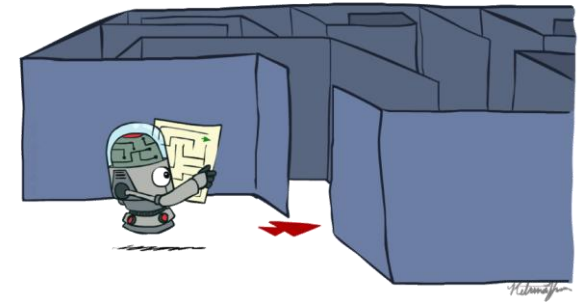


Uninformed Search

CSE 473: Introduction to Artificial Intelligence



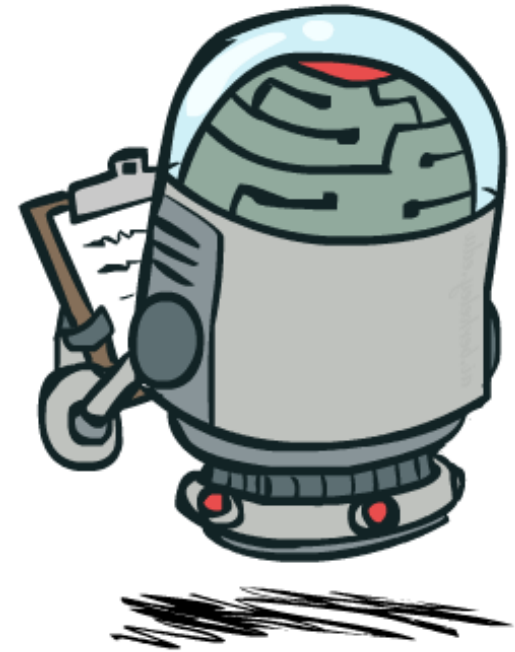
Administrivia

ANNOUNCEMENTS

- Office hours updated on schedule
- Remember to complete the **practice problems** for each lecture
- For each project, make sure to fill out the **AI usage reflection** (cs.uw.edu/473/projects)

ASSIGNMENTS

- **Homework 1** released today
 - due next Monday (10/12)
- **Project 1** released today
 - due in two weeks (10/19)



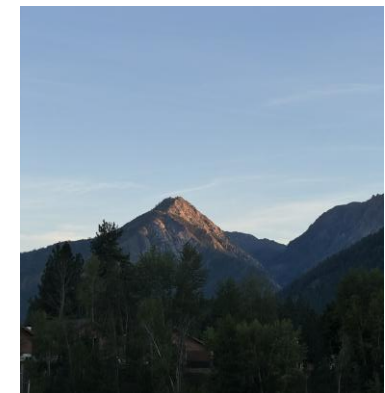
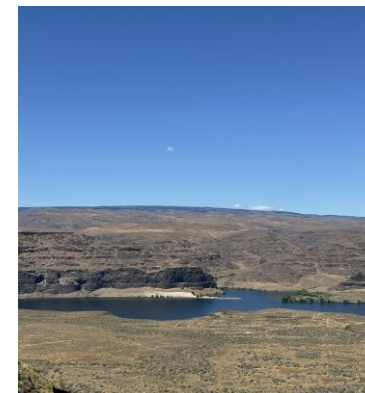
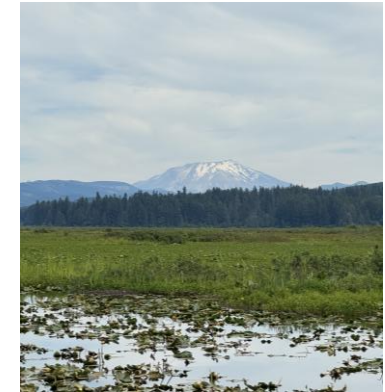
Search

FOUNDATIONS



Washington Search Problem

CONTEXT

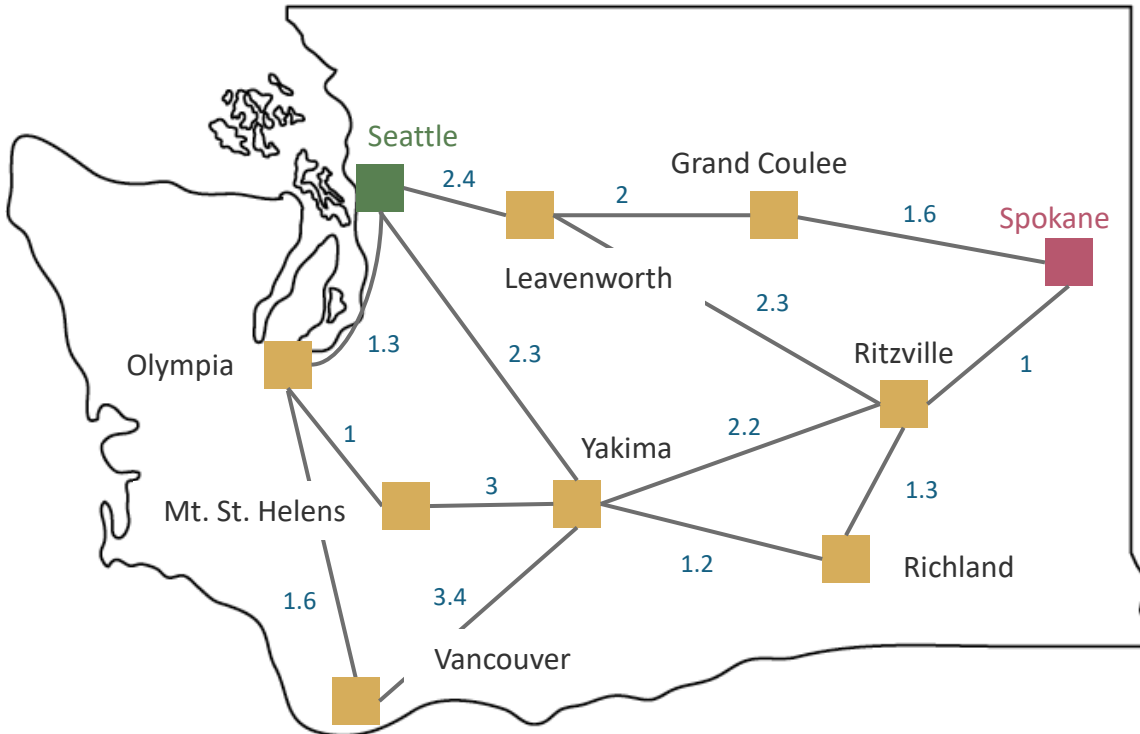


W

Spokane to Seattle



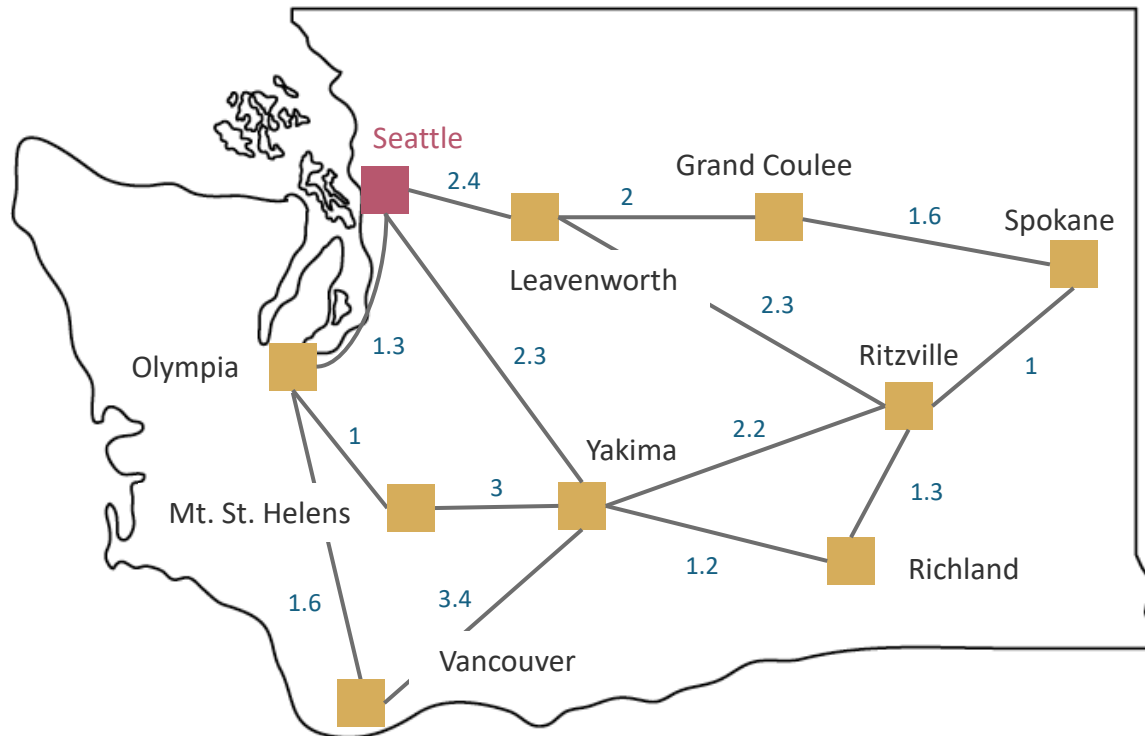
EXAMPLE



- state space
 - cities
- successor function
 - roads connect adjacent cities with cost = travel time
- start state
 - **Spokane**
- goal test
 - state == **Seattle**?

Touring Washington

EXAMPLE



- state space
 - Cities & (**visited** boolean array)
- successor function
 - roads connect adjacent cities with cost = travel time
- start state
 - **Seattle**
- goal test
 - **visited** is all **true** and city == **Seattle**

State Space Graphs

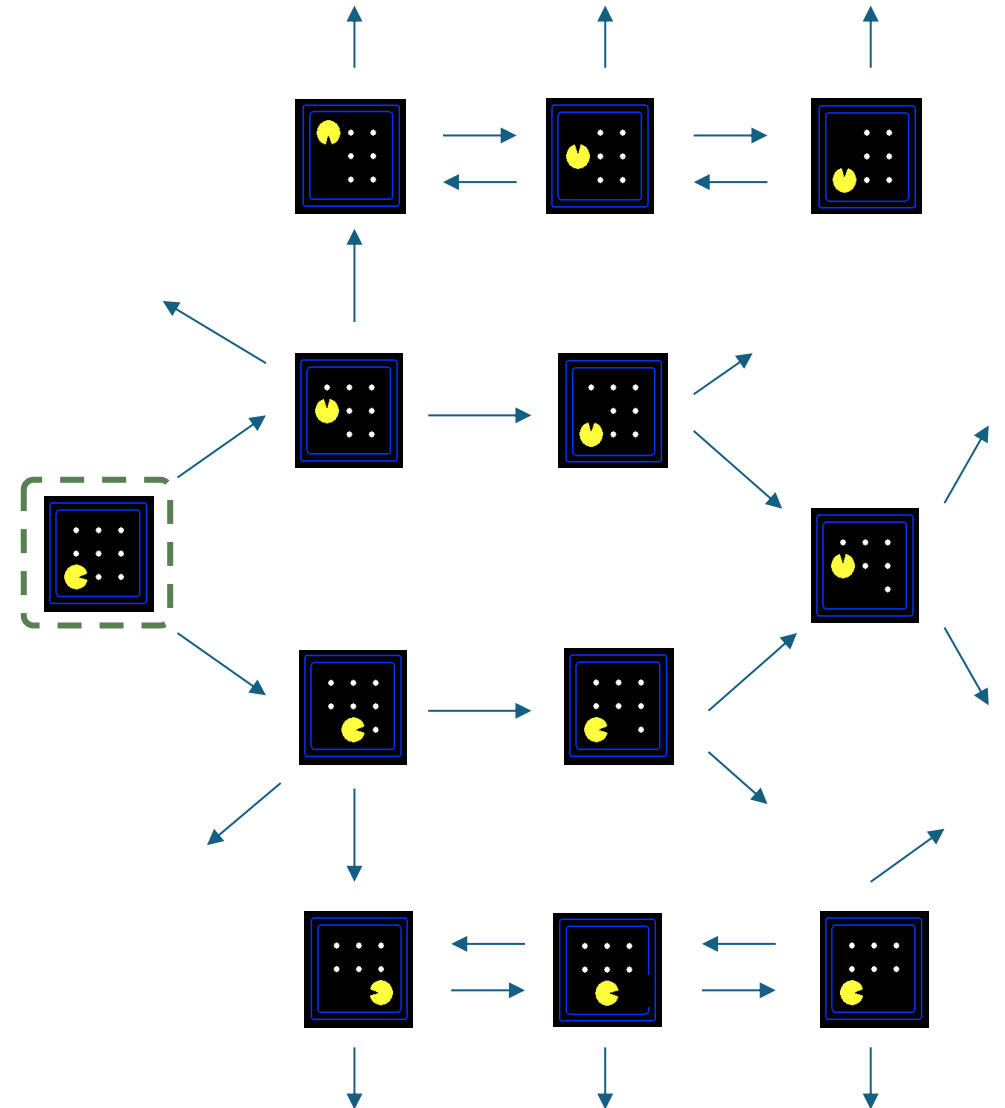
FOUNDATIONS

Represent transitions between states as a graph

- **Nodes** are states
- **Arcs** represent successors (action results)
- **Goals** are a set of nodes
- Each state is represented once

Can we build the full graph?

