Artificial Intelligence

Unit-2 (State Space Search)

01CE0702

Department of Computer Engineering

Shilpa Singhal

Topics to be Covered

- Defining The Problems As A State Space Search
- Production Systems, Production Characteristics, Production System Characteristics and Issues in the Design of Search Programs

Problem solving

1.**Problem solving** is a process of generating solutions from observed data.

- a 'problem' is characterized by a set of goals,
- a set of objects, and
- a set of operations.

These could be ill-defined and may evolve during problem solving.

2.A **'problem space'** is an abstract space.

- ✓ A problem space encompasses all valid states that can be generated by the application of any combination of operators on any combination of objects.
- ✓ The problem space may contain one or more solutions. A solution is a combination of operations and objects that achieve the goals.

3.A **'search'** refers to the search for a solution in a problem space.

- ✓ Search proceeds with different types of 'search control strategies'.
- ✓ The depth-first search and breadth-first search are the two common search strategies.

Defining The Problems As A State Space Search

A **State Space** represents a problem in terms of states and operators that change states. A state space consists of:

- A representation of the states the system can be in. For example, in a board game, the board represents the current state of the game.
- A set of operators that can change one state into another state. In a board game, the operators are the legal moves from any given state. Often the operators are represented as programs that change a state representation to represent the new state.
- An initial state.
- A set of final states; some of these may be desirable, others undesirable. This set is often represented implicitly by a program that detects terminal states.

State Space Search: Water Jug Problem

“You are given two jugs, a 4-litre one and a 3-litre one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 litres of water into 4-litre jug.”

State Space Search: Water Jug Problem

- State: (x, y)

  x = 0, 1, 2, 3, or 4   y = 0, 1, 2, 3

  x →Quantity of water in 4 gallon jug

  y→ Quantity of water in 3 gallon jug
- Start state: (0, 0).
- Goal state: (2, n) for any n.
- Attempting to end up in a goal state.

Production Rules for Water Jug Problem

Here, let ***x*** denote the 4-gallon jug and ***y*** denote the 3-gallon jug.

S.No.	Initial State	Condition	Final state	Description of action taken
1.	(x,y)	If x<4	(4,y)	Fill the 4 gallon jug completely
2.	(x,y)	if y<3	(x,3)	Fill the 3 gallon jug completely
3.	(x,y)	If x>0	(x-d,y)	Pour some part from the 4 gallon jug
4.	(x,y)	If y>0	(x,y-d)	Pour some part from the 3 gallon jug
5.	(x,y)	If x>0	(0,y)	Empty the 4 gallon jug
6.	(x,y)	If y>0	(x,0)	Empty the 3 gallon jug
7.	(x,y)	If (x+y)<7	(4, y-[4-x])	Pour some water from the 3 gallon jug to fill the four gallon jug
8.	(x,y)	If (x+y)<7	(x-[3-y],y)	Pour some water from the 4 gallon jug to fill the 3 gallon jug.
9.	(x,y)	If (x+y)<4	(x+y,0)	Pour all water from 3 gallon jug to the 4 gallon jug
10.	(x,y)	if (x+y)<3	(0, x+y)	Pour all water from the 4 gallon jug to the 3 gallon jug

Rules to define a Legal moves

1. $(x, y) \rightarrow (4, y)$ if $x < 4$
2. $(x, y) \rightarrow (x, 3)$ if $y < 3$
3. $(x, y) \rightarrow (x - d, y)$ if $x > 0$
4. $(x, y) \rightarrow (x, y - d)$ if $y > 0$

5. $(x, y) \rightarrow (0, y)$
   if $x > 0$
6. $(x, y) \rightarrow (x, 0)$
   if $y > 0$
7. $(x, y) \rightarrow (4, y - (4 - x))$
   if $x + y \geq 4, y > 0$
8. $(x, y) \rightarrow (x - (3 - y), 3)$
   if $x + y \geq 3, x > 0$

9. $(x, y) \rightarrow (x + y, 0)$
   if $x + y \leq 4, y > 0$
10. $(x, y) \rightarrow (0, x + y)$
   if $x + y \leq 3, x > 0$
11. $(0, 2) \rightarrow (2, 0)$

12. $(2, y) \rightarrow (0, y)$

One Solution

1. Current state = (0, 0)
2. Loop until reaching the goal state (2, 0)
   - Apply a rule whose left side matches the current state
   - Set the new current state to be the resulting state

(0, 0)	
(0, 3)	Rule-2
(3, 0)	Rule-9
(3, 3)	Rule-2
(4, 2)	Rule-7
(0, 2)	Rule-5 or Rule-12
(2, 0)	Rule-9 or Rule-11

Special-purpose rules

Special-purpose rules to capture special-case knowledge that can be used at some stage in solving a Problem.

11. (0, 2) $\rightarrow$ (2, 0)

12. (2, y) $\rightarrow$ (0, y)

State Space Search: Summary

1. Define a state space that contains all the possible configurations of the relevant objects.
2. Specify the initial states.
3. Specify the goal states.
4. Specify a set of rules:
   - What are unstated assumptions?
   - How general should the rules be?
   - How much knowledge for solutions should be in the rules?

Production System

Production systems provide appropriate structures for performing and describing the search processes. A production system has four basic components as enumerated below.

- A **set of rules** each consisting of a left side that determines the applicability of the rule and a right side that describes the operation to be performed if the rule is applied.
- A **Database or Knowledge Base** of current facts established during the process of inference.
- A **Control Strategy** that specifies the order in which the rules will be compared with facts in the database and also specifies how to resolve conflicts in selection of several rules or selection of more facts.
- A **Rule** firing module

Characteristics of Production Systems

- A **Monotonic Production System** is a production system in which the application of a rule never prevents the later application of another rule that could also have been applied at the time the first rule was selected.
- A **Non-Monotonic Production System** is one in which this is not true.
- A **Partially Communicative Production System** is a production system with the property that if the application of a particular sequence of rules transforms state P into state Q, then any combination of those rules that is allowable also transforms state P into state Q.
- A **Commutative Production System** is a production system that is both monotonic and partially commutative.

Features of Production System

- **1. Simplicity**

• The structure of each sentence in a production system is unique and uniform as they use "IF-THEN" structure.

• This structure provides simplicity in knowledge representation.

- **2. Modularity**

• This means production rule code the knowledge available in discrete pieces. Information can be treated as a collection of independent facts which may be added or deleted from the system with essentially no delete side effects.

- **3. Modifiability**

• This means the facility of modifying rules.

• It allows the development of production rules in a skeletal form first and then it is accurate to suit a specific application.

- **4. Knowledge intensive**

• knowledge base of production system stores pure knowledge.

• Each production rule is normally written as an English sentence; the problem of semantics is solved by the very structure of the representation.

Advantages of Production System

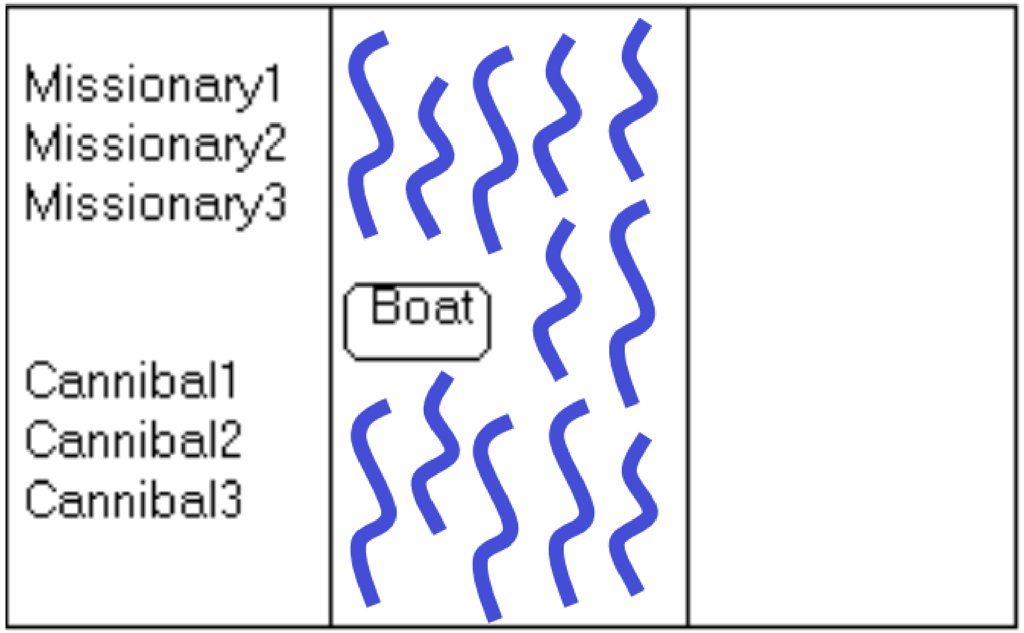
- Provides **excellent tools** for structuring AI programs .
- The system is highly **modular** because individual rules can be added, removed or modified independently.
- Separation of **knowledge** and **Control-Recognises Act Cycle**
- A natural **mapping** onto state-space research data or goal-driven.
- The system uses pattern directed control which is more **flexible** than algorithmic control.
- Provides opportunities for **heuristic control** of the search.
- A good way to model the **state-driven nature** of intelligent machines.
- Quite helpful in **a real-time** environment and applications.

