Artificial Intelligence

By Dr. Naglaa fathy
Lecturer, Computer Science Department
Faculty of Computers and Artificial Intelligence
Benha University

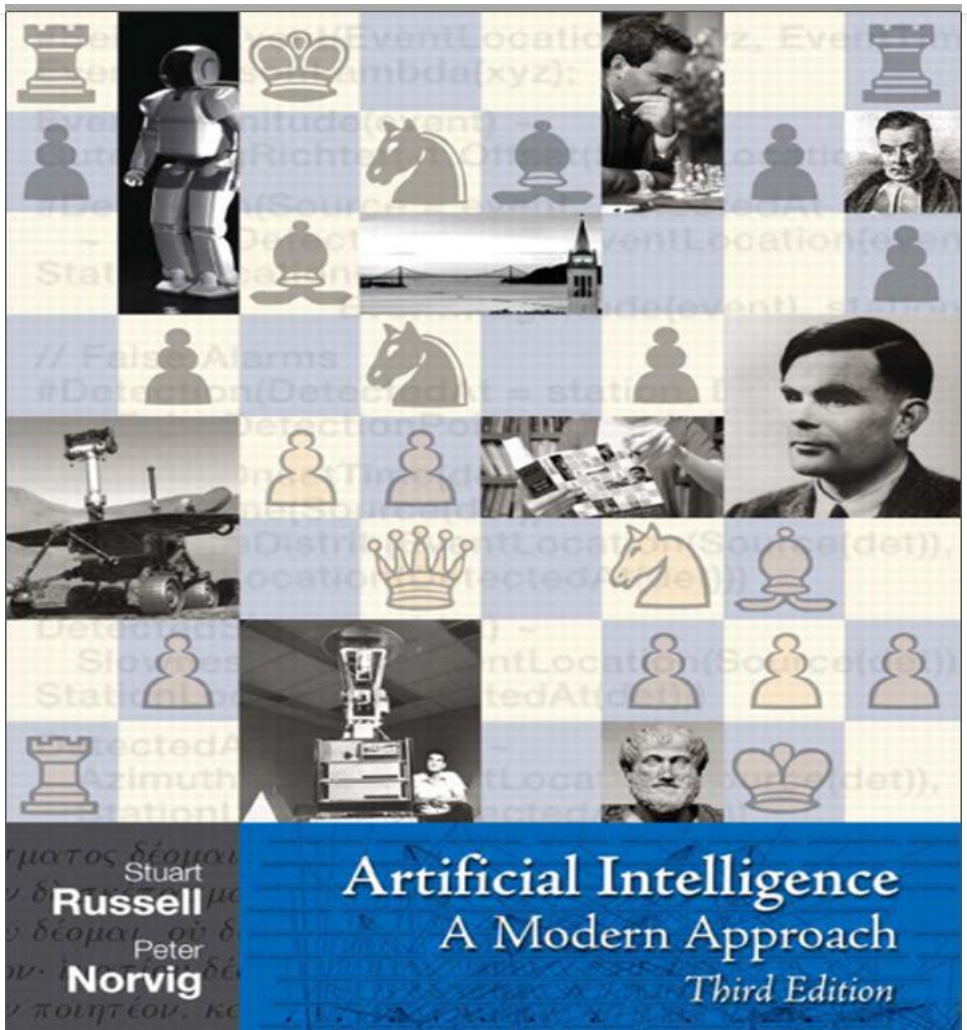
Lecture 3
Solving problem by searching

Lectures References

Artificial Intelligence
A Modern Approach

Third Edition

Stuart J. Russell and Peter Norvig



Problem-solving Agents

- Intelligent agents are supposed to maximize their performance measures.
- Represent the problem as STATES and OPERATORS that transform one state into another state.
- A solution to the problem is an OPERATOR SEQUENCE that transforms the INITIAL STATE into a GOAL STATE.
- Finding the sequence requires SEARCHING the STATE SPACE by GENERATING the paths connecting the two.
-

Problem-solving Agent

- **Goal formulation**
  - based on the current situation and the agent's performance measure
- **Problem formulation**
  - the process of deciding what actions and states to consider, given a goal.
    - Initial state
    - Goal state
    - Actions
    - Transition model(successor)
    - Path cost
- **Search**
  - examine different possible sequences of actions that lead to states known value and choose the best sequences
- **Execute**
  - perform actions based on the solution

Ok…Let's review

- What was the initial state?
- What was the goal state?
- What was the set of operations that took us from the initial state to the goal state?
- What is the path that, if followed, would get us from the initial state to the goal state?
- What would be the STATE SPACE?

Basic concepts

- State: finite representation of the world that you want to explore at a given time.
- Operator: a function that transforms a state into another (also called rule, transition, successor function, production, action).
- Initial state: The problem at the beginning.
- Goal state: desired end state (can be several)
- Goal test: test to determine if the goal has been reached.
- Solution Path: The sequence of actions that get you from the initial state to the goal state.

Basic concepts *cont.*,

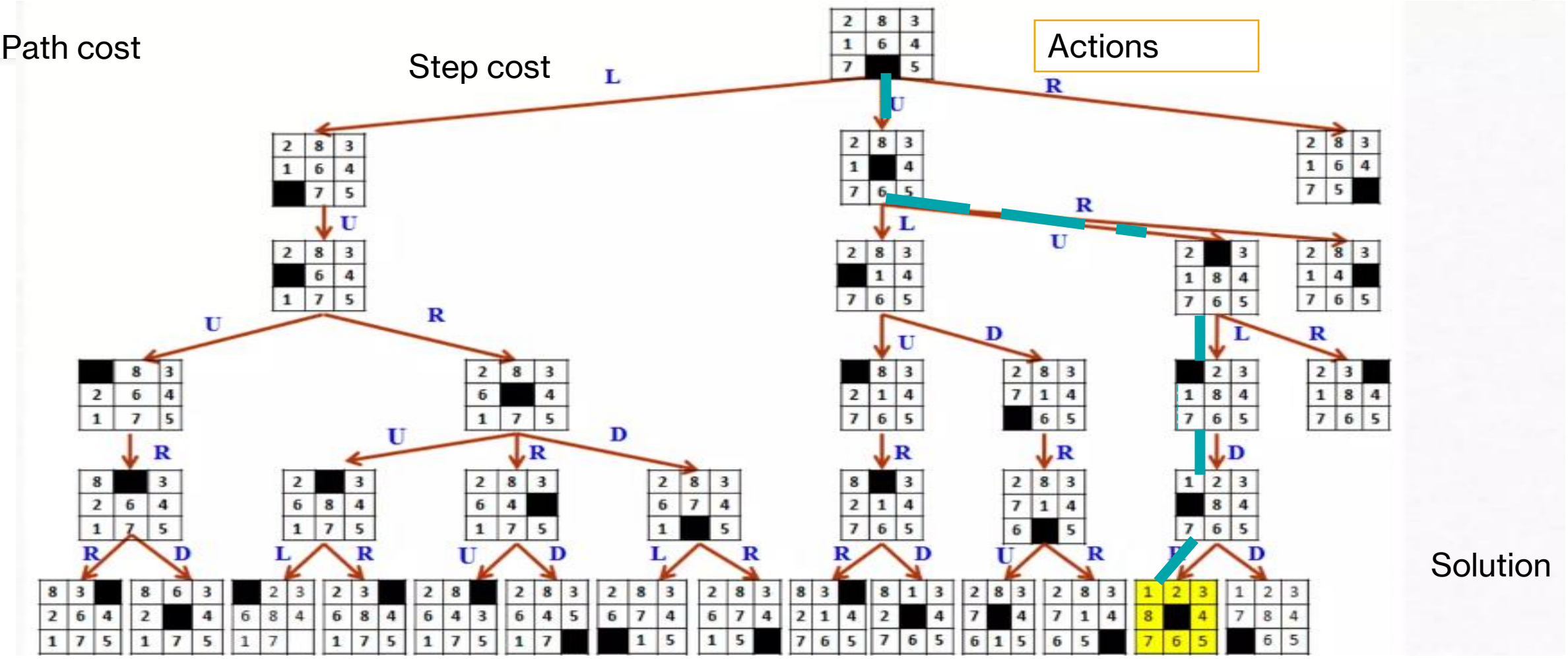
- State space: set of all reachable states from the initial state (possibly infinite).
- Cost function: a function that assigns a cost to each operation
- A path cost function that assigns a numeric cost to each path.
- A solution to a problem is an action sequence that leads from the initial state to a goal state.

