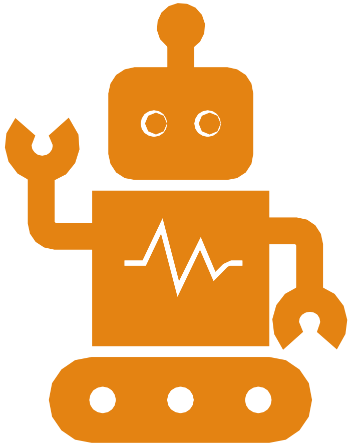


CS341
Artificial Intelligence

Lecture 4

DR. HEBA MOHSEN

Problem Solving

Problems generally represented as graphs (State space)

Problem solving ~ searching a graph

State Space

State space = Directed graph

Nodes ~ Problem situations

Arcs ~ Actions, legal moves

Problem = (State space, Start, Goal condition)

Note: several nodes may satisfy goal condition

Solving a problem ~ Finding a path

Problem solving ~ Graph search

Problem solution ~ Path from start to a goal node

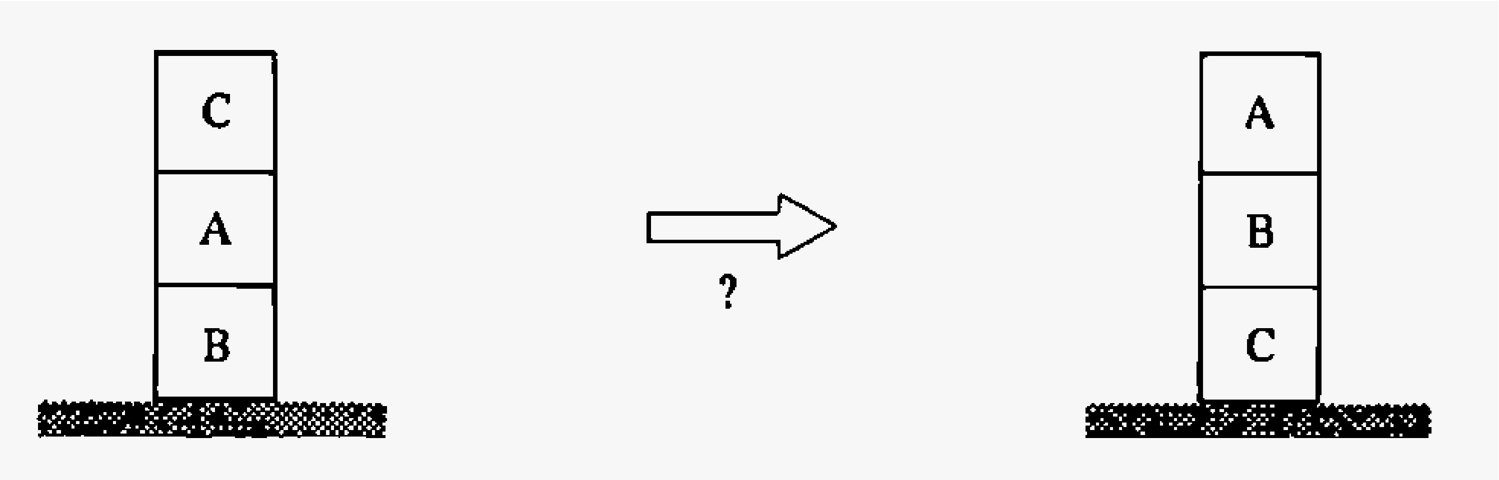
Examples of representing problems in state space