Can't usually fit in memory



Search Trees

FOUNDATIONS

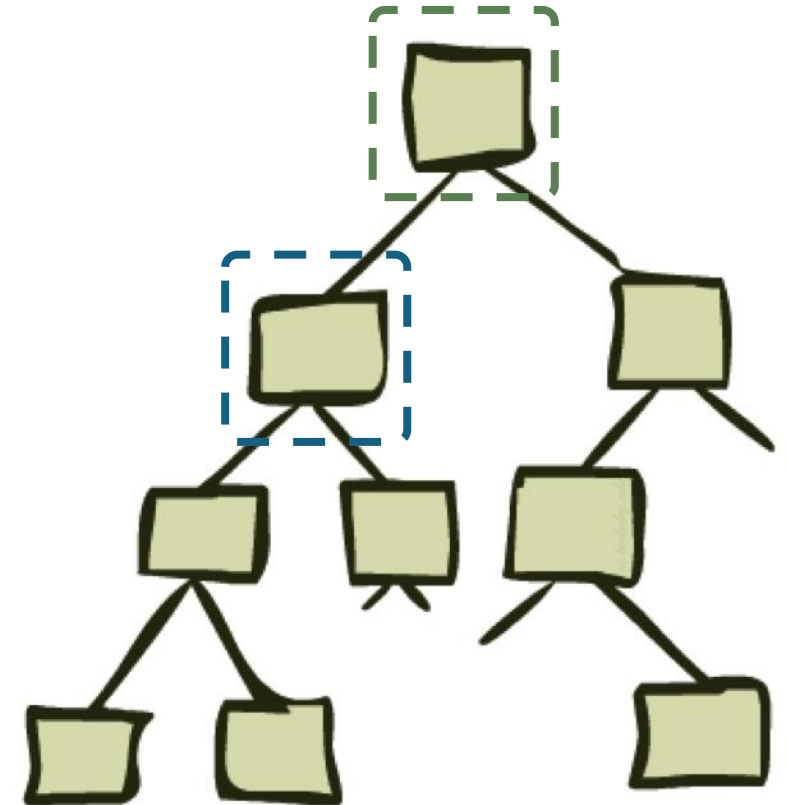


Represent sequences of actions on paths to root

- **Nodes** are states
- **Children** represent successors (action results)
- States *can* be repeated

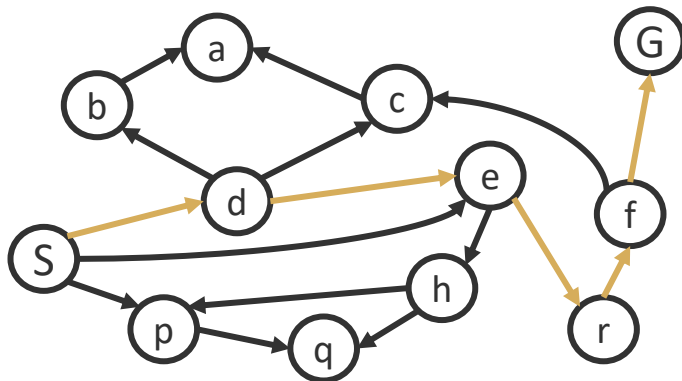
Can we build the full tree?

Not always possible...



Graphs vs. Trees

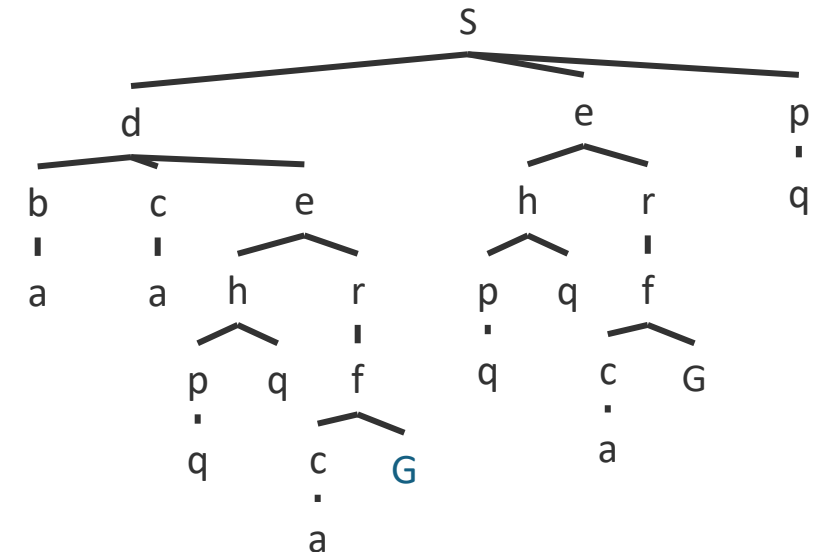
State Space Graph



Each **node** in a search tree represents an **entire path** from the root.

Trees get very large, so we construct on demand and as little as possible.

Search Tree



```

graph LR
    S((S)) --> a((a))
    S((S)) --> b((b))
    a((a)) --> b((b))
    b((b)) --> a((a))
    a((a)) --> G((G))
    b((b)) --> G((G))

```

...

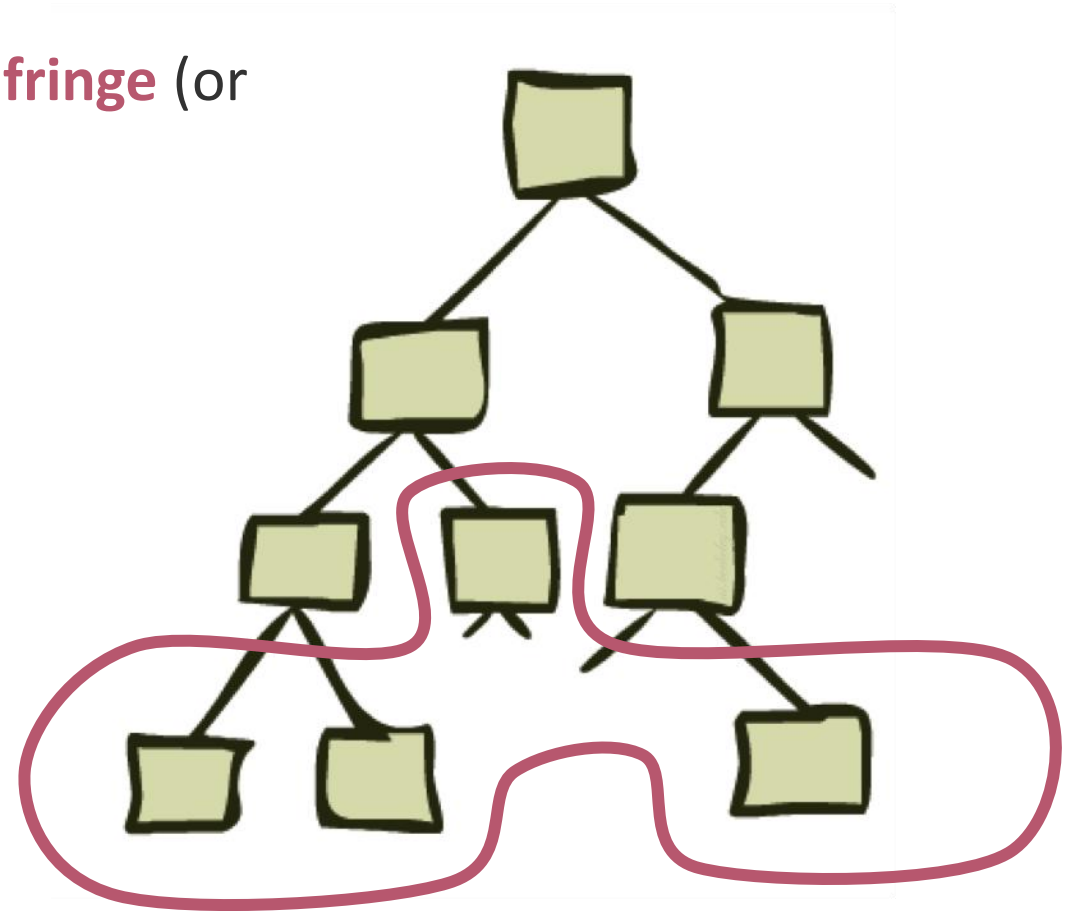
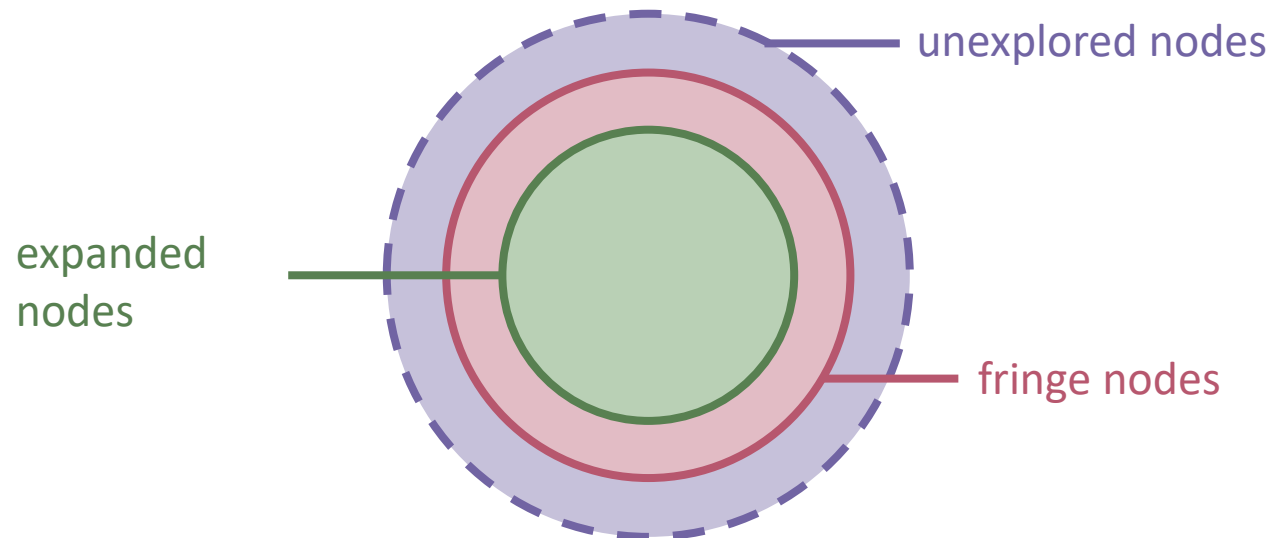
Questions?

