EE211: Robotic Perception and Intelligence

Lecture 4 Basic Search Methods

Jiankun WANG

Department of Electronic and Electrical Engineering
Southern University of Science and Technology

Undergraduate Course, Oct 2024

Outline

Outline

Elements of A Planning Problem

- State space
- Transition system
- Goal
- Constraints
- Cost or reward function

Emilio Frazzoli, Planning and Decision Making for Autonomous Robots, ETH.

State Space

- The set $\mathcal{X}$ of all situations that could possibly arise
- The state $x \in \mathcal{X}$ is a description of one of these situations
  - 2D Position and heading of a car
  - 3D position, attitude, linear and angular velocity of a drone
- World state, state of the world, the state of everything else besides what we can control directly
  - States of other robots
  - Sates of environmental features

Transition System

- Set of states $\mathcal{S}$
- Set of actions $\mathcal{A}$
- Transition relation $s_1 \xrightarrow{a} s_2$
- Dynamic system, $\frac{dx(t)}{dt} = F(x(t), u(t))$ given a control input $u(t)$

Remark

Relation: (discrete) transition systems are typically used algorithmically for computing plans that are executed on (continuous) dynamical systems!

Goal, Constraints & Cost

- Initial state
  - State of the system at the beginning of the plan
- Goal state
  - A set of states that the system can reach at the end of the plan
- Constraints
  - Obstacle avoidance, velocity limitation...
- Cost or reward function
  - Trajectory length, time cost, energy consumption...

Concept of The Shortest Path Problem

- Given
  - State space $\mathcal{X}$, including free space $\mathcal{X}_{free}$ and obstacle space $\mathcal{X}_{obs}$
  - an initial state s_0
  - a set of goal states $\mathcal{S}_{goal} = s_{g1}, s_{g2}, \ldots$
  - a transition system that determine $s_1 \xrightarrow{a} s_2$
- Find

$$
\begin{aligned}
\sigma^* = & \arg\min_{\sigma \in \Sigma} \; c(\sigma) \\
s.t.\; & \sigma(0) = s_0, \\
& \sigma(T) \in \mathcal{S}_{goal}, \\
& \sigma(t) \in \mathcal{X}_{free}.
\end{aligned} \tag{1}
$$

Properties of Search Algorithms

- Completeness: A search algorithm is complete if it gives a solution if one exists or returns failure in finite time.
- Optimality: If a valid solution is the best (lowest cost) among all generated solutions, then that solution is optimal.
- Time complexity: Time cost (or number of steps) to complete a task, as a function of input size.
- Space complexity: Maximum storage or memory to complete a task, as a function of input size.

A Framework of Search Algorithms

Algorithm 1: Forward search algorithm

```
Q ← {s_0};                                          // Initialize the queue
V ← {s_0};                                    // Initialize the visited set
Parent(s_0) ← null;
while Q is not empty do
    Take the first element s from Q;
    if s ∈ S_goal then
        return σ;                                  // A valid path is found
    for all s, s' such that s --a--> s' do
        if s' ∉ V then
            insert s' into Q;                       // Add new states to Q
            add s' to V;                     // ...and mark them as visited
            Parent(s') ← s;               // Reconstruct the path backward
return failure;                                    // No valid path exists
```

