CS 461
ARTIFICIAL INTELLIGENCE

Lecture # 03
March 10, 2021
SPRING 2021
FAST – NUCES, CFD Campus

Dr. Rabia Maqsood
rabia.maqsood@nu.edu.pk

Today’s Topics

- Problem solving agents
  - *Introduction*
  - *Understanding search problems*
- State space graph
- Search strategies
  - *Uninformed search algorithms*
    - Depth First Search
    - Breadth First Search

Search

- Finding the sequence of actions to achieve some goal(s) given a start state is called search, and the agent which does this is called the ***problem-solving agent***.

Search

- Often, we are not given an algorithm to solve a problem, but only a specification of what a solution is, and

*we have to **search** for a solution.*

- A typical approach:
  - *Enumerate a set of potential partial solutions*
  - *Check to see if they are solutions or could lead to one*

A Simple Search Agent

Deterministic, goal-driven agent

- Agent is in a **start state**
- Agent is given a **goal** (subset of possible states)
- Environment changes only when the agent acts
- Agent perfectly knows:
  - *actions that can be applied in any given state*
  - *the **state** it is going to end up in when an action is applied in each state*

- The sequence of actions (and appropriate ordering) taking the agent from the start state to a goal state is the solution

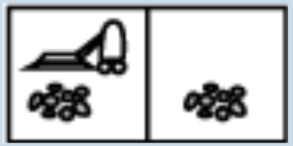
Definition of a search problem

- **Initial state(s)**
- Set of **actions** (operators) available to the agent
- An **action function *(or transition model)*** that, given a state and an action, returns a new state
- **Goal state(s)**
- By combining initial state, actions and transition model we get, **state space**
  - ***State space** is a set of states that will be searched for a path from initial state to goal, given the available actions*
  - ***states are nodes** and **actions are links** between them.*
  - *Not necessarily given explicitly (state space might be infinite)*
- **Path Cost** (we ignore this for now)

Example: vacuum world

- States
  - *Two rooms: r1, r2*
  - *Each room can be either dirty or not*
  - *Vacuum agent can be in either in r1 or r2*
  - *How many total states?*

