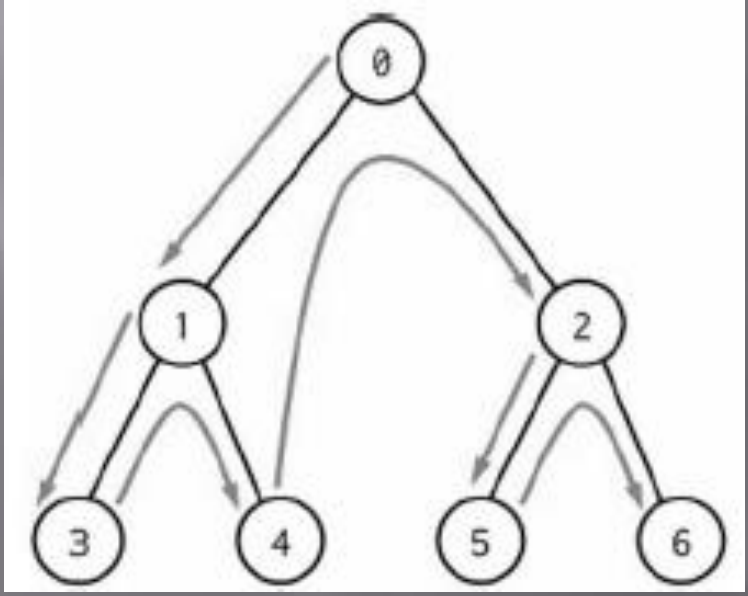


Stage 4. The process is continued until we get a node representing the goal state, namely, (2, 0).

Depth-first search

Algorithm

1. If the initial state is a goal state, quit and return success.
2. Otherwise, do the following until success or failure is signaled:
   (a) Generate a successor E of the initial state. If there are no more successors, signal failure.
   (b) Call depth-first search with E as the initial state.
   (c) If success is returned, signal success. Otherwise continue in this loop.

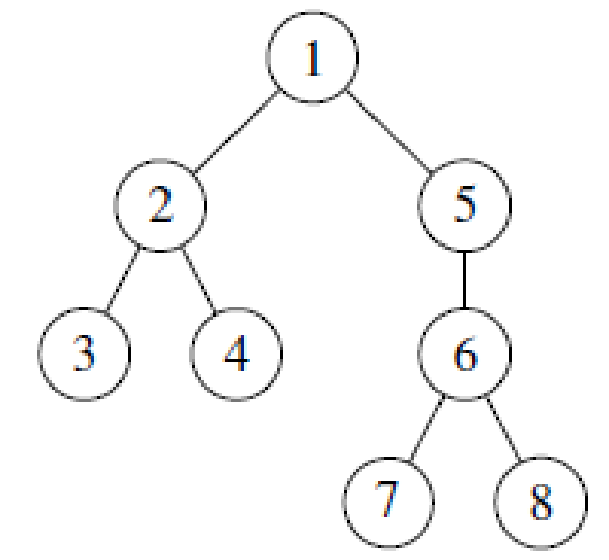


The order in which the nodes are explored while applying the DFS algorithm:

0 ➡ 1 ➡ 3 ➡ 4 ➡ 2 ➡ 5 ➡ 6

Example 1

Let the search tree be as in Figure, the initial and goal states being 1 and 7 respectively. Apply the depth-first search algorithm to find the goal state.



DFS(x) an invocation of the depth-first search algorithm with x as the initial vertex. The various nodes are visited in the following order:

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7$$

Examples

Example 2

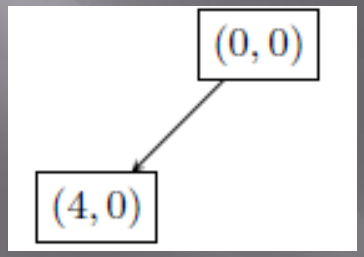
Consider the water jug problem and the associated production rules. We now construct the search tree using the depth-first search algorithm.

Stage 1. Construct a tree with the initial state (0, 0) as its root.

