# CS 486/686: Introduction to Artificial Intelligence

Search

# Plan for Today

- Uniformed Search Methods
- Thinking about the importance of abstractions

# Introduction

Search was one of the first topics studied in AI

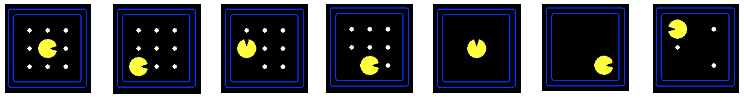
- Newell and Simon (1961) *General Problem Solver*

Central component to many AI systems

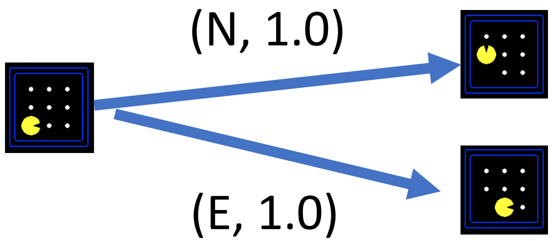
- Automated reasoning, theorem proving, robot navigation, scheduling, game playing, machine learning...

# Search Problems

- A **search problem** consists of
  - a state space

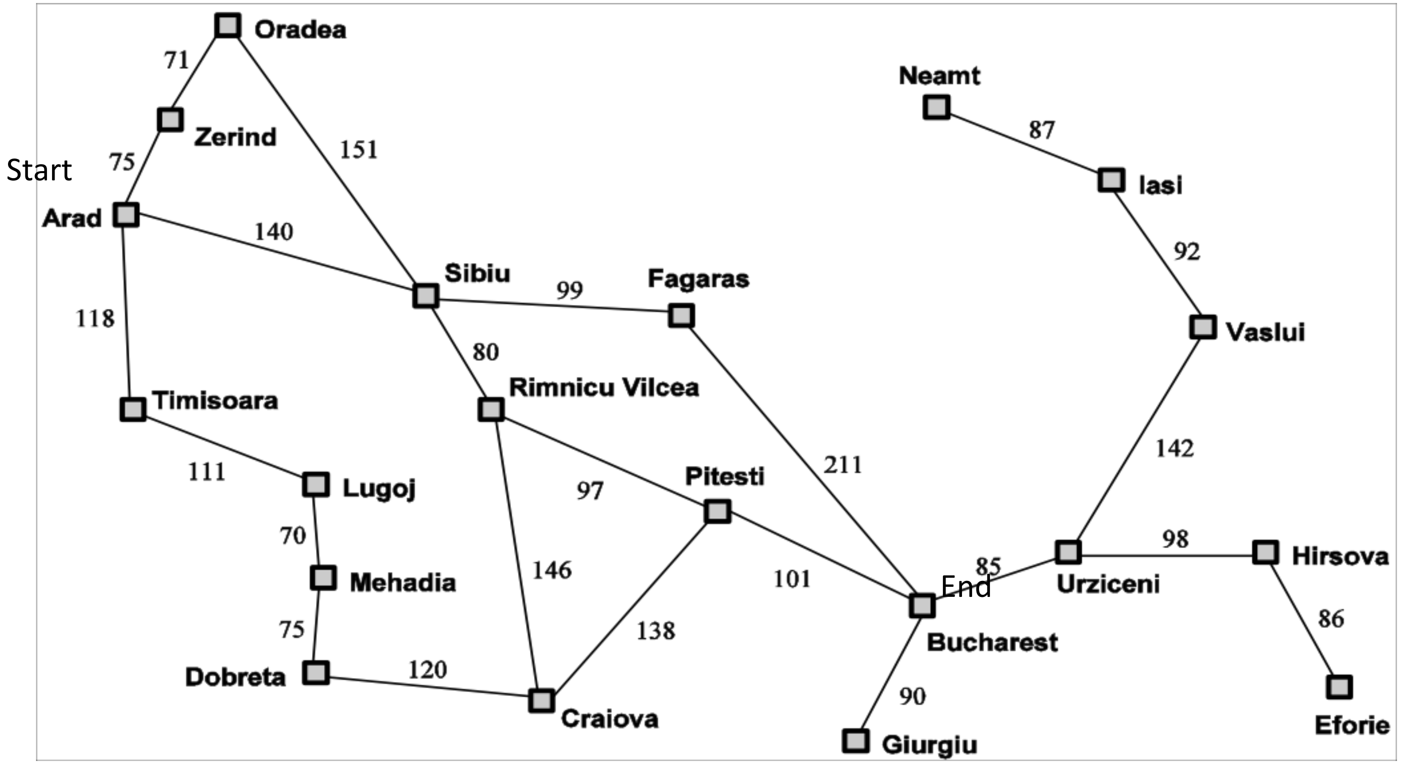


  - a successor function (actions, cost)



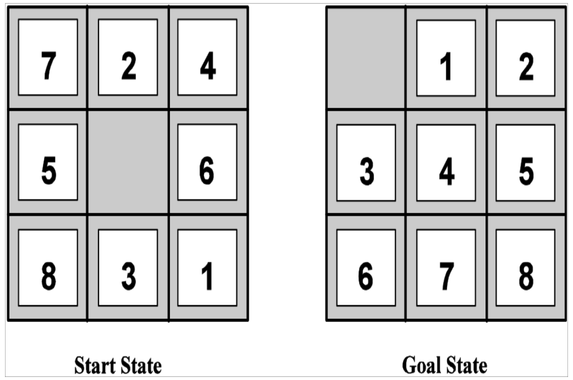

  - a start state and a goal test
- A **solution** is a sequence of actions (plan) from the start state to a goal state

# Example: Traveling in Romania




- States:
- Initial State:
- Successor Function:
- Goal test:
- Solution:

# Examples of Search Problems



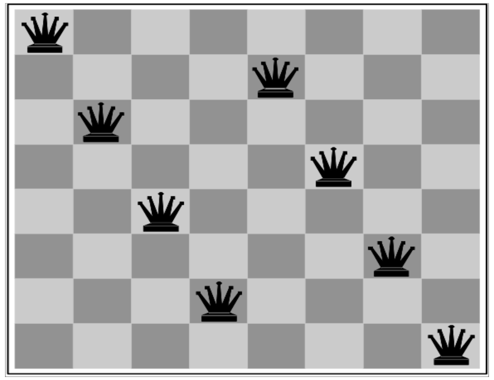

States:

Initial State:

Successor Function:

Goal test:

Solution:



States:

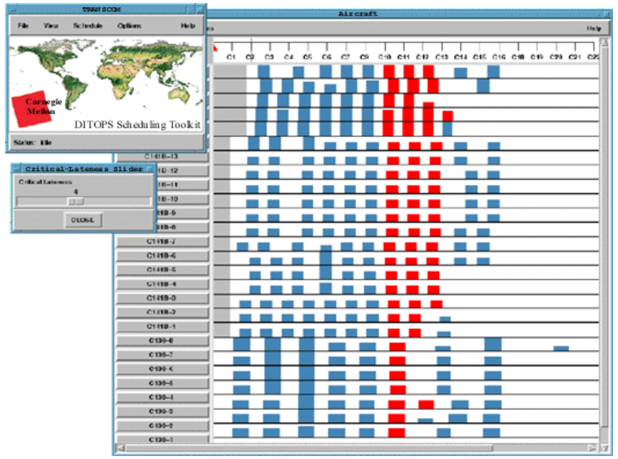
Initial State:

