CIS 471/571(Fall 2020): Introduction to Artificial Intelligence

Lecture d Search

Thanh H. Nguyen

Most slides are by Pieter Abbeel, Dan Klein, Luke Zettlemoyer, John DeNero, Stuart Russell, Andrew Moore, or Daniel Lowd
Source: http://ai.berkeley.edu/home.html

Announcement

- Project 1
  - Deadline: Oct 13^{th}, 2020

- Written Assignmen

  - Will be posted today

  - Deadline: Oct 10^{th}, 2020

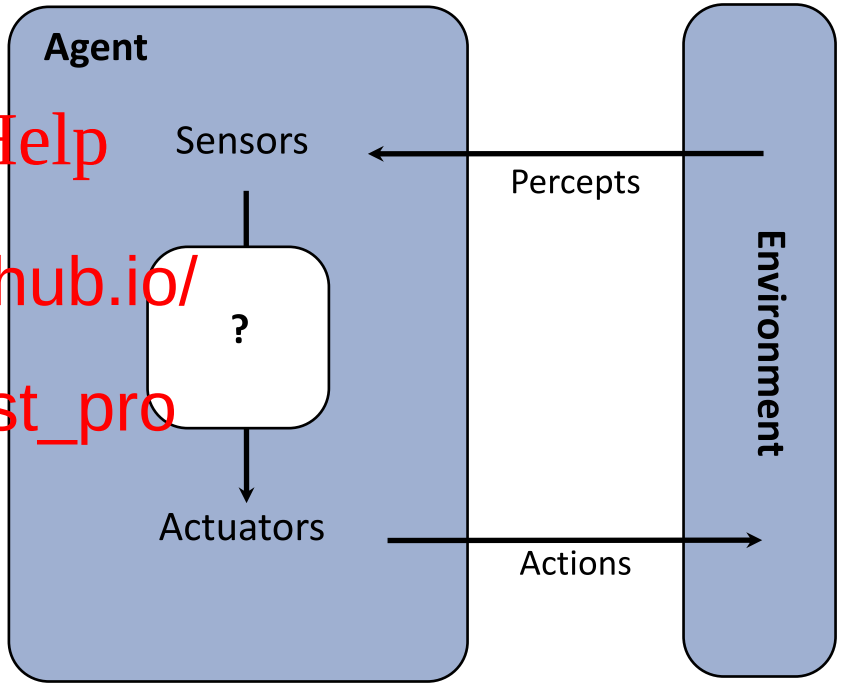
Today

- Agents that Plan Ahead

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

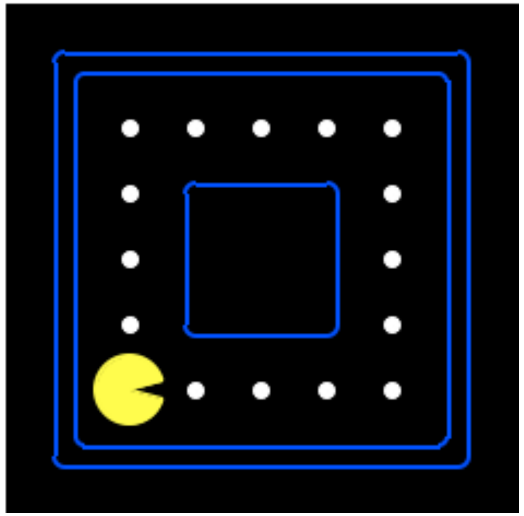
Rational Agents

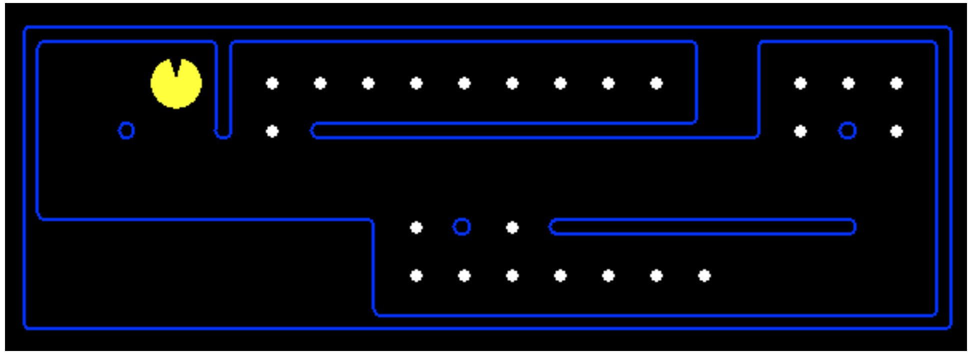
- An **agent** is an entity that *perceives* and *acts*.
- A **rational agent** selects actions that maximize its **utility fu**
- Characteristics of the **p** **environment,** and **action space** dic techniques for selecting rational actio



Reflex Agents

- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - Do not consider future c tions
  - Consider how the world
- Can a reflex agent be rational?