Fig 1: (0,0)

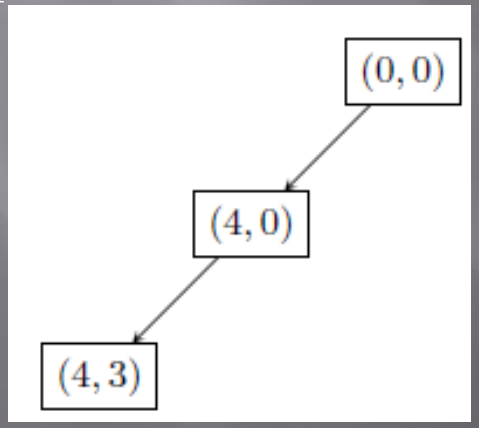
Stage 2. We generate a successor E for (0, 0). We choose the successor obtained by the application of Rule 1 in the production rules. Then we get a tree as in Fig 2.

Fig 2:



Stage 3. Now for the leaf node in Figure 2, generate a successor by applying an applicable rule. The tree at this point is Fig 3.

Fig 3:



Stage 4. The process is continued until we get a node representing the goal state, namely, (2, 0).

Breadth-first search vs. depth-first search

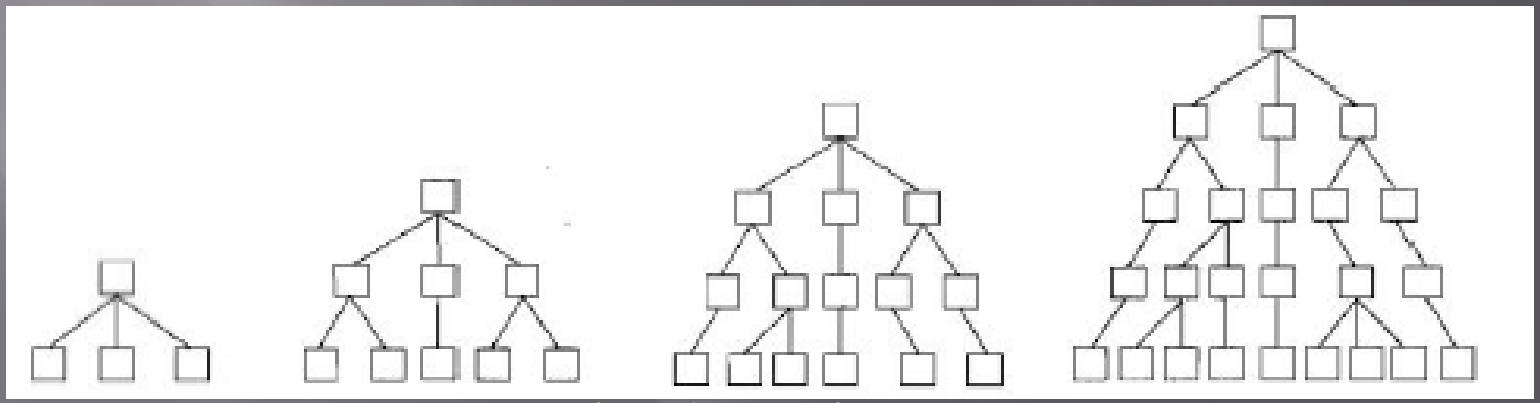
Sl No	Breadth-first	Depth-first
1	Requires more memory because all of the tree that has so far been generated must be stored.	Requires less memory because only the nodes in the current path are stored.
2	All parts of the search tree must be examined at level n before any node on level $n+1$ can be examined.	May find a solution without examining much of the search space. It stops when one solution is found.
3	Will not get trapped exploring a blind alley.	May follow an unfruitful path for a long time, perhaps for ever.
4	If there is a solution, Guaranteed to find a solution if there exists one. If there are multiple solutions then a minimal solution (that is, a solution that takes a minimum number of steps) will be found.	May find a long path to a solution in one part of the tree when a shorter path exists in some other unexplored path of the tree.
5	Breadth-first search uses queue data structure.	Depth-first search uses stack data structure.
6	More suitable for searching vertices which are closer to the given source.	More suitable when there are solutions away from source.

Depth-first iterative deepening (DFID) search

Iterative deepening combines the benefits of depth-first and breadth-first searches. Like breadth-first search, the iterative deepening search is guaranteed to find a solution if one exists. In general, iterative deepening is the preferred uninformed search method when the search space is large and the depth of the solution is not known.