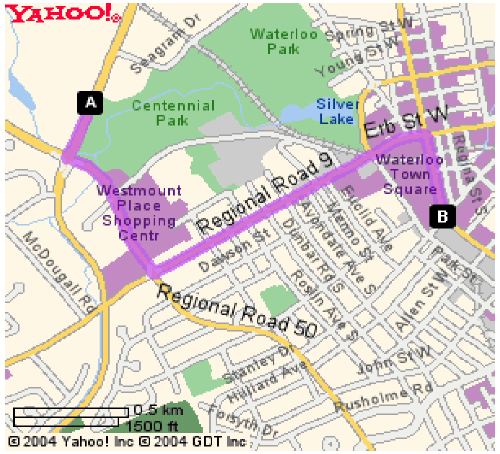
Successor Function:

Goal test:

Solution:

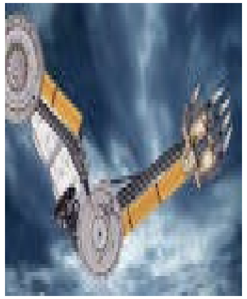
# Examples of Search Problems



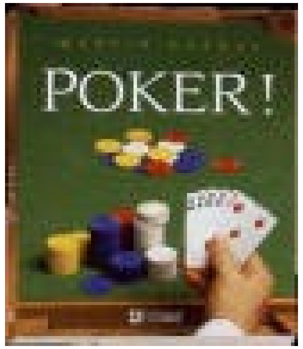

# Our Definition Excludes...

Chance



Adversaries

Continuous states

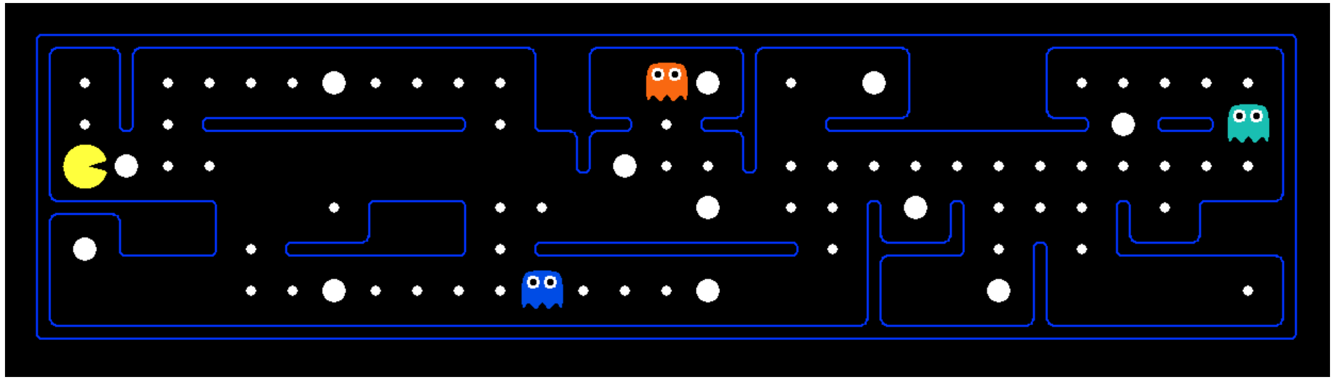



Partial Observability

All of the above

# What is is a state space?

The **world state** includes every last detail of the environment



A **search state** keeps only the details needed for planning (abstraction)

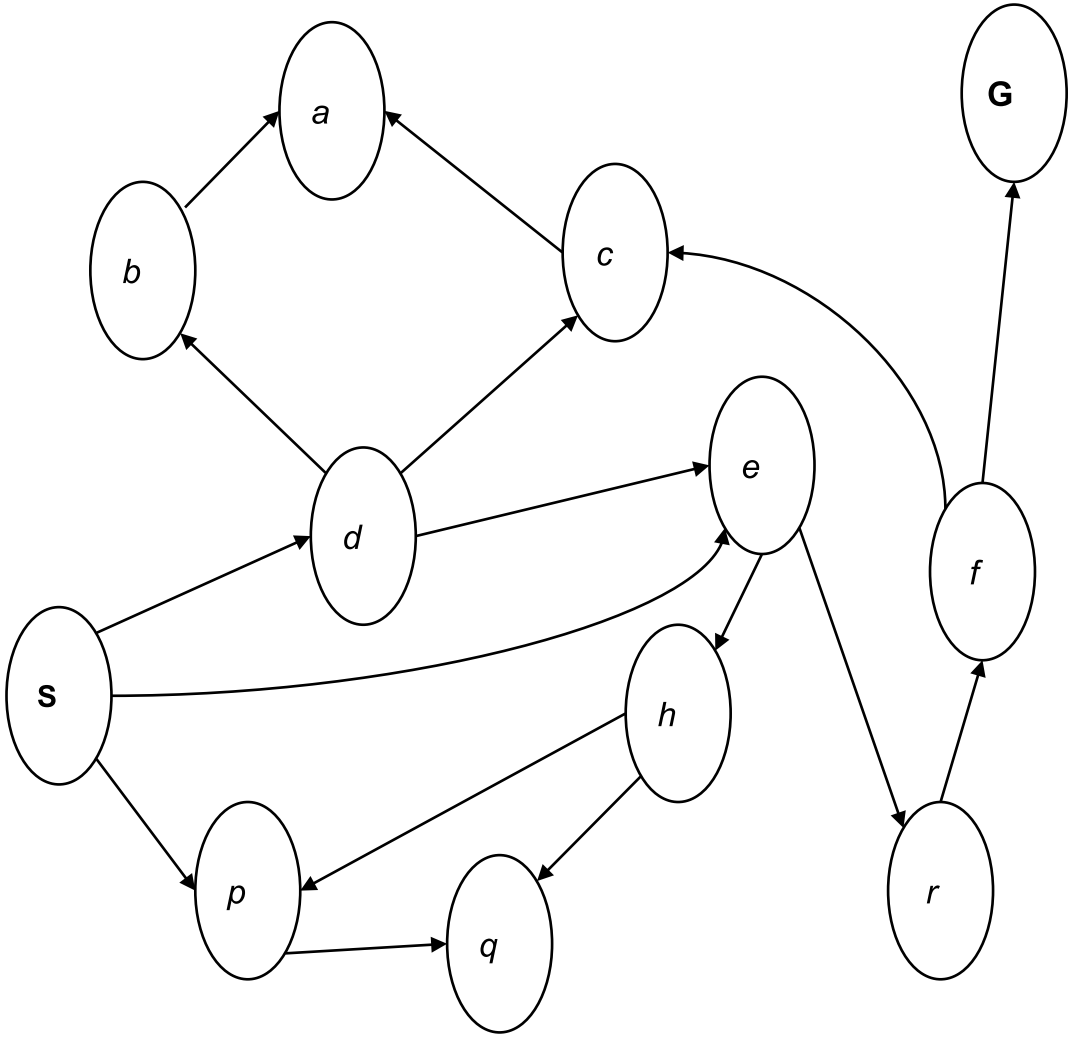
- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END

- Problem: Eat-All-Dots
  - States: {(x,y), dot booleans}
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean
  - Goal test: dots all false

