CSC 480: Artificial Intelligence : Search

- The materials in this course are modified and adapted. Many thanks to the authors and their institutions for making these materials available with an Open Courseware License or for use for educational purposes only.

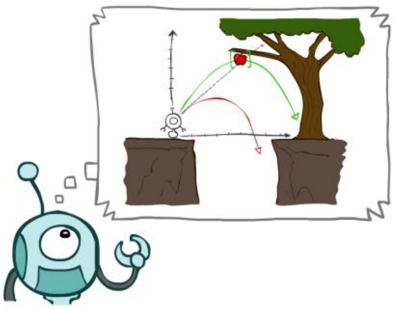
Sources:

- **Harvard's CS50: Introduction to Artificial Intelligence with Python**
  Brian Yu and David J. Malan
- **UCB's course materials from CS 188: Introduction to Artificial Intelligence**
  Dan Klein, Pieter Abbeel University of California, Berkeley
- **Cal Poly, San Luis Obispo CS 480:** Franz Kurfess

- **Artificial Intelligence: A Modern Approach**
  Copyright © 2021 Pearson Education

Planning and Search

- Agents that Plan Ahead
- Search Problems
- **Uninformed Search Methods**
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search

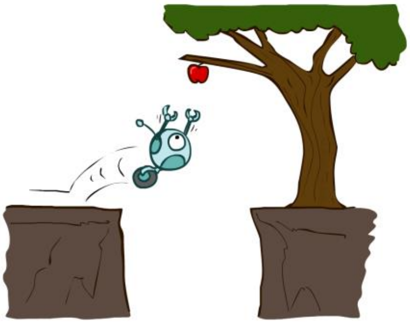


Today Key Ideas for Search

- Planning Agent
- Search Problems
- State Space
- Search Tree
- State Space Graphs
- Tree Search vs Graph Search
- Breadth First Search
- Depth First Search
- General Search Algorithm
- Uniform Cost Search

Reflex Agents

- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - May have memory or a model of the world's current state
  - Do not consider the future consequences of their actions
  - Consider how the world IS
- *Can a reflex agent be rational?*



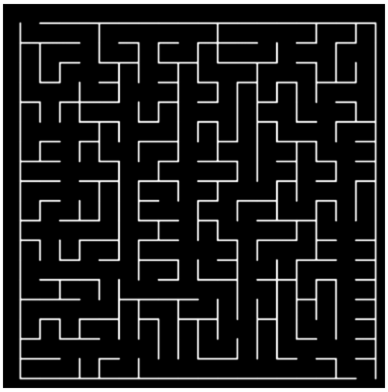
Planning Agents in search

Planning agents:

- Ask “what if” and **simulate** the world
- Decisions based on (hypothesized) consequences of actions
- Must have a model of how the world evolves in response to actions
- Must formulate a goal (test)
- Consider how the world WOULD BE IF – **simulates** finding the goal
- Returns a path using information it gains in searching for the goal

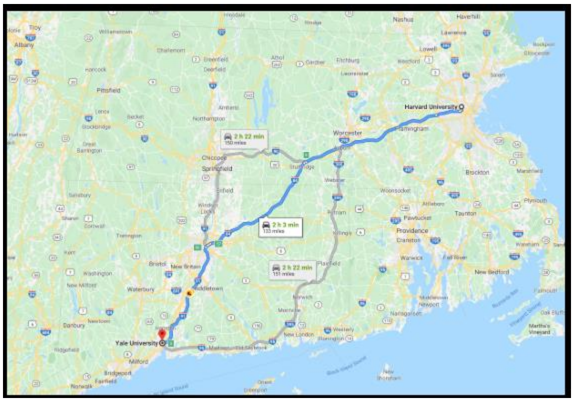
Search Problems: 15 Puzzle

Maze Solving



7

Route Planner



8

Representing a search problem

- Problem Representation:
  – Defining the problem in a way that an AI algorithm can understand and work with
  – Typically involves defining the initial state, the goal state or states, the set of possible states
  – The actions that enable transitions between states.
- Problem representation and state space search are closely linked.
  – A well-structured problem representation defines the state space's boundaries and the transitions between states.
  – Effective problem representation simplifies the task of searching through the state space to find a solution.