Tree Search

DEFINITION

Tree search is a search algorithm that maintains a **fringe** (or frontier) of nodes to consider exploring

- Nodes on the fringe are **expanded** to find new nodes



General Tree Search

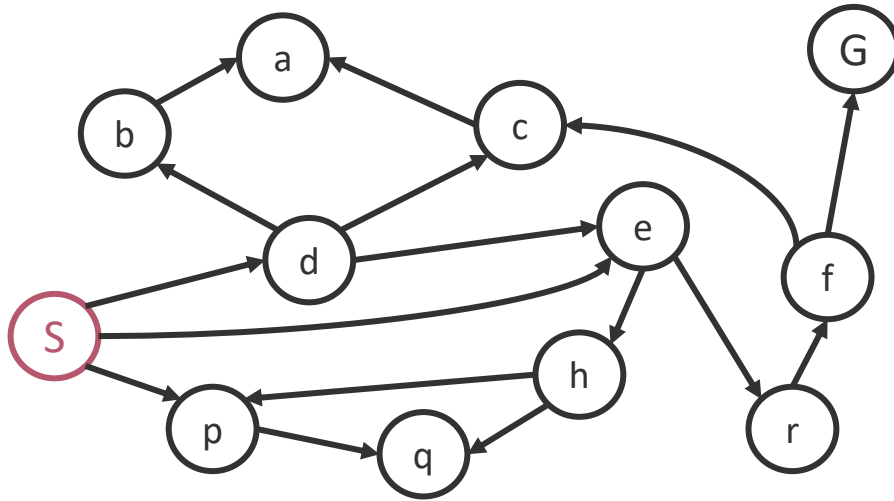
ALGORITHM

```
function TREE-SEARCH(problem, strategy) returns solution or failure
  initialize fringe using initial state of problem
  loop do
    if fringe empty then return failure
    node ← remove node from fringe according to strategy
    if node is goal state then return corresponding solution
    expand node and add resulting nodes to fringe
  end
```

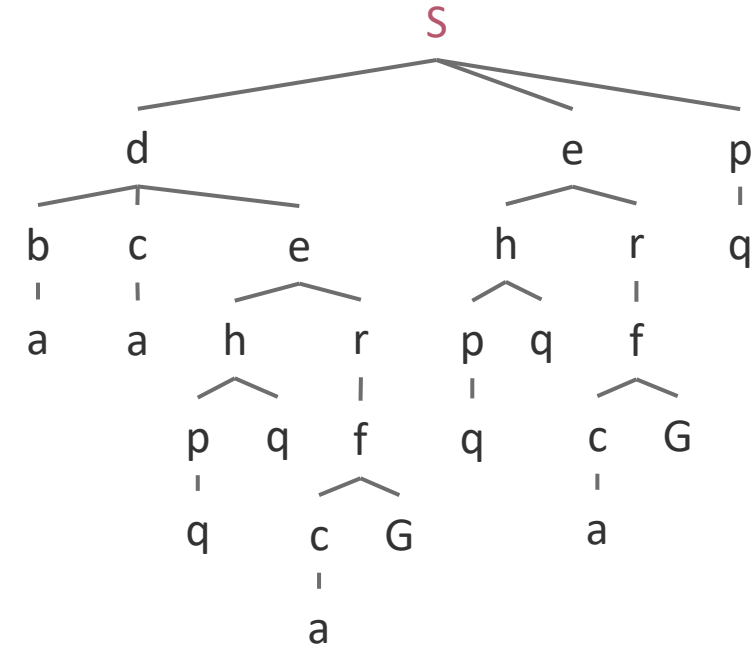
Strategy: How to order fringe?

Random Exploration: Step 1

EXAMPLE



Expanded

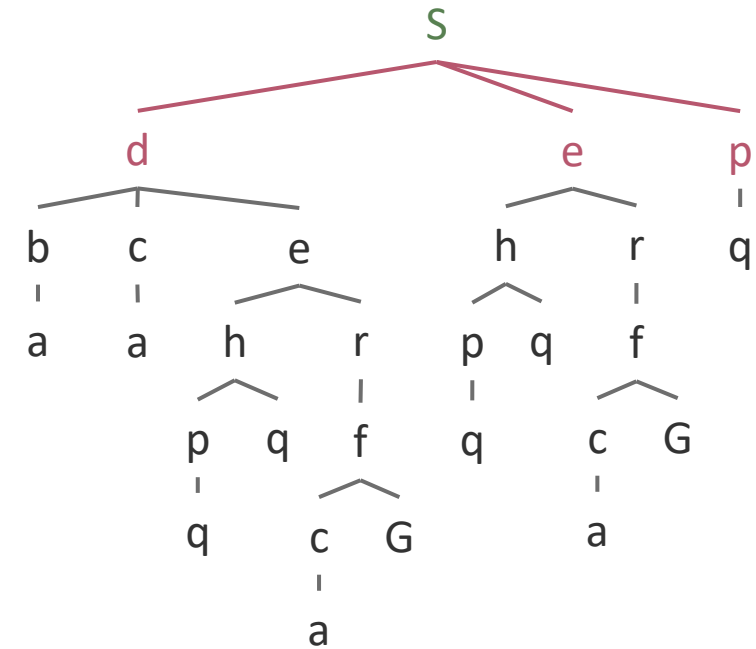
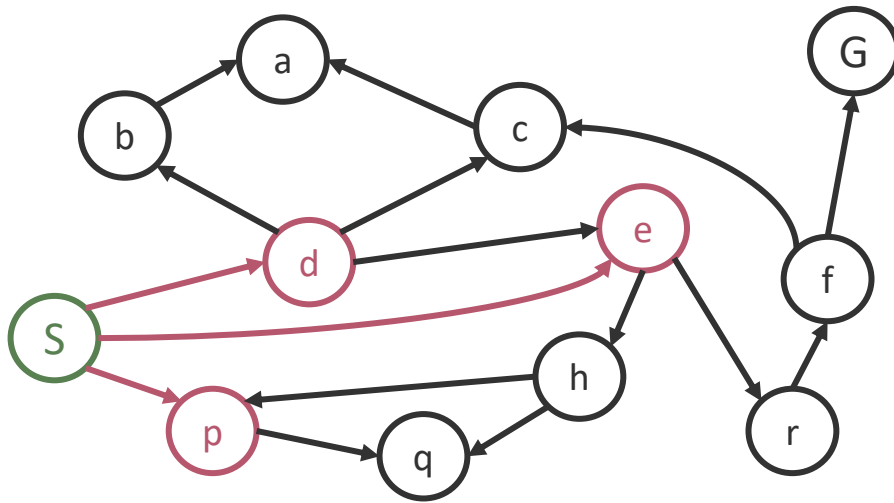


Fringe

S

Random Exploration: Step 2

EXAMPLE



Expanded

S

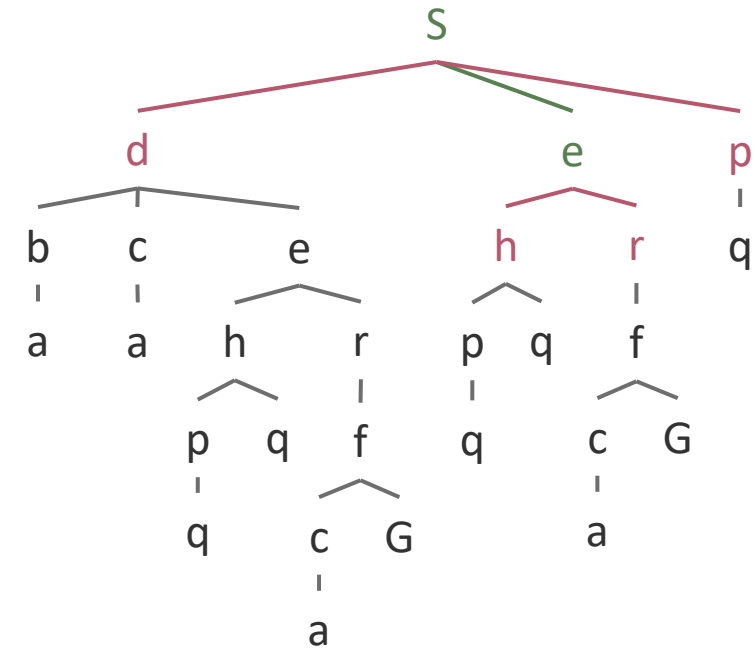
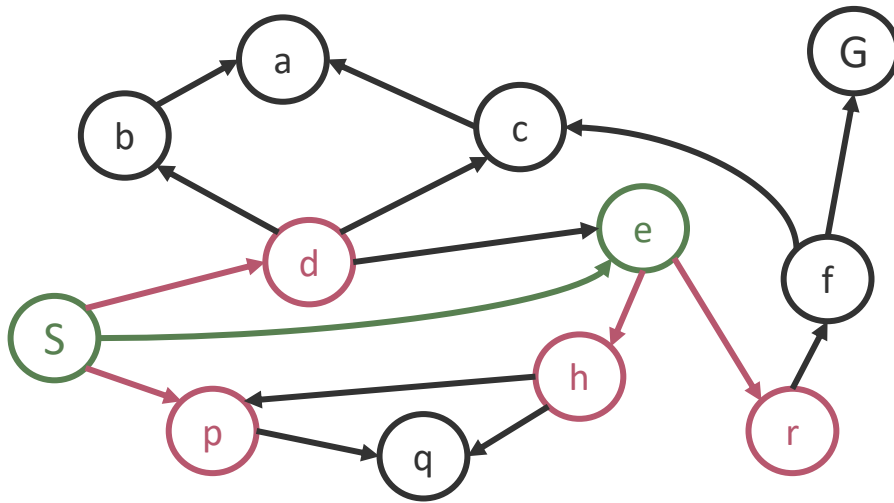
Expand
S

