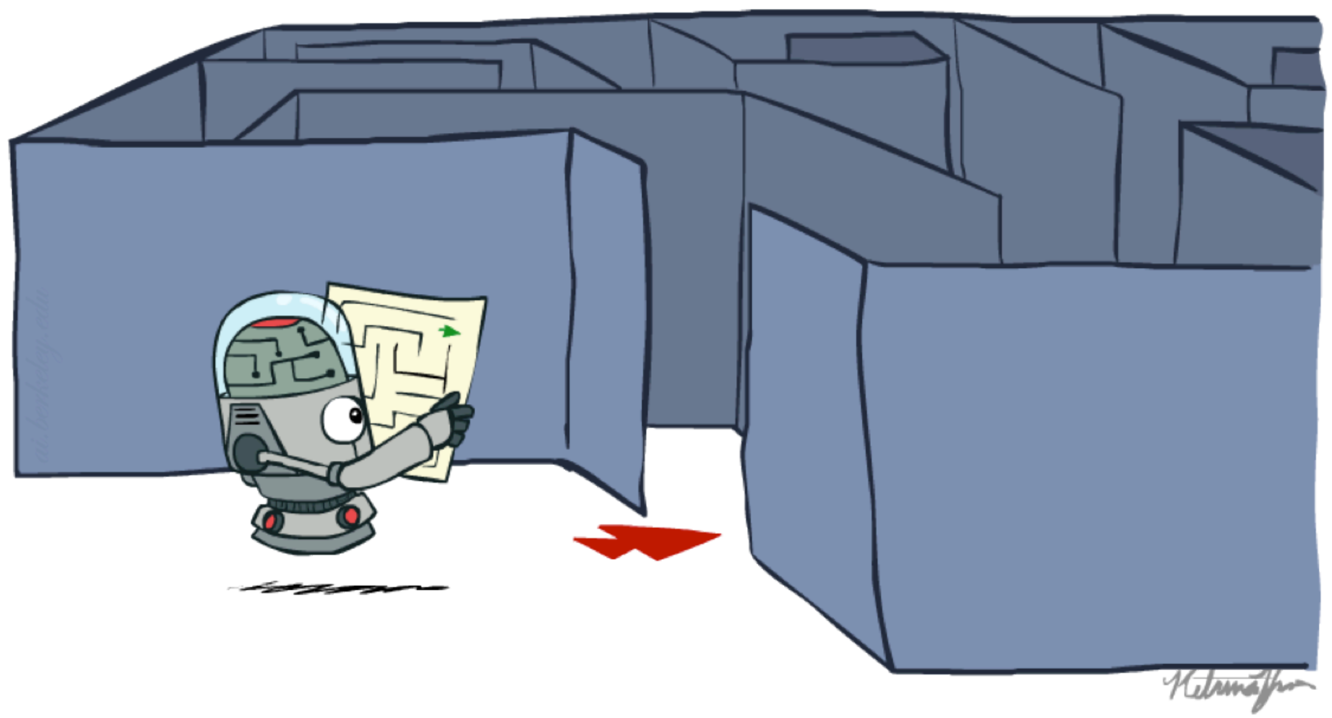
Announcements

- Project 0: Python Tutorial
  - Due today at 11:59pm
- HW1 will be released today
- P1 will be released on Wednesday
- Survey about flipped classroom

Ve492: Introduction to Artificial Intelligence

Search Problem and Uninformed Search



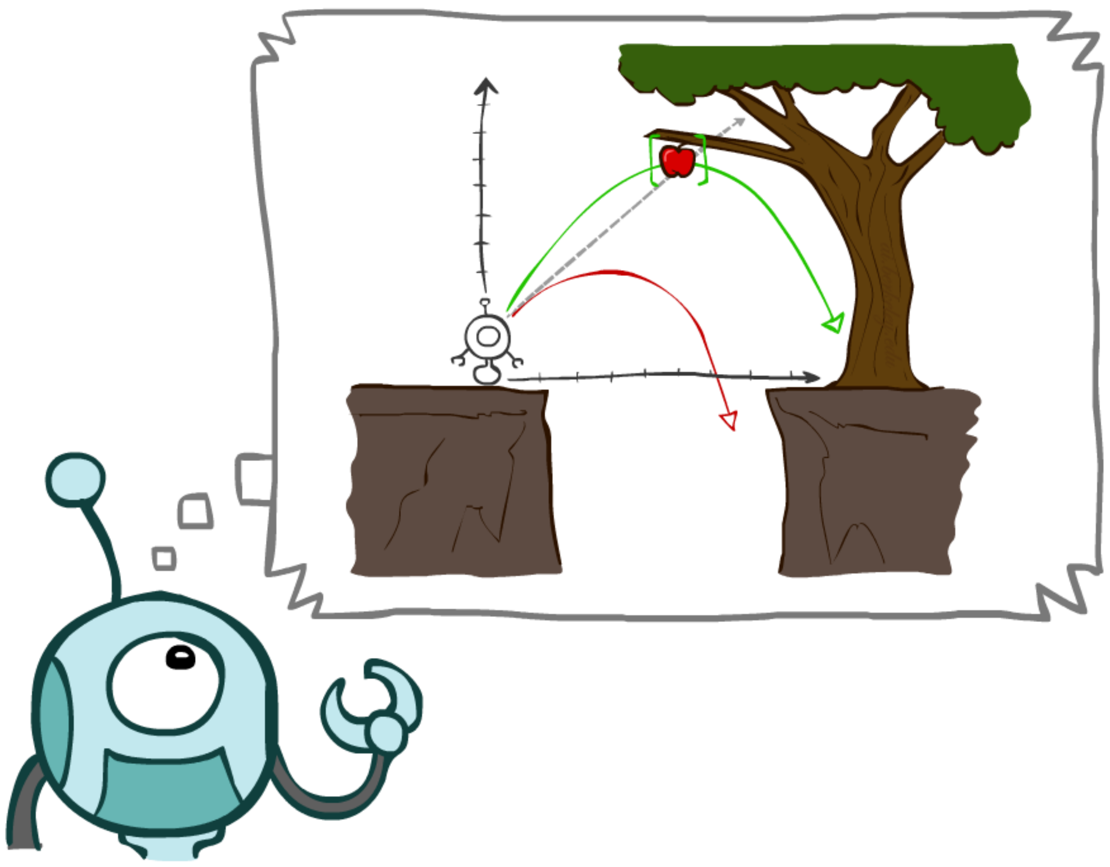
Paul Weng

UM-SJTU Joint Institute

Slides adapted from http://ai.berkeley.edu, AIMA, UM, CMU

Outline

- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Iterative Deepening Search
  - Uniform-Cost Search



