TRƯỜNG ĐẠI HỌC
SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
HCMC University of Technology and Education
HCMUTE
Faculty of Information Technology

Search for Solutions

(ARTIFICIAL INTELLIGENCE)

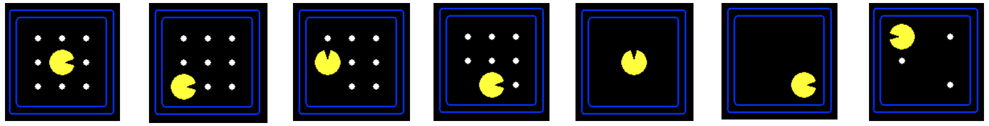
Instructor: Assoc. Prof. PhD. Hoàng Văn Dũng

Email: dunghv@hcmute.edu.vn

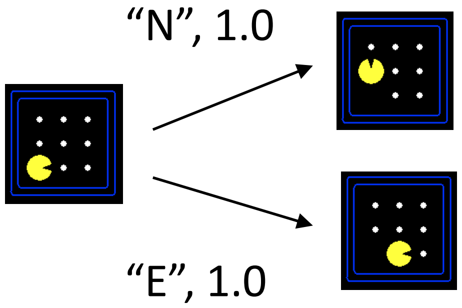
(slides adapted from Dan Klein, Pieter Abbeel, Anca Dragan, et al)

Search Problems

- In AI, search techniques are universal problem-solving methods



- 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

Search Problems

- Terminologies:
  - –Search: Searching is a step by step procedure to solve problem.
  - –3 main factors: Search Space; Start State; Goal test
  - –Search tree: represent a search problem.
  - –Actions.
  - –Transition model.
  - –Path Cost.
  - –Solution.
  - –Optimal Solution.

Search Problems

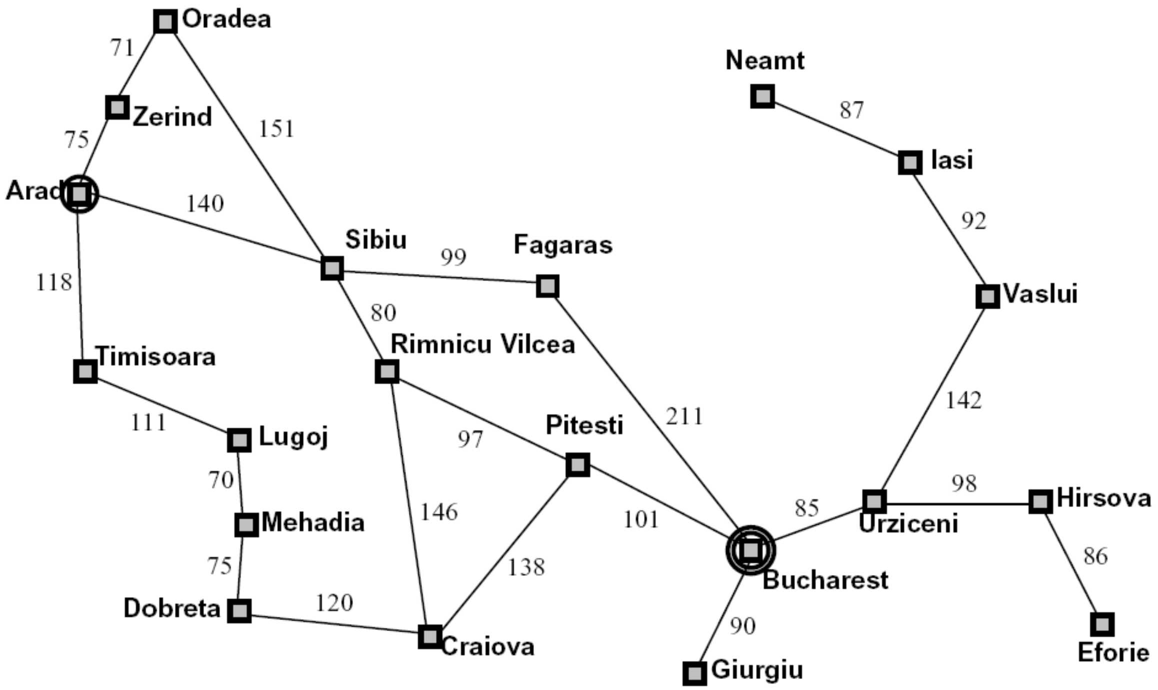
Components of search problem

- State space:
  - Cities
- Successor function:
  - Action: Roads
  - Cost

  Go to adjacent city
  with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?

- Solution?

Example: Traveling in Romania

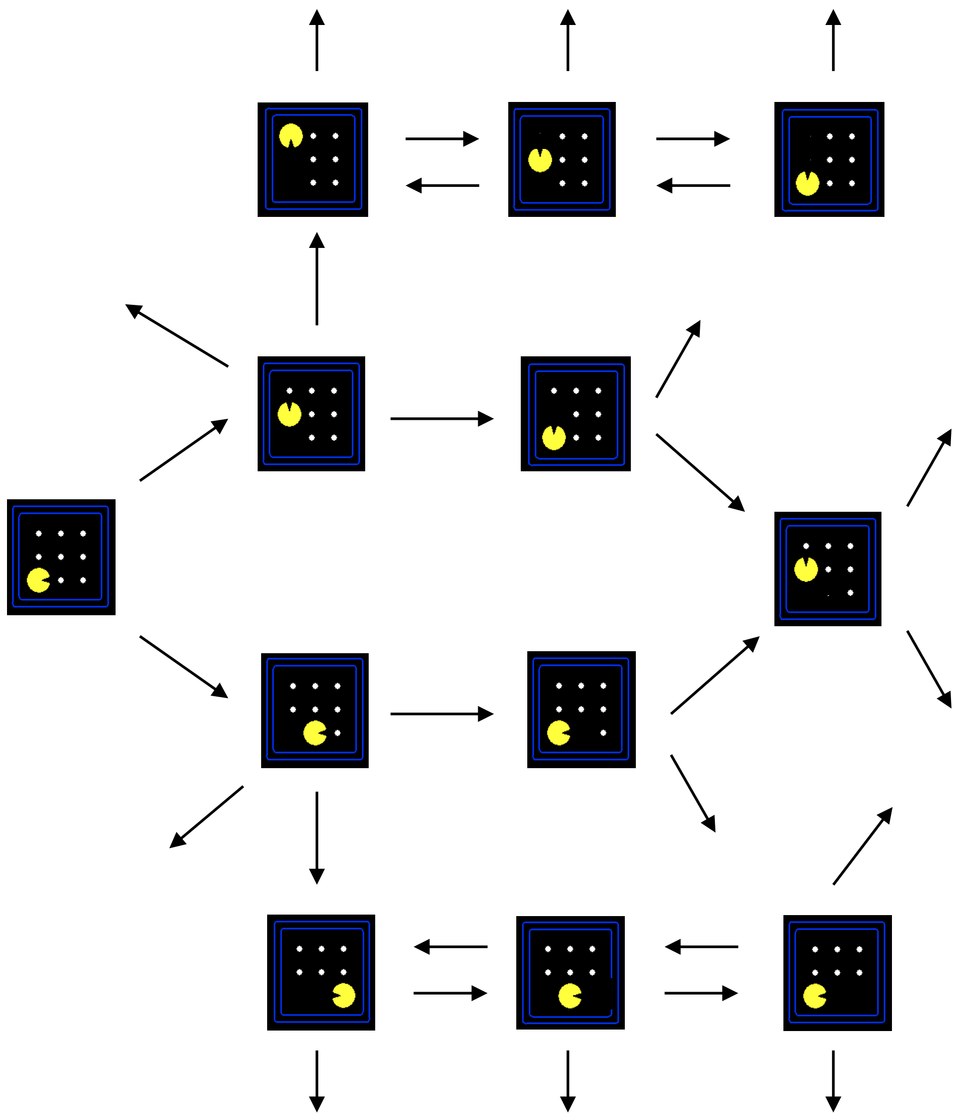


State Space Graphs

- 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 nodes (maybe only one)

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

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

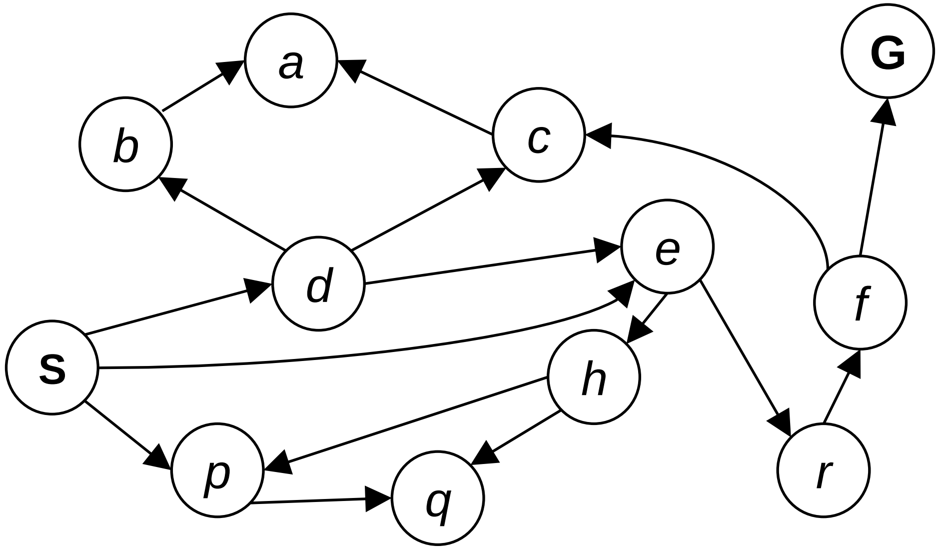


State Space Graphs

- 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 nodes (maybe only one)