Blocks world planning

8-puzzle

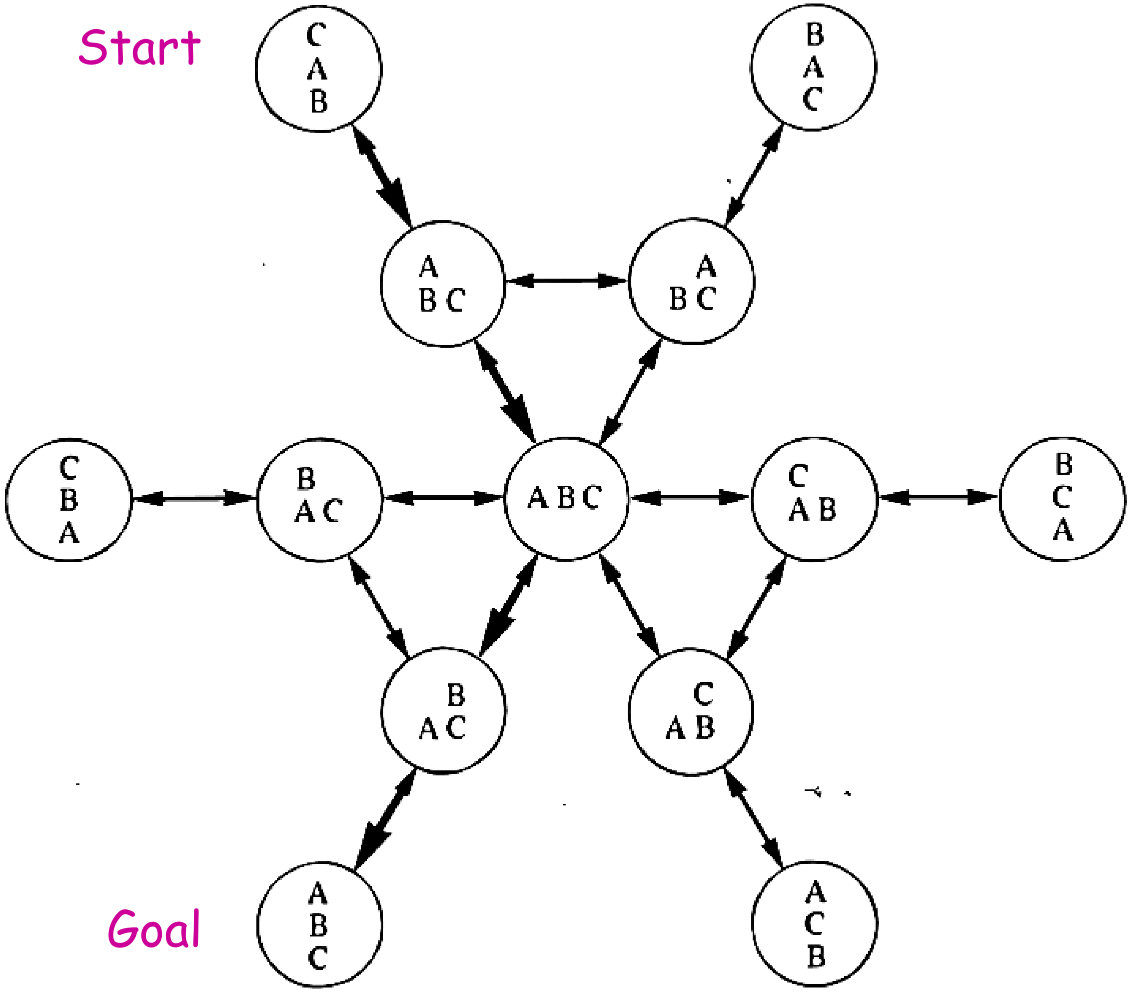
Travelling salesman

A problem from blocks world

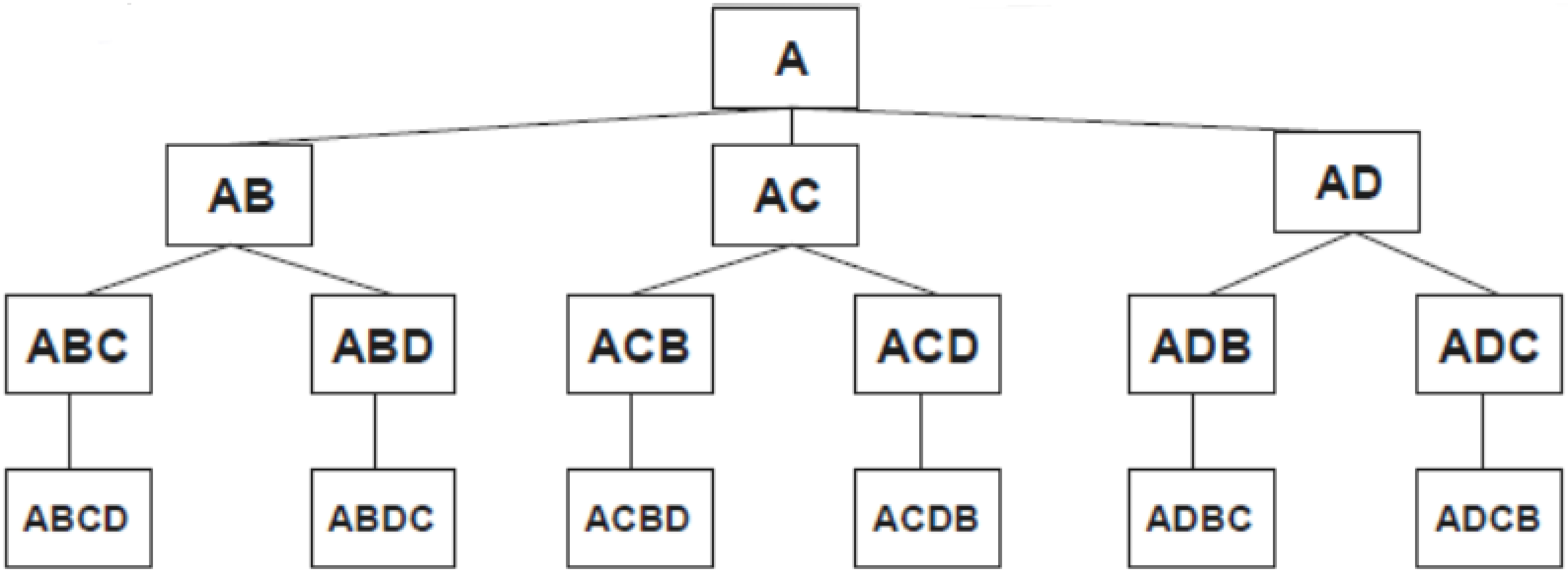


Find a sequence of robot moves to re-arrange blocks

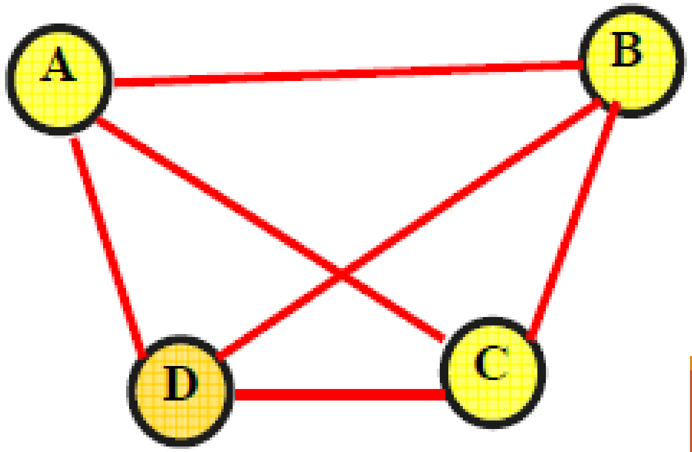
Blocks World State Space



Travel salesman problem state space



N= 4 cites, then different paths = 3!=6



In order to solve the problem:

- Define the problem state space including the start and the goal states and a set of operators for moving in that space.

- The problem can then be solved by searching for a path through the space from the initial state to the goal state, So the process of search is fundamental to the problem-solving process.

Search strategies

- A search strategy is defined by picking the order of node expansion
- To measure the strategies performance they are evaluated by:
  - Completeness: Does it always find a solution if one exists?
  - Time Complexity: How long does it take to find a solution? (number of nodes generated)
  - Space Complexity: How much memory is needed? (maximum number of nodes in memory)
  - Optimality: Does it always find a least-cost solution?

Search strategies

Blind (Uninformed) strategies:

Systematically search complete graph, unguided

- Breadth-first search.
- Depth-first search.

Heuristic (Informed) strategies:

Use problem specific information to guide search in promising directions

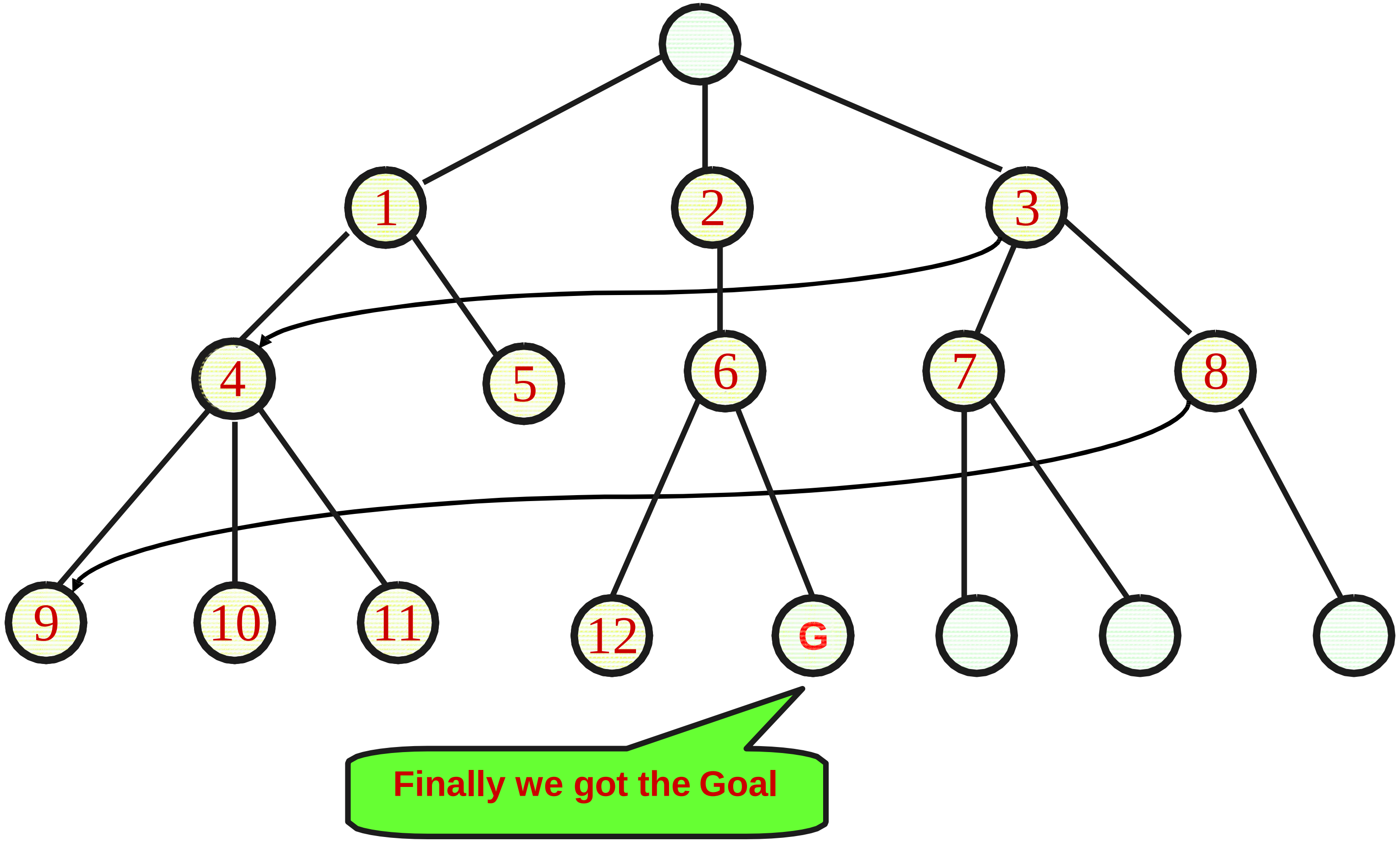
→ What is "promising"? Domain specific knowledge

Uninformed search strategies

Uninformed:

→While searching you have no clue whether one non-goal state is better than any other. Your search is blind. You don't know if your current exploration is likely to be fruitful.