Search Problems

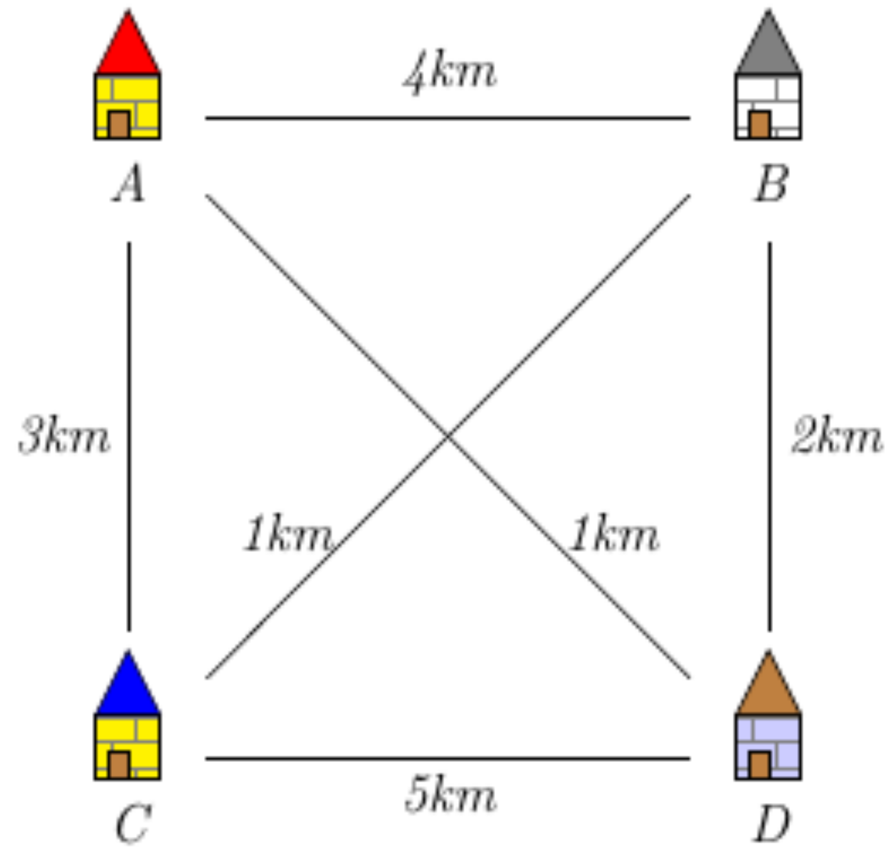
Solving Peg Solitaire is a Search Problem?

- True or False?
- If True, what are the states, actions, initial state, goal test, and costs?
- If True, what is an upper-bound on the size of the state space?
- If True, what is an upper-bound on the number of actions?

Solving Chinese Chess is a Search Problem?

- True or False?
- If True, what are the states, actions, initial state, goal test, and costs?
- If True, what is upper-bound on the size of the state space?
- If True, what is an upper-bound on the number of actions?

Traveling Salesman Problems are Search Problems?

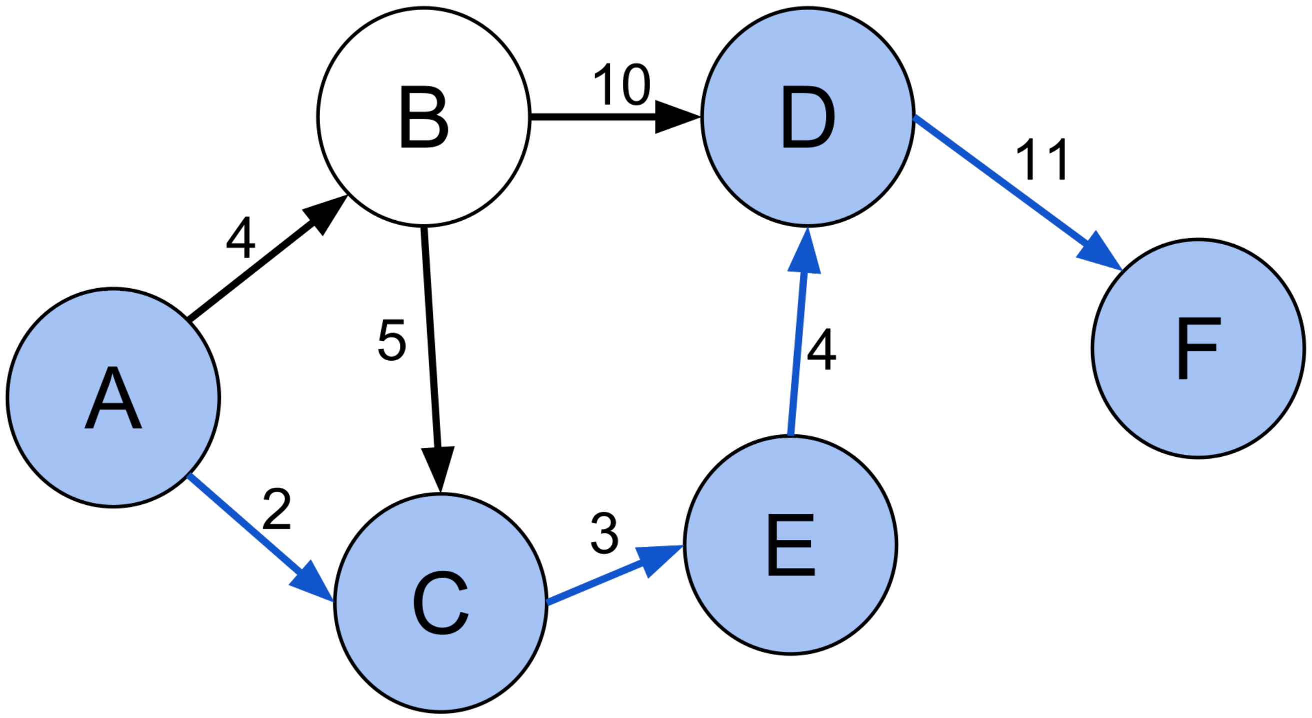


- True or False?
- If True, what are the states, actions, initial state, goal test, and costs?
- If True, what is upper-bound on the size of the state space?
- If True, what is an upper-bound on the number of actions?

Environment Type for a Search Problem

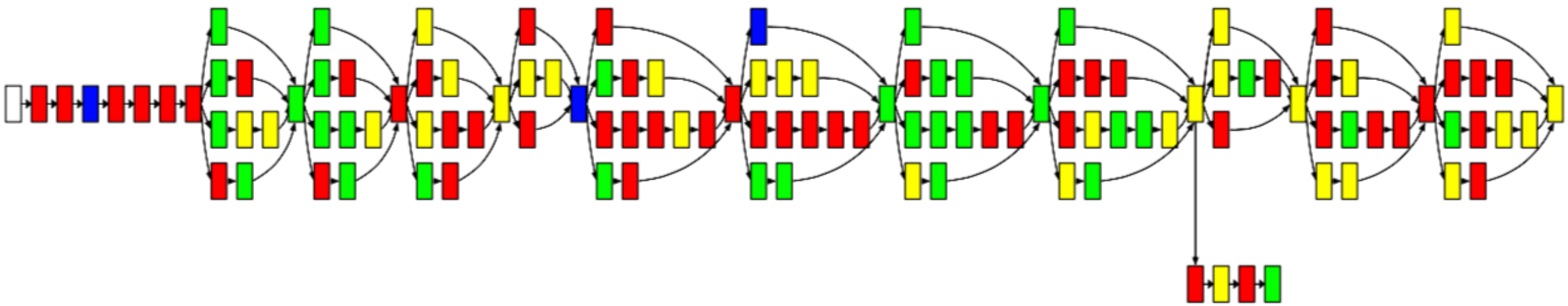
	Search Problem
Fully or partially observable	Fully
Single agent or multi-agent	Single agent
Deterministic or non-deterministic	Deterministic
Static or dynamic	Static
Discrete or continuous	Discrete
Episodic or sequential	Sequential