- In a state space graph, each state occurs 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

- **Example:**

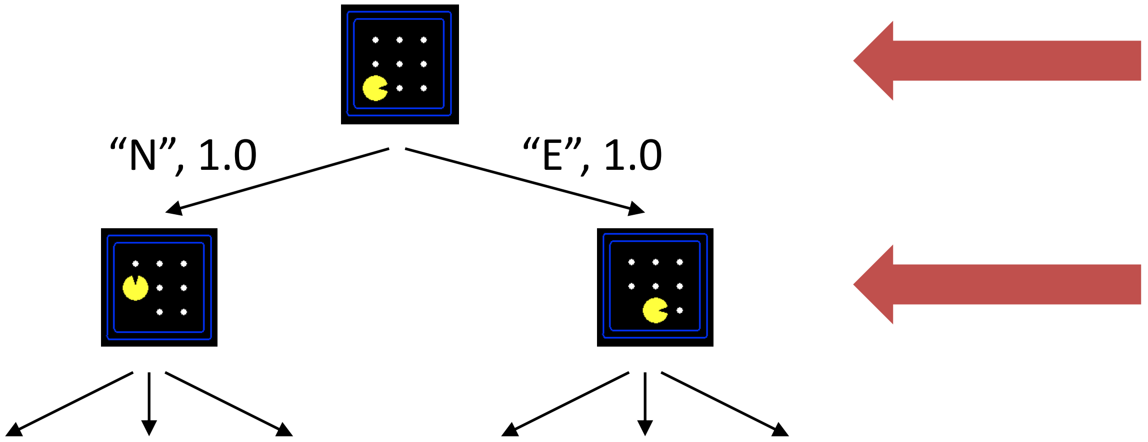
Searching for a solution to the 8-puzzle.

2	8	3
1	6	4
7		5

Start state

1	2	3
8		4
7	6	5

Goal



This is now / start

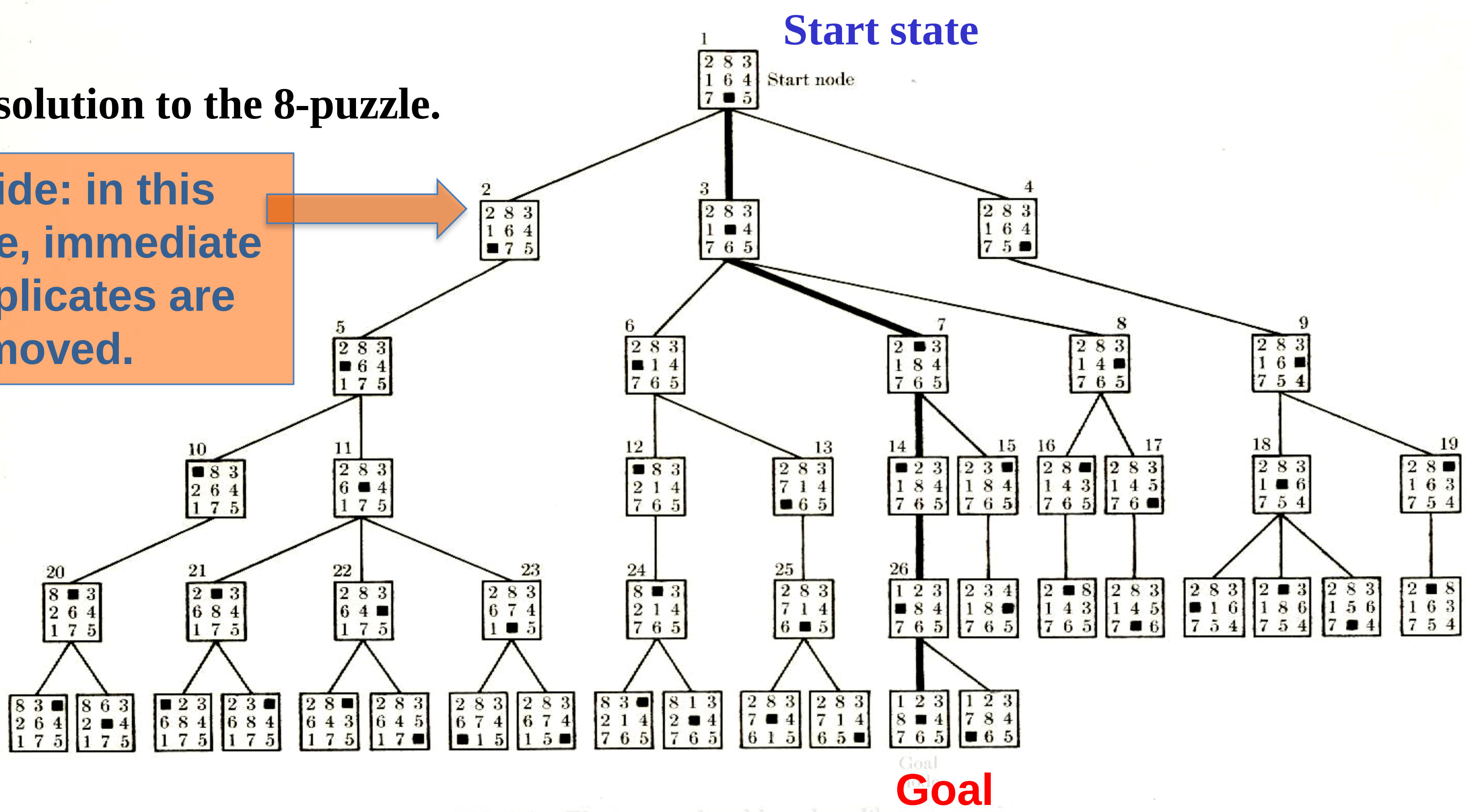
Possible futures

- 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

Search Trees

Example:
Searching for a solution to the 8-puzzle.

Aside: in this tree, immediate duplicates are removed.



A breadth-first search tree. (More detail soon.)

Branching factor 1, 2, or 3 (max). So, approx. 2 --- # nodes roughly doubles at each level. *Number states of explored nodes grows exponentially with depth.*

Tree Search

- General Tree Search

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

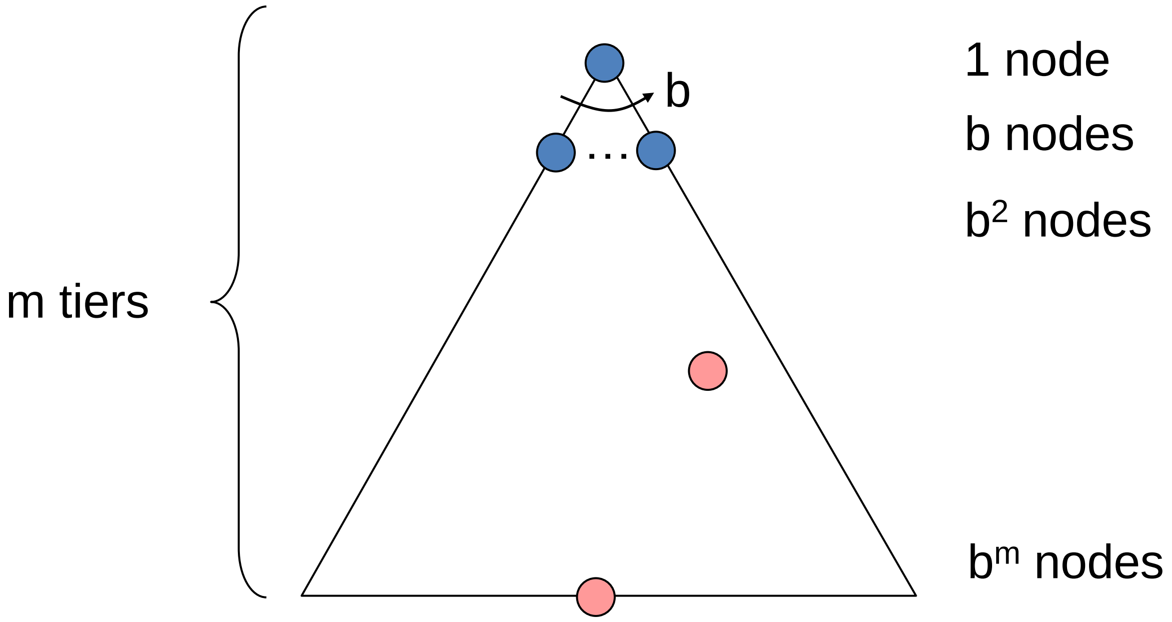
- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?