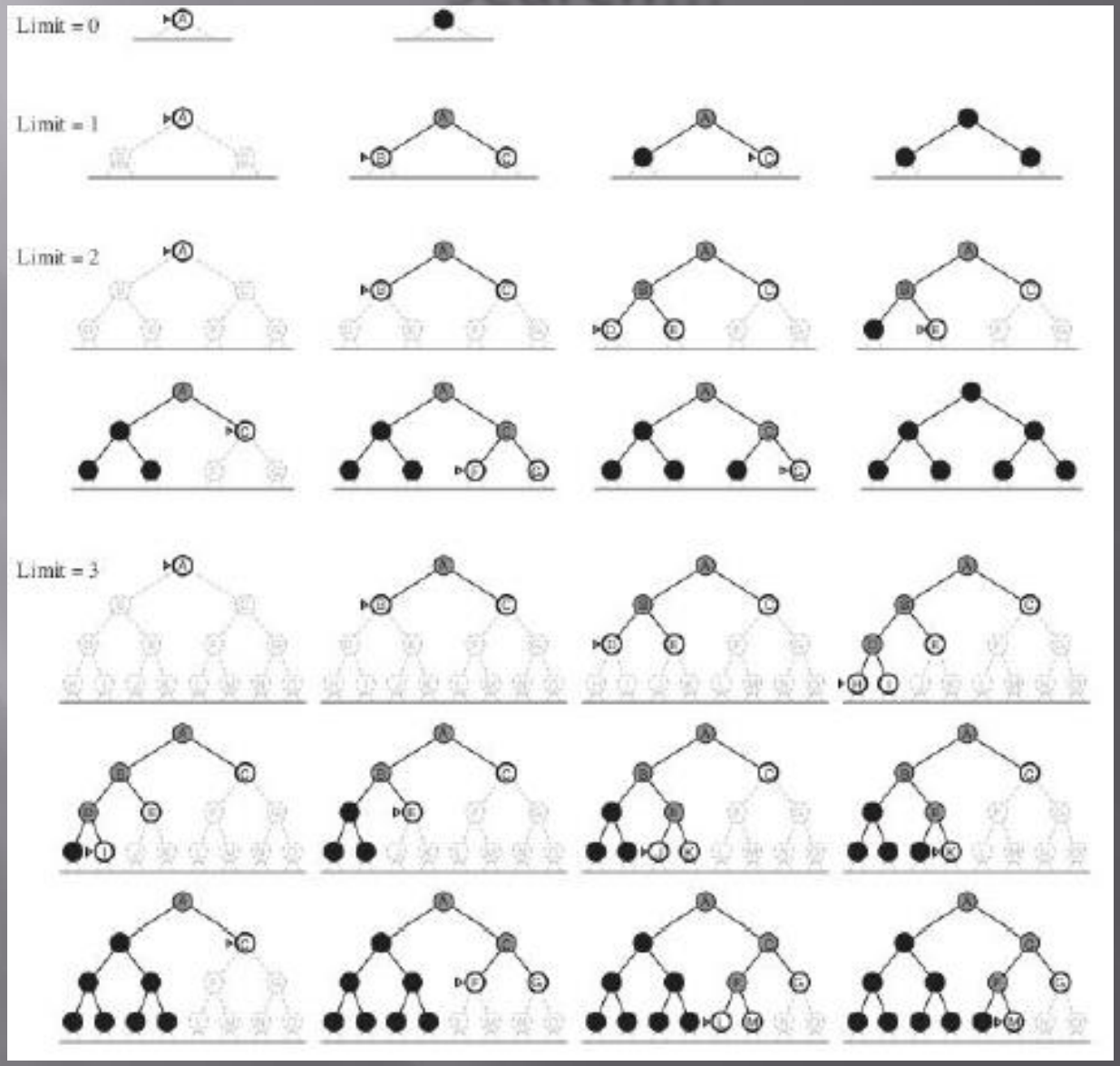
Algorithm

1. Set SEARCH-DEPTH = 1.
2. Conduct a depth-first search to a depth of SEARCH-DEPTH. If a solution is found, then return it.
3. Otherwise, increment SEARCH-DEPTH by 1 and go to step 2.



Iterative deepening

Depth-first iterative deepening (DFID) search…



The order in which the nodes are explored in iterative deepening

Depth-first iterative deepening (DFID) search…

Advantages and disadvantages

1. The DFID will always find the shortest solution path to the goal state if it exists.
2. The maximum amount of memory used is proportional to the number of nodes in the solution path.
3. All iterations but the final one are wasted.
4. The DFID is slower than the depth-first search only by a constant factor.
5. DFID is the optimal algorithm in terms of space and time for uninformed search.