CSC520 - Artificial Intelligence
Lecture 4

Dr. Scott N. Gerard

North Carolina State University

Jan 16, 2025

Agenda

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

Problem Recap

Agent

- Function
- Program
- Architecture

Task Environment

- Fully vs. Partially Observable
- Deterministic vs. Stochastic
- Episodic vs. Sequential
- Static vs. Dynamic
- Discrete vs. Continuous
- Known vs. Unknown
- Single vs. Multi-agent

Search Problem

- State Space
- Initial State
- Goal State
- Actions
- Transition Model
- Action Cost Function

Roadmap

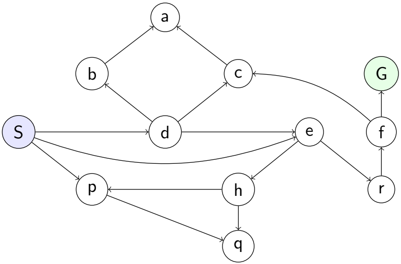
Ethics

Logic, Probability, Uncertainty, ...

Search

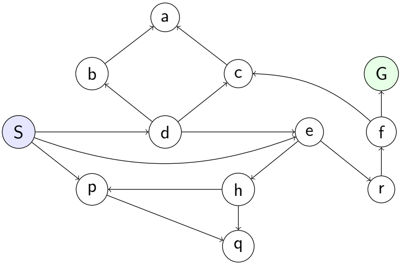
Problem

State Space vs Search Tree

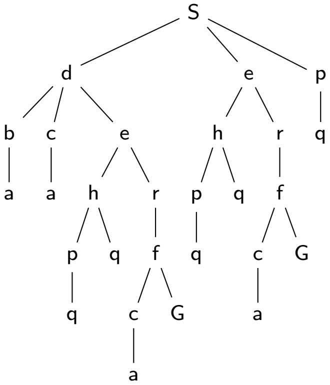


State Space

State Space vs Search Tree

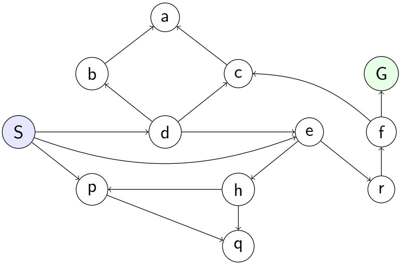


State Space

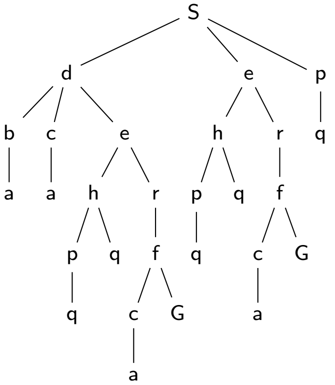


Search Tree

State Space vs Search Tree



State Space



Search Tree

- What if the state space has a cycle?

Uninformed Search

- No knowledge of how close a state is to the goal state
- Systematically expand the nodes from the start node
  - A path is *redundant* if it contains a cycle OR if a lower cost path exists
  - *Tree* search does not check for redundant paths
  - *Graph* search checks for redundant paths
- Frontier is the set of unexpanded nodes
- Structure of a search tree node
  - State
  - Parent
  - Action
  - Path cost

General Tree Search Algorithm

function TREE-SEARCH(*problem*, *strategy*) **return** a solution or failure
    Add the initial state to *frontier*
    while true **do**
        if *frontier* is empty **then**
            return failure
        else
            Remove a *node* for expansion from *frontier* according to *strategy*
            if *node* is a goal state **then**
                return solution
            else
                Expand *node* and add the resulting nodes to the frontier

- *strategy* = Different search algorithms based on how a node is selected for expansion

Uninformed Search

- Breadth-first search
  - Expand shallowest node first (FIFO queue)

Uninformed Search

- Breadth-first search
  - Expand shallowest node first (FIFO queue)
- Depth-first search
  - Expand deepest node first (LIFO queue or stack)