Video of Demo Reflex Optimal

Video of Demo Reflex Odd

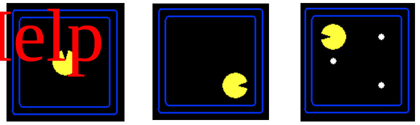
Goal-based Agents

- Goal-based agents:
  - Plan ahead
  - Ask "what if"
  - Decisions based on (h
    consequences of actio
  - Must have a model of how the worl
    evolves in response to actions
  - Act on how the world WOULD BE

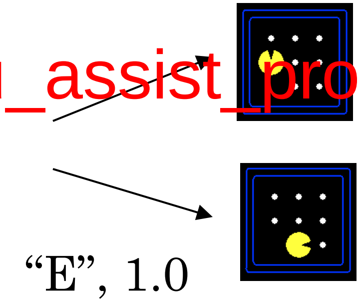
Video of Demo Mastermind

Search Problem

- A search problem consists of:
  - A state space



  - A successor function (with actions, costs)



  - A start state and a goal test

- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Example: Romania

- State space:
  - Cities
- Successor function:
  - Go to adj city with cos = dist
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

What is in State Space

The world state includes every last detail of the environment

Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro

- 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

State Space Size

- Search Problem:
  Eat all of the food

- Pacman positions: 10 x 12 = 120
- Pacman facing: up, down,

- Food Count: 30

- Ghost positions: 12

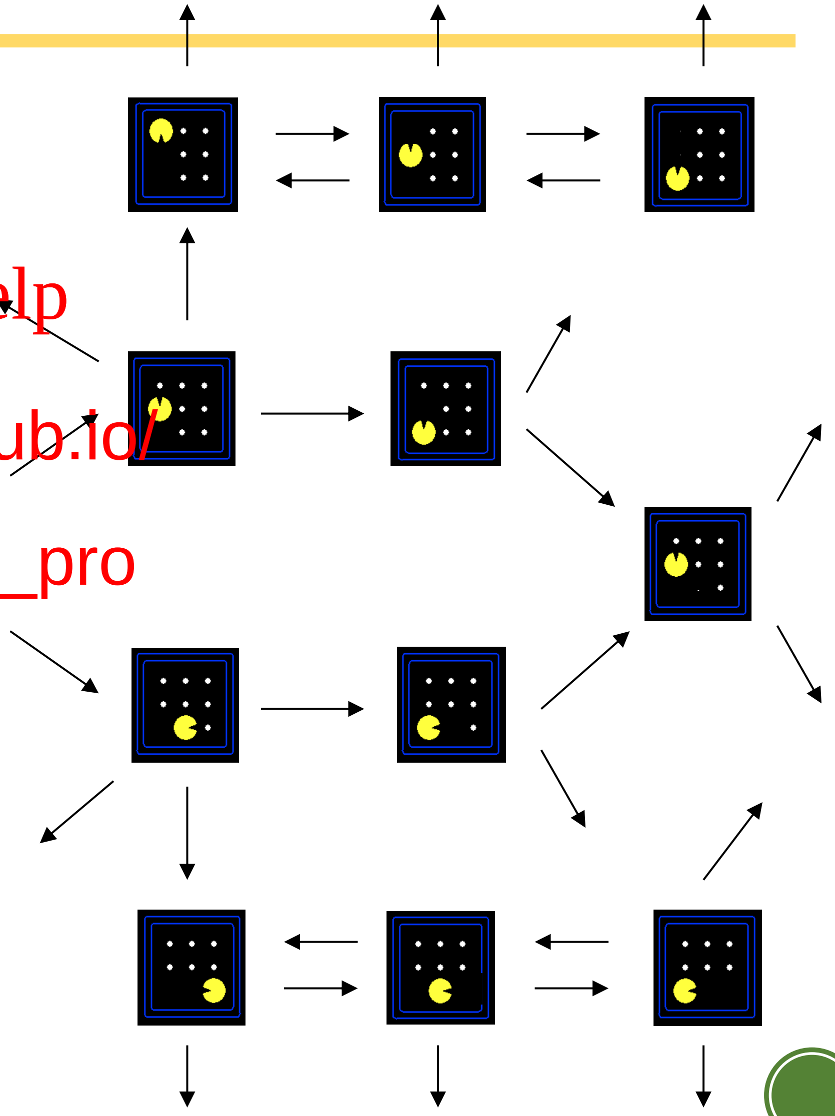
- How many
- World states? $120*(2^{30})*(12^{2})*4$
- States for pathing? 120
- States for eat-all-dots? $120*(2^{30})$

State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal no one)

- In a state space graph, each state occur only once!

- We can rarely build this full graph in memory (it's too big), but it's a useful idea

