AI基础

Lecture 2： Problem Solving: Search

Bin Yang
School of Data Science and Engineering
byang@dase.ecnu.edu.cn

[Some slides adapted from Nikita Kitaev, UCB]

Lecture 1 ILOs

	Human	Rational
Behavior	Act like people Acting Humanly The Turing test approach	Act rationally The rational agent approach
Thought	Think like people Thinking humanly The cognitive modeling approach	Think rationally The “laws of thought” approach, Logic

- What is AI?
  - Understand the four approaches
    - Human vs. rationality
    - Behavior vs. thought
  - Beneficial machines
    - Value alignment, AI risks
- AI history
- Agents
  - Key concepts: agent, environment, agent function, agent program, rational agent
  - Performance measure and rational agent
    - Maximize expected performance measure
  - Task environments, and their categorization
    - PEAS, Fully/Partially observable, single-/multi-agent, (non)deterministic, episodic vs. sequential, static vs. dynamic, discrete vs. continuous, and (un)known.
  - Agent programs, and their categorization
    - Simple reflex agents, model-based reflex agents, Goal-based agents, Utility-based agents
    - Learning agents

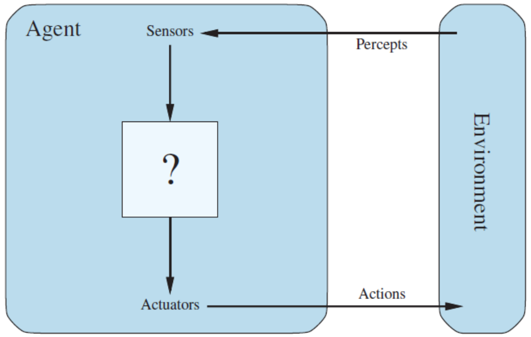


Figure 2.1 Agents interact with environments through sensors and actuators.

Lecture 2 ILOs

- Problem-solving agent
  - What is a problem-solving agent
  - Search problems and solutions
- Example problems
  - Standardized problems
  - Real-world problems
- Search algorithms
  - Uninformed search vs. informed search
  - Performance measure: completeness, optimality, time complexity, space complexity
  - Best-First-Search
- Uninformed search
  - Breath-First-Search
  - Uniform cost search/Dijkstra's algorithm
  - Depth-First-Search
  - Iterative deepening search
  - Bidirectional search

Outline

- Problem-solving agent
- Search problem and solutions
- Uninformed Search Algorithms

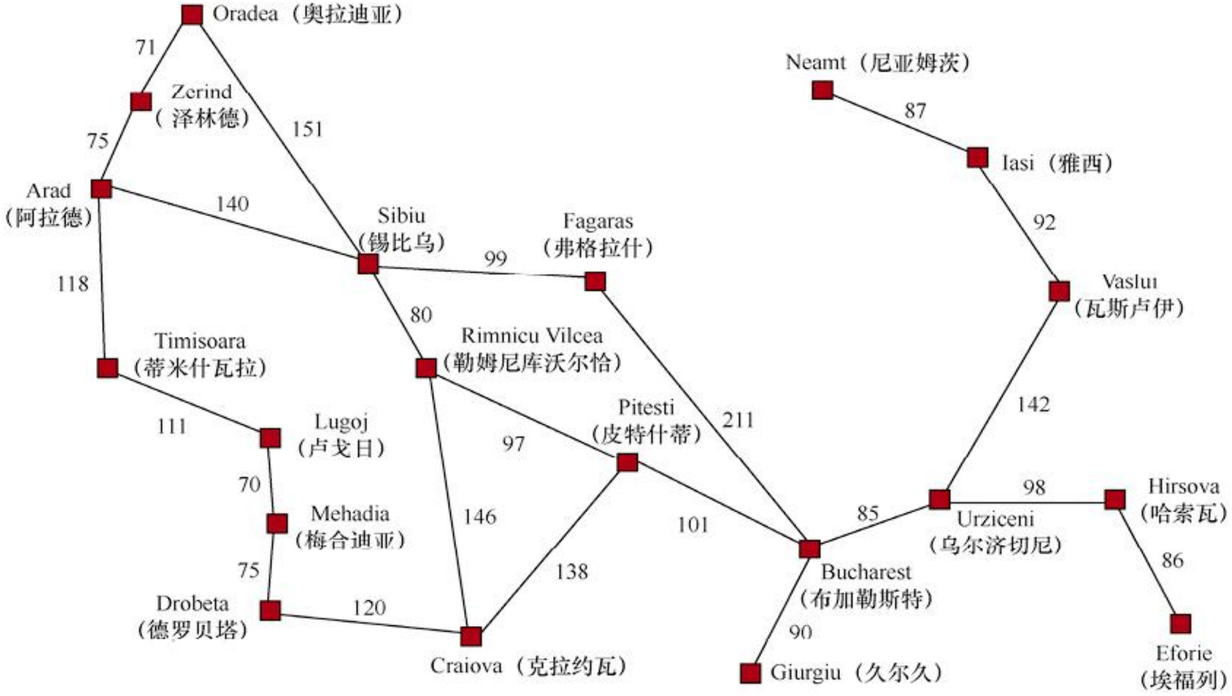
Solving problems by searching

- When it is useful?
  - When the correct action to take is not immediately obvious
- Problem-solving agent
  - An agent may need to plan ahead: a sequence of actions that form a path to a goal state
- Search
  - Computational process that a problem-solving agent takes
  - Informed searching vs. uninformed searching
    - Whether the agent can estimate how far it is from the goal

- Environment, simplest
  - Episodic, single agent, fully observable, deterministic, static, discrete, and known.
  - We will also show a motivating example on more difficult environments.

An example

- Arad to Bucharest
  - Known: with map
    - Three roads leaving Arad
  - Unknown: no map



Problem-solving agent

- Goal formulation
  - Bucharest.
  - Goals organize behavior by limiting the objectives and hence the actions to be considered.
- Problem formulation
  - States and actions necessary to reach the goal
- Search
  - The agent simulates sequences of actions in its model, searching until it finds a sequence of actions that reaches the goal. This is called a solution.
- Execution
  - Execute the actions in the solution

Open-loop and closed-loop system

- Fully observable, deterministic, and known environment
  - The solution to any problem is a **fixed sequence of actions**
  - **Open-loop**, ignore percepts while executing breaks the loop between agent and environment.
  - Example: shortest path

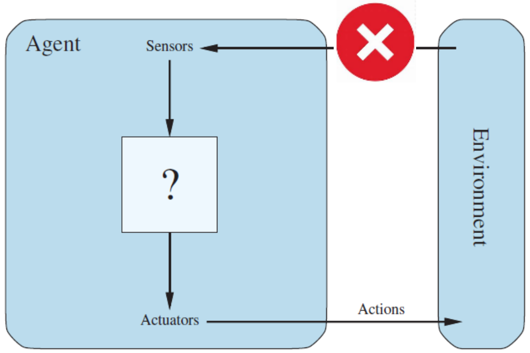
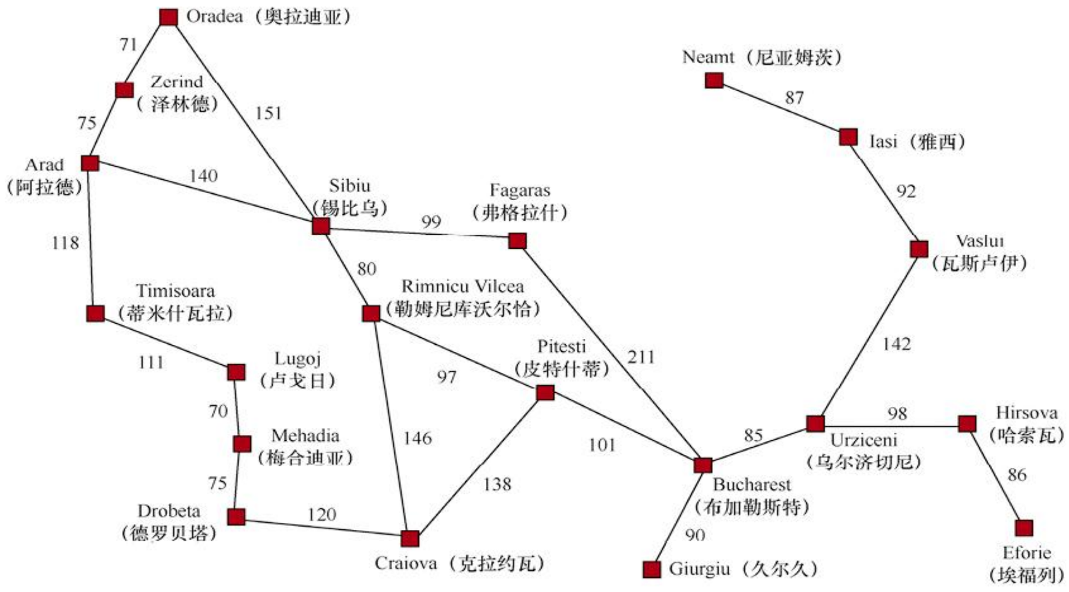


Figure 2.1 Agents interact with environments through sensors and actuators.