Breadth-first search



Breadth-first search

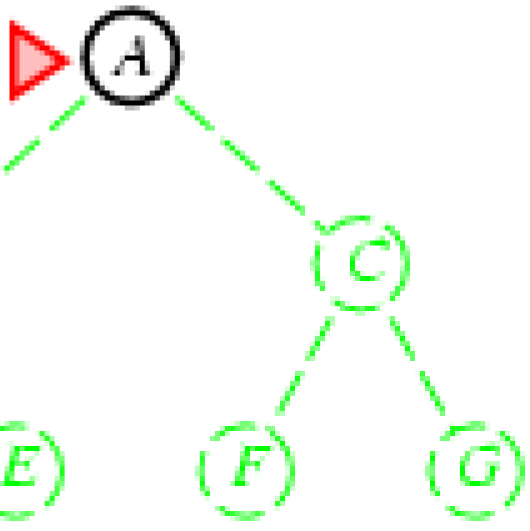
Expand shallowest unexpanded node

Fringe: nodes waiting in a queue to be explored

Implementation:

- *Fringe (or the* OPEN list) is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.

Is A a goal state?



Breadth-first search

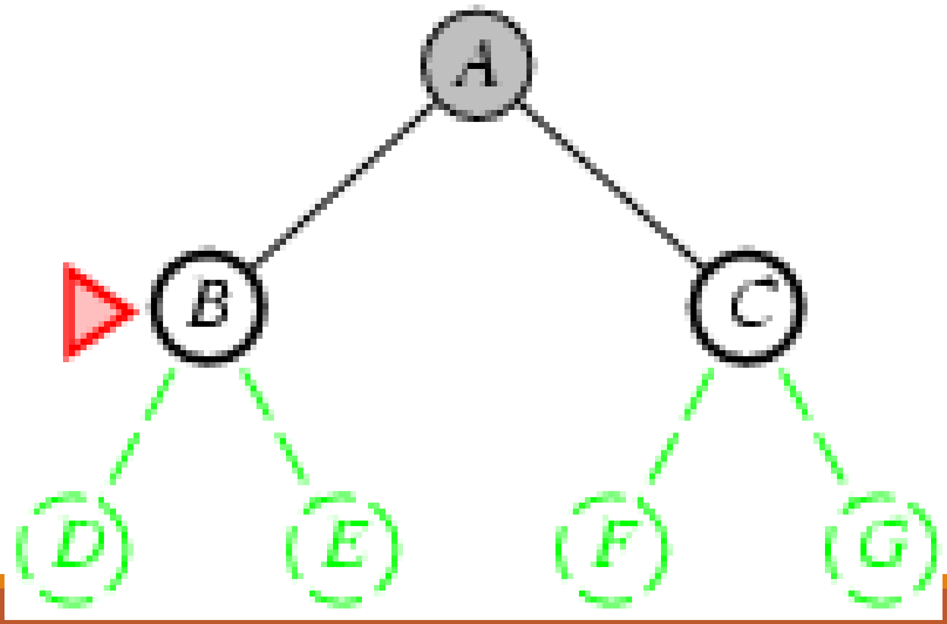
Expand shallowest unexpanded node

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

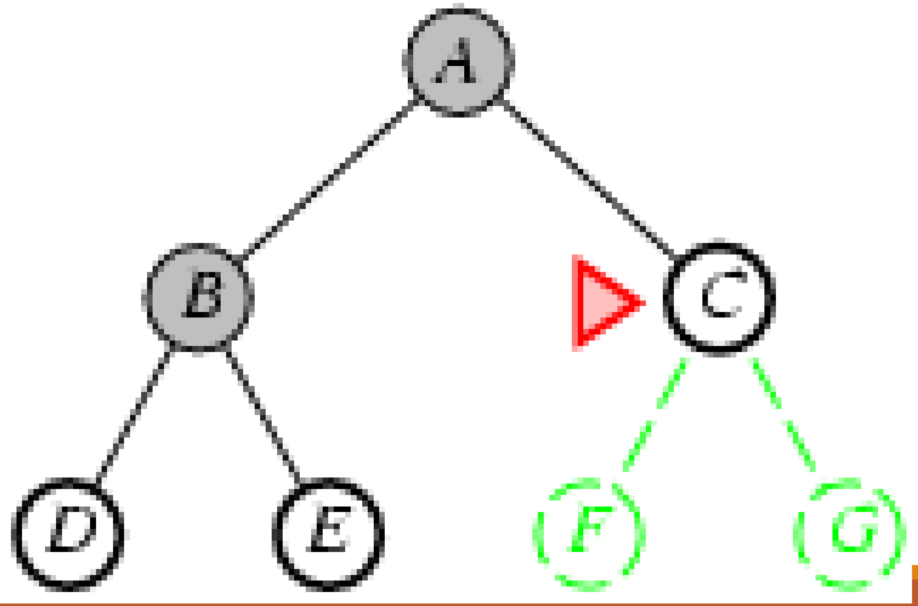
Expand shallowest unexpanded node

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

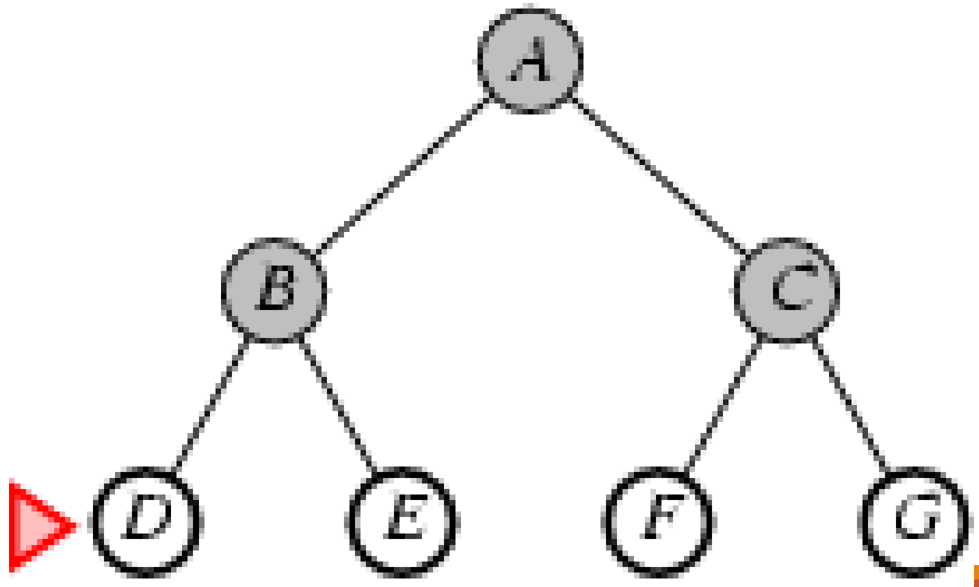
Expand shallowest unexpanded node

Implementation:

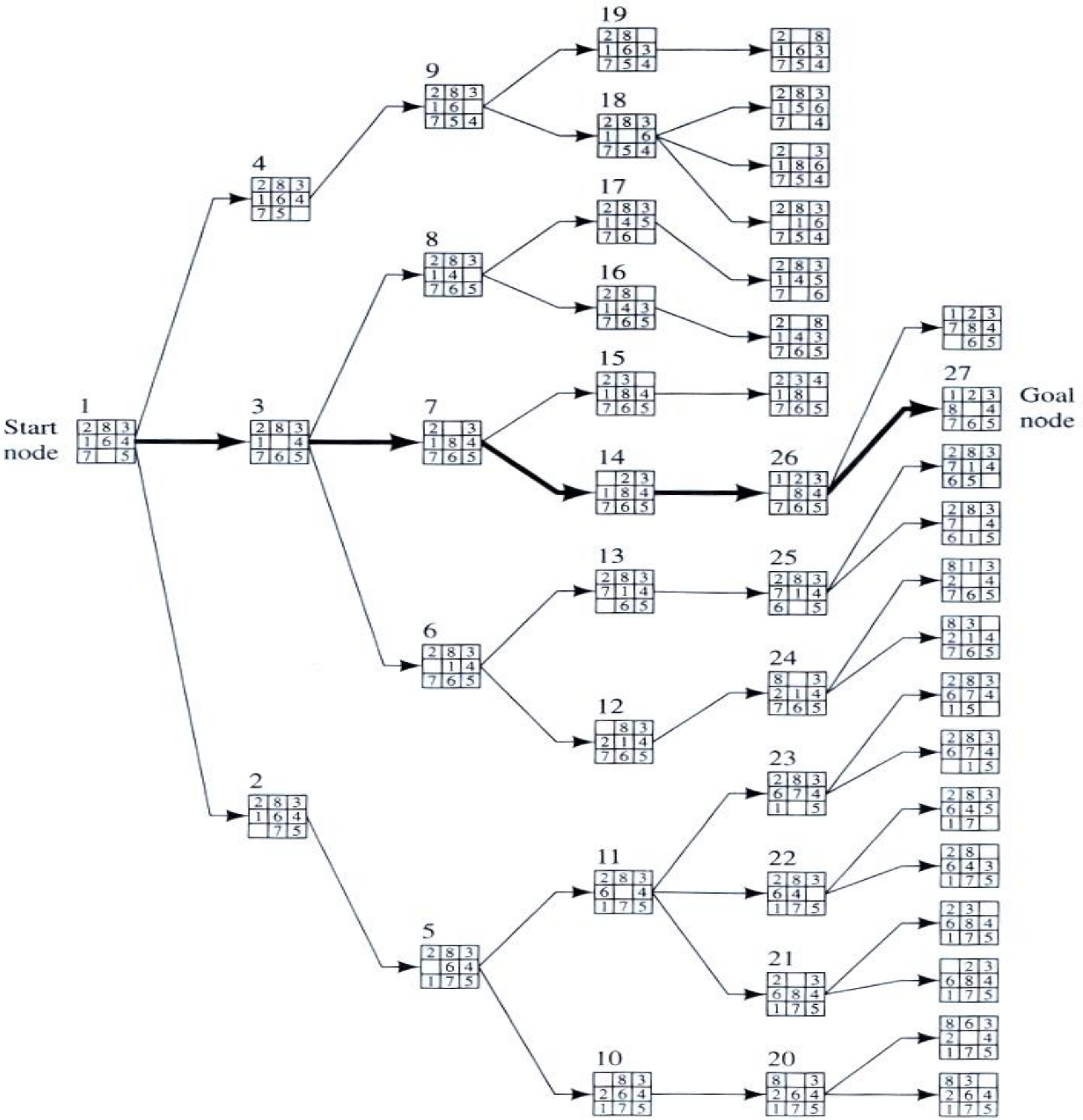
- *fringe* is a FIFO queue, i.e., new successors go at end

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

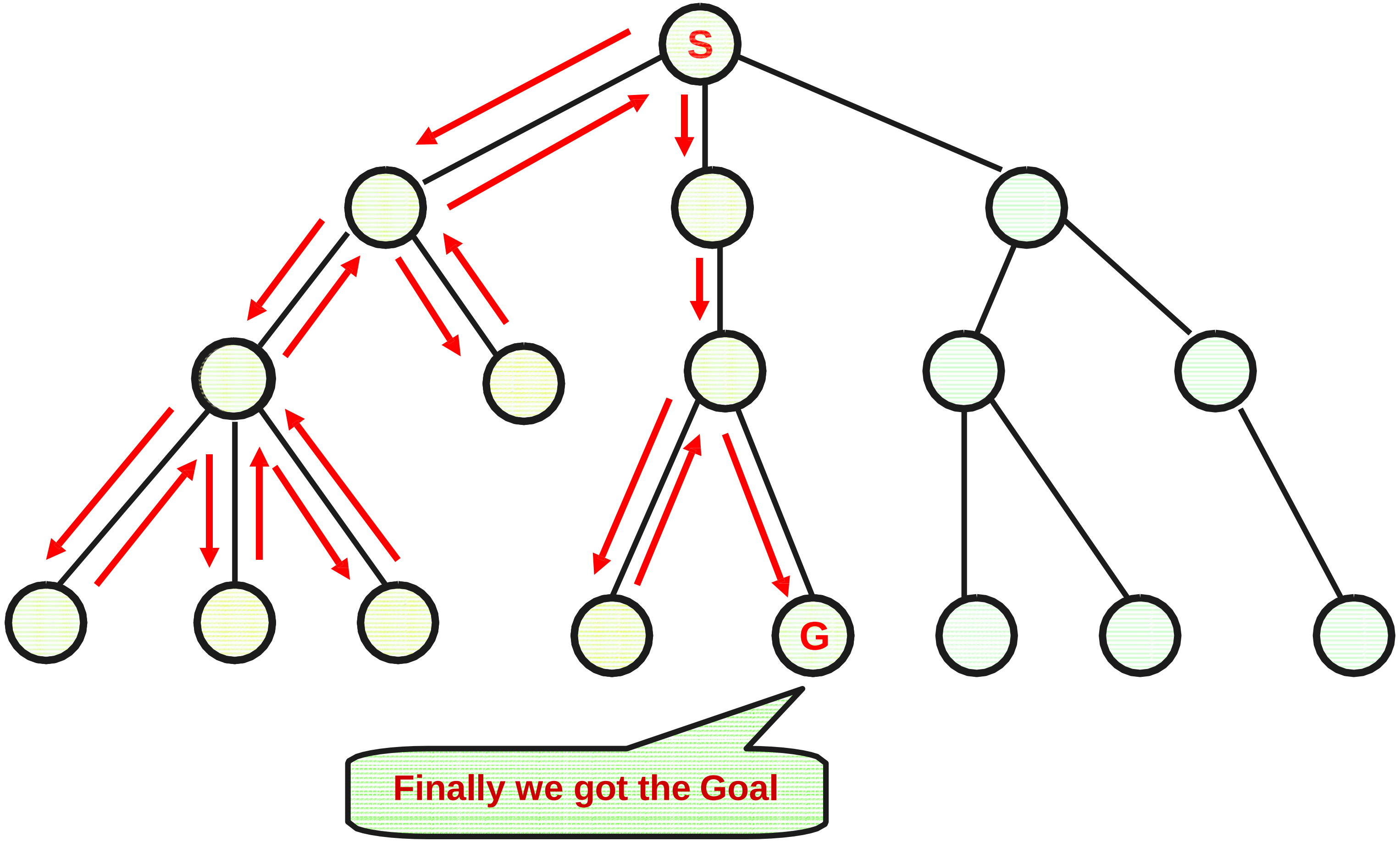
Is D a goal state?