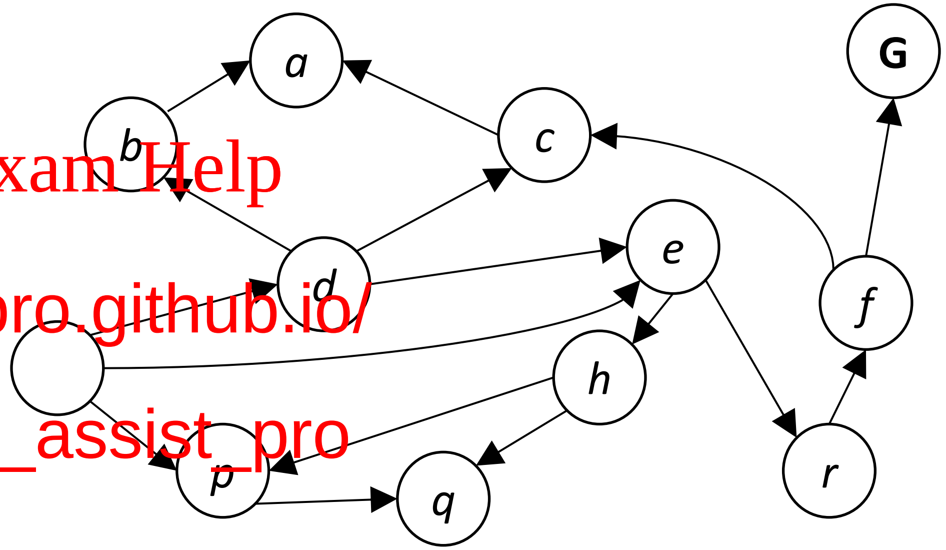


State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal no
    one)

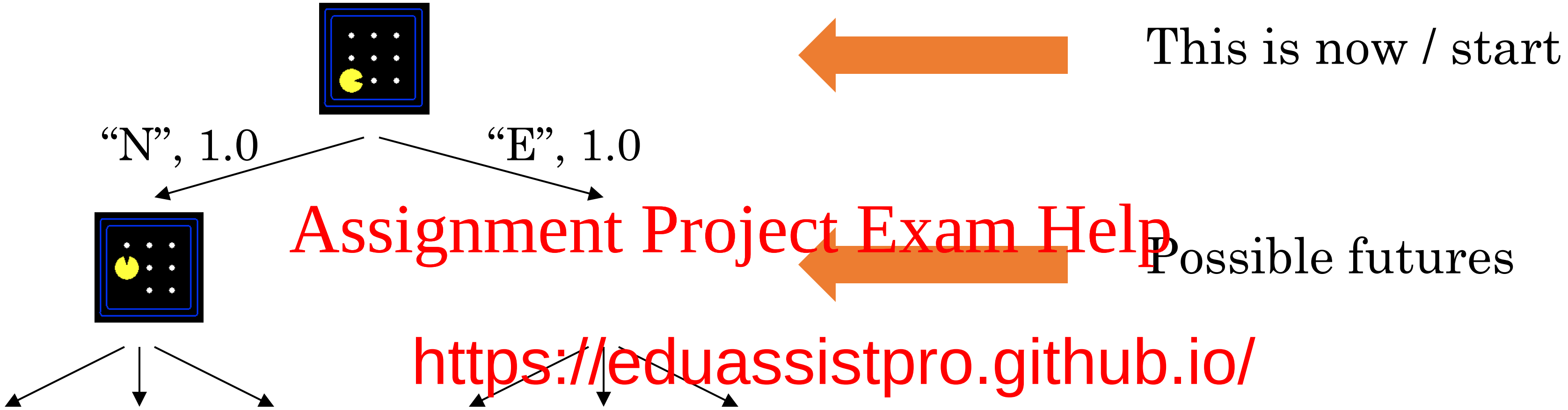
- In a state space graph, each state occur only once!

- We can rarely build this full graph in memory (it’s too big), but it’s a useful idea



Tiny state space graph for a tiny search problem

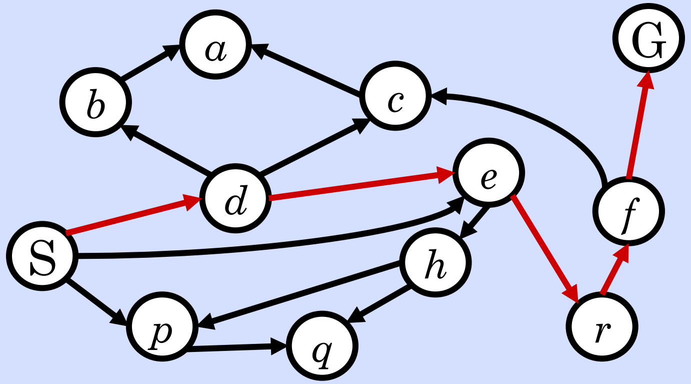
Search Trees



- A search tree:
  - 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
  - For most problems, we can never actually build the whole tree