DFS & BFS

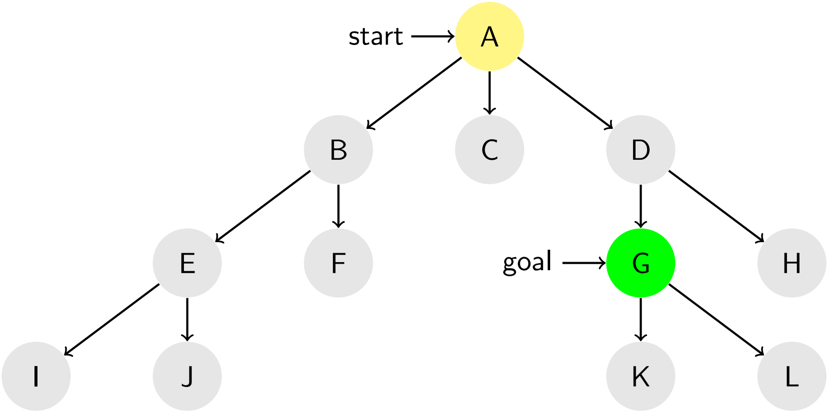
Depth-First Search

In Depth-First Search, new states are added at the front of the queue.

Breadth-First Search

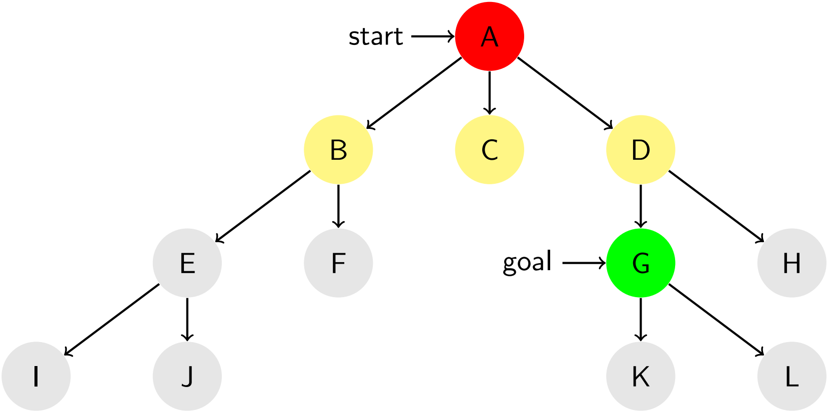
In Breadth-First Search, new states are added at the back of the queue.

Depth-First Search: Step 1



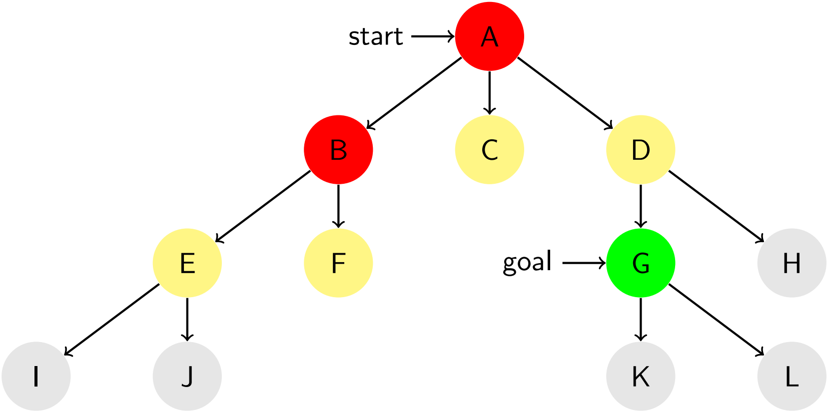
- $Q = \{A\}$
- $V = \{A\}$

Depth-First Search: Step 2



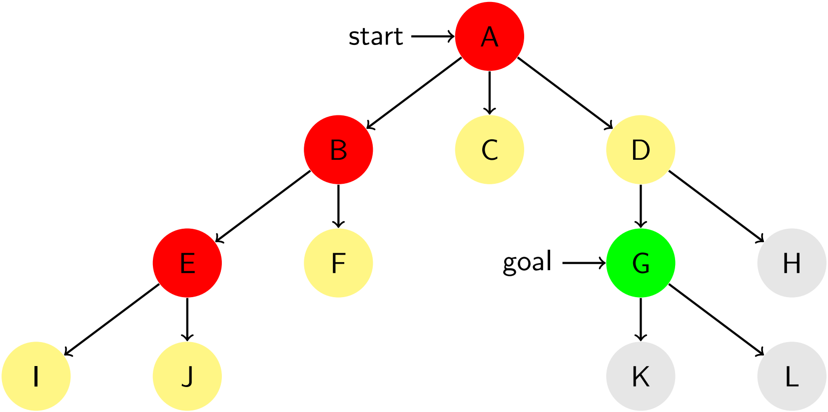
- $Q = \{B, C, D\}$
- $V = \{A, B, C, D\}$

