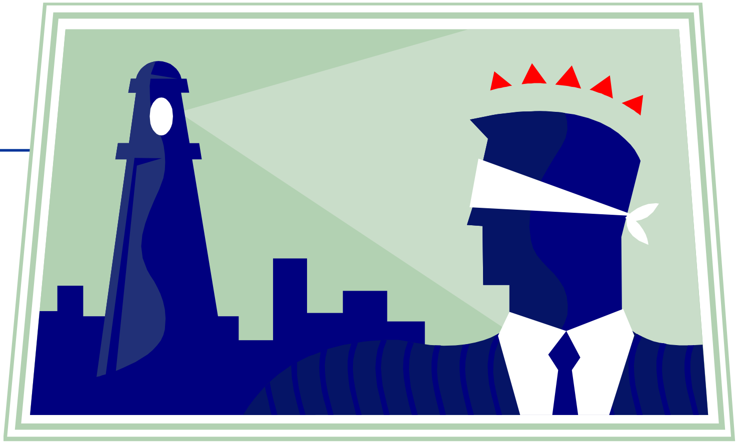
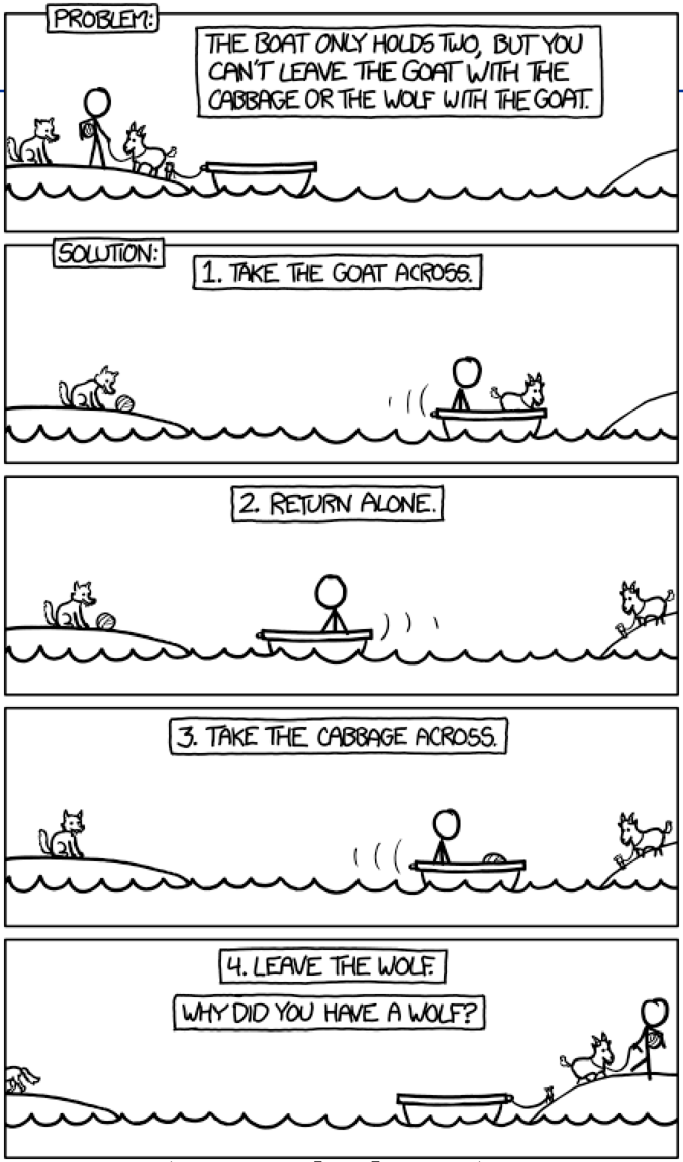


Uninformed Search Strategies

AIMA 3.4

The Goat, Cabbage, Wolf Problem



(From xkcd.com)

But First: Missionaries & Cannibals

Three missionaries and three cannibals come to a river. A rowboat that seats two is available. If the cannibals ever outnumber the missionaries on either bank of the river, the missionaries will be eaten. ***(problem 3.9)***

How shall they cross the river?