Search Algorithm Properties

Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

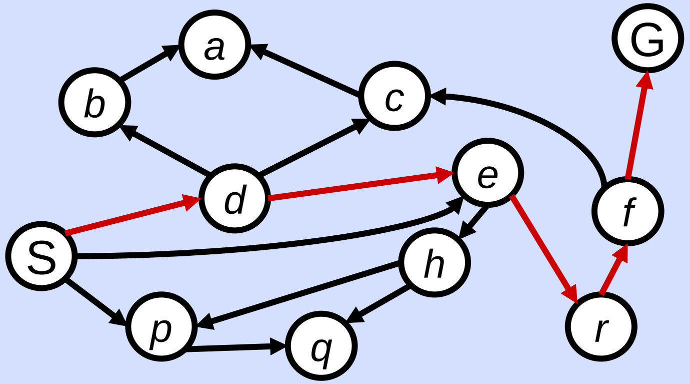
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



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

State Space Graphs vs. Search Trees

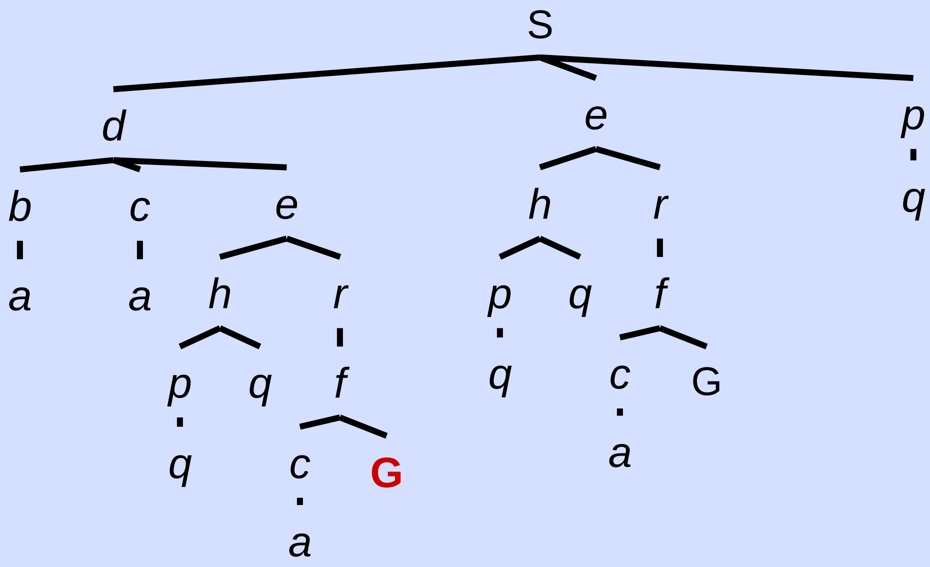
State Space Graph



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

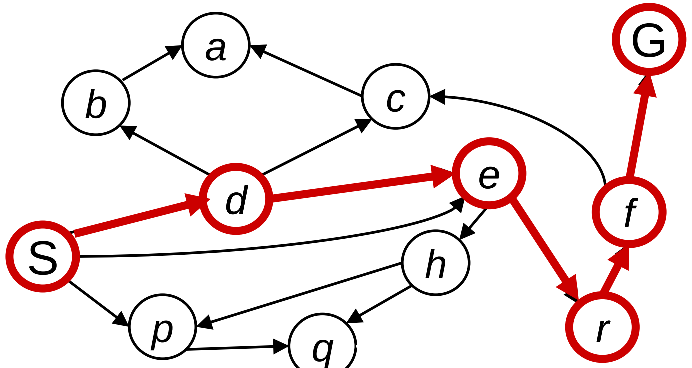
We construct both on demand – and we construct as little as possible.

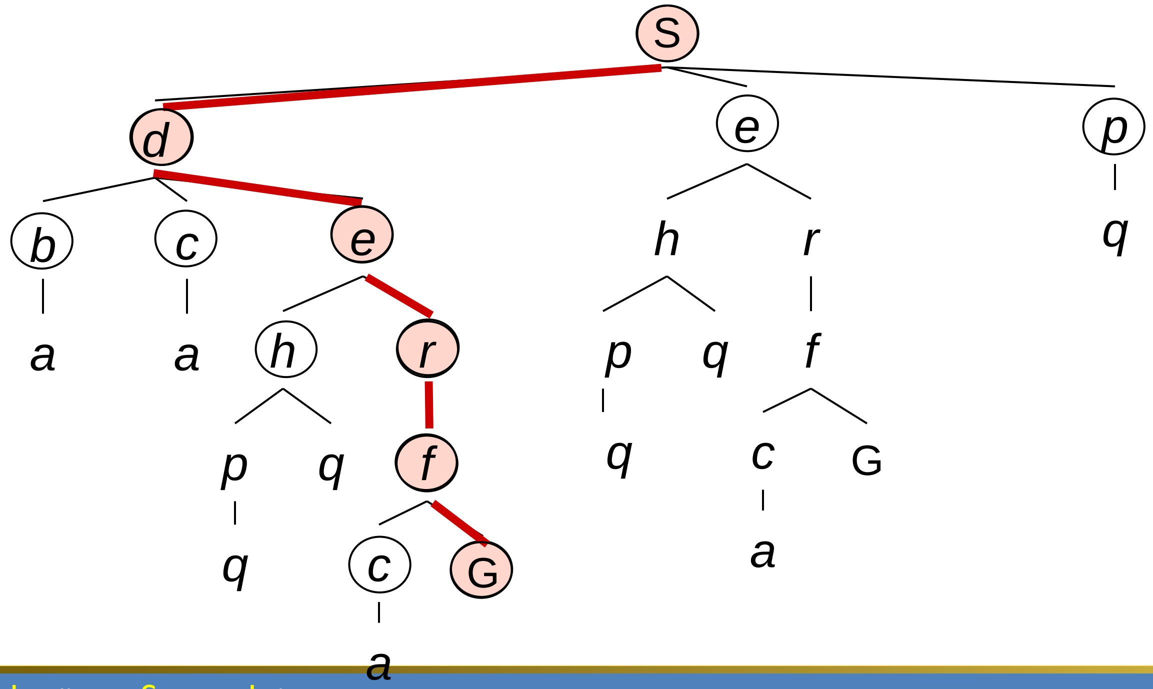
Search Tree



State Space Graphs vs. Search Trees

- 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~~

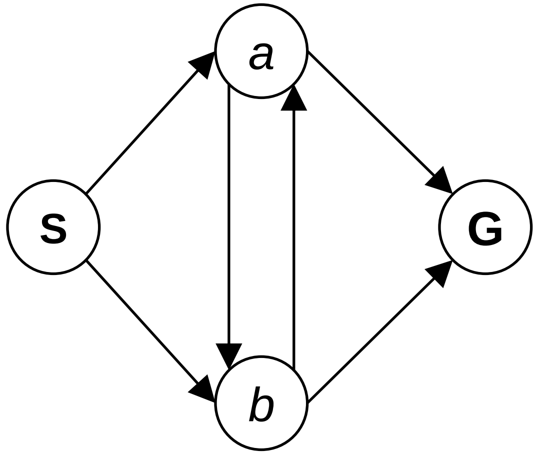
s → d → e → r → f → c

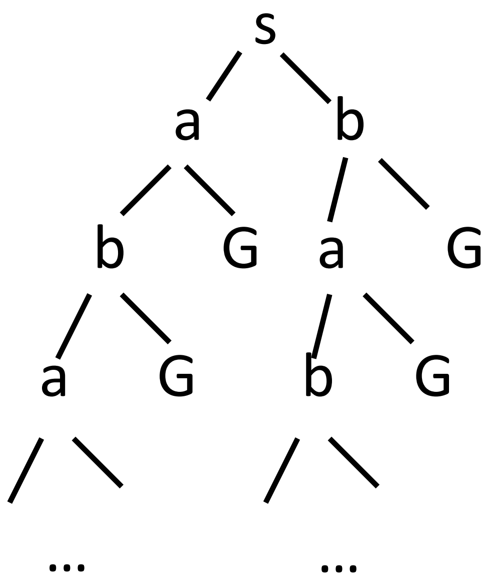
~~s → d → e → r → f → G~~

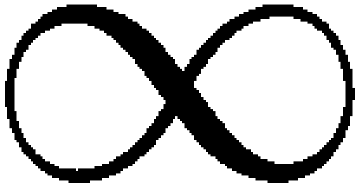
Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

How big is its search tree (from S)?





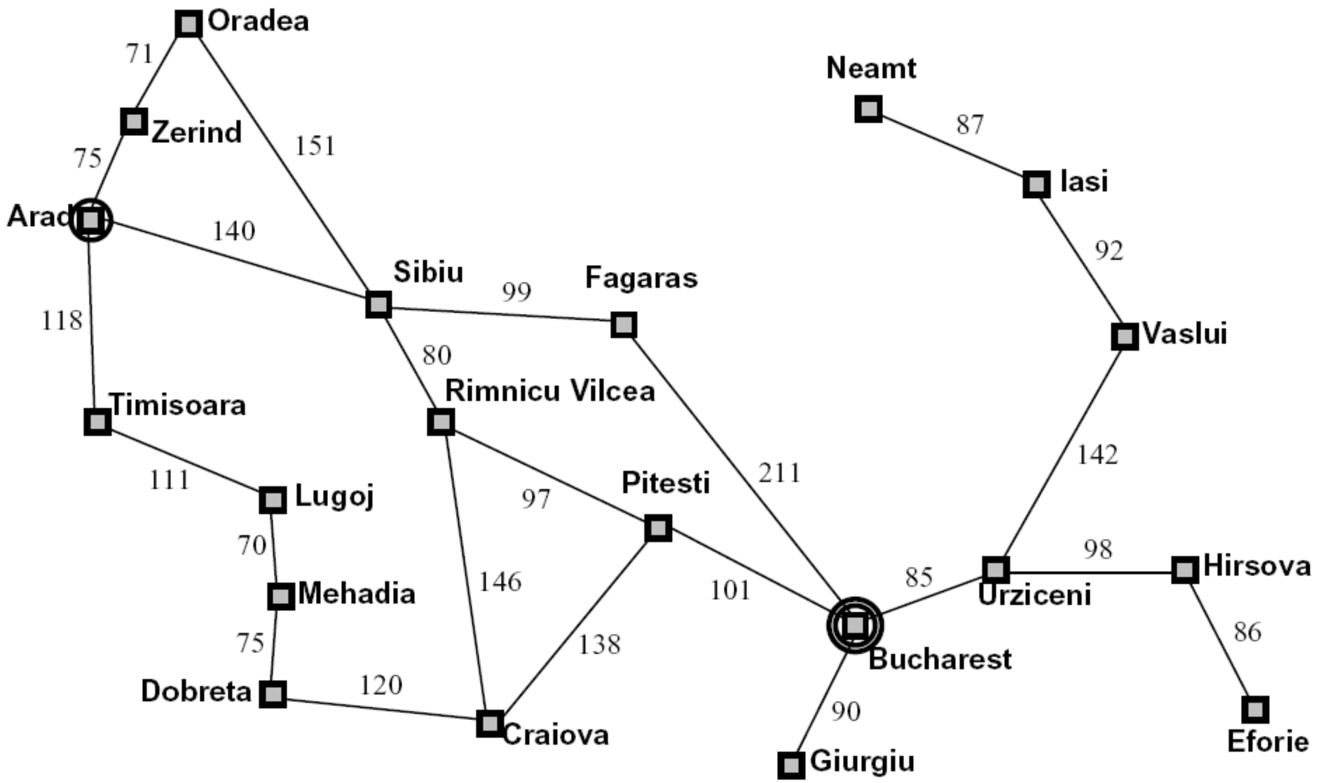


Có bao nhiêu node trên cây tìm kiếm tương ứng với đồ thị trạng thái sau?