Depth-First Search: Step 3



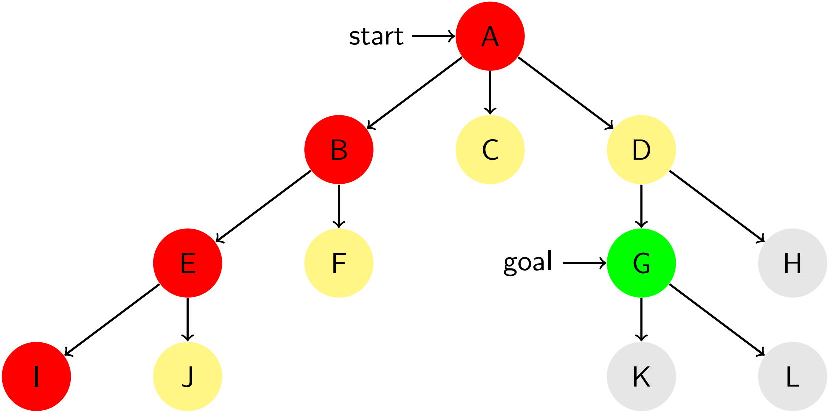
- $Q = \{E, F, C, D\}$
- $V = \{A, B, C, D, E, F\}$

Depth-First Search: Step 4



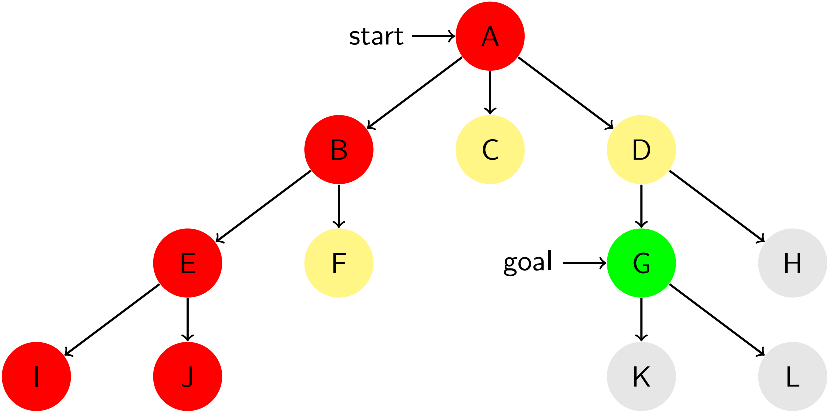
- $Q = \{I, J, F, C, D\}$
- $V = \{A, B, C, D, E, F, I, J\}$

Depth-First Search: Step 5



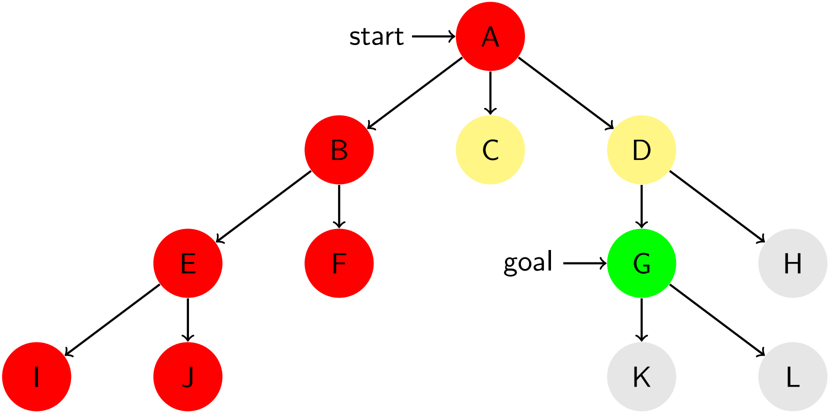
- $Q = \{J, F, C, D\}$
- $V = \{A, B, C, D, E, F, I, J\}$