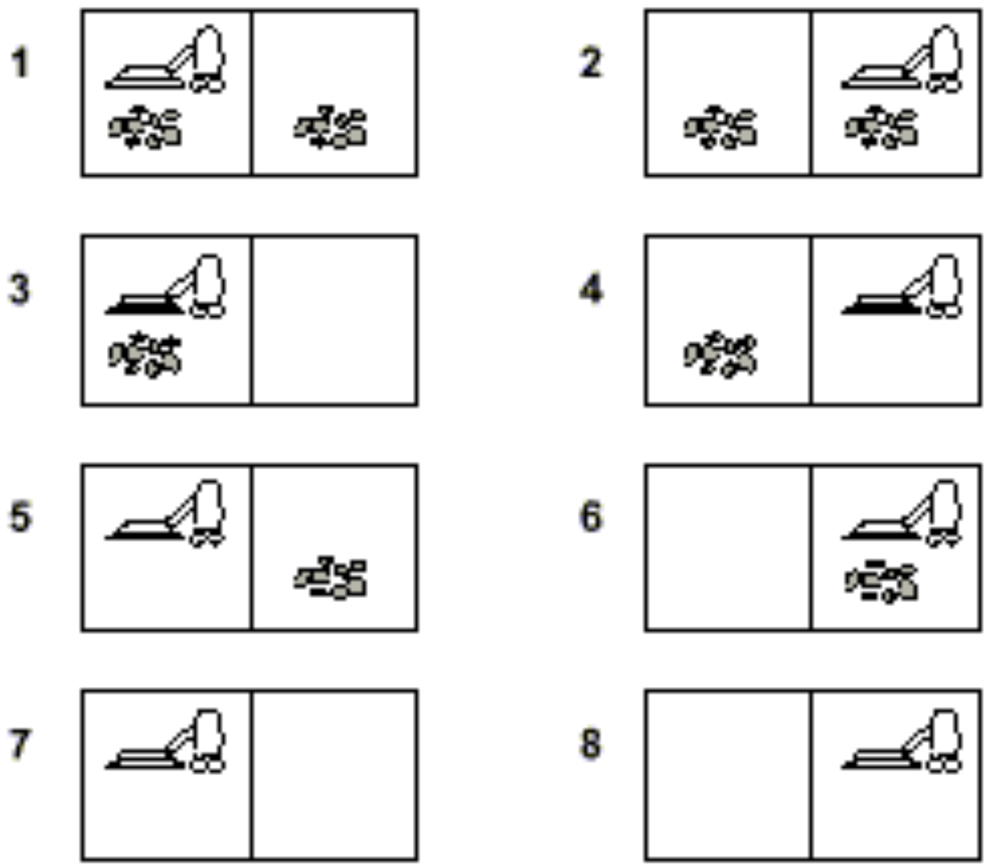
8



Possible start state

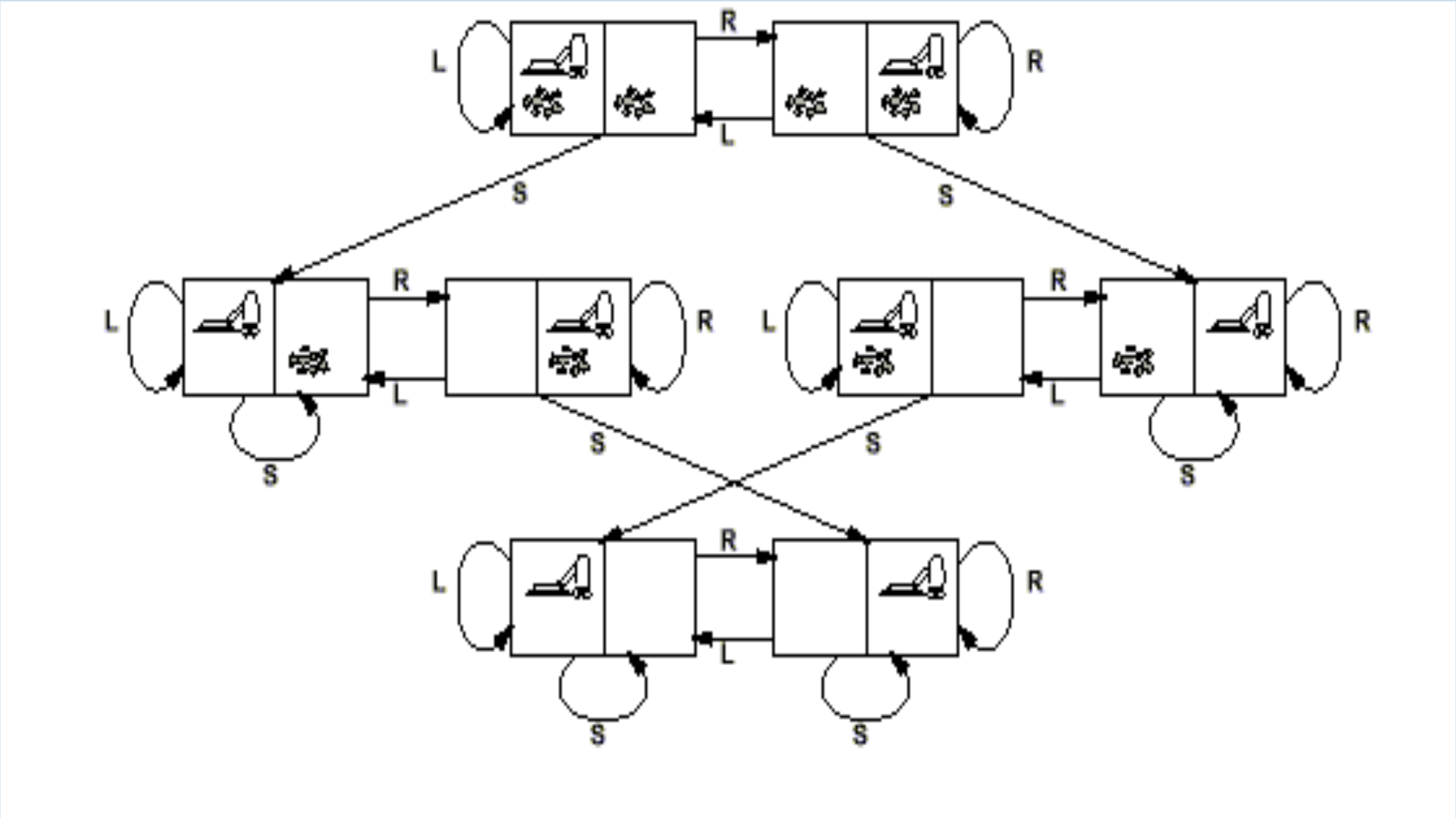
Possible goal state

Example: vacuum world



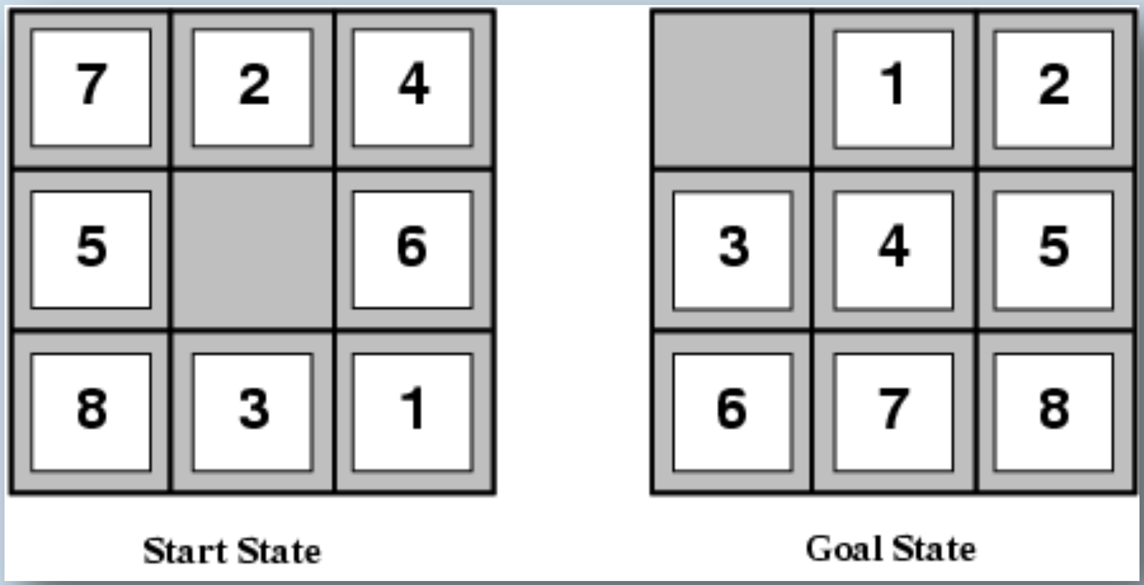
- States – one of the eight states in the picture
- Actions –*left, right, suck*
- Possible Goal – no dirt

Search Space



- Actions – left, right, suck
  - Successor states in the graph describe the effect of each action applied to a given state
- Possible Goal – no dirt