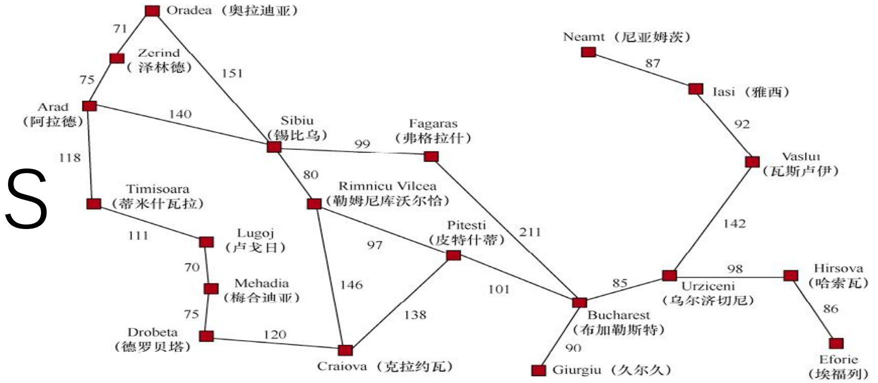
- Partially observable, non-deterministic
  - Actions depends on what percepts arrive
  - Example: traffic based fastest path
  - The solution is a **strategy**
    - Different future actions based on percepts
  - **Closed-loop**



Outline

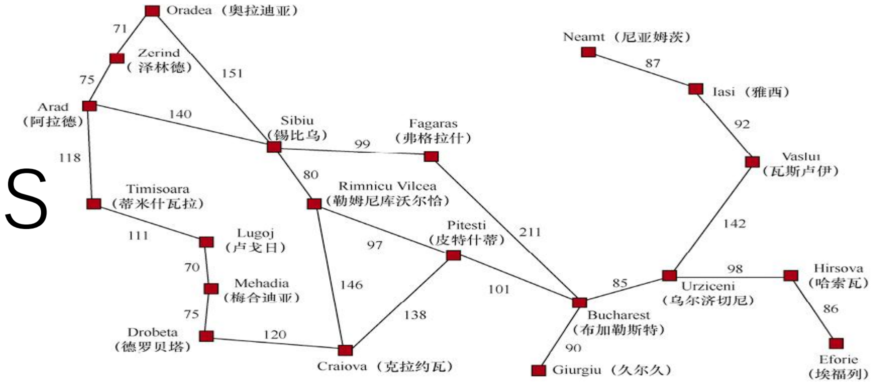
- Problem-solving agent
- Search problem and solutions
- Uninformed Search Algorithms

Search problems and solutions



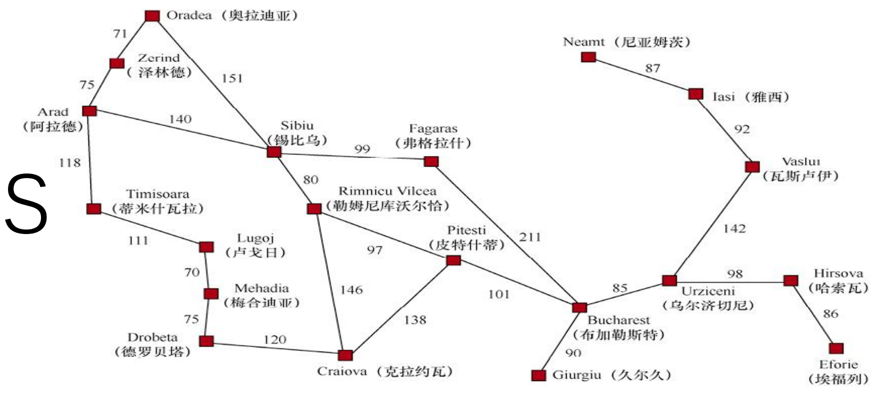
- State space
  - A set of possible states that the environment can be in.
- Initial state
  - Where the agent starts.
  - E.g., Arab
- Goal states
  - One goal vs. many goals
  - E.g., Bucharest vs. no dirt in any location (vacuum-cleaner)
  - Is-Goal method

Search problems and solutions



- Actions
  - The actions available to the agent at state s.
  - Actions(s) returns a finite set of actions that can be executed in state s.
  - Action(Arab) = {ToSibiu, ToTimisoara, ToZerind}
- Transition model
  - Result(s, a): returns the state from doing action a in state s.
  - Result(Arab, ToZerind) = Zerind
- Action cost function
  - Action-Cost(s, a, s’): numeric cost of applying action a in state s to reach a new state s’
  - Action-Cost(Arab, ToZerind, Zerind)=75

Search problems and solutions



- The state space can be modeled as a graph
  - **State-space-graph**
  - Nodes: states
  - Edges: actions
  - Cities, nodes, states
  - Roads, edges, actions (one road corresponds to two actions, one in each direction)

- Path: a sequence of actions
- Solution: a path from the initial state to a goal state
- Optimal solution: the solution with the lowest path cost
  - Assume action costs are additive. The cost of a path is the sum of the costs of the individual actions in the path.
  - This assumption may not be always true.

Example Problems

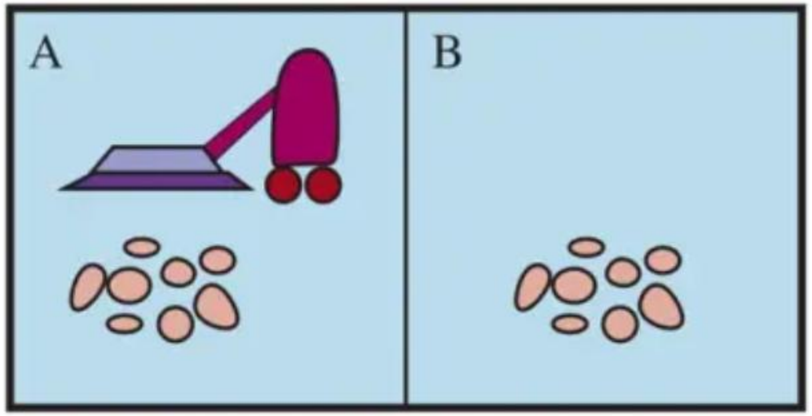
- Standardized problems vs. real-world problems
  - Concise, abstract, exact description. Suitable for benchmarking performance of different problem-solving algorithms
  - Whose solution people actually can use, formulation is idiosyncratic

- Standardized problems
  - Grid world, vacuum world
- Real-world problems
  - Routing-finding
  - Touring problems
  - VLSI layout
  - Robot navigation

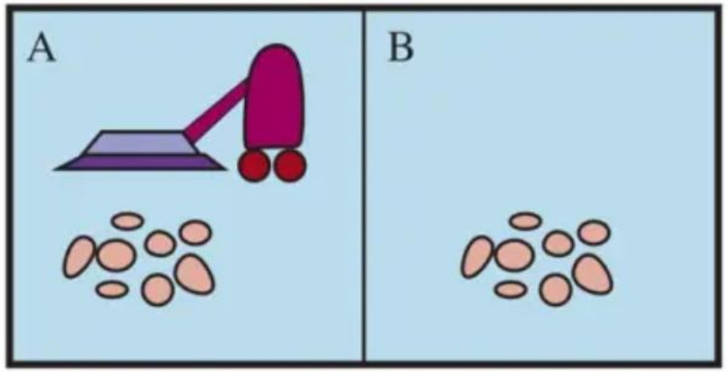
Grid world

- 2D rectangular array of cells
- An agent can move from cell to cell
- Cells contain objects
- The agent can operate on the objects

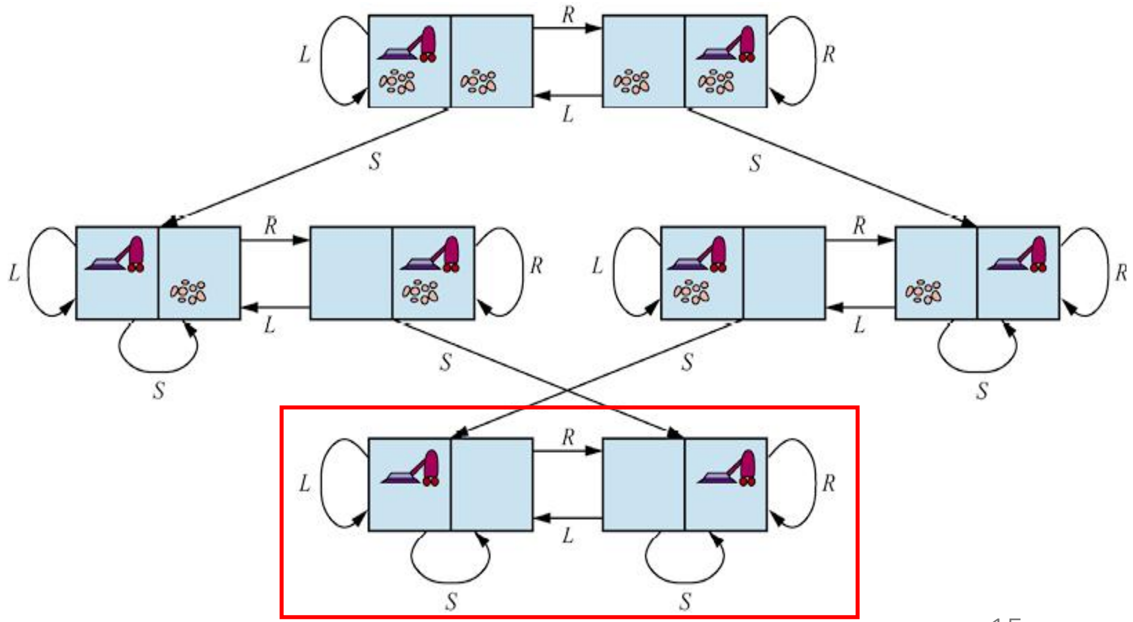
- Vacuum world



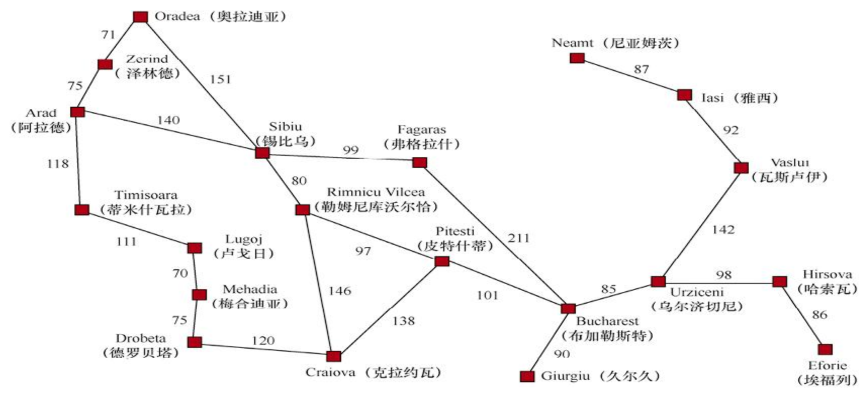
Vacuum world



- States:
  - Which objects (agent/dirt) are in which cells
  - The agent can be in one of the two cells
  - Each cell can either contain dirt or not
  - Quiz: how many states do we have?
  - 2 cells example: 8 states
  - n cells: $n*2^n$ states
- Initial state
  - Any state in the 8 states
- Actions
  - Suck, left, right
- Transition model
- Goal states: every cell is clean
- Action cost: constant
- State-space-graph
  - We can rarely build full graph in memory (it's too big) when n is big