Formulation: Missionaries & Cannibals

- ***States:*** **(CL, ML, BL)**
- ***Initial state*****: (331)**
- ***Goal test*****: True if all M, C, and boat on other bank (000)**
- ***Actions: Travel Across Travel Back***

Travel Across	Travel Back
-101	101
-201	201
-011	011
-021	021
-111	111

Outline for today's lecture

- **Introduction to Uninformed Search**
  - (Review of Breadth first and Depth-first search
- **Iterative deepening search**
  - Strange Subroutine: Depth-limited search
  - Depth-limited search + iteration = WIN!!
- **Briefly: Bidirectional search**
- **If time: Uniform Cost Search**

Uninformed search strategies:

- AKA “Blind search”
- Uses only information available in problem definition

Informally:

- *Uninformed search:* All non-goal nodes in frontier look equally good
- *Informed search*: Some non-goal nodes can be ranked above others.

Search Strategies

- ***Review: Strategy* = order of tree expansion**
  - Implemented by different queue structures (LIFO, FIFO, priority)

- **Dimensions for evaluation**
  - *Completeness-* always find the solution?
  - *Optimality* - finds a least cost solution (lowest path cost) first?
  - *Time complexity* - # of nodes generated *(worst case)*
  - ***Space complexity* - # of nodes in memory** *(worst case)*

- **Time/space complexity variables**
  - *b, maximum branching factor* of search tree
  - *d, depth* of the shallowest goal node
  - *m*, maximum length of any path in the state space (potentially ∞)

Introduction to *space* complexity

- **You know about:**
  - “Big O” notation
  - *Time complexity*

- ***Space complexity* is analogous to time complexity**

- **Units of space are arbitrary**
  - Doesn’t matter because Big O notation ignores constant multiplicative factors
  - Space units:
    - —One Memory word
    - —Size of any fixed size data structure
      - – eg Size of fixed size node in search tree

Review: Breadth-first search

- **Idea:**
  - Expand *shallowest* unexpanded node

- **Implementation:**
  - *frontier* is FIFO (First-In-First-Out) Queue:
    - Put successors at the *end* of *frontier* successor list.

Breadth-first search (simplified)

function BREADTH-FIRST-SEARCH*(problem)* **returns** a solution, or failure
    node <- a node with STATE = *problem*.INITIAL-STATE, *PATH-COST*=0
    if problem.GOAL-TEST(node.STATE) **then return** SOLU*TION(node)*
    frontier <- a FIFO queue with node as the only element
    loop do
        if EMPTY?(*frontier)* **then return** failure
        node <- POP*(frontier)* // chooses the shallowest node in frontier
        add *node*.STATE to explored
        for each *action* **in** *problem*.ACTIONS*(node*.STATE*)* **do**
            child <- CHILD-NODE*(problem, node, action)*
            if *problem*.GOAL-TEST*(child*.STATE*)* **then return** SOLUTION*(child)*
            frontier <- INSERT*(child, frontier)*

Position within queue of new items determines search strategy

Subtle: *Node inserted into queue only after testing to see if it is a goal state*

From Figure 3.11 Breadth-first search (ignores loops, repeated nodes)

Properties of breadth-first search

- **Complete?** **Yes (if *b* is finite)**
- **Time Complexity?** $1+b+b^2+b^3+\ldots+b^d = O(b^d)$
- **Space Complexity?** $O(b^d)$ **(keeps every node in memory)**
- **Optimal?** **Yes, if cost = 1 per step**
  (not optimal in general)

b: maximum branching factor of search tree

d: depth of the least cost solution

m: maximum depth of the state space (∞)

Exponential Space (and time) Not Good...

- Exponential complexity uninformed search problems *cannot* be solved for any but the smallest instances.
- *(Memory* requirements are a bigger problem than *execution* time.)

DEPTH	NODES	TIME	MEMORY
2	**110**	**0.11 milliseconds**	**107 kilobytes**
4	**11110**	**11 milliseconds**	**10.6 megabytes**
6	**10^6**	**1.1 seconds**	**1 gigabytes**
8	**10^8**	**2 minutes**	**103 gigabytes**
10	**10^{10}**	**3 hours**	**10 terabytes**
12	**10^{12}**	**13 days**	**1 petabytes**
14	**10^{14}**	**3.5 years**	**99 petabytles**

Fig 3.13 Assumes b=10, 1M nodes/sec, 1000 bytes/node

Review: Depth-first search

- **Idea:**
  - Expand *deepest* unexpanded node

- **<u>Implementation:</u>**
  - *frontier* is LIFO (Last-In-First-Out) Queue:
    - —Put successors at the *front* of *frontier* successor list.