Adapted from UC Berkeley's CS188 Course

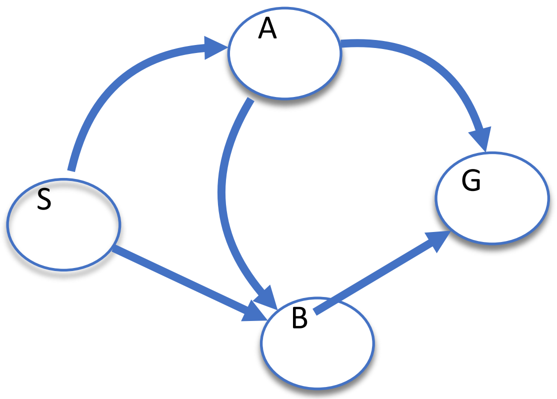
# Representing Search

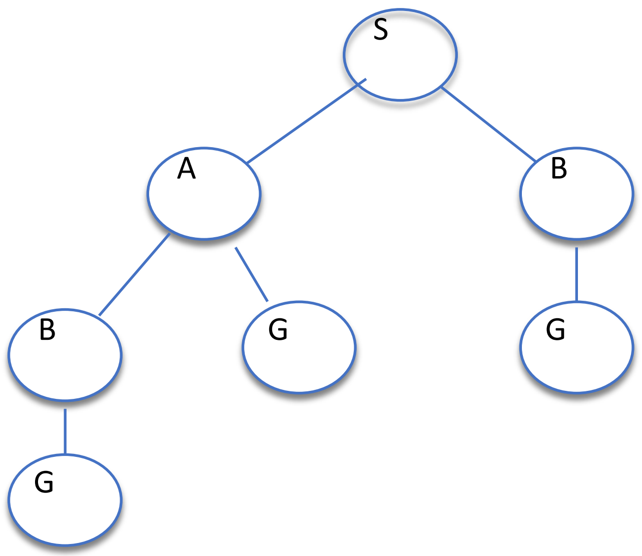
- **State space graph**
  - Vertices correspond to states (one vertex for each state)
  - Edges correspond to successors
  - Goal test is a set of goal nodes
- We search for a solution by building **a search tree** and traversing it to find a goal state



# Search Tree

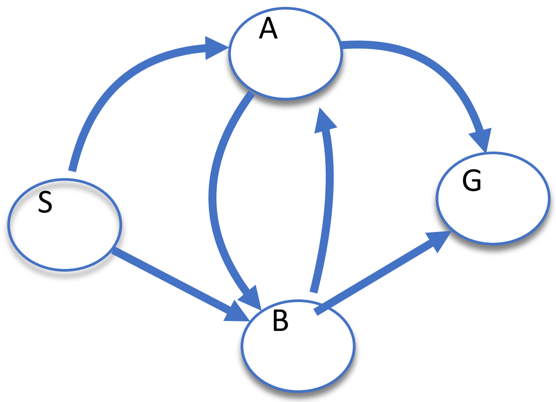
- **A search tree**:
  - Start state is the root of the tree
  - Children are successors
  - A plan is a path in the tree. A solution is a path from the root to a goal node.
  - For most problems we do not actually generate the entire tree



# Quiz

- Given this state graph, how large is the search tree?



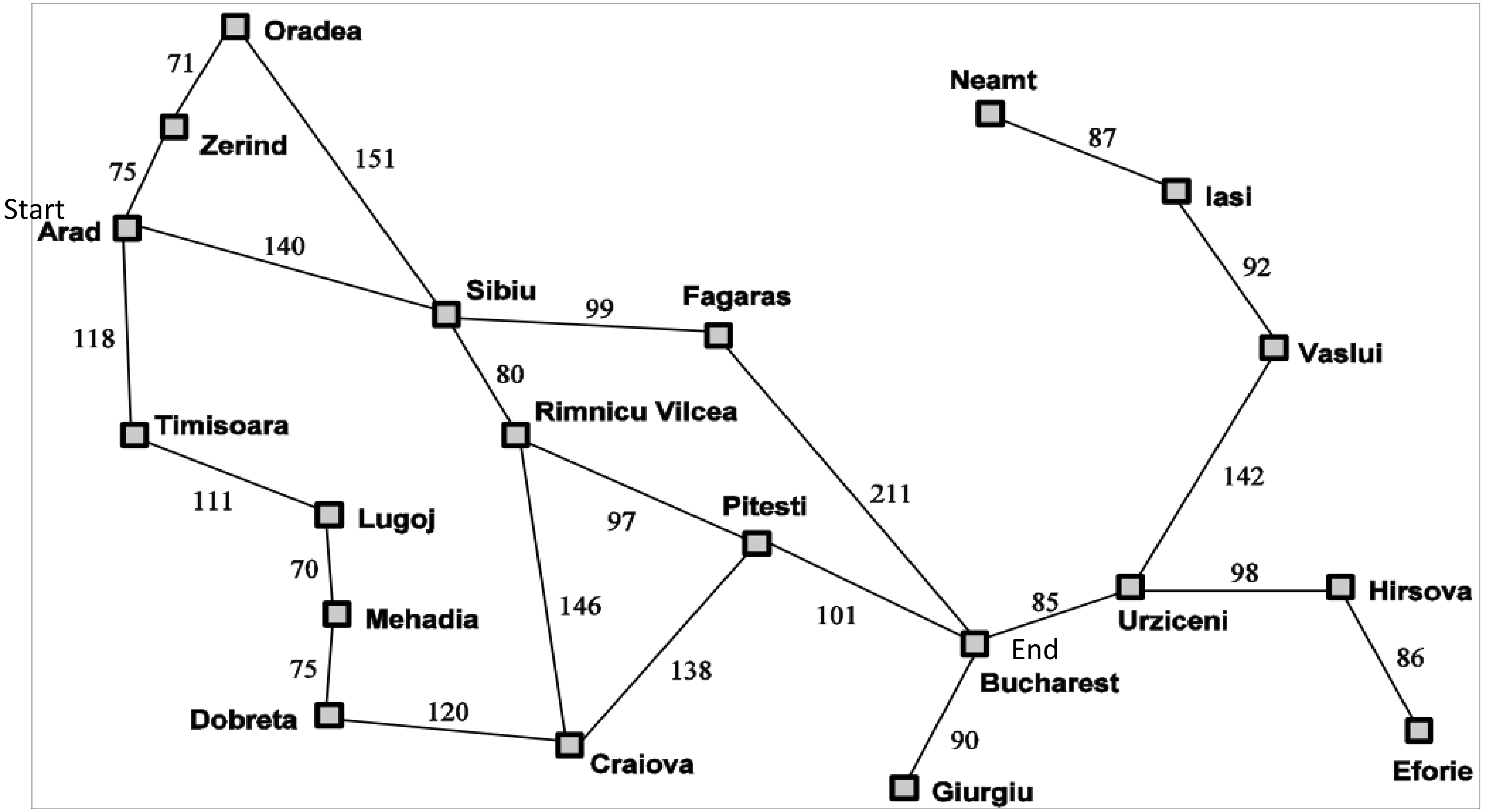
# Expanding Nodes

Expanding a node:

Applying all legal operators to the state contained in the node

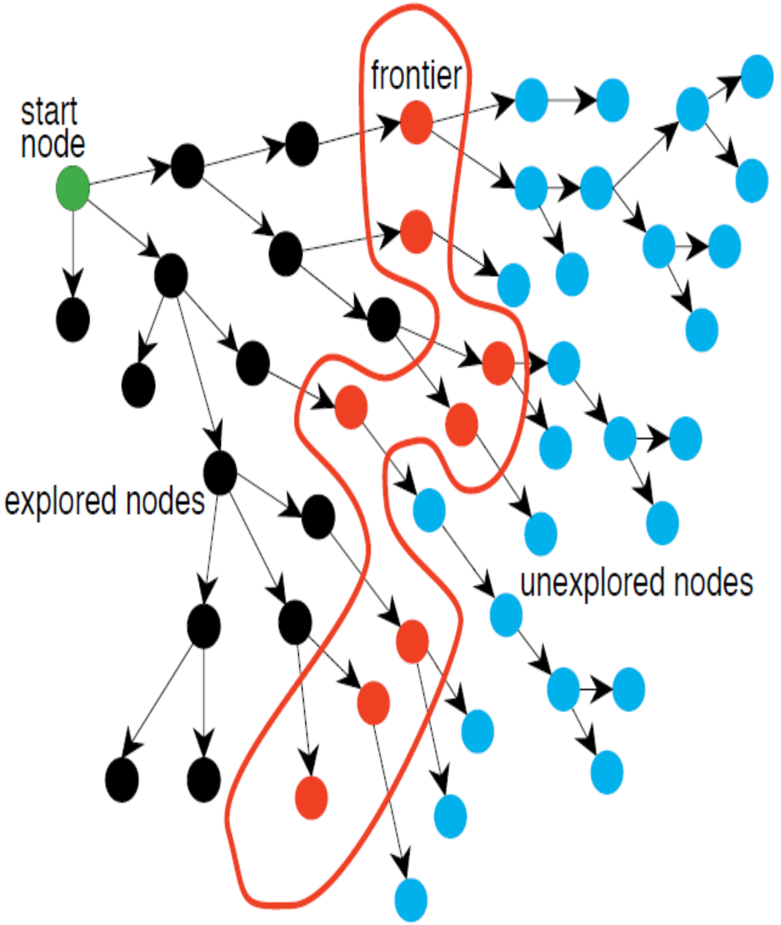
Generating nodes for all corresponding successor states

# Example: Traveling in Romania



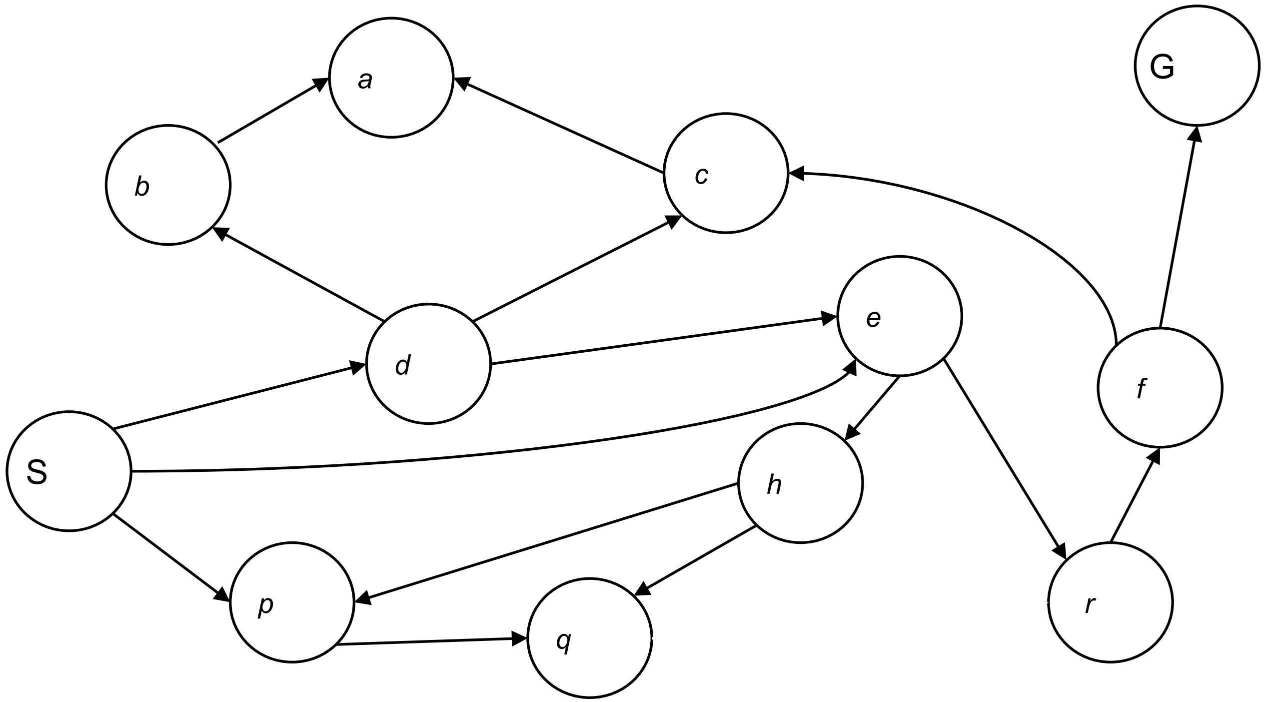
# Generic Search Algorithm

- Initialize with initial state of the problem
- Repeat
  - If no candidate nodes can be expanded **return failure**
  - Choose leaf node for expansion, according to **search strategy**
  - If node contains goal state, **return solution**
  - Otherwise, expand the node. Add resulting nodes to the tree