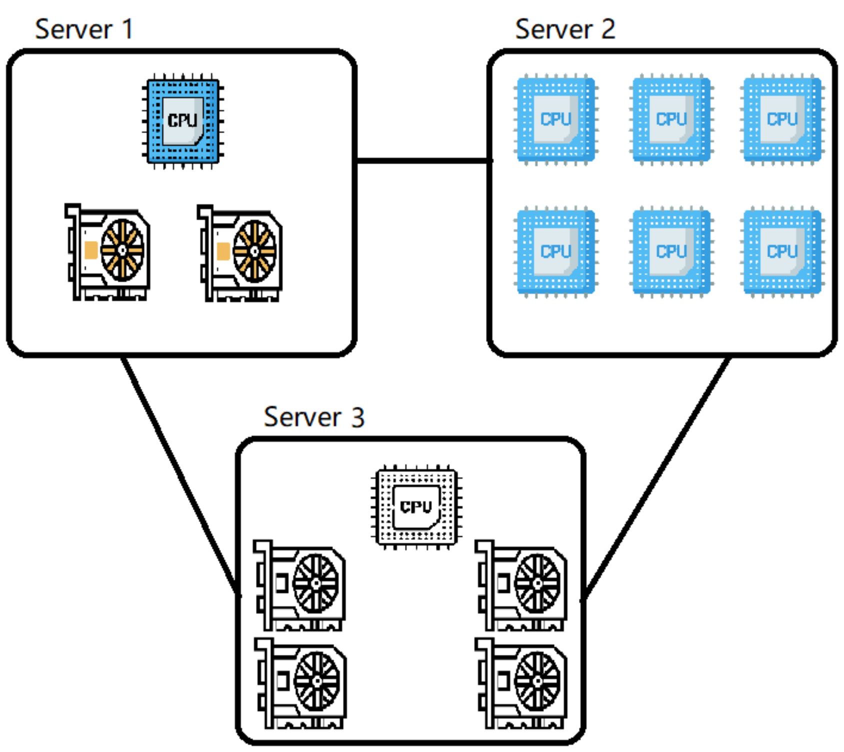
Search Problems are Generalization of Shortest Path Problems



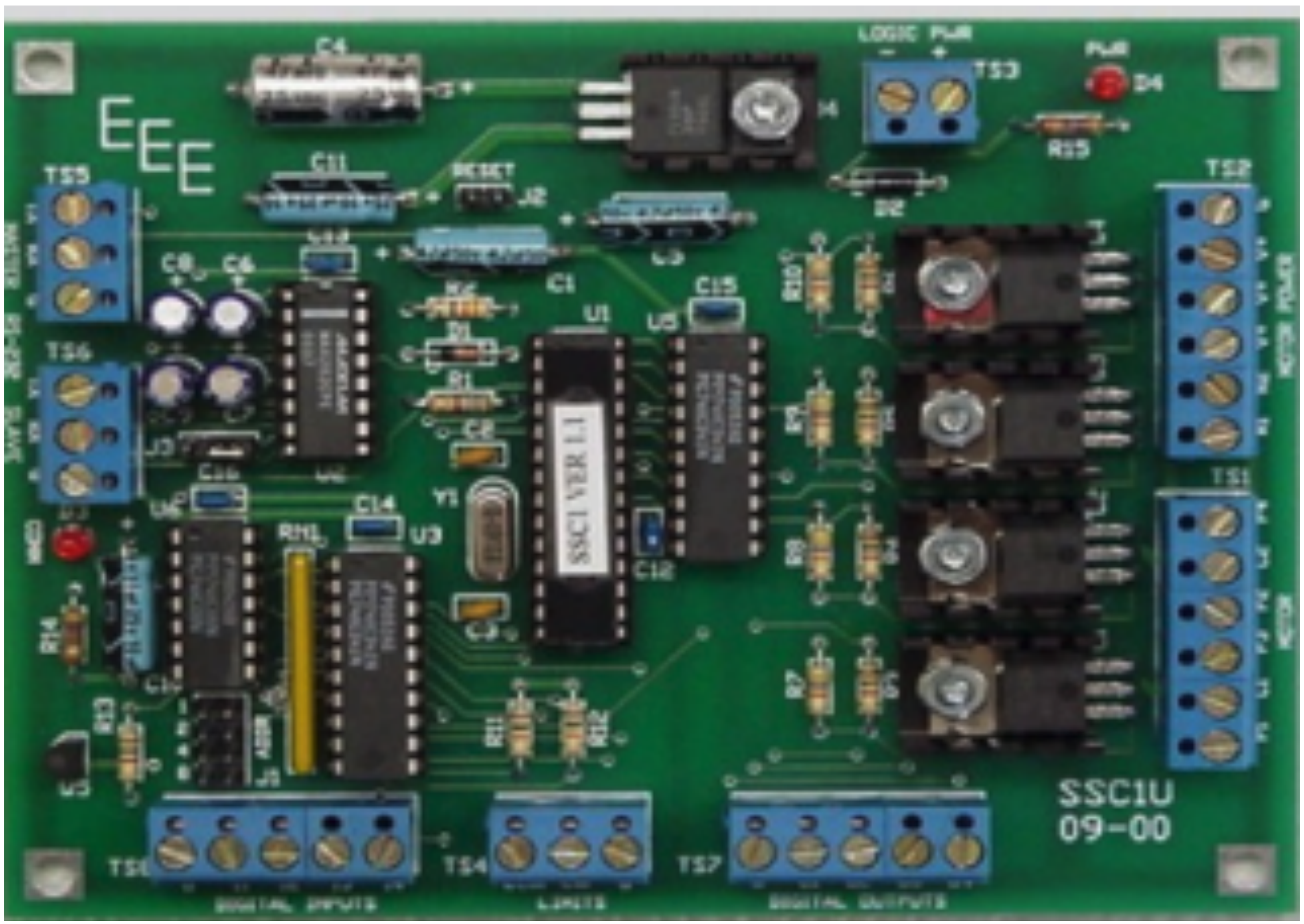
- Shortest Path Problems are in class P.
- However, state space is generally huge.