Uninformed Search

- Breadth-first search
  - Expand shallowest node first (FIFO queue)
- Depth-first search
  - Expand deepest node first (LIFO queue or stack)
- Uniform cost search (Dijkstra's algorithm)
  - Actions have different costs
  - Expand node with lowest path cost (priority queue)

Search Algorithm Properties

- Completeness
  - Return a solution if one exists or returns failure

Search Algorithm Properties

- Completeness
  - Return a solution if one exists or returns failure
  - In a finite state space, an algorithm that systematically explores every state is complete

Search Algorithm Properties

- Completeness
  - Return a solution if one exists or returns failure
  - In a finite state space, an algorithm that systematically explores every state is complete
  - In an infinite state space, an algorithm that can eventually reach every reachable state is complete

Search Algorithm Properties

- Completeness
  - Return a solution if one exists or returns failure
  - In a finite state space, an algorithm that systematically explores every state is complete
  - In an infinite state space, an algorithm that can eventually reach every reachable state is complete
- Cost optimality
  - Return a solution with lowest cost

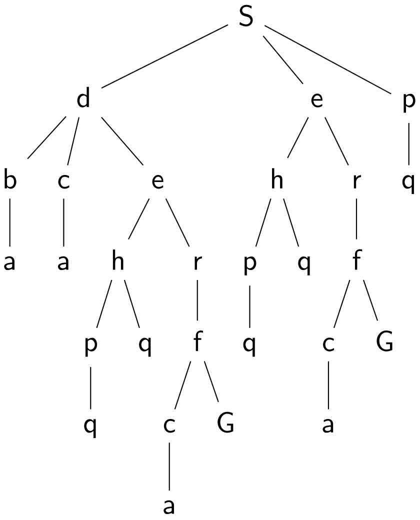
Search Algorithm Properties

- Completeness
  - Return a solution if one exists or returns failure
  - In a finite state space, an algorithm that systematically explores every state is complete
  - In an infinite state space, an algorithm that can eventually reach every reachable state is complete
- Cost optimality
  - Return a solution with lowest cost
- Time complexity
  - Time taken by the algorithm measured in terms of states and actions

Search Algorithm Properties

- Completeness
  - Return a solution if one exists or returns failure
  - In a finite state space, an algorithm that systematically explores every state is complete
  - In an infinite state space, an algorithm that can eventually reach every reachable state is complete
- Cost optimality
  - Return a solution with lowest cost
- Time complexity
  - Time taken by the algorithm measured in terms of states and actions
- Space complexity
  - Memory required by the algorithm measured in terms of states or nodes

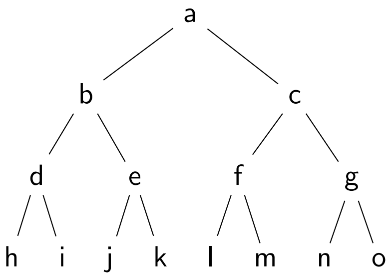
Breadth First Search



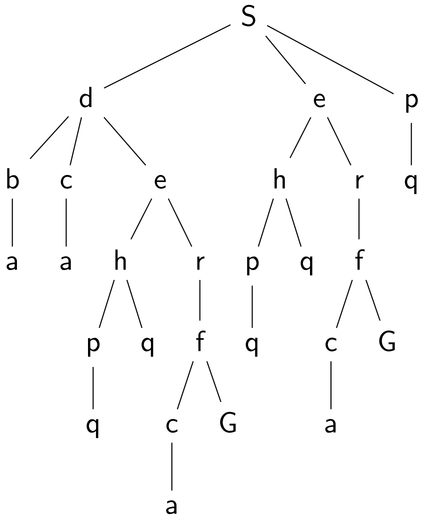
Expands shallowest node first

Breadth First Search Properties

- Completeness:
- Cost optimality:
- Time complexity
  - b is branching factor
  - d is depth of solution
  - $O(b^d)$
- Space complexity: $O(b^d)$