State Space Graphs vs. Search Trees

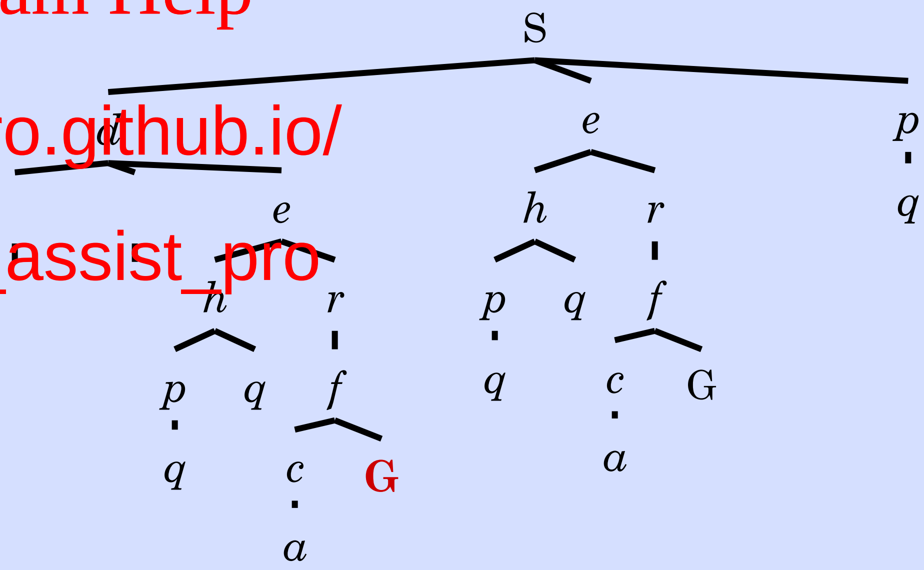
State Space Graph



Each NODE in
in the search tree
is an entire
PATH in the

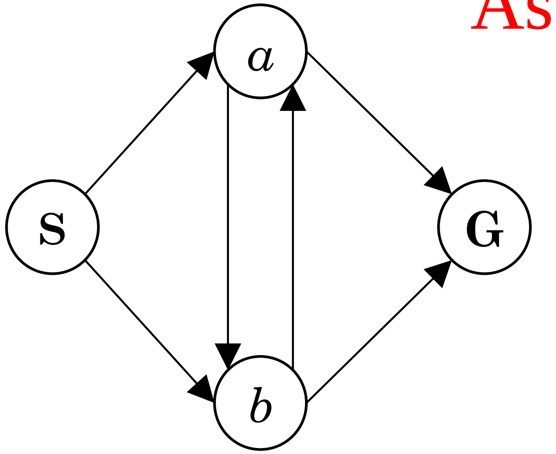
We construct both
on demand – and
we construct as
little as possible.

Search Tree



Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

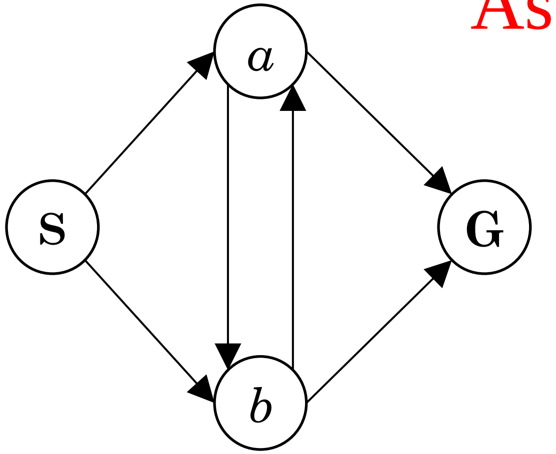


How big is its search tree (from S)?

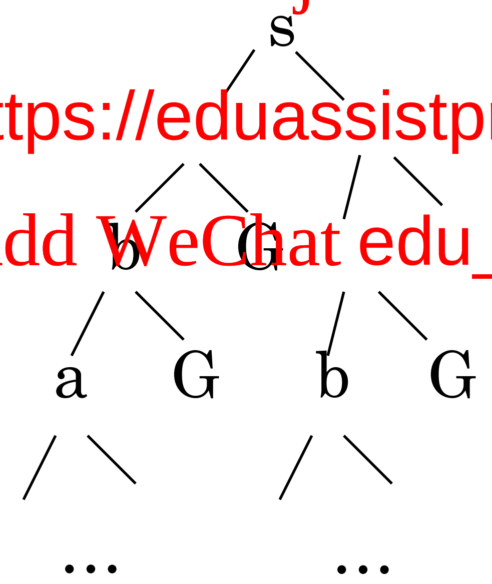
∞

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

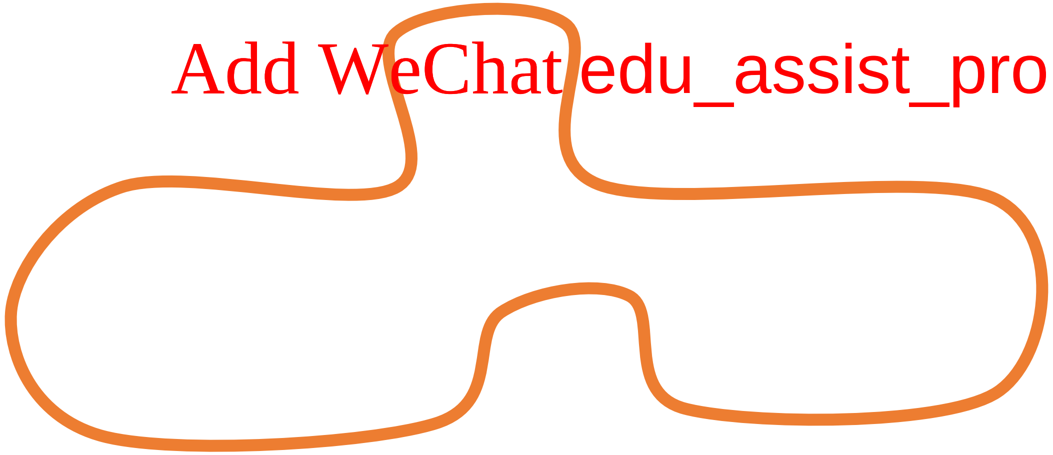


How big is its search tree (from S)?