Example: Scheduling Problem

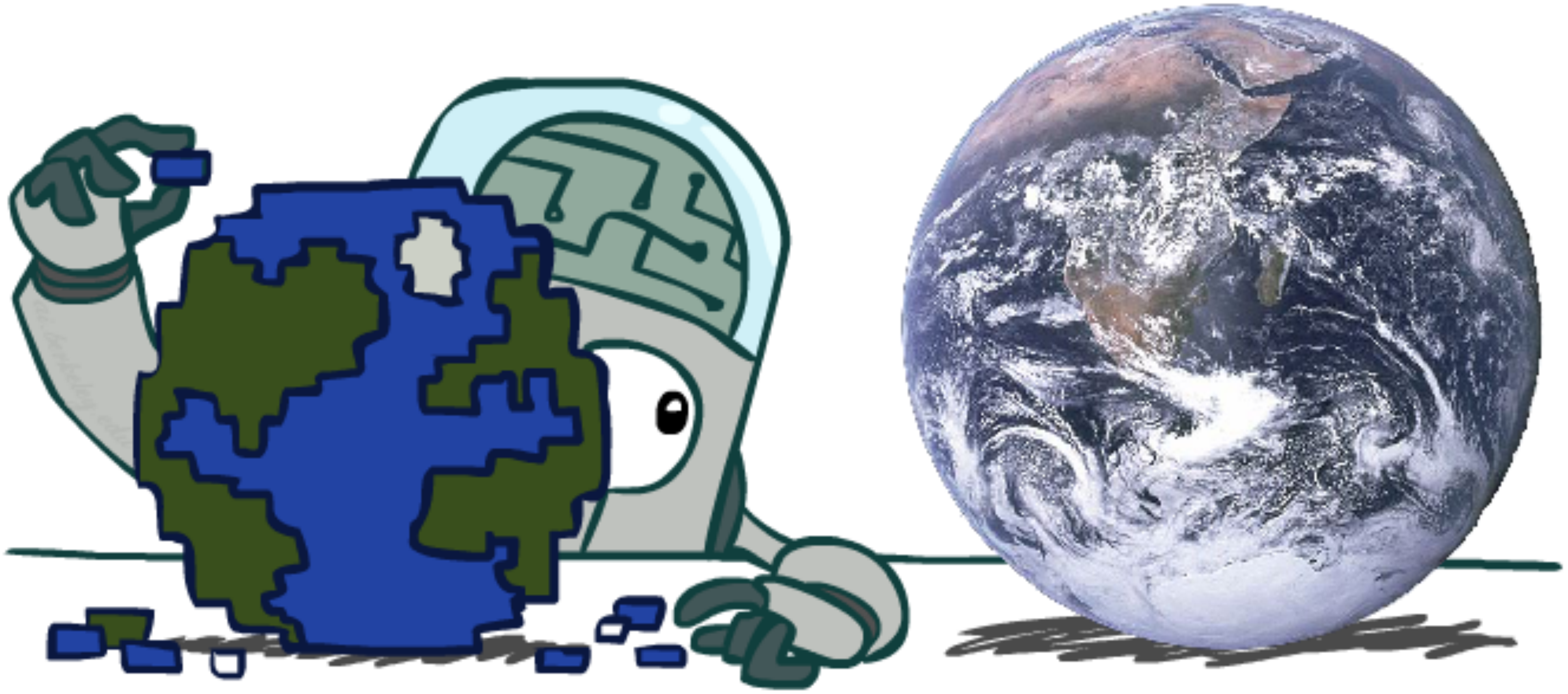




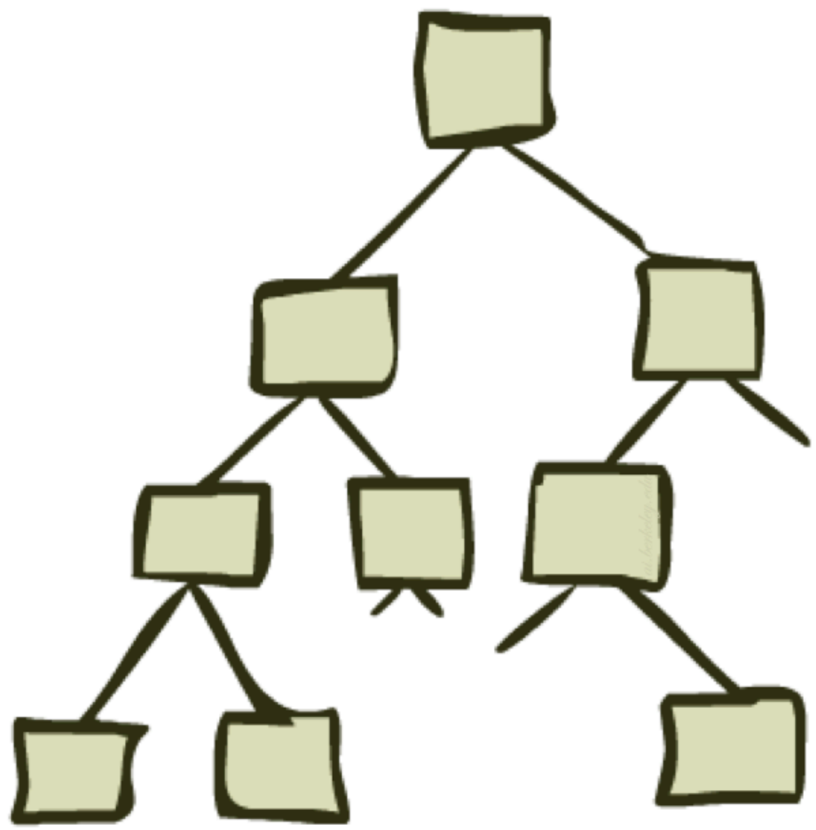
Example: Electronic Design Automation



Search Problems Are Models



State Space Graphs and Search Trees

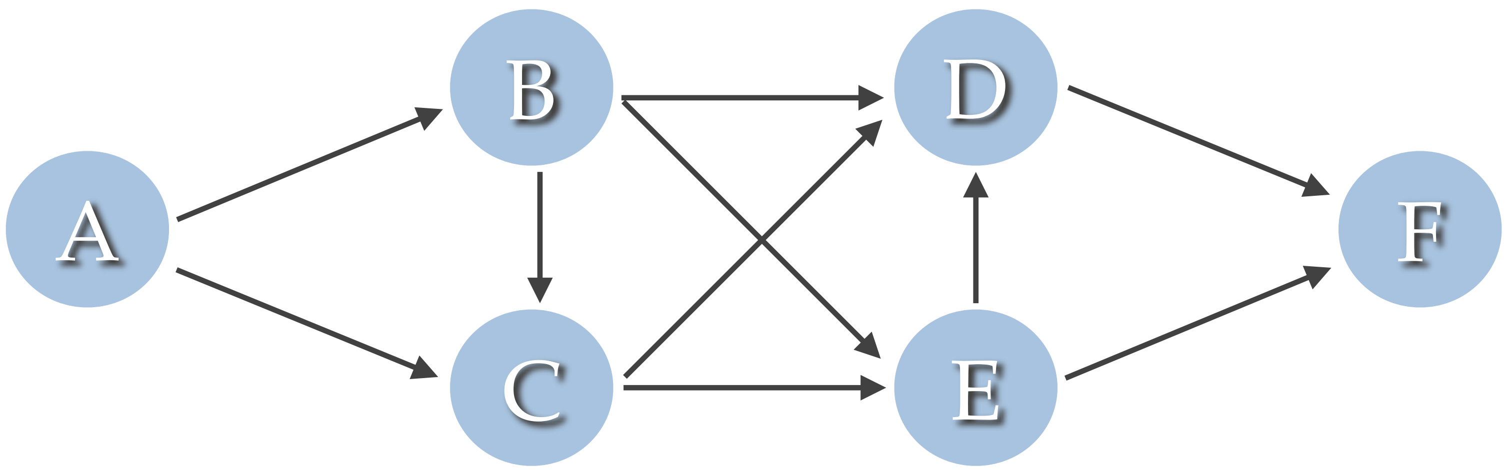


True or False: Search Tree

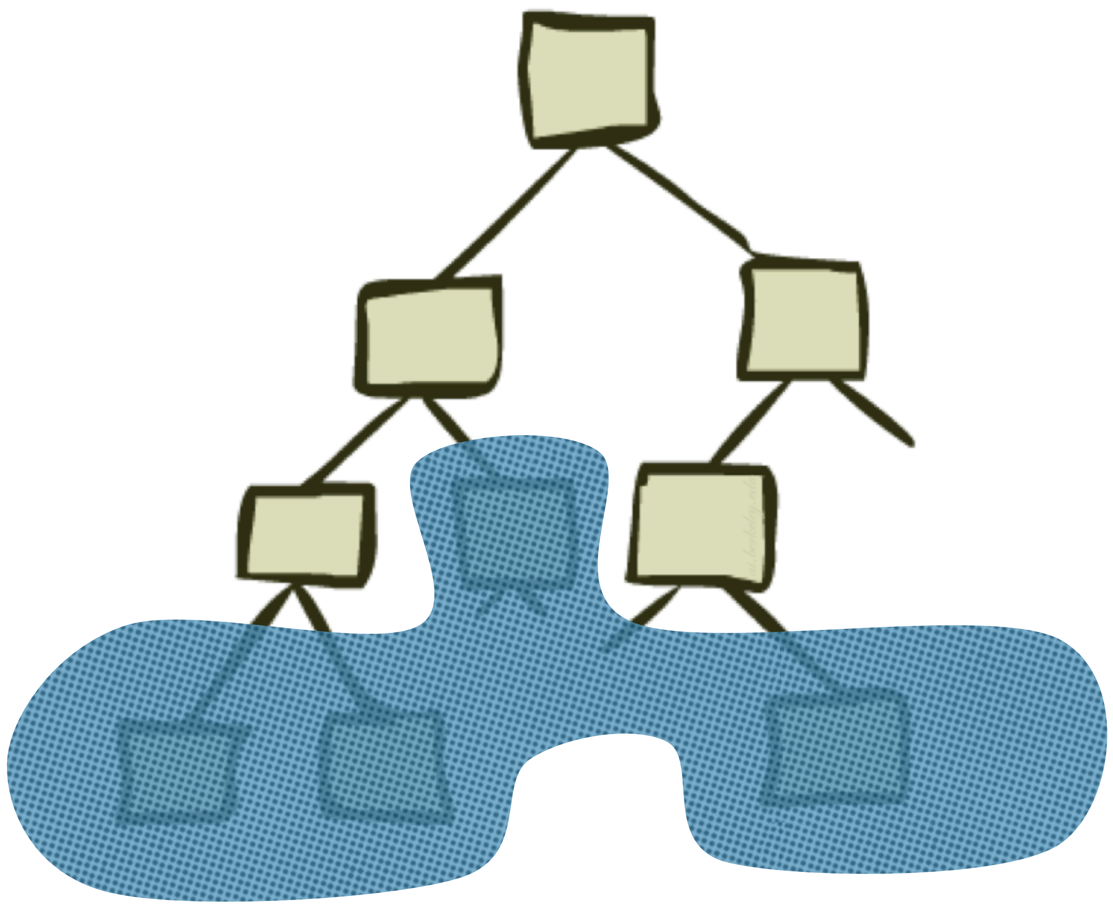
- A search problem implicitly defines a graph. T
- A graph implicitly defines a search tree. F don't know the root
- A search problem implicitly defines a search tree. T
- A node of a search tree corresponds to a state in the search problem. F