- Performance (not for ALL uninformed):
  - cost of the final operator sequence
  - cost of finding the sequence

path

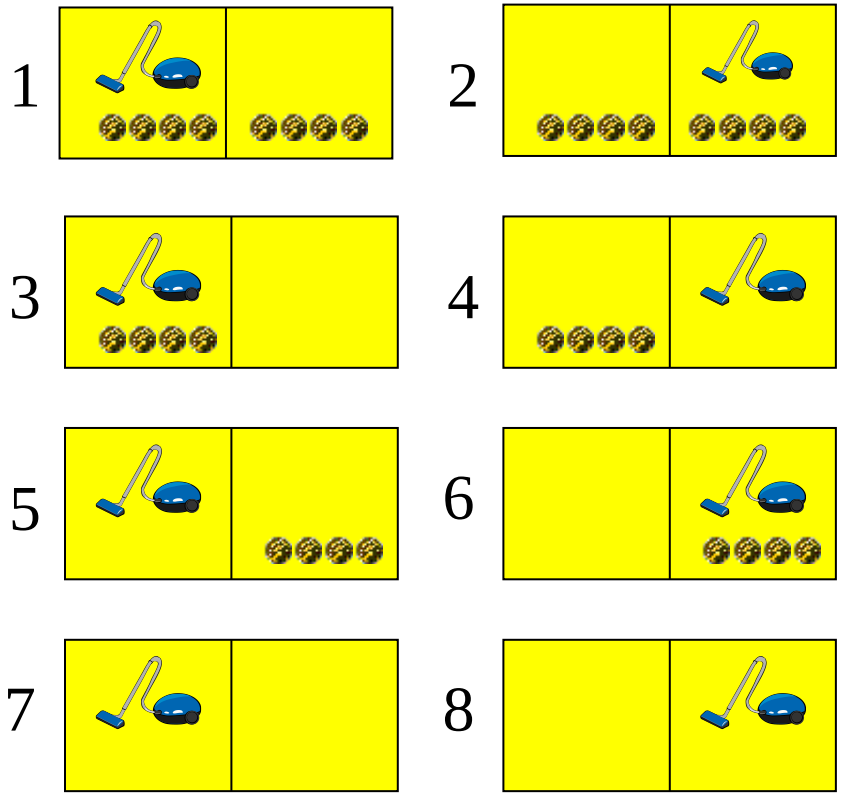
Path cost

Step cost



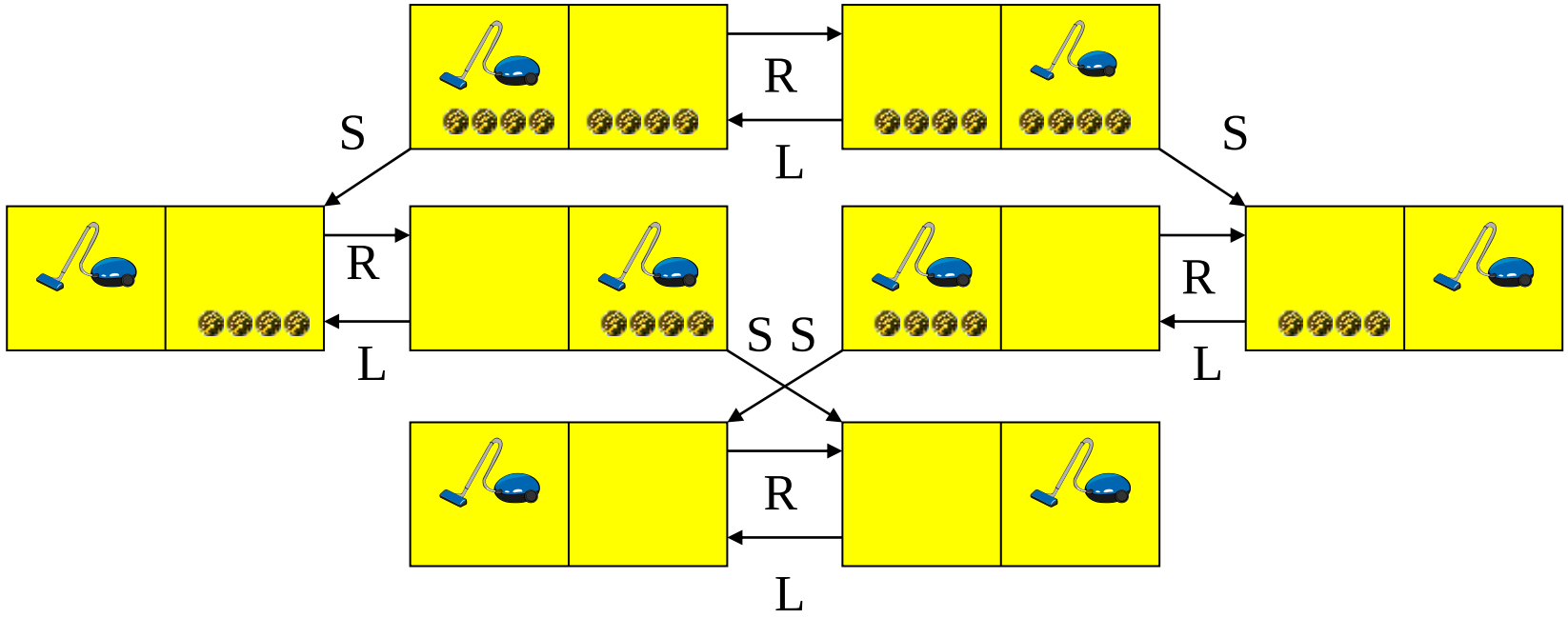
Toy Problems
The vacuum world

- The vacuum world
  - The world has only two *locations*
  - Each location may or may not contain *dirt*
  - The agent may be in one location or the other
  - 8 possible *world states*
  - Three possible actions: *Left*, *Right*, *Suck*
  - *Goal*: clean up all the dirt



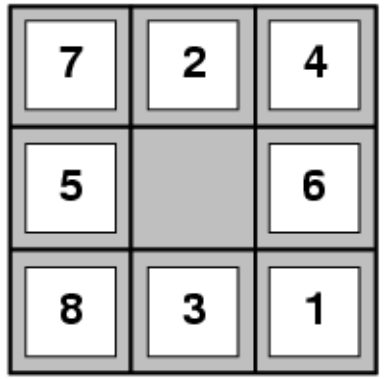
Toy Problems
The vacuum world

- *States*: one of the 8 states given earlier
- *Operators(actions)*: move left, move right, suck
- *Goal test*: no dirt left in any square
- *Path cost*: each action costs one

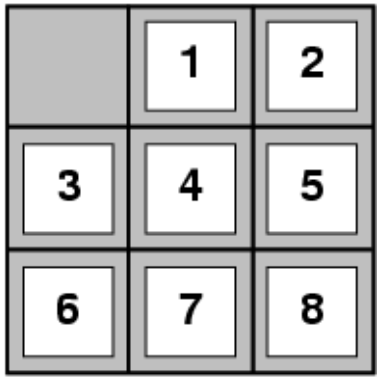


Example: The 8-puzzle

- states? locations of tiles
- Initial state? Any configuration
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move