Important: Lots of repeated structure in the search tree!

Assignment

- Path planning using Roma city map
  - ➢ Starting state: **Arad**
  - ➢ Goal state: **Bucharest**



- Building a searching tree (to search a solution)

trạng thái xuất phát

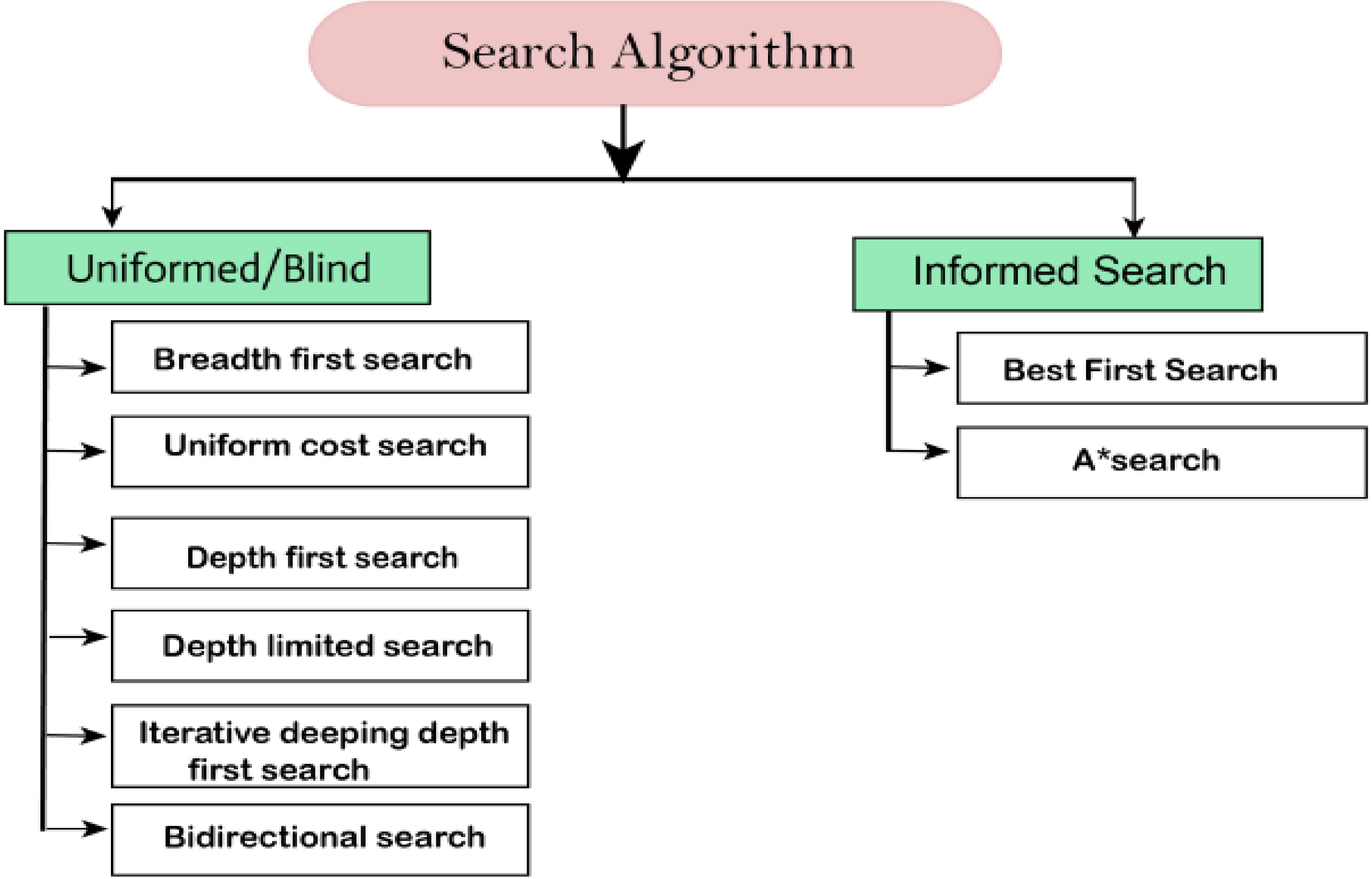
2	6	5
	8	7
4	3	1

trạng thái đích

1	2	3
4	5	6
7	8	

Search Algorithm

- Search approaches



Uninformed search algorithms

Breadth-first search

Depth-first search

Iterative Deepening

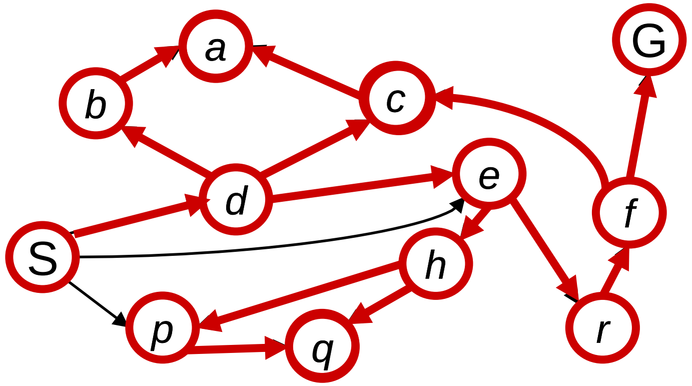
Uniform Cost Search