Eight Puzzle



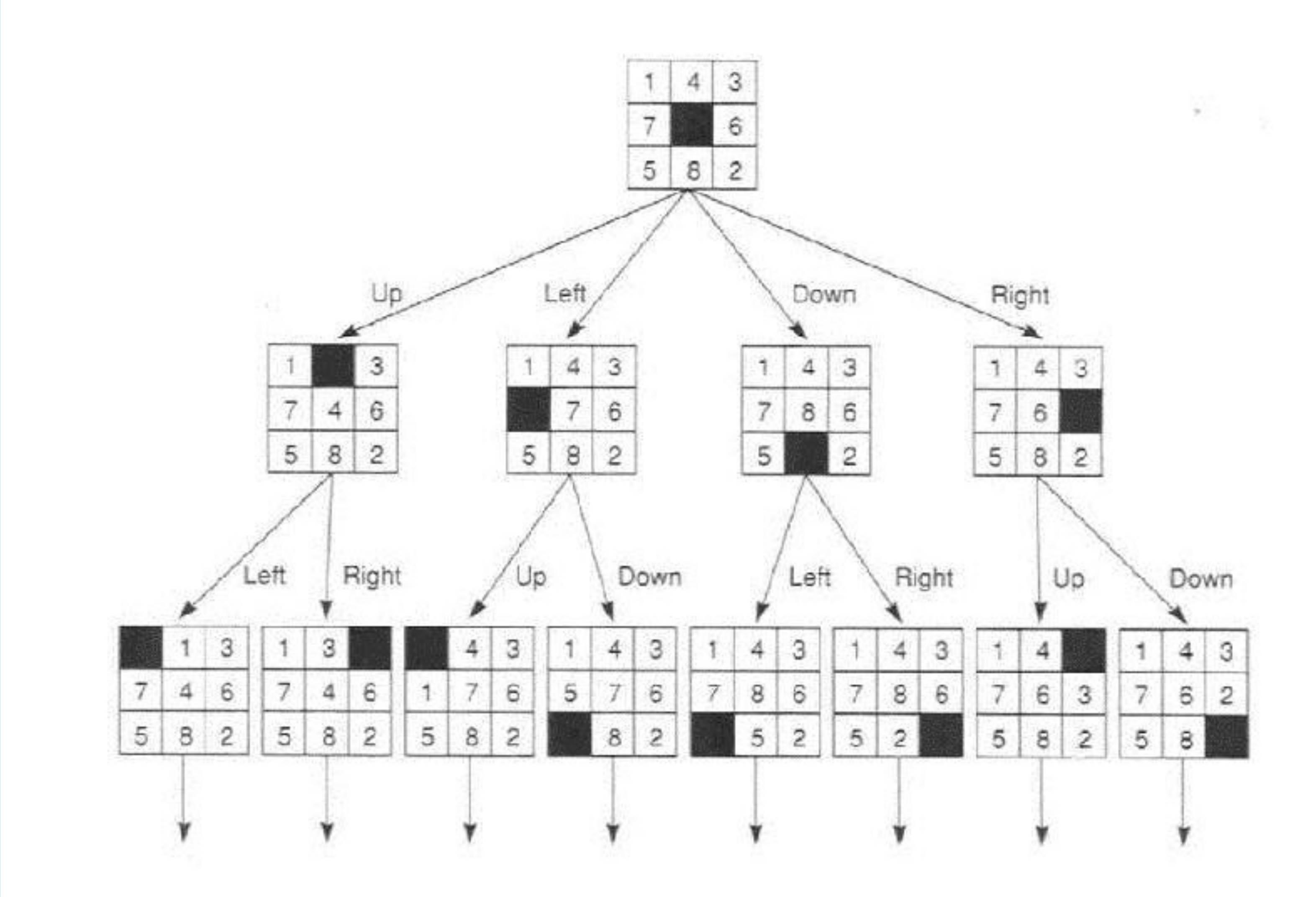
States: each state specifies which number/blank occupies each of the 9 tiles

HOW MANY STATES ? *9!*

Actions: blank moves left, right, up down

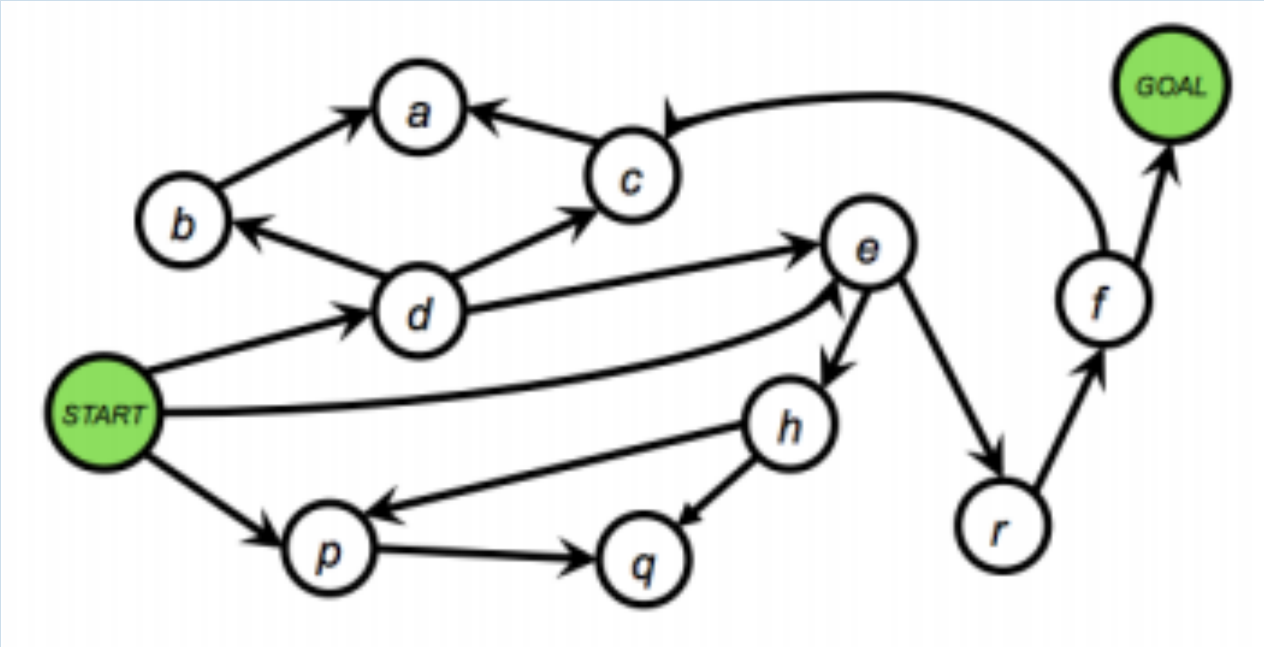
Goal: configuration with numbers in right sequence