Properties of depth-first search

- **Complete?** **No: fails in infinite-depth spaces, spaces with loops**
  - Modify to avoid repeated states along path
    → complete in finite spaces
- **Time?** **$O(b^m)$: terrible if *m* is much larger than *d***
  - but if solutions are dense, may be much faster than breadth-first
- **Space?** **$O(b*m)$, i.e., linear space!**
- **Optimal?** **No**

b: maximum branching factor of search tree

d: depth of the least cost solution

m: maximum depth of the state space (∞)

Depth-first vs Breadth-first

- **Use depth-first if**
  - *Space is restricted*
  - There are many possible solutions with long paths and wrong paths are usually terminated quickly
  - Search can be fine-tuned quickly

- **Use breadth-first if**
  - *Possible infinite paths*
  - Some solutions have short paths
  - Can quickly discard unlikely paths

Iterative Deepening Search

Search Conundrum

- **Breadth-first**
  - ☑ Complete,
  - ☑ Optimal
  - ☒ *but* uses $O(b^d)$ space
- **Depth-first**
  - ☒ Not complete *unless m is bounded*
  - ☒ Not optimal
  - ☒ Uses $O(b^m)$ time; terrible if m >> d
  - ☑ *but* only uses *O(**b∗m) space***

How can we get the best of both?

Depth-limited search: A building block

- **Depth-First search** ***but with depth limit ℓ.***
  - i.e. nodes at depth ℓ *have no successors.*
  - No infinite-path problem!

- **If ℓ = *d* (by luck!), then optimal**
  - But:
    - **If ℓ < *d* then incomplete ☹**
    - **If ℓ > *d* then not optimal ☹**

- **Time complexity:** $O(b^l)$
- **Space complexity:** $O(bl)$ ☺

Iterative deepening search

- **A general strategy to find best depth limit l.**
  - **Key idea: use Depth-limited search as subroutine, with increasing l.**

```
For d = 0 to ∞ do
      depth-limited-search to level d
      if it succeeds
            then return solution
```

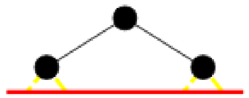
  - ***Complete & optimal*: Goal is always found at depth *d*, the depth of the shallowest goal-node.**

Could this possibly be efficient?

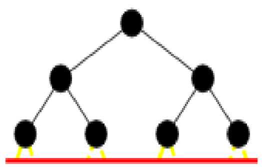
Nodes constructed at each deepening

- Depth 0: 0 (Given the node, doesn't *construct* it.)

- Depth 1: b^1 nodes



- Depth 2: b nodes + b^2 nodes



- Depth 3: b nodes + b^2 nodes + b^3 nodes
- …

Total nodes constructed:

- **Depth 0: 0 (Given the node, doesn't *construct* it.)**
- **Depth 1: $b^1 = b$ nodes**
- **Depth 2: b nodes + b^2 nodes**
- **Depth 3: b nodes + b^2 nodes + b^3 nodes**
- **…**

Suppose the first solution is the last node at depth 3:

Total nodes constructed:

3**b nodes +* ***2*****b^2 nodes +* ***1*****b^3 nodes*

ID search, Evaluation II: Time Complexity

- **More generally, the time complexity is**
  - $N(IDS) = (d)b + (d-1)b^2 + \ldots + (1)b^d =$ *$O(b^d)$*

- ***As efficient in terms of O(…)*** **as Breadth First Search:**
  - $N(BFS) = b + b^2 + \ldots + b^d =$ *$O(b^d)$*