Depth-First Search: Step 6



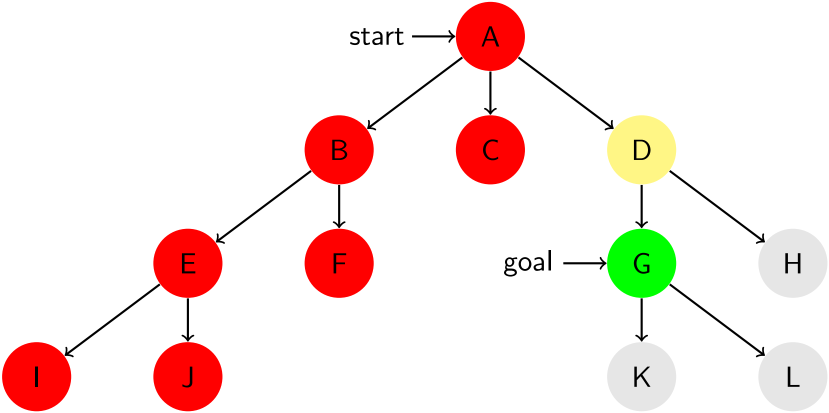
- $Q = \{F, C, D\}$
- $V = \{A, B, C, D, E, F, I, J\}$

Depth-First Search: Step 7



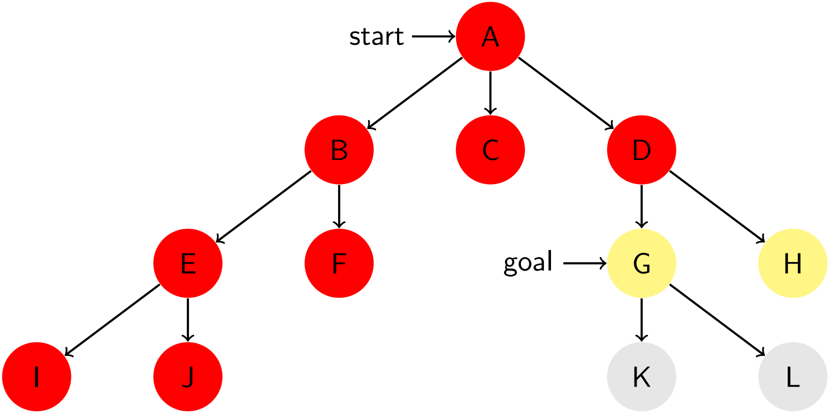
- $Q = \{C, D\}$
- $V = \{A, B, C, D, E, F, I, J\}$

Depth-First Search: Step 8



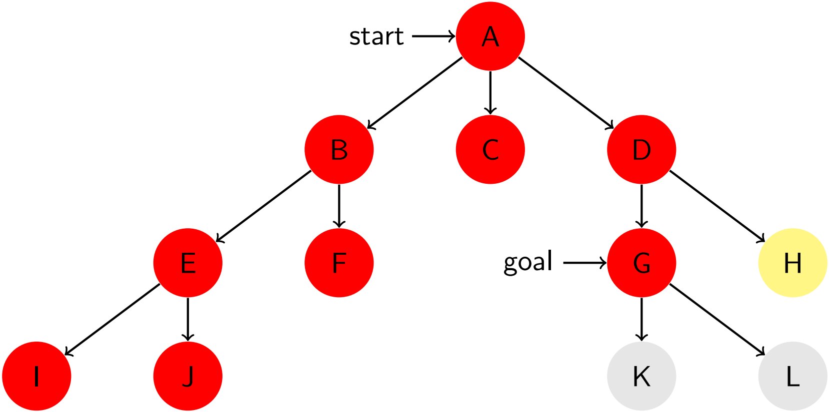
- $Q = \{D\}$
- $V = \{A, B, C, D, E, F, I, J\}$

