

ECE 484: Principles of Safe Autonomy (Fall 2026)
Lecture 15: Search and Planning

Professor: Huan Zhang
huanz@illinois.edu

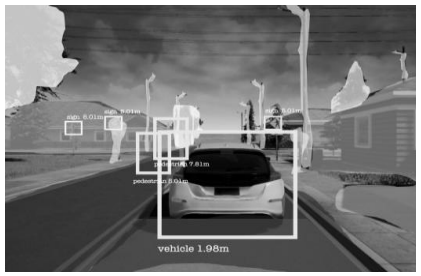
Slides adapted from Prof. Sayan Mitra's slides for Spring 2025

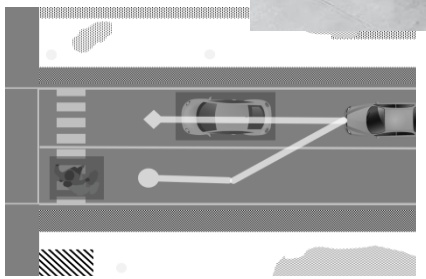
Autonomy pipeline

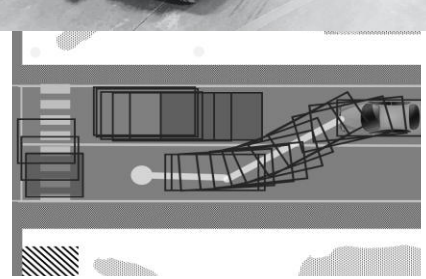
GEM platform











Sensing	Perception	Decisions and planning	Control
Physics-based models of camera, LIDAR, RADAR, GPS, etc.	Programs for object detection, lane tracking, scene understanding, etc.	Programs and multi-agent models of pedestrians, cars, etc.	Dynamical models of engine, powertrain, steering, tires, etc.

Final review of localization module

Fundamentals: **Conditional Probability**, **Bayes' rule**

Motion model and measurement model

Bayes Filters: prediction + correction steps

- Discrete Bayes Filter
- Particle Filter
- Kalman Filter

Pros and cons of each type of filter

From localization to planning

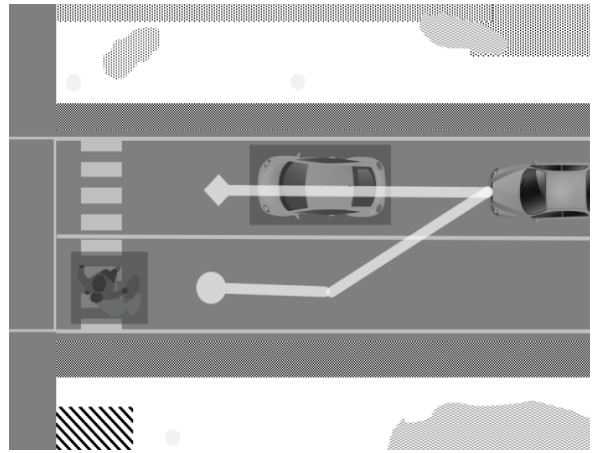
Where am I?

Localization estimates the vehicle state.

Where should I go next?

Planning uses the state, a map, and a goal to find a collision-free path.

Control follows the planned path.



Decisions and planning
Programs and multi-agent models of pedestrians, cars, etc.

Outline

Deterministic search

- Uninformed search
- Informed search
- Next: combine cost and heuristic with A*

Later: randomized search

- Probabilistic roadmaps (PRM)
- Rapidly-exploring random trees (RRT)



Search and planning problems appear in different levels of autonomy stack

Global path planner --- invoked at each new *checkpoint*

- finds paths from every point in the map to the next checkpoint

Road navigation

- For each path, the planner rolls out several discrete trajectories that are parallel to the smoothed center of the lane

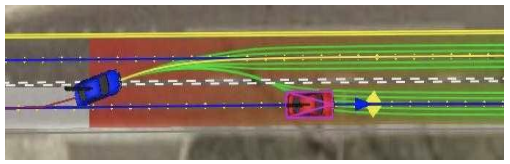
Freeform navigation (parking lots)

- Generate arbitrary trajectories (irrespective of road structure) using modified A*





Road navigation: passing



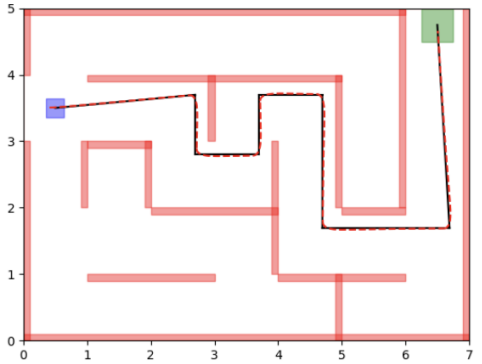
Junior: The Stanford Entry in the Urban Challenge, Montemerlo et al., 2008

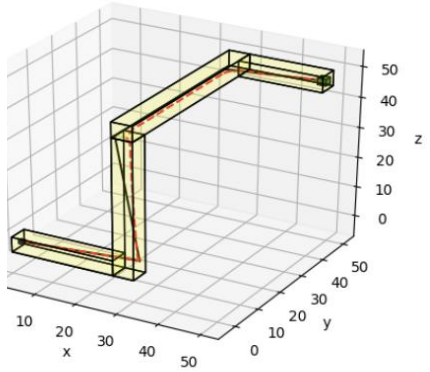
Planning as graph search

- Search for collision-free trajectories can be viewed abstractly as a **graph search** problem, where the nodes represent blocks of free space and the edges connect spatially adjacent nodes.
- We can solve such problems using the **graph search** algorithms like (uninformed) Breadth-First Search and Depth-First Search
- Roadmaps are not arbitrary graphs: Some paths are **preferable** than others (e.g., shorter, faster, less costly in terms of fuel/tolls/fees, more stealthy, etc.)
- Good guesses for **distances** can be made, even without knowing the full graph or the optimal paths
- Can we *utilize this information* to find efficient paths, efficiently?