States and Actions

- **States**: In the context of a state space
  – A "state" refers to a specific configuration or situation that the problem-solving agent can occupy.
  – States can represent a wide range of conditions, depending on the problem.
- **Actions** (or Transitions or Operators):
  – Represent the means the problem-solving agent moves from one state to another within the state space.
  – Actions or transitions represent what the agent that can do that result in moving from a given state to a resulting state.

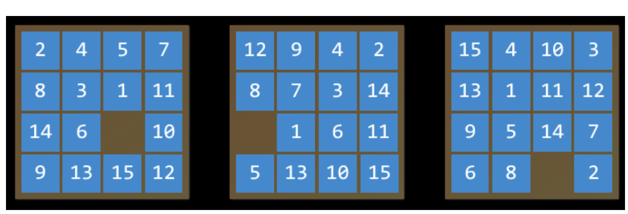
Formal Search Problem Definition

A search problem is defined by:

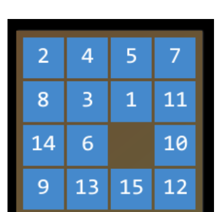
- **A State Space = set of all possible states**:
  - **Initial State:** The state from which the search algorithm starts
  - **Goal Test:** A condition that determines whether or not a state is a goal state (could be specified as a subset of the state space)
- **Actions:** Choices that can be made in a state.
- **Transition Model:** A description of what state results from performing any applicable action in any state.
  Trans (s,a) → s’ (or more generally give the probability of s’)
- **Action Cost Model:** Act-Cost (s, a, s’) → a measure of cost

Example: Four (or 15) Puzzle

- States

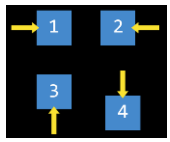


- Init (start) state

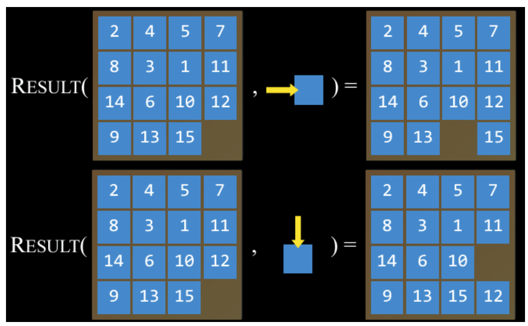


Example: Four (or 15) Puzzle

- Actions



- Transitions



15 puzzle: Examples

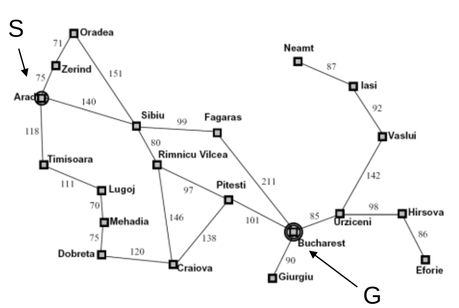
- **Agent**: In a 15 puzzle, for example, the agent would be a player who would be given a configuration and take actions that would attain the goal state.
- **State**: For example, in a 15 puzzle, a state is any one way that all the numbers are arranged on the board.
  - **Initial State**: In a navigator app, that would be the current location.
  - **Goal State**: Destination
- **Actions**: For example, in a *15 puzzle*, the actions of a given state are the ways you can slide squares in the current configuration (4 if the empty square is in the middle, 3 if next to a side, 2 if in the corner).

Search Problems

- **Transition Model**: For example, given a certain configuration of a *15 puzzle* (state s), moving a square in any direction (action a) will bring to a new configuration of the puzzle (the new state).
- **State Space:** For example, in a 15 puzzle, the state space consists of all the 16!/2 configurations on the board that can be reached from any initial state. The state space can be visualized as a directed graph with states, represented as nodes, and actions, represented as arrows between nodes.

state space the set of all states **reachable** from the initial state by any sequence of actions

Example: Traveling in Romania



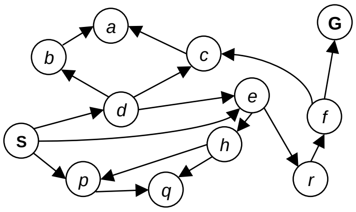
- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

Search Problems Are Models

- Models are abstractions of **a** world
- All models are *wrong* (for any interesting world)
- Models can be *good enough* to provide
  - Answers
  - Guidance
  - Insight