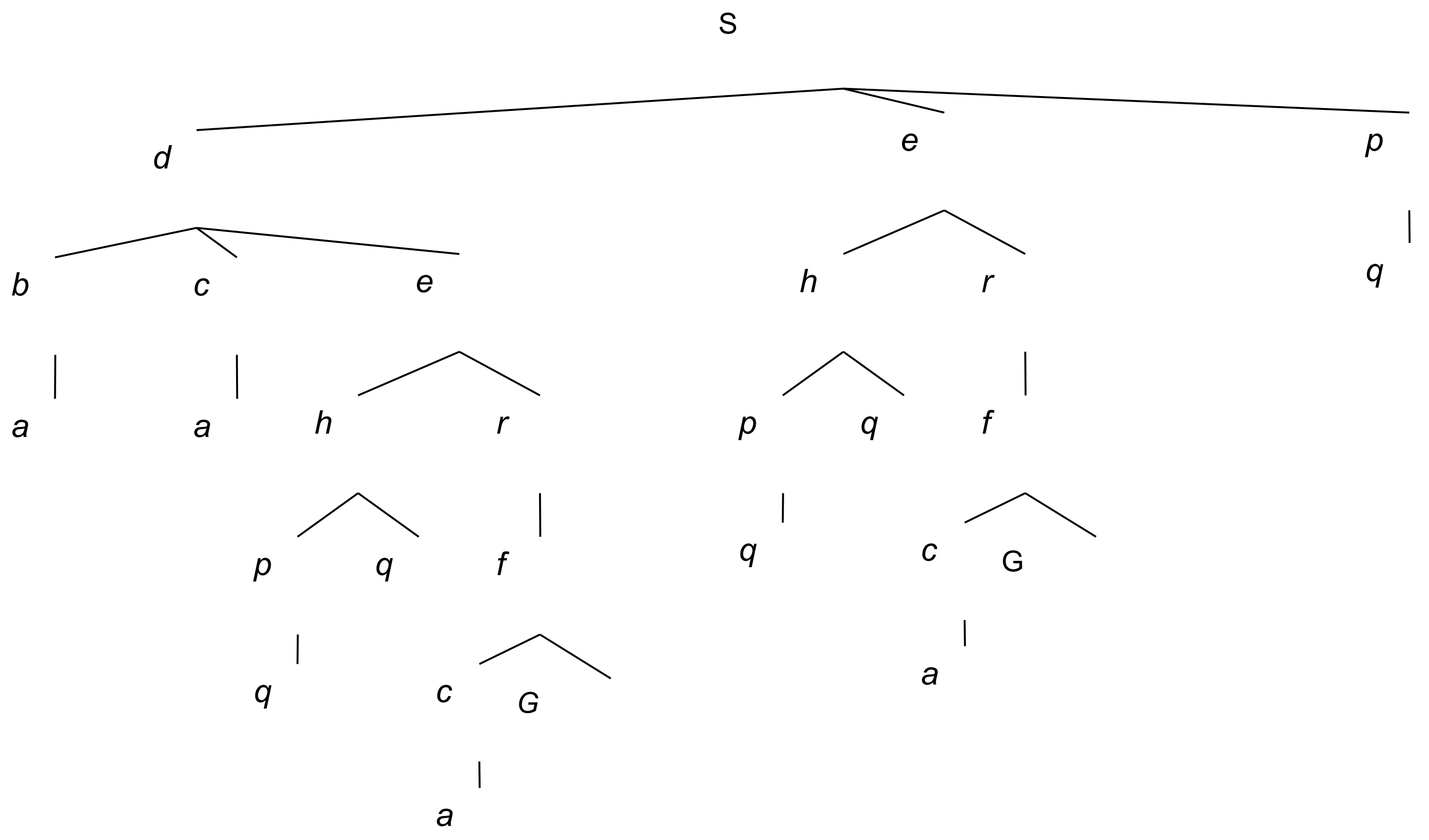
# Implementation Details

# Search Strategies




Adapted from UC Berkeley's CS188 Course

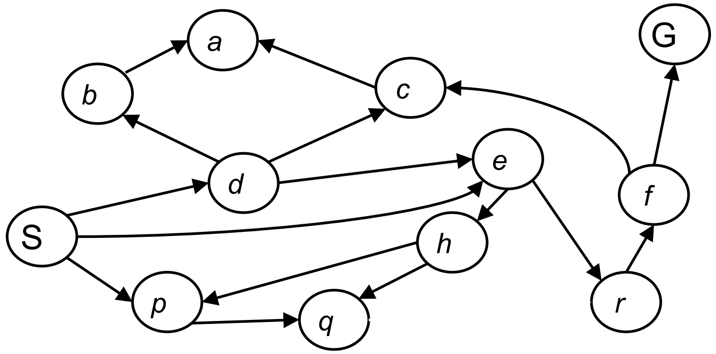
# Search Strategies

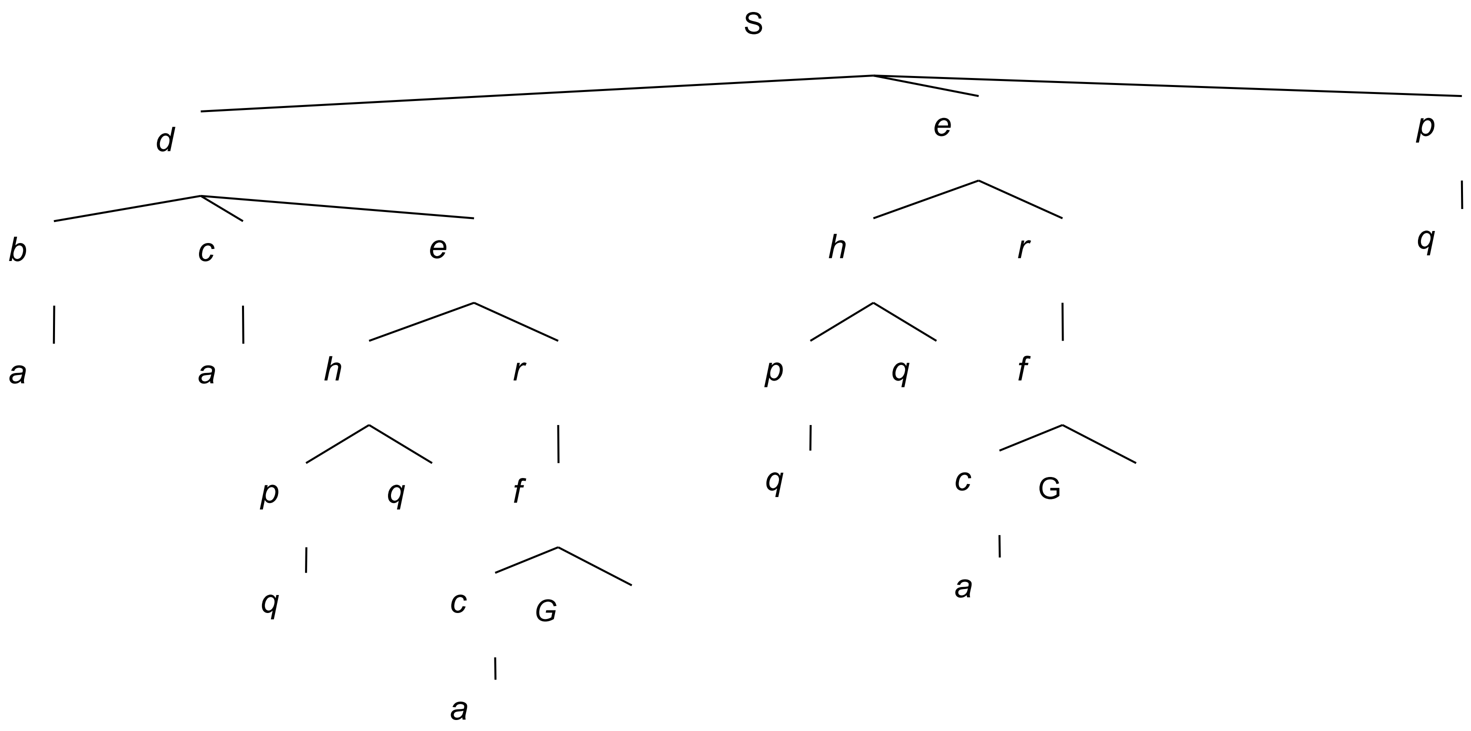


# Depth-First Search

**Strategy**: Expand deepest node first

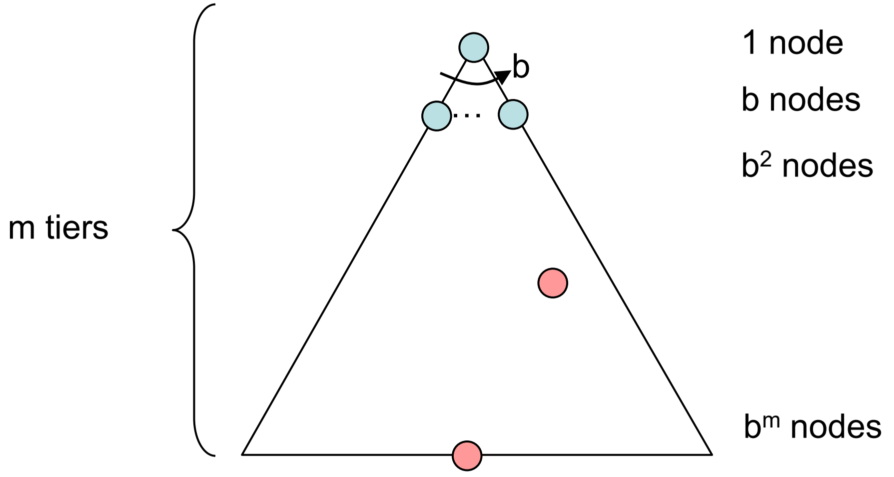
**Implementation**: LIFO stack



# Key Properties

- **Completeness**: Is the alg. guaranteed to find a solution if the solution exists?
- **Optimality:** Does the alg. find the optimal solution?
- **Time complexity**
- **Space complexity** (size of the fringe)