Example BFS


Start node
Goal 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

Depth-first search



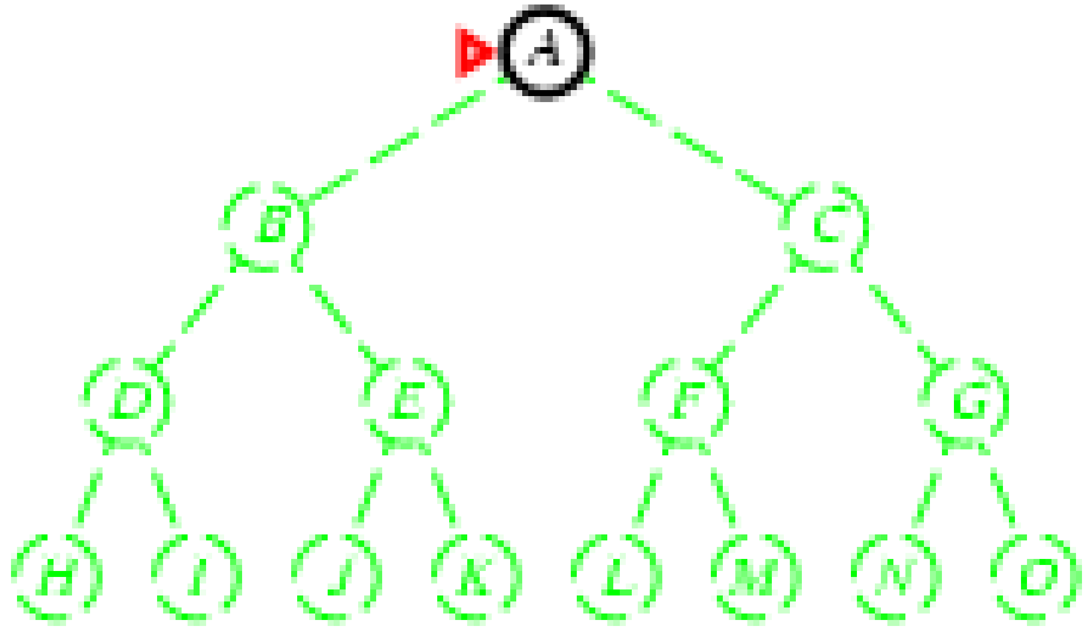
Depth-first search

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

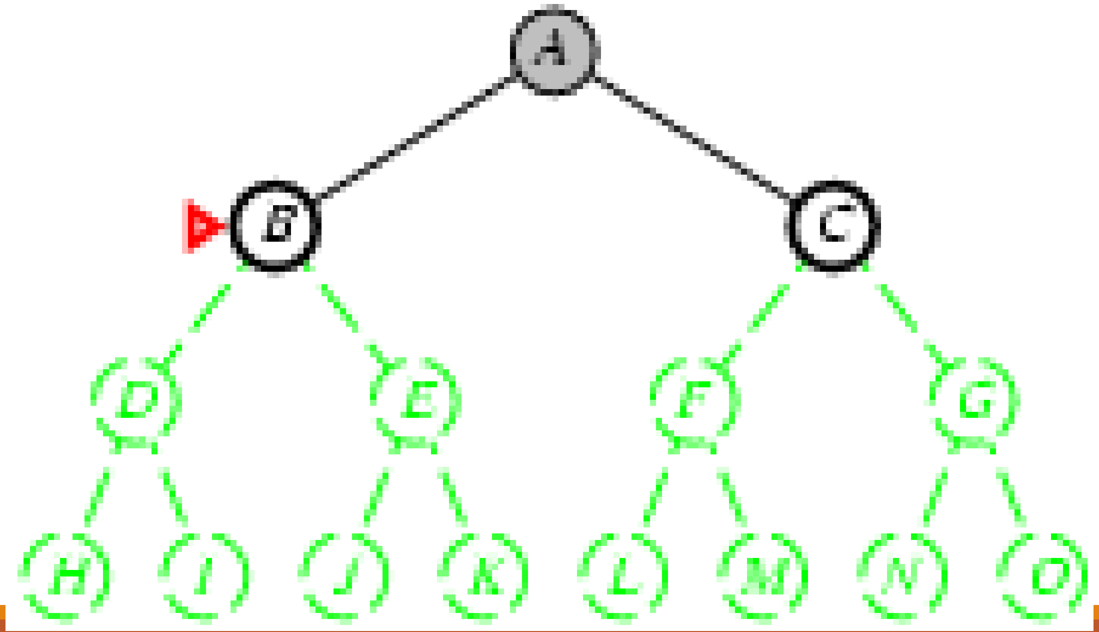
Expand deepest unexpanded node

Implementation:

- fringe = LIFO queue, i.e., put successors at front

queue=[B,C]

Is B a goal state?



Depth-first search

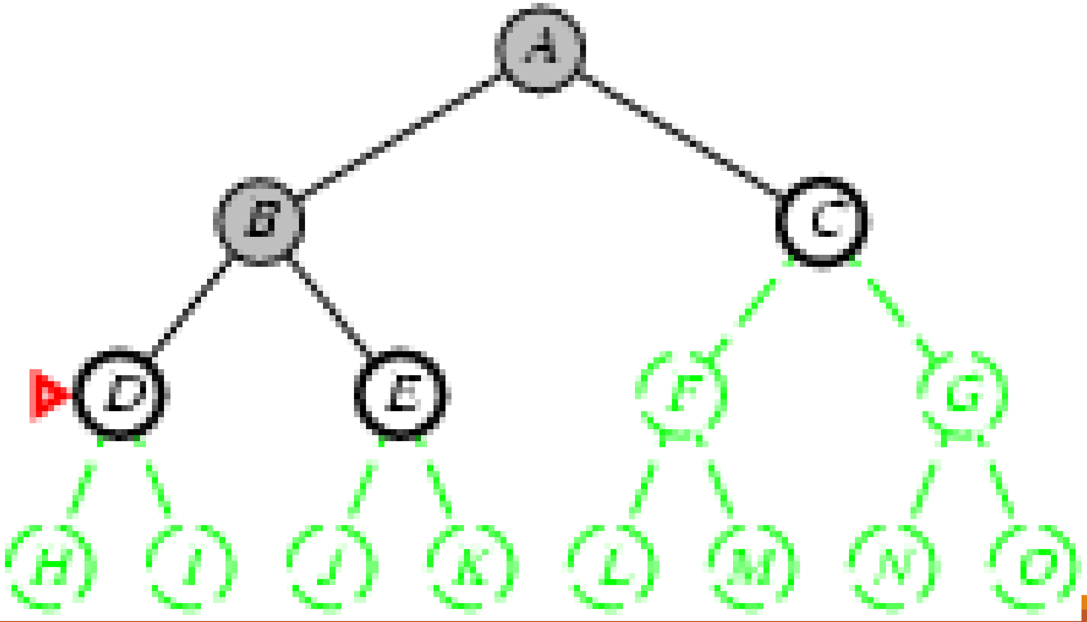
Expand deepest unexpanded node

Implementation:

- fringe = LIFO queue, i.e., put successors at front

queue=[D,E,C]

Is D = goal state?