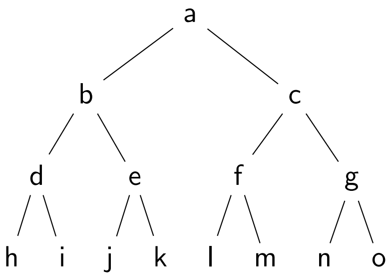
Depth First Search



Expands deepest node first

Depth First Search Properties

- Completeness:
- Cost optimality:
- Time complexity
  - b is branching factor
  - m is max depth
  - $O(b^m)$
- Space complexity: $O(bm)$



Iterative Deepening Search

- Combines benefits of BFS (completeness and optimality) and DFS (low space complexity)

Iterative Deepening Search

- Combines benefits of BFS (completeness and optimality) and DFS (low space complexity)
- Limit depth of DFS to prevent infinite paths

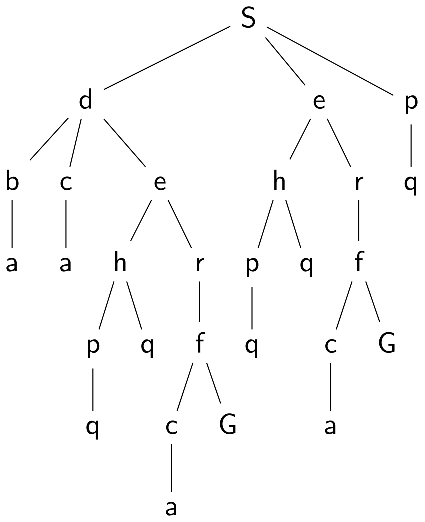
Iterative Deepening Search

- Combines benefits of BFS (completeness and optimality) and DFS (low space complexity)
- Limit depth of DFS to prevent infinite paths
- Iterative deepening search, run DFS with depth limits $0, 1, 2, \ldots, L$

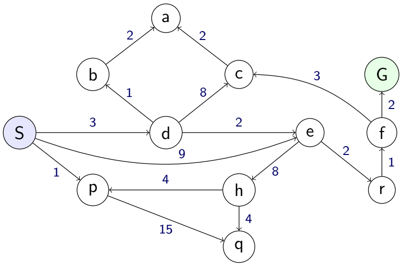
Iterative Deepening Search

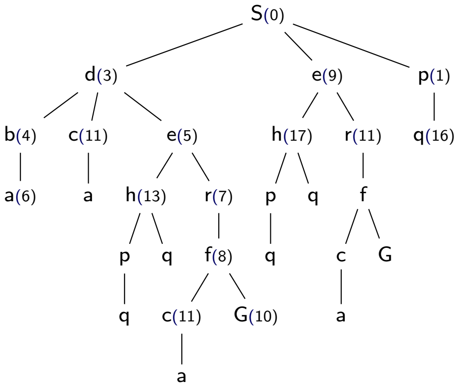
- Combines benefits of BFS (completeness and optimality) and DFS (low space complexity)
- Limit depth of DFS to prevent infinite paths
- Iterative deepening search, run DFS with depth limits $0, 1, 2, \ldots, L$
- L can be choosen based on the knowledge of the problem
  - For e.g., for the map of Romania, a suitable value of L is 19
- Completeness: Yes
- Cost Optimal: Yes
- Time complexity: $O(b^d)$
- Space complexity: $O(bd)$

Iterative Deepening Search



Uniform Cost Search





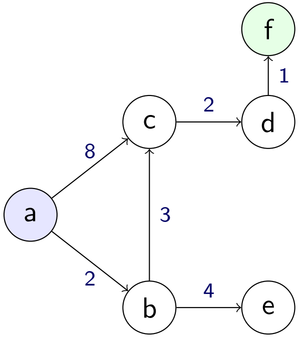
Expand node with lowest path cost

Uniform Cost Search Properties

- Expands nodes with cost less than cheapest solution
- Suppose C^* is the cheapest cost and ϵ is the least action cost. Then the tree depth will roughly be C^*/ϵ
- Completeness: Yes
- Cost optimality: Yes
- Time complexity: $O(b^{C^*/\epsilon})$
- Space complexity: $O(b^{C^*/\epsilon})$

Class Exercise

- Find the solutions generated by DFS, BFS, and UCS algorithms on the following problem.



Start State: a, Goal State: f