b: branching factor
m: maximum depth
d: depth of shallowest goal node

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

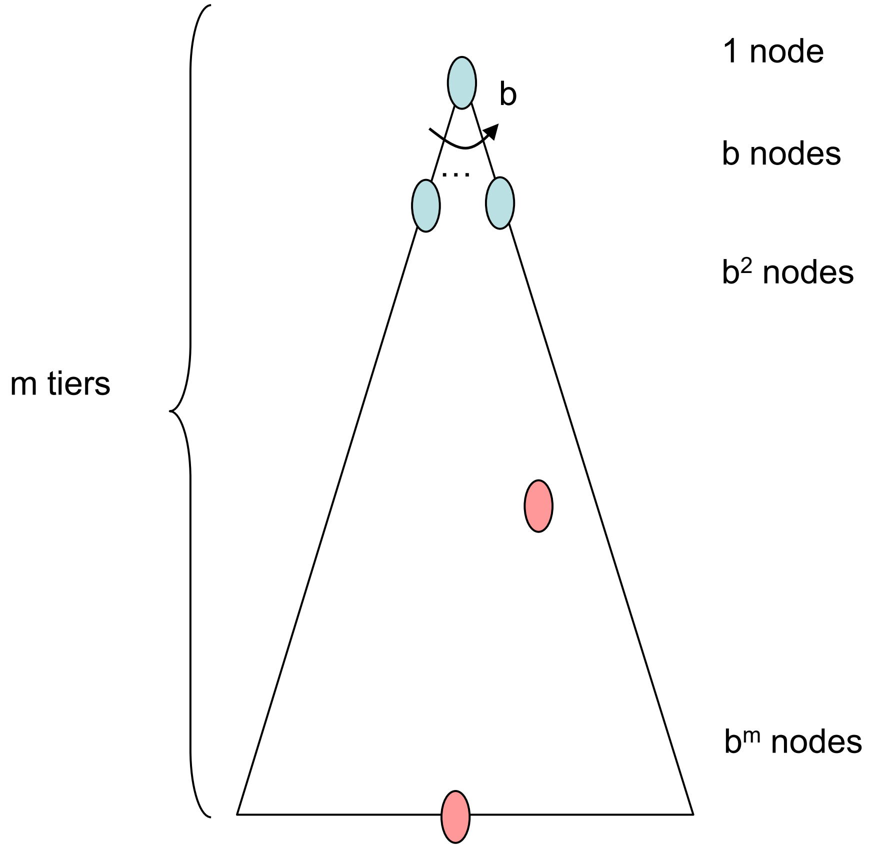
# DFS Properties

**Complete?**

**Optimal?**

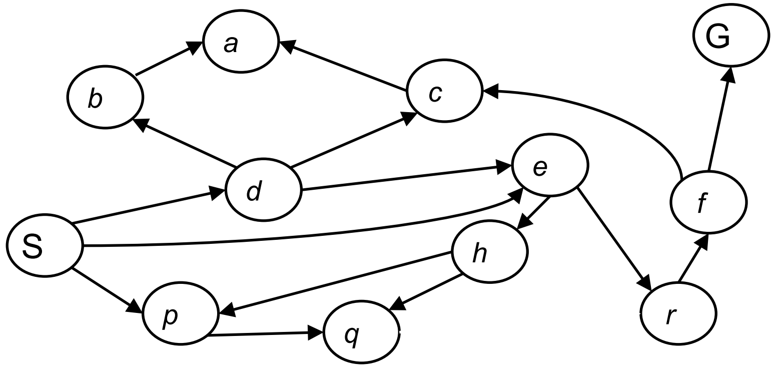
**Time complexity**

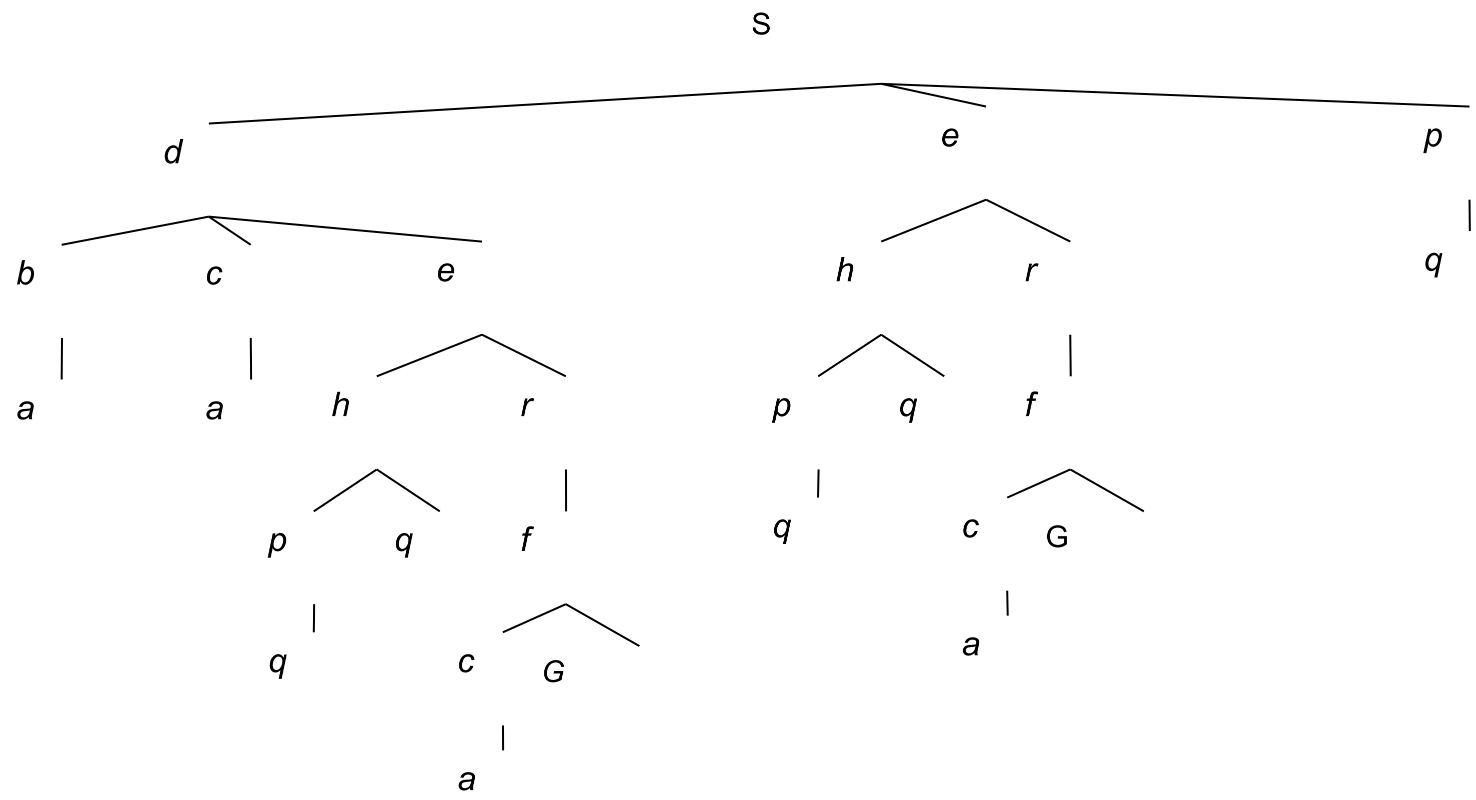
**Space complexity**



# Breadth-First Search

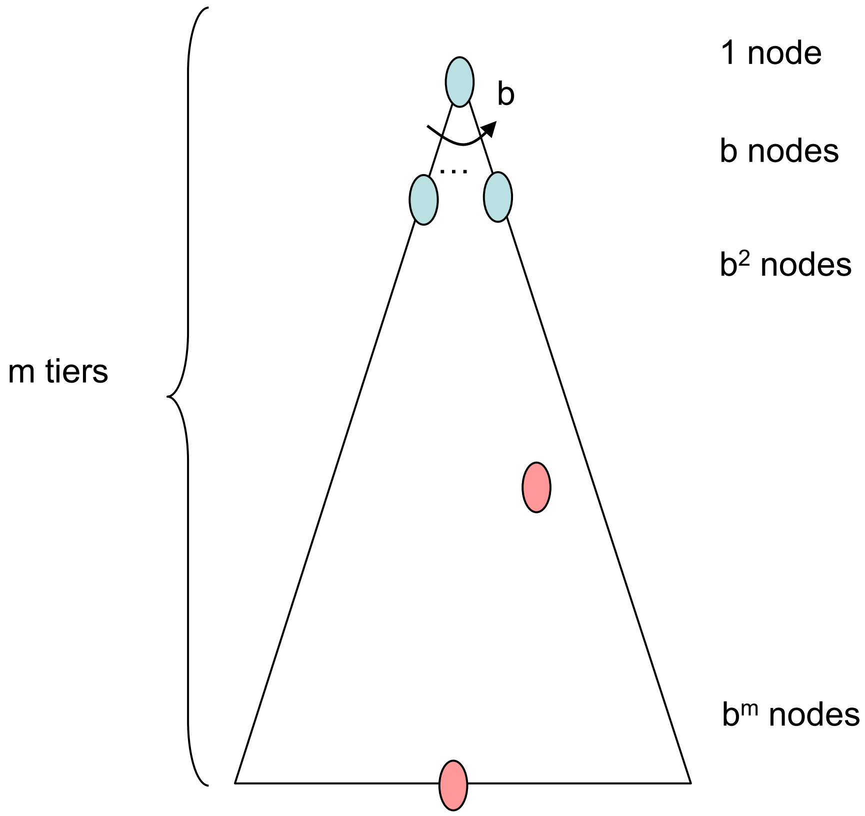