Important: Lots of repeated structure in the search tree!

Tree Search



Search Example: Romania

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

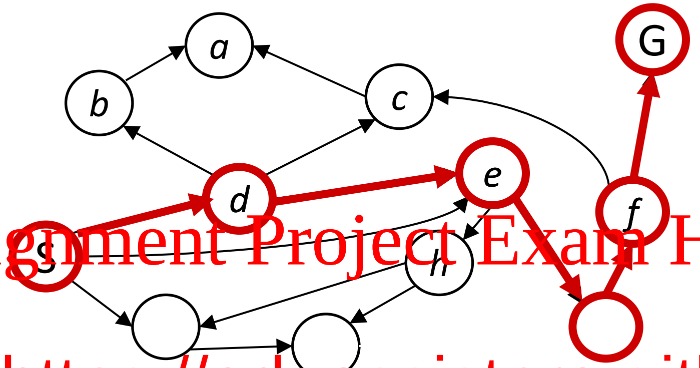
- Tree Search
  - Initialize the *root node* of the search tree with the *start state*

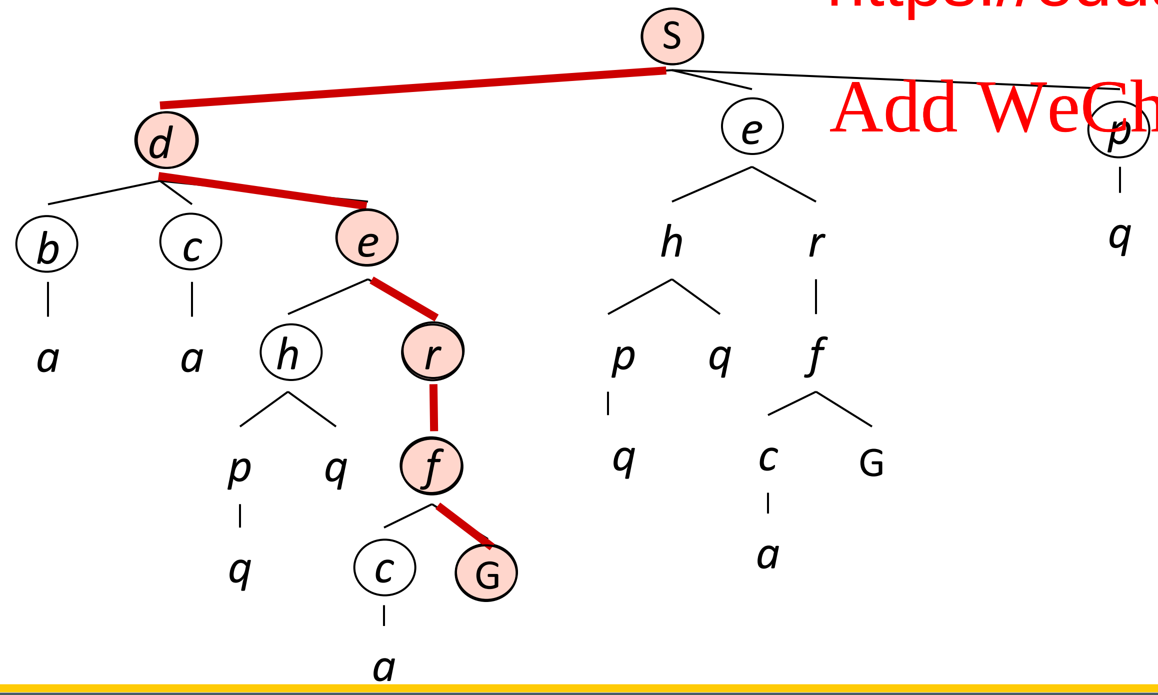
  - While there a es (fringe):

    - Choose a le
    - If the node contains a goa

      return the correspondi
    - Else: expand the node and add its children to the tree
- Important ideas:
  - Fringe
  - Expansion
- Strategy: which fringe nodes to explore?

Example: Tree Search





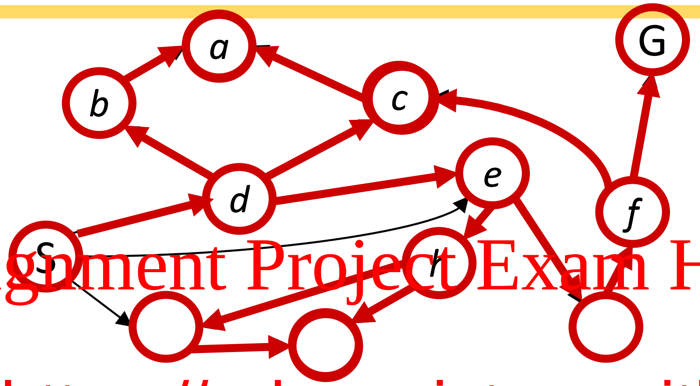
~~s~~
~~s → d~~
~~s → e~~
s → p
s → d → b
s → d → c
~~s → d → e~~
s → d → e → h
~~s → d → e → r~~
~~s → d → e → r → f~~
s → d → e → r → f → c
~~s → d → e → r → f → G~~