Depth-First Search: Step 9



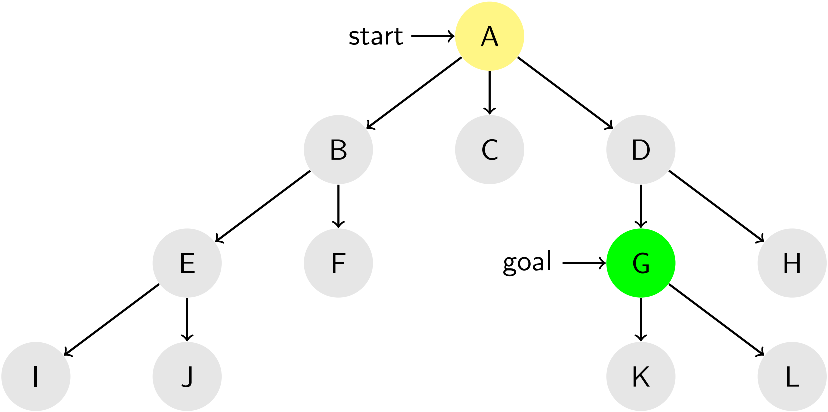
- $Q = \{G, H\}$
- $V = \{A, B, C, D, E, F, I, J, G, H\}$

Depth-First Search: Step 10



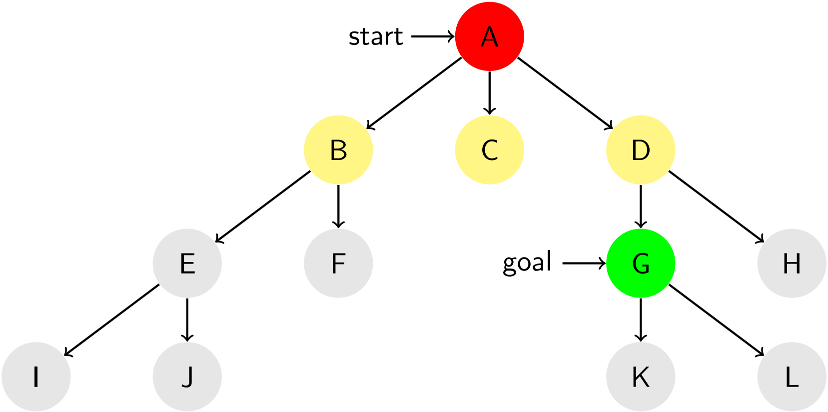
- $Q = \{H\}$
- $V = \{A, B, C, D, E, F, I, J, G, H\}$
- Return $\{A, D, G\}$

