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

Lecture 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

Reminder

- Homework 1: Search
  - Deadline: Oct 10, 2020

- Project 1: Search
  - Deadline: Oct 13, 2020

Today

- Uninformed Search
  - Uniform Cost Search

- Informed Searc
  - Heuristics
  - Greedy Search
  - A* Search

- Graph Search

Recap: Search

- Search problem:
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test

- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)

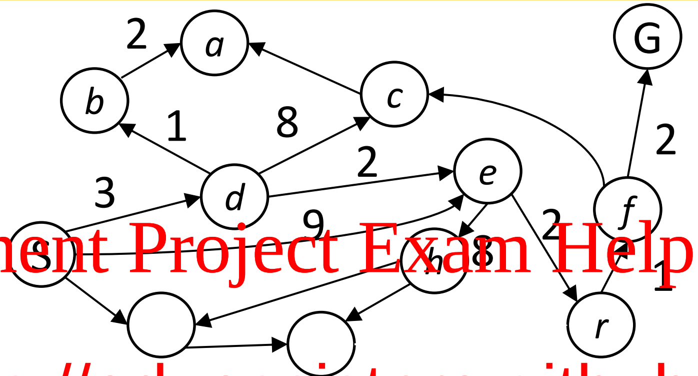
- Search algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans

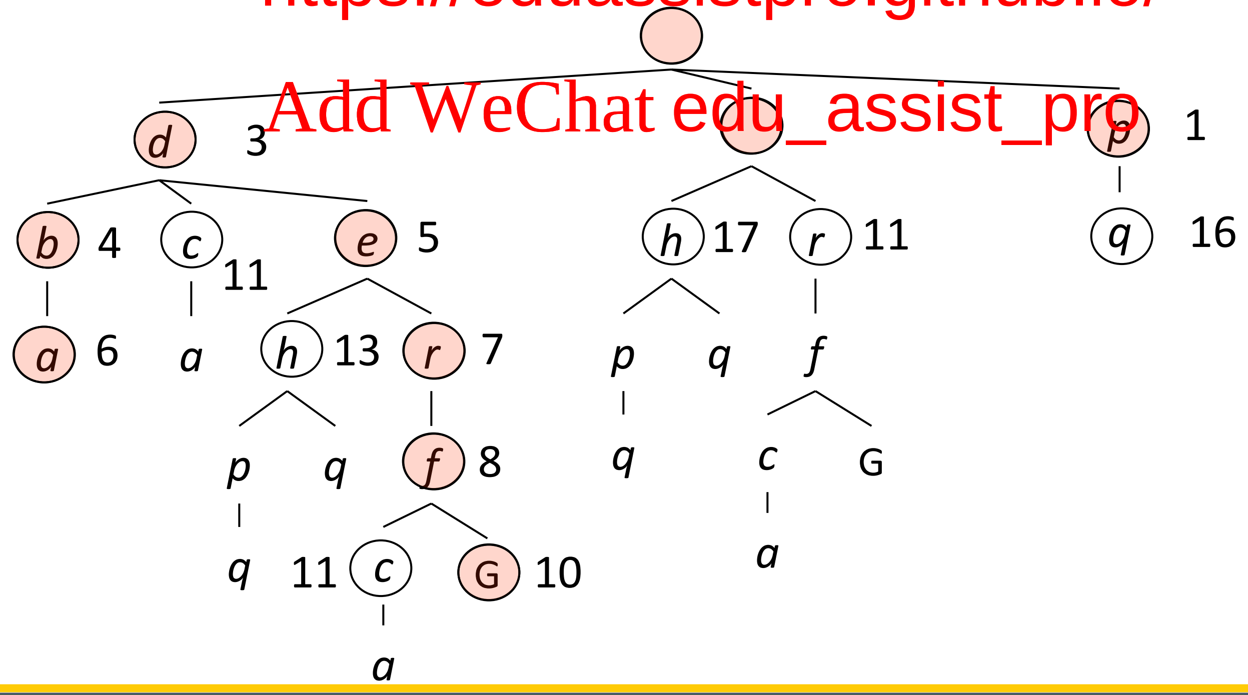
Uninformed Search

Uniform-Cost Search (UCS)

Strategy: expand a cheapest node first:

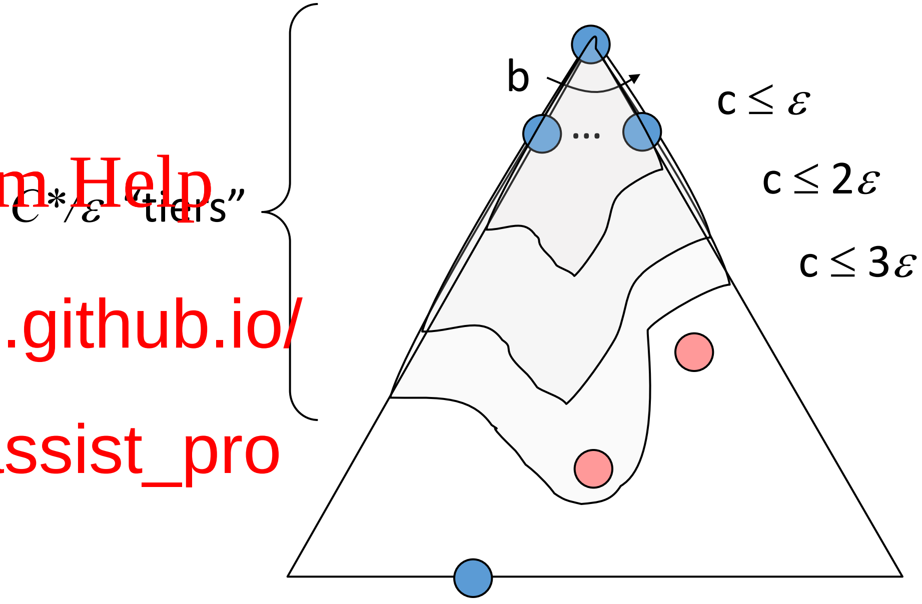
Fringe is a priority queue (priority: cumulative cost)





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 $O(b^{C^*/\varepsilon})$ (exponentia



- How much space does the fri
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$

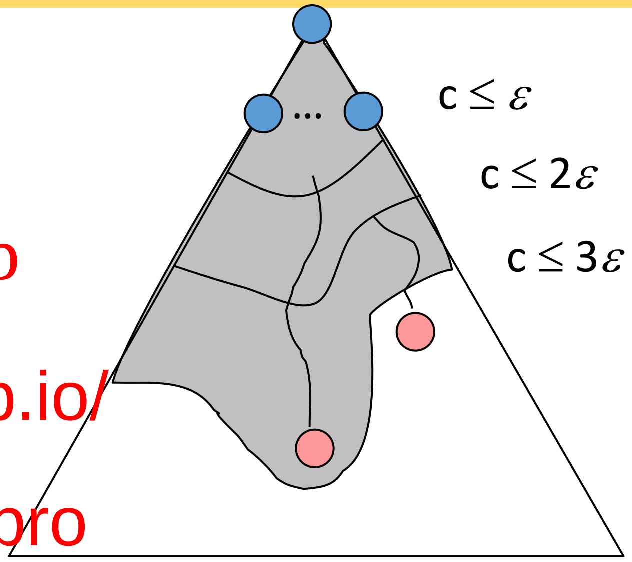
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!

- Is it optimal?
  - Yes!