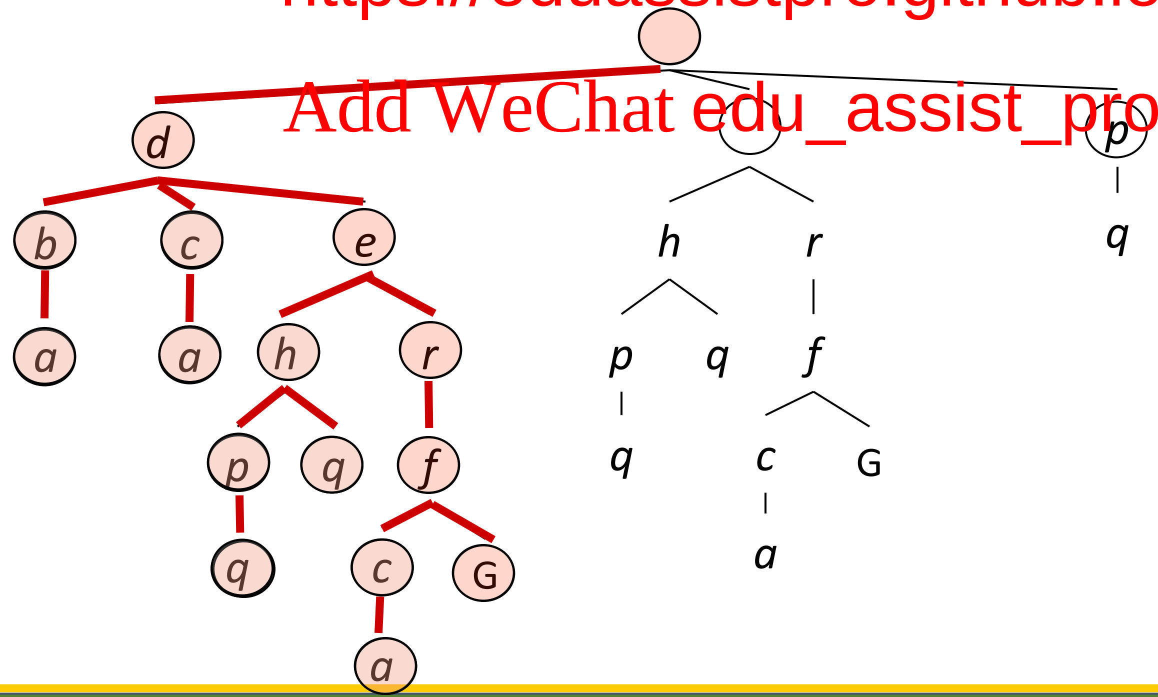
Depth-First Search (DFS)

Depth-First Search (DFS)

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack

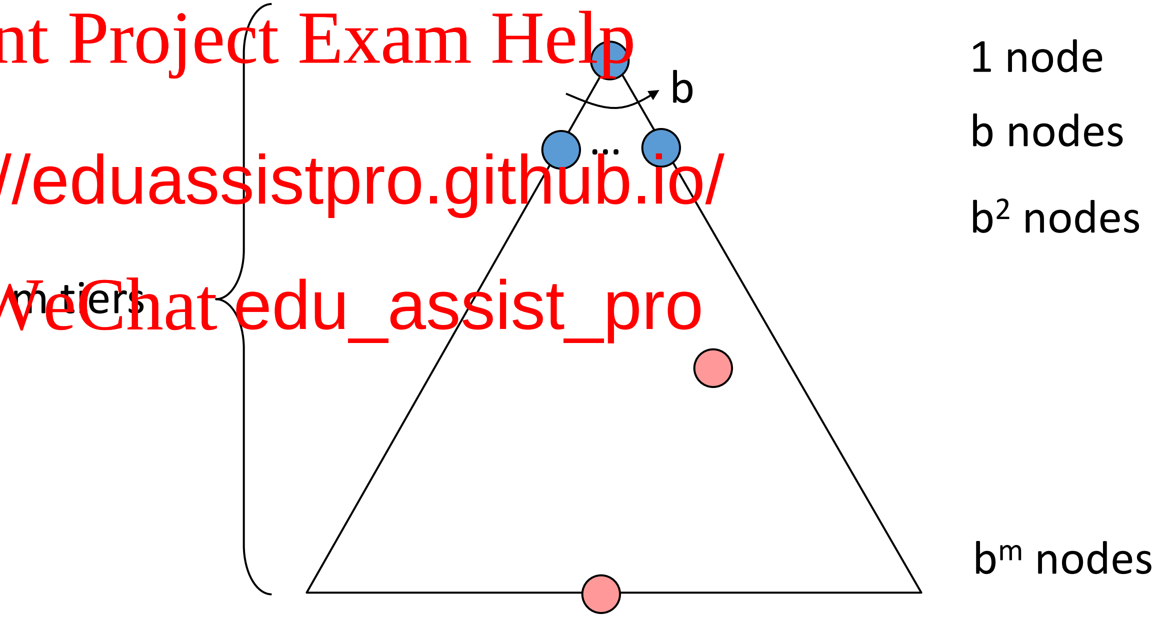




Search Algorithm Properties

- 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?
  - $1 + b + b^2 + ... b^m = O(b^m)$