Disadvantages of Production System

- It is very **difficult** to analyze the flow of control within a production system.
- It describes the operations that can be performed in a search for a solution to the problem.
- There is an **absence of learning** due to a rule-based production system that does not store the result of the problem for future use.
- The rules in the production system should not have any type of **conflict resolution** as when a new rule is added to the database it should ensure that it does not have any conflict with any existing rule.

Missionaries and Cannibals Problem

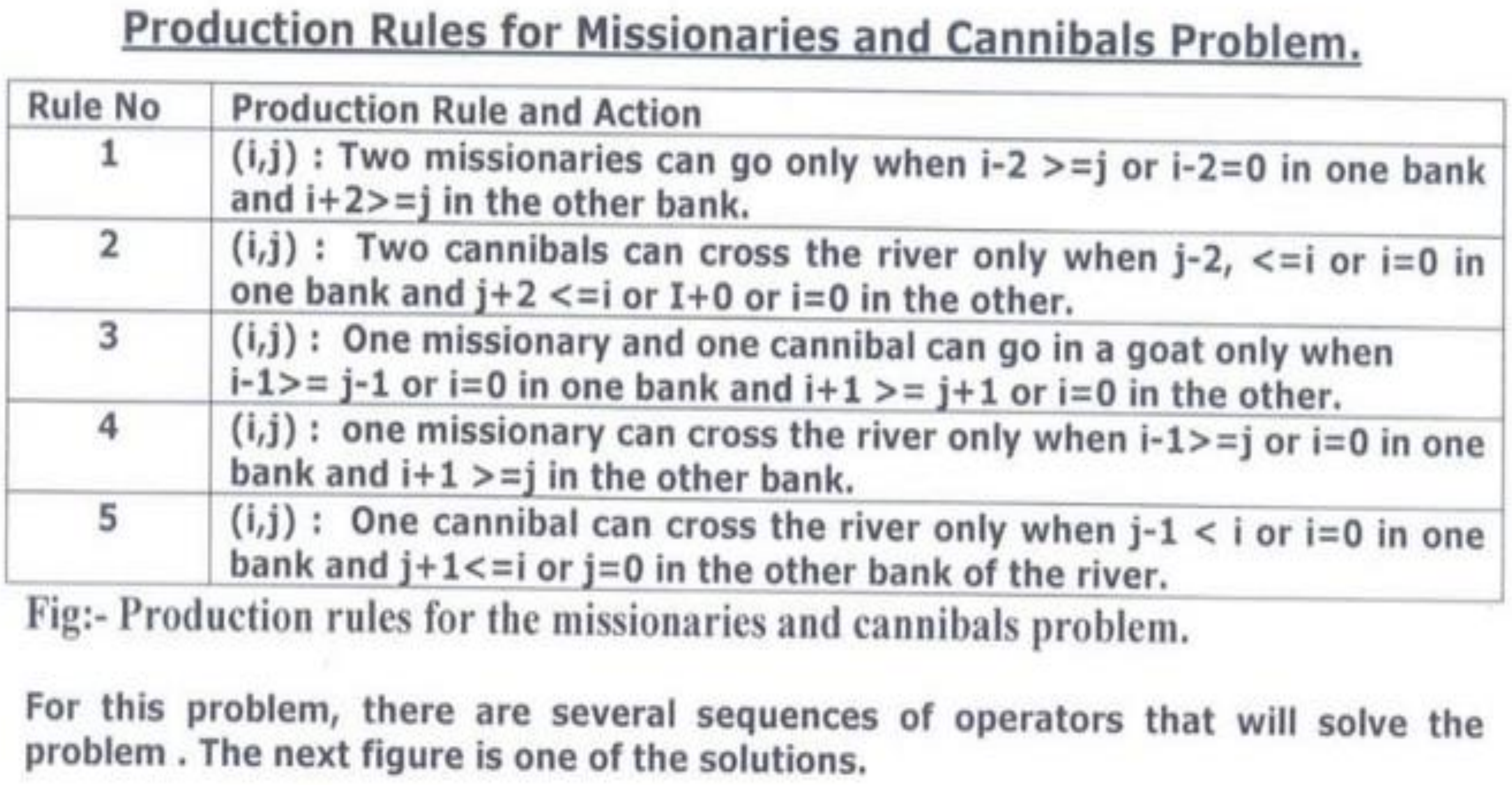
Three missionaries and three cannibals wish to cross the river. They have a small boat that will carry up to two people. Everyone can navigate the boat. If at any time the Cannibals outnumber the Missionaries on either bank of the river, they will eat the Missionaries. Find the smallest number of crossings that will allow everyone to cross the river safely.



Missionaries and Cannibals

Goal: Move all the missionaries and cannibals across the river.
Constraint: Missionaries can never be outnumbered by cannibals on either side of river, or else the missionaries are killed.
State: configuration of missionaries and cannibals and boat on each side of river.
Initial State: 3 missionaries, 3 cannibals and the boat are on the near bank
Operators: Move boat containing some set of occupants across the river (in either direction) to the other side.

Missionaries and Cannibals



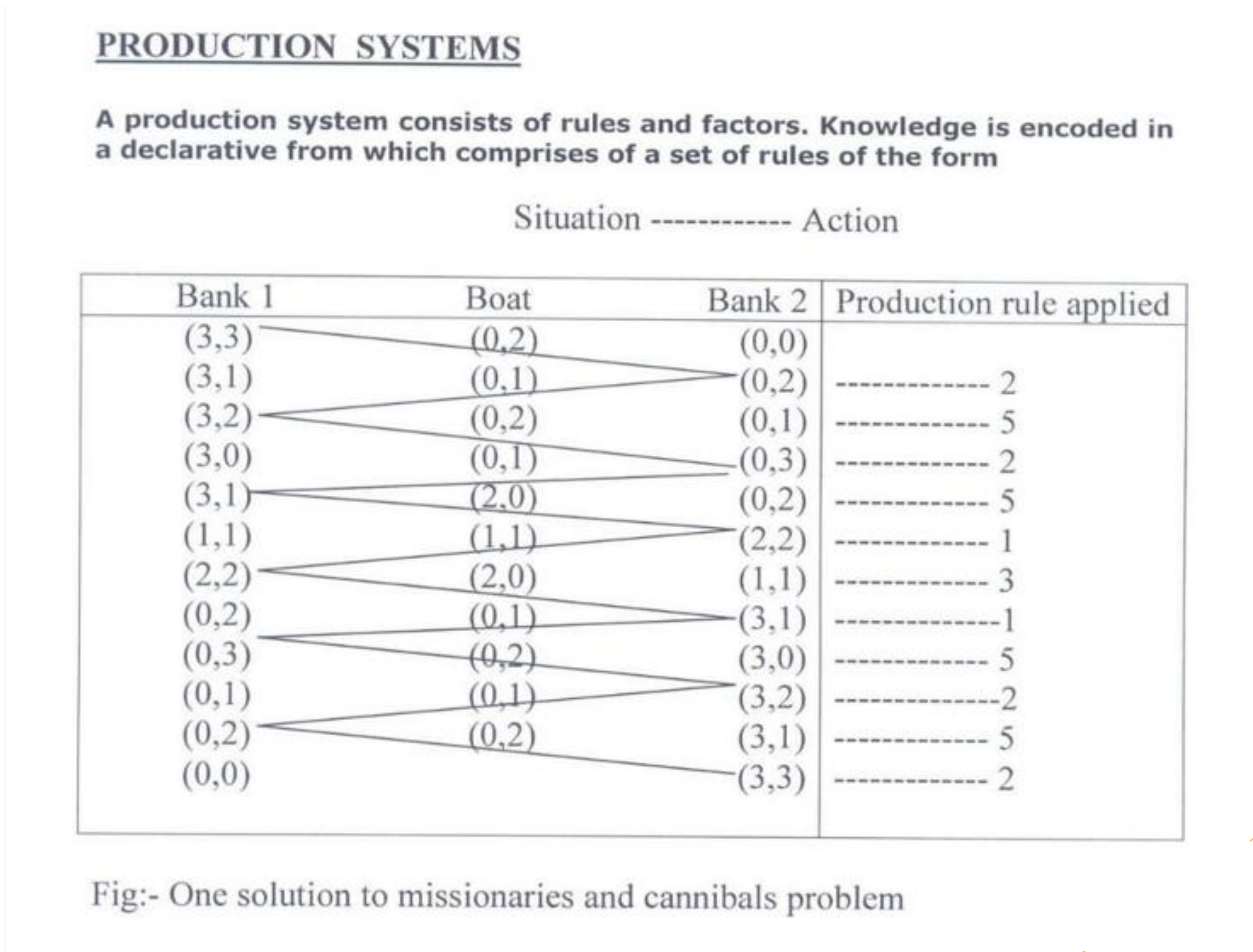
Production Rules for Missionaries and Cannibals Problem.