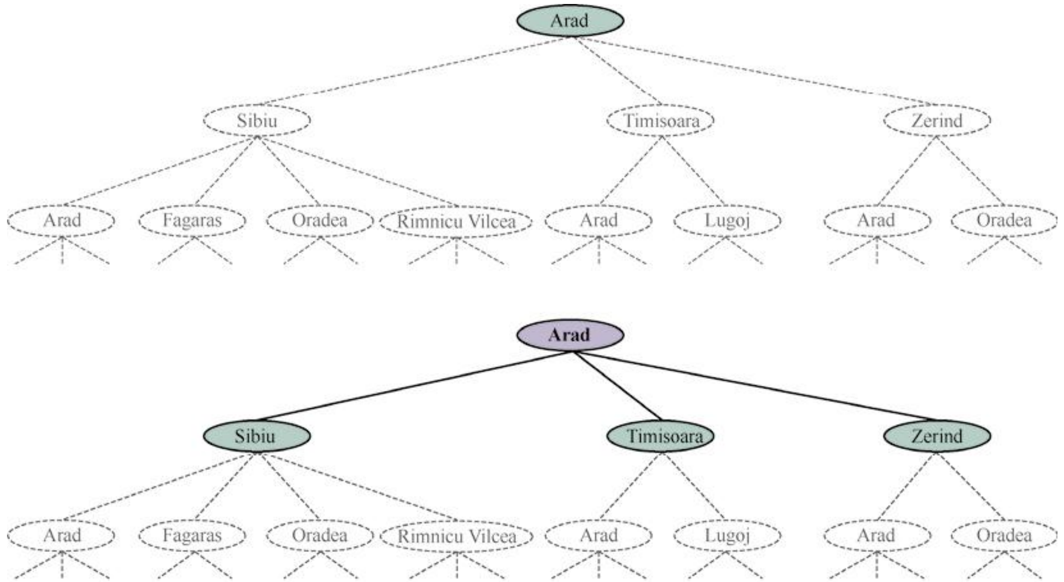


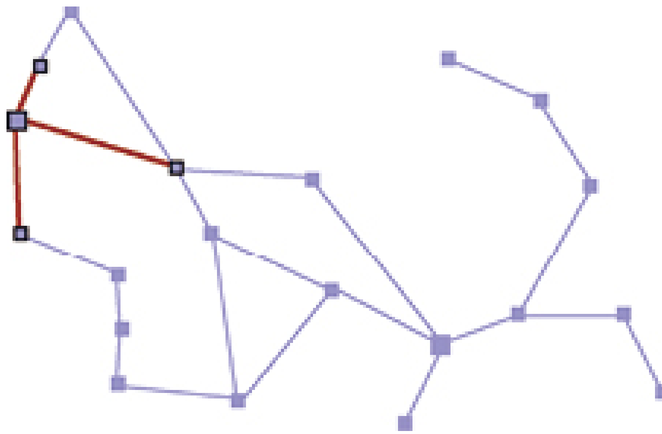
Search algorithms



- Input: a search problem
- Output: a solution, or an indication of failure (no solution exists)

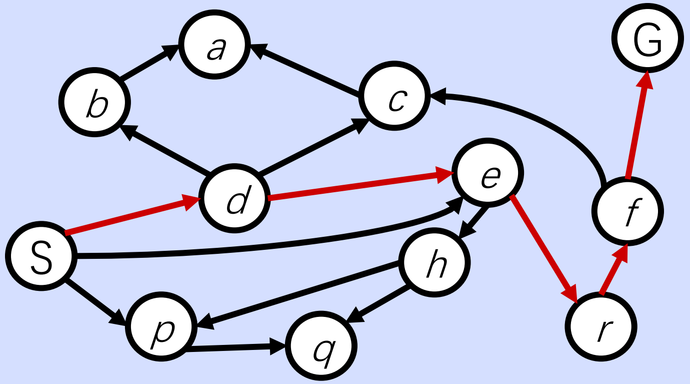
- Algorithms that superimpose a **search tree** over the state-space graph
  - Nodes: states
  - Edges: actions
  - Root: initial state





State Space Graphs vs. Search Trees

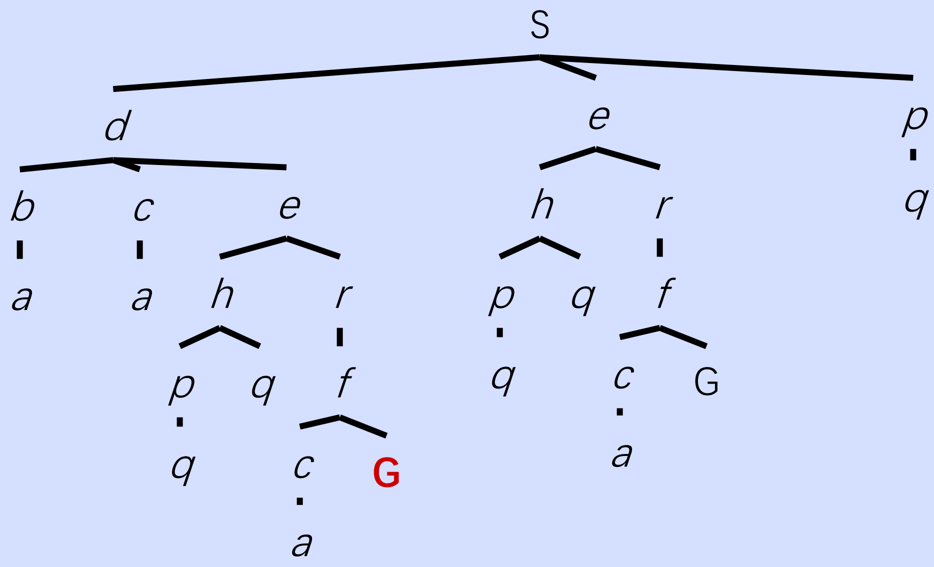
State Space Graph



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

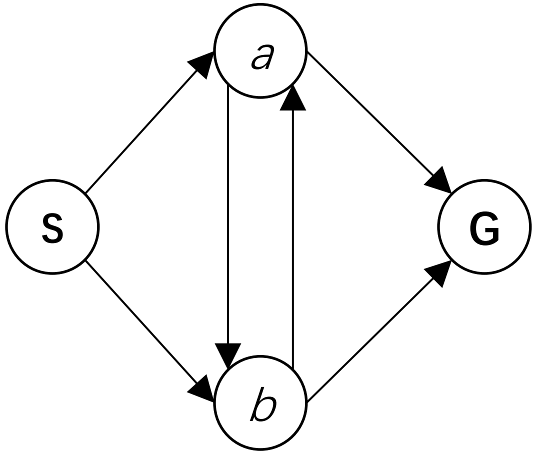
We construct both on demand – and we construct as little as possible.

Search Tree

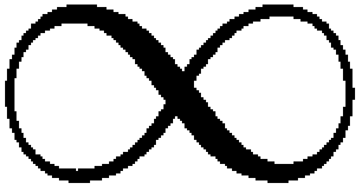


Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

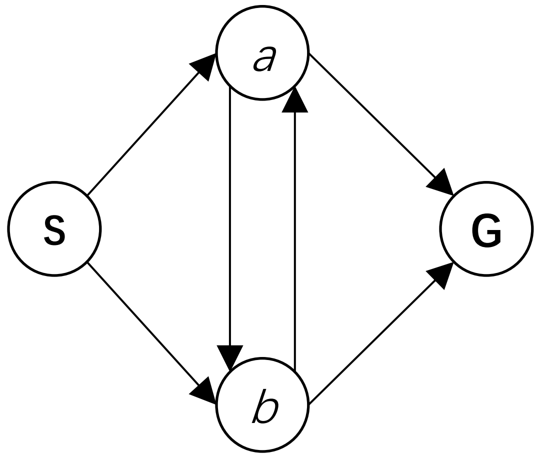


How big is its search tree (from S)?

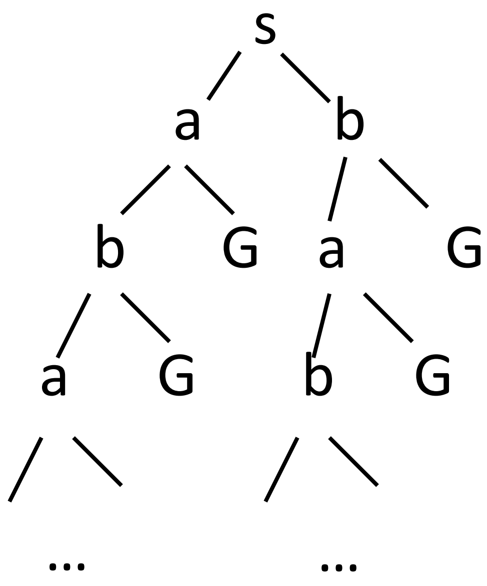


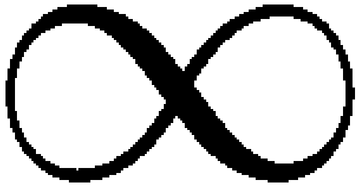
Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



How big is its search tree (from S)?