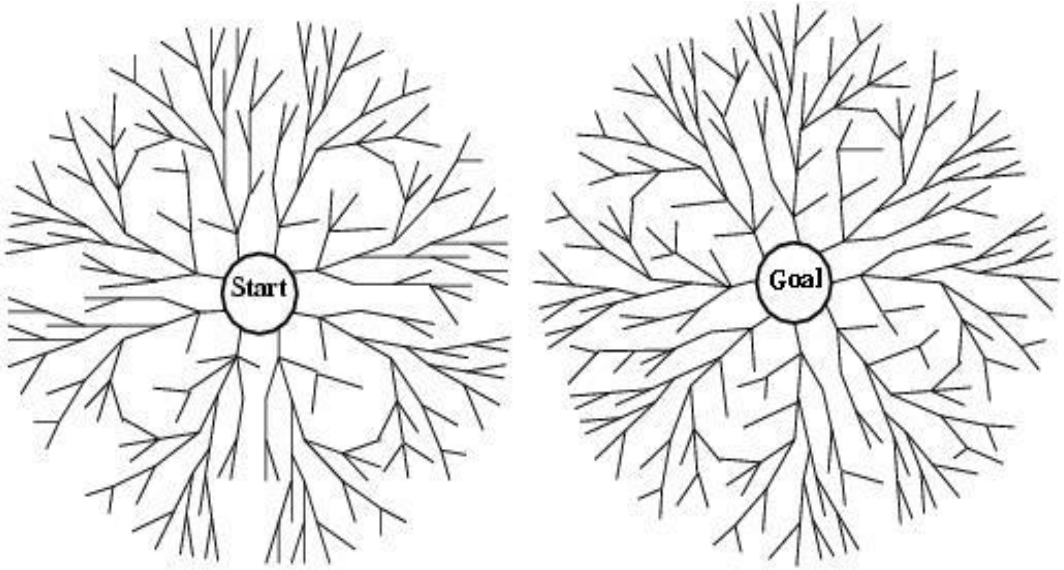
ID search, Evaluation III

- **Complete: YES (no infinite paths)**
- **Time complexity:** $O(b^d)$
- **Space complexity:** $O(bd)$
- **Optimal: YES if step cost is 1.**

Summary of algorithms

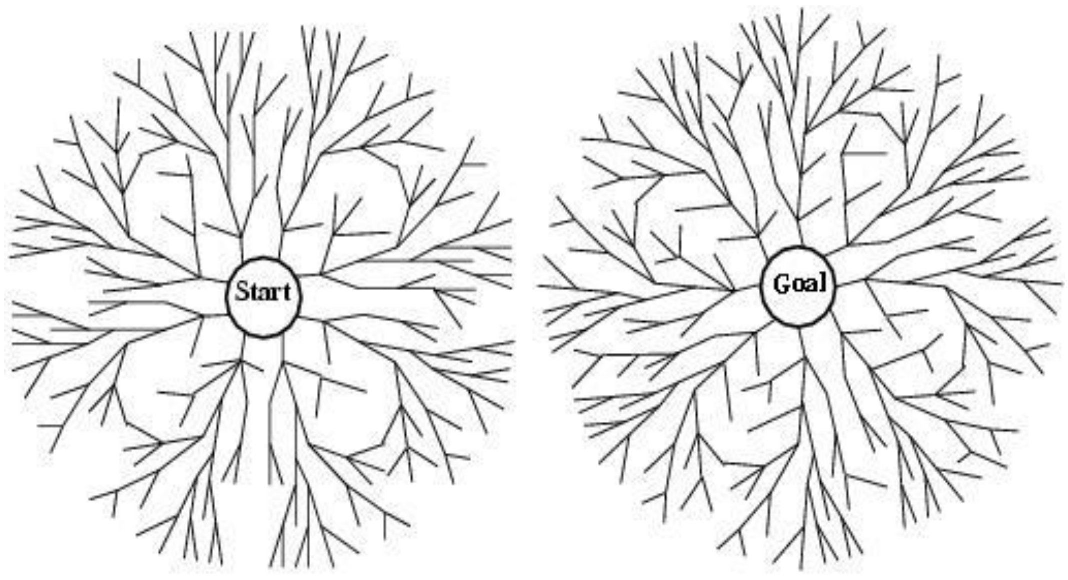
Criterion	Breadth-First	Depth-First	Depth-limited	Iterative deepening
Complete?	**YES**	**NO**	**NO**	**YES**
Time	b^d	b^m	b^l	b^d
Space	b^d	bm	bl	bd
Optimal?	**YES**	**NO**	**NO**	**YES**

Very briefly: Bidirectional search



- **Two simultaneous searches from start an goal.**
  - Motivation: $$b^{d/2} + b^{d/2} < b^{d}$$
- **Check whether the node belongs to the other frontier before expansion.**
- **Space complexity is the most significant weakness.**
- **Complete and optimal if both searches are Breadth-First.**

How to search backwards?