Rule No	Production Rule and Action
1	(i,j) : Two missionaries can go only when i-2 >=j or i-2=0 in one bank and i+2>=j in the other bank.
2	(i,j) : Two cannibals can cross the river only when j-2, <=i or i=0 in one bank and j+2 <=i or I+0 or i=0 in the other.
3	(i,j) : One missionary and one cannibal can go in a goat only when i-1>= j-1 or i=0 in one bank and i+1 >= j+1 or i=0 in the other.
4	(i,j) : one missionary can cross the river only when i-1>=j or i=0 in one bank and i+1 >=j in the other bank.
5	(i,j) : One cannibal can cross the river only when j-1 < i or i=0 in one bank and j+1<=i or j=0 in the other bank of the river.

Fig:- Production rules for the missionaries and cannibals problem.

For this problem, there are several sequences of operators that will solve the problem . The next figure is one of the solutions.

Missionaries and Cannibals



PRODUCTION SYSTEMS

A production system consists of rules and factors. Knowledge is encoded in a declarative from which comprises of a set of rules of the form

Situation ------------ Action

Bank 1	Boat	Bank 2	Production rule applied
(3,3)	(0,2)	(0,0)	
(3,1)	(0,1)	(0,2)	------------- 2
(3,2)	(0,2)	(0,1)	------------- 5
(3,0)	(0,1)	(0,3)	------------- 2
(3,1)	(2,0)	(0,2)	------------- 5
(1,1)	(1,1)	(2,2)	------------- 1
(2,2)	(2,0)	(1,1)	------------- 3
(0,2)	(0,1)	(3,1)	--------------1
(0,3)	(0,2)	(3,0)	------------- 5
(0,1)	(0,1)	(3,2)	--------------2
(0,2)	(0,2)	(3,1)	------------- 5
(0,0)		(3,3)	------------- 2

Fig:- One solution to missionaries and cannibals problem

Search Strategies/Control Strategies

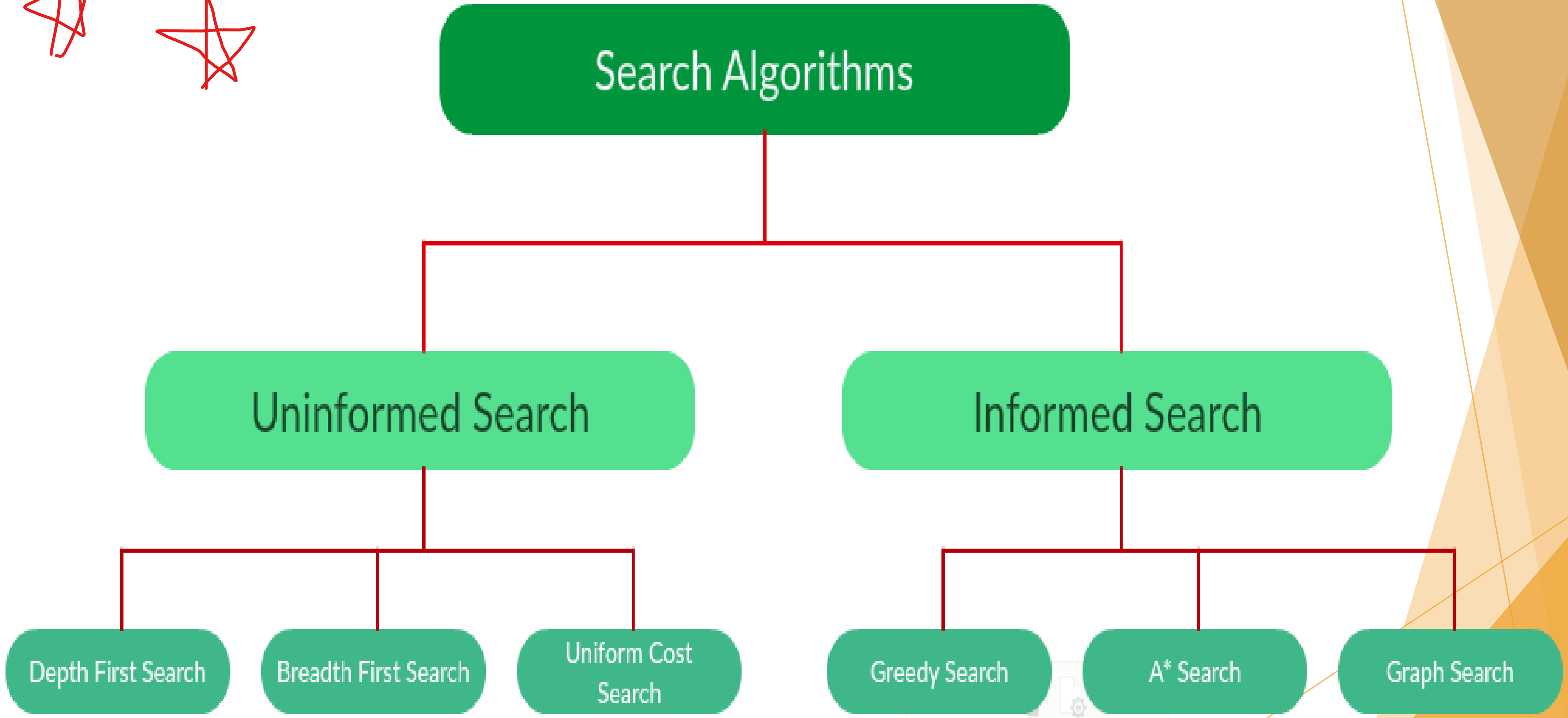
- The word ‘search’ refers to the search for a solution in a problem space.

- Properties of Search Algorithms

- **Completeness:** A search algorithm is complete when it returns a solution for any input if at least one solution exists for that particular input.
- **Optimality:** If the solution deduced by the algorithm is the best solution, i.e. it has the lowest path cost, then that solution is considered as the optimal solution.
- **Time and Space Complexity:** Time complexity is the time taken by an algorithm algorithm to complete its task, and space complexity is the maximum storage space needed during the search operation.

Types of Search strategies

1. **Uninformed search (blind search)**
   Having no information about the number of steps from the current state to the goal.
2. **Informed search (heuristic search)**
   More efficient than uninformed search.


Search Algorithms
Uninformed Search
Informed Search
Depth First Search
Breadth First Search
Uniform Cost Search
Greedy Search
A* Search
Graph Search

Difference Between Search Algorithms

Criteria	Uninformed Search	Informed Search
Utilizing Knowledge	It does not require using any knowledge during the process of searching.	It uses knowledge during the process of searching.
Speed	Finding the solution is much slower comparatively.	Finding the solution is quicker.
Completion	It is always bound to give optimal solution.	It may or may not give optimal solution.
Consumption of Time	Due to slow searches, it consumes comparatively more time.	Due to a quicker search, it consumes much less time.
Cost Incurred	The expenses are comparatively higher.	The expenses are much lower.
Suggestion/ Direction	The AI does not get any suggestions regarding what solution to find and where to find it. Whatever knowledge it gets is out of the information provided.	The AI gets suggestions regarding how and where to find a solution to any problem.
Efficiency	It costs more and generates slower results. Thus, it is comparatively less efficient.	It costs less and generates quicker results. Thus, it is comparatively more efficient.
Examples	Breadth-First Search or BFS and Depth-First Search or DFS.	Graph Search ,Greedy Search , A*,AO*

Uninformed Search

- The search algorithms in this section have no additional information on the goal node other than the one provided in the problem definition.
- The plans to reach the goal state from the start state differ only by the order and/or length of actions. Uninformed search is also called Blind search.

1. Depth First Search
2. Breath First Search
3. Uniform Cost Search

Uninformed Search

Each of these algorithms will have:

1. A problem graph, containing the start node S and the goal node G.

2. A strategy, describing the manner in which the graph will be traversed to get to G .

3. A fringe, which is a data structure used to store all the possible states (nodes) that you can go from the current states.

4. A tree, that results while traversing to the goal node.

5. A solution plan, which the sequence of nodes from S to G.

Queue for Frontier

☆ ☆ ☆

- FIFO (First In, First Out)
  - Results in Breadth-First Search
- LIFO (Last In, First Out)
  - Results in Depth-First Search
- Priority Queue sorted by path cost so far
  - Results in Uniform Cost Search
- Iterative Deepening Search uses Depth-First
- Bidirectional Search can use either Breadth-First or Uniform Cost Search

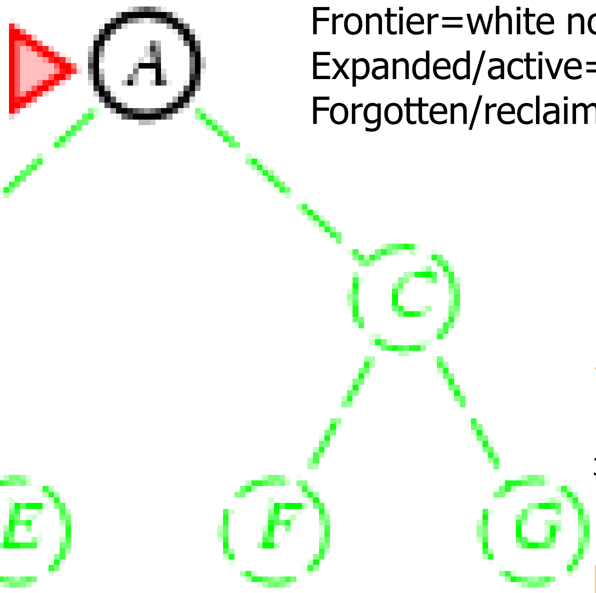
BFS-Breadth First Search

- Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures.
- It starts at the tree root, and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

Breadth-first search

- Expand shallowest unexpanded node
- *Frontier* (or fringe): nodes in queue to be explored
- *Frontier* is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.
- *Goal-Test* when inserted.