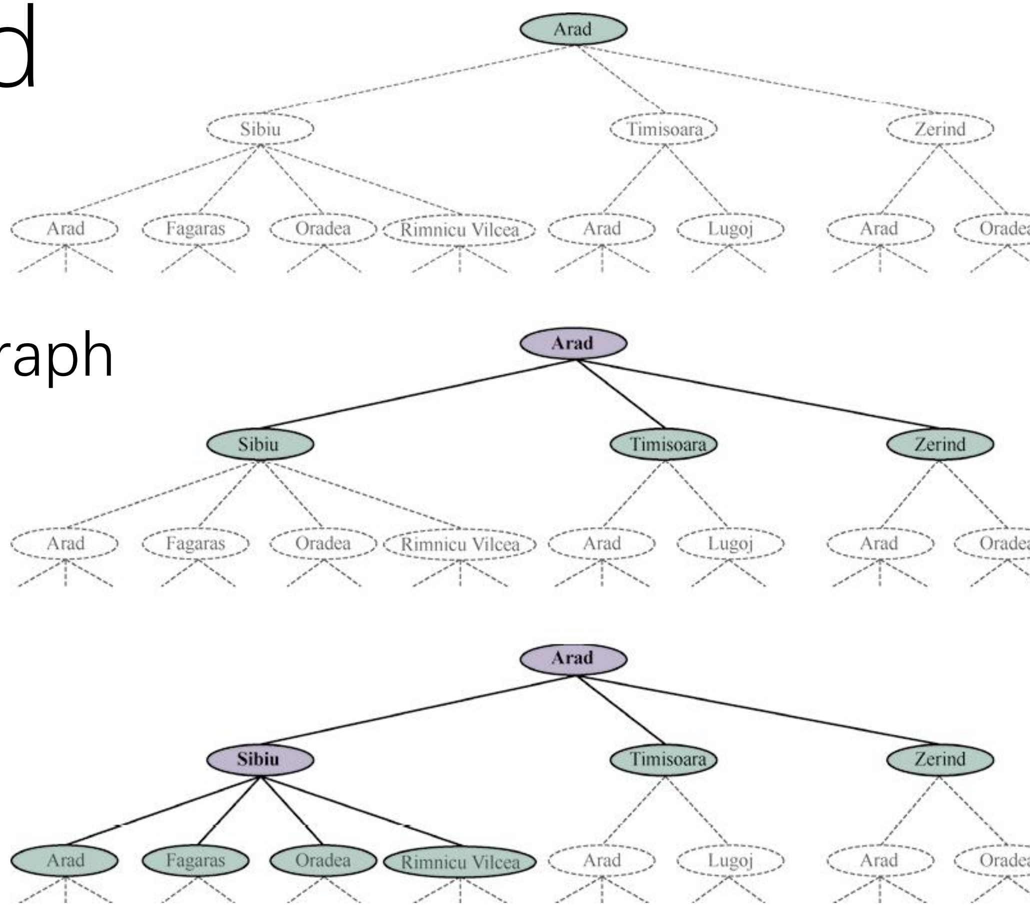


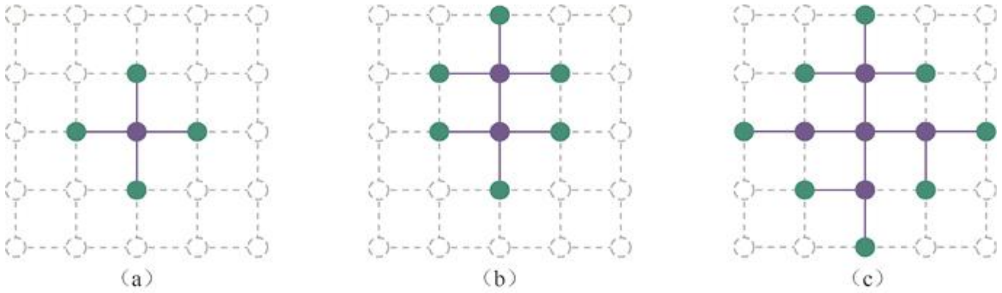


Important: Lots of repeated structure in the search tree!

Frontier, reached, expanded

- Frontier: reached, but not yet expanded
  - Separates two regions of the state-space graph
    - Interior region: expanded
    - Exterior region: not yet reached





Best-first search

- Which node from the frontier to expand next?
  - We choose a node n with the minimum value of evaluation function f(n)

- Each iteration
  - Choose a node on the frontier with minimum f(n) to expand/ stop
    - Priority queue
  - The node has not been reached before, or re-added