Breadth-First Search: Step 1



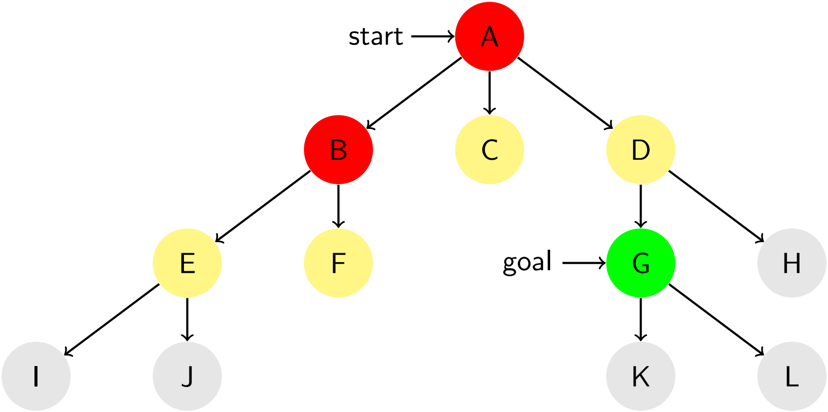
- $Q = \{A\}$
- $V = \{A\}$

Breadth-First Search: Step 2



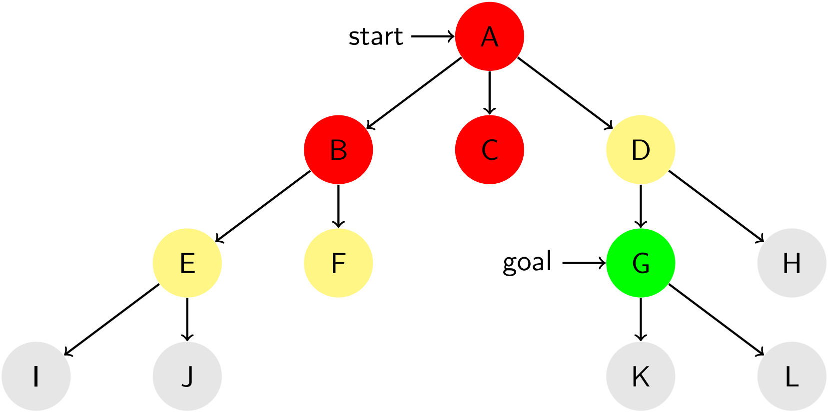
- $Q = \{B, C, D\}$
- $V = \{A, B, C, D\}$

Breadth-First Search: Step 3



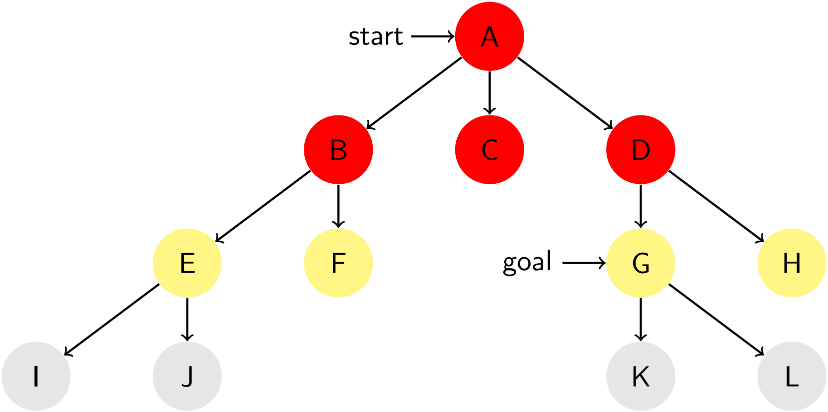
- $Q = \{C, D, E, F\}$
- $V = \{A, B, C, D, E, F\}$

Breadth-First Search: Step 4



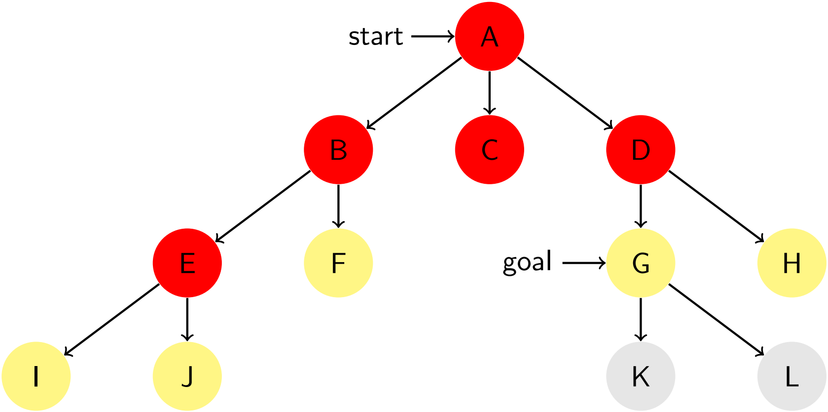
- $Q = \{D, E, F\}$
- $V = \{A, B, C, D, E, F\}$