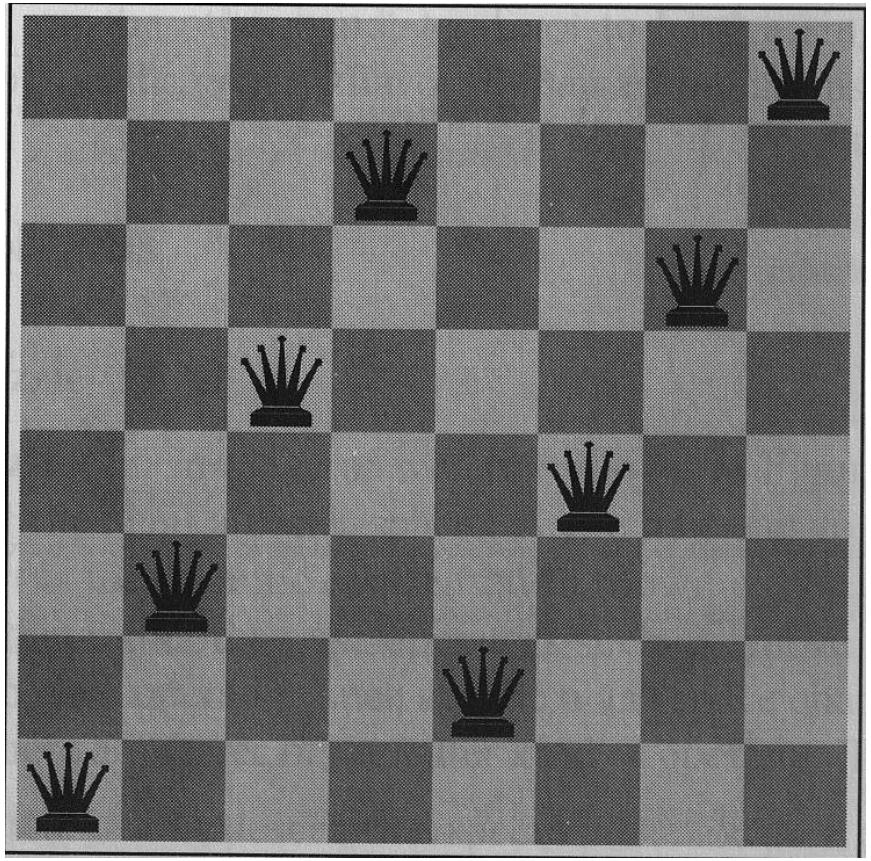
Start State



Goal State

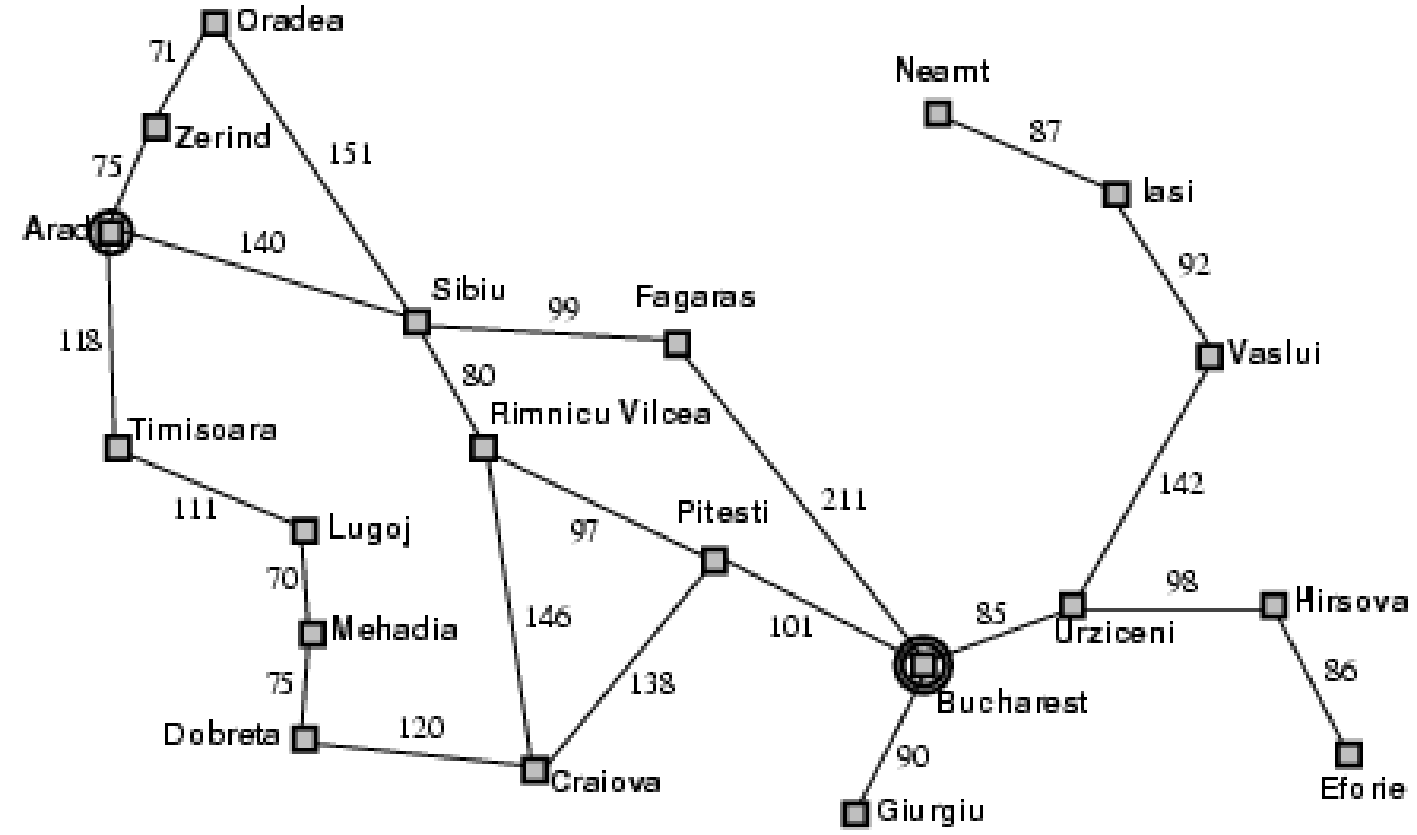
Example : 8-queens

- State? any arrangement of up to 8 queens on the board
- Operation? add a queen (incremental), move a queen (fix-it)
- Initial state? no queens on board
- Goal state? 8 queens, with no queen, are attacked (A queen attacks any piece in the same row, column, or diagonal).
- Solution Path? The set of operations that allowed you to get to the board that you see above at the indicated positions.

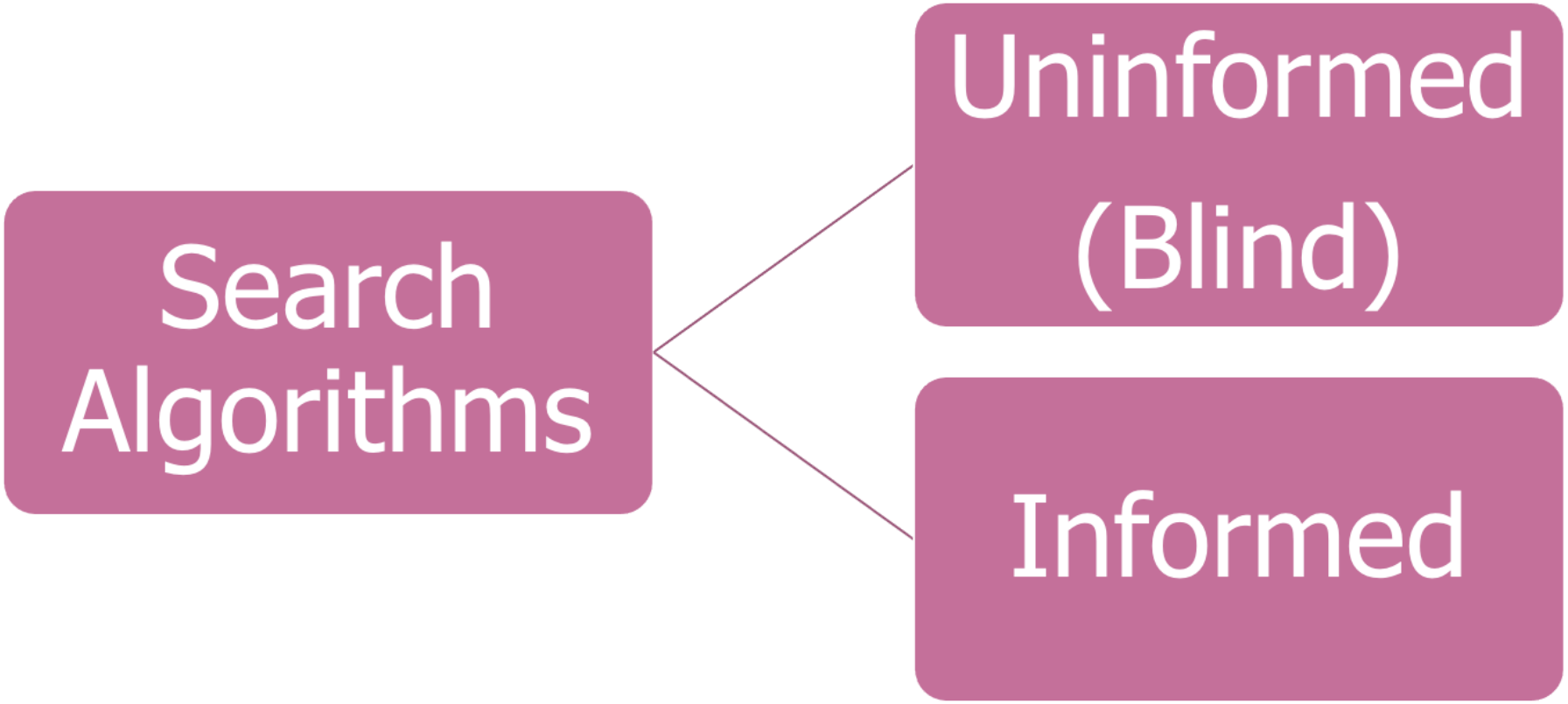


Example: Romania

- On holiday in Bucharest, currently in Arad
- states? Various cities
- Initial state? Arad
- actions? Driver between cities or choose the next city
- goal test? Bucharest
- path cost? Distance in km
- Goal state? Optimal sequences of solution
- Solution: sequence of cities. Arad, Sibiu, Fagaras, Bucharest