Fringe

d, e, p

Random Exploration: Step 3

EXAMPLE



Expanded

S, e

Expand

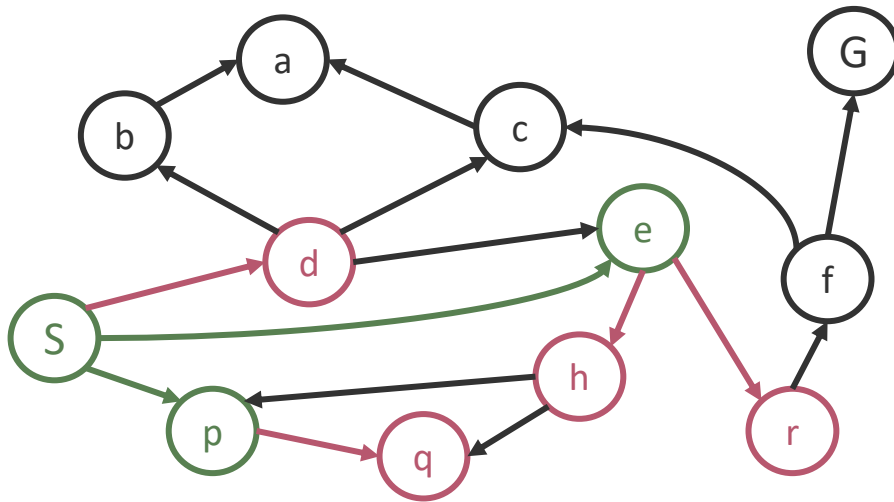
e

Fringe

d, p, h, r

Random Exploration: Step 4

EXAMPLE



Expanded

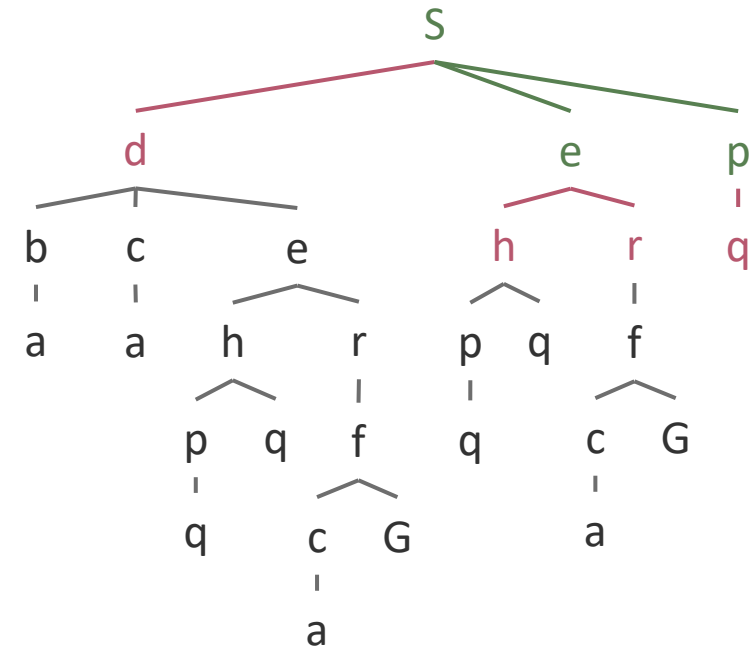
S, e, p

Expand

p

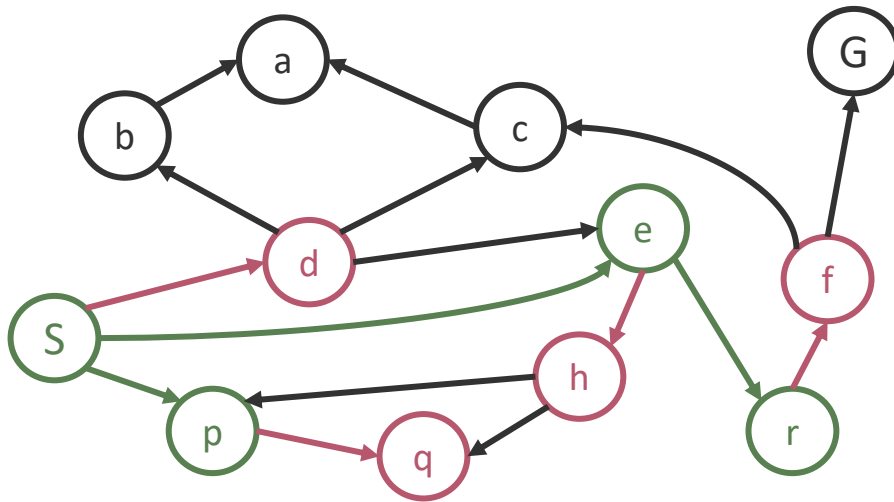
Fringe

d, h, r, q



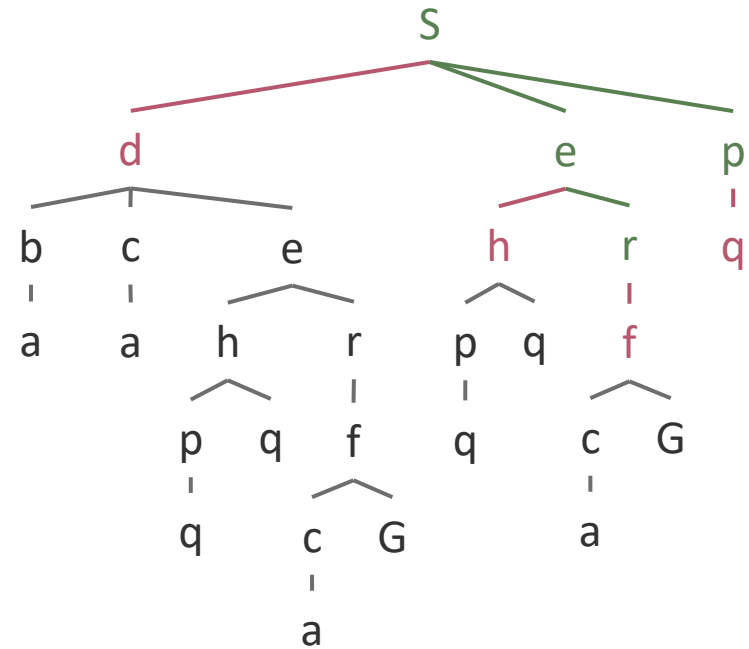
Random Exploration: Step 5

EXAMPLE



Expanded

S, e, p, r

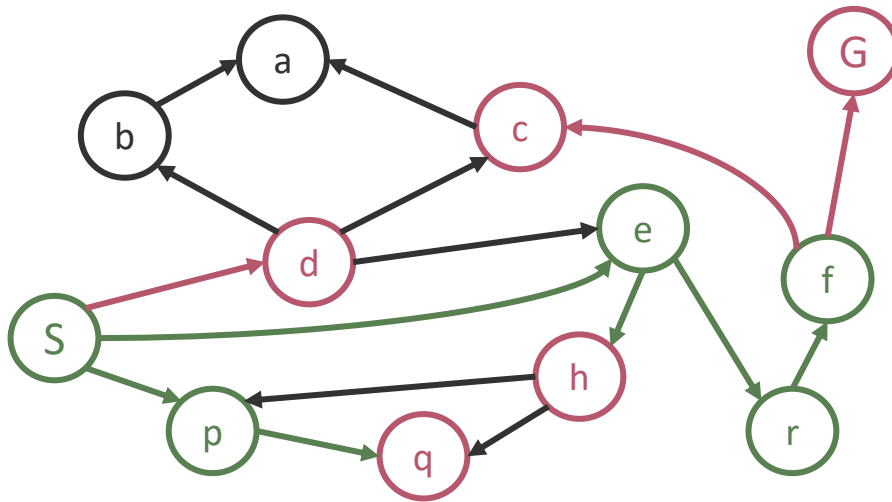
Expand
r

Fringe

d, h, q, f

Random Exploration: Step 6

EXAMPLE



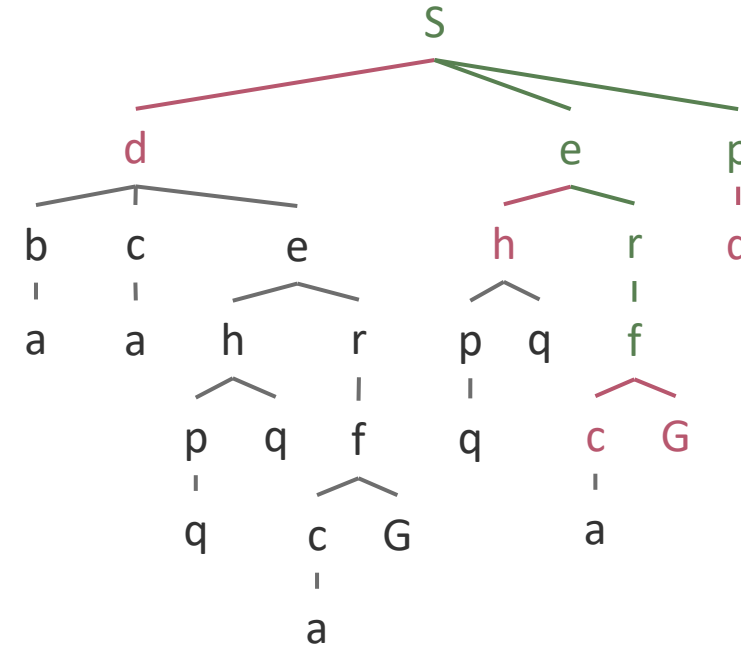
Expanded

S, e, p, r, f

Expand
f

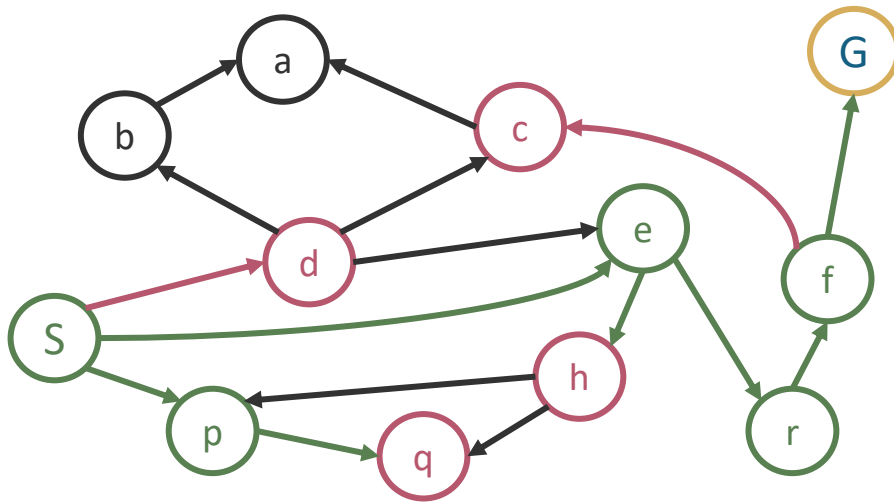
Fringe

d, h, q, c, G



Random Exploration: Step 7

EXAMPLE

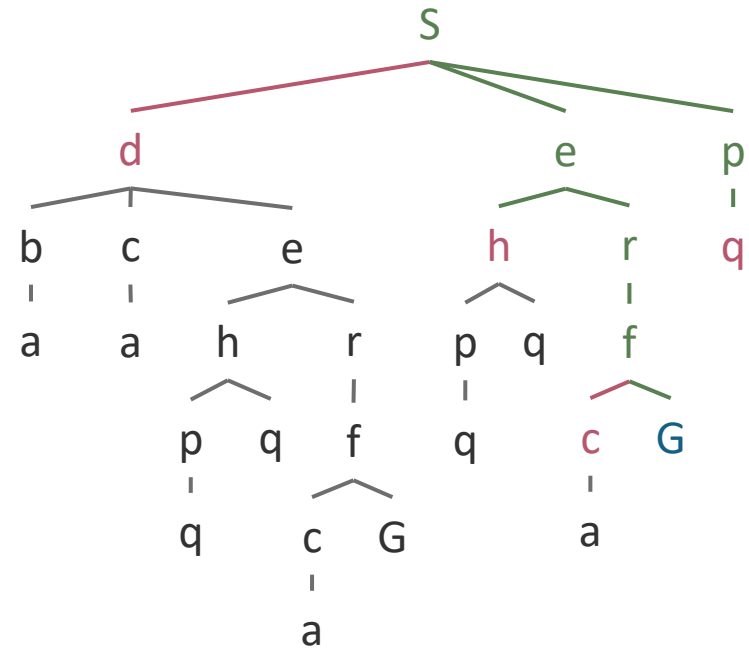


Expanded

S, e, p, r, f

“Expand”

G



Fringe

d, h, q, c

Questions?

Properties of DFS

DEFINITION

- Complete?

Complete: guaranteed to find a solution if it exists.

- m could be infinite, so only if no cycles

- Optimal?

Optimal: finds lowest cost solution

- No. DFS does not care about cost!