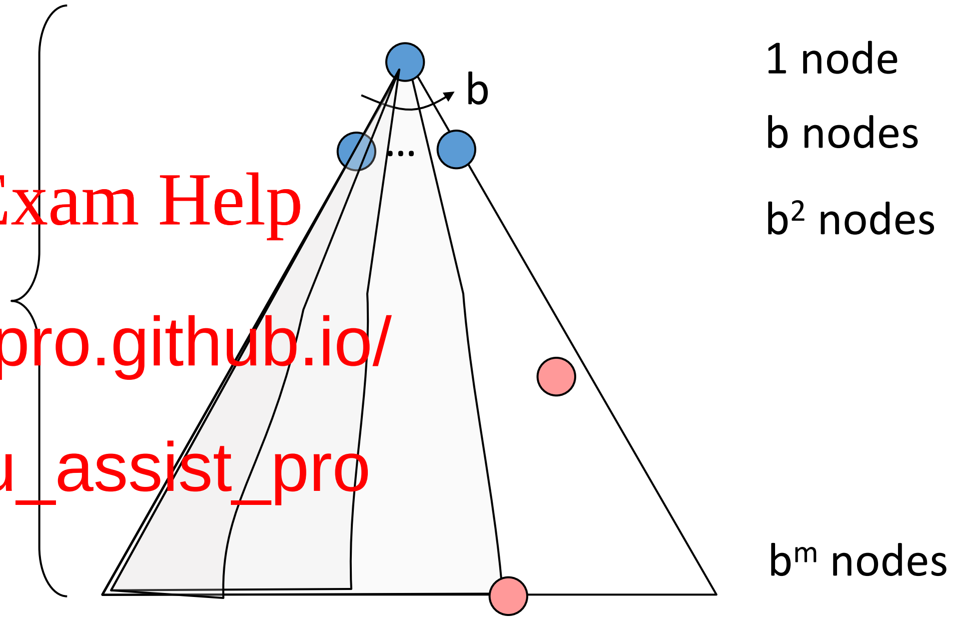
DFS Properties

- 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 frin
  - 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

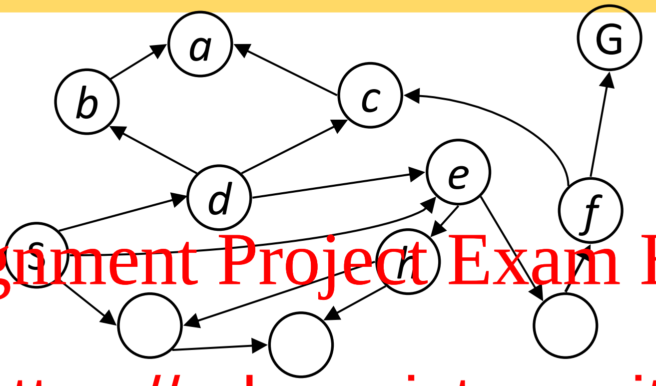


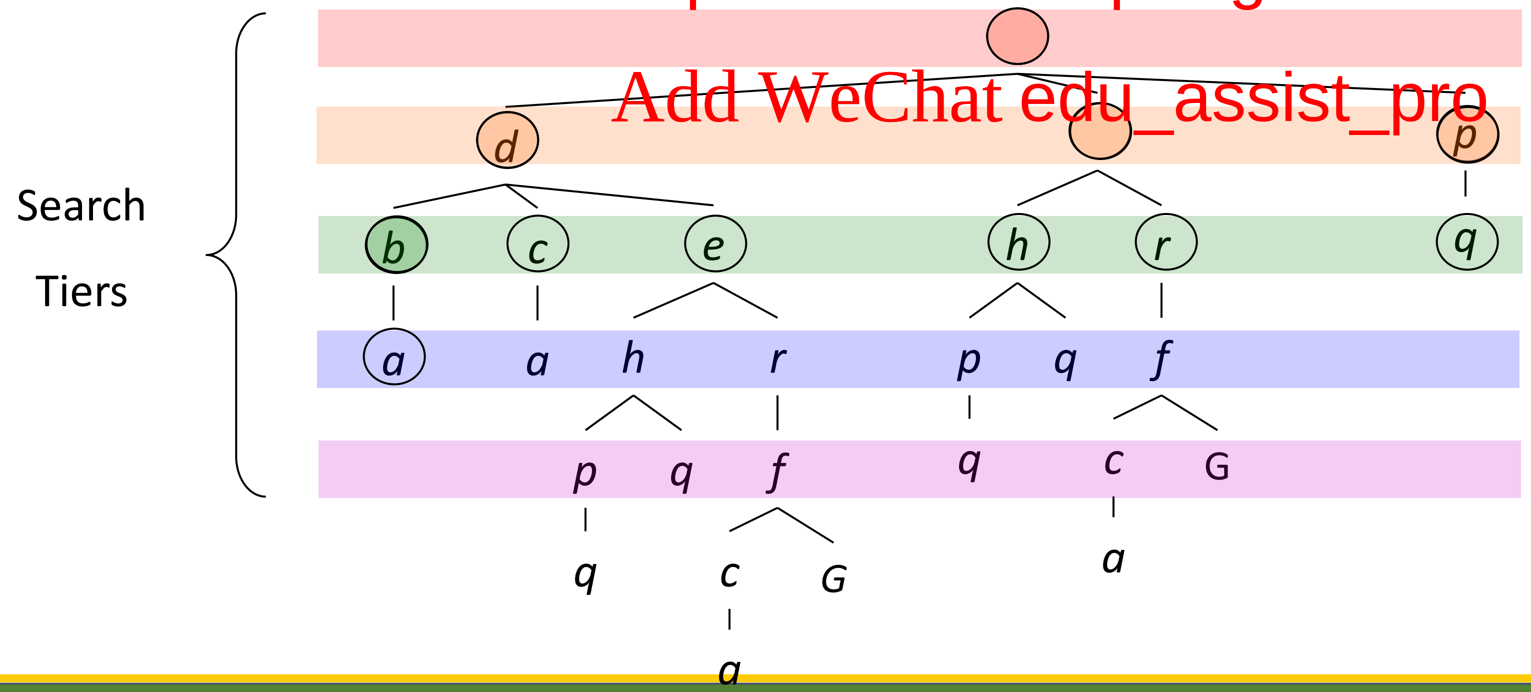
Breadth-First Search(BFS)