- A node of a search tree corresponds to a subpath in the graph. T
- A search tree is a compact way to represent a set of path. T
- If a graph has a finite number of states, the search tree has a finite number of nodes. F

Search Tree

- Draw the search tree corresponding to this graph where the initial state is A and the goal test is F:



Tree Search



True or False: General Tree Search

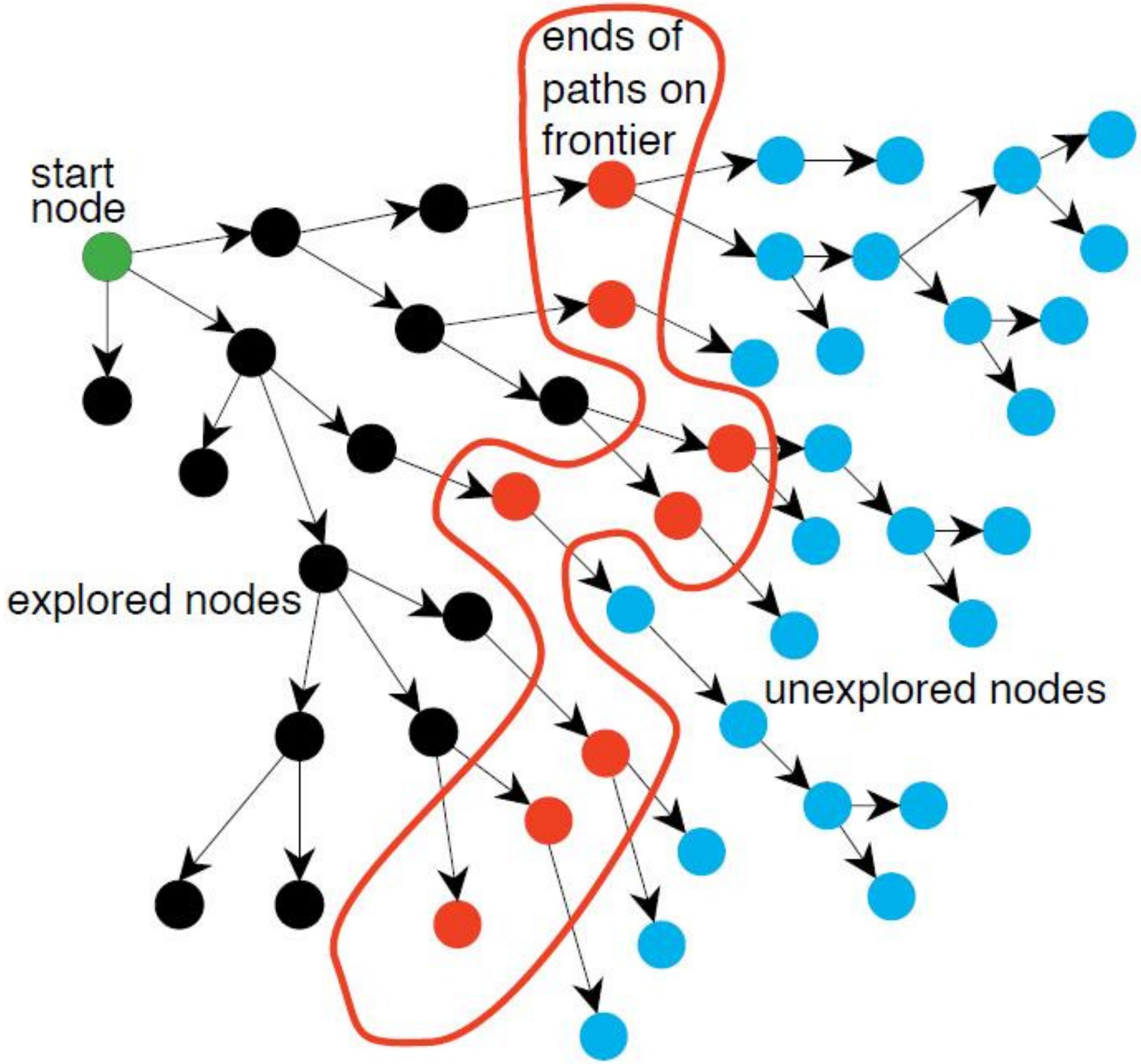
- The fringe corresponds to the nodes that have already been visited. F
- Expanding a node means visiting all its neighbors. F
- A state that is visited cannot be visited again. T