Search space for Eight Puzzle



Search problem representation: “Just” a Graph

- States: nodes
- Actions: edges
- Path: sequence of edges



How can we find a solution?

- How can we find a sequence of actions and their appropriate ordering that led to the goal?
- Need smart ways to **search** the **state space graph**

Graph search: basic idea

- Given a graph, start node, and goal nodes, incrementally explore paths from the start node.
- Maintain a *frontier* of paths from the start node that have been explored.
- As search proceeds, the *frontier* expands into the unexplored nodes until a goal node is encountered.

Graph search: basic idea

Input:

- a graph
- a set of start nodes
- Boolean procedure goal(n) testing if n is a goal node

frontier:= [<s>: s is a start node];

While frontier is not empty:

select and **remove** path $<n_o,\ldots,n_k>$ from frontier;

If goal(n_k)

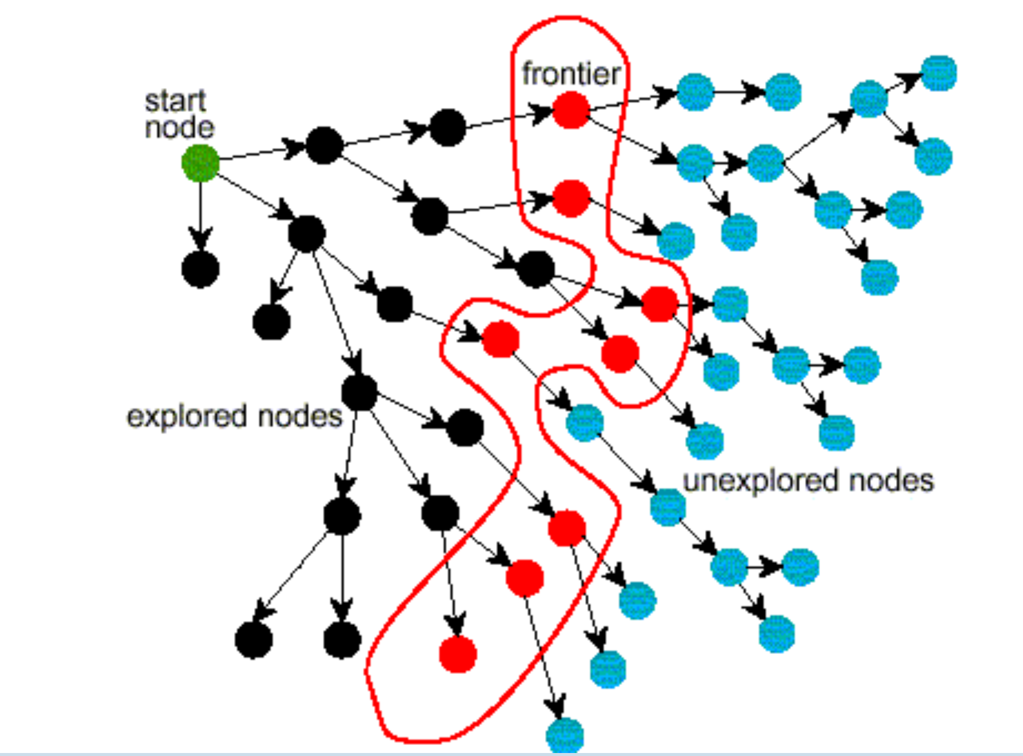
return $<n_o,\ldots,n_k>$;

For every neighbor n of n_k,

add $<n_o,\ldots,n_k, n>$ to frontier;

end

The way in which the frontier is expanded defines the **search strategy**



Comparing Search Strategies

- Algorithms are evaluated along the following four dimensions:

1. *Completeness*
2. *Optimality*
3. *Time complexity*
4. *Space complexity*

Comparing Search Strategies

1. Completeness
   - *A search algorithm is* ***complete*** *if whenever there is at least one solution, the algorithm* ***is guaranteed to find it*** *within a finite amount of time.*

2. Optimality
   - *A search algorithm is* ***optimal*** *if when it finds a solution, it is* ***the best one***

Comparing Search Strategies

3. Time complexity
   - *The* ***time complexity*** *of a search algorithm is the* **worst- case** *amount of time it will take to run, expressed in terms of:*
     - ***b*** *– maximum branching factor* of the search tree
     - ***d*** *– depth of the least-cost solution*
     - ***m*** *– maximum depth of the state space*

4. Space complexity
   - *The* ***space complexity*** *of a search algorithm is the* **worst-case** *amount of memory that the algorithm will use (i.e., the maximum number of nodes on the frontier), also expressed in terms of* **b, d** *and* **m***.*

UNINFORMED SEARCH STRATEGIES

A.K.A "Blind Search"

Uninformed Search Strategies

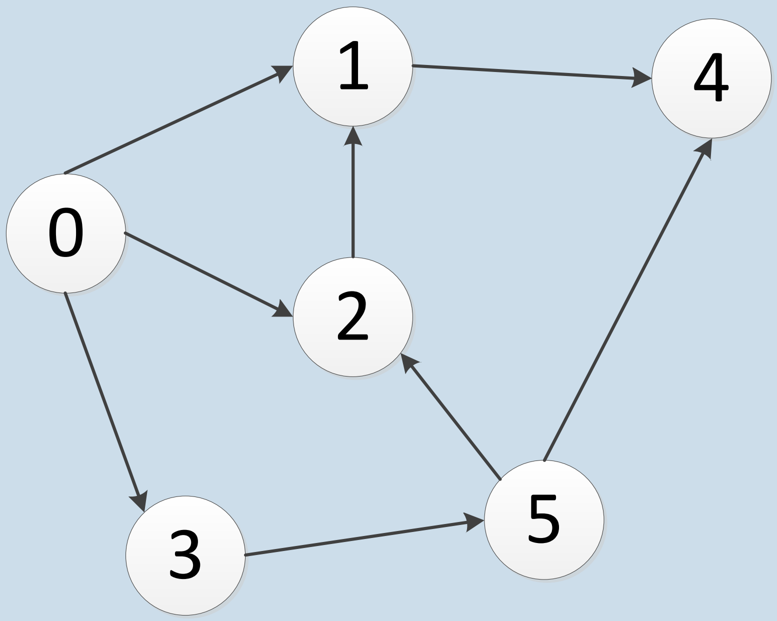
- *Uninformed* strategies use only the information available in the problem definition (i.e., no additional information is available)
  - *Breadth-first search*
  - *Uniform cost search*
  - *Depth-first search*
  - *Depth limited search*
  - *Iterative deepening search*
  - *Bidirectional search*

Depth-First Search

In DFS, the frontier is a last-in-first-out (stack)

- Basic idea
  - *explores each path on the frontier until its end (or until a goal is found) before considering any other path.*