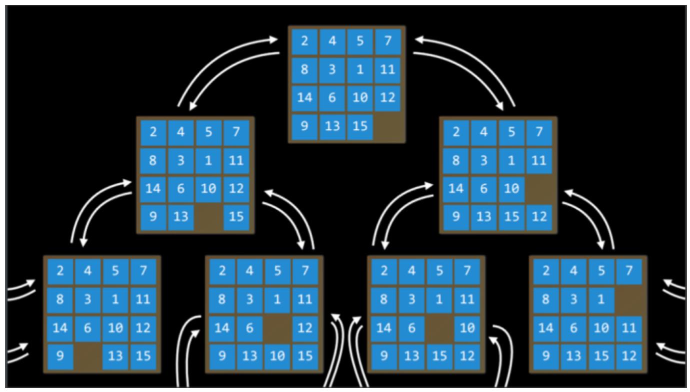
State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations "Search State" may contain less info than "World State"
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it's too big), but it's a useful idea

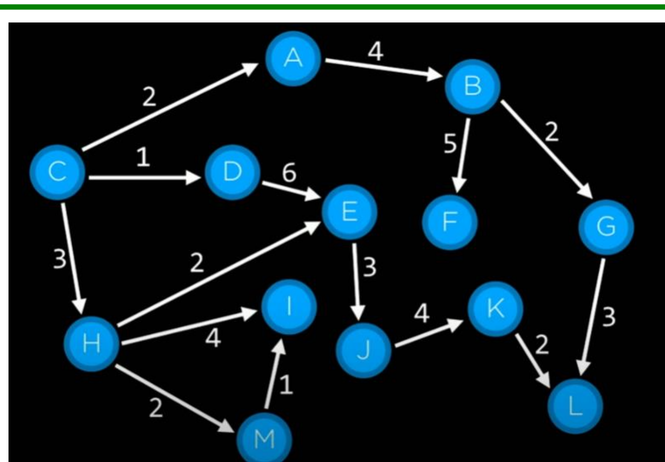


Tiny search graph for a tiny search problem

Part of a State Space Graph for 15 puzzle



Path Costs

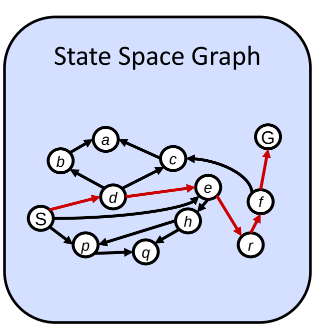


Search Trees

A search tree represents:

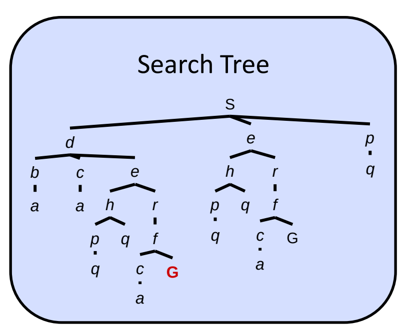
- A “what if” tree of plans and their outcomes
- The start state is the root node
- Children correspond to successors
- Nodes show states, but correspond to PLANS that achieve those states. The plan is the path from the root to the node
- For most problems, we can never actually build the whole tree

State Space Graphs vs. Search Trees



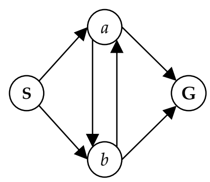
Each NODE in in the search tree is an entire PATH in the state space graph.

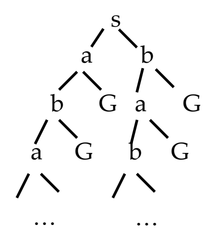
We construct both on demand – and we construct as little as possible.



State Space Graphs vs. Search Trees

Consider this 4-state graph: How big is its search tree (from S)?





Important: Lots of repeated structure in the search tree!

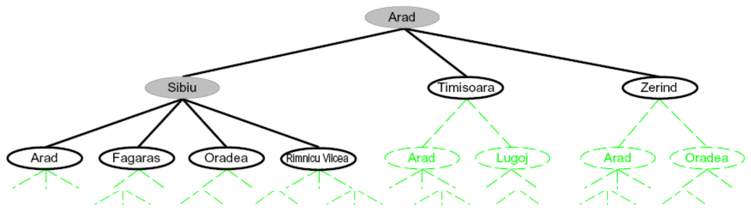
Big Picture: *Big Problems*

- Artificial Intelligence Problems generally involve huge state spaces.
- Both the state space graph and the state space tree are too large to fit in memory