General Tree Search

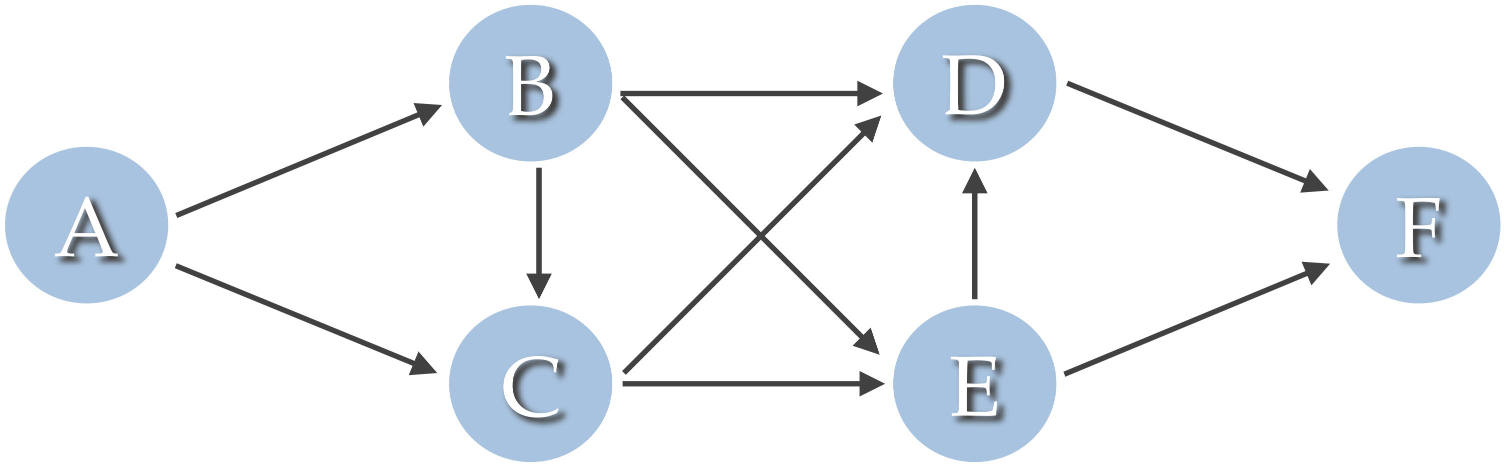
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?

Search

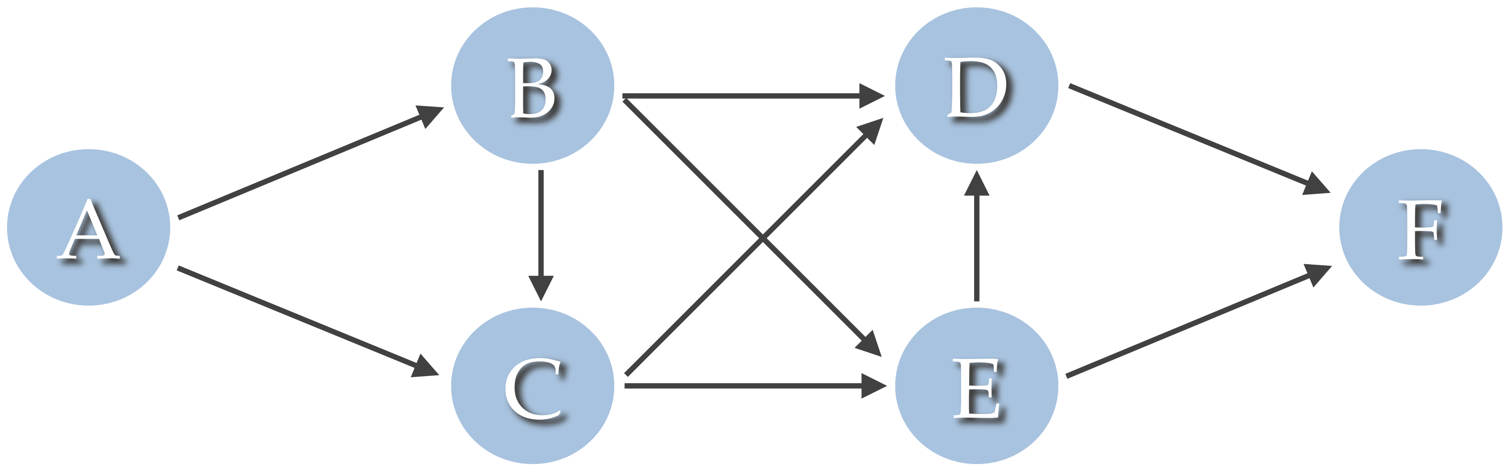

ends of
paths on
frontier
start
node
explored nodes
unexplored nodes

Run Depth First Search



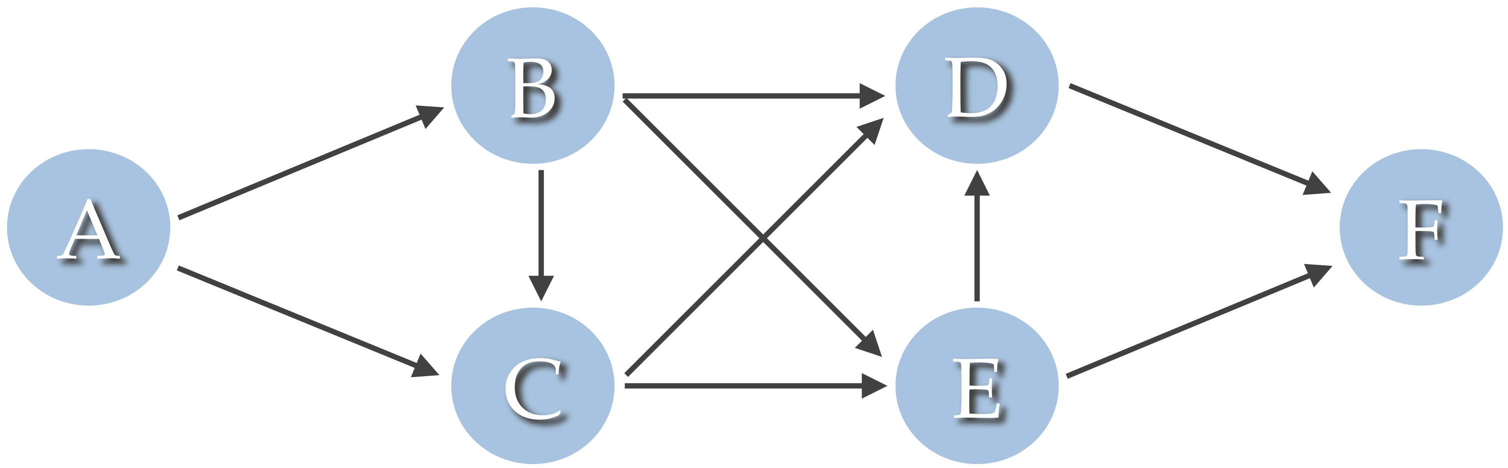
- When there's a choice, nodes are chosen in alphabetic order.