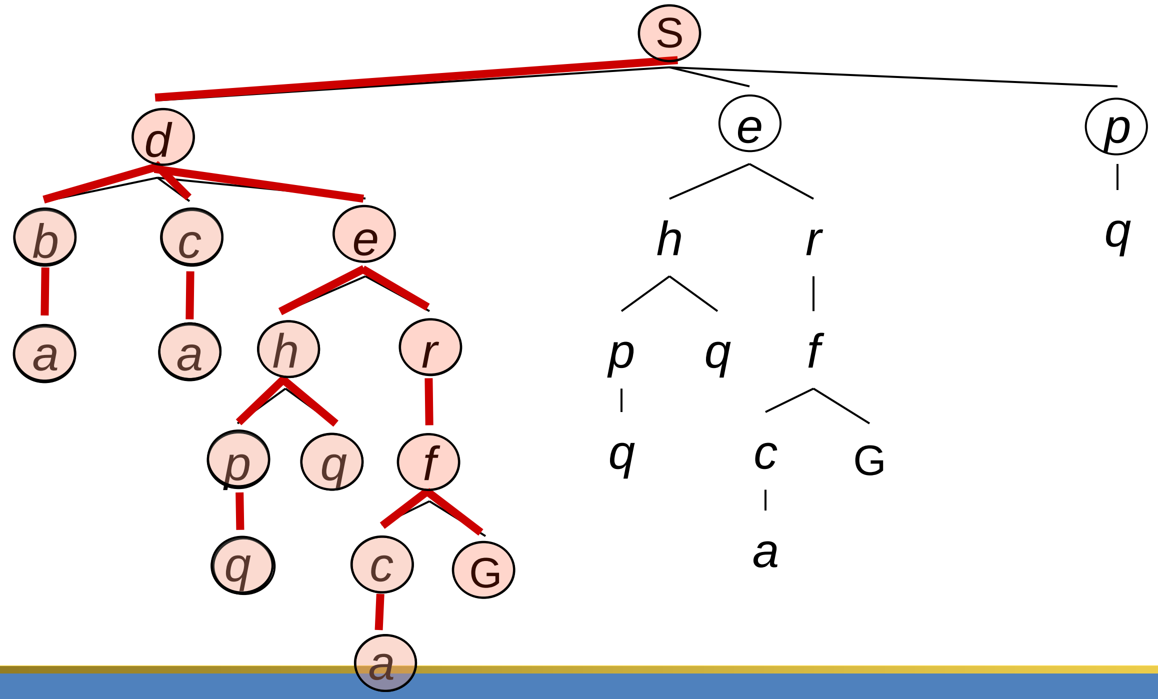
Depth-First Search

Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack

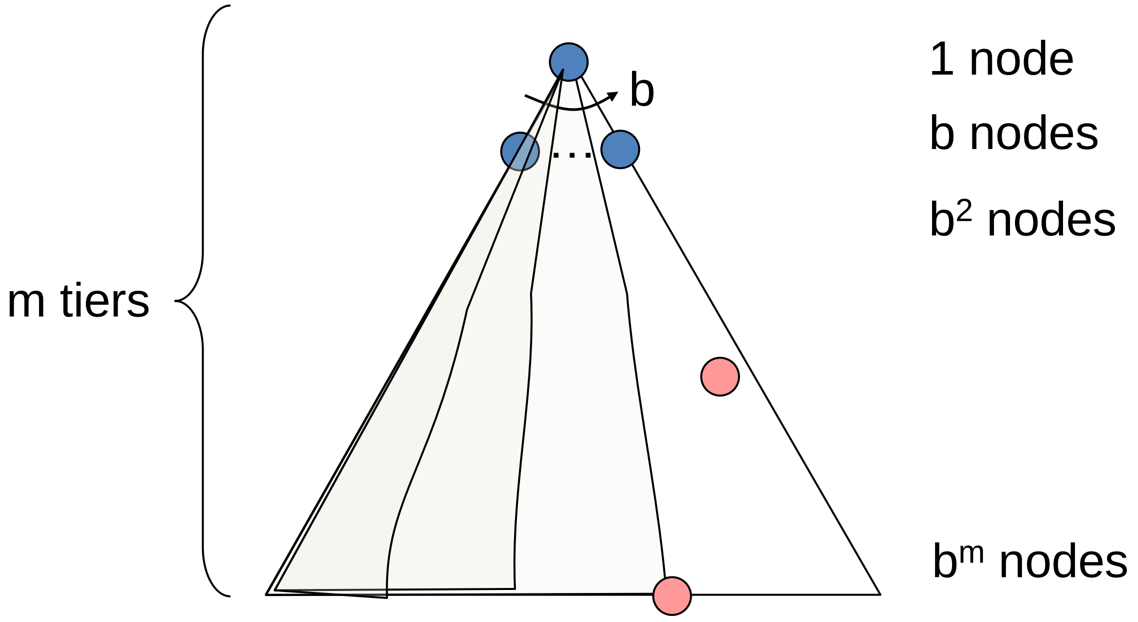




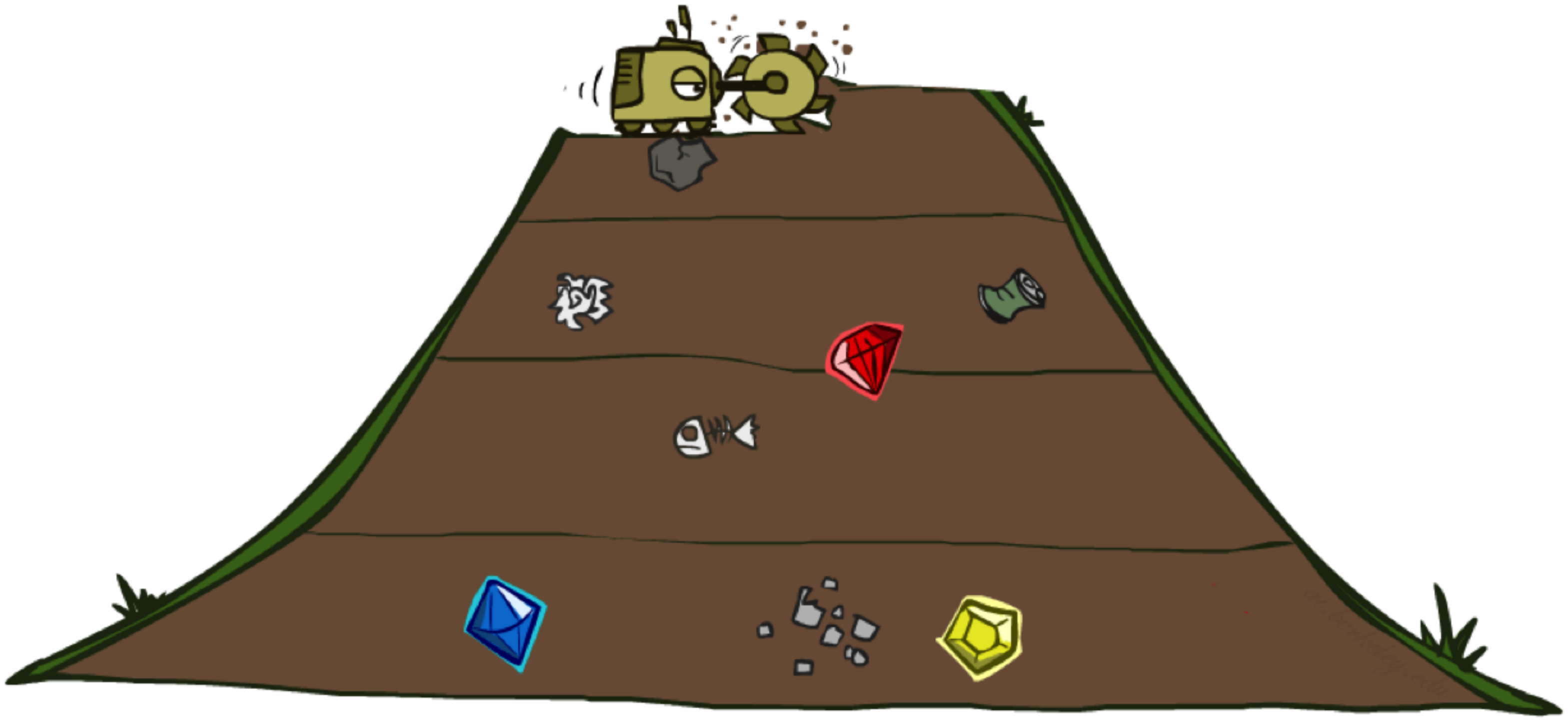
Depth-First Search

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 fringe take?
  - 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



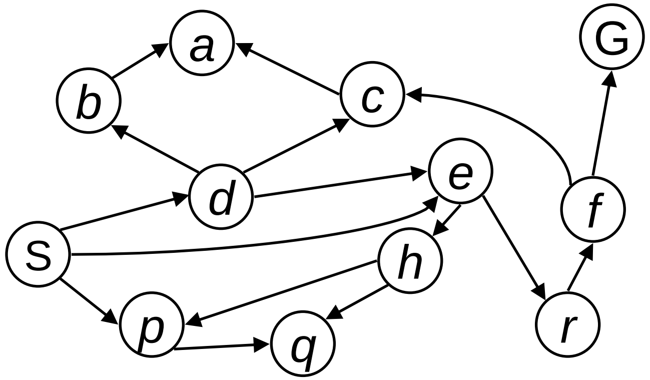
Breadth-First Search



Breadth-First Search

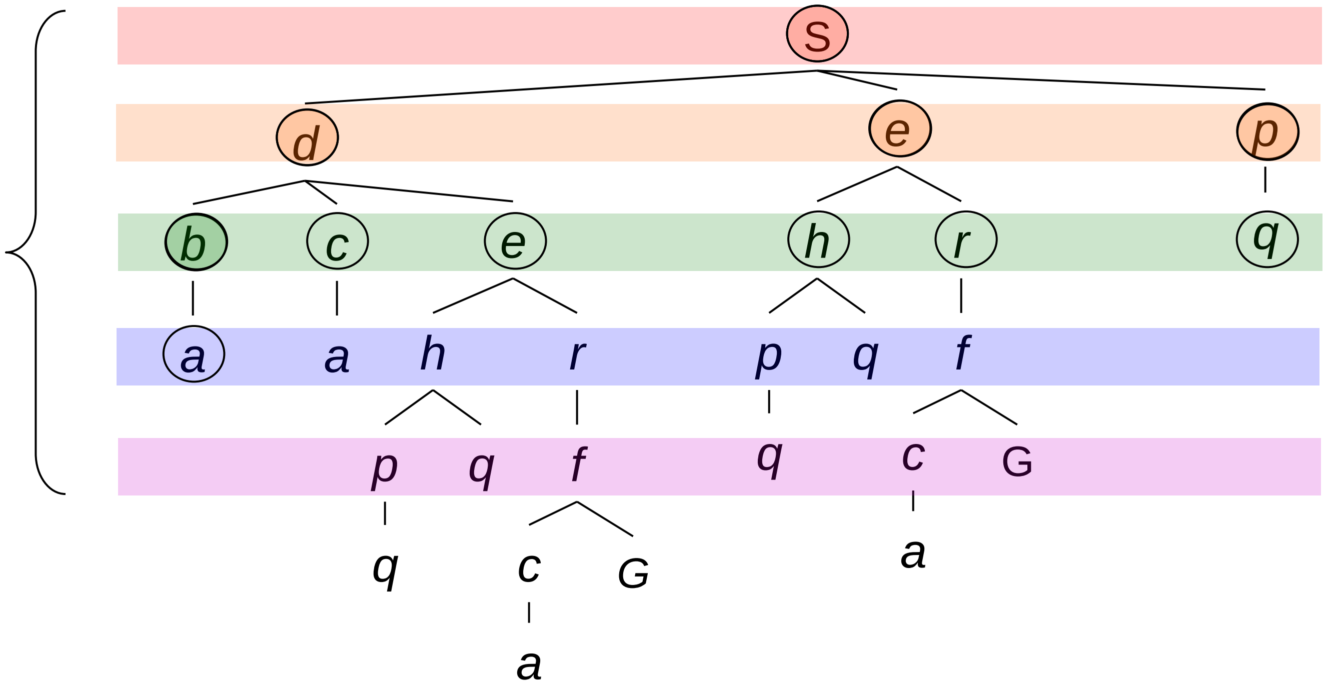
Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue



Search

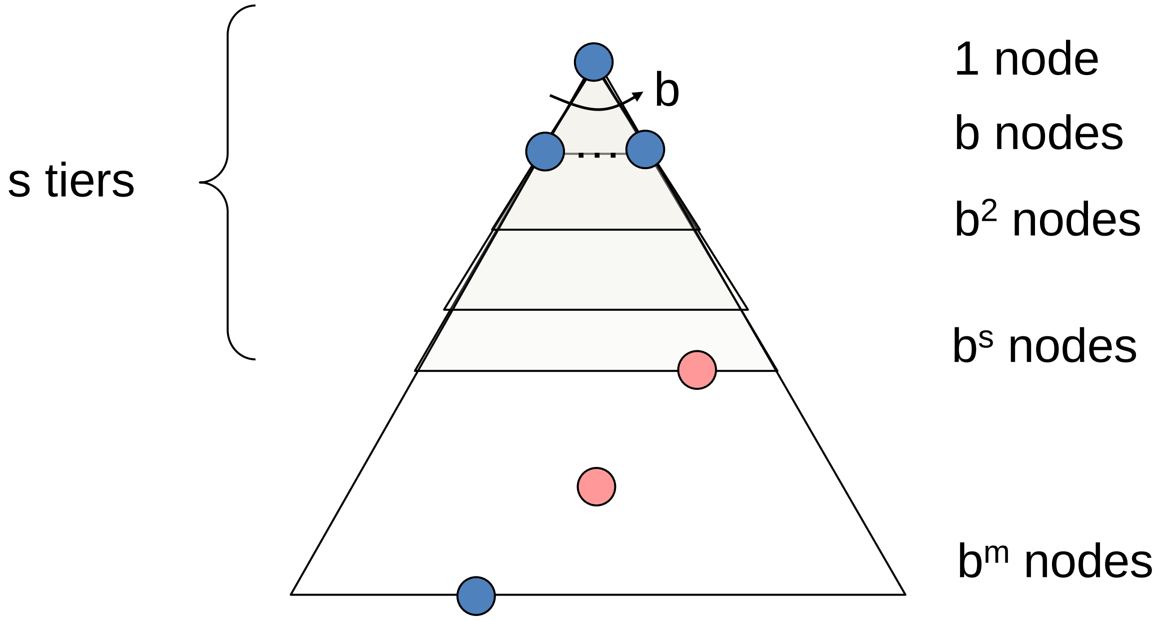
Tiers



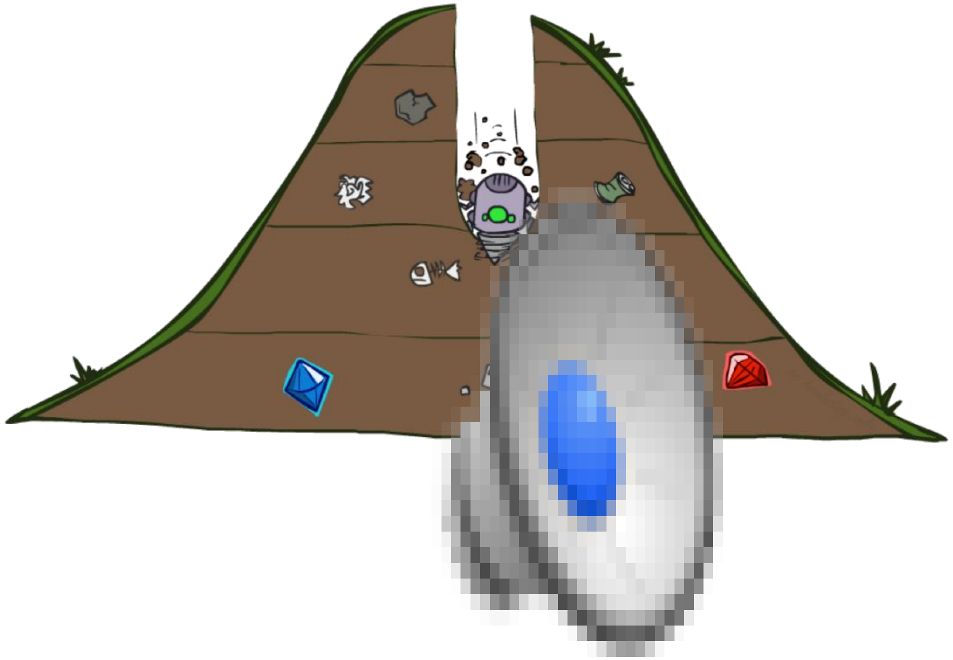
Breadth-First Search

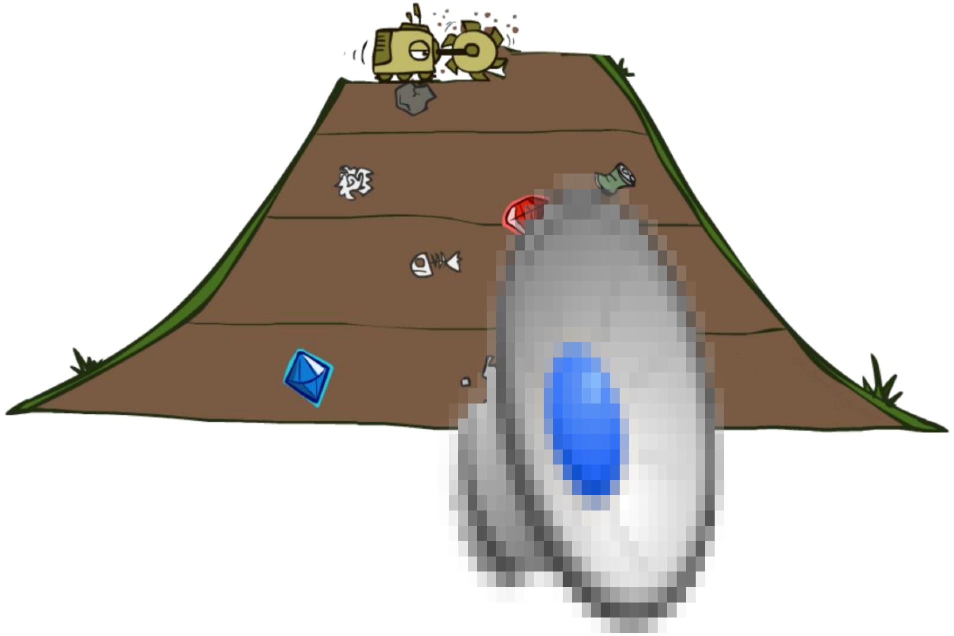
BFS Properties

- 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 fringe take?
  - Has roughly the last tier, so $O(b^s)$
- 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



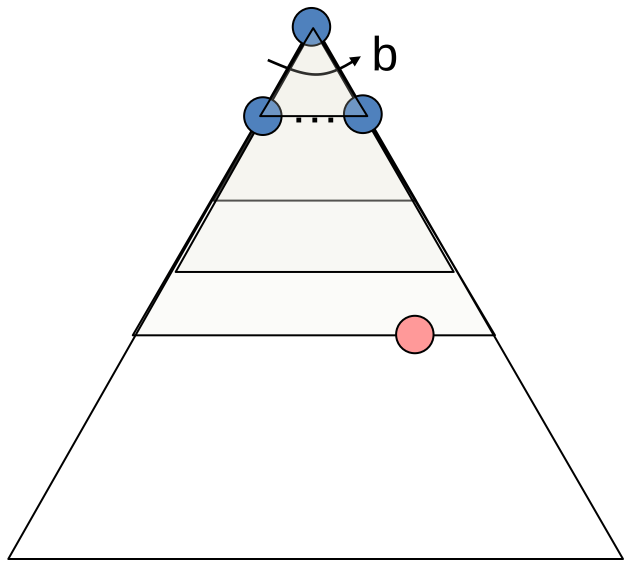


Video of
Demo Maze
Water
DFS/BFS

- When will BFS outperform DFS?
- When will DFS outperform BFS?