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes



Initial state = A
Is A a goal state?

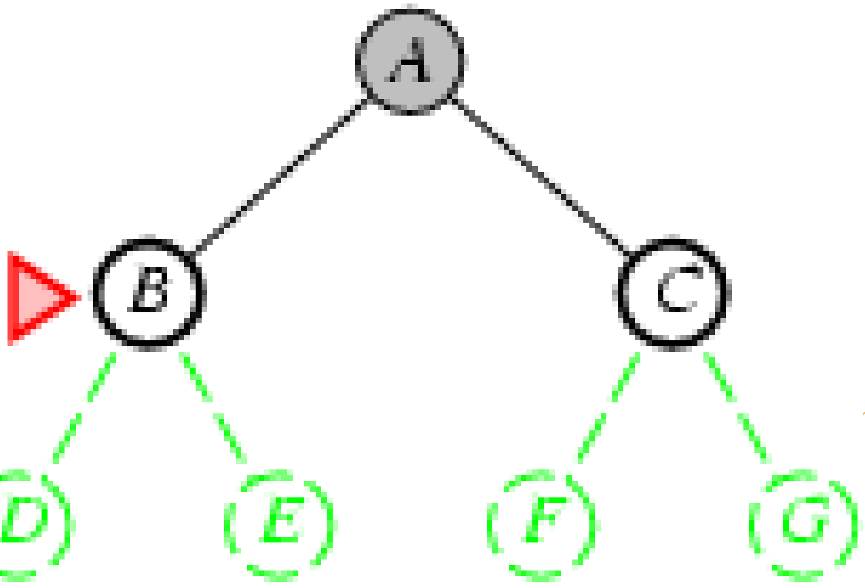
Put A at end of queue.
frontier = [A]

Breadth-first search

- Expand shallowest unexpanded node
- *Frontier* is a FIFO queue, i.e., new successors go at end

Expand A to B, C.
Is B or C a goal state?

Put B, C at end of queue.
frontier = [B,C]

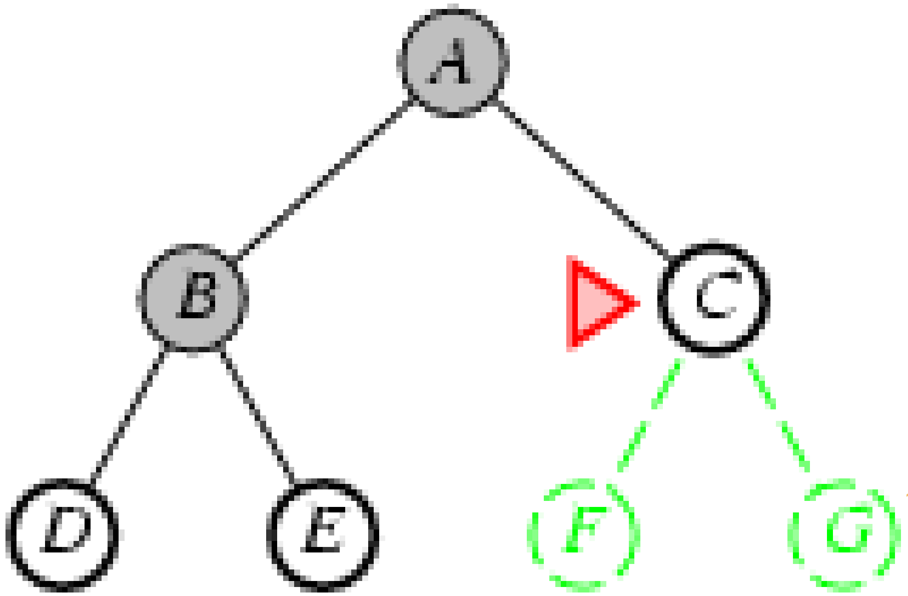


Breadth-first search

- Expand shallowest unexpanded node
- *Frontier* is a FIFO queue, i.e., new successors go at end

Expand B to D, E
Is D or E a goal state?

Put D, E at end of queue
frontier=[C,D,E]

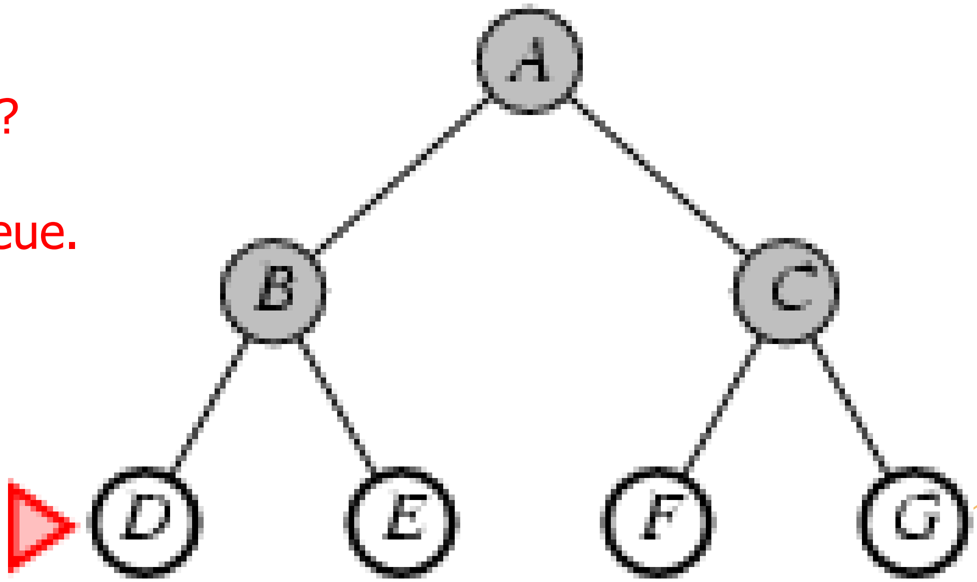


Breadth-first search

- Expand shallowest unexpanded node
- *Frontier* is a FIFO queue, i.e., new successors go at end

Expand C to F, G.
Is F or G a goal state?

Put F, G at end of queue.
frontier = [D,E,F,G]

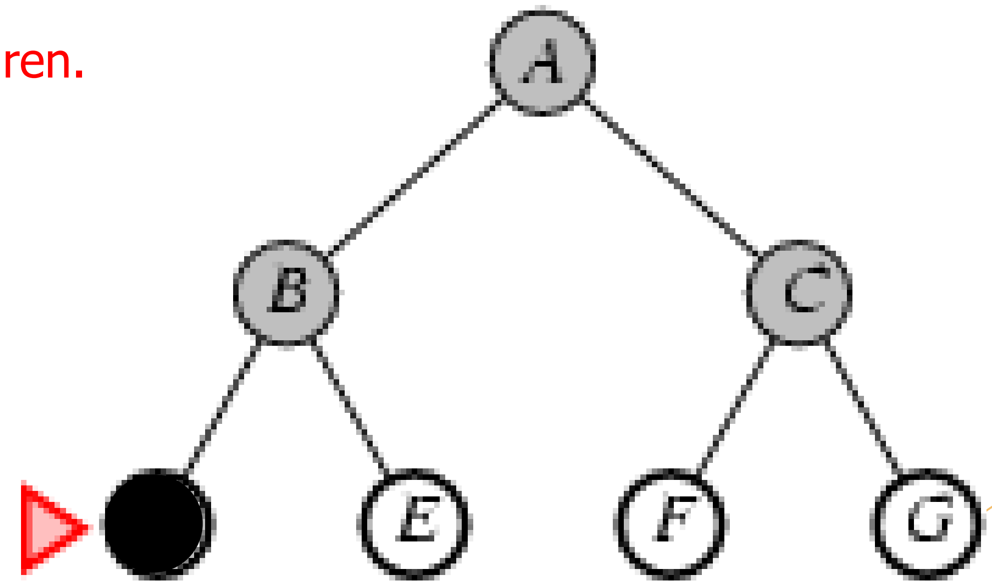


Breadth-first search

- Expand shallowest unexpanded node
- *Frontier* is a FIFO queue, i.e., new successors go at end

Expand D to no children.
Forget D.

frontier = [E,F,G]