Search Algorithms



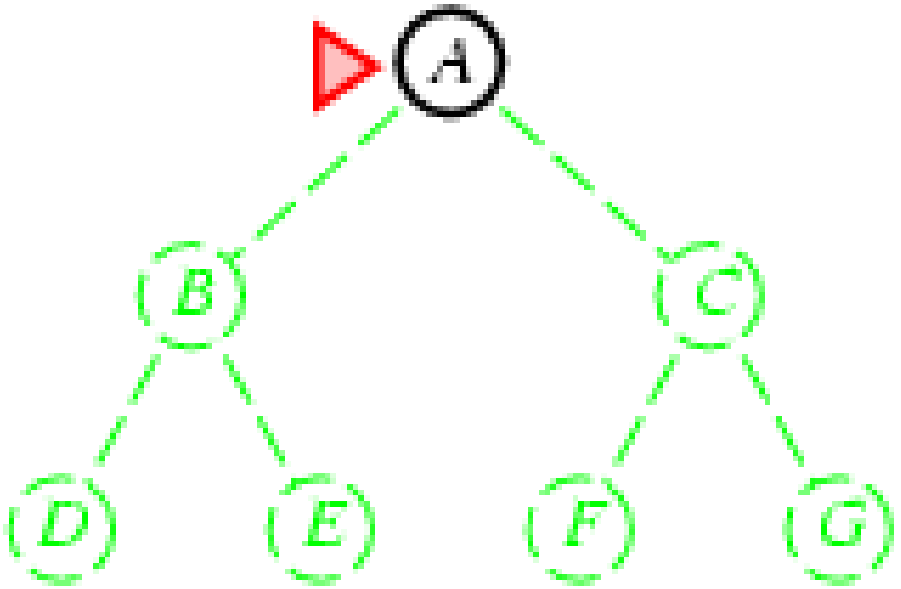
Uninformed search strategies

- Uninformed: While searching you have no clue whether one non-goal state is better than any other. Your search is blind.
- Various blind strategies:
- Breadth-first search(BFS)
- Uniform-cost search(UCS)
- Depth-first search(DFS)
- Iterative deepening search(IDS)

Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a first-in-first-out (FIFO) queue, i.e., new successors go at the end of the queue.

Is A a goal state?

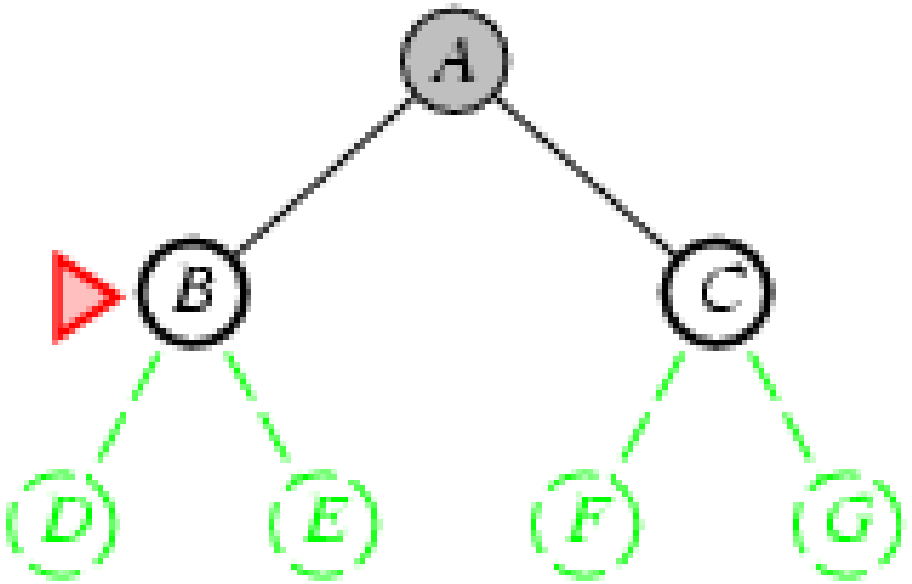


Breadth-first search

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

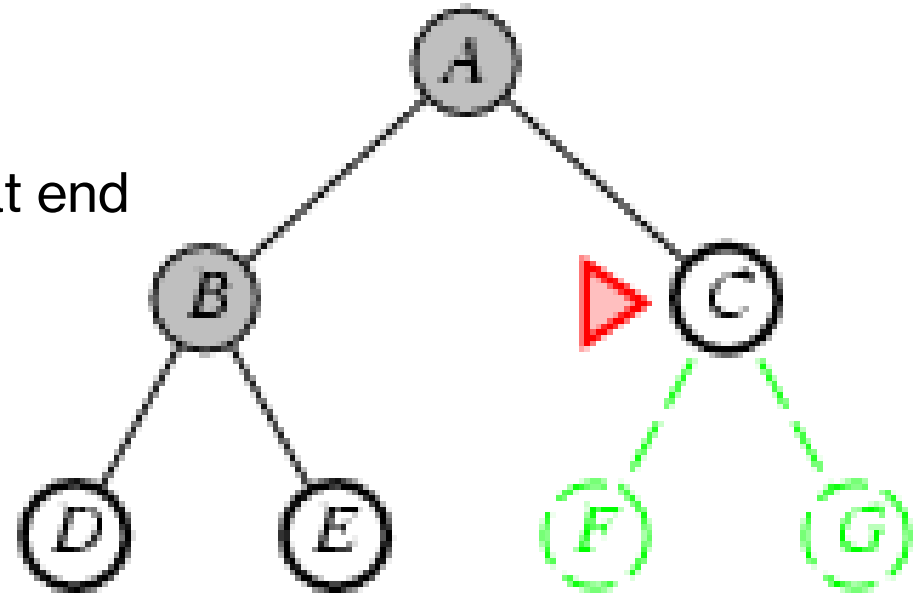
Expand:
fringe = [B,C]

Is B a goal state?



Breadth-first search

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



Expand:
fringe=[C,D,E]

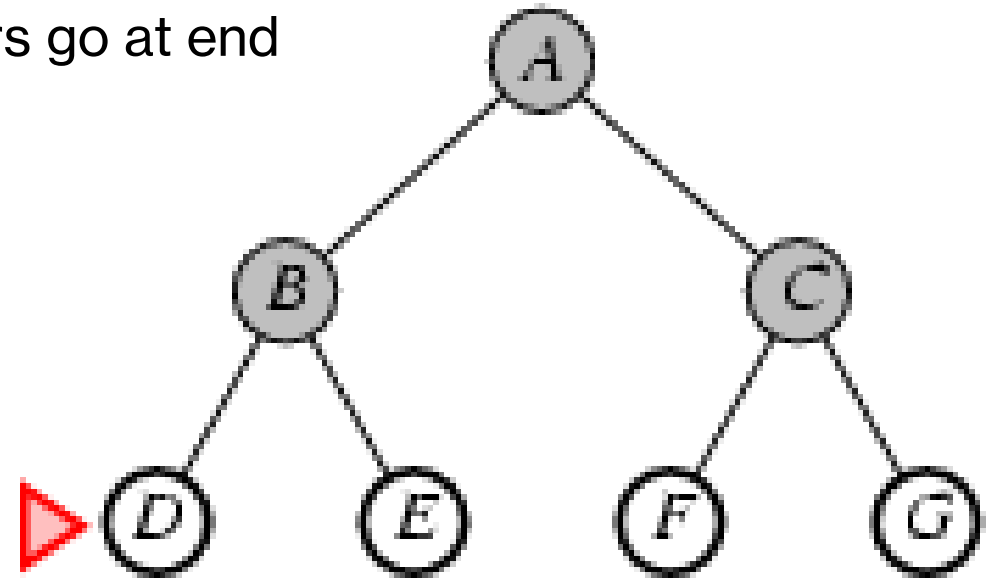
Is C a goal state?