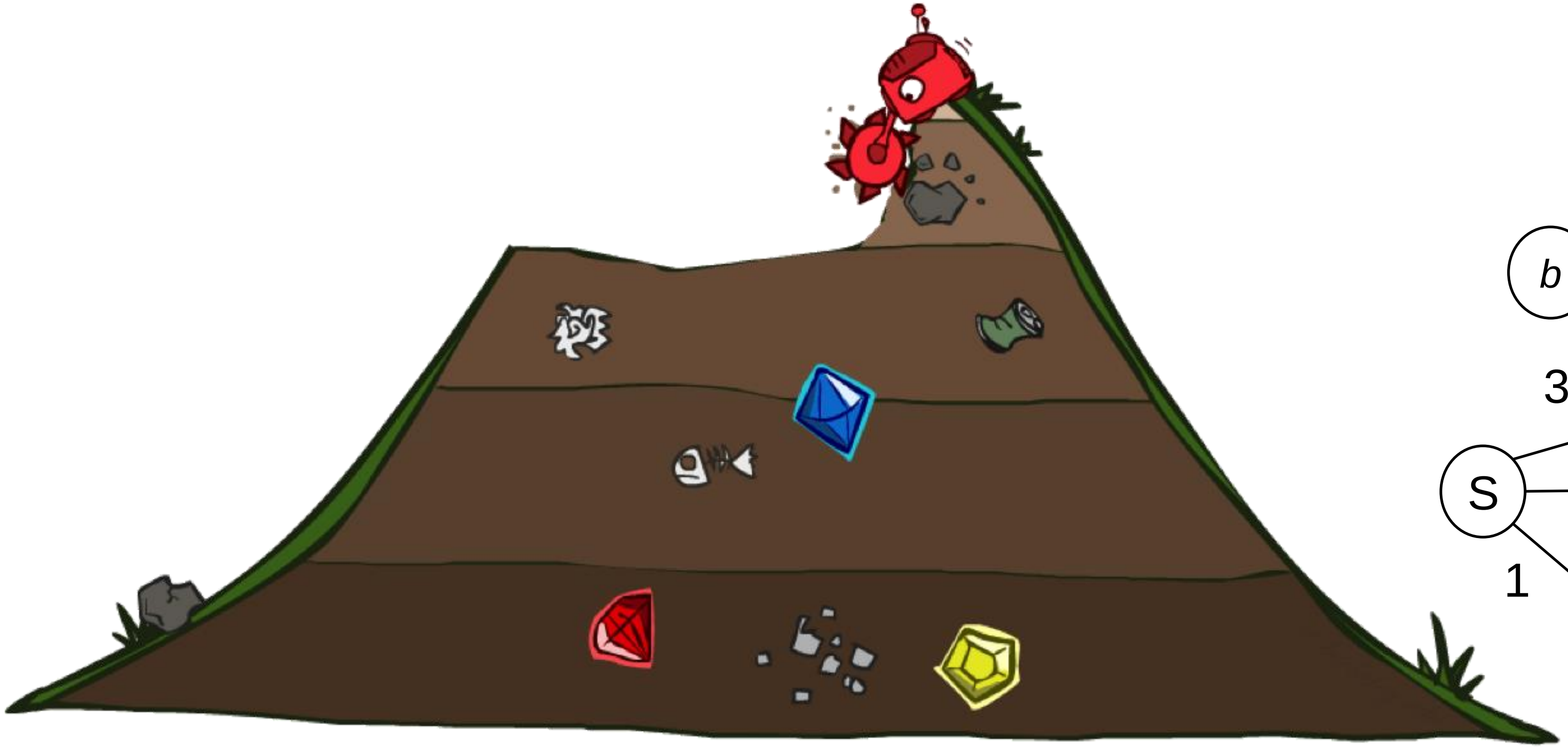
Iterative Deepening

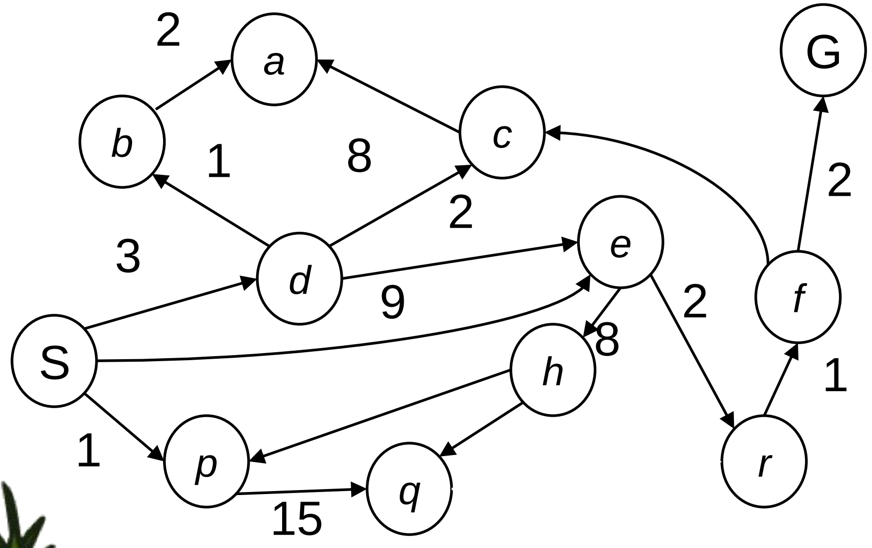
- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..



- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!

Uniform Cost Search

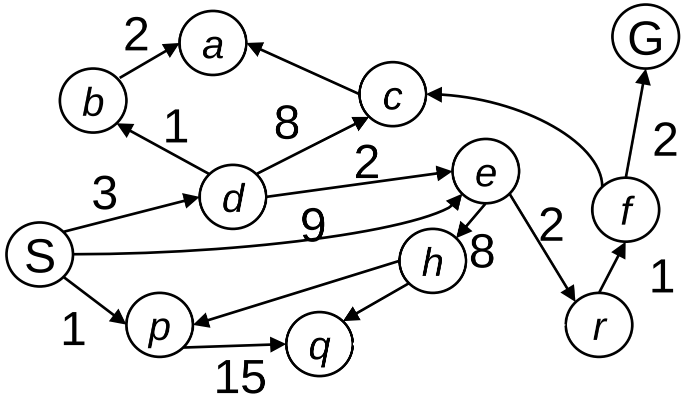


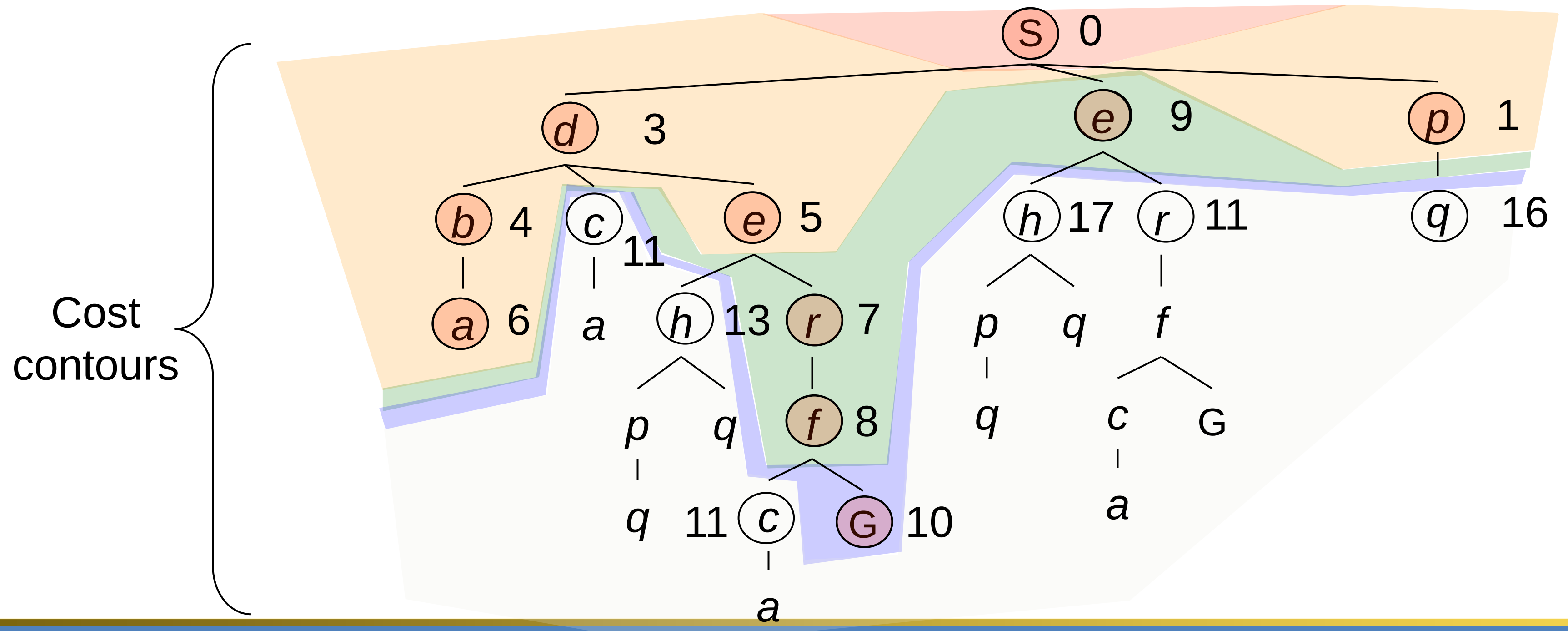


Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)

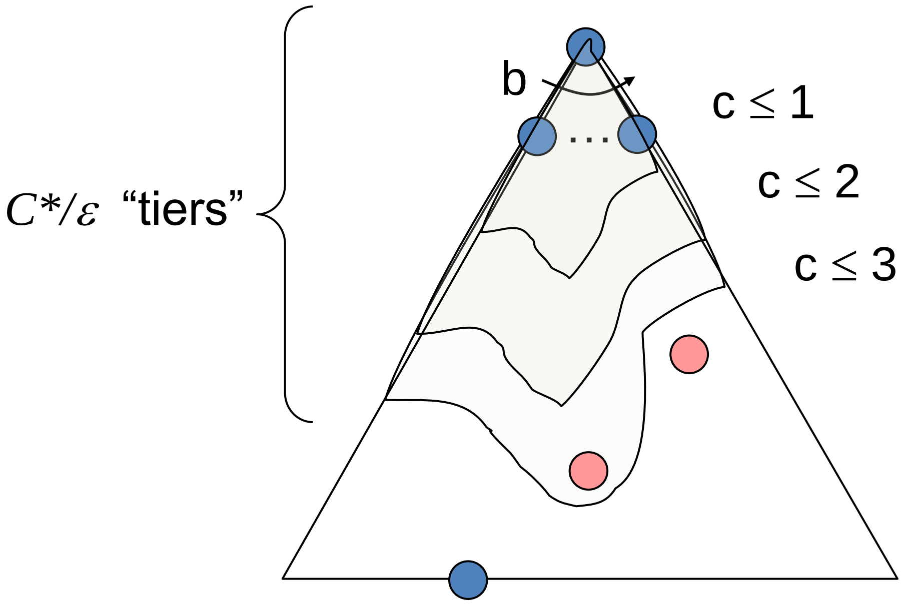




Uniform Cost Search

UCS Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs 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 arc cost is positive, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)