Run Breadth First Search



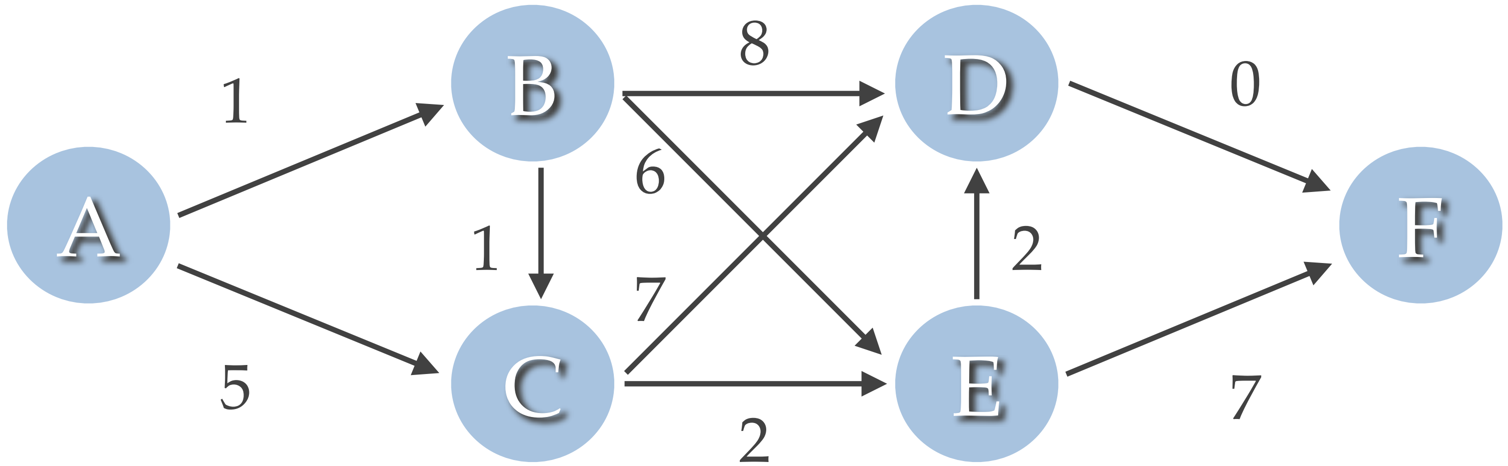
- When there's a choice, nodes are chosen in alphabetic order.

Run Iterative Deepening Search



- When there's a choice, nodes are chosen in alphabetic order.

Run Uniform Cost Search



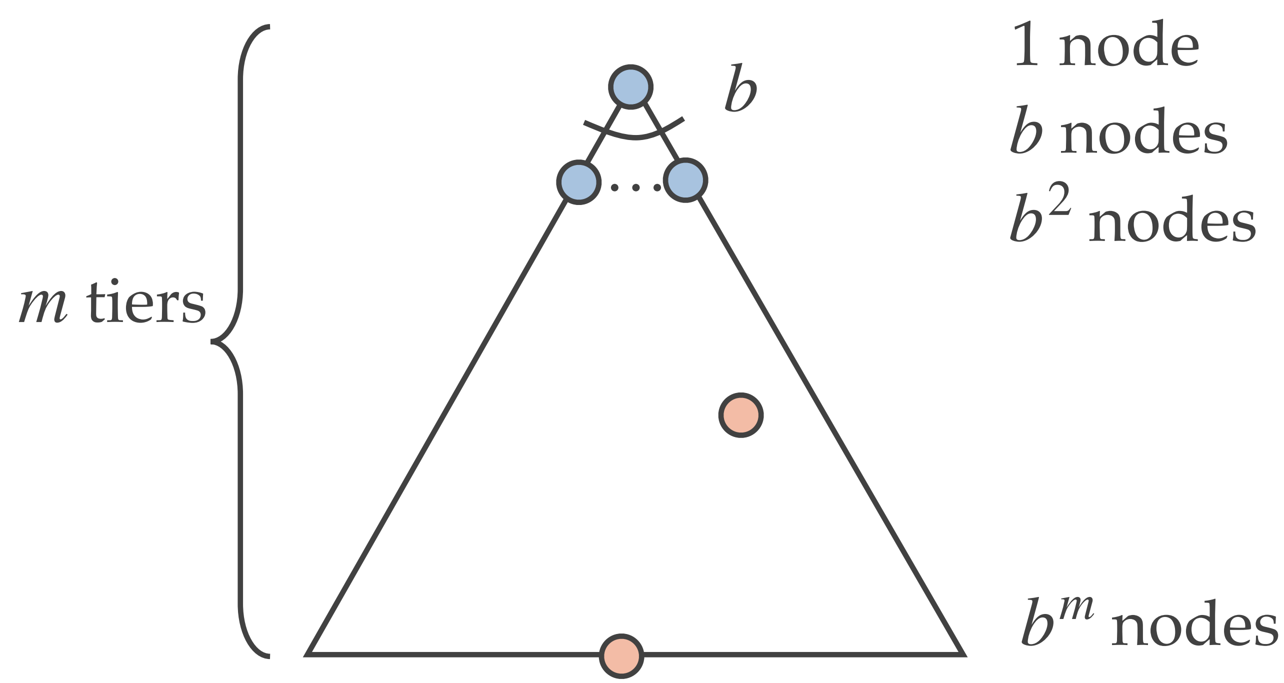
- When there's a choice, nodes are chosen in alphabetic order.

True or False: DFS, BFS, UCS

- Consider a search problem where for every action, the cost is equal to ϵ, with $\epsilon > 0$.
  - Depth-first search is guaranteed to return an optimal solution. F
  - Breadth-first search is guaranteed to return an optimal solution. T
  - Uniform-cost search is guaranteed to return an optimal solution. T
- Consider a search problem where for every action, the cost is at least ϵ, with $\epsilon > 0$.
  - Depth-first search is guaranteed to return an optimal solution. F
  - Breadth-first search is guaranteed to return an optimal solution. F
  - Uniform-cost search is guaranteed to return an optimal solution. T