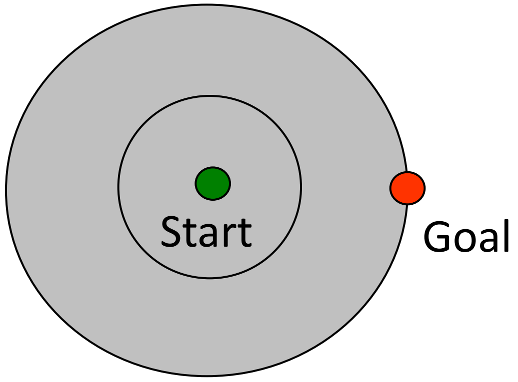
Uniform Cost Search

- Strategy: expand lowest path cost

- The good: UCS is com



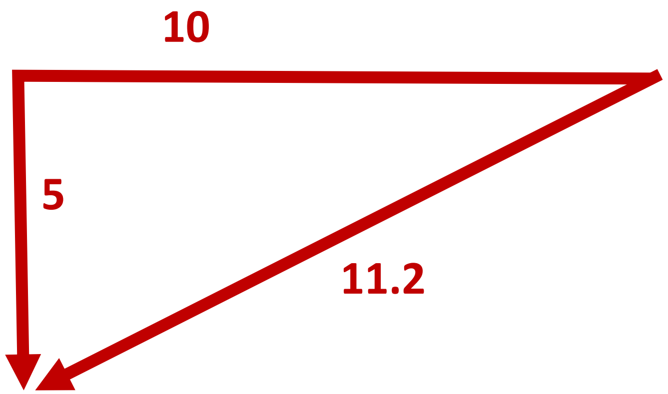
- The bad:
  - Explores options in every “direction”
  - No information about goal location

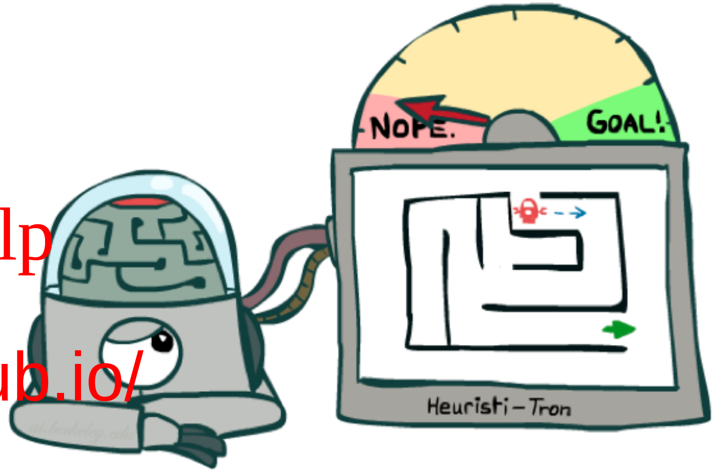


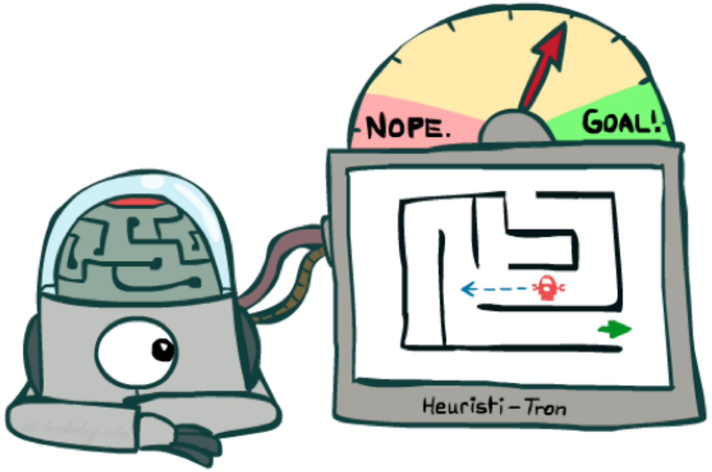
Informed Search

Search Heuristics

- A heuristic is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing



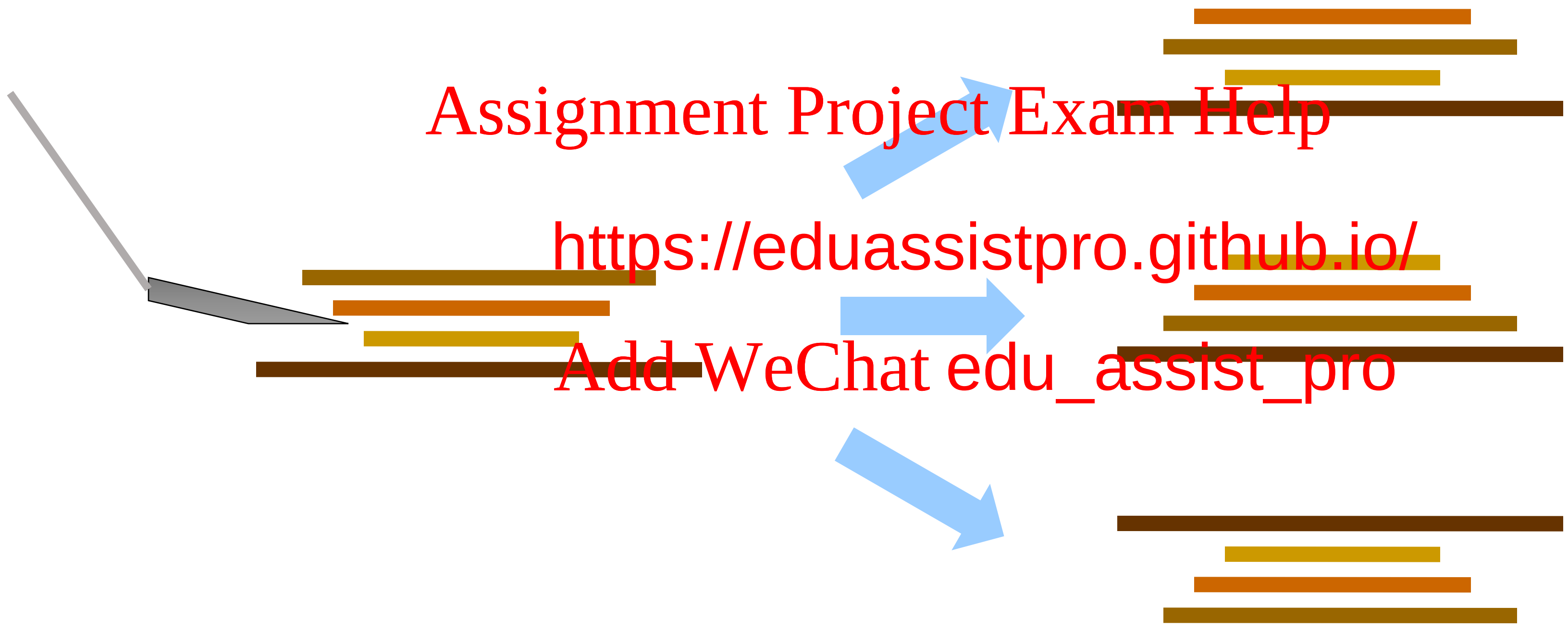




Example: Heuristic Function

h(x)

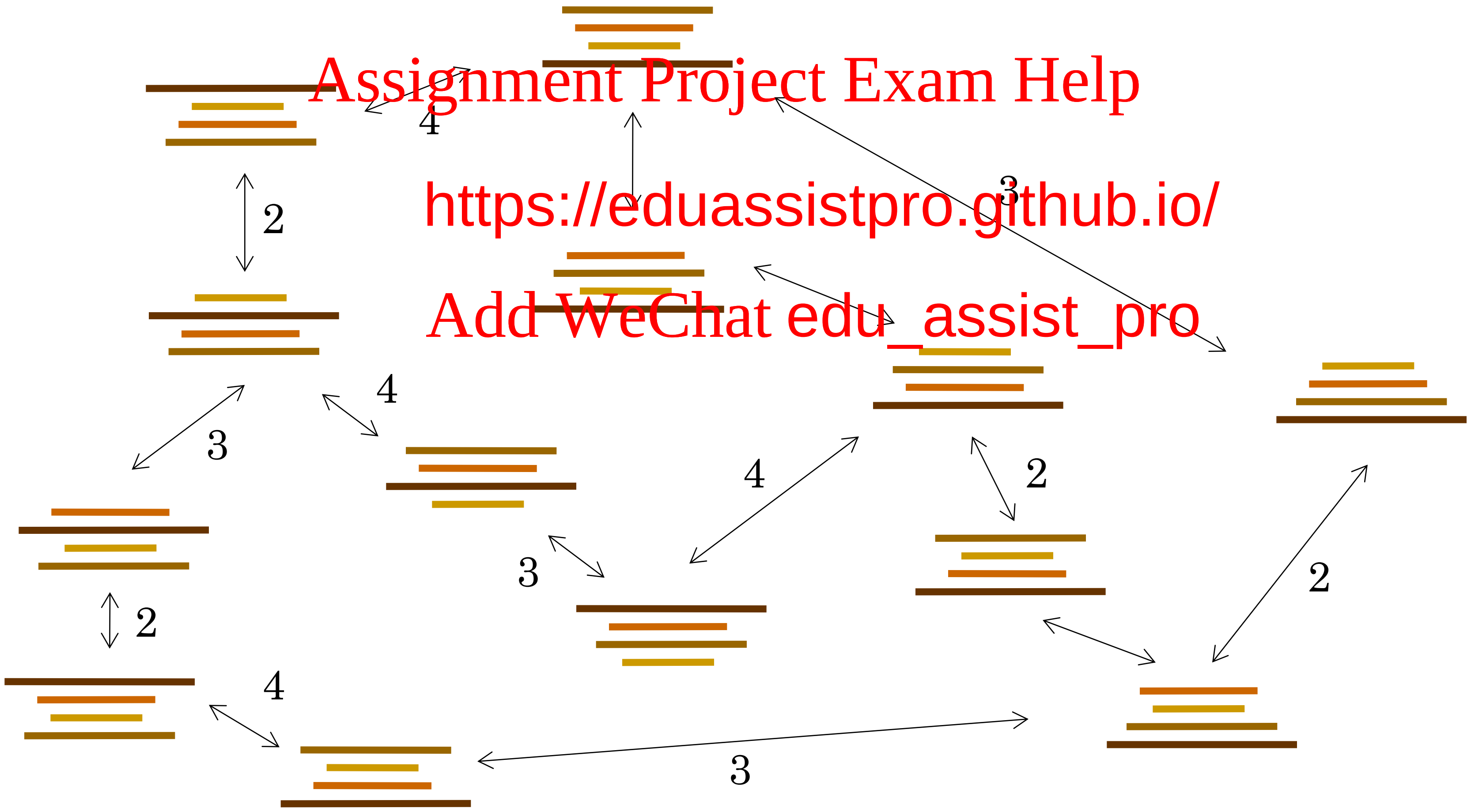
Example: Pancake Problem



Cost: Number of pancakes flipped

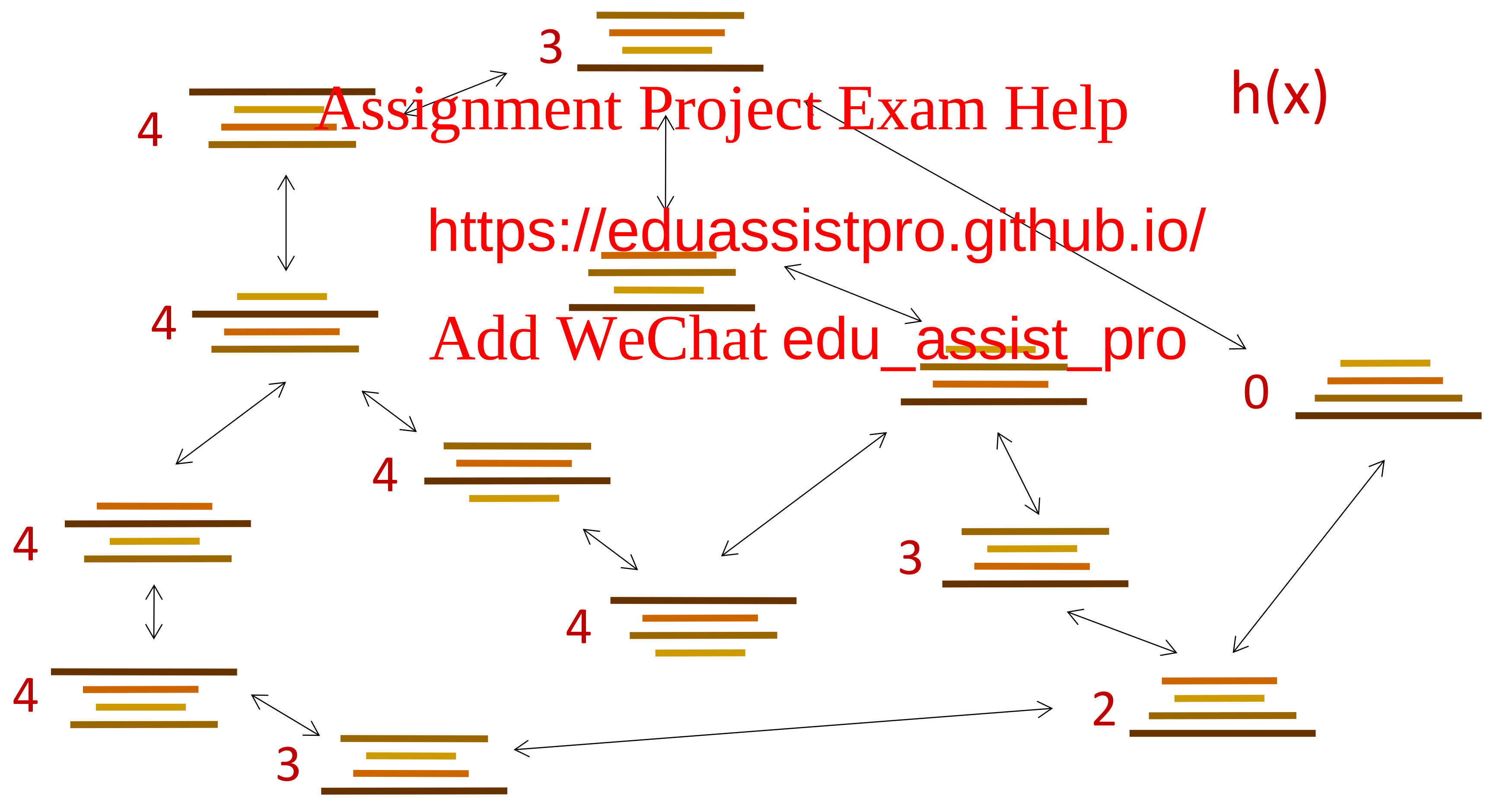
Example: Pancake Problem

State space graph with costs as weights



Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place



Greedy Search

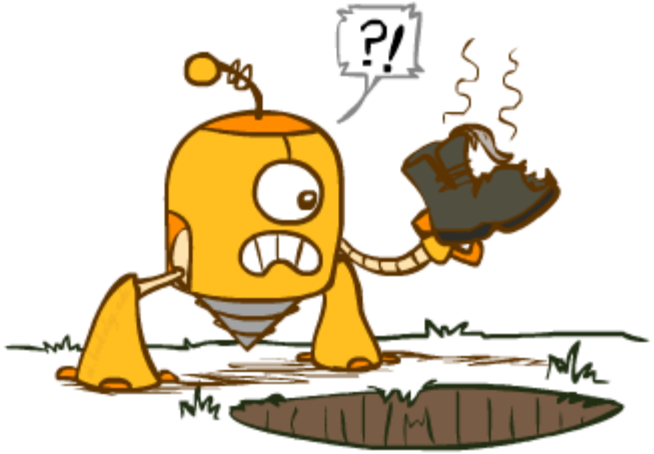
Greedy Search

- Expand the node that seems closest…

h(x)

Greedy Search

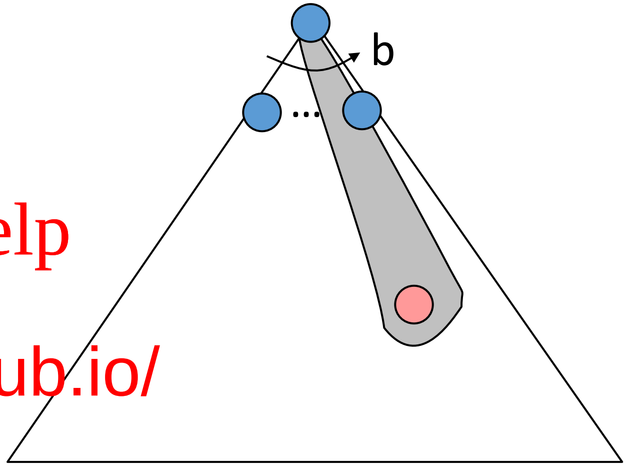
- Expand the node that seems closest…



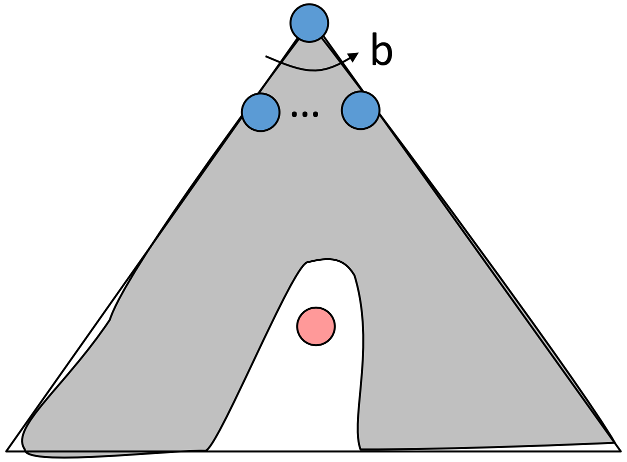
- What can go wrong?

Greedy Search

- A common case:
  - Best-first takes you stra l



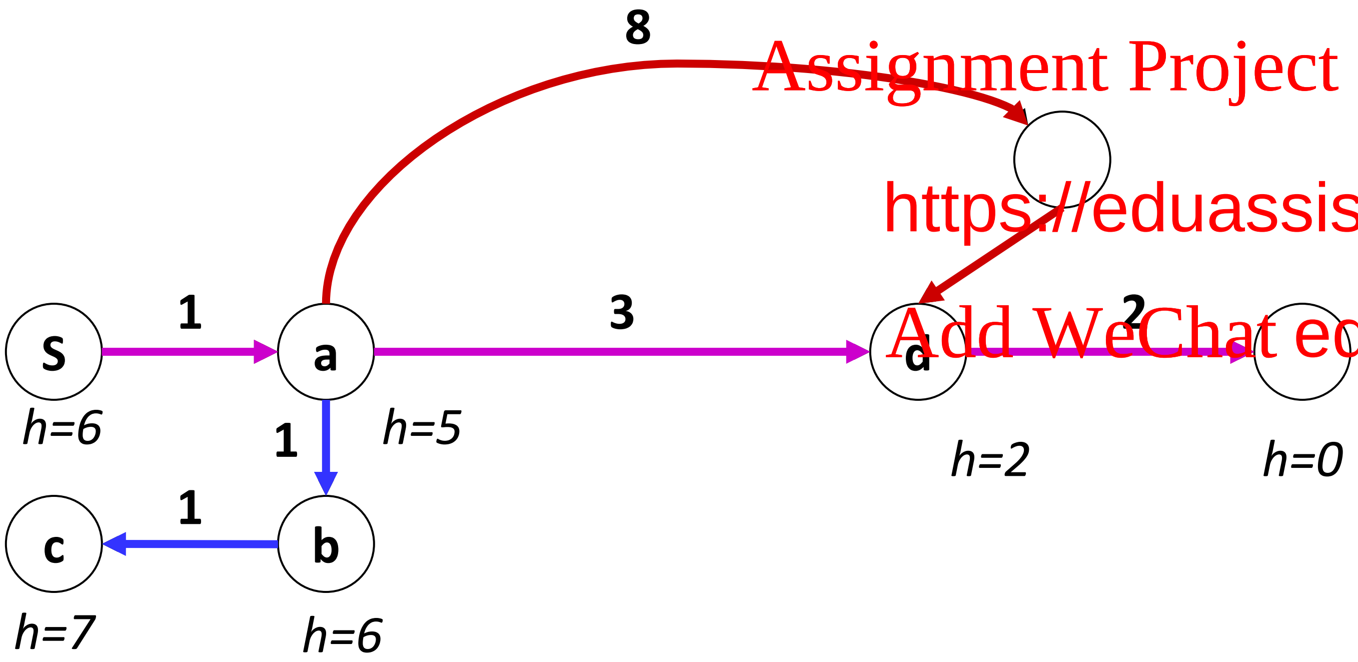
- Worst-case: like a badly-guided DFS

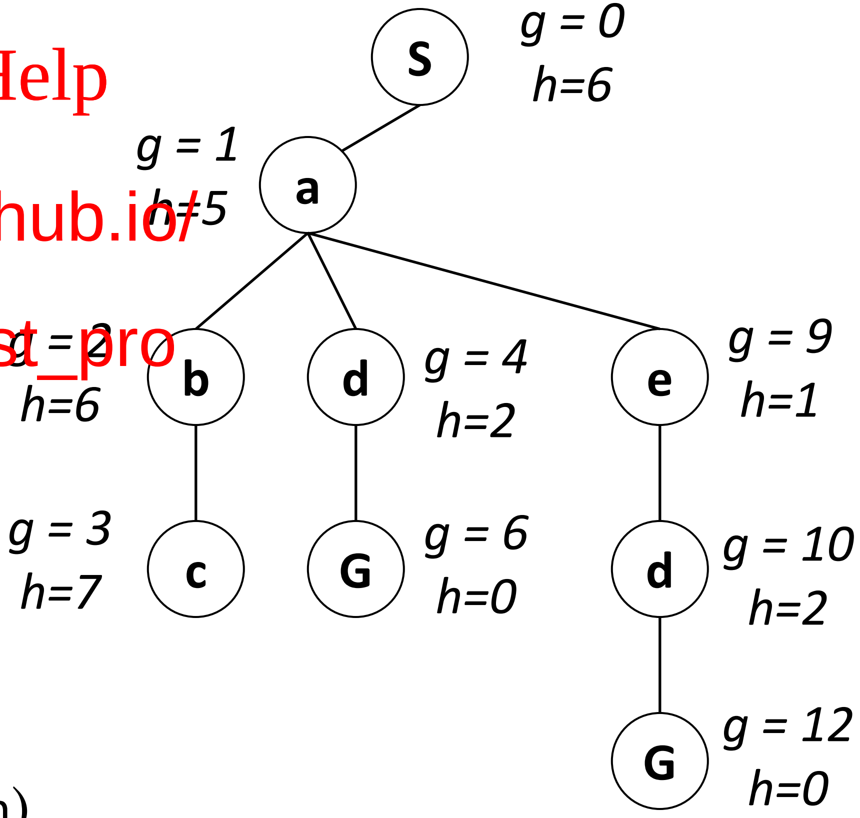


A* Search

Combining UCS and Greedy

- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)

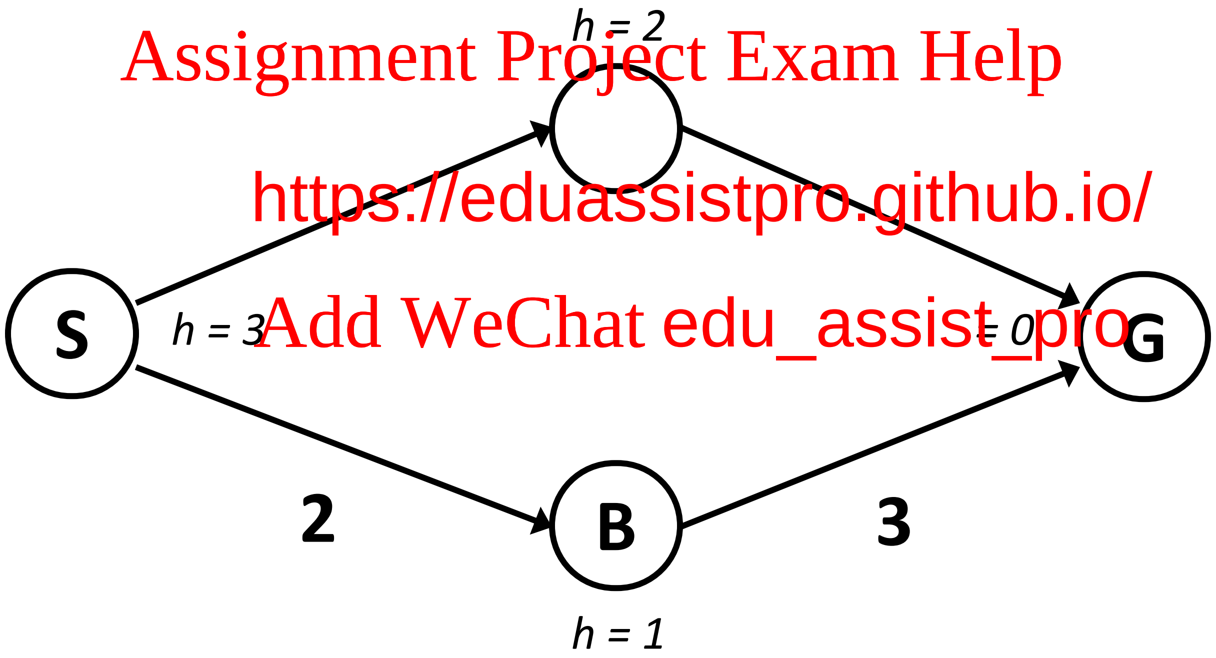




- A* Search orders by the sum: f(n) = g(n) + h(n)

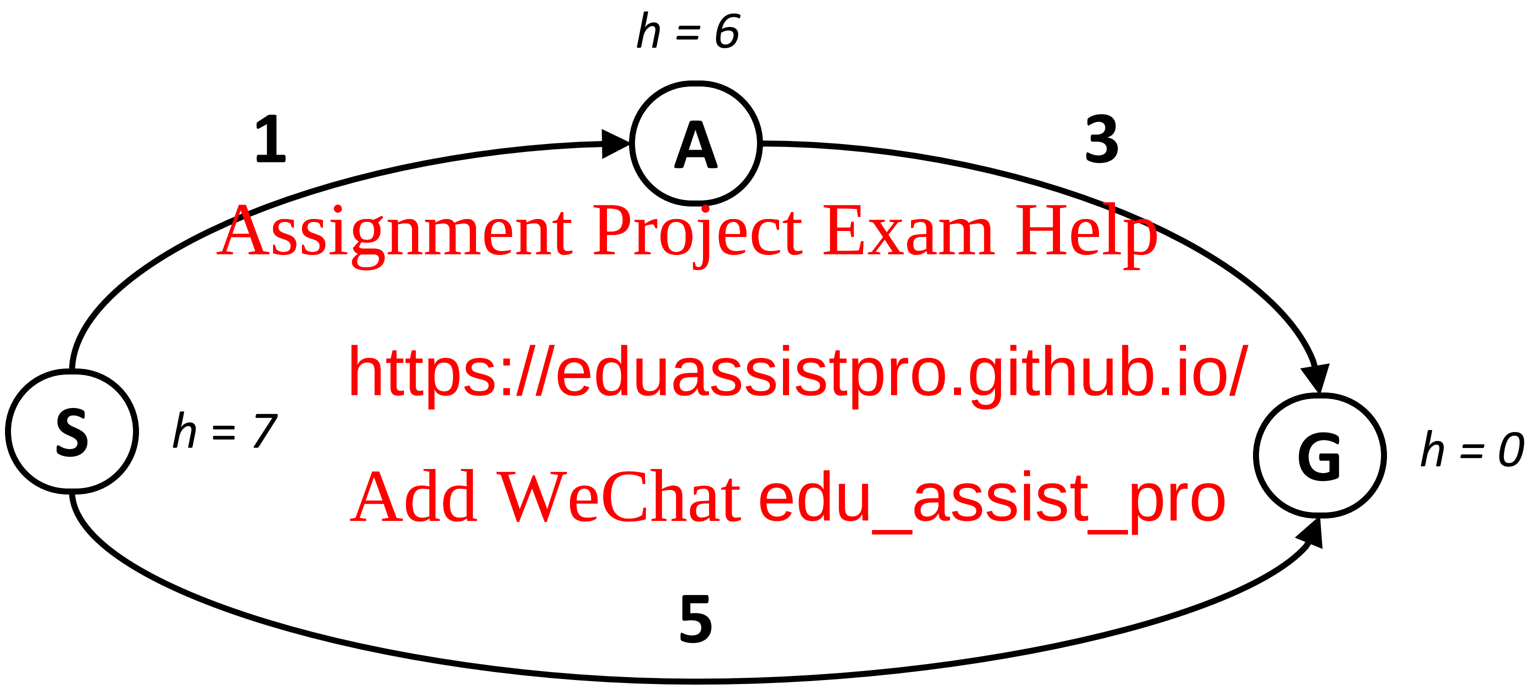
When should A* terminate?

- Should we stop when we enqueue a goal?



- No: only stop when we dequeue a goal

Is A* Optimal?



- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

Admissible Heuristics

Idea: Admissibility

Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe

Admissible (optimistic) heuristics never outweigh true costs

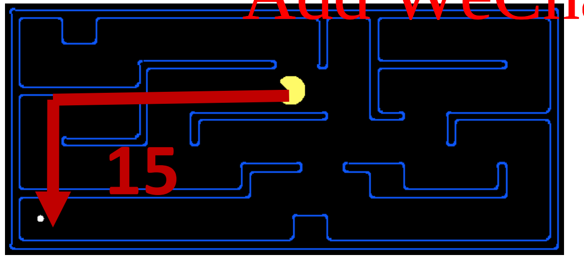
Admissible Heuristics

- A heuristic h is *admissible* (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is th earest goal

- Examples:



- Coming up with admissible heuristics is most of what's involved in using A* in practice.

Optimality of A* Tree Search

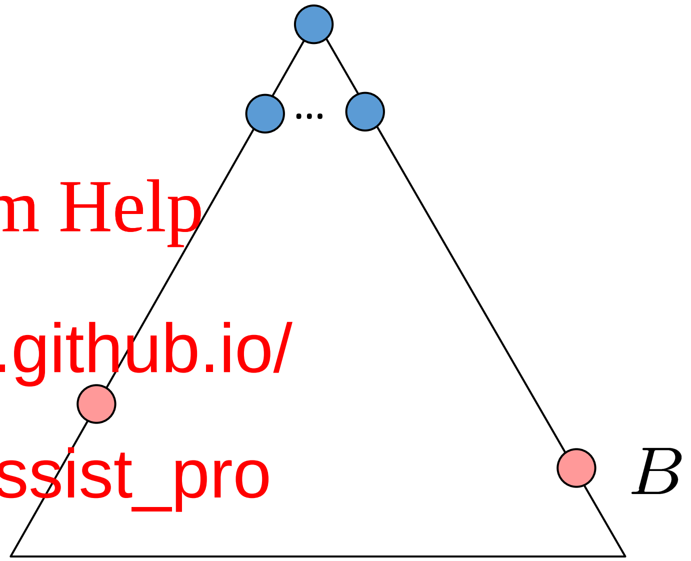
Optimality of A* Tree Search

- Heuristic function h is admissible
- Claim: A* tree search is optimal

Optimality of A* Tree Search

Assume:

- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible



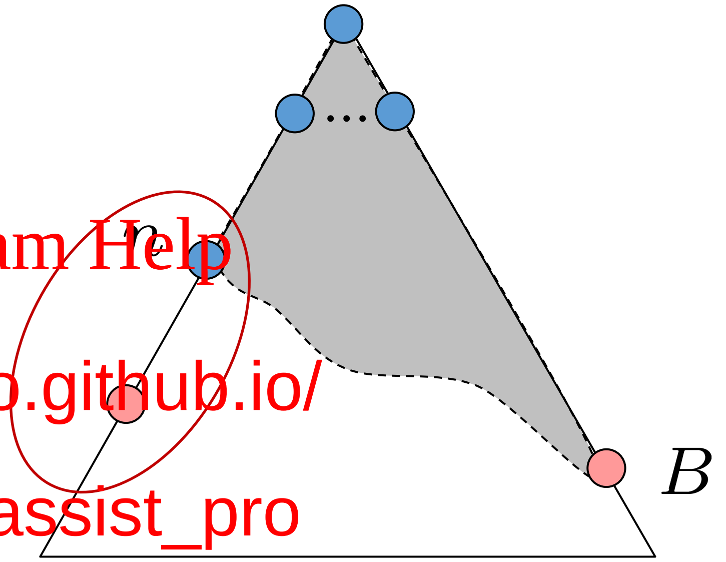
Claim:

- A will exit the fringe before B

Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded
  1. f(n) is less or equal to f(A)

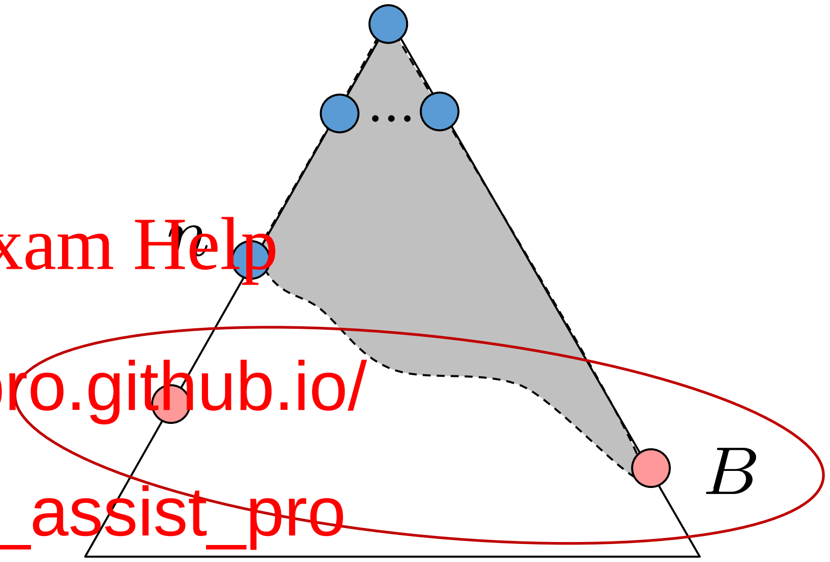


$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)



$$g(A) < g(B)$$

B is suboptimal

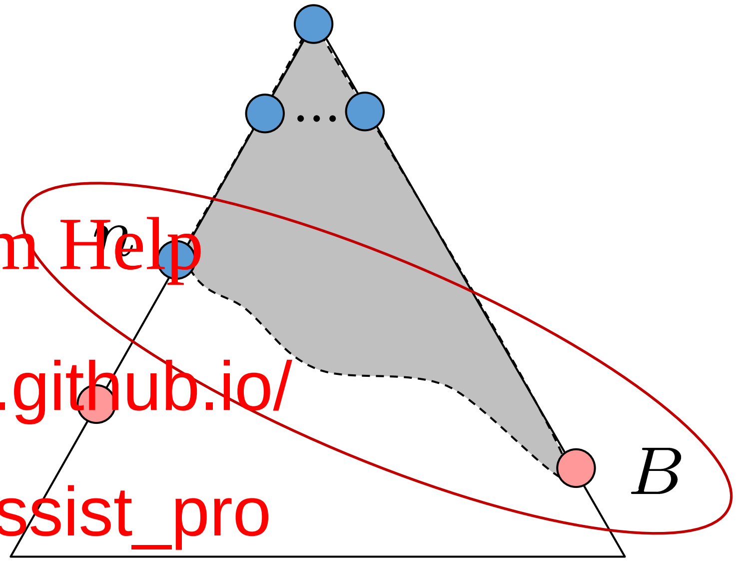
$$f(A) < f(B)$$

h = 0 at a goal

Optimality of A* Tree Search: Blocking

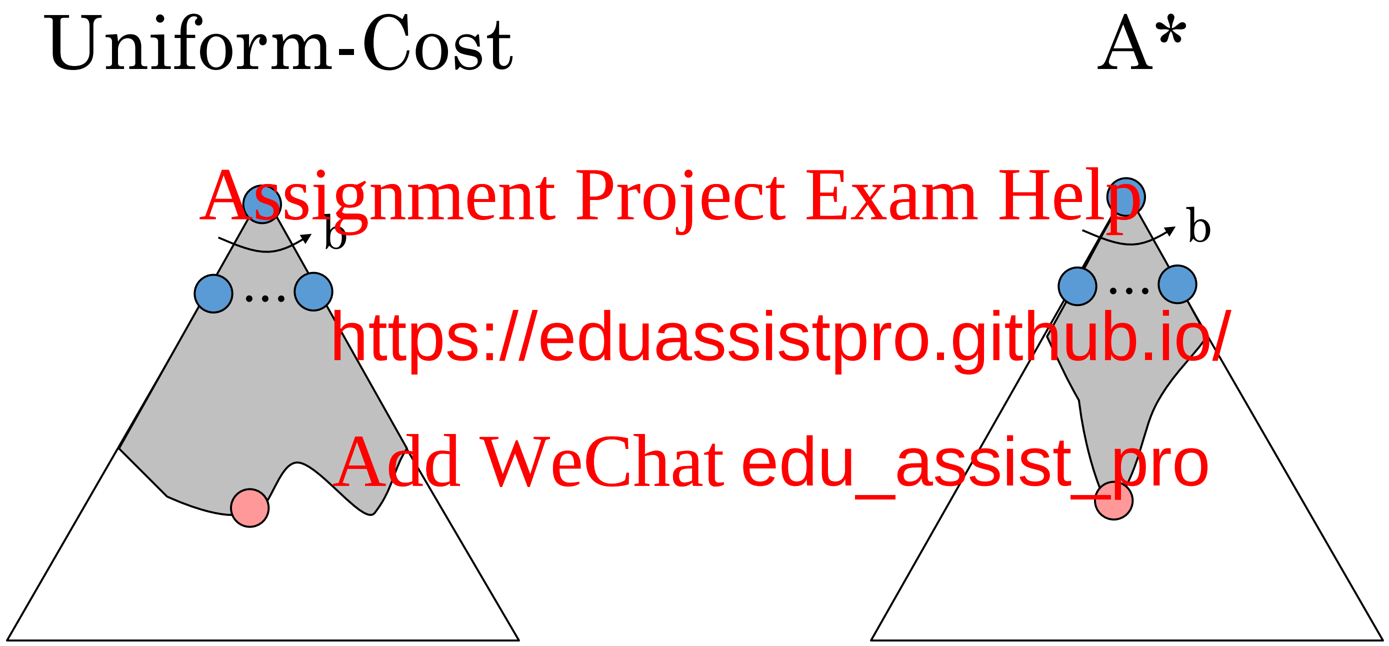
Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)
  3. n expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal



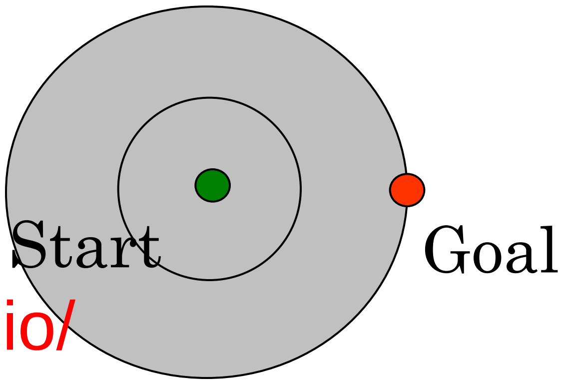
$$f(n) \leq f(A) < f(B)$$

Properties of A*

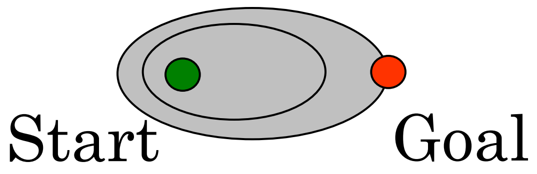


UCS vs A* Contours

- Uniform-cost expands equally in all "directions"



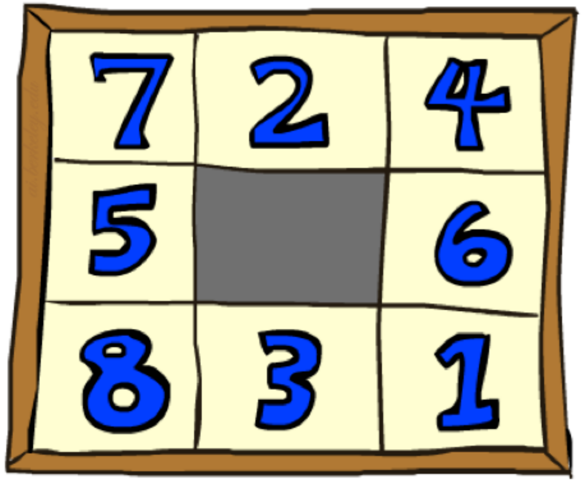
- A* expands mainly toward t goal, but does hedge its bets to ensure optimality



A* Applications

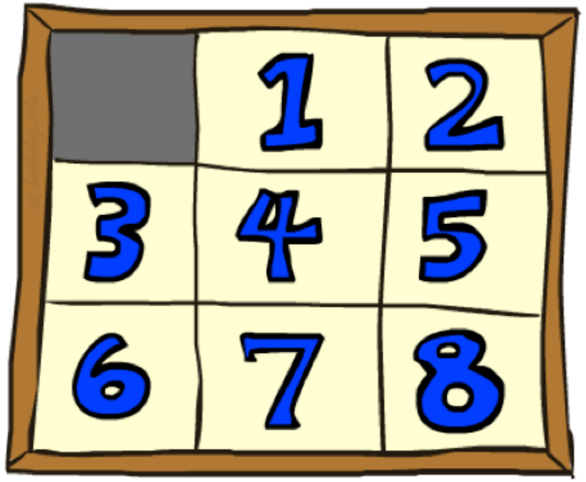
- Video games
- Pathing / routing problems
- Resource planning
- Robot motion plann
- Language analysis
- Machine translation
- Speech recognition
- …

Example: 8 Puzzle



Start State

A

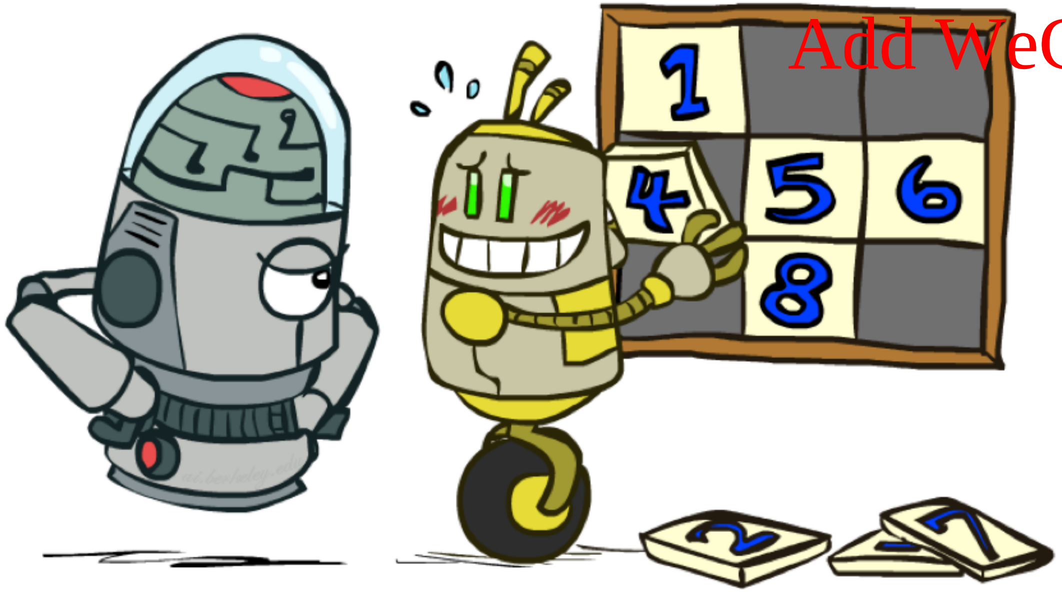


Goal State

- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle I

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a *relaxed-probl*



t State

Goal State

	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

Statistics from Andrew Moore

8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?

- Total *Manhattan* distance

tate

Goal State

- Why is it admissible?

- h(start) =

  3 + 1 + 2 + … = 18

	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
TILES	13	39	227
MANHATTAN	12	25	73

8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?

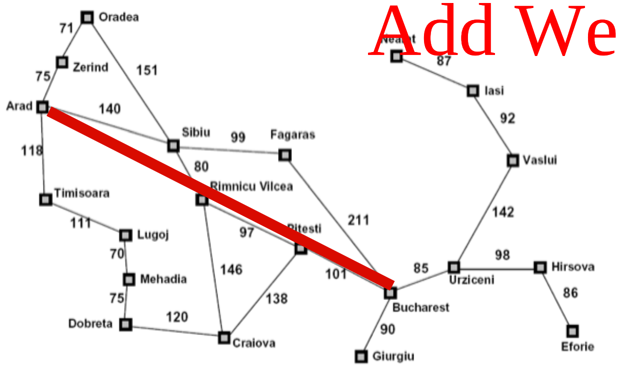
- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

- Often, admissible heuristics are solutions to *relaxed problems*, wher lable

366



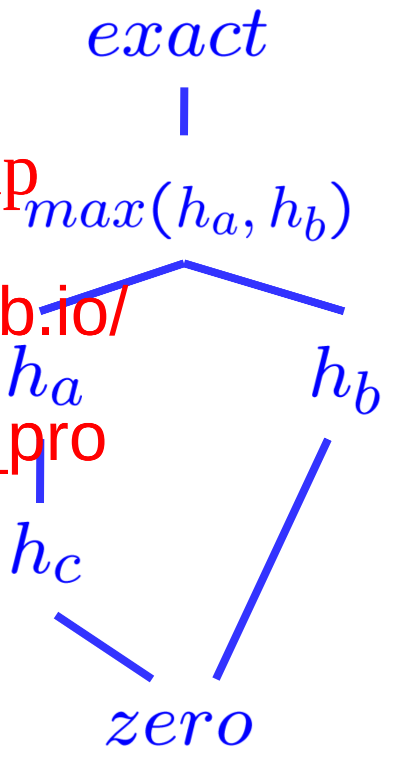
- Inadmissible heuristics are often useful too (why?)

Trivial Heuristics, Dominance

- Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

- Heuristics form a
  - Max of admissibl



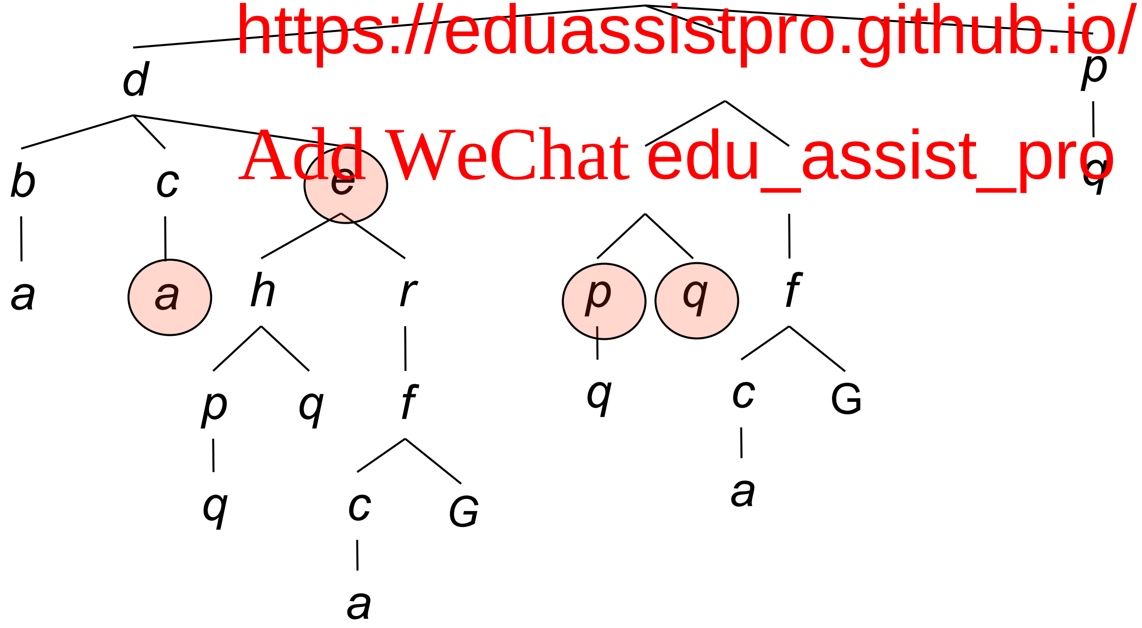
- Trivial heuristics
  - Bottom of lattice is the zero heuristic (what does this give us?)
  - Top of lattice is the exact heuristic

Tree Search: Extra Work!

- Failure to detect repeated states can cause exponentially more work. Why?

Graph Search

- In BFS, for example, we shouldn't bother expanding some nodes (which, and why?)



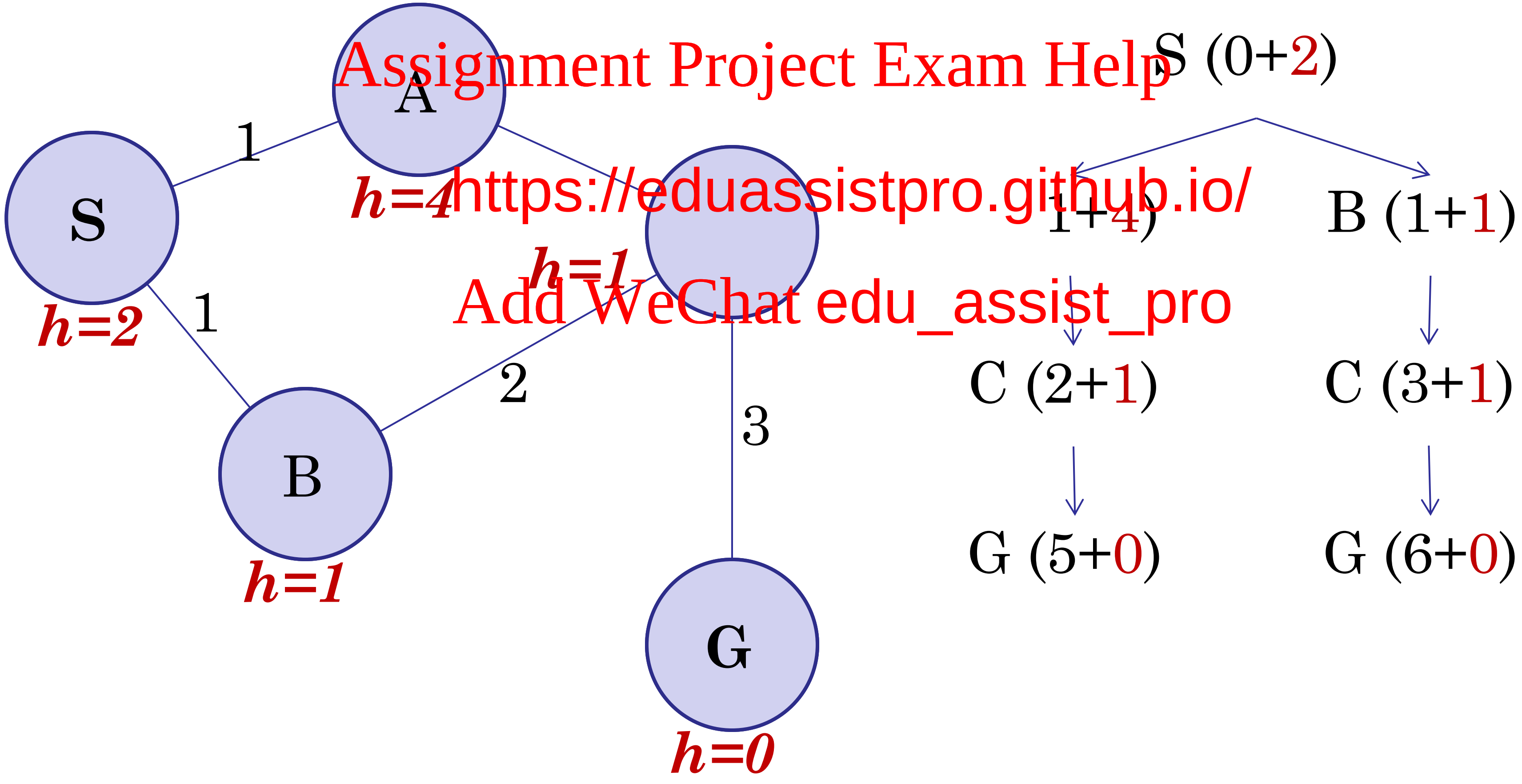
Graph Search

- Very simple fix: never expand a state type twice

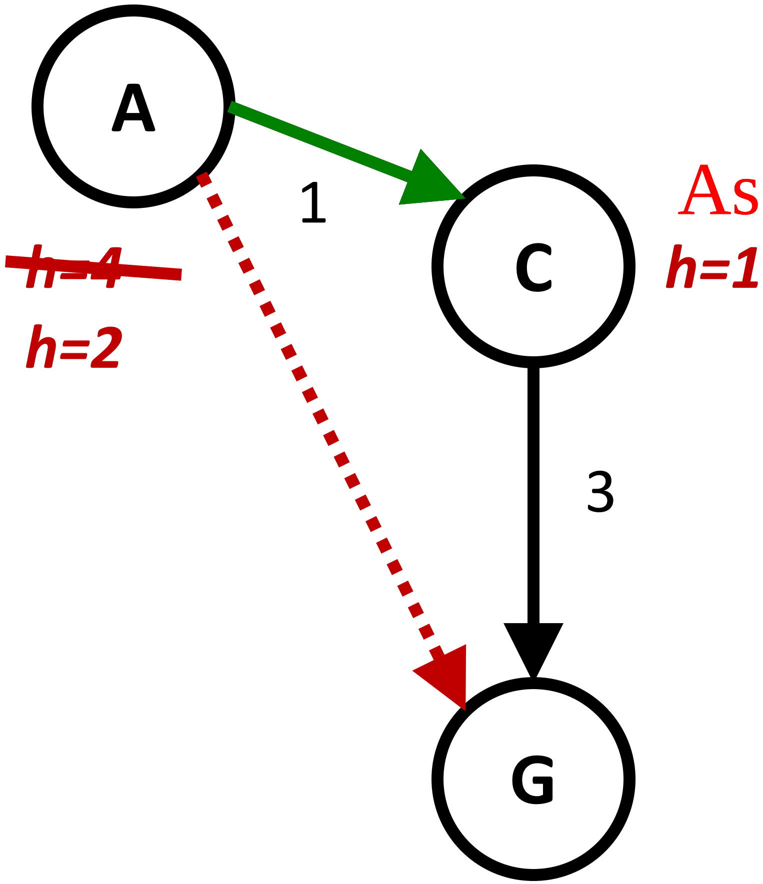
- Can this wreck completeness? Why or why not?
- How about optimality? Why or why not?

A* Graph Search Gone Wrong

State space graph **Search tree**



Consistency of Heuristics



- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

    h(A) ≤ actual cost from A to G

  - "arc" cost ≤ actual cost for each arc

    h(A) – ... to C)

- Consequences of consistency:
  - The f value along a path never decreases

    h(A) ≤ cost(A to C) + h(C)

    f(A) = g(A) + h(A) ≤ g(A) + cost(A to C) + h(C) ≤ f(C)

  - A* graph search is optimal

Optimality of A* Graph Search

Optimality of A* Graph Search

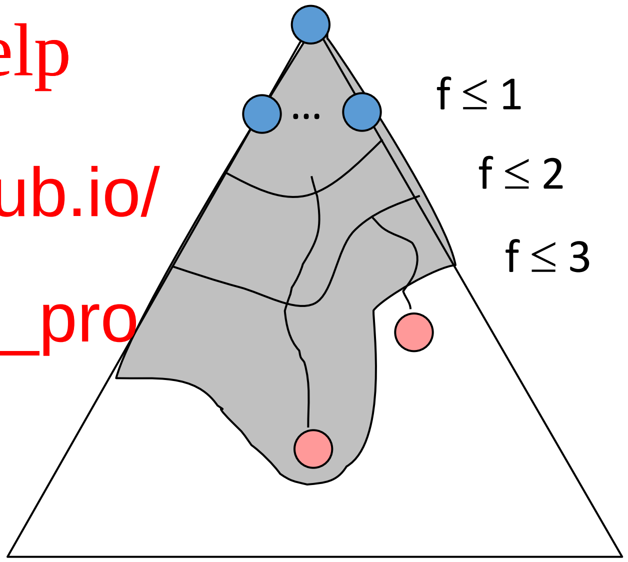
- Sketch: consider what A* does with a consistent heuristic:

  - Fact 1: In tree search, A* expands nodes in increasing t

    contours)

  - Fact 2: For every state s, nodes tha reach s optimally are expanded before nodes that reach s suboptimally

  - Result: A* graph search is optimal



Optimality

- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)

- Graph search:
  - A* optimal if heuristic is cons
  - UCS optimal (h = 0 is consistent)

- Consistency implies admissibility

- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems

A*: Summary

A*: Summary

- A* uses both backward costs and (estimates of) forward costs

- A* is optimal with stent heuristics

- Heuristic design is key: often xed problems

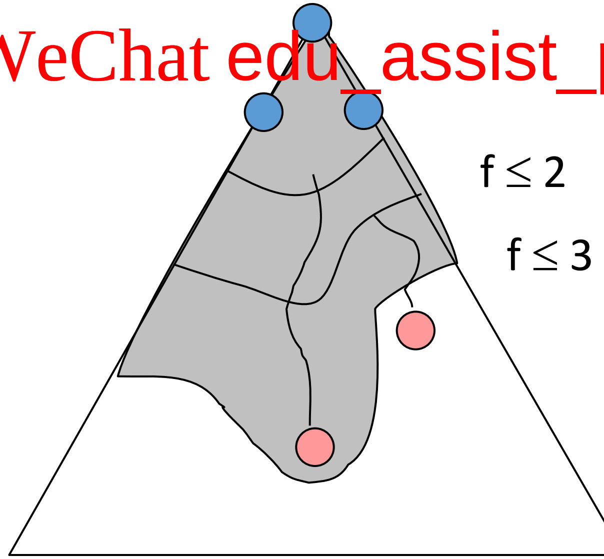
Tree Search Pseudo-Code

Graph Search Pseudo-Code

Optimality of A* Graph Search

- Consider what A* does:
  - Expands nodes in increasing total f value (f-contours)
    Reminder: f(n) = g(n) + h(n) = cost to n + heuristic
  - Proof idea: the optimal f
    value, so it must get exp

There's a problem with this argument.
What are we assuming is true?



Optimality of A* Graph Search

Proof:

- New possible problem: some *n* on path to G* isn't in queue when we need it, because some worse *n'* for the same state dequeued and expande
- Take the highest such
- Let *p* be the ancestor of *n* that was on t queue when *n'* was popped
- *f(p)* < *f(n)* because of consistency
- *f(n)* < *f(n')* because *n'* is suboptimal
- *p* would have been expanded before *n'*
- Contradiction!