Depth-first search

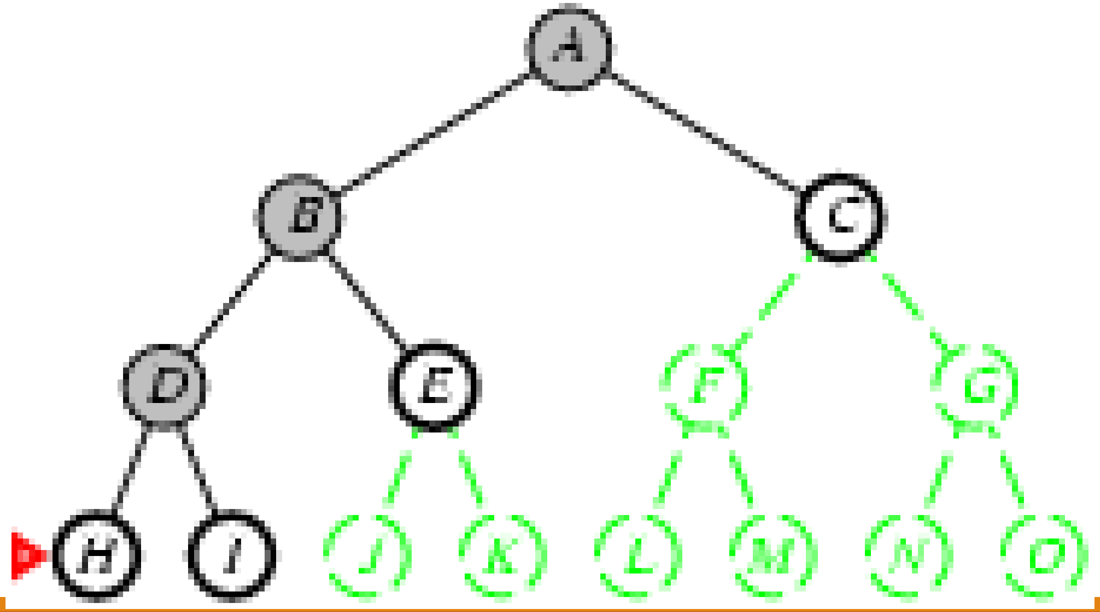
Expand deepest unexpanded node

Implementation:

- fringe = LIFO queue, i.e., put successors at front

queue=[H,I,E,C]

Is H = goal state?



Depth-first search

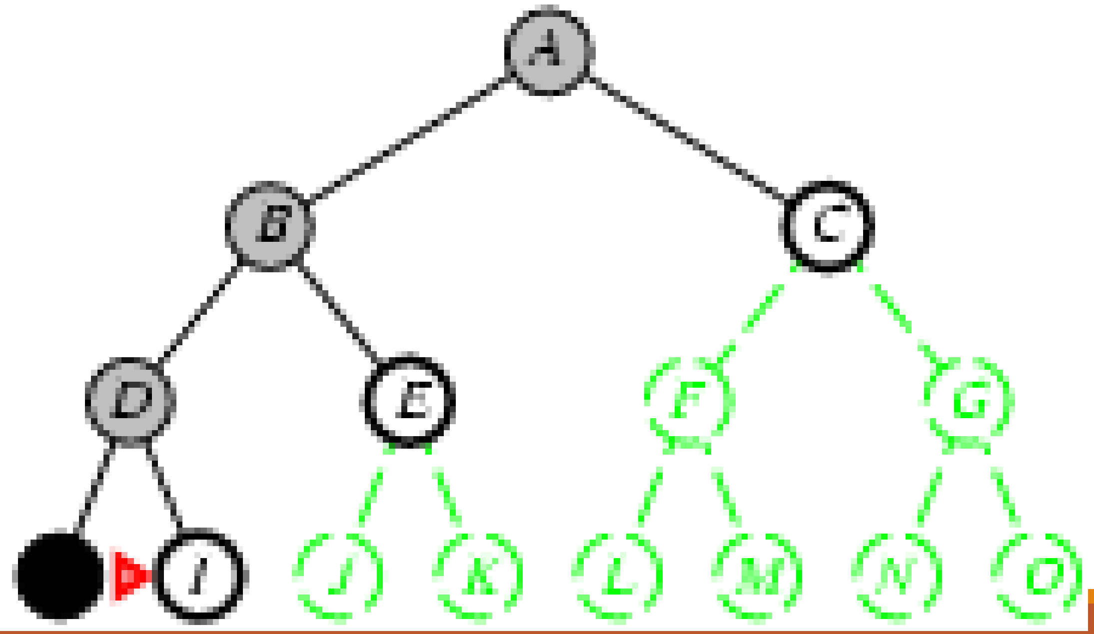
Expand deepest unexpanded node

Implementation:

- fringe = LIFO queue, i.e., put successors at front

queue=[I,E,C]

Is I = goal state?



Depth-first search

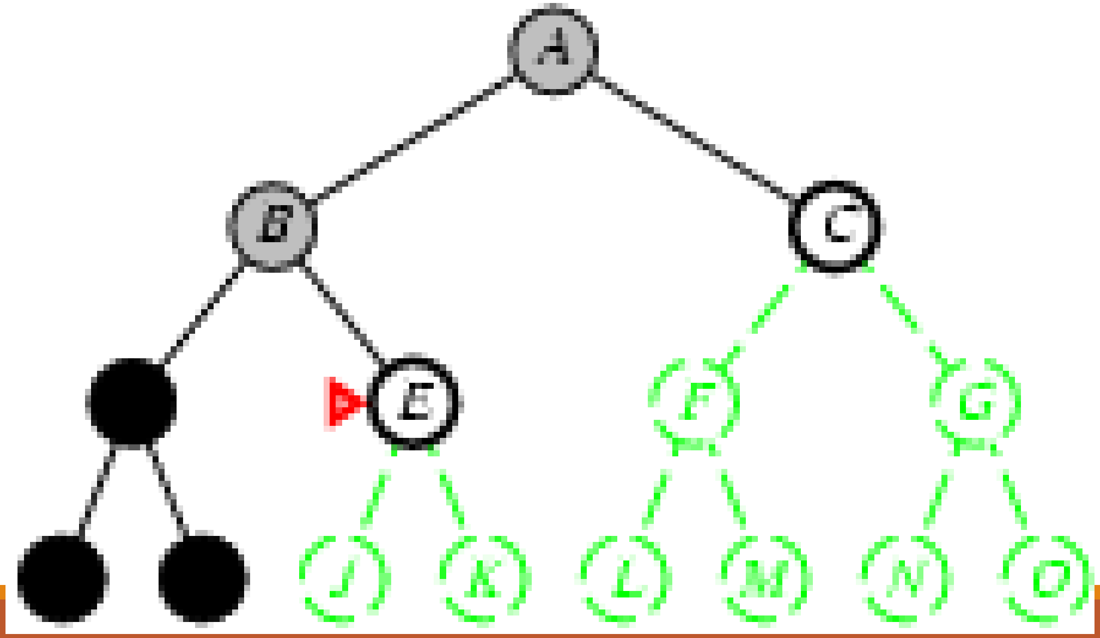
Expand deepest unexpanded node

Implementation:

- fringe = LIFO queue, i.e., put successors at front

queue=[E,C]

Is E = goal state?