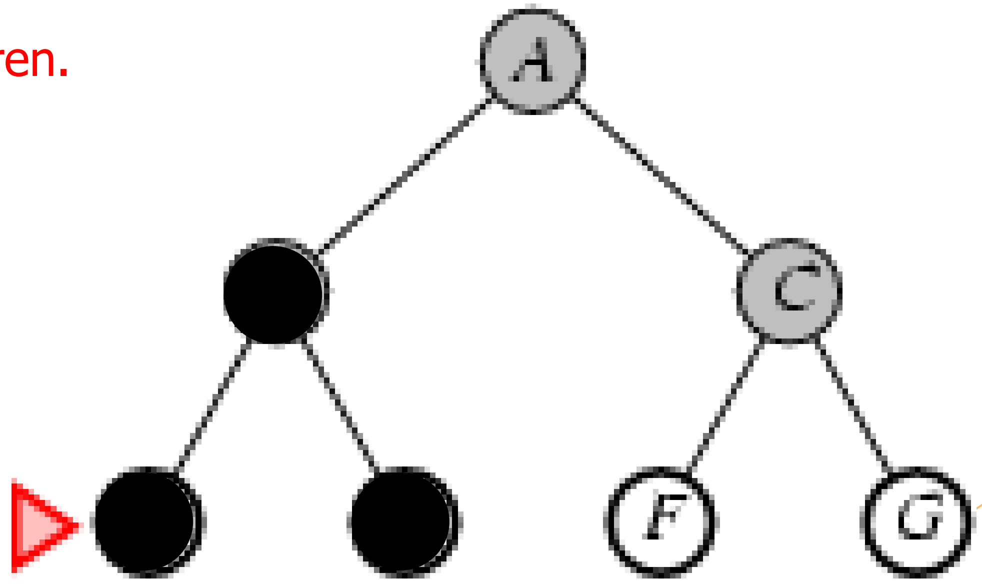


Breadth-first search

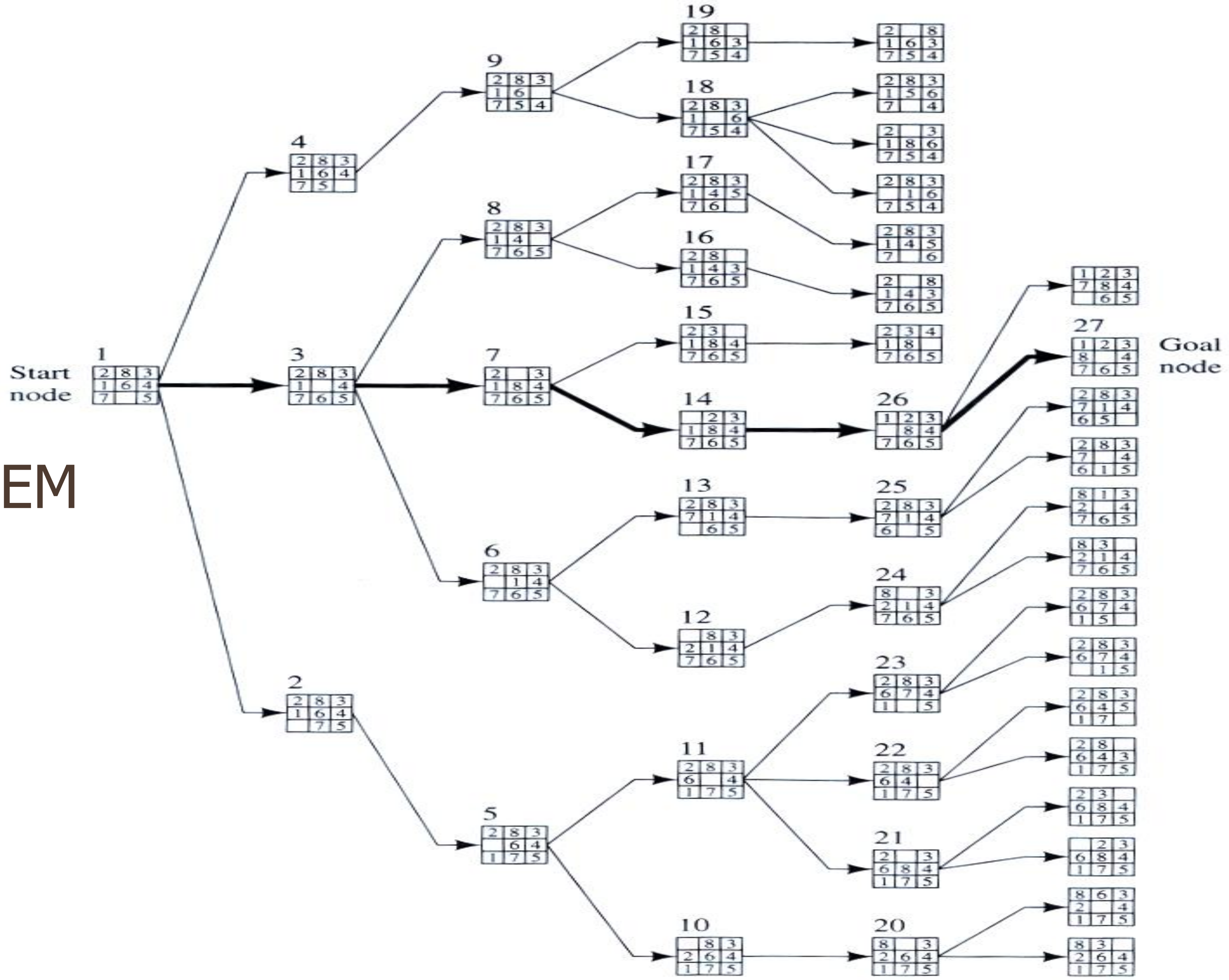
- Expand shallowest unexpanded node
- *Frontier* is a FIFO queue, i.e., new successors go at end

Expand E to no children.
Forget B,E.

frontier = [F,G]



Example
BFS
8 PUZZLE PROBLEM



Properties of breadth-first search

- Complete? Yes, it always reaches a goal (if *b* is finite)
- Time? $1+b+b^2+b^3+\ldots + b^d = O(b^d)$
  (this is the number of nodes we generate)
- Space? $O(b^d)$ (keeps every node in memory,
  either in fringe or on a path to fringe).
- Optimal? No, for general cost functions.
  Yes, if cost is a non-decreasing function only of depth.
  - With $f(d) \geq f(d-1)$, e.g., step-cost = constant:
    - All optimal goal nodes occur on the same level
    - Optimal goal nodes are always shallower than non-optimal goals
    - An optimal goal will be found before any non-optimal goal
- Space is the bigger problem (more than time)

DFS-Depth First Search

- Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures.
- The algorithm starts at the root node and explores as far as possible along each branch before backtracking.

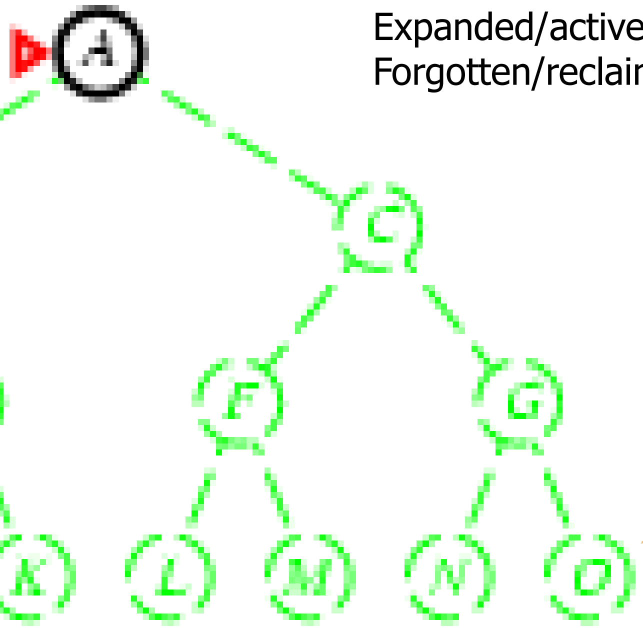
Depth-first search

- Expand *deepest* unexpanded node
- *Frontier* = Last In First Out (LIFO) queue, i.e., new successors go at the front of the queue.
- *Goal-Test* when inserted.

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

Initial state = A
Is A a goal state?