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s′ ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s′)
    yield NODE(STATE=s′, PARENT=node, ACTION=action, PATH-COST=cost)
```

Search data structures

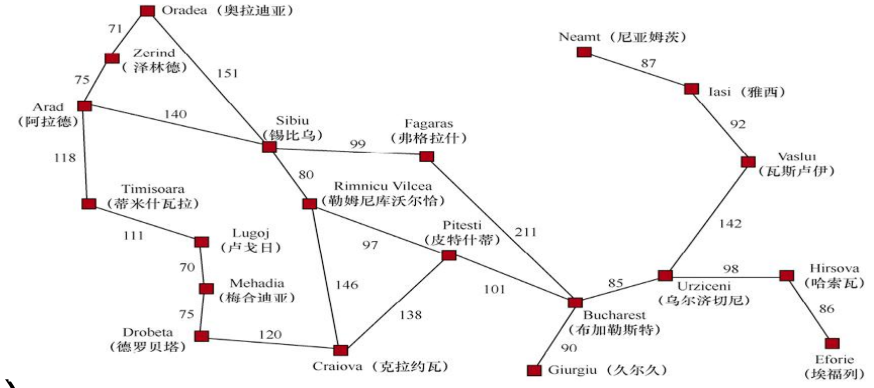
- Node
  - State
  - Parent: the node in the tree that generated this node
  - Action: the action applied on the parent node
  - Path-cost: total cost of path from the initial state to this node
- Frontier, maintain in a “queue”
  - Priority queue, best-first search
  - FIFO queue, breadth-first search
  - LIFO queue (stack), depth-first search

Measuring problem-solving performance

- Completeness
  - The algorithm is guaranteed to find a solution when there is one, and to correctly report failure when there is not.
- Cost optimality
  - Find a solution with the lowest path cost among all solutions.
- Time complexity
  - How long does it take to find a solution. In terms of the number of states and actions.
- Space complexity
  - How much memory is needed.

- Complexity
  - When the graph is an explicit data structure
    - Map: V, E
  - Implicit graph: transition model, actions
    - d, depth, or the number of actions in an optimal solution;
    - m, the maximum number of actions in any path;
    - b, branching factor, number of successors of a node that need to be considered

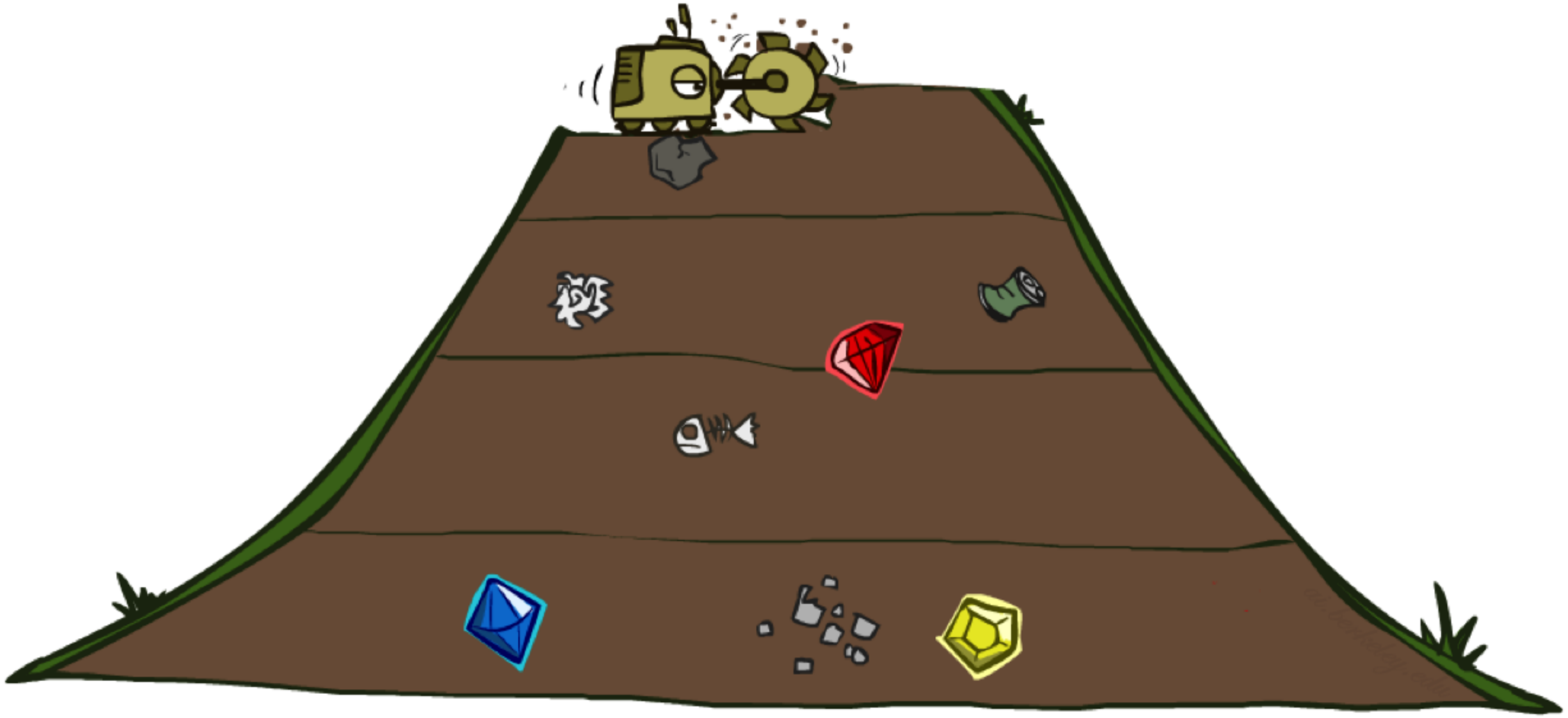
Uninformed search strategies



- No idea on how close a state is to the goal(s).

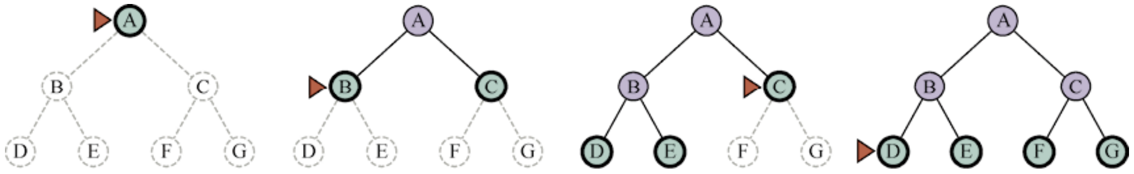
- Consider the example from Arab to Bucharest
- Uninformed agent
  - No knowledge of Romanian geography, Zerind or Sibiu?
- Informed agent
  - Knows the location of each city and knows Sibiu is closer to the goal.

Breadth-First Search (BFS)



Breadth-first search (BFS)

- When all actions have the same cost
- Root node is expanded
- Then, all the successors of the root node are expanded
- Then, the successors of the successors are expanded

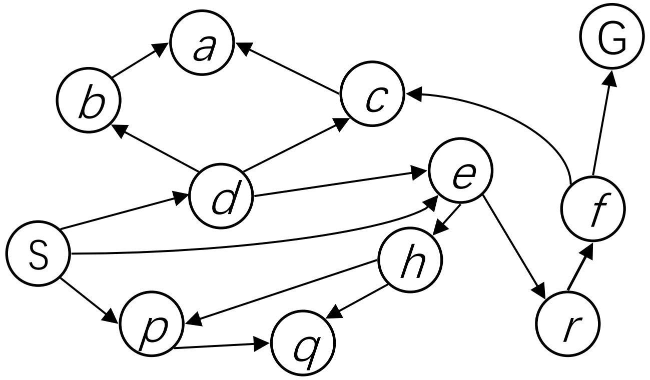


- When using the Best-First-Search, f(n) is the depth of the node
- More efficient implementation of BFS is to use a FIFO queue.

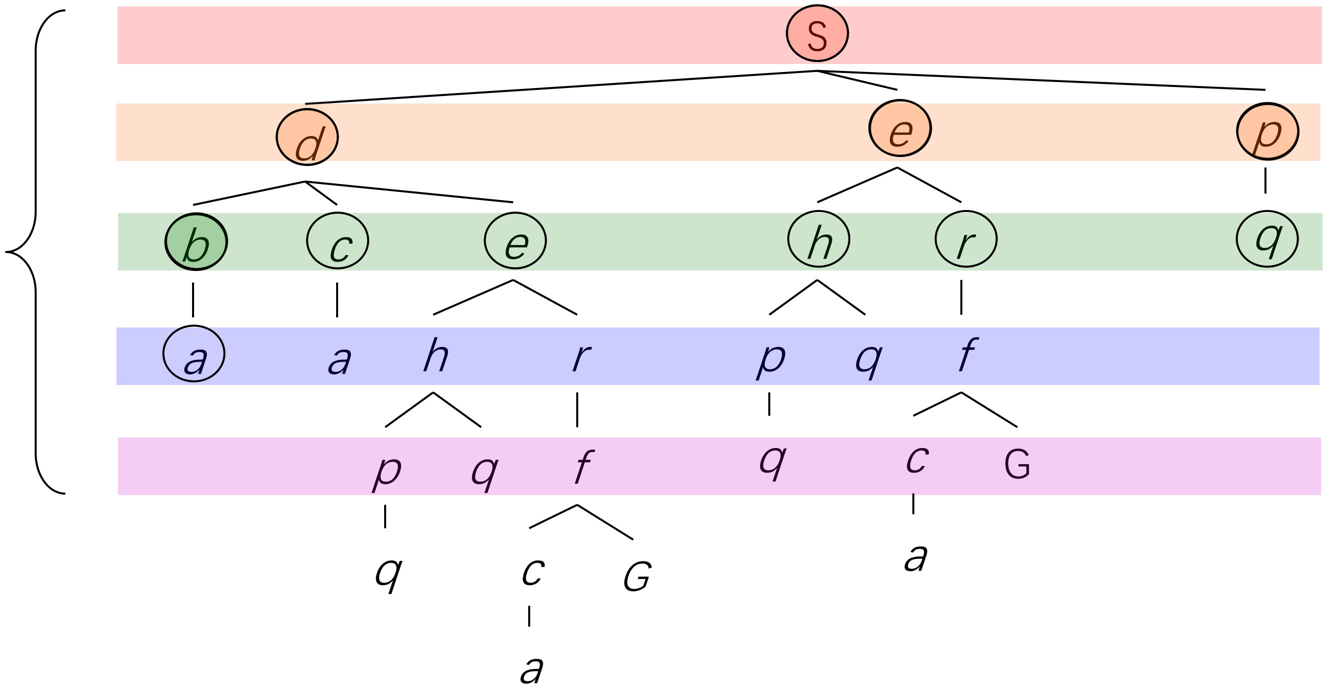
Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Frontier is a FIFO queue



Search Tiers



Search Algorithm Properties

- Complete: Guaranteed to find a solution if o
- Optimal: Guaranteed to find the least cost pa
- Time complexity?
- Space complexity?

Geometric series