**Strategy**: Expand shallowest node first
**Implementation**: FIFO queue



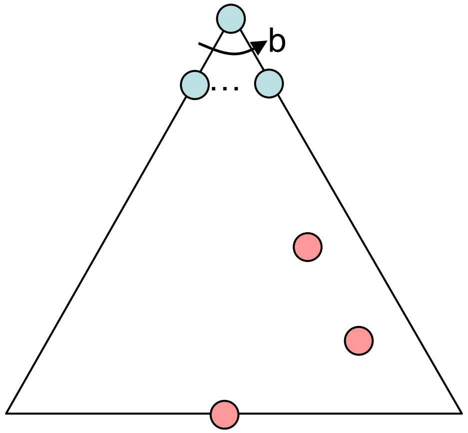



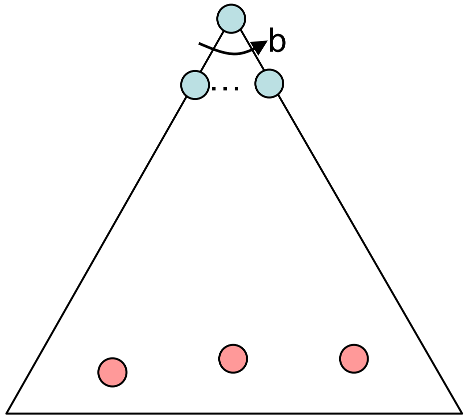
# BFS Properties

- **Complete?**
- **Optimal?**
- **Time complexity**
- **Space complexity**



# Quiz: DFS vs BFS



# Iterative Deepening Search

- Can we combine search methods to take advantage of DFS space complexity and BFS completeness/shallow solution advantage?

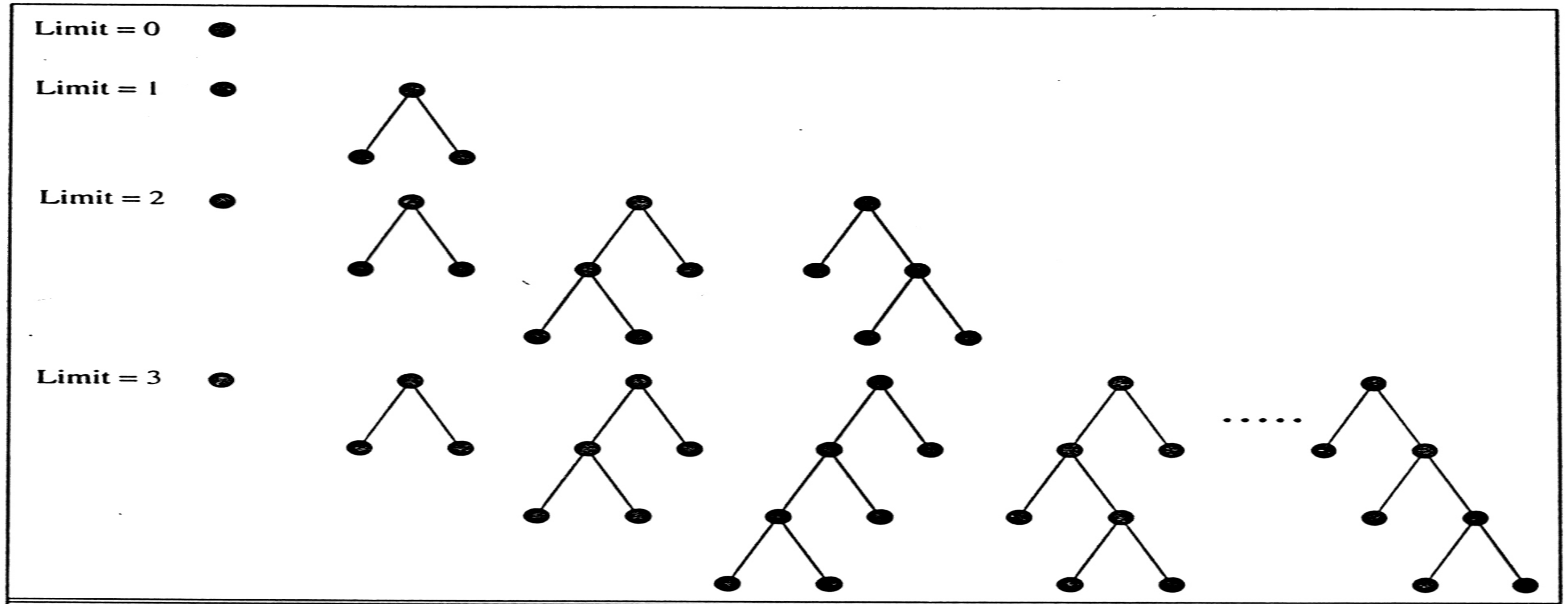
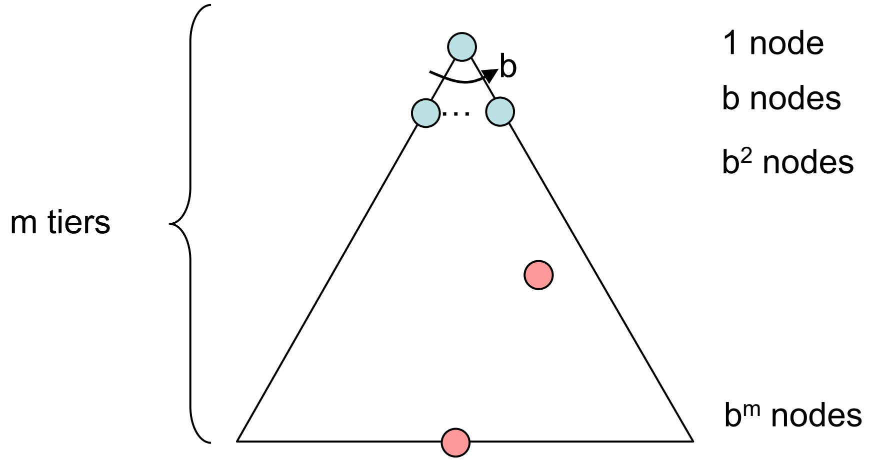