Depth-first search

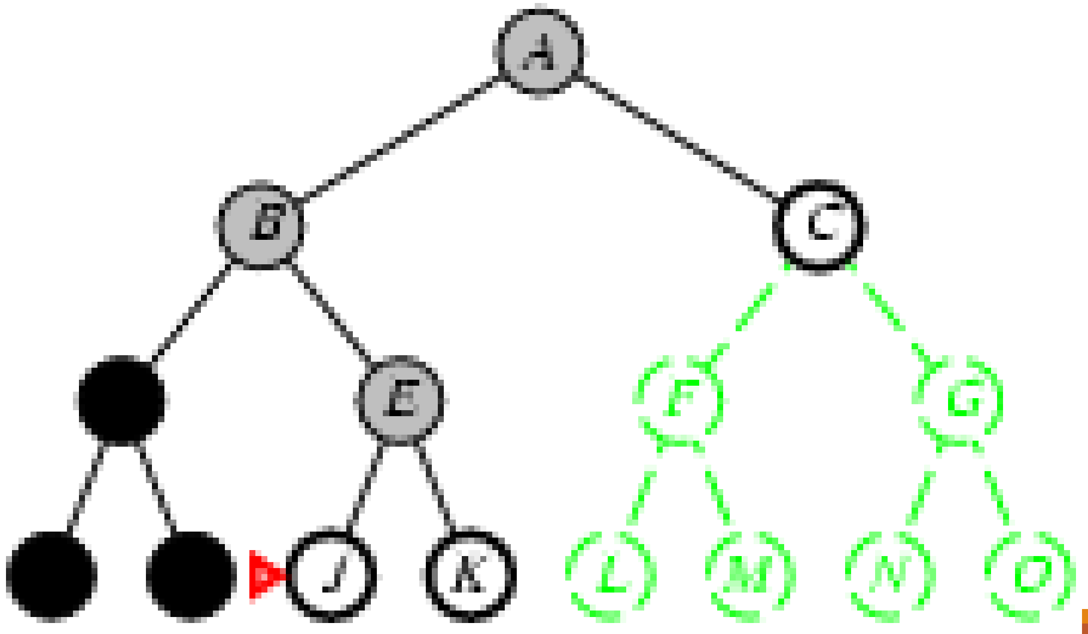
Expand deepest unexpanded node

Implementation:

- fringe = LIFO queue, i.e., put successors at front

queue=[J,K,C]

Is J = goal state?



Depth-first search

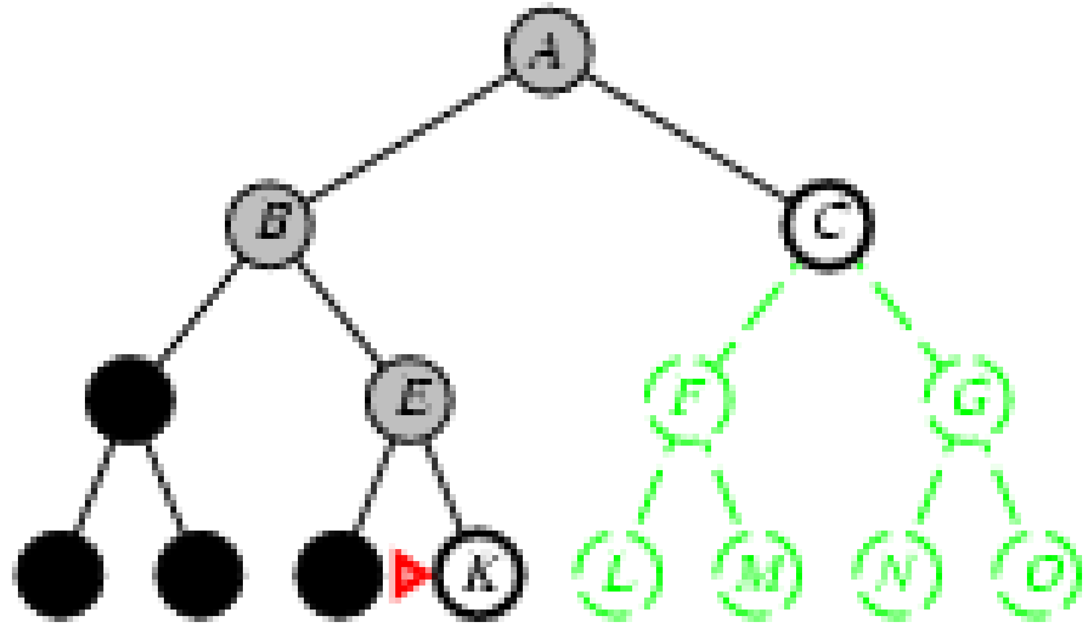
Expand deepest unexpanded node

Implementation:

- fringe = LIFO queue, i.e., put successors at front

queue=[K,C]

Is K = goal state?



Depth-first search

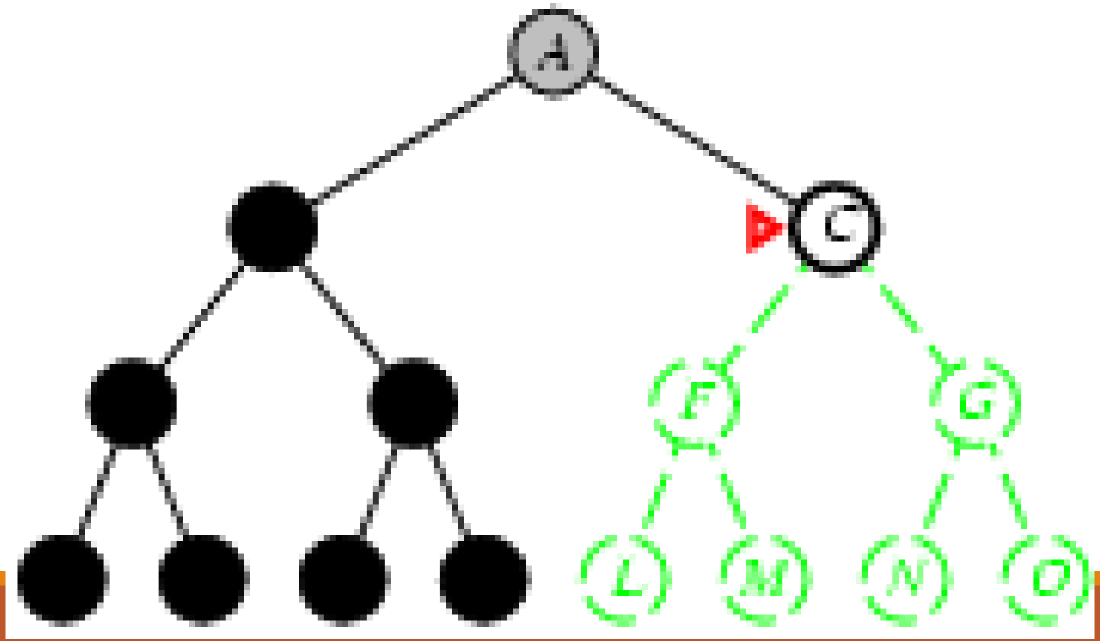
Expand deepest unexpanded node

Implementation:

- fringe = LIFO queue, i.e., put successors at front

queue=[C]

Is C = goal state?