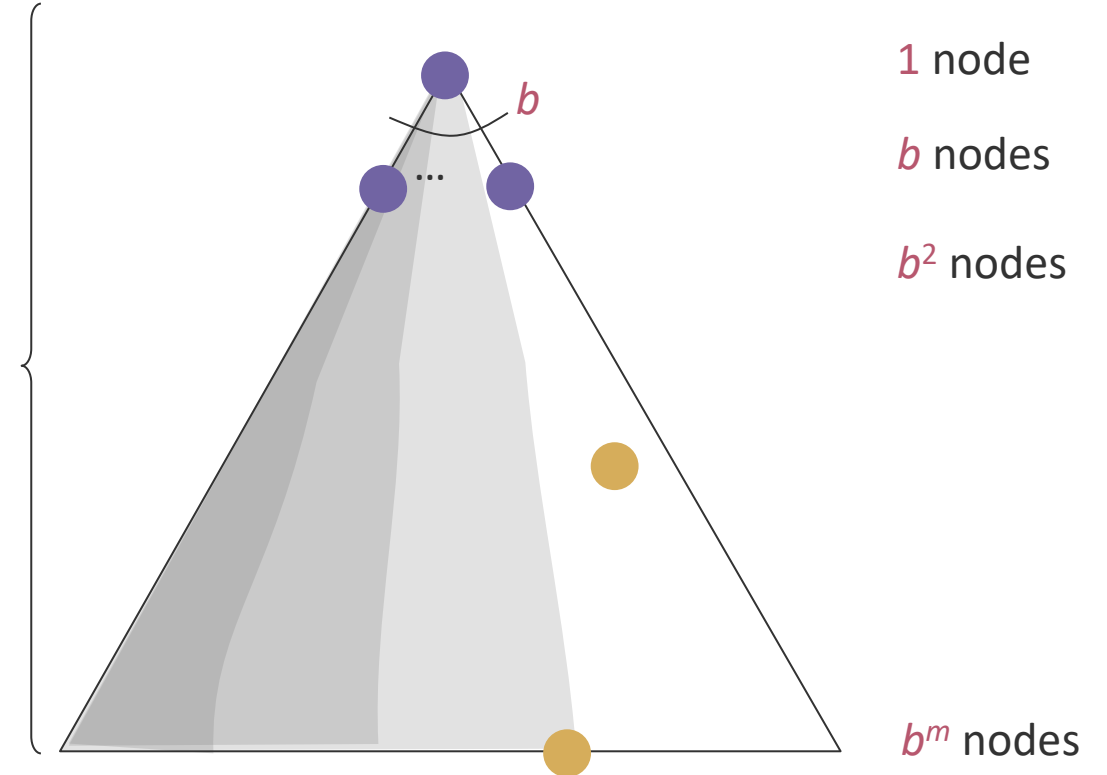
m tiers

- Time Complexity

- $O(bm)$ if m is finite

- Space Complexity

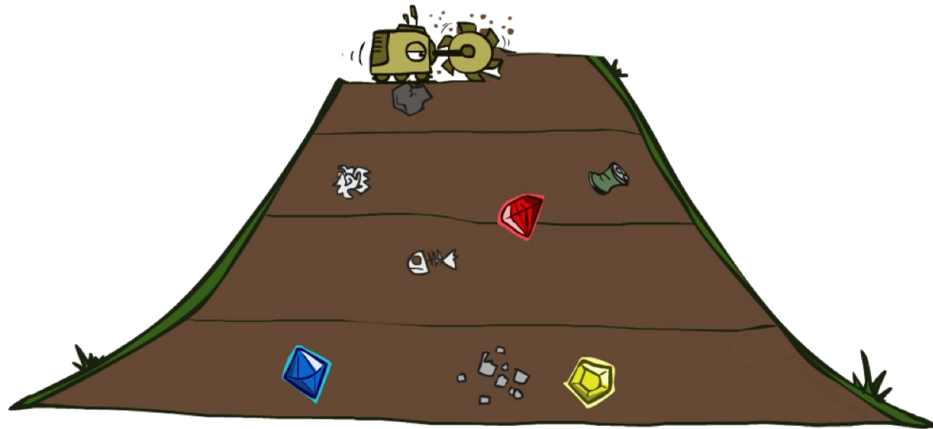
- $O(bm)$



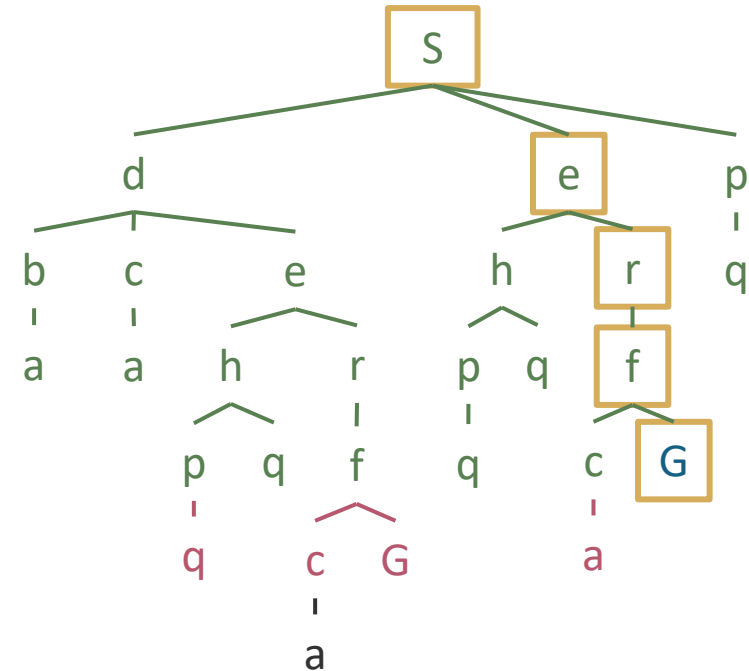
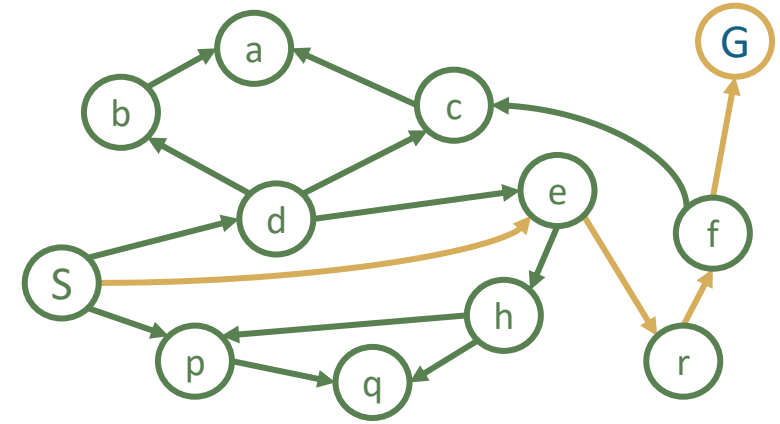
Breadth-First Search

FOUNDATIONS

- Strategy
 - Expand **shallowest** (oldest) node first
- Implementation
 - Fringe is a **FIFO queue**
 - Usually break ties alphabetically



“wait your turn!”

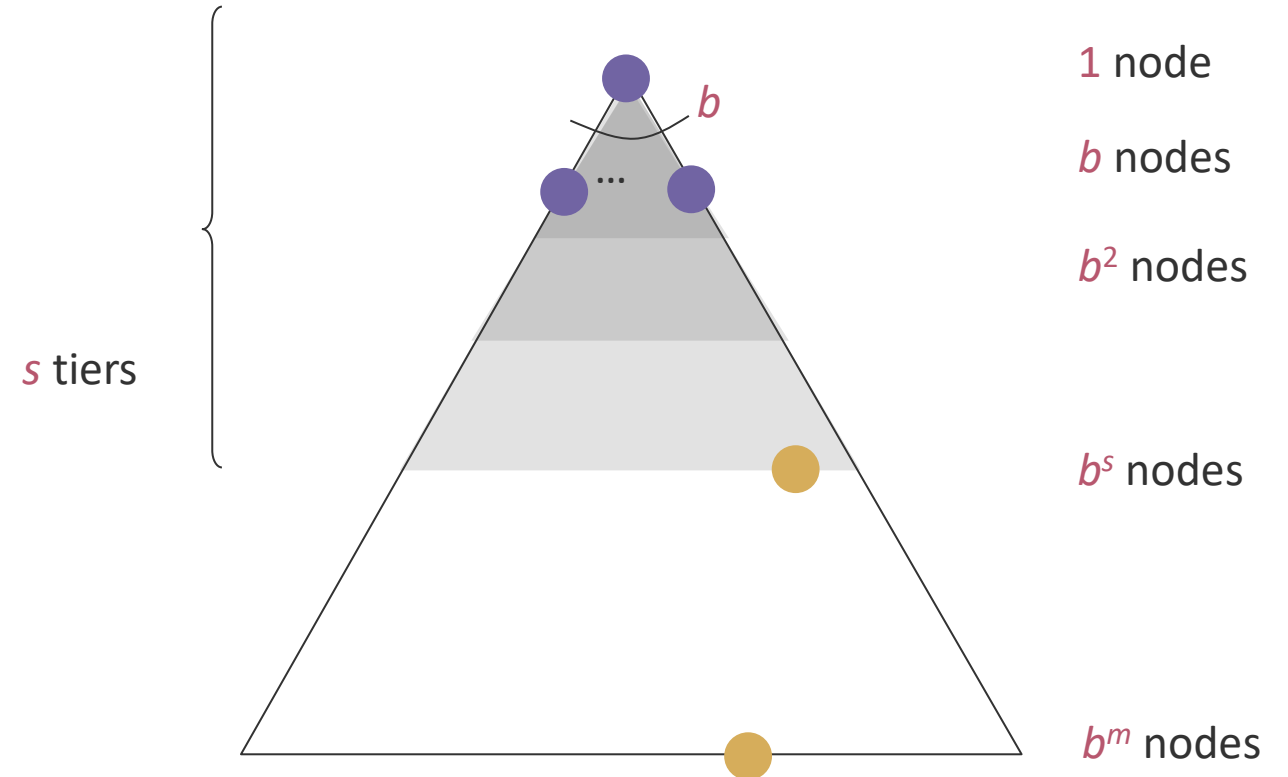


Properties of BFS

DEFINITION

- Complete?
 - Yes (s must be finite for a solution to exist)
- Optimal?
 - Only if costs are all 1
- Time Complexity
 - $O(b^s)$
- Space Complexity
 - $O(b^s)$ (roughly all nodes in tier s)

s = depth of shallowest solution



DFS vs. BFS?