**Figure 3.16** Four iterations of iterative deepening search on a binary tree.

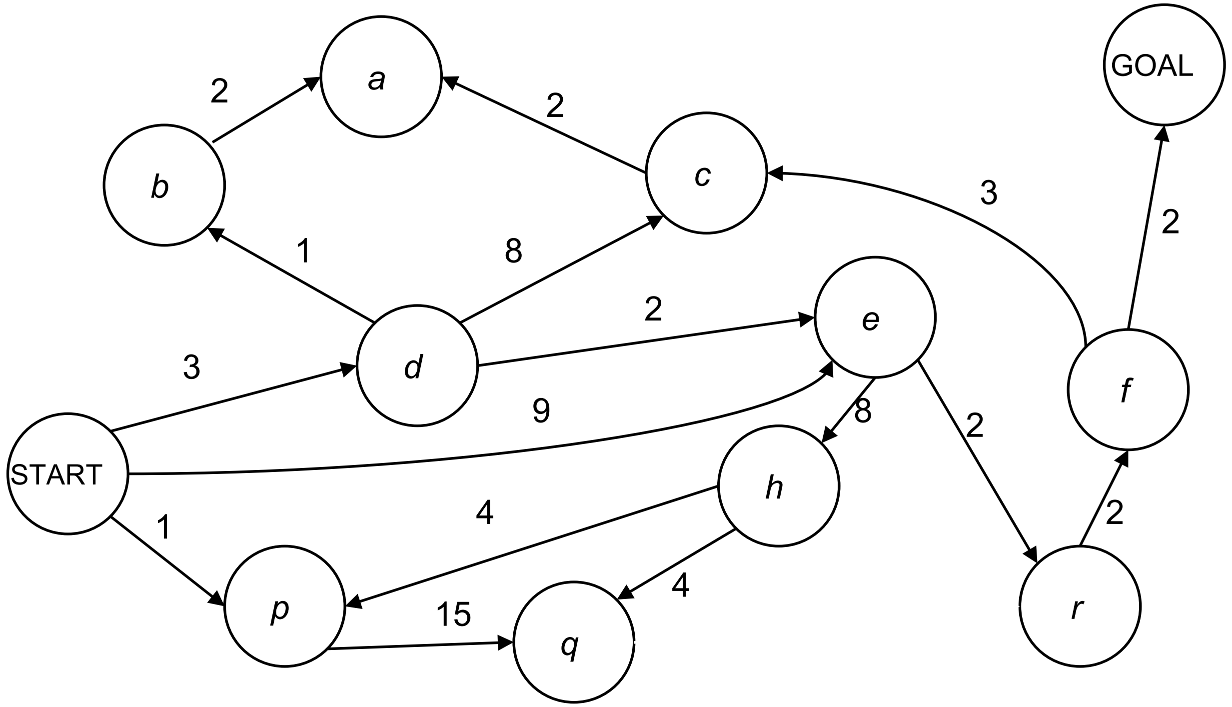
# IDS Properties

- **Complete?**
- **Optimal?**
- **Time complexity**
- **Space complexity**




**Wasteful?** Most nodes found in lowest level of search so not too bad

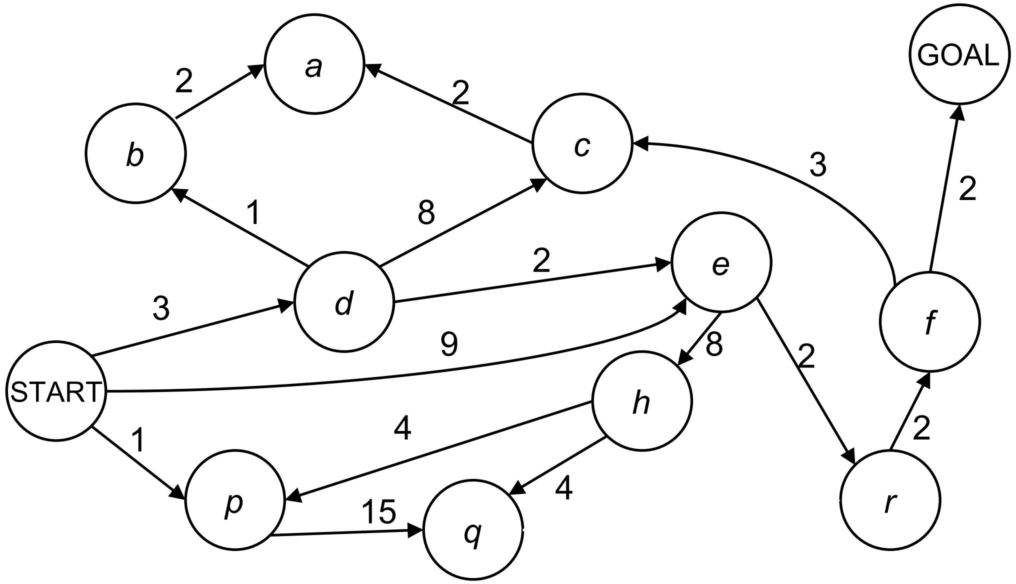
# Cost-Sensitive Search




Recall that BFS was only optimal under some conditions (i.e. we only cared about number of actions taken). What can we do if actions have different costs?

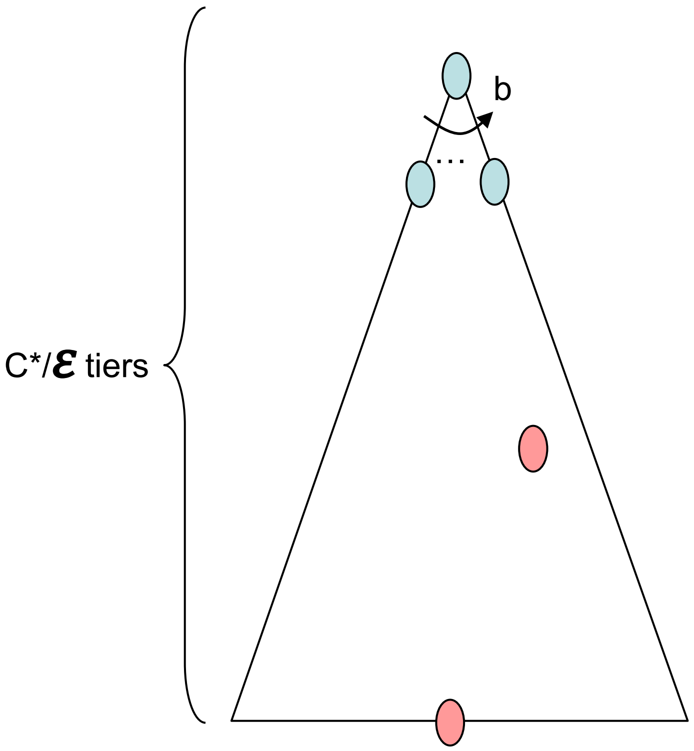
# Uniform Cost Search

**Strategy**: Expand cheapest node first
**Implementation**: Priority queue



# UCS Properties

- **Complete?**
- **Optimal?**
- **Time complexity**
- **Space complexity**



# Summary

- These algorithms are basically the same except for the order in which they expand nodes
    - Basically all priority queues with different ways to determining priorities

- How successful the search is depends heavily on your model!

# Questions?

- Next class: Informed search