Making a Drone Smarter With Motion Planning Nicholas Rehm





https://kmmille.github.io/FACTEST/

Problem statement: find shortest path

Input: $\langle V, E, w, start, goal \rangle$

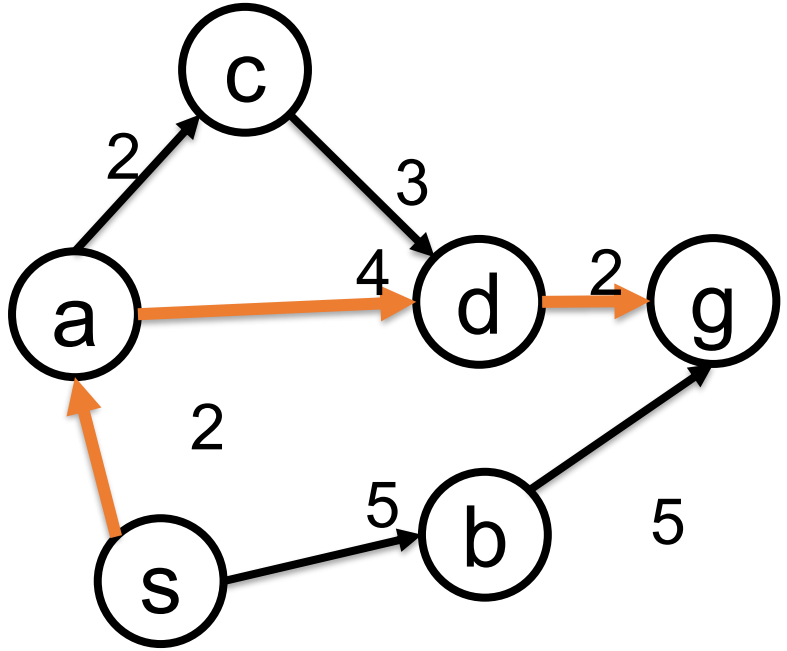
- V: (finite) set of vertices
- $E \subseteq V \times V$: (finite) set of edges
- $w : E \to \mathbb{R}_{>0}$: assigns to each edge e a positive weight $w(e)$
- $start, goal \in V$: start and end vertices.

A path is a sequence of vertices $p = v_0 \ldots v_k$ such that $(v_i, v_{i+1}) \in E$

$w(p) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$ $\quad \text{first}(p) = v_0$ $\quad \text{head}(p) = v_k$

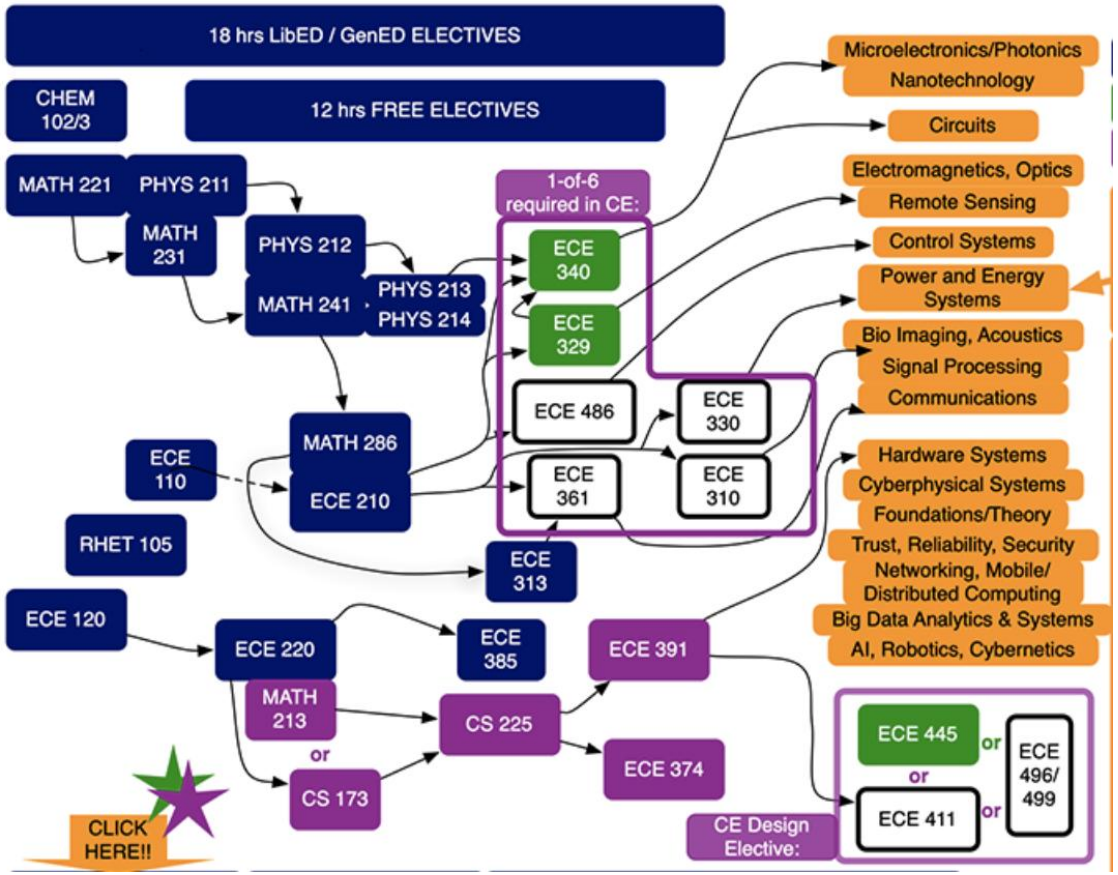
Output: a path p with $\text{first}(p) = start$ and $\text{head}(p) = goal$, such that its $w(p)$ is minimal among all such paths

Algorithms: Dijkstra O((|V|+|E|)log|V|) , Bellman-Ford O(|V|x|E|),..



Queue paths are listed head-first:
⟨d, a, s⟩ means s → a → d.

Example graph with imperfectly known information



A planning state: courses already completed.
An action: take a course whose prerequisites are met.
Unknown information: workload, future offerings, and interests.

Historical prerequisite diagram — an illustration, not current degree requirements.