For real $x \neq 1$, the summation

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n$$

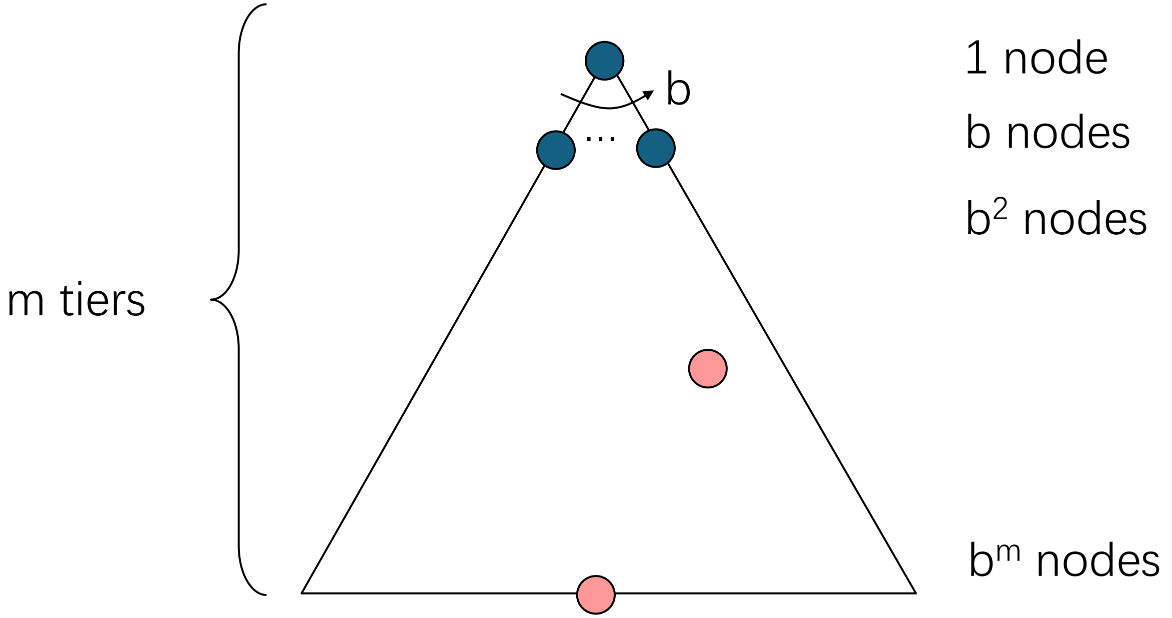
is a ***geometric*** or ***exponential series*** and has the value

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1} . \qquad \text{(A.5)}$$

When the summation is infinite and $|x| < 1$, we have the infinite decreasing geometric series

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x} . \qquad \text{(A.6)}$$

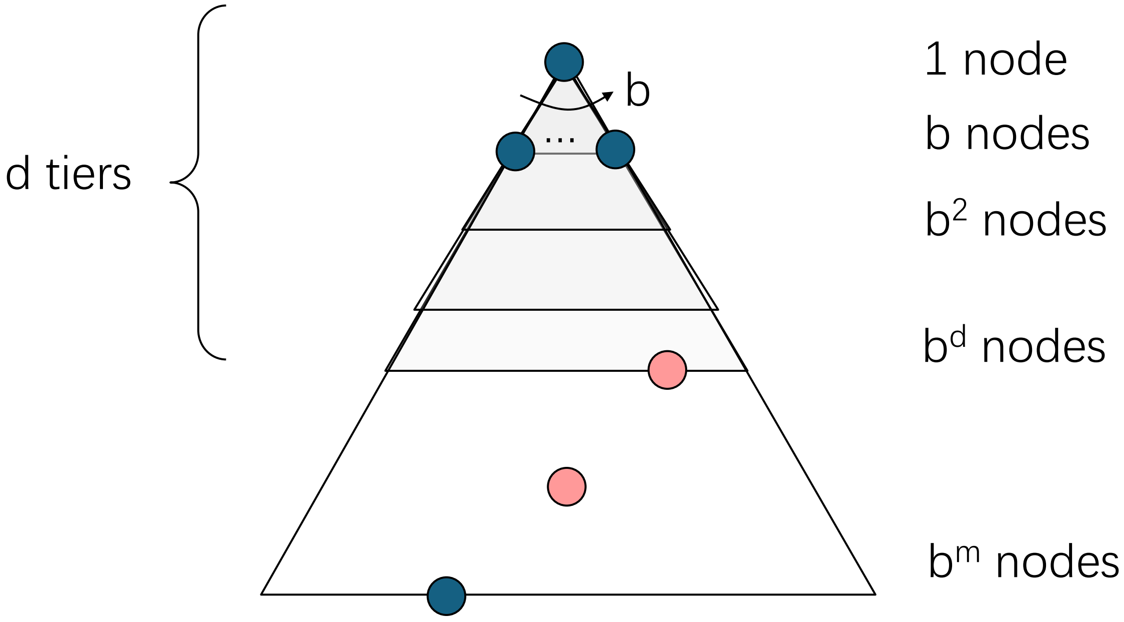
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



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

Breadth-First Search (BFS) Properties

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



Dijkstra's algorithm (Theoretical CS)
Uniform-cost search (AI)

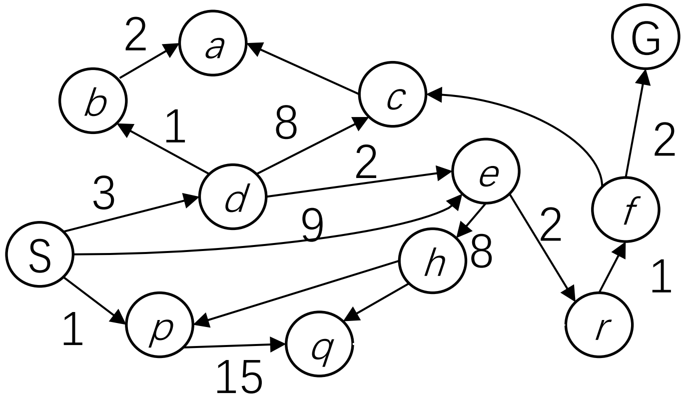
- When actions have different costs
- Best-first-search: f(n) is the cost of path from the root to the current node n.

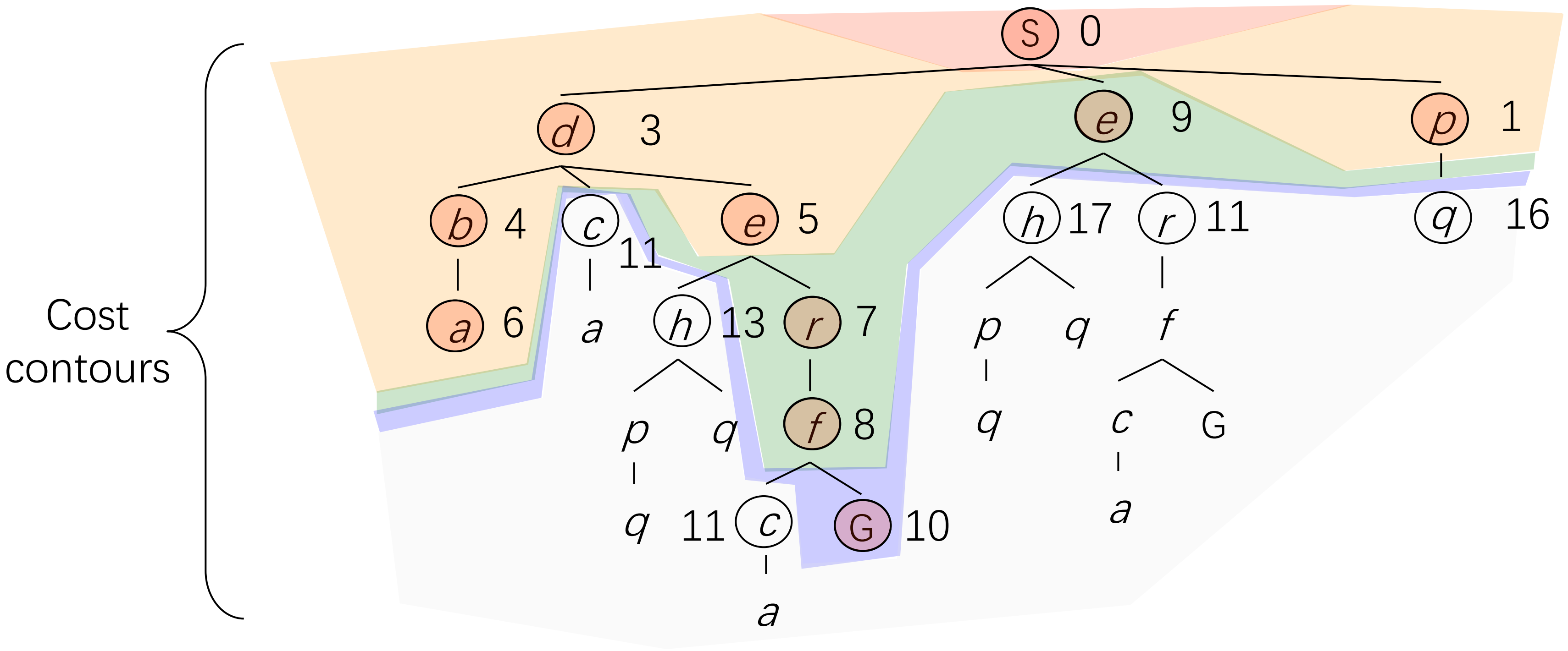
- Breadth first search
  - First depth 1, second depth 2, third depth 3, …
  - When exploring nodes in depth d, all nodes with depth <d have been explored.
- Uniform cost search
  - When exploring nodes with cost c, all nodes with cost <c have been explored.

Uniform Cost Search

Strategy: expand a cheapest node first

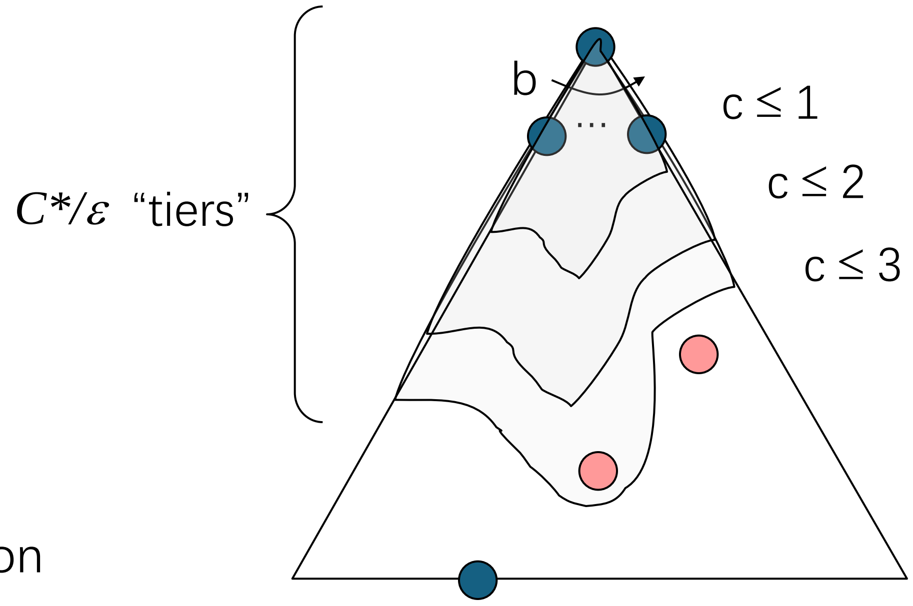
Frontier is a priority queue (priority: cumulative cost)





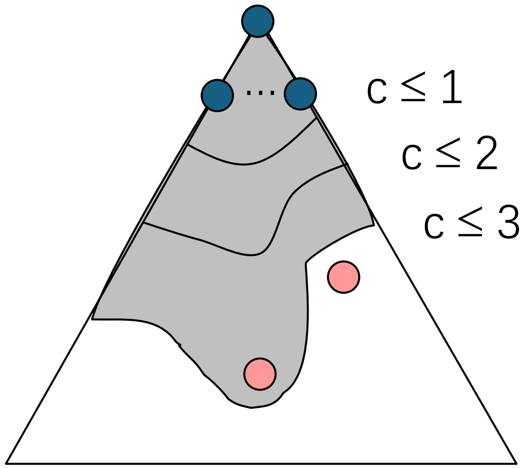
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and action cost at least ε , then the "effective depth" is roughly C^*/ε
  - Takes time O($b^{C^*/\varepsilon}$) (exponential in effective depth)
- How much space does the fringe take?
  - Has roughly the last tier, so O($b^{C^*/\varepsilon}$)
- Is it complete?
  - Assuming best solution has a finite cost and minimum action cost is positive, yes!
- Is it optimal?
  - Yes!
  - The first solution it finds will have a cost that is at least as low as the cost of any other node in frontier.



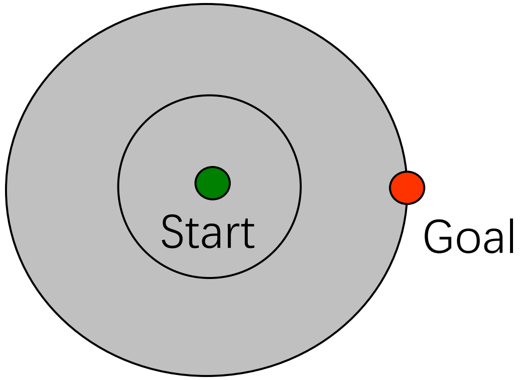
Uniform Cost Issues

- Remember: UCS explores increasing cost contours

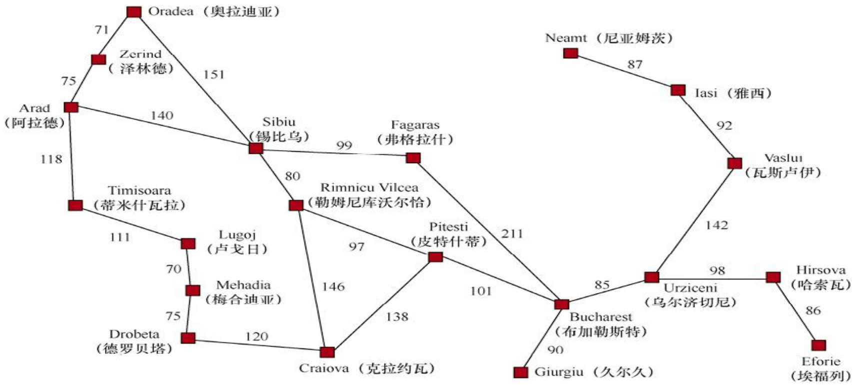


- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every “direction”
  - No information about goal location

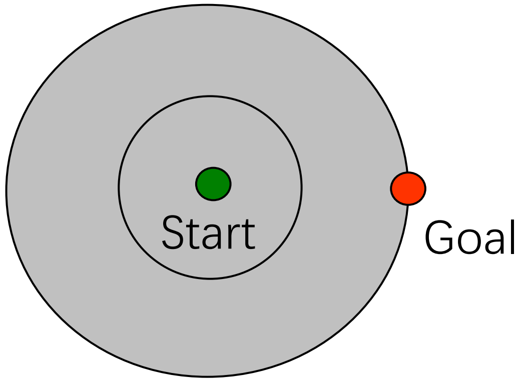


- We’ll fix that soon!

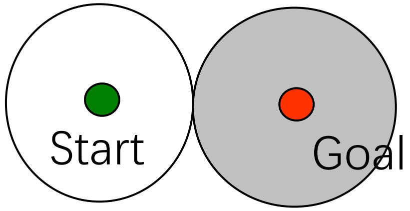


Bidirectional search

- Search forward from the initial state and search backwards from the goal state(s).