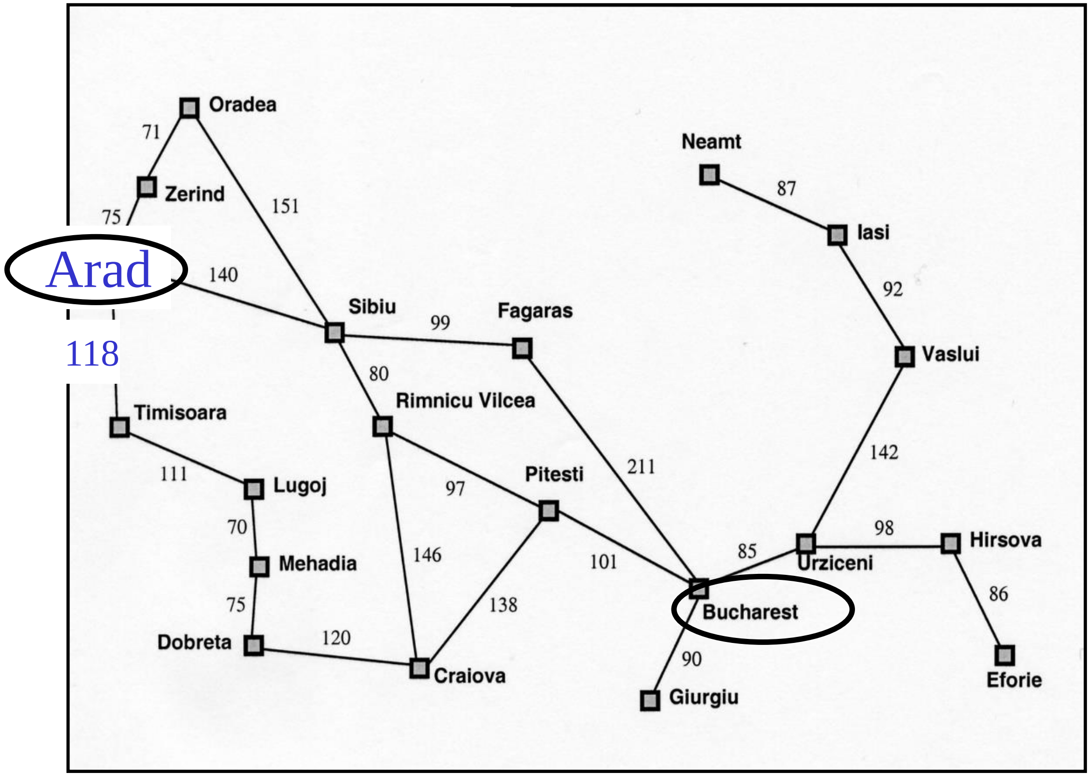


- **The predecessor of each node must be efficiently computable.**
  - Works well when actions are easily reversible.

"Uniform Cost" Search

"In computer science, *uniform-cost* search (UCS) is a tree search algorithm used for traversing or searching a *weighted* tree, tree structure, or graph." - Wikipedia

Motivation: Romanian Map Problem



- **All our search methods so far assume**
  step-cost = 1
- ***This is only true for some problems***

g(N): the *path cost* function

- **If all moves equal in cost:**
  - Cost = # of nodes in path-1
  - g(N) = depth(N) in the search tree
  - Equivalent to what we've been assuming so far

- **Assigning a (potentially) unique cost to each step**
  - N_0, N_1, N_2, N_3 = nodes visited on path *p* from N_0 to N_3
  - C(i,j): Cost of going from N_i to N_j
  - If N_0 the root of the search tree,

$$g(N_3)=C(0,1)+C(1,2)+C(2,3)$$

Uniform-cost search (UCS)

- **Extension of BF-search:**
  - **Expand node with *lowest path cost***
- **Implementation:**
  ***frontier* = priority queue ordered by g(n)**
- **Subtle but significant difference from BFS:**
  - Tests if a node is a goal state when it is selected for expansion, not when it is added to the frontier.
  - Updates a node on the frontier if a better path to the same state is found.
  - So always enqueues a node *before checking whether it is a goal.*

WHY???

Uniform Cost Search

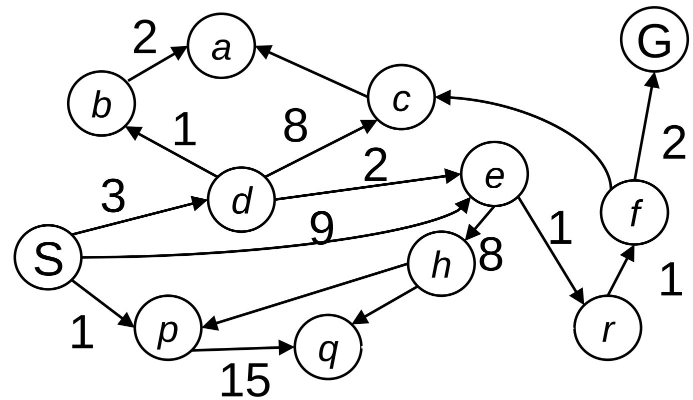
Slide from Stanford CS 221 (from slide by Dan Klein (UCB) and many others)