Solving Search Problems

- **Solution:** A sequence of actions that leads from the initial state to the goal state.
- **Optimal Solution: Solution with lowest path cost among all solutions.**
- **In a search process**, data is often stored in a **node**, a data structure that contains the following data:
  - A ***state***
  - Its ***parent node***, through which the current node was generated
  - The ***action*** that was applied to the state of the parent to get to the current node
  - The ***path cost*** from the initial state to this node

Idea of Searching with a Search Tree



- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

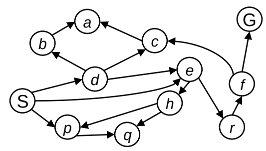
General Tree Search

```
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: possible nodes to explore next
  - Expansion: when node explored – determine neighbors to explore
  - Exploration strategy: which fringe node to pick e.g. DFS, BFS,

Depth-First Search

Example: Tree Search ???

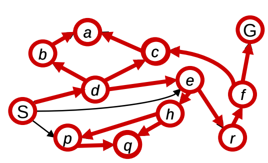


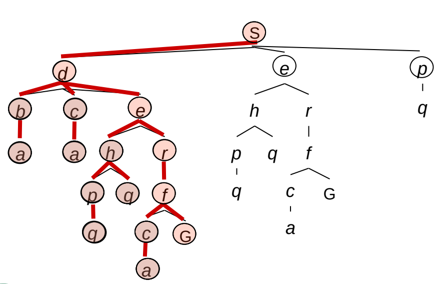
29

Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





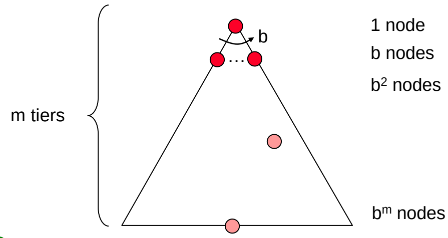
s
s → d
s → e
s → p
s → d → b
s → d → c
s → d → e
s → d → e → h
s → d → e → r
s → d → e → r → f
s → d → e → r → f → c
s → d → e → r → f → G

30

Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

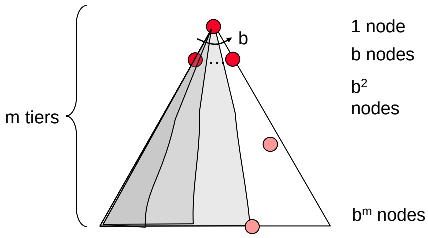
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots. b^m = O(b^m)$

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 $O(b^m)$



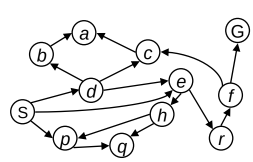
- How much space does the fringe take?
  - Only has siblings on path to root, so O(bm)
- Is it complete?
  - m could be infinite, so only if we prevent cycles (more later)
- Is it optimal?
  - No, it finds the “leftmost” solution, regardless of depth or cost

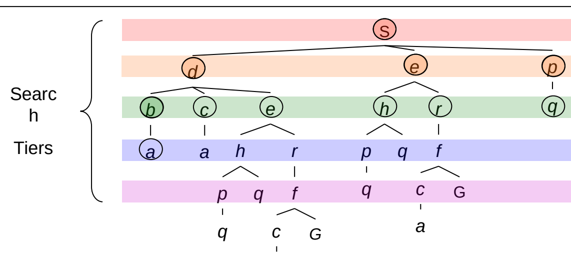
Breadth-First Search

Breadth-First Search

Strategy: expand a shallowest node first

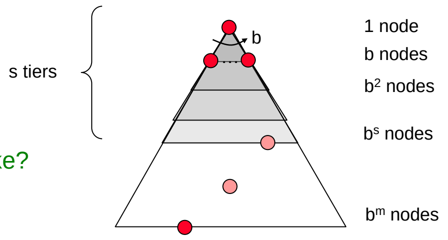
Implementation: Fringe is a FIFO queue





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 $O(b^s)$
- Is it complete?
  - s must be finite if a solution exists
- Is it optimal?
  - Only if costs are all 1 (more on costs later)

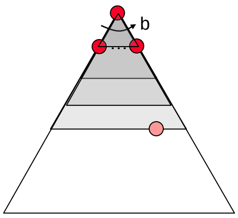


DFS vs BFS?

- When will BFS outperform DFS?
- When will DFS outperform BFS?

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!