Breadth-First Search: Step 5



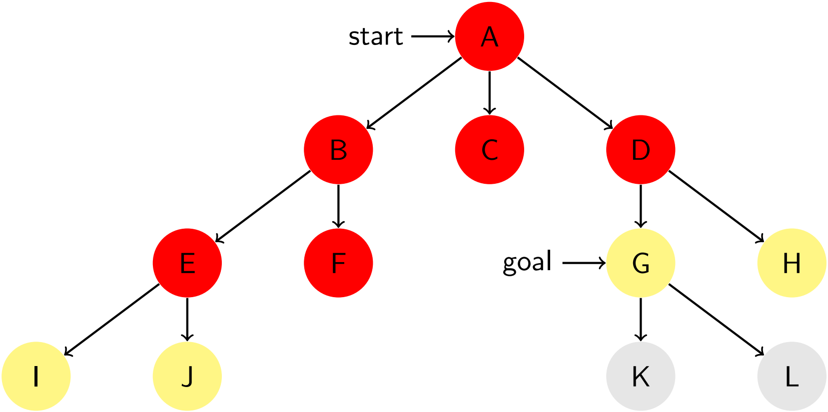
- $Q = \{E, F, G, H\}$
- $V = \{A, B, C, D, E, F, G, H\}$

Breadth-First Search: Step 6



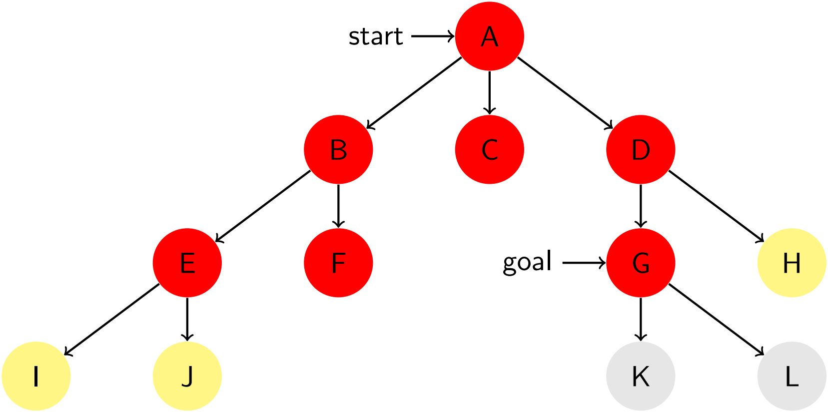
- $Q = \{F, G, H, I, J\}$
- $V = \{A, B, C, D, E, F, G, H, I, J\}$

Breadth-First Search: Step 7



- $Q = \{G, H, I, J\}$
- $V = \{A, B, C, D, E, F, G, H, I, J\}$

Breadth-First Search: Step 8



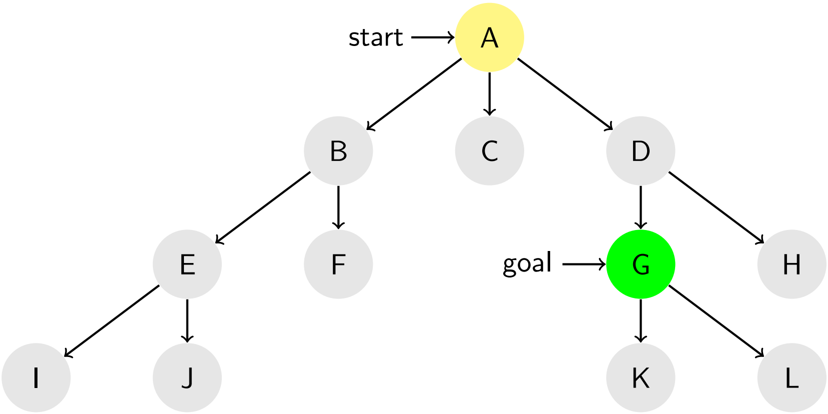
- $Q = \{H, I, J\}$
- $V = \{A, B, C, D, E, F, G, H, I, J\}$
- Return $\{A, D, G\}$