Put A at front of queue.
frontier = [A]

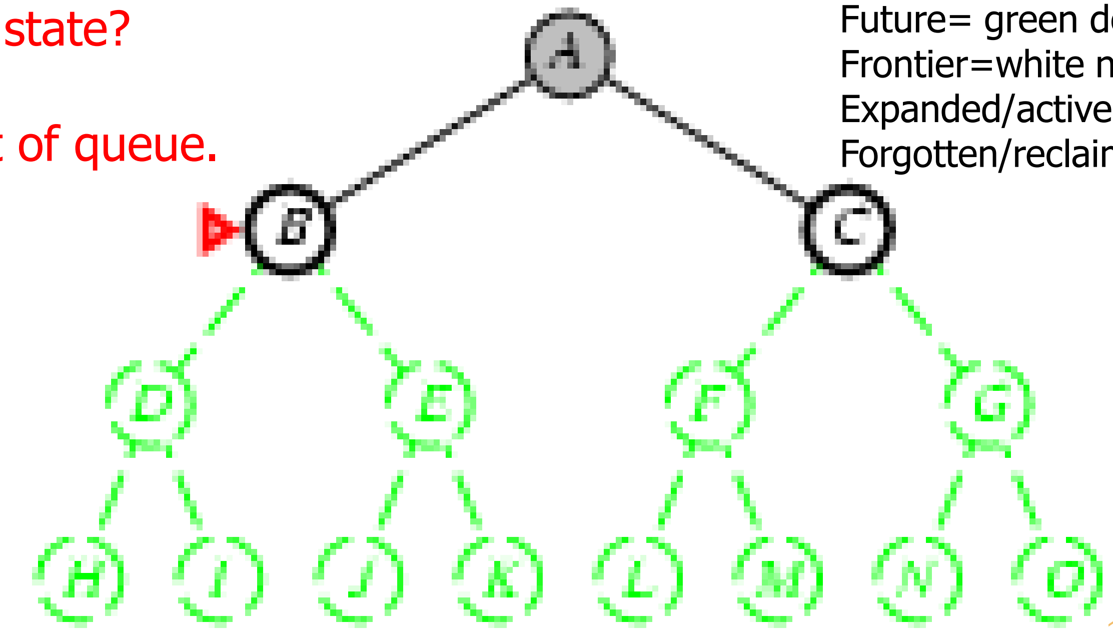


Depth-first search

- Expand deepest unexpanded node
  - *Frontier* = LIFO queue, i.e., put successors at front

Expand A to B, C.
Is B or C a goal state?

Put B, C at front of queue.
frontier = [B,C]



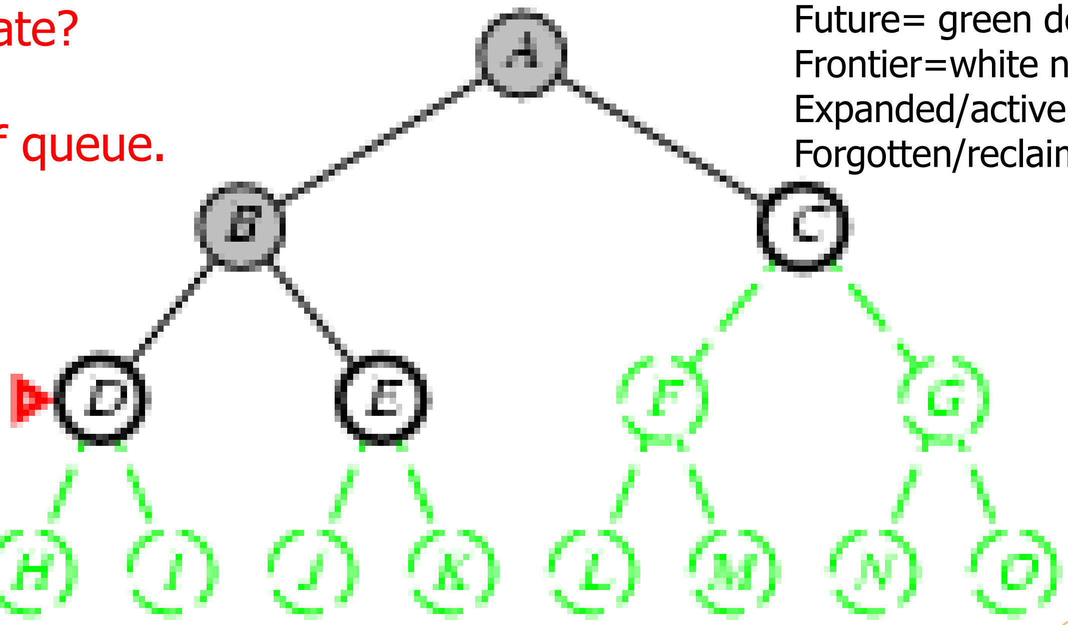
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

Depth-first search

- Expand deepest unexpanded node
  - *Frontier* = LIFO queue, i.e., put successors at front

Expand B to D, E.
Is D or E a goal state?

Put D, E at front of queue.
frontier = [D,E,C]



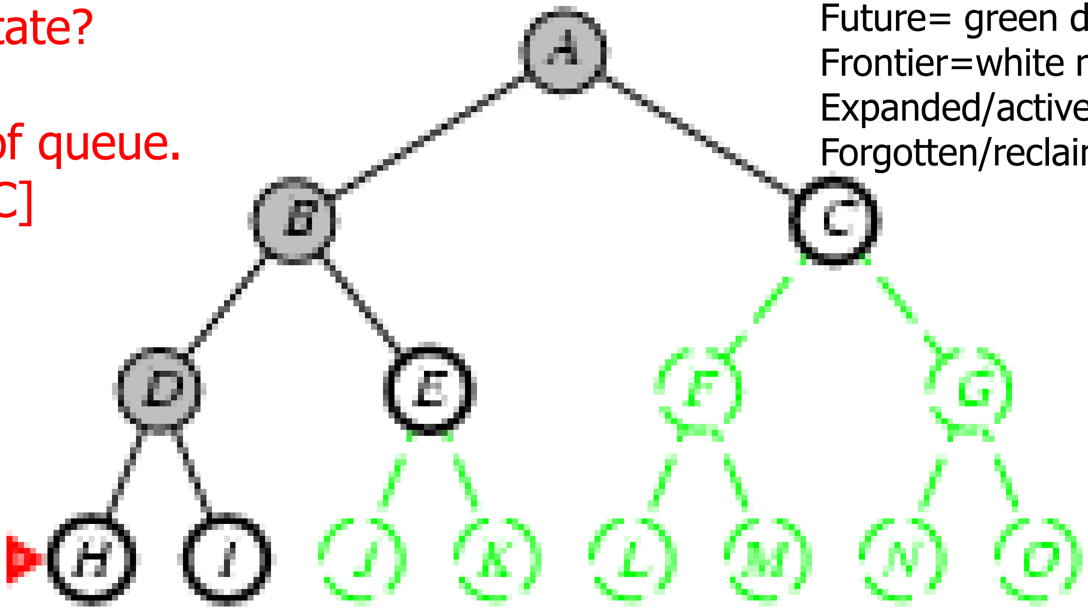
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

Depth-first search

- Expand deepest unexpanded node
  - *Frontier* = LIFO queue, i.e., put successors at front

Expand D to H, I.
Is H or I a goal state?

Put H, I at front of queue.
frontier = [H,I,E,C]



Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

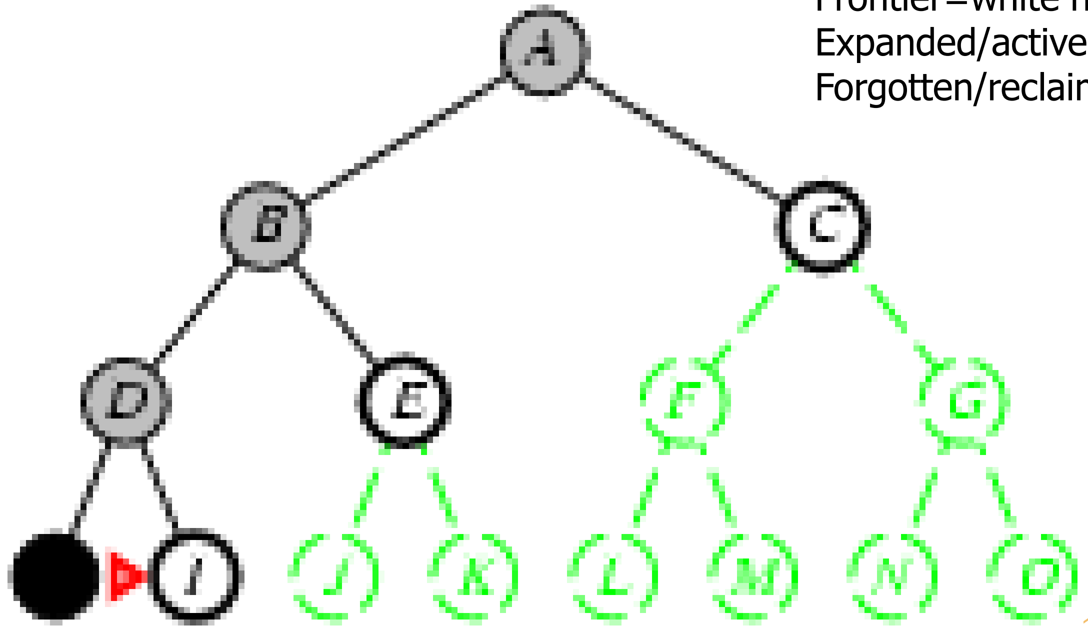
Depth-first search

- Expand deepest unexpanded node
  - *Frontier* = LIFO queue, i.e., put successors at front

Expand H to no children.
Forget H.

frontier = [I,E,C]