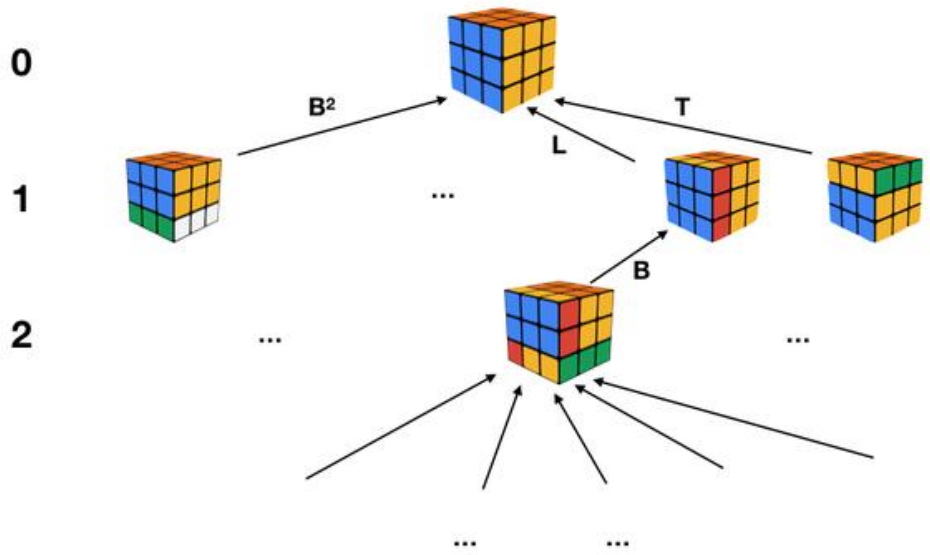
The Graph Can be Large

Solving Rubik's cube as a graph problem

Number of states or vertices
43,252,003,274,489,856,000

Yet, maximum length of a shortest path to solution was shown to be 20, regardless of the initial **state**.

T. Rokicki and collaborators, using Google's computers, proved "God's number" to be 20, in 2010.



One move = one face turn (90° or 180°).



Source: cube20.org (2010)

Search Algorithm Performance Metrics

Soundness: when a solution is returned, is it guaranteed to be **a correct path**?

Completeness: is the algorithm **guaranteed to find a solution** when one exists?

Optimality: is the found solution guaranteed to have minimum cost?

Space complexity: how much memory is needed?

Time complexity: what is the running time? Can it be used for online planning?

Uniform cost search (Uninformed search)

$Q \leftarrow \langle start \rangle$ // maintains paths **sorted by cost**

// initialize queue with start

while $Q \neq \emptyset$:

pick (and remove) the path P with lowest cost $\mathrm{w}(P)$

if $head(P) = goal$ **then return** P ; // Reached the goal

foreach $vertex\ v\ such\ that\ (head(P), v) \in E$, **do** // for all neighbors

add $\langle v, P \rangle\ to\ Q$; // Add expanded paths

return $FAILURE$; // nothing left to consider

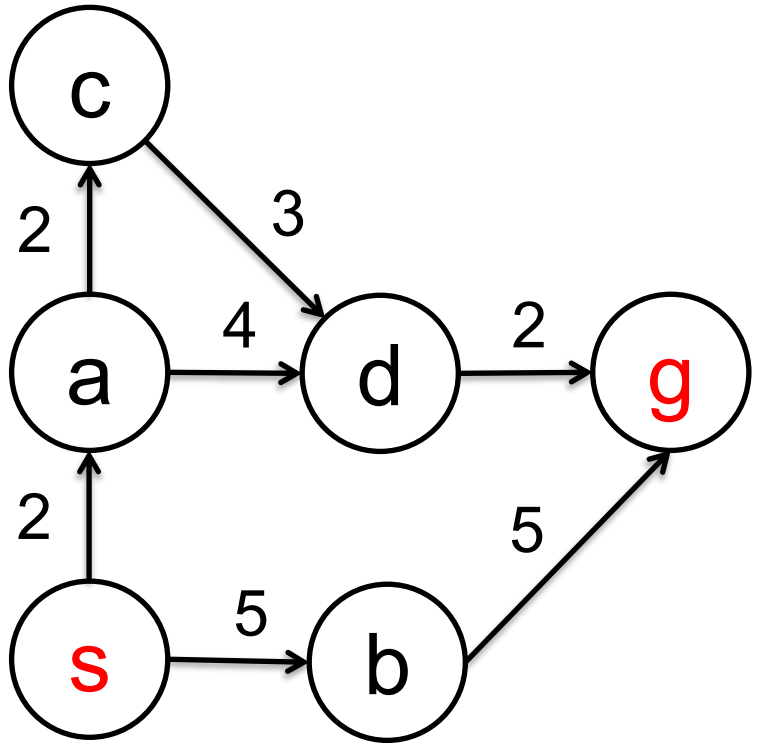
No visited list; uses path costs, but no estimate of distance to the goal.

Q: **frontier** of paths • ⟨v, P⟩: extend P to vertex v

Example of Uniform-Cost Search (s to g)

Q:

Path	Cost
$\langle s \rangle$	0

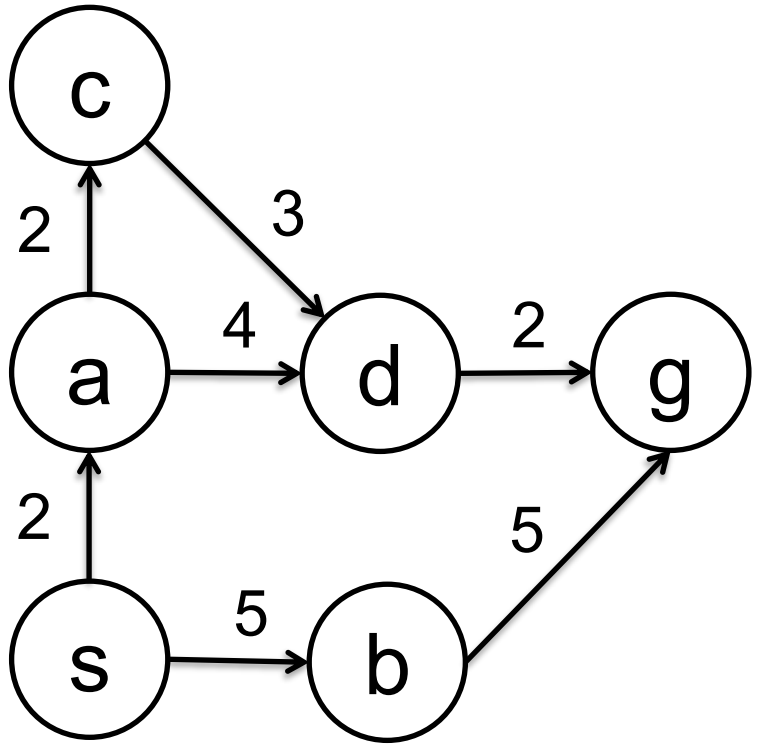


Select the path with the smallest cost.

Example of Uniform-Cost Search

Q:

Path	Cost
$\langle a, s \rangle$	2
$\langle b, s \rangle$	5



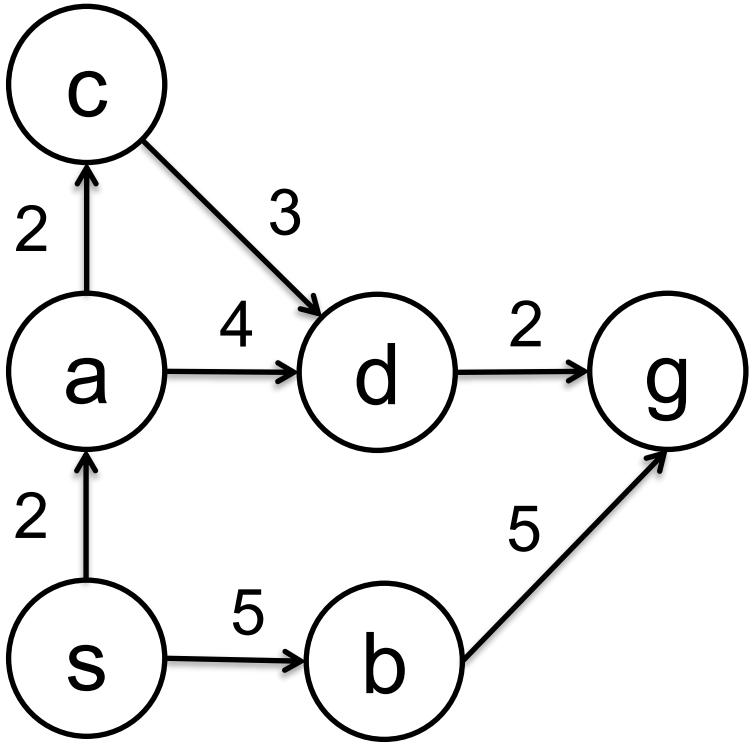
Select the path with the smallest cost.

Example of Uniform-Cost Search

Q:

Path	Cost
$\langle c, a, s \rangle$	4
$\langle b, s \rangle$	5
$\langle d, a, s \rangle$	6

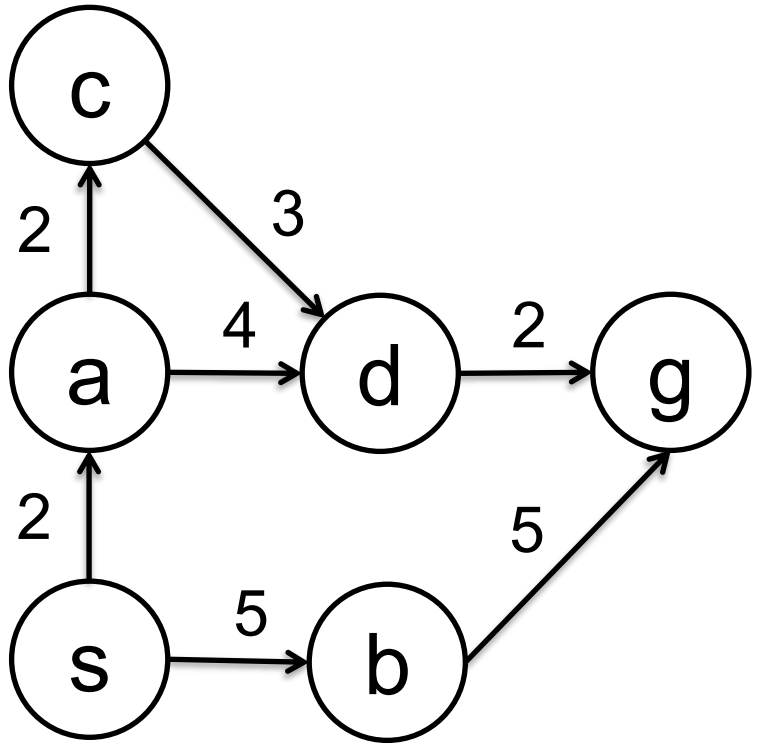
Select the path with the smallest cost.



Example of Uniform-Cost Search

Q:

Path	Cost
$\langle b, s \rangle$	5
$\langle d, a, s \rangle$	6
$\langle d, c, a, s \rangle$	7