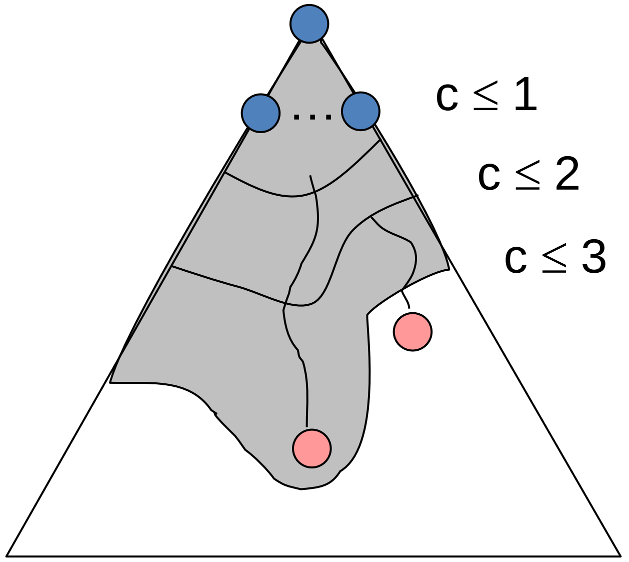


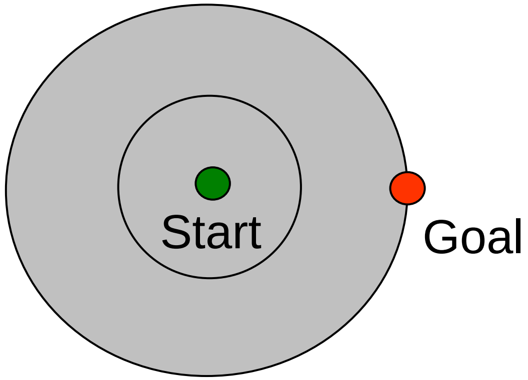
Uniform Cost Search

- Strategy: expand lowest path cost

- good: UCS is complete and optimal!

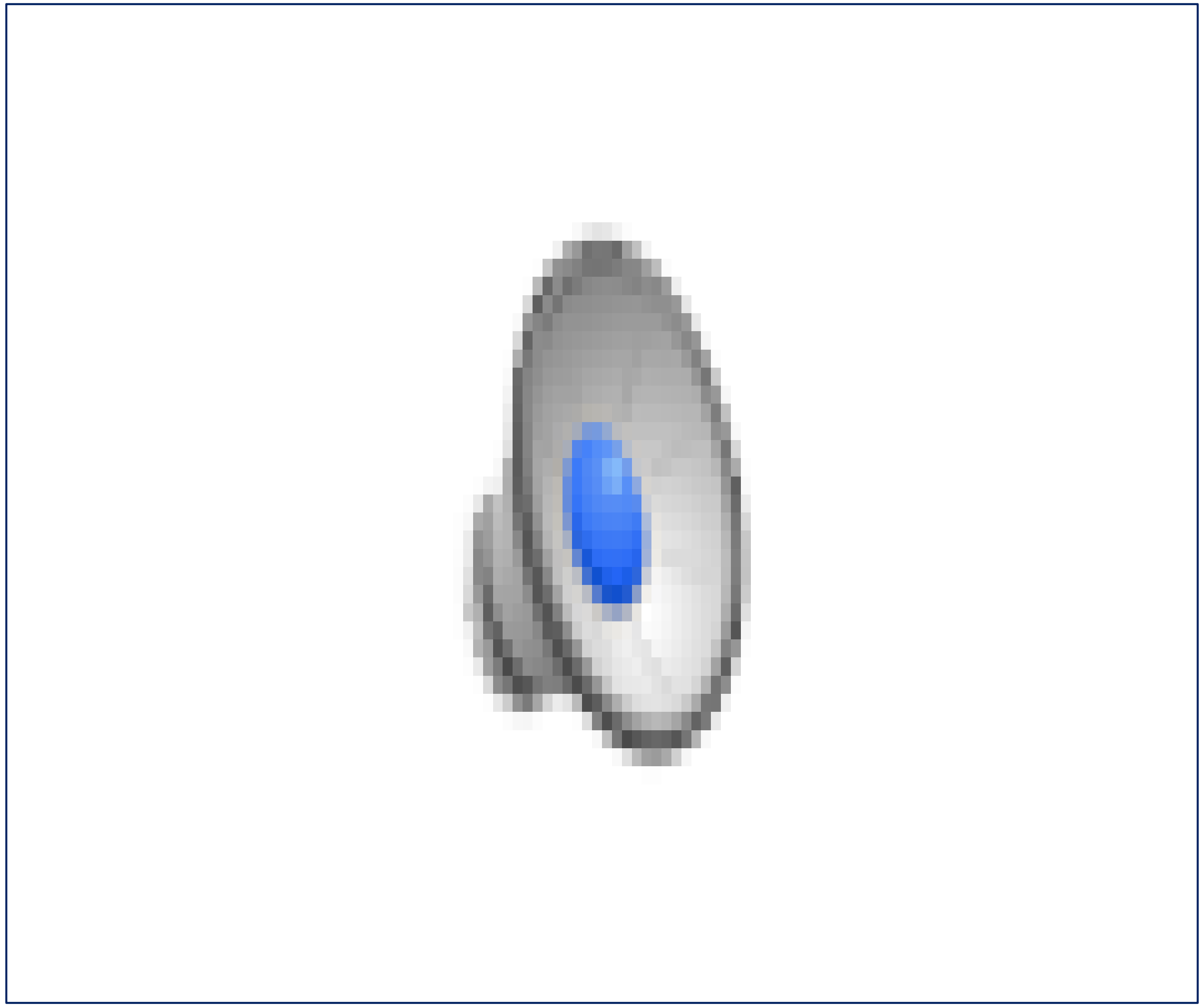
- The bad:
  - Explores options in every "direction"
  - No information about goal location





Uniform Cost Search

- Video of Demo Empty UCS



Video of Demo Empty UCS

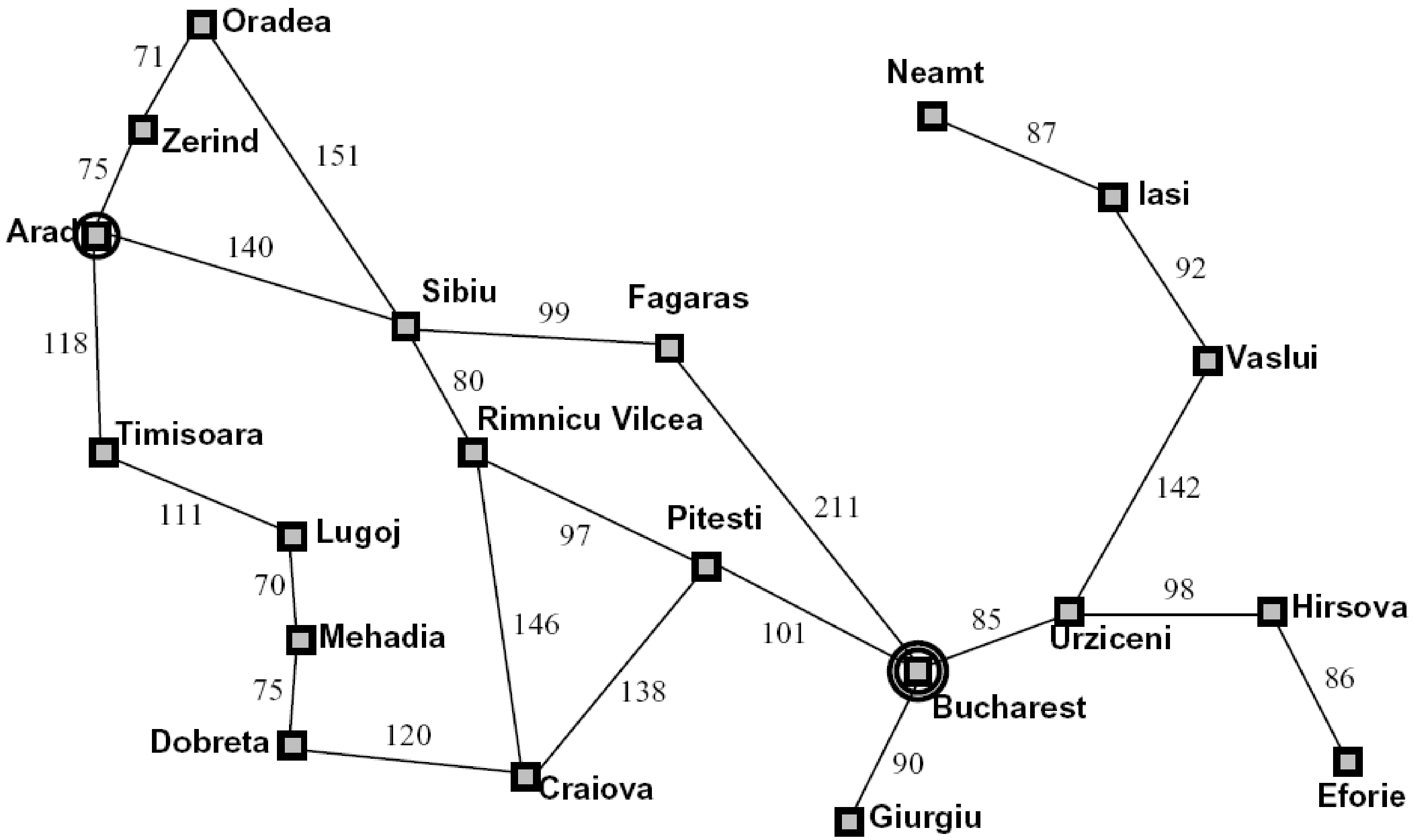
Demo Maze with Deep/
Shallow Water:
DFS, BFS or UCS?

Assignment

- Thể hiện quá trình tìm đường đi bằng phương pháp DFS, BFS, ID sâu dần và UCS, với

➢ Trạng thái đầu: **Arad**

➢ Trạng thái đích: **Bucharest**



(*) Viết chương trình python

Thanks for your attention!

Q&A