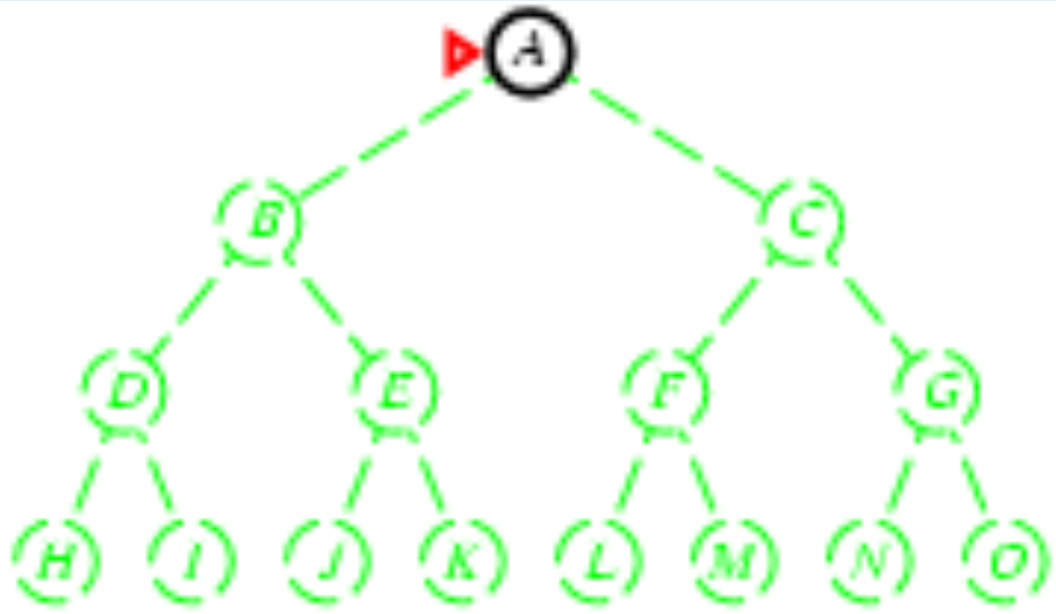
- Example: run the DFS for the given graph.
  - *Take 0 as the start state & 5 as the goal state*
  - *Use lexicographic order to put states into the frontier (i.e., a stack in this case)*



Depth-First Search

In DFS, the frontier is a
last-in-first-out (stack)

- Basic idea
  - *explores each path on the frontier until its end (or until a goal is found) before considering any other path.*
- Implementation
- Properties
  - *Complete?*
  - *Optimal?*
  - *Time?*
  - *Space?*



Depth-First Search

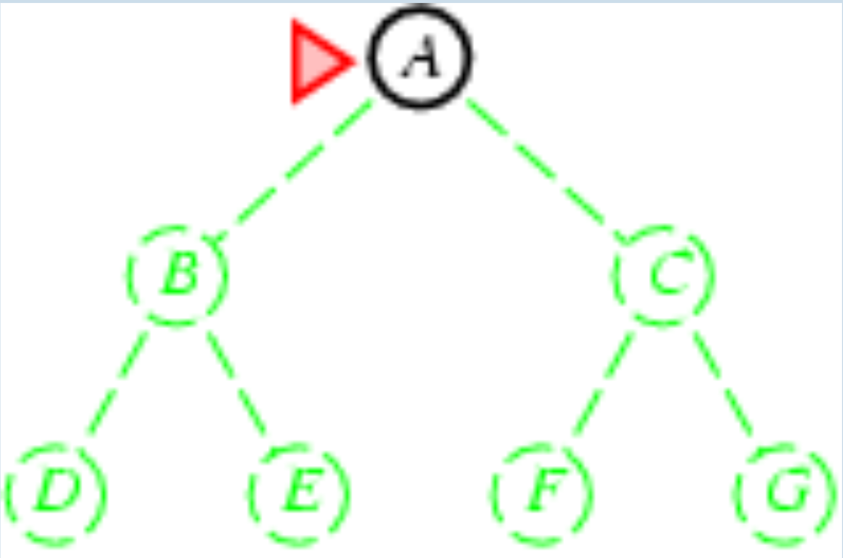
- Complete? No
  - *fails in infinite-depth spaces, spaces with loops*
  - *Complete in finite spaces*
- Optimal? No
- Time? $O(b^m)$
  - *terrible if m is much larger than d*
  - *But if solutions are dense, may be much faster than breadth-first*
- Space? $O(bm)$, i.e., linear space!

- Preventing loop paths?

Breadth-First Search

In BFS, the frontier is a
first-in-first-out (queue)

- Basic idea
  - *explores all paths of length k on the frontier, before looking at path of length k + 1*
- Implementation
- Properties
  - *Complete?*
  - *Optimal?*
  - *Time?*
  - *Space?*



Breadth-First Search

- Complete? Yes (if *b* is finite)
- Optimal? Yes (if cost = 1 per step); not optimal in general
- Time? $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$ (*i.e., exponential in d*)
- Space? $O(b^d)$ (keeps every node in memory)
- Space is the bigger problem (more than time).

BFS vs. DFS: which one to use?

- The search graph has cycles or is infinite

  BFS

- We need the shortest path to a solution

  BFS

- There are only solutions at great depth

  DFS

- There are some solutions at shallow depth and others deeper

  BFS

- No way the search graph will fit into memory

  DFS

Reading Material

- Russell & Norvig: Chapter # 3
- David Poole: Chapter # 3