Expand cheapest node first:

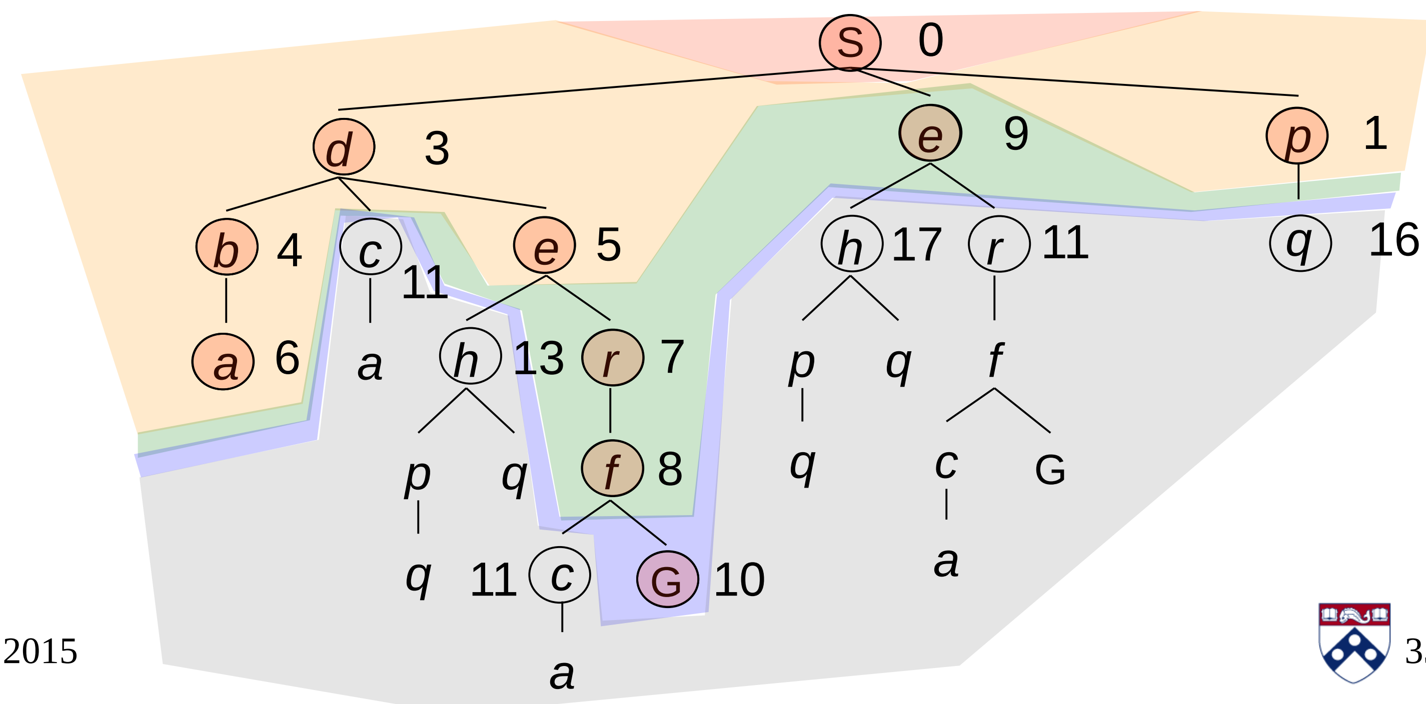
Frontier is a priority queue

No longer ply at a time, but follows *cost contours*

Therefore: Must be optimal



Cost contours



Complexity of UCS

- **Complete!**
- **Optimal!**
  - if the cost of each step exceeds some positive bound ε.
- **Time complexity:** $O(b^{1 + C^*/\varepsilon})$
- **Space complexity:** $O(b^{1 + C^*/\varepsilon})$

where C is the cost of the optimal solution*

(if all step costs are equal, this becomes $O(b^{d+1})$

NOTE: Dijkstra's algorithm just UCS without goal

Summary of algorithms (for notes)

Criterion	Breadth-First	Uniform-cost	Depth-First	Depth-limited	Iterative deepening	Bidirectional search
Complete?	**YES**	**YES**	**NO**	**NO**	**YES**	**YES**
Time	b^d	$b^{1+C^*/e}$	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	$b^{1+C^*/e}$	bm	bl	bd	$b^{d/2}$
Optimal?	**YES**	**YES**	**NO**	**NO**	**YES**	**YES**

Assumes b is finite