Depth-first search

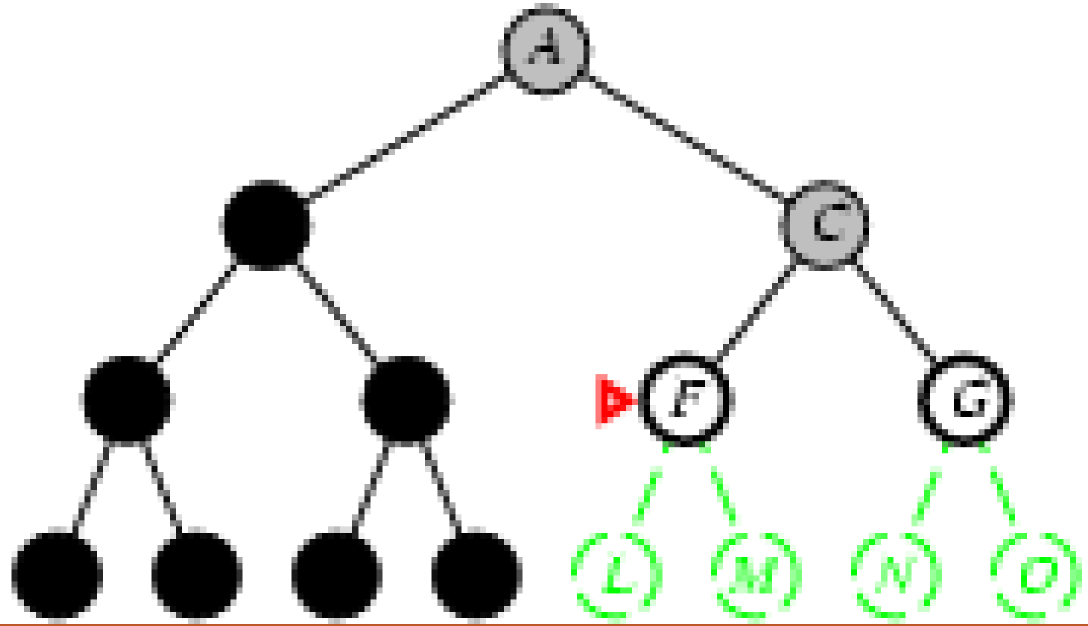
Expand deepest unexpanded node

Implementation:

- fringe = LIFO queue, i.e., put successors at front

queue=[F,G]

Is F = goal state?



Depth-first search

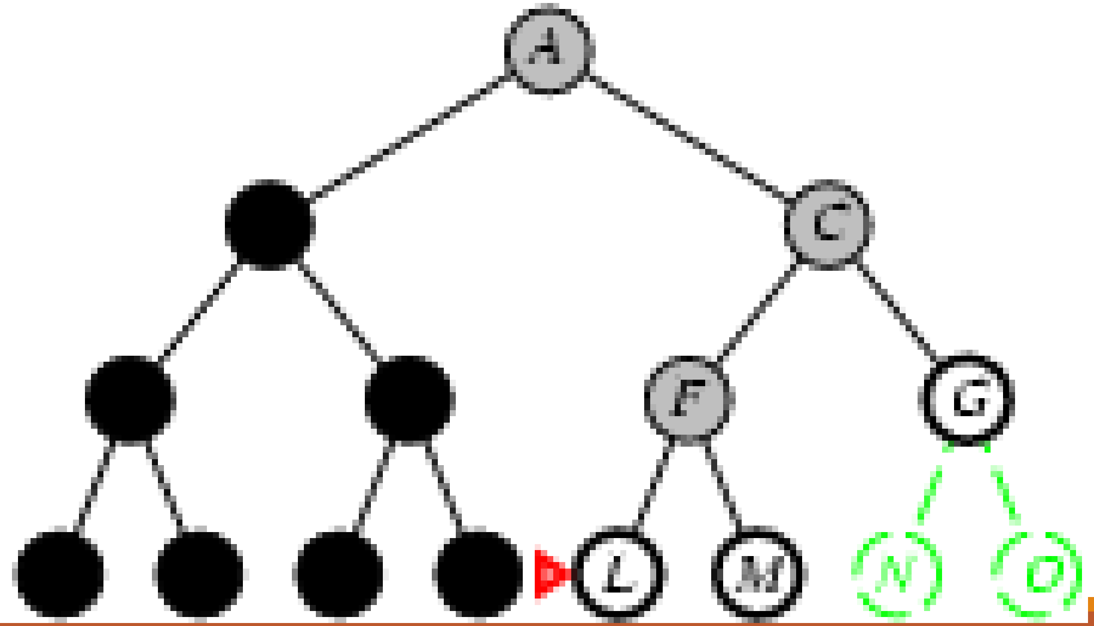
Expand deepest unexpanded node

Implementation:

- fringe = LIFO queue, i.e., put successors at front

queue=[L,M,G]

Is L = goal state?



Depth-first search

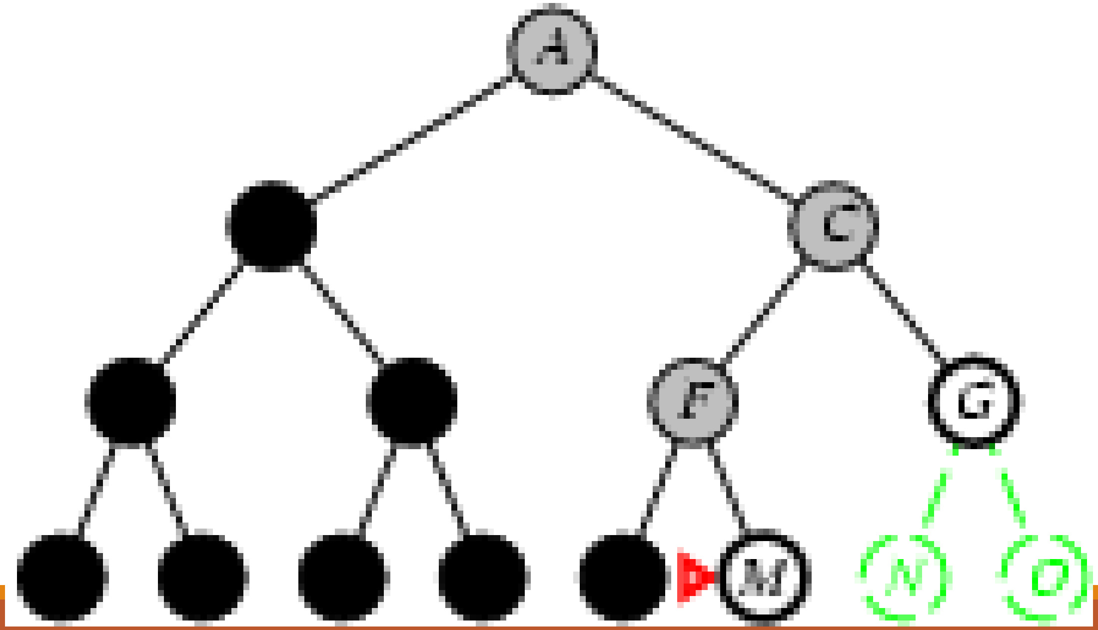
Expand deepest unexpanded node

Implementation:

- fringe = LIFO queue, i.e., put successors at front

queue=[M,G]

Is M = goal state?



Comparison between Depth-first and Breadth-first

Depth-first search	Breadth-first search
▪ It requires less memory since only the nodes on the current path are stored. ▪ By chance it may find a solution without examining much of the search space. ▪ It may follow a wrong path for a very long time. ▪ It may find a long path solution	▪ All the tree that so far has been generated must be stored. ▪ All the tree must be examined to level (n) before any node on level (n+1). ▪ It will not follow a wrong path for a long time. ▪ If there are multiple solutions then the minimal solution will be found so the longer paths never explored before shorter one.