- When the two frontiers collide, we have a solution.
- Grow a ball around start and goal when goal is scanned
- Grow a ball around each end (start and goal) until they "meet".
- Can be used with other speed-up methods, e.g., contraction hierarchies.

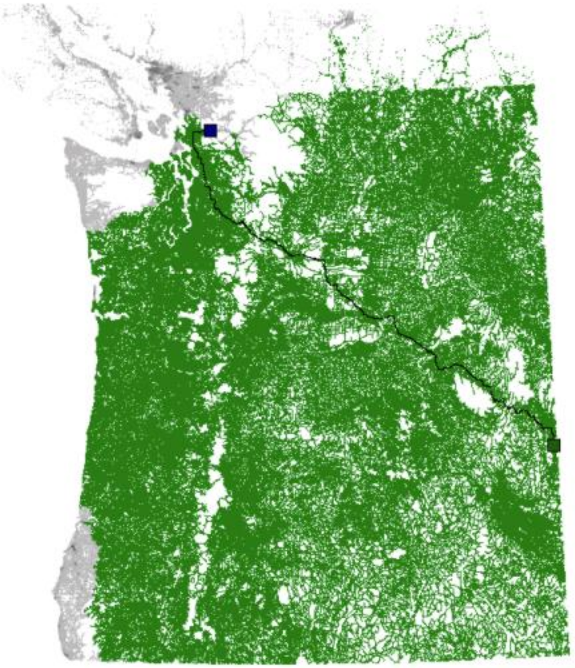


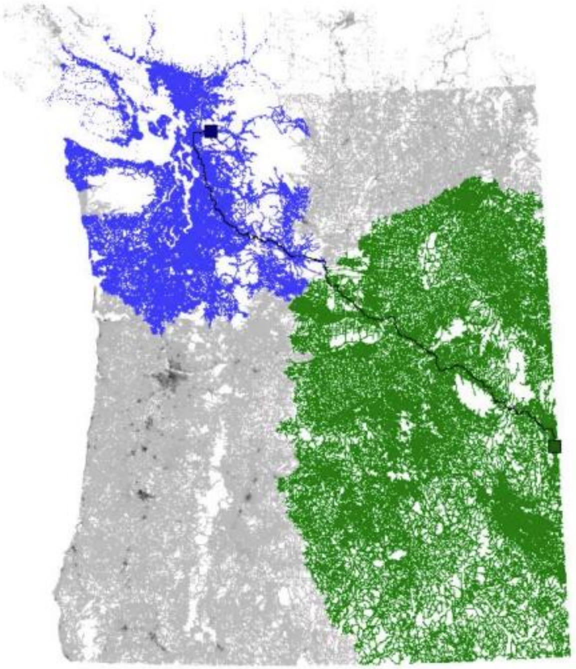
$O(b^d)$



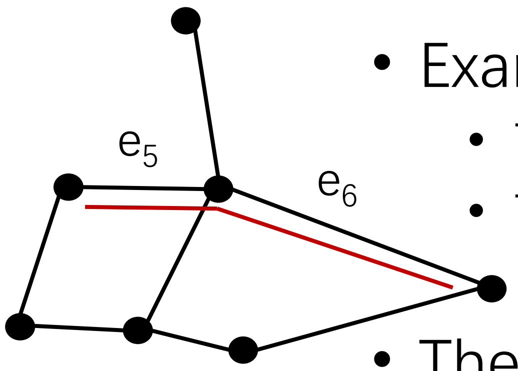
$O(b^{d/2})$

Bidirectional search





Uncertainty, episodic vs. sequential, deterministic vs. nondeterministic



- Example: $P=<e_5, e_6>$
  - Trajectory 1: 10 s, 20 s
  - Trajectory 2: 15 s, 25 s

e_5

Cost	Prob
10	0.5
15	0.5

e_6

Cost	Prob
20	0.5
25	0.5

- The distribution of a path is computed by summing the distributions of edges while assuming they are intendent.

Independence

$<e_5, e_6>$

Cost	Prob
10, 20	0.25
15, 20	0.25
10, 25	0.25
15, 25	0.25

Joint distribution

P

Cost	Prob
30	0.25
35	0.50
40	0.25

Dependence

$<e_5, e_6>$

Cost	Prob
10, 20	0.5
15, 25	0.5

Path-centric weight

P

Cost	Prob
30	0.5
40	0.5

Yang, Dai, Guo, Jensen, Hu. PACE: A PAth-CEntric Paradigm For Stochastic Path Finding. The VLDB Journal 27(2): 153-178 (2018).
Dai, Yang, Guo, Jensen, Hu. Path Cost Distribution Estimation Using Trajectory Data. PVLDB 10(3): 85-96 (2016).
Yang, Guo, Jensen, Kaul, Shang. Stochastic Skyline Route Planning Under Time-Varying Uncertainty. ICDE 2014, 136-147, 2014.

Depth-First Search

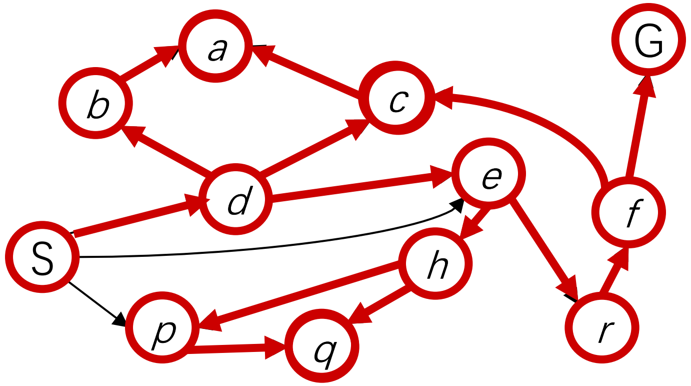
Depth-first search (DFS)

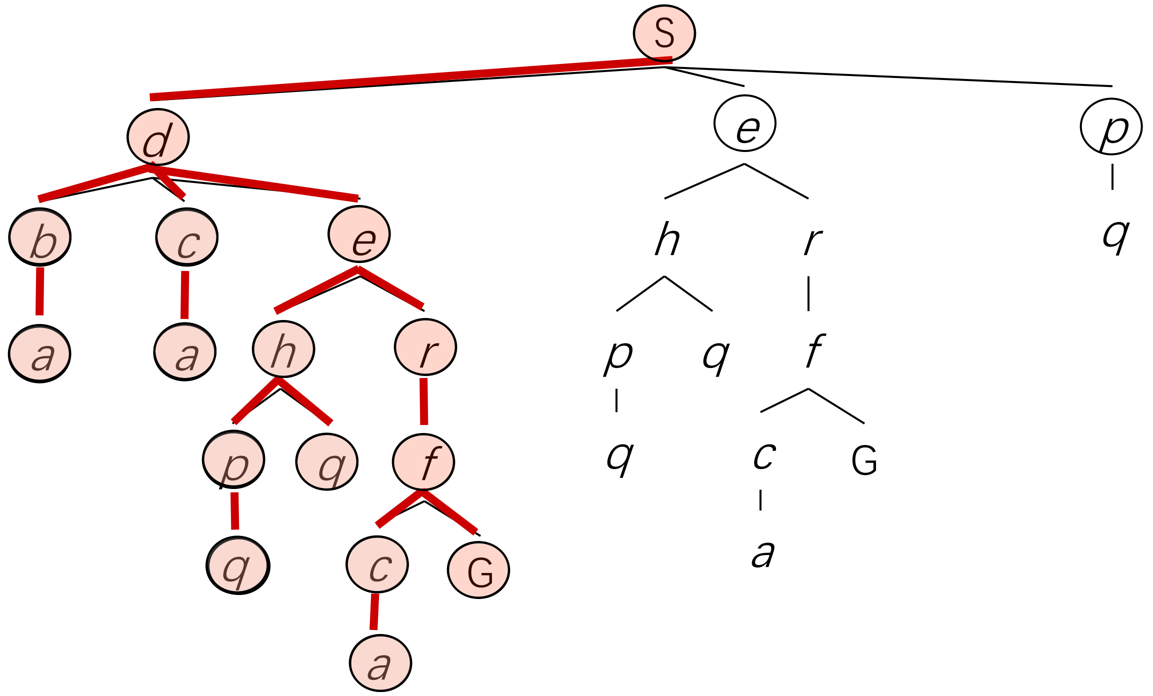
- When using the Best-First-Search, f(n) is the negative of the depth of the node
- Search proceeds immediately to the deepest level of the search tree, where the nodes have no successors.
    - Then, backtrack to the next deepest node that still has unexpanded successors.
- Not cost-optimal, it returns the first solution it finds.

Depth-First Search

Strategy: expand a deepest node first

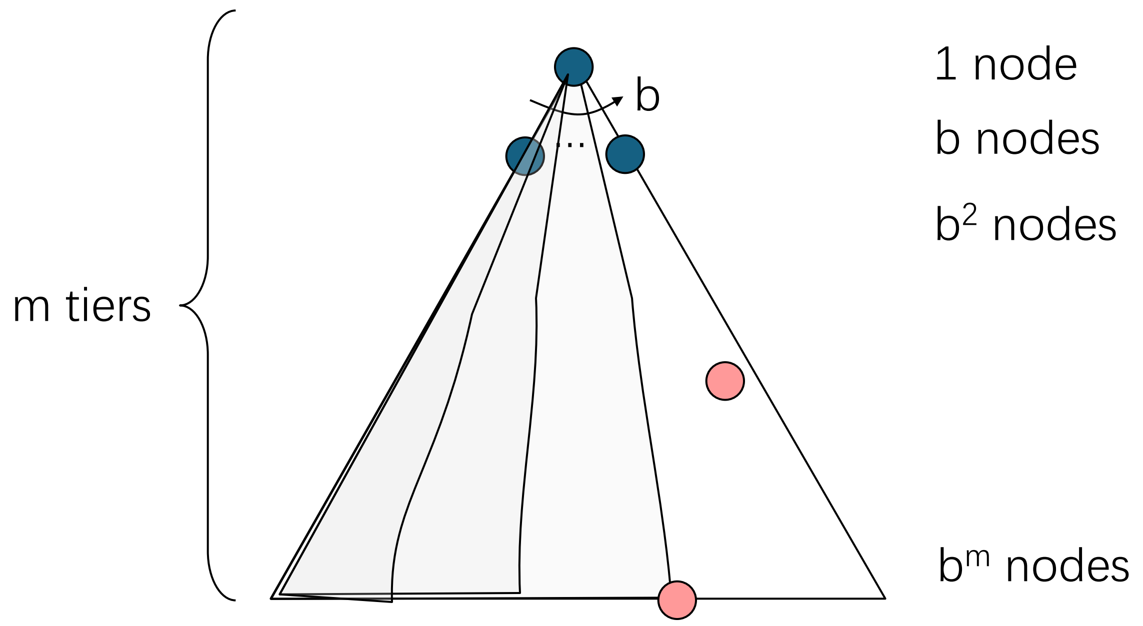
Implementation: Frontier is a LIFO stack

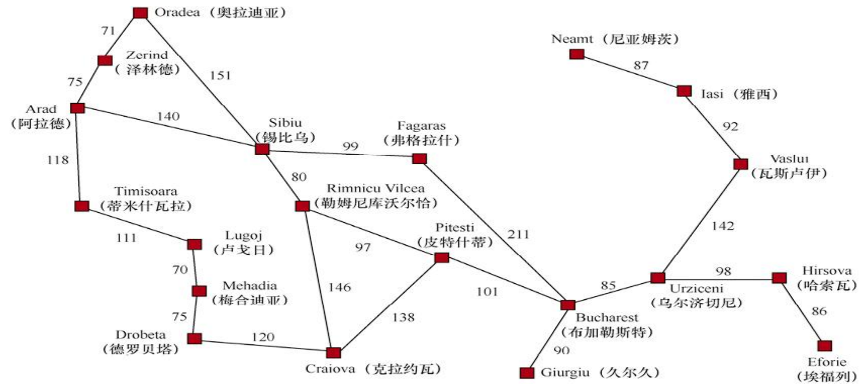




Depth-First Search (DFS) Properties

- 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 frontier take?
  - Only has siblings on path to root, so O(bm)
  - Good space complexity
- Is it complete?
  - m could be infinite, so only if we prevent cycles
- Is it optimal?
  - No, it finds the “leftmost” solution, regardless of depth or cost
- Not complete and not optimal, then why we care DFS?