Breadth-First Search (BFS)

Strategy: expand a shallowest node first

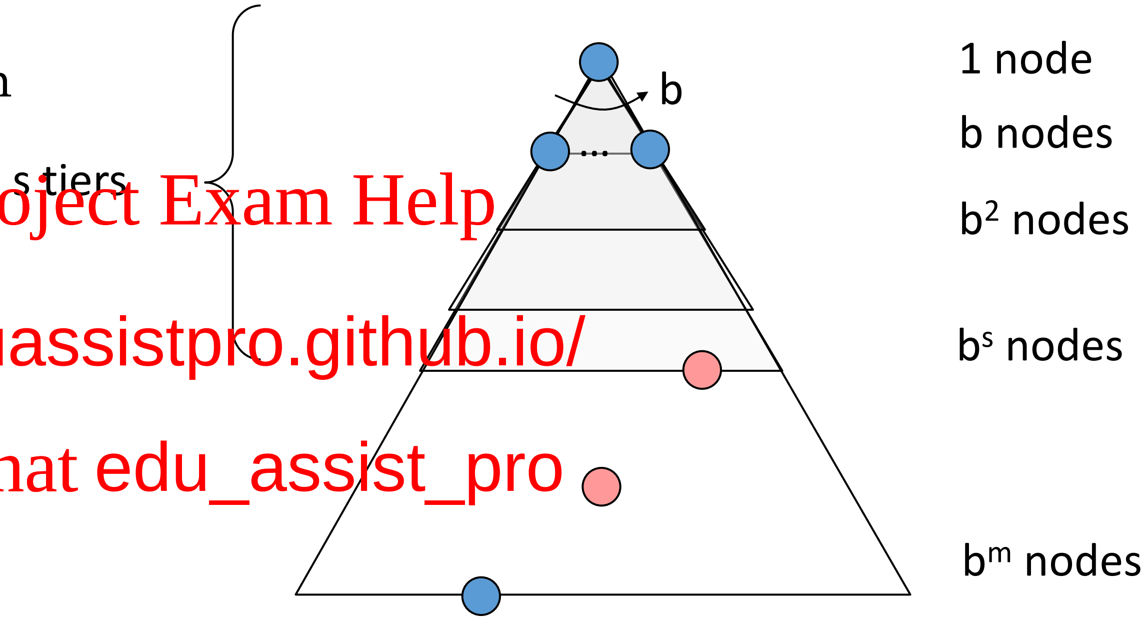
Implementation: Fringe is a FIFO queue





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 fri
  - $O(b^{s+1})$
- 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)



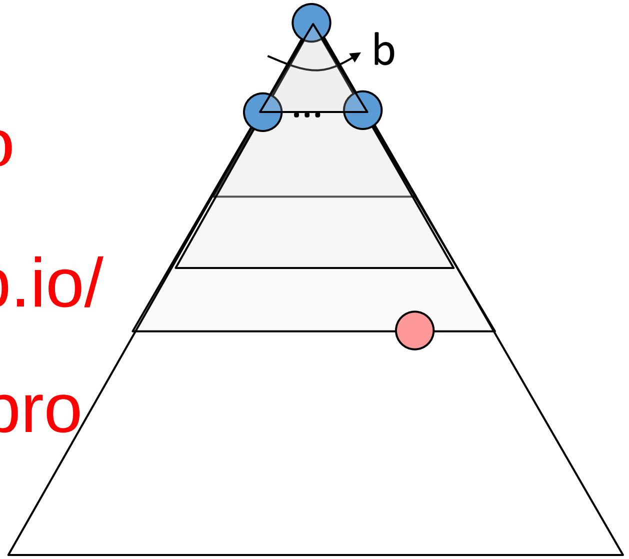
DFS vs BFS

- When will BFS outperform DFS?

- When will DFS outperform

Iterative Deepening

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth li 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!