Breadth-first search

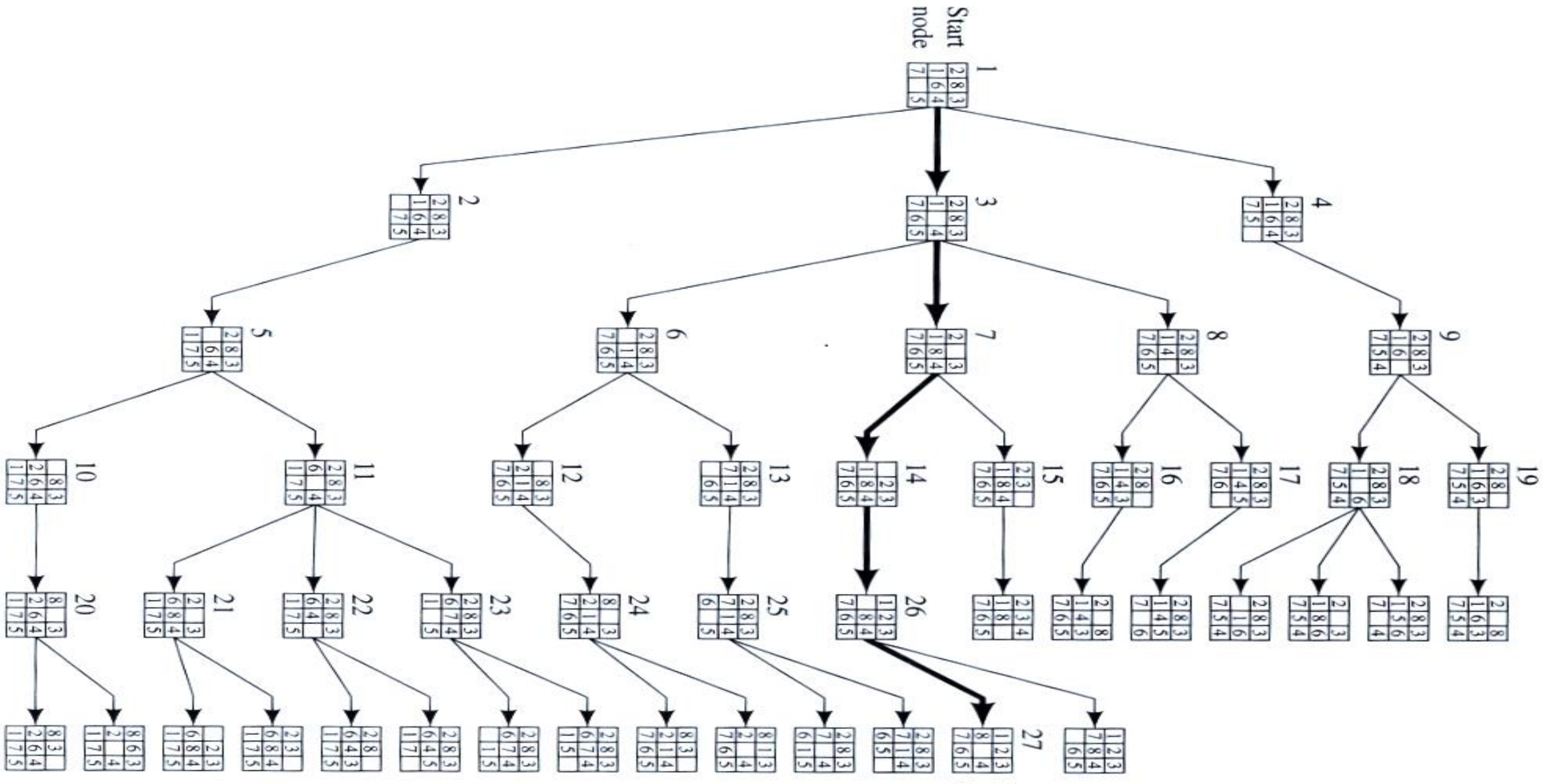
- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe=[D,E,F,G]

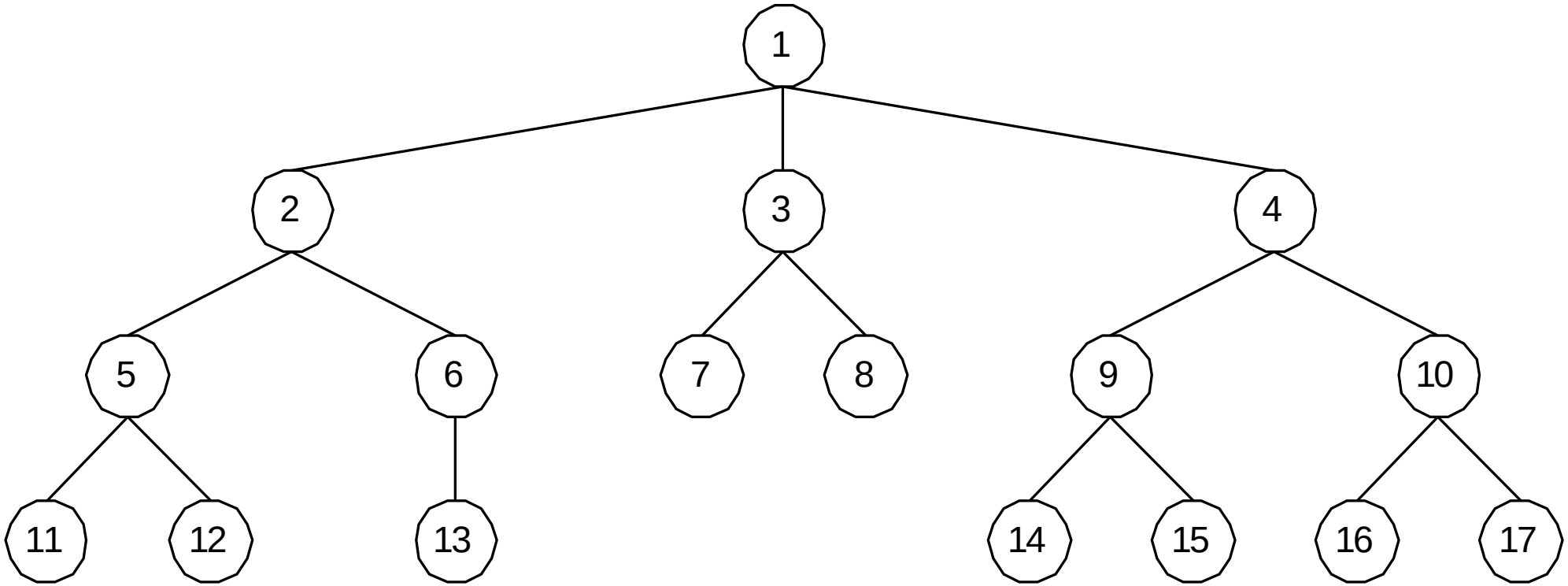
Is D a goal state?