Search Algorithm Properties

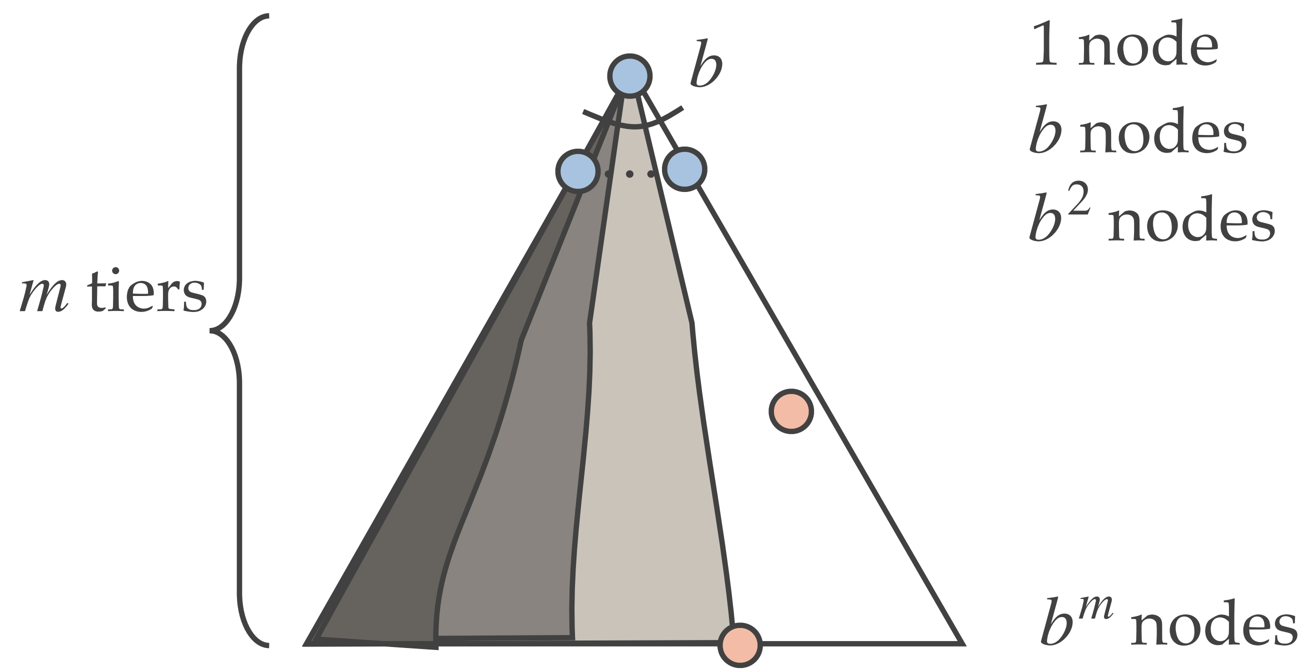
- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths
- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots + b^m = O(b^m)$



True or False: Complexity

- The time complexity of a search algorithm depends on the number of nodes that are visited. T
- The space complexity of a search algorithm depends on the number of nodes that are visited. F
- The time complexity of IDS is the same as DFS. F
- The space complexity of IDS is the same as BFS. F
- The ratio of the time complexity of IDS and BFS tends to 1 when s tends to infinity. T?

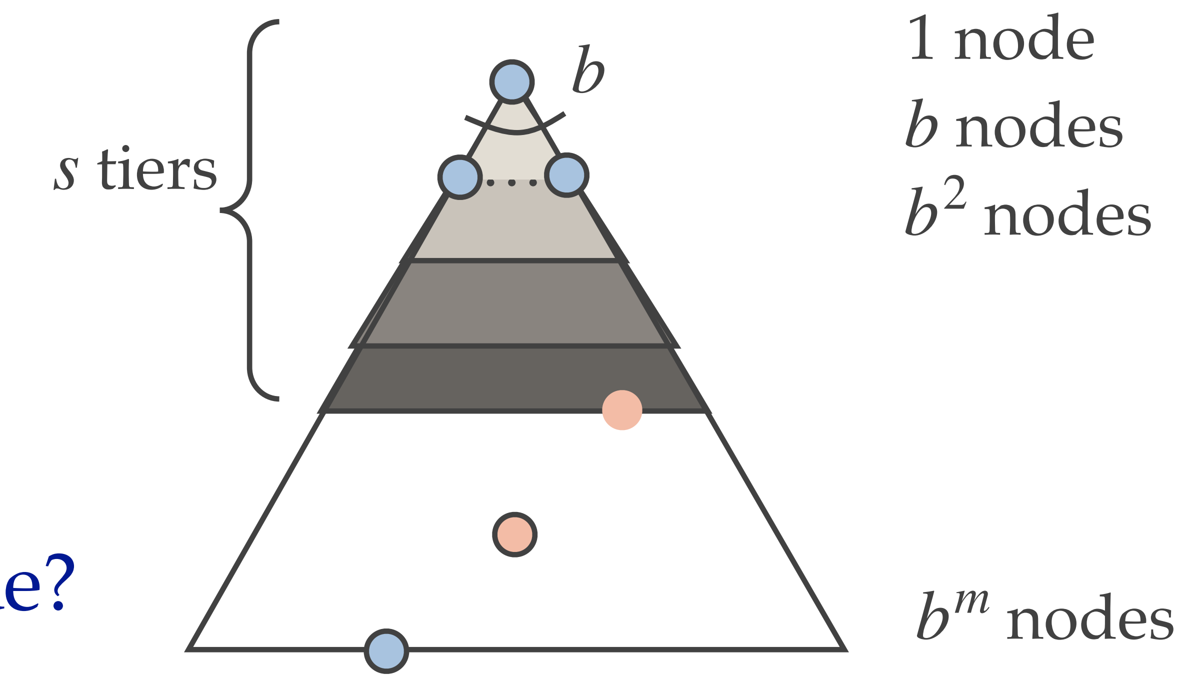
Depth-First Search (DFS) Properties



- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the fringe take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - m could be infinite, so only if we prevent cycles (more later)
- Is it optimal?
  - No, it finds the "leftmost" solution, regardless of depth or cost

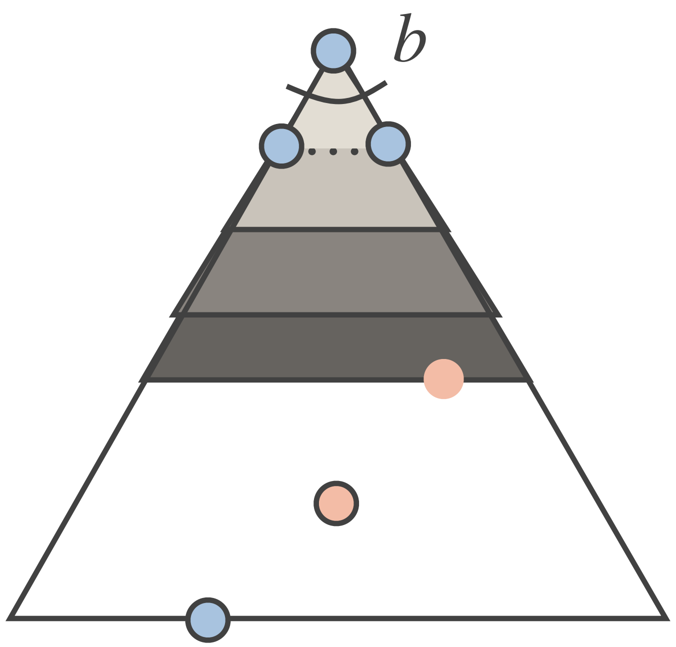
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the fringe take?
  - Has roughly the last tier, so (b^s)
- Is it complete?
  - s must be finite if a solution exists, so yes!
- Is it optimal?
  - Only if costs are all 1 (more on costs later)



Iterative Deepening

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..



- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!

Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε , then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{\frac{C^*}{\varepsilon}})$ (exponential in effective depth)
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^{\frac{C^*}{\varepsilon}})$
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)