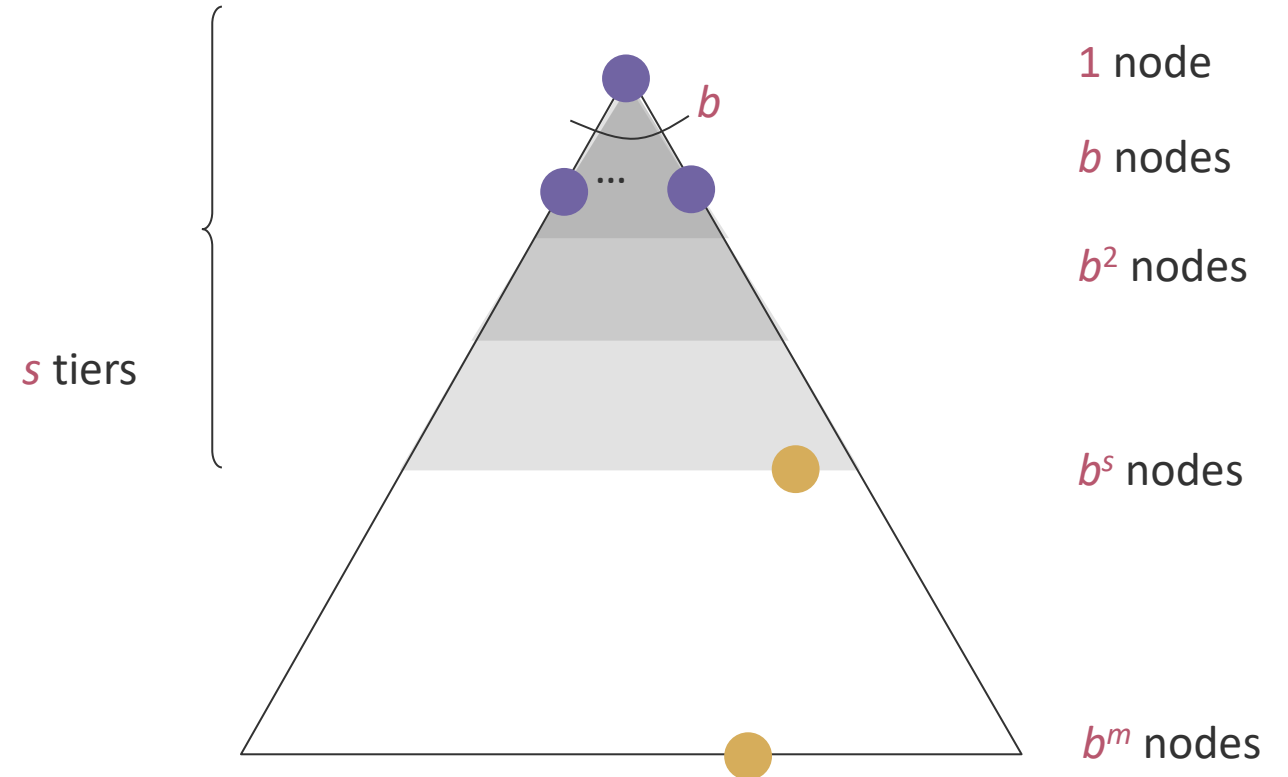
Iterative Deepening

FOUNDATIONS



s = depth of shallowest solution

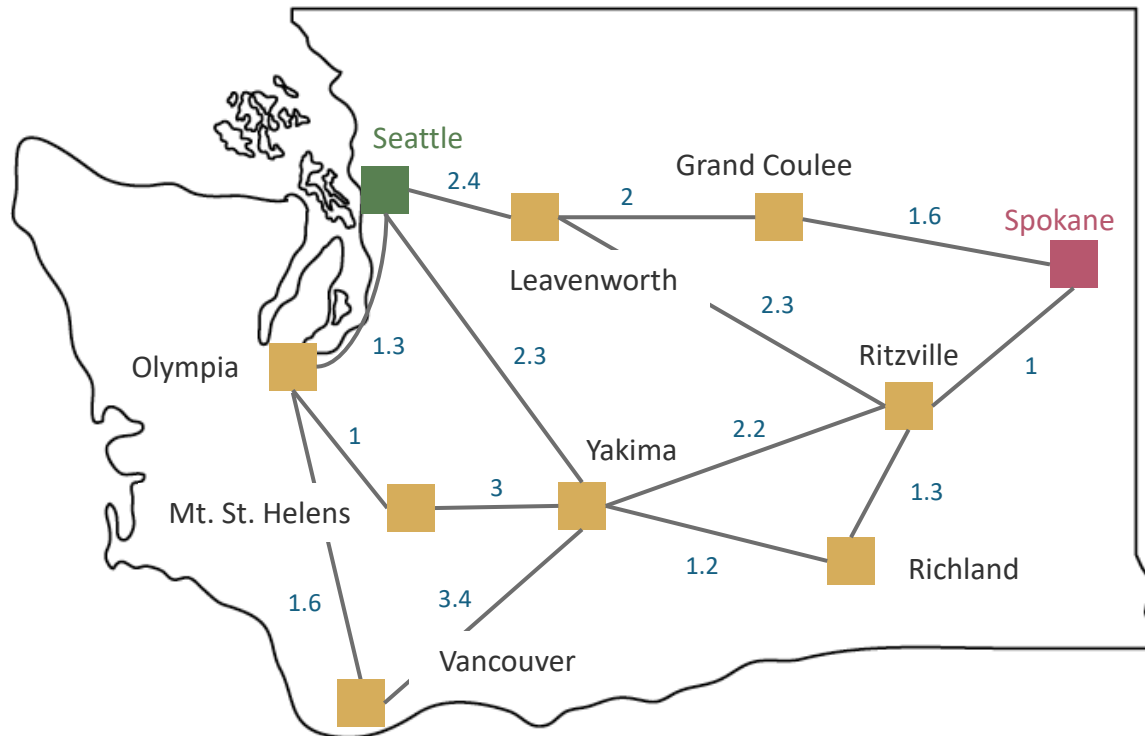
- Idea
 - Combine space advantage of DFS with BFS time advantage
- Implementation
 - Run DFS with depth 1
 - Run DFS with depth 2
 - Run DFS with depth 3
 - ...



Actions Have Costs...



EXAMPLE

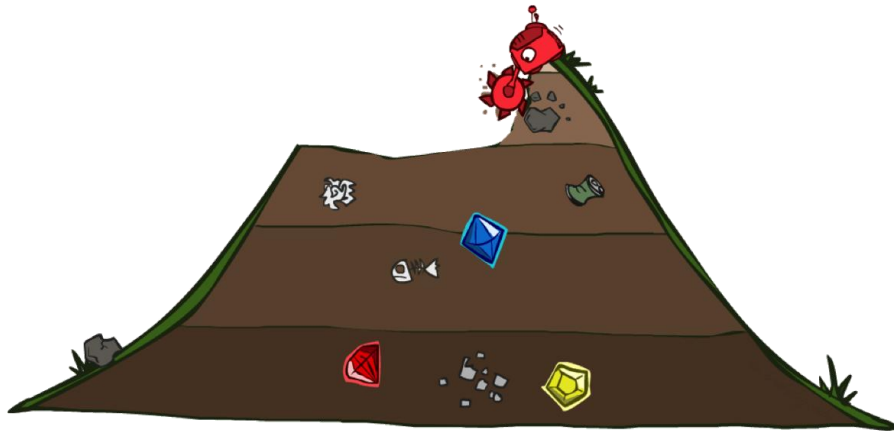


How to find the lowest cost path?

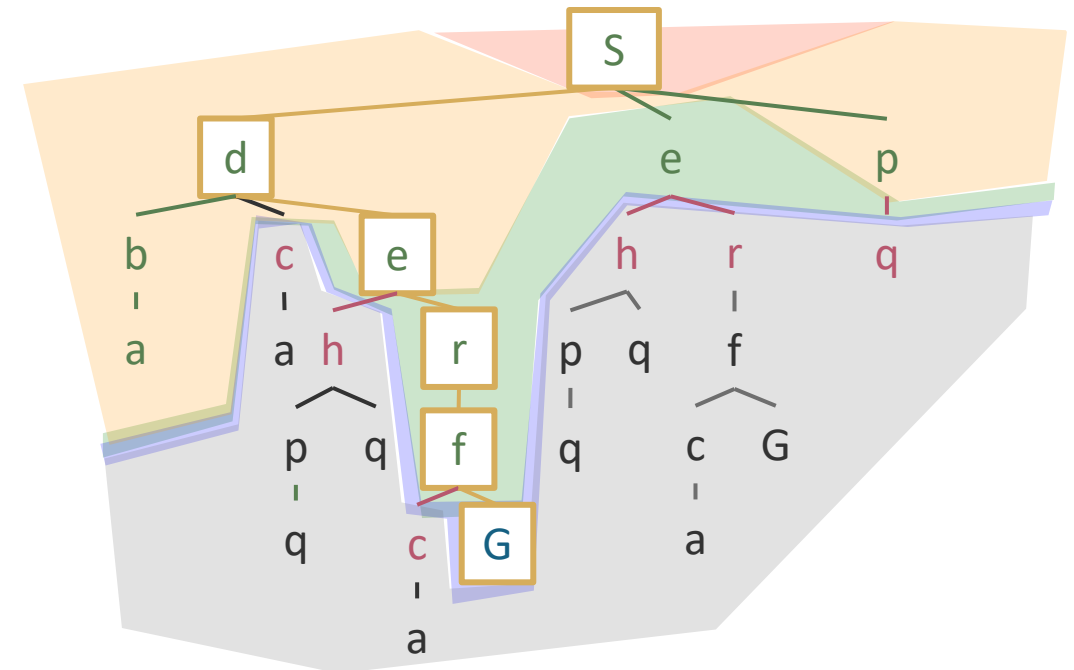
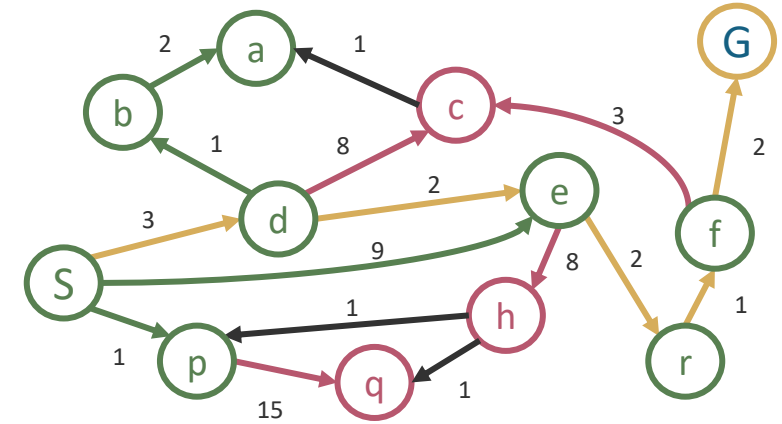
Uniform Cost Search

FOUNDATIONS

- Strategy
 - Expand **cheapest** (least cost) node first
- Implementation
 - Fringe is a **priority queue**
 - Usually break ties alphabetically



“how much does it cost?”

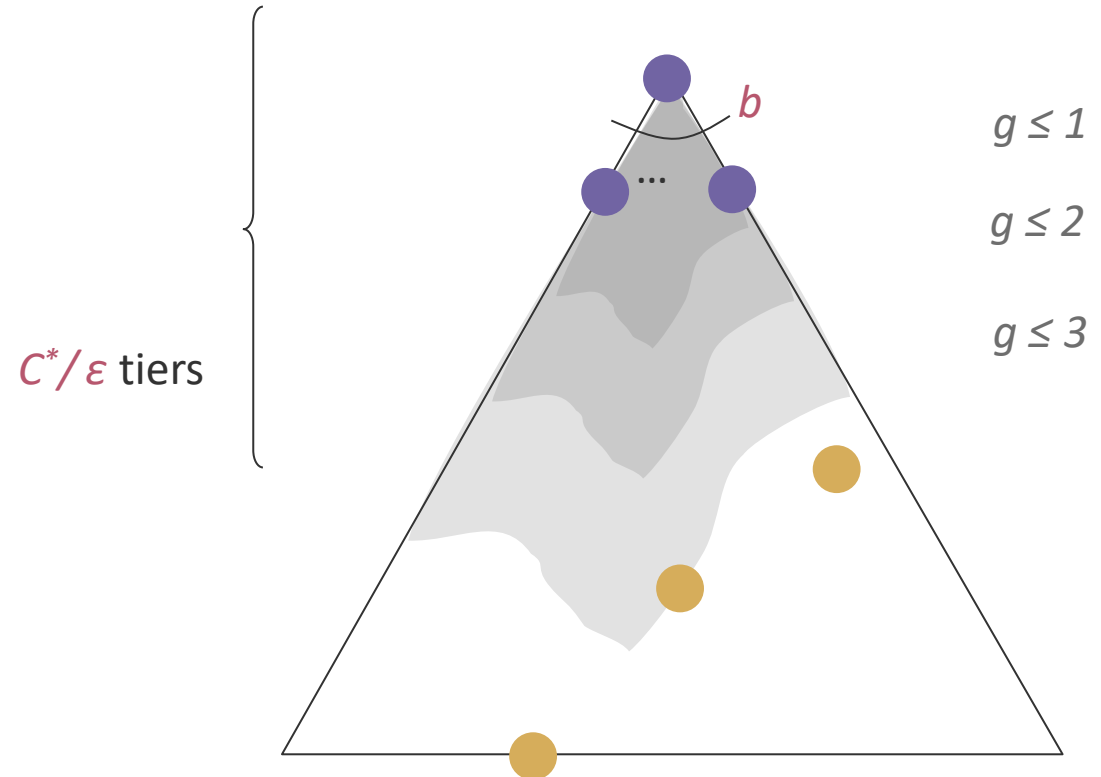


Properties of UCS

DEFINITION

- Complete?
 - Yes (assuming C^* is finite and $\epsilon > 0$)
- Optimal?
 - Yes!
- Time Complexity
 - $O(b^{C^*/\epsilon})$
- Space Complexity
 - $O(b^{C^*/\epsilon})$ (roughly last tier)