Example BFS


Start node
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

Example



```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    frontier ← a FIFO queue with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)   /* chooses the shallowest node in frontier */
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
                frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

Measuring problem-solving performance

Completeness: Is the algorithm guaranteed to find a solution when there is one?

Optimality: Does the strategy find the optimal solution?

Time complexity: How long does it take to find a solution?

Space complexity: How much memory is needed to perform the search?

Complexity

- Complexity is expressed in terms of three quantities：
  - b, the **branching factor** or maximum number of successors of any node;
  - d, the **depth** of the shallowest goal node (i.e., the number of steps along the path from the root);
  - m, the maximum length of any path in the state space.

Properties of breadth-first search

Completeness: Yes it always reaches goal (if b is finite)

Optimality: Yes (if we guarantee that deeper solutions are less optimal).

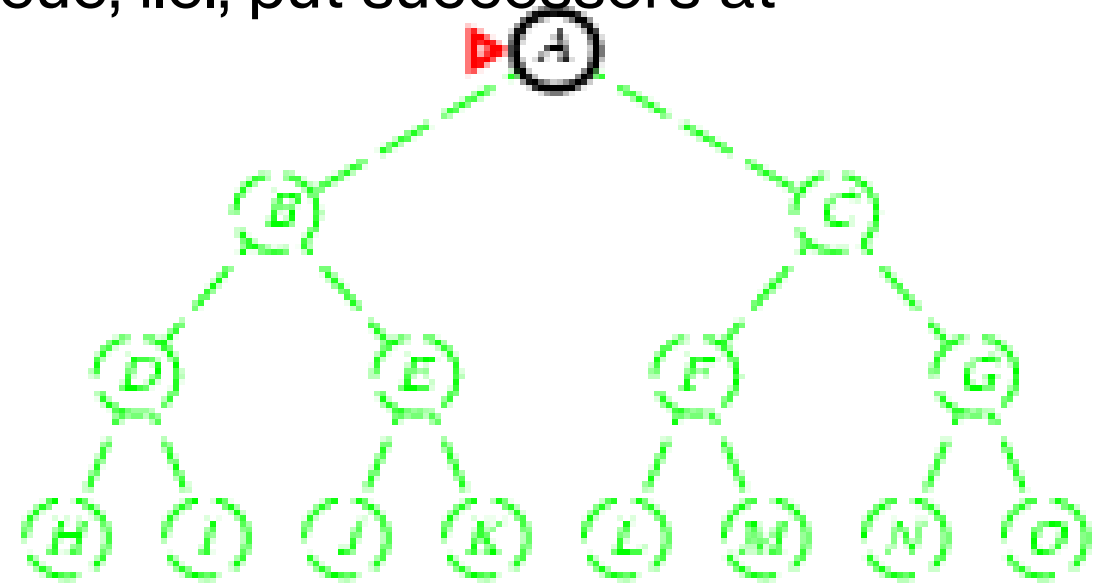
Time complexity: $1+b+b^2+b^3+\ldots+b^d + (b^{d+1}-b)) = O(b^d)$
(this is the number of nodes we generate)