Properties of DFS & BFS

- Completeness
  - BFS is complete on finite or countably infinite transition systems
  - DFS is complete only on finite transition systems
- Time complexity
  - Proportional to the number of visited nodes
- Space complexity
  - Proportional to the max size of the priority queue

Worst-case Complexity of BFS & DFS

- Branching factor b=3, maximum depth m=4, minimum goal depth d=3



Iterative Deepening

Algorithm 2: Iterative Deepening

$d \leftarrow 1$;
while $d \leq m$ **do**
    Run DFS up to depth d;
    if *a path* σ *is found* **then**
        return σ;
    $d \leftarrow d + 1$;
return *failure*;

- Branching factor b=3, maximum depth m=4, minimum goal depth d=3
- Explore the graph in breadth-first order, using depth-first search.

Concept of The Shortest Path Problem

- Given
  - State space $\mathcal{X}$, including free space $\mathcal{X}_{free}$ and obstacle space $\mathcal{X}_{obs}$
  - an initial state s_0
  - a set of goal states $\mathcal{S}_{goal} = s_{g1}, s_{g2}, \ldots$
  - a transition system that determine $s_1 \xrightarrow{a} s_2$
- Find

$$
\begin{aligned}
\sigma^* = \arg\min_{\sigma \in \Sigma} \; & c(\sigma) \\
s.t.\ \sigma(0) &= s_0, \\
\sigma(T) &\in \mathcal{S}_{goal}, \\
\sigma(t) &\in \mathcal{X}_{free}.
\end{aligned} \tag{2}
$$

Concept of The Shortest Path Problem

- $c(\sigma) := \sum_{i=1}^{n} w(s_{i-1}, a_i, s_i)$
- State transitions on a transition system, or edges in a graph, are often abstractions of physical motions
- We know or can estimate in advance what the cost of a particular transition is

Outline

Concept of Uniform-Cost Search

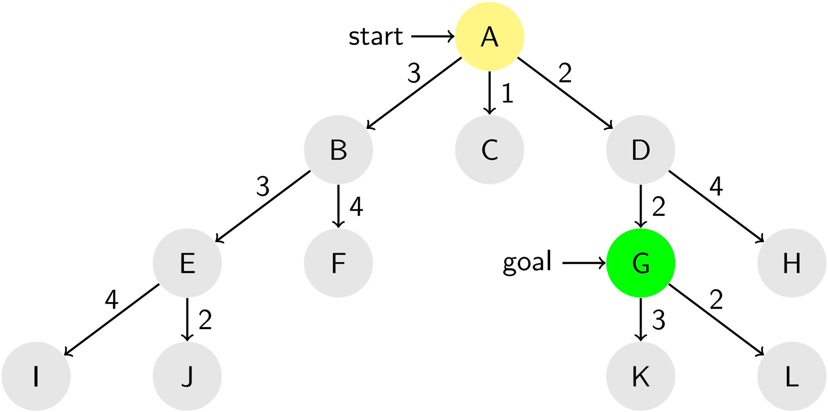
- BFS can find the "minimum depth" path (Recall: In Breadth-First Search, new states are added at the back of the queue.)
- Idea: Use "cost" instead of "depth" when sorting nodes in the queue
- Keep track of the "costToCome" of each visited state, and its Parent (The costToCome of unvisited states is implicitly initialized to $+\infty$)

Uniform-Cost Search

Algorithm 3: Uniform-Cost Search