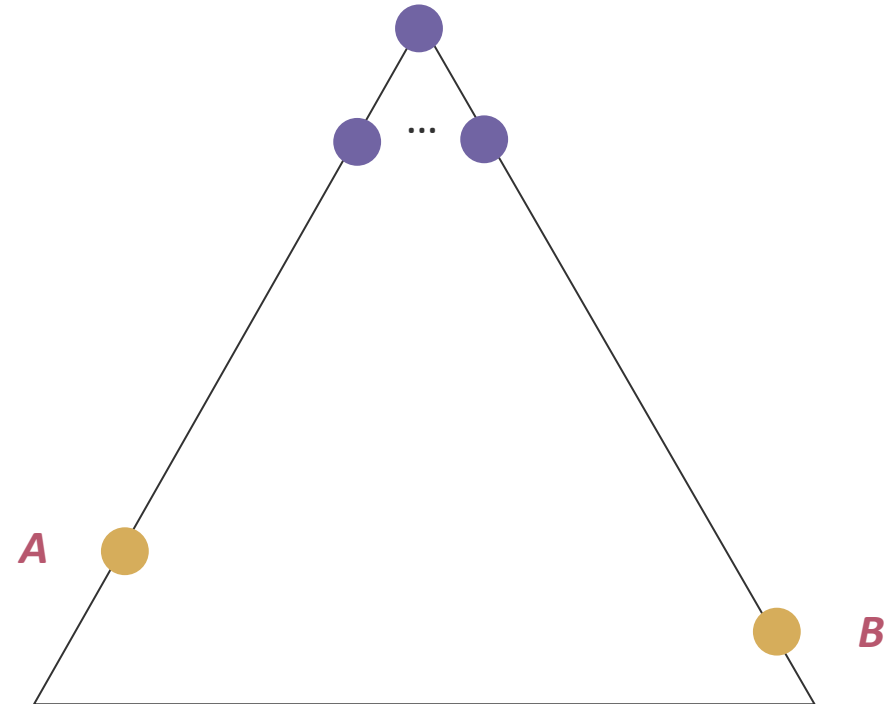
$g(n)$ = cost from root to node n
 C^* = solution cost
 ϵ = minimum arc cost
effective depth = C^*/ϵ



Optimality of UCS Tree Search

PROOF

- Assume
 - *A* is an optimal goal node
 - *B* is a suboptimal goal node
- Claim
 - *A* will be chosen for expansion before *B*

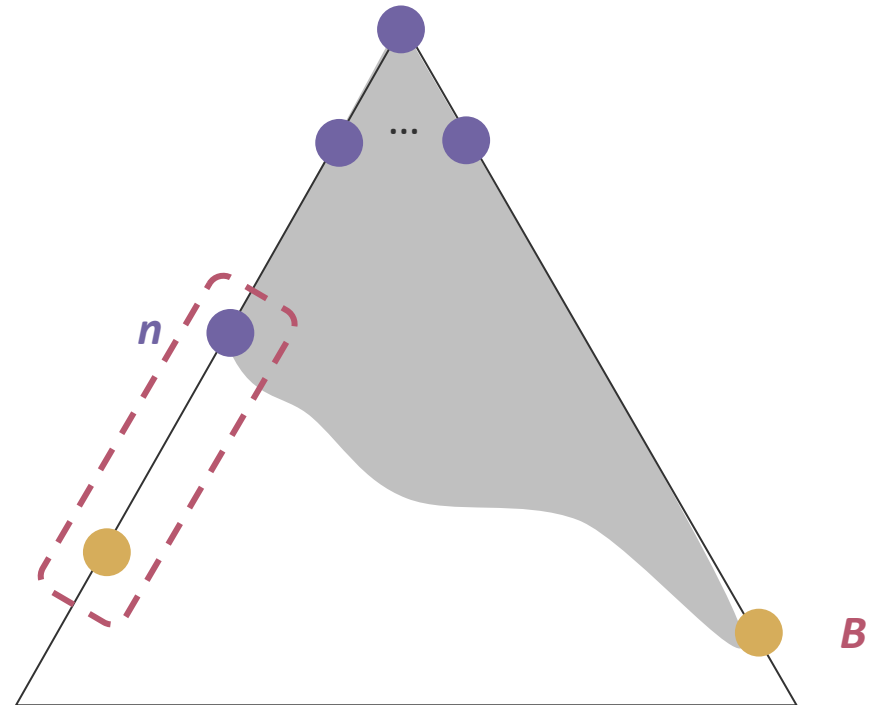


Optimality of UCS Tree Search (1)

PROOF

- Proof
 - Let B be on the fringe
 - Some ancestor n of A is on the fringe, too
 - (maybe A itself)
- Claim
 - n will be expanded before B
- Proof
 - $g(n) \leq g(A)$

$$\epsilon > 0$$

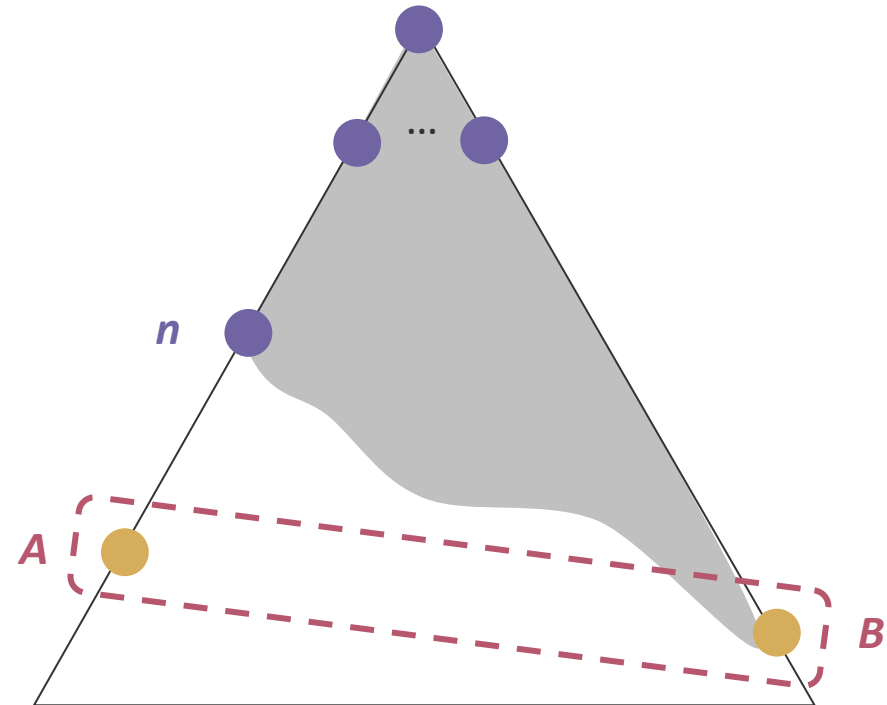


Optimality of UCS Tree Search (2)

PROOF

- Proof
 - Let B be on the fringe
 - Some ancestor n of A is on the fringe, too
 - (maybe A itself)
- Claim
 - n will be expanded before B
- Proof
 - $g(n) \leq g(A)$
 - $g(A) < g(B)$
 - $\rightarrow g(n) \leq g(A) < g(B)$

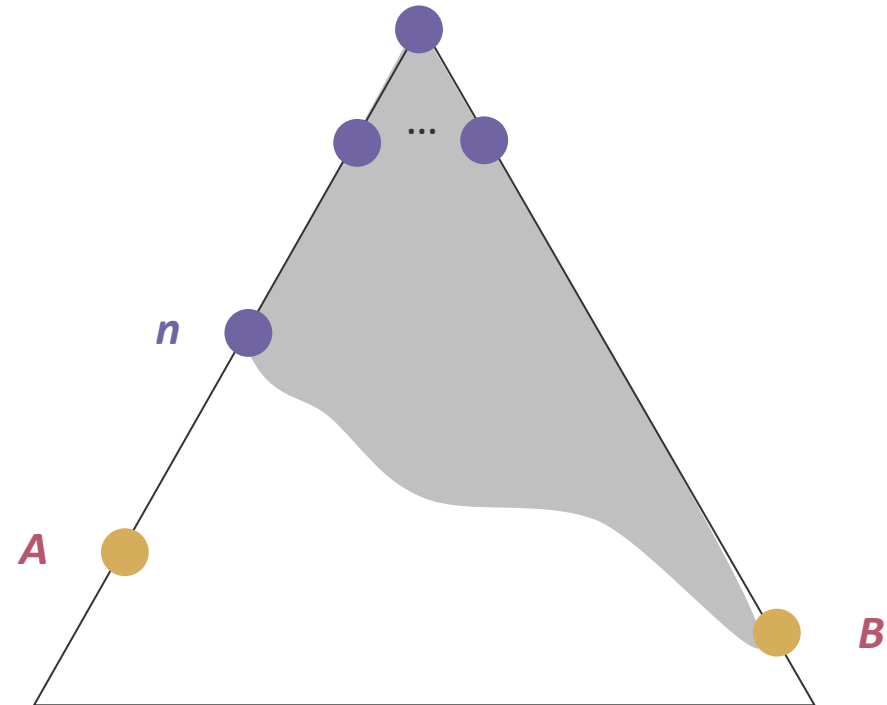
B is suboptimal



Optimality of UCS Tree Search (3)

PROOF

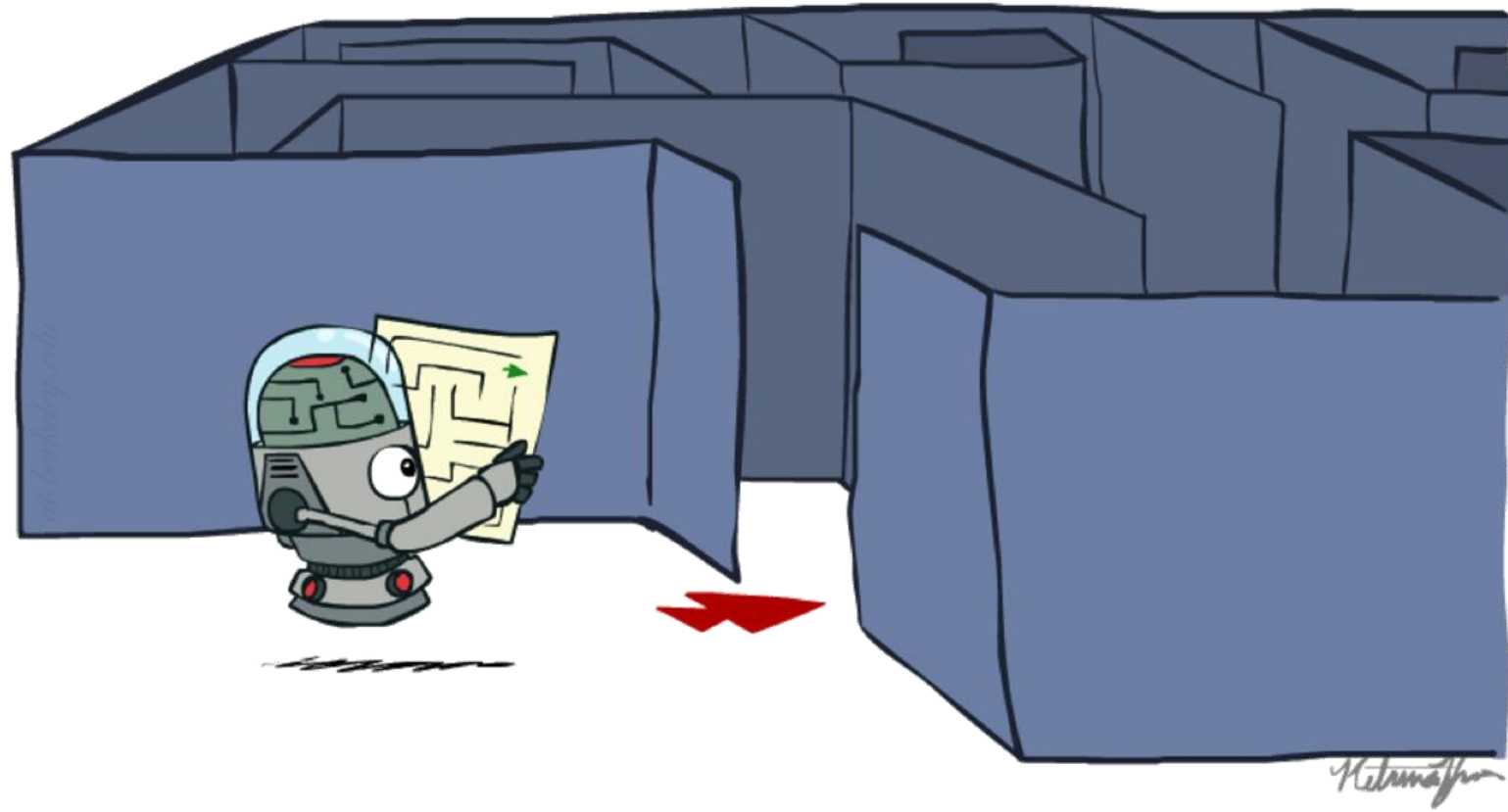
- Proof
 - Let B be on the fringe
 - Some ancestor n of A is on the fringe, too
 - (maybe A itself)
- Proof
 - $\rightarrow g(n) \leq g(A) < g(B)$
- Conclusion
 - All ancestors of A are expanded before B
 - Therefore A itself is expanded before B
 - Therefore **UCS tree search is optimal**