Space complexity: $O(b^d)$ (keeps every node in memory,
either in fringe or on a path to fringe).

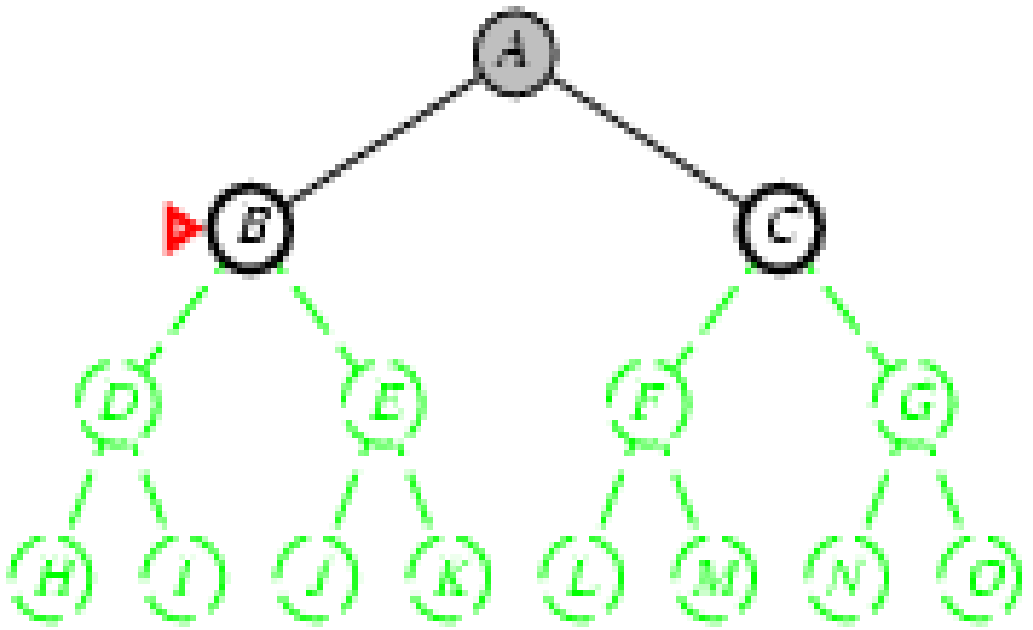
Depth-first search

- Expand *deepest* unexpanded node
- Implementation:
  - *fringe* = Last In First Out (LIFO) queue, i.e., put successors at front

Is A a goal state?



Depth-first search



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

    queue=[B,C]

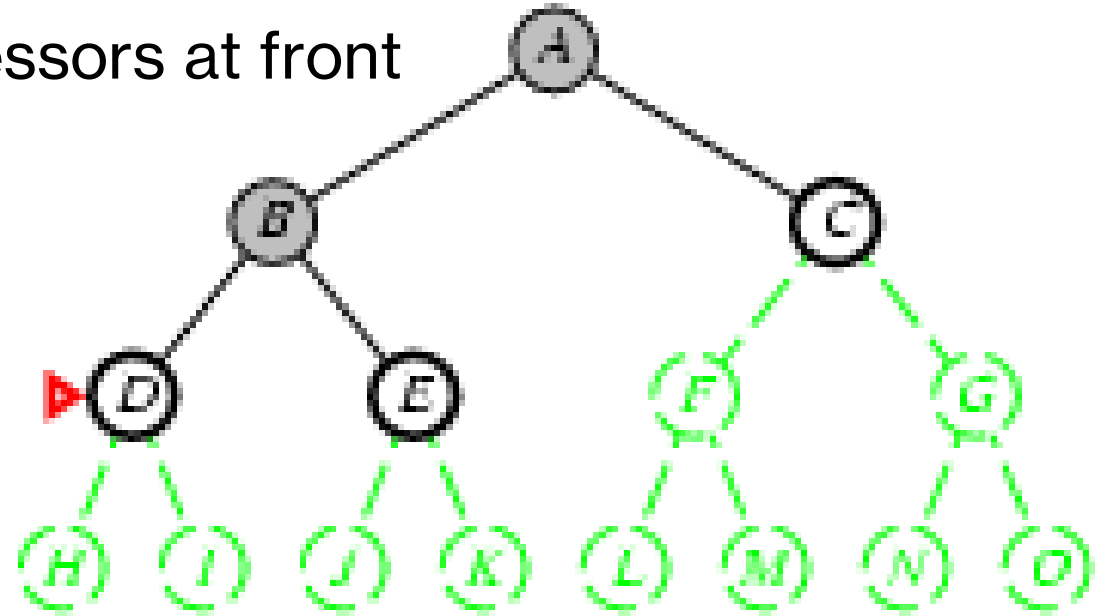
    Is B a goal state?

Depth-first search

- Expand the deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[D,E,C]

Is D = goal state?

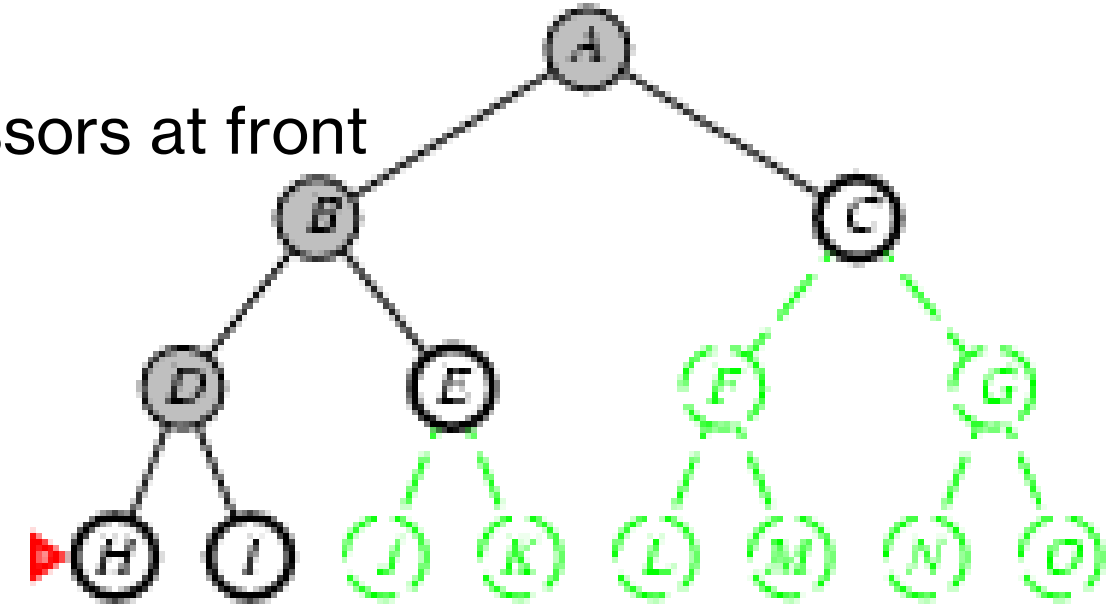


Depth-first search

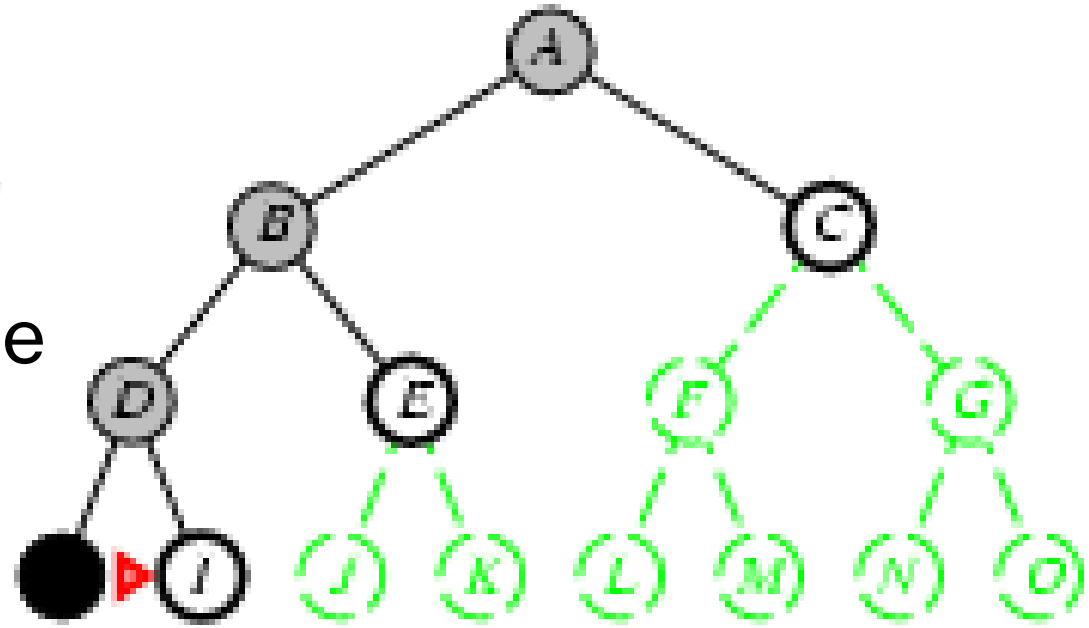
- Expand the deepest unexpanded node
-
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[H,I,E,C]

Is H = goal state?



Depth-first search



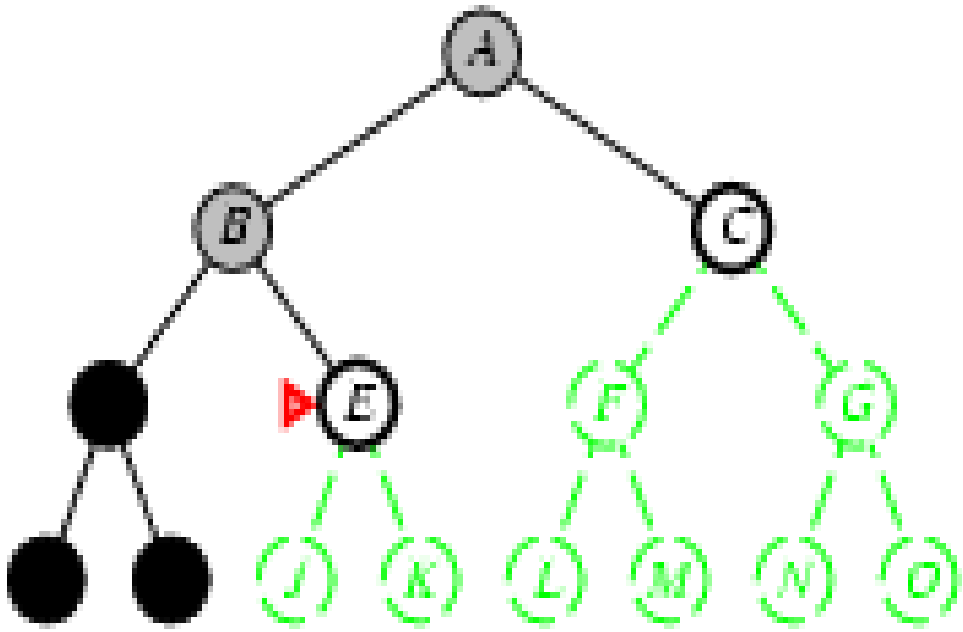
- Expand deepest unexpanded node
-
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[I,E,C]

Is I = goal state?

Depth-first search

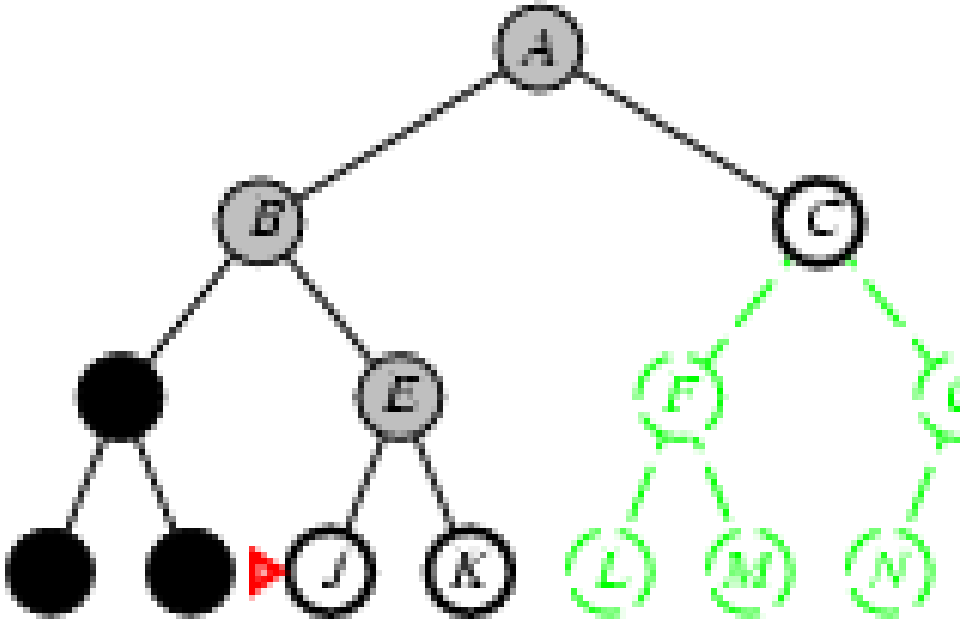
- Expand the deepest unexpanded nc
- Implementation:
  - *fringe* = LIFO queue, i.e., put successor



queue=[E,C]

Is E = goal state?

Depth-first search



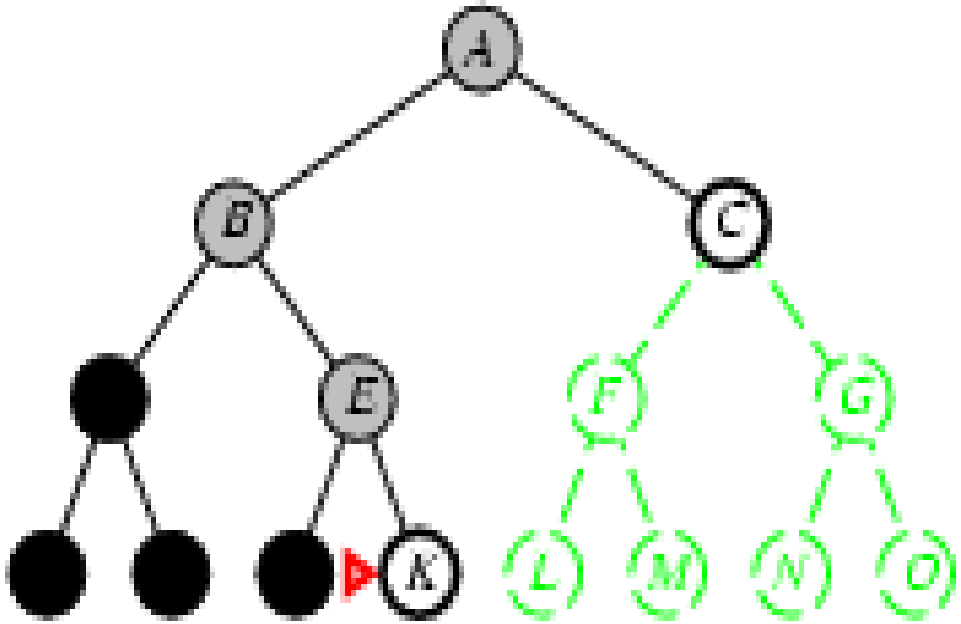
- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[J,K,C]

Is J = goal state?

Depth-first search

- Expand the deepest unexpanded noc