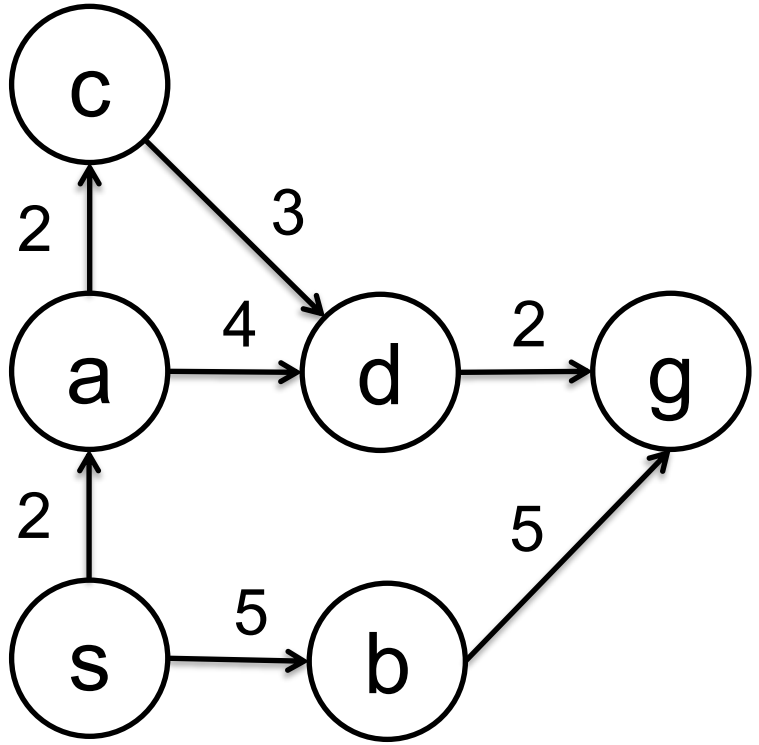


Select the path with the smallest cost.

Example of Uniform-Cost Search

Q:

Path	Cost
$\langle d, a, s \rangle$	6
$\langle d, c, a, s \rangle$	7
$\langle g, b, s \rangle$	10



A goal path is in Q. Can we stop yet?

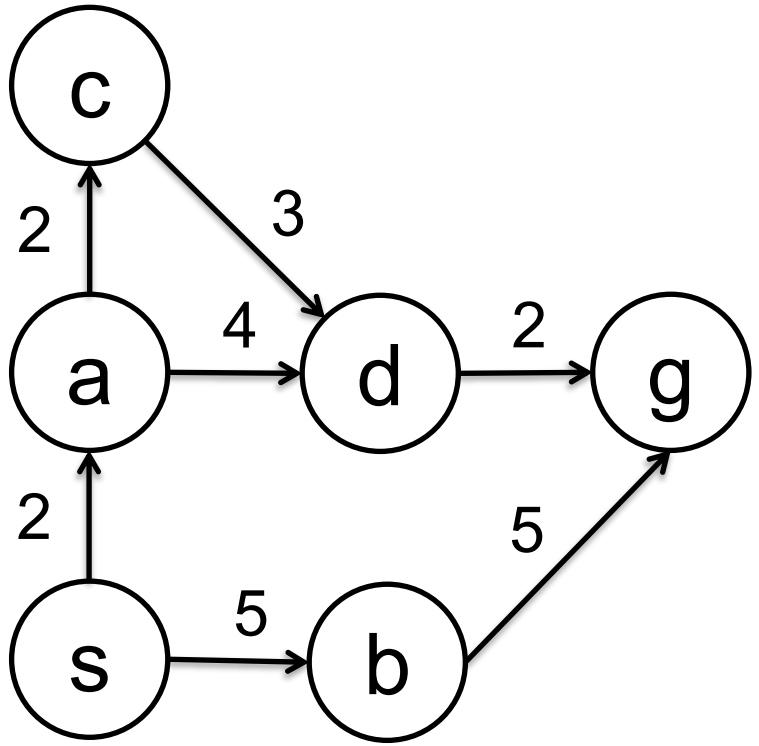
Select the path with the smallest cost.

Notice a path to goal <g, b, s> has been found but the algorithm does not terminate, since the path is currently not at the head of Q

Example of Uniform-Cost Search

Q:

Path	Cost
$\langle d, c, a, s \rangle$	7
$\langle g, d, a, s \rangle$	8
$\langle g, b, s \rangle$	10



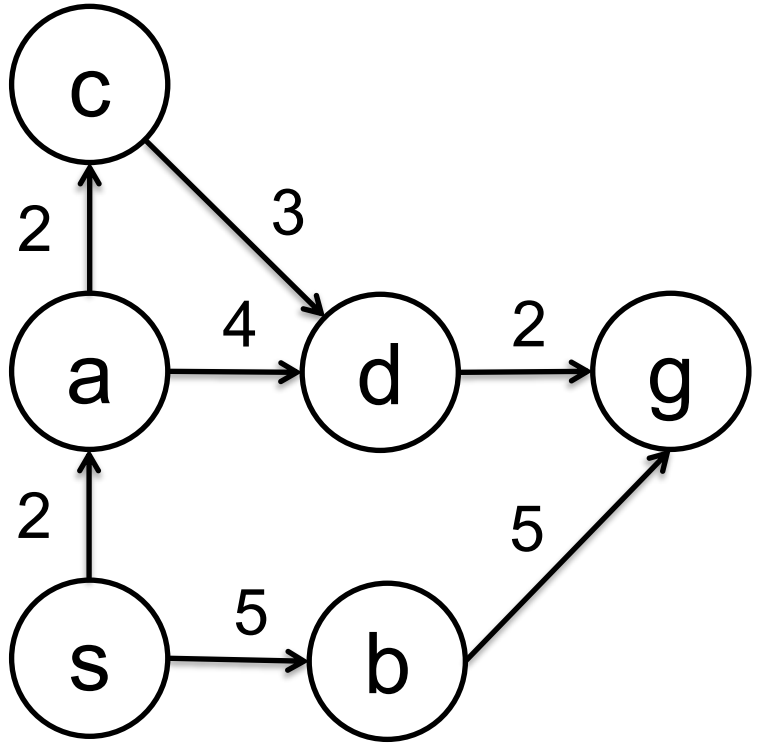
Select the path with the smallest cost.

Another path to goal <g, d, a, s> has been found but the algorithm does not terminate, since the path is currently not at the head of Q

Example of Uniform-Cost Search

Q:

Path	Cost
$\langle \boldsymbol{g}, d, a, s \rangle$	8
$\langle g, d, c, a, s \rangle$	9
$\langle g, b, s \rangle$	10



Select the path with the smallest cost.

Algorithm stops when smallest cost path in Q has head = g

Properties of Uniform Cost Search

UCS is an extension of BFS to the weighted-graph case (UCS = BFS if all edges have the same cost)

UCS is **sound, complete and optimal** on our finite graph with positive edge weights.