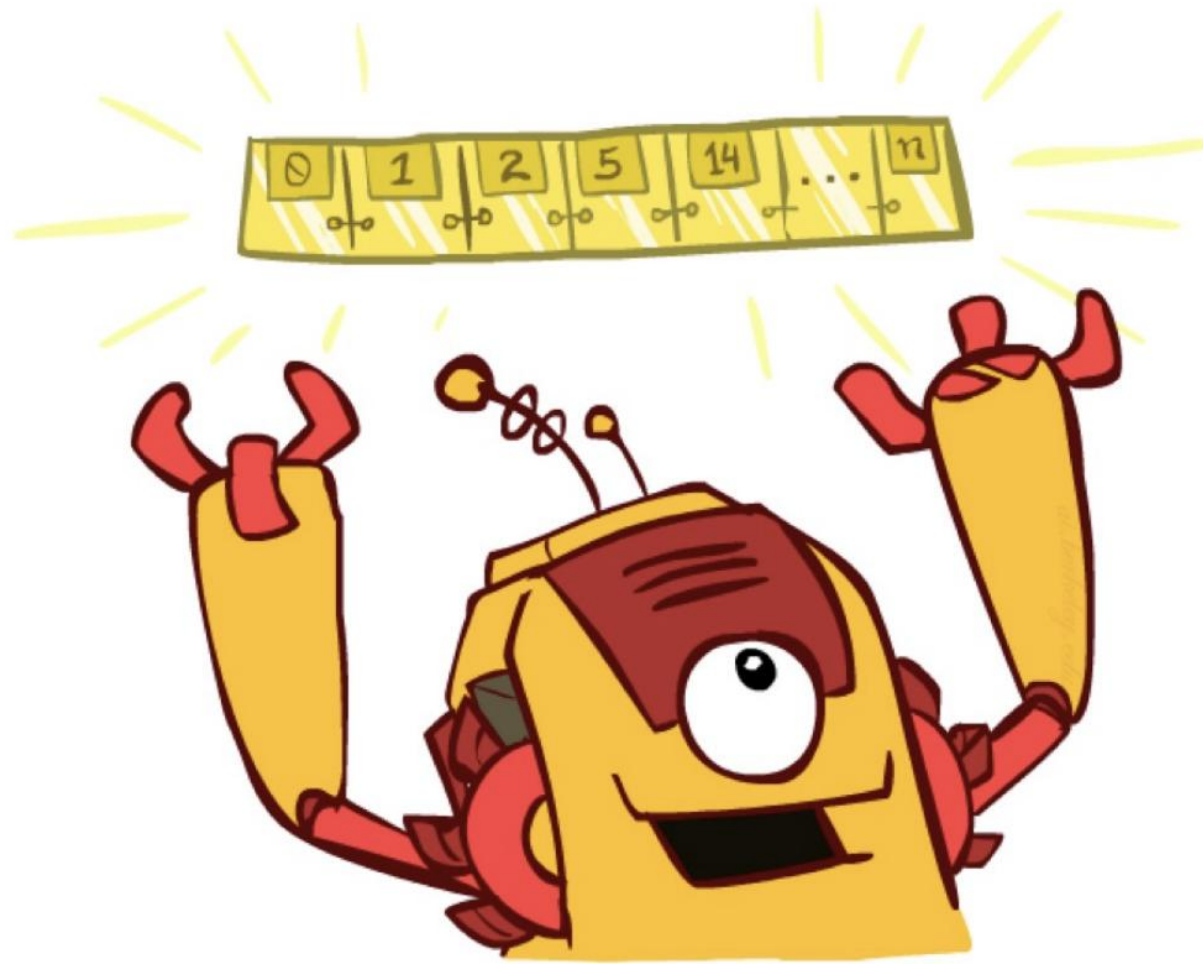


Questions?

Pacman Wayfinding

EXAMPLE





The One (Priority) Queue

General Graph Search



ALGORITHM

```
function GRAPH-SEARCH(problem, strategy) returns solution or failure
  closed ← empty set
  initialize fringe using initial state of problem
  loop do
    if fringe empty then return failure
    node ← remove node from fringe
    if node is goal state then return corresponding solution
    if node not in closed then
      expand node and add resulting nodes to fringe
      add node to closed
  end
```

General Tree Search

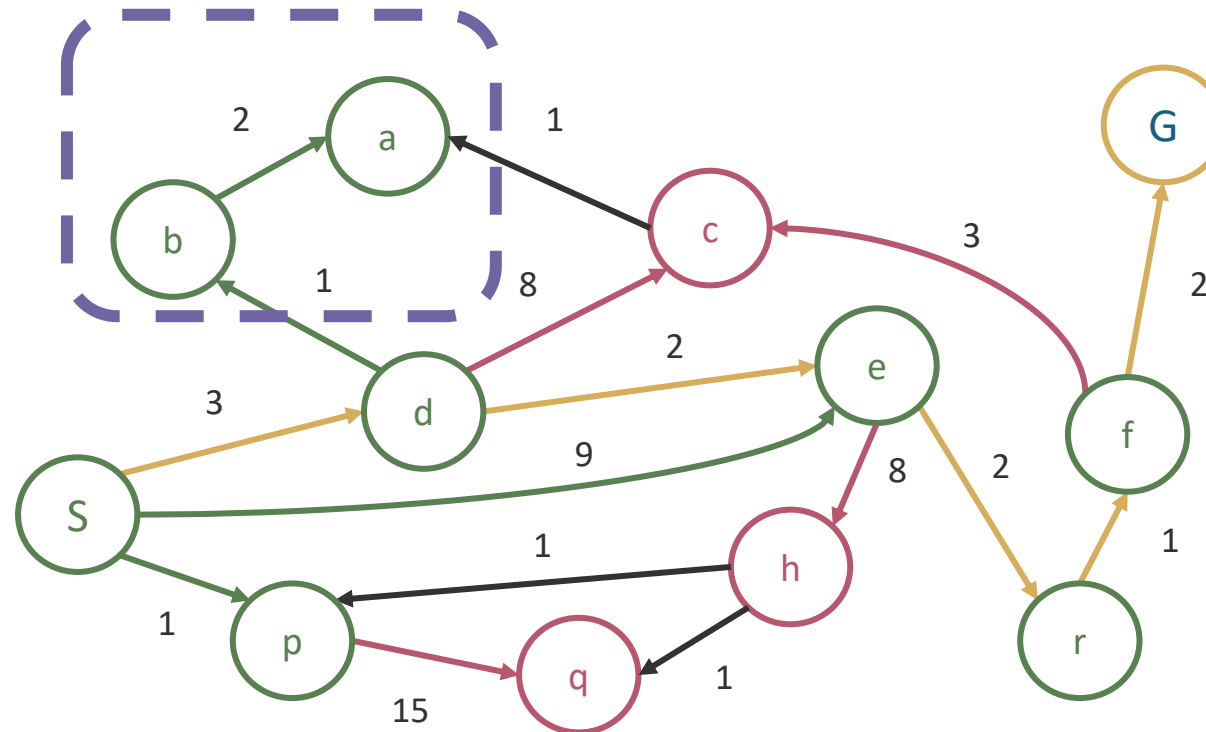
ALGORITHM

```
function TREE-SEARCH(problem, strategy) returns solution or failure
  initialize fringe using initial state of problem
  loop do
    if fringe empty then return failure
    node ← remove node from fringe according to strategy
    if node is goal state then return corresponding solution
    expand node and add resulting nodes to fringe
  end
```

What's Next?

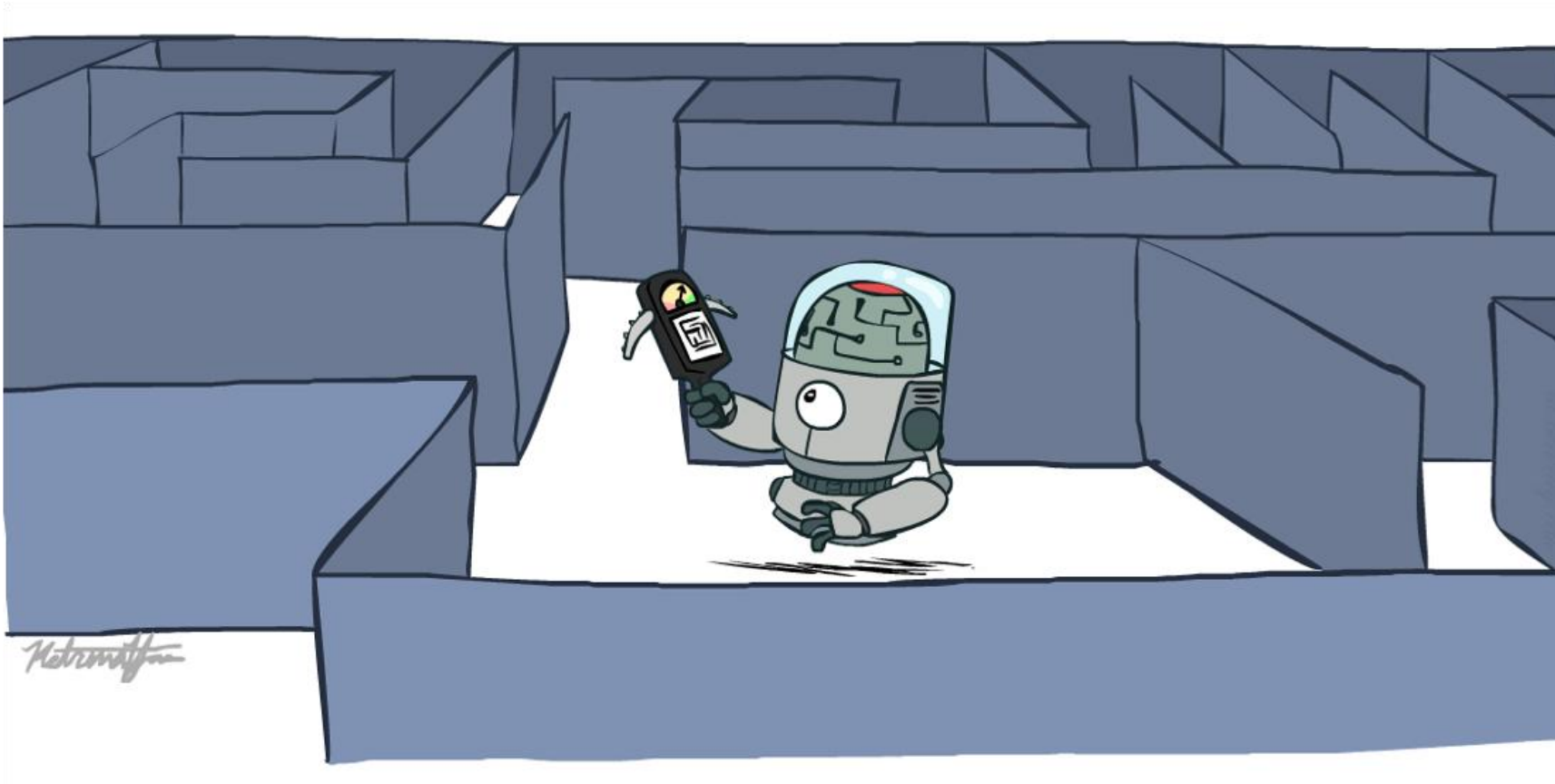
CONTEXT

dead end
can we avoid it?



Can We *Inform* Our Search?

CONTEXT



that's it for today!

SUMMARY

- *Which fringe nodes to expand?*
- DFS (stack) vs. BFS (queue)
vs. UCS (priority queue)

UPCOMING

- Search with Heuristics
- Multi-Agent Search

REMINDERS

- Complete **Practice Problem 3**
- Do **Homework 1** (due 7.2)
- Do **Project 1** (due 7.9)