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

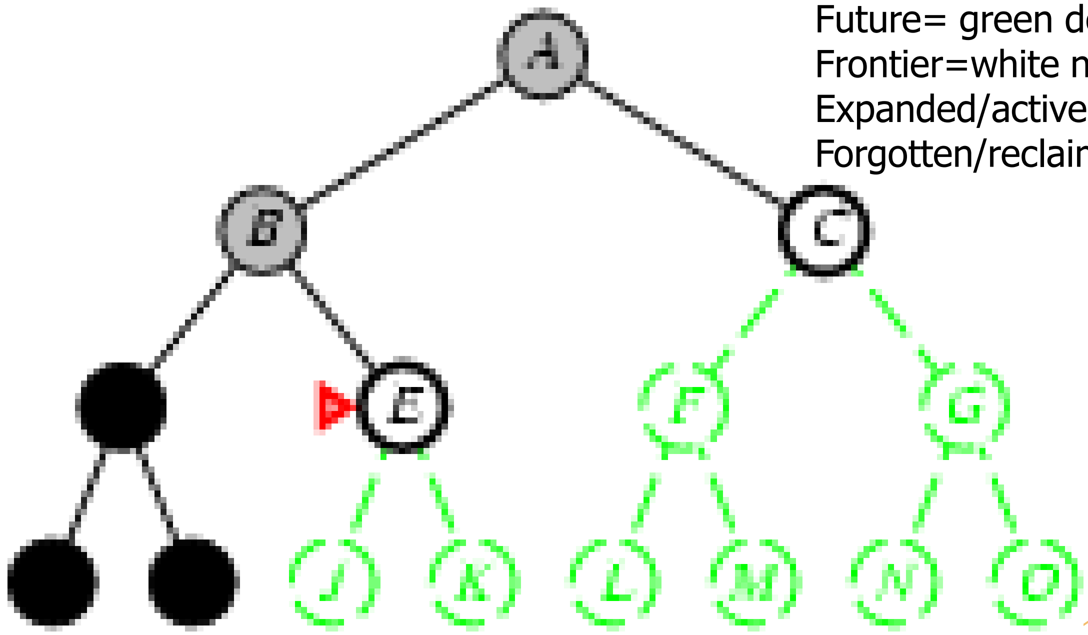


Depth-first search

- Expand deepest unexpanded node
  - *Frontier* = LIFO queue, i.e., put successors at front

Expand I to no children.
Forget D, I.

frontier = [E,C]



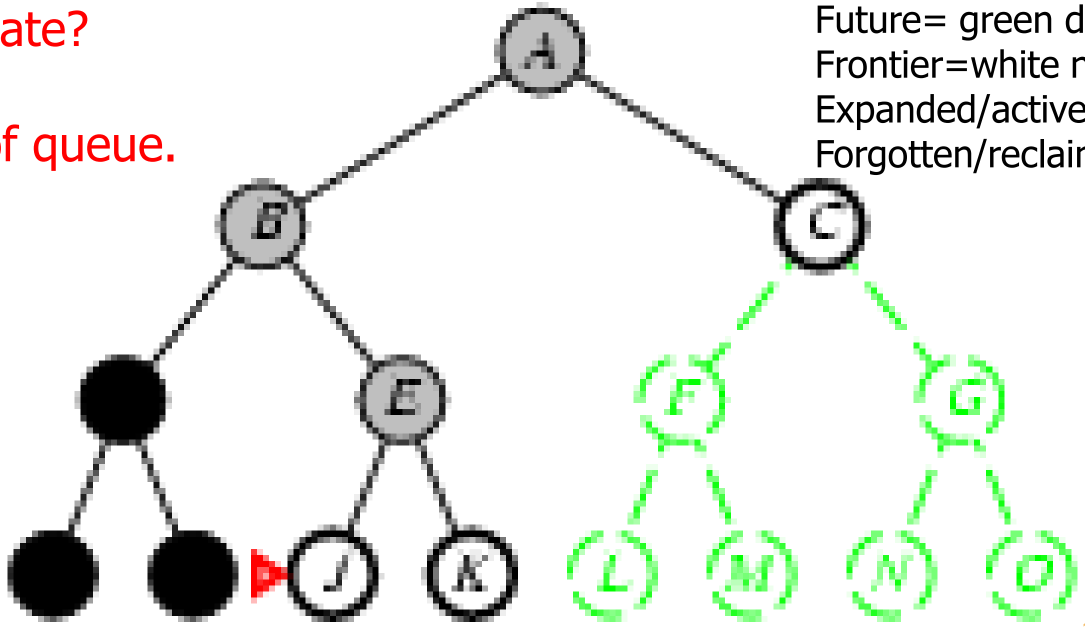
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

Depth-first search

- Expand deepest unexpanded node
  - *Frontier* = LIFO queue, i.e., put successors at front

Expand E to J, K.
Is J or K a goal state?

Put J, K at front of queue.
frontier = [J,K,C]



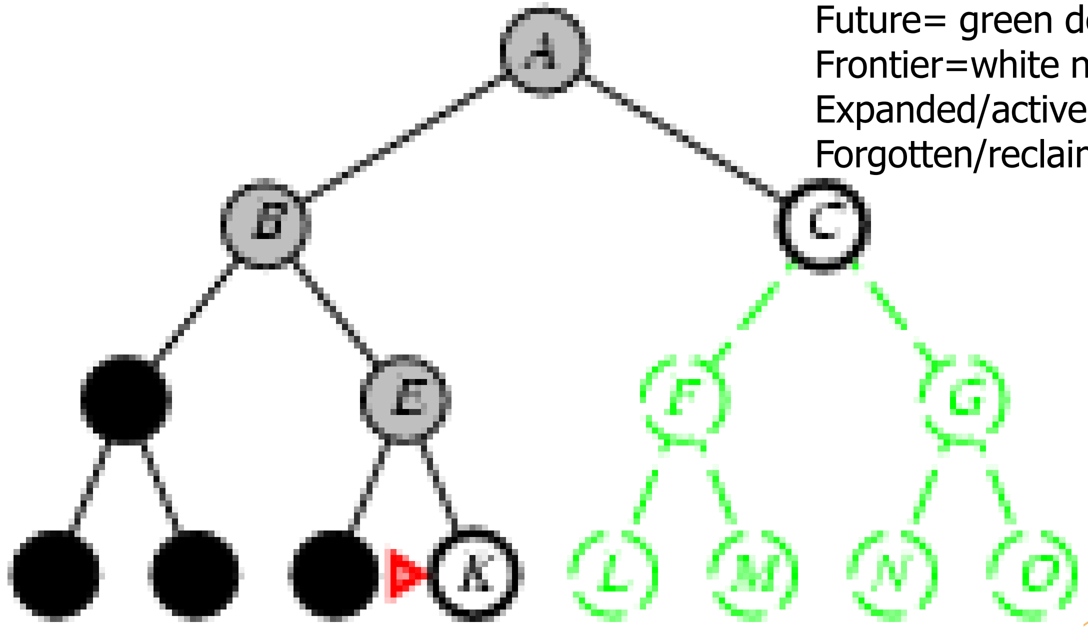
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

Depth-first search

- Expand deepest unexpanded node
  - *Frontier* = LIFO queue, i.e., put successors at front

Expand I to no children.
Forget D, I.

frontier = [E,C]



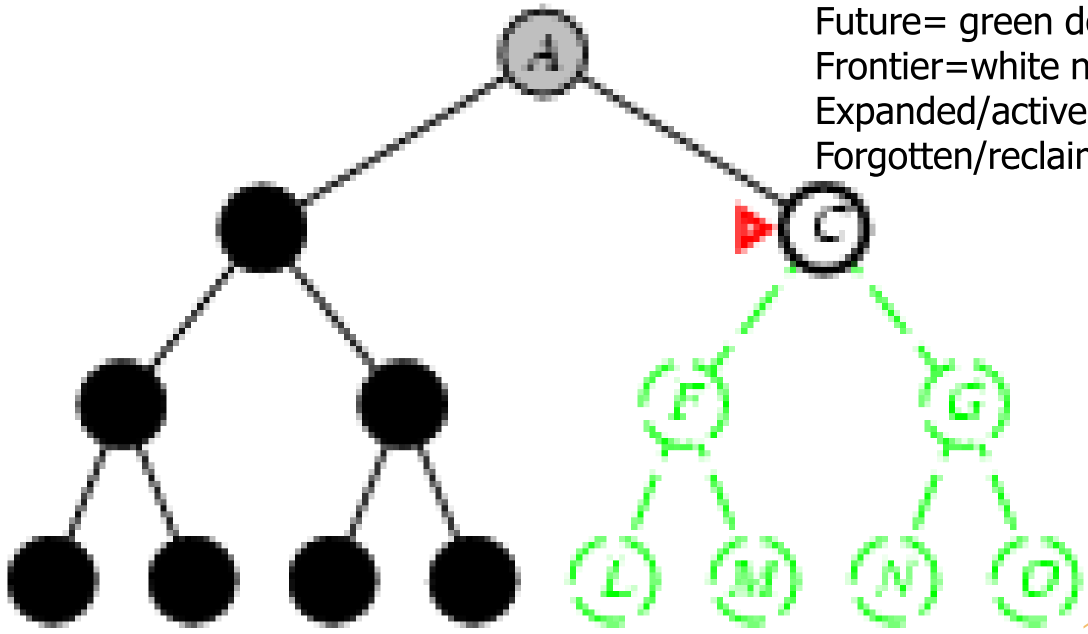
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

Depth-first search

- Expand deepest unexpanded node
  - *Frontier* = LIFO queue, i.e., put successors at front

Expand K to no children.
Forget B, E, K.

frontier = [C]



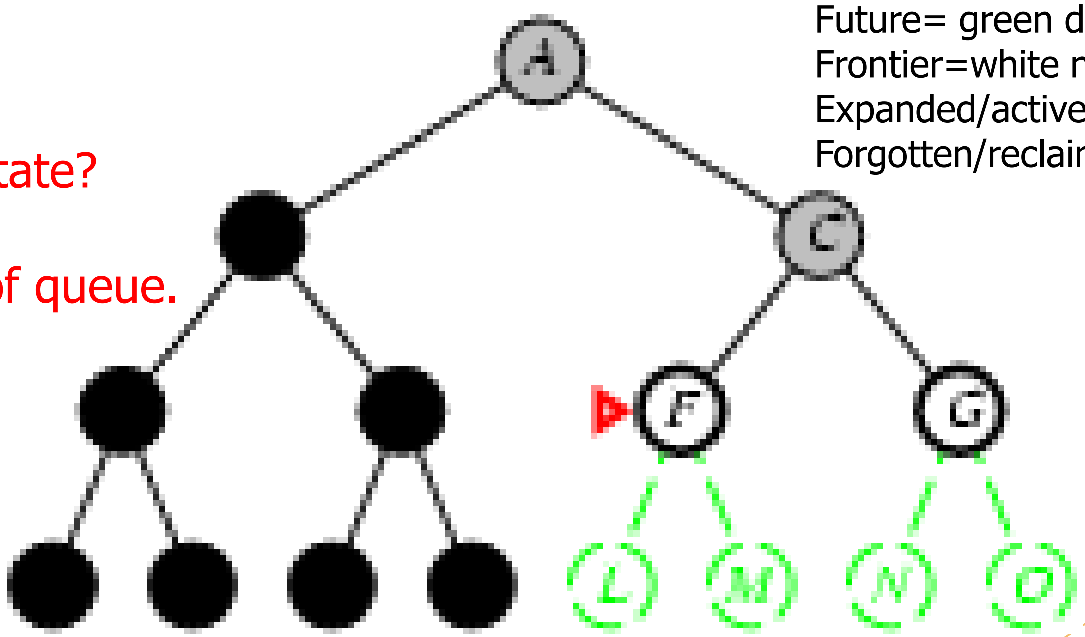
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

Depth-first search

- Expand deepest unexpanded node
  - *Frontier* = LIFO queue, i.e., put successors at front

Expand C to F, G.
Is F or G a goal state?

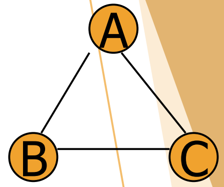
Put F, G at front of queue.
frontier = [F,G]



Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

Properties of depth-first search

- Complete? No: fails in loops/infinite-depth spaces
  - Can modify to avoid loops/repeated states along path
    - check if current nodes occurred before on path to root
  - Can use graph search (remember all nodes ever seen)
    - problem with graph search: space is exponential, not linear
  - Still fails in infinite-depth spaces (may miss goal entirely)
- Time? $O(b^m)$ with m =maximum depth of space
  - Terrible if m is much larger than d
  - If solutions are dense, may be much faster than BFS
- Space? $O(bm)$, i.e., linear space!
  - Remember a single path + expanded unexplored nodes
- Optimal? No: It may find a non-optimal goal first



Uniform-cost search

Breadth-first is only optimal if path cost is a non-decreasing function of depth, i.e., $f(d) \geq f(d-1)$; e.g., constant step cost, as in the 8-puzzle.

Can we guarantee optimality for variable positive step costs $\geq \varepsilon$?

(Why $\geq \varepsilon$? To avoid infinite paths w/ step costs 1, ½, ¼, …)

Uniform-cost Search:

Expand node with smallest path cost g(n).

- *Frontier* is a priority queue, i.e., new successors are merged into the queue sorted by g(n).
  - Can remove successors already on queue w/higher g(n).
    - Saves memory, costs time; another space-time trade-off.
- *Goal-Test* when node is popped off queue.

Uniform-cost search

Uniform-cost Search:

Expand node with smallest path cost g(n).

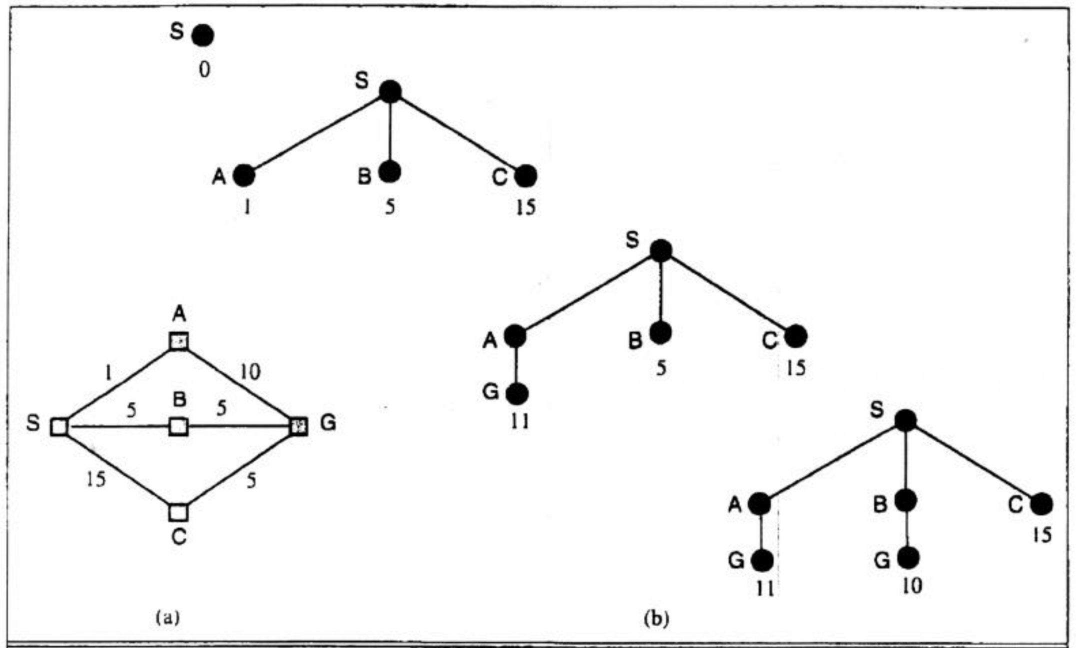


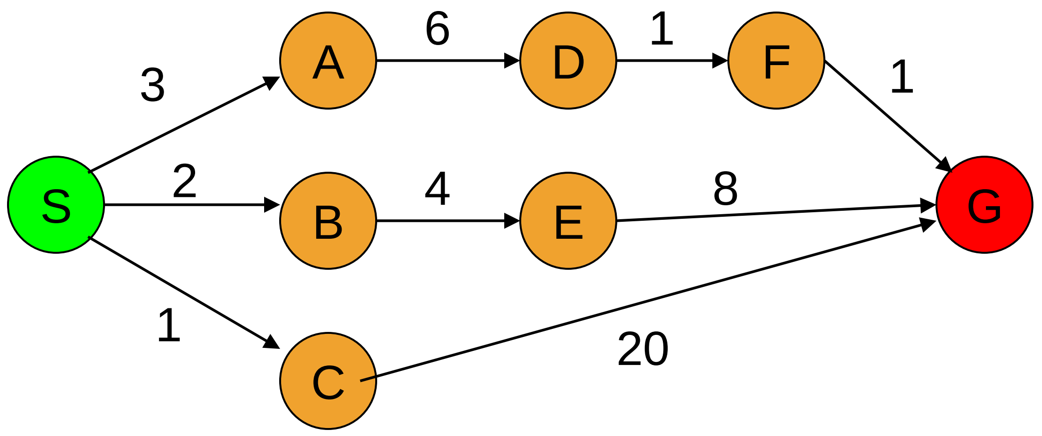
Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

Proof of Completeness:

Given that every step will cost more than 0, and assuming a finite branching factor, there is a finite number of expansions required before the total path cost is equal to the path cost of the goal state. Hence, we will reach it.

Proof of optimality given completeness:

Assume UCS is not optimal.
Then there must be an (optimal) goal state with path cost smaller than the found (suboptimal) goal state (invoking completeness).
However, this is impossible because UCS would have expanded that node first by definition.
Contradiction.



The graph above shows the step-costs for different paths going from the start (S) to the goal (G).

Use uniform cost search to find the optimal path to the goal.

Exercise for at home

References

(Chapter 3 – AI Rich & Knight)