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors



queue=[K,C]

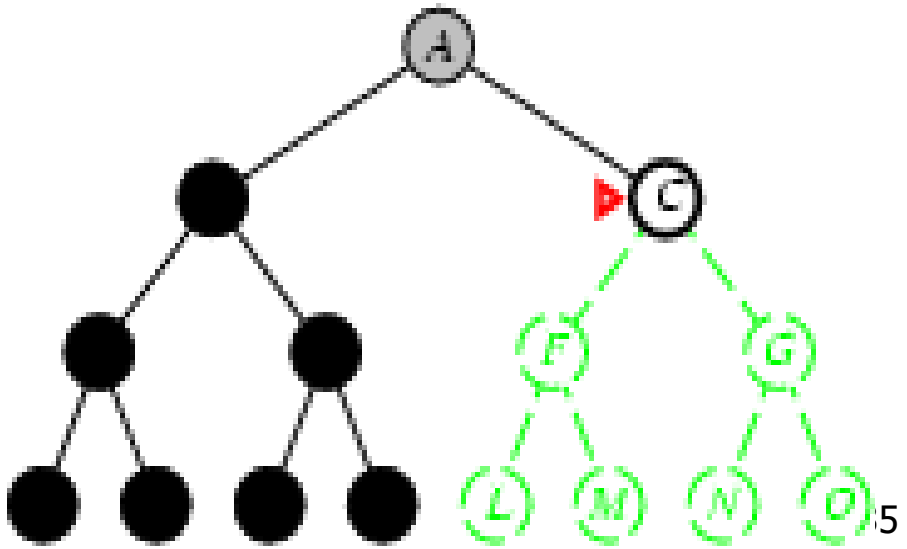
Is K = goal state?

Depth-first search

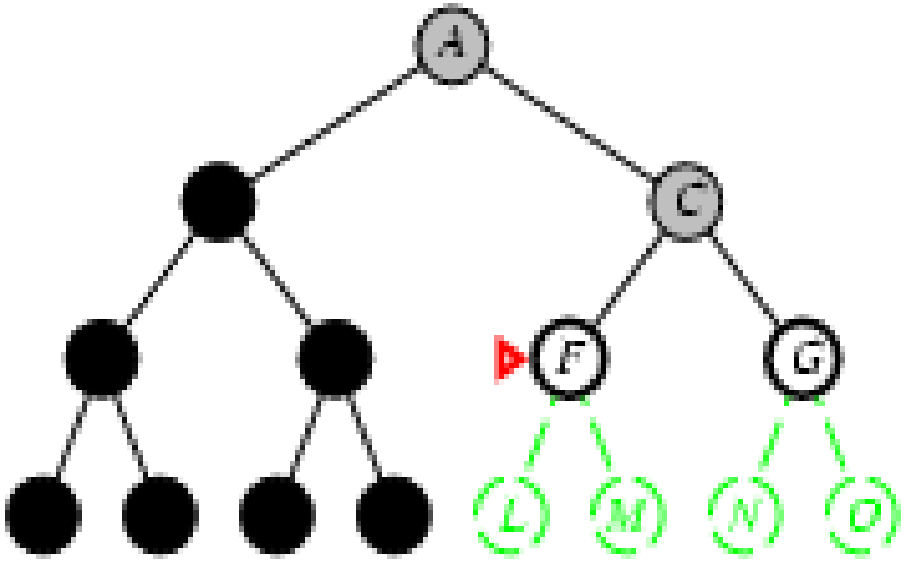
- Expand the deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[C]

Is C = goal state?



Depth-first search



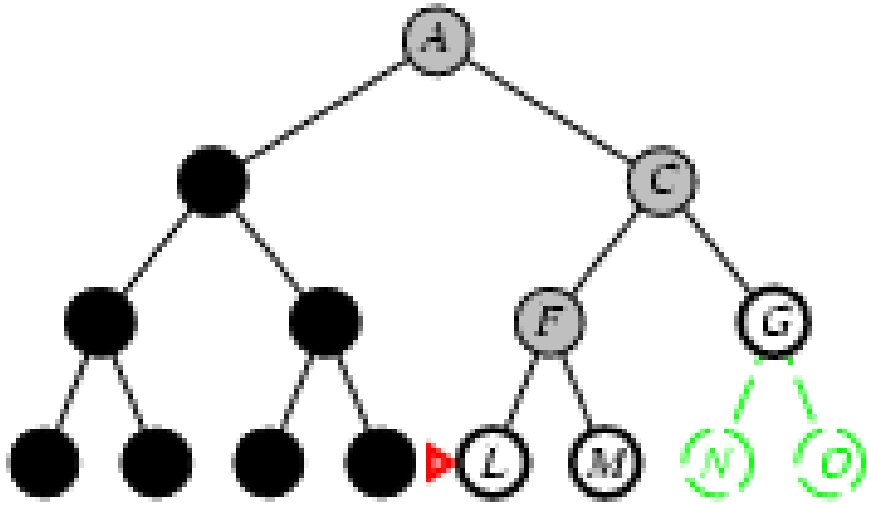
- Expand the deepest unexpanded no
-
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front
  -

    queue=[F,G]

    Is F = goal state?

Depth-first search

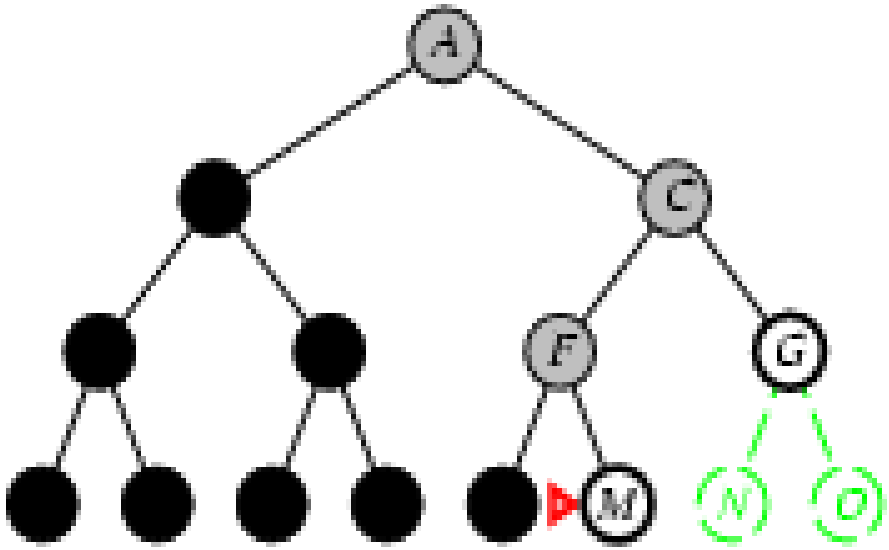
- Expand the deepest unexpanded noc
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors



queue=[L,M,G]

Is L = goal state?

Depth-first search

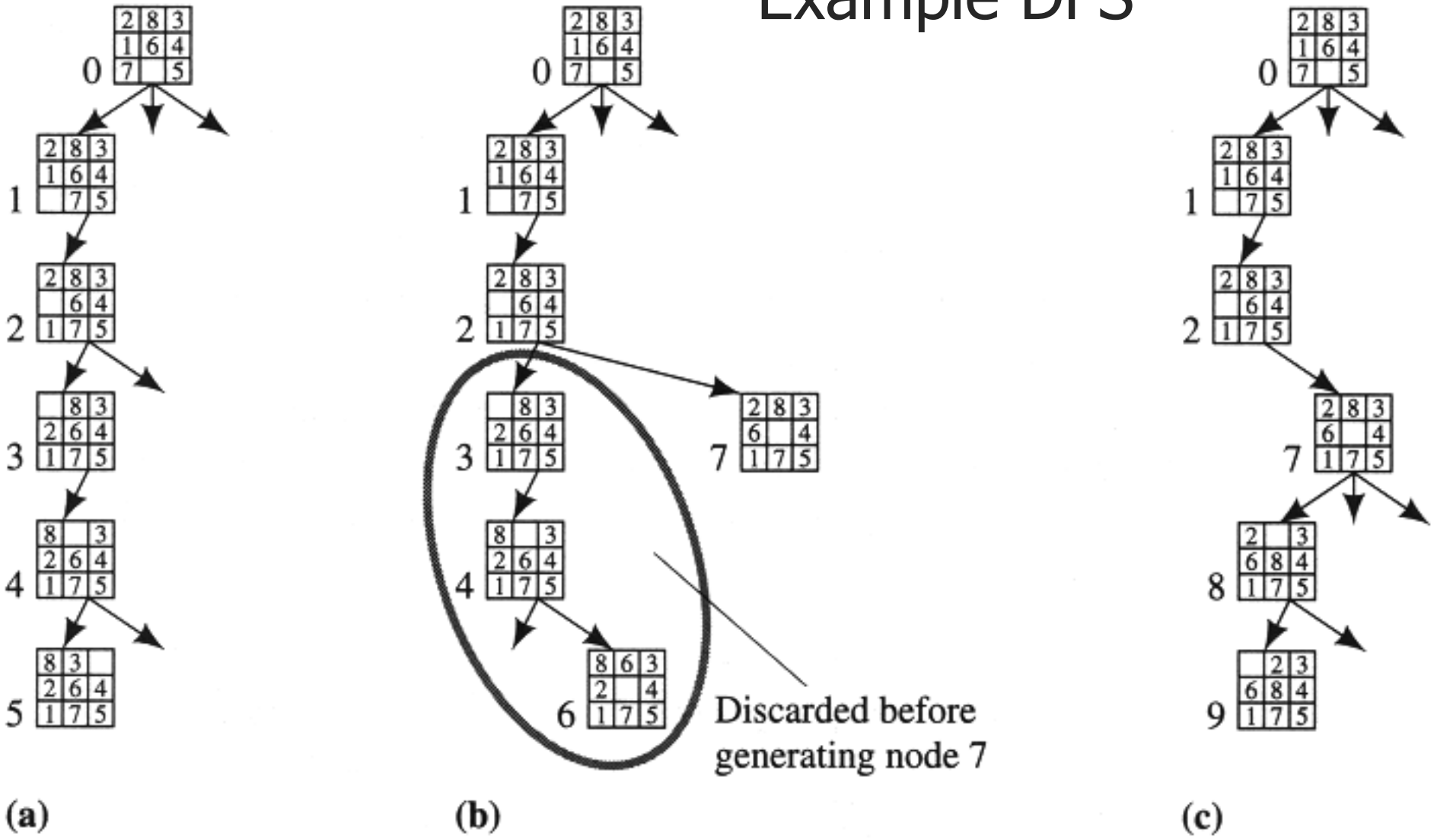


- Expand the deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[M,G]

Is M = goal state?

Example DFS



Generation of the First Few Nodes in a Depth-First Search

Properties of breadth-first search

Completeness: ? No: fails in infinite-depth spaces

Optimality: No (It may find a non-optimal goal first)

Time complexity: $O(b^m)$ with m=maximum depth terrible if *m* is much larger than *d* but if solutions are dense, may be much faster than breadth-first

Space complexity: *O(bm)*, i.e., linear space! (we only need to remember a single path + expanded unexplored nodes)


AI
THANKYOU