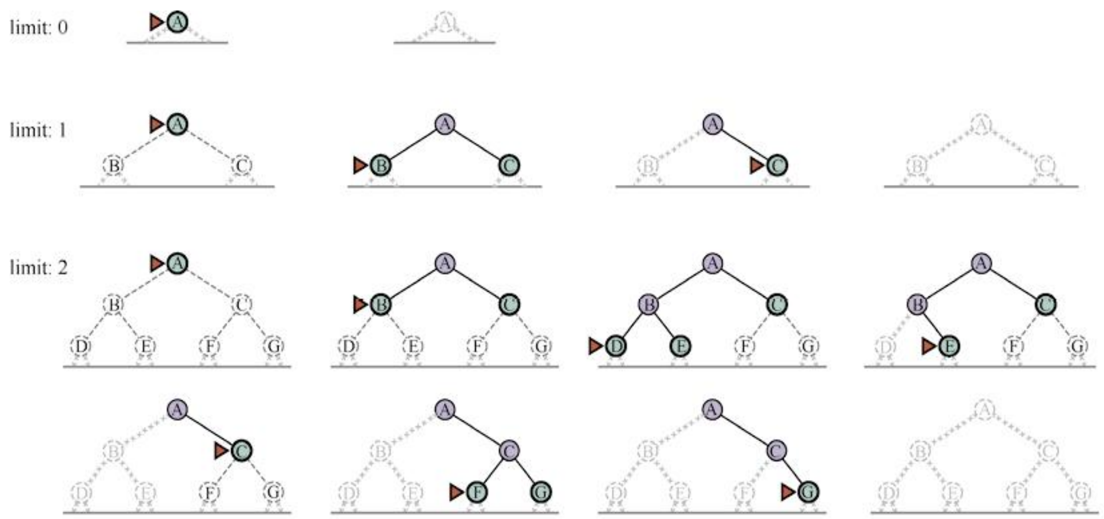
Depth-limited DFS

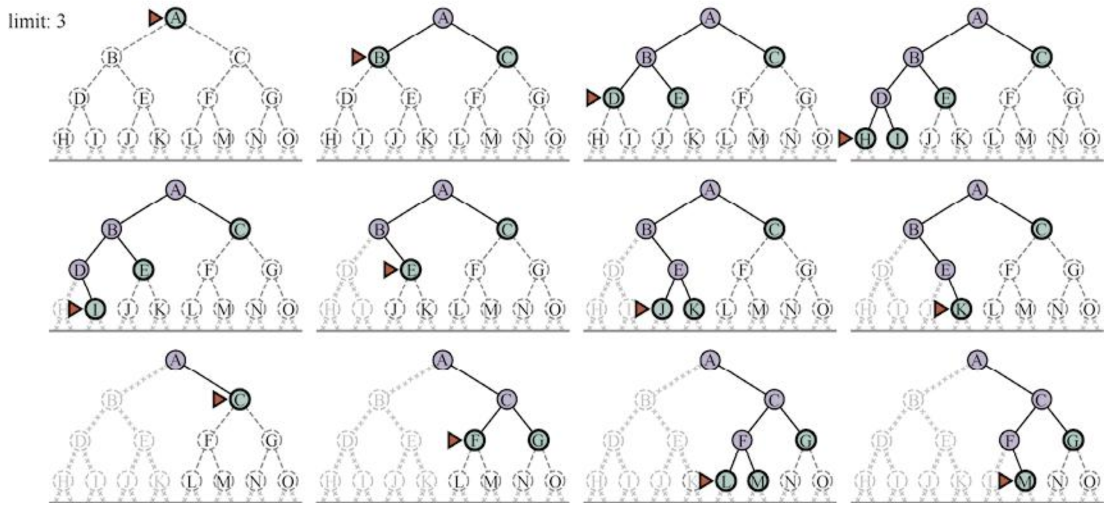
- Given a depth limit l, and treat all nodes at depth l as if they had no successors.
- Time complexity $O(b^l)$
- Space complexity $O(bl)$

- Romania map example
  - 20 cities in total, so l=19 is a valid limit.
  - One city can be reached from any other city in at most 9 actions. l=9.
  - Diameter of the state-space graph

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. …..

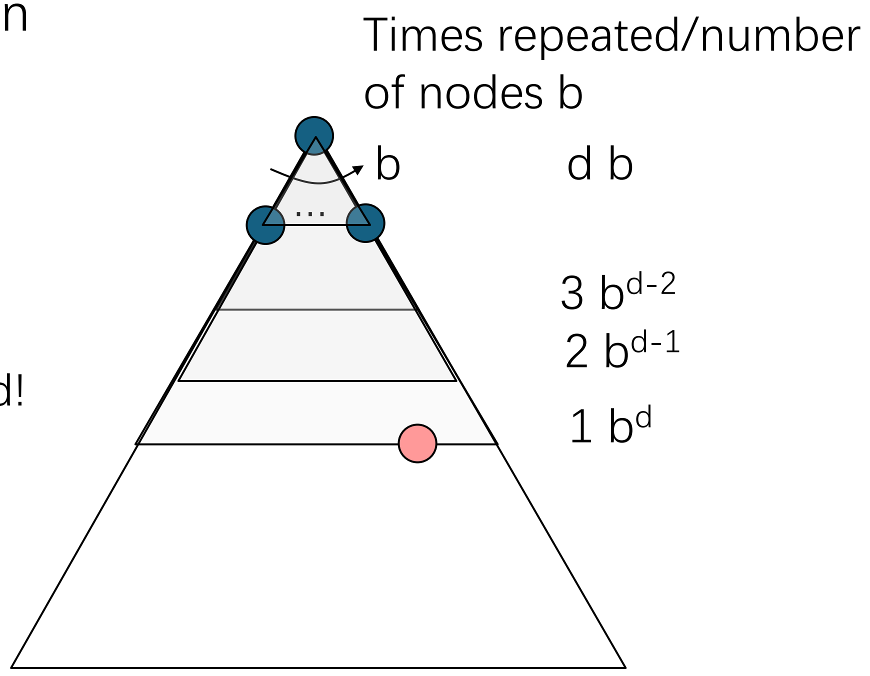




Iterative Deepening

- 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!
  - Time complexity : like BFS, still $O(b^d)$
  - $b=10$, $d=5$
  - BFS: $10+10^2+10^3+10^4+10^5=111{,}110$
  - IDS: $5*10+4*10^2+3*10^3+2*10^4+10^5=123{,}450$



- Space complexity, like DFS
- Optimal, if all actions have the same cost, like BFS
- Complete, finite acyclic state spaces

Uninformed Search Summary

- Solution costs $C*$
- b, branching factor, number of successors of a node that need to be considered
- d, depth, or the number of actions in an optimal solution;
- m, the maximum number of actions in any path;
- l, a depth limit

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Figure 3.15 Evaluation of search algorithms. b is the branching factor; m is the maximum depth of the search tree; d is the depth of the shallowest solution, or is m when there is no solution; ℓ is the depth limit. Superscript caveats are as follows: [1] complete if b is finite, and the state space either has a solution or is finite. [2] complete if all action costs are $\geq \epsilon > 0$; [3] cost-optimal if action costs are all identical; [4] if both directions are breadth-first or uniform-cost.

Lecture 2 ILOs

- Problem-solving agent
  - What is a problem-solving agent
  - Search problems and solutions
    - State space, initial state, goal states, action, transition model, action cost function
    - A solution: a path from the initial state to a goal state
- Example problems
  - Standardized problems
  - Real-world problems
- Search algorithms
  - Uninformed search vs. informed search
  - Performance measure: completeness, optimality, time complexity, space complexity
  - Best-First-Search: Which node from the frontier to expand next? Node with minimum f(n).
- Uninformed search
  - Breath-First-Search
  - Uniform cost search/Dijkstra’s algorithm
  - Bidirectional search
  - Depth-First-Search
  - Iterative deepening search