- Exercise: Prove the above claims
- Hint: Soundness: every entry p in Q is a path with first(p) = start.

UCS is guided by path cost rather than path depth, so it may get in trouble if some edge costs are very small

Worst-case search size: $O(b^{1+\lfloor W^*/\varepsilon \rfloor})$. b: maximum successors; W^*: optimal cost; ε: minimum edge cost.

Greedy best-first search

UCS explores paths in all directions, with no bias towards the goal state

What if we try to get “closer” to the goal?

We need a measure of distance to the goal

It would be ideal to use the length of the shortest path...

but this is exactly what we are trying to compute!

We can ***estimate*** the distance to the goal through a heuristic function

$h : V \rightarrow \mathbb{R}_{\geq 0}$. E.g., the Euclidean distance to the goal (as the crow flies)

$h(v)$ is the estimate of the distance from v to goal

A reasonable strategy is to always try to move in such a way to minimize the estimated distance to the goal: this is the basic idea of the greedy (best-first) search

Greedy best-first search

$Q \leftarrow \langle start \rangle$ *// initialize queue with start*

while $Q \neq \emptyset$:

  pick $(and\ remove)\ the\ path\ P\ with\ lowest\ heuristic\ cost\ h(head(P))\ from\ Q$

  if $head(P) = goal$ **then return** P *// Reached the goal*

  foreach $vertex\ v\ such\ that\ (head(P), v) \in E$, **do** *// for all neighbors*

    add $\langle v, P \rangle\ to\ Q$; *// Add expanded paths*

return $FAILURE$; *// nothing left to consider*

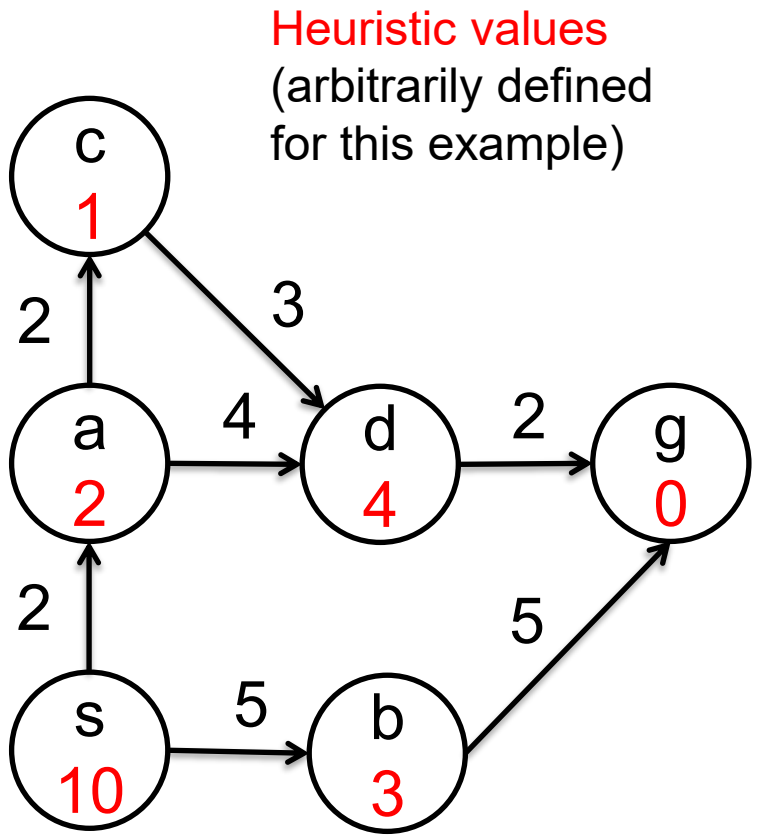
Q: frontier of paths • ⟨v, P⟩: extend P to vertex v

Example of Greedy search

Q:

Path	Cost	h
$\langle s \rangle$	0	10

Rows are not sorted. Select the smallest h.



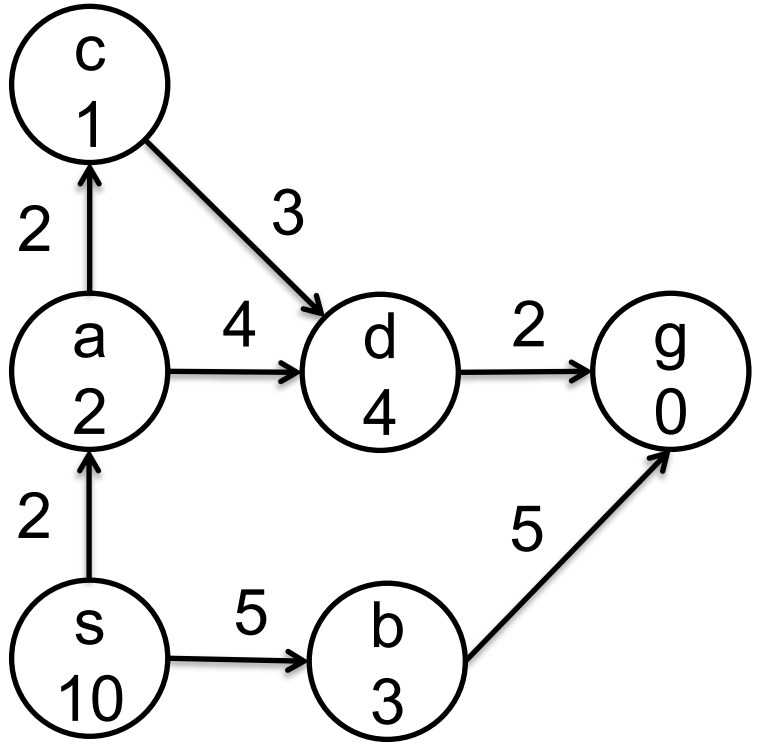
h = estimated remaining cost

Example of Greedy search

Q:

Path	Cost	h
$\langle a, s \rangle$	2	2
$\langle b, s \rangle$	5	3

Rows are not sorted. Select the smallest h.



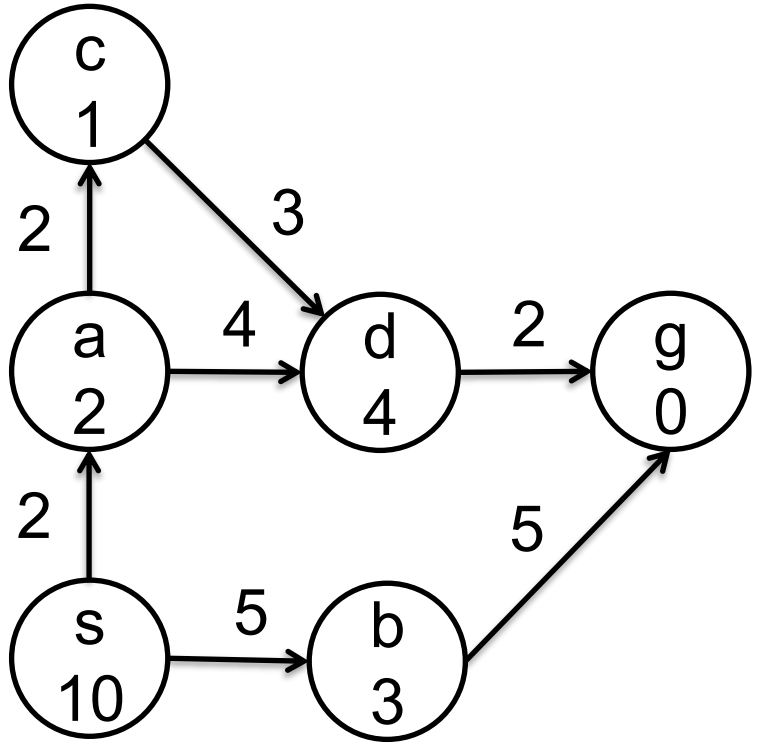
h = estimated remaining cost

Example of Greedy search

Q:

Path	Cost	h
$\langle c, a, s \rangle$	4	1
$\langle d, a, s \rangle$	6	4
$\langle b, s \rangle$	5	3

Rows are not sorted. Select the smallest h.



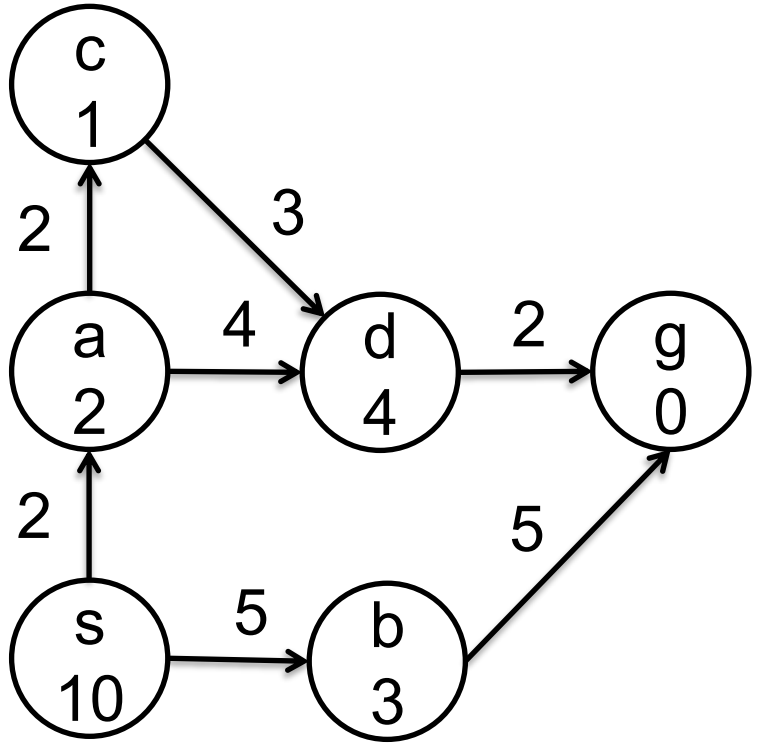
h = estimated remaining cost

Example of Greedy search

Q:

Path	Cost	h
$\langle d, c, a, s \rangle$	7	4
$\langle d, a, s \rangle$	6	4
$\langle b, s \rangle$	5	3

Rows are not sorted. Select the smallest h.



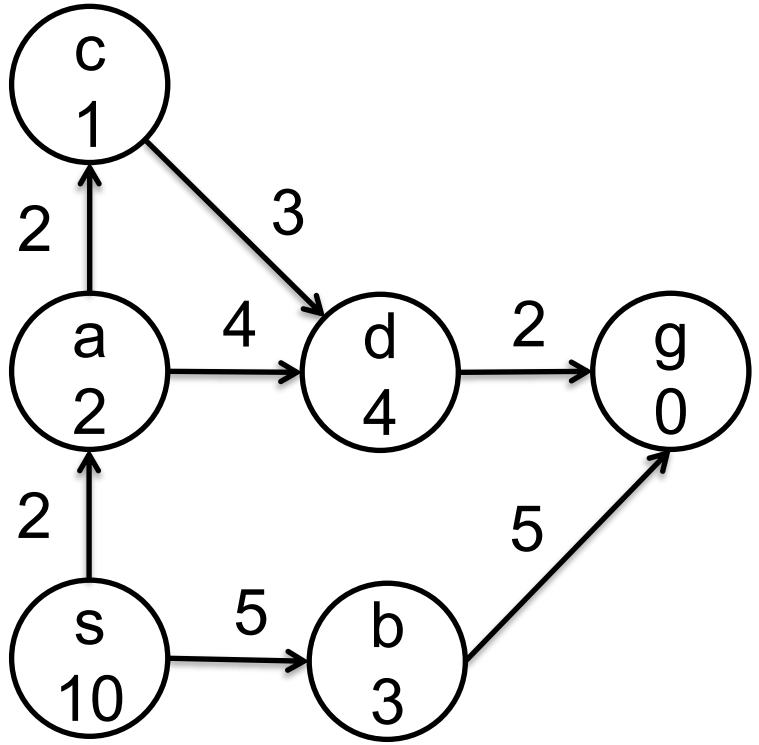
h = estimated remaining cost

Example of Greedy search

Q:

Path	Cost	h
$\langle d, c, a, s \rangle$	7	4
$\langle d, a, s \rangle$	6	4
$\langle g, b, s \rangle$	10	0

Rows are not sorted. Select the smallest h.



h = estimated remaining cost

Terminates with suboptimal path

Remarks on greedy/best-first search

Greedy best-first search selects the smallest h over the entire frontier.

- It can switch branches before reaching a dead end.

Not complete (why?) and not optimal, but is often fast and efficient, depending on the heuristic function h

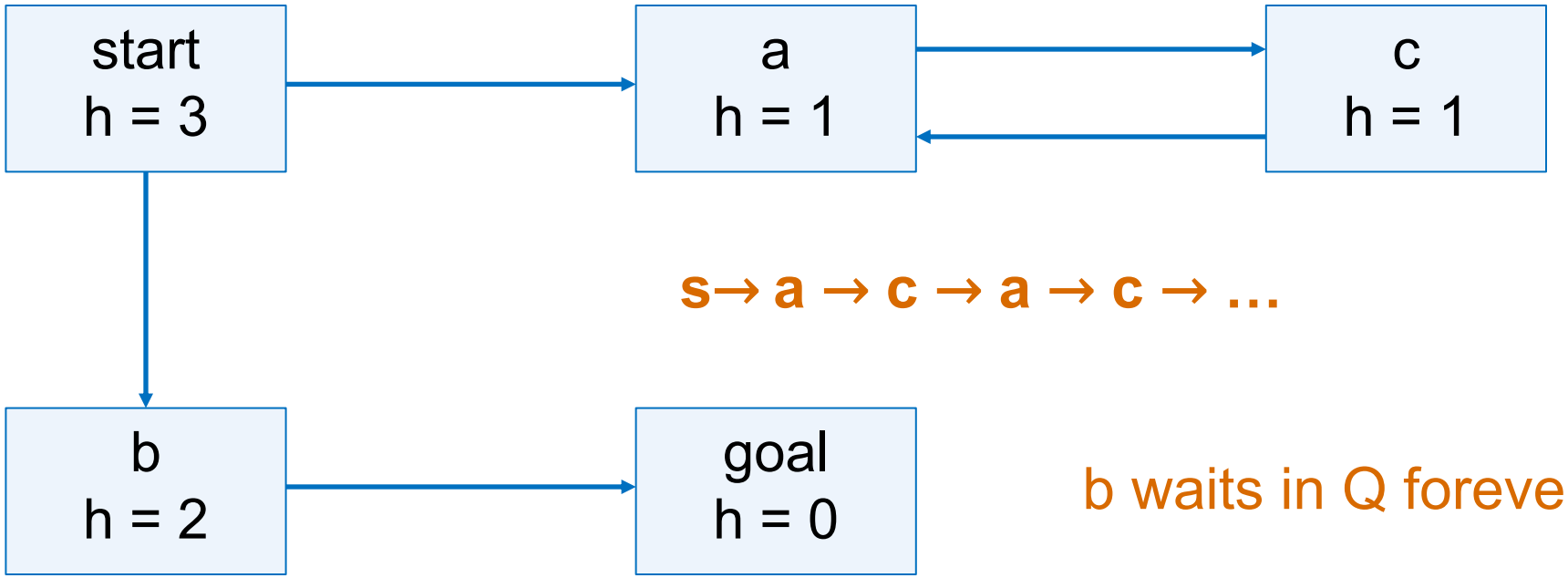
- Exercise: Find a counter-example where path exists but bad heuristic function makes the algorithm loop forever

Worst-case time and space complexity?

No visited list: cycles can make the search run forever.

Why can greedy search loop forever?

Start from s. All edge costs are 1. Inside each vertex: its heuristic h.



s→ a → c → a → c → …

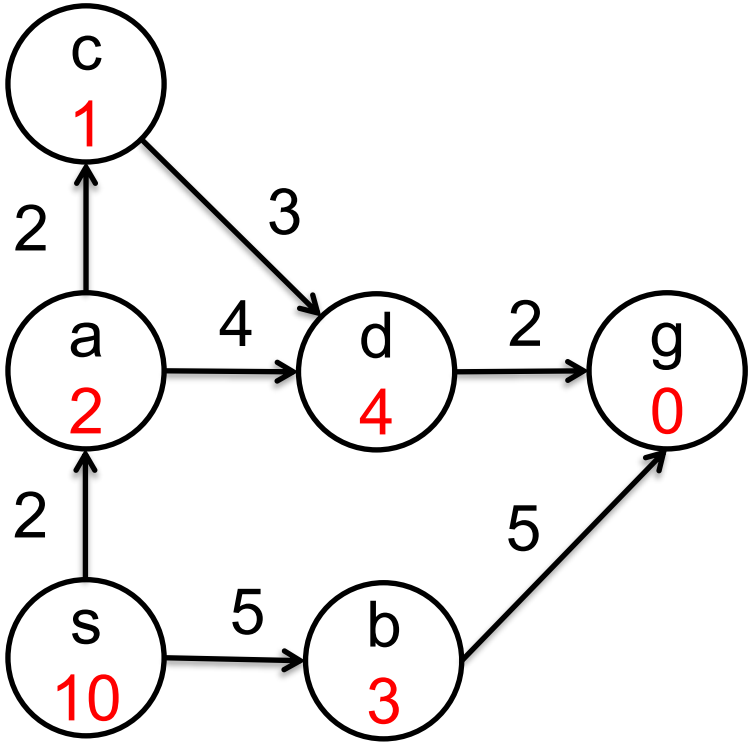
b waits in Q forever.

Which change would prevent this loop?

Summary: path cost, heuristic, or both?

Uniform-cost search
Choose smallest path cost.
Our example: s → a → d → g

Greedy best-first search
Choose smallest heuristic h.
Our example: s → b → g



Next class: A* uses $f(P) = g(P) + h(\text{head}(P))$.

g(P) = w(P): path cost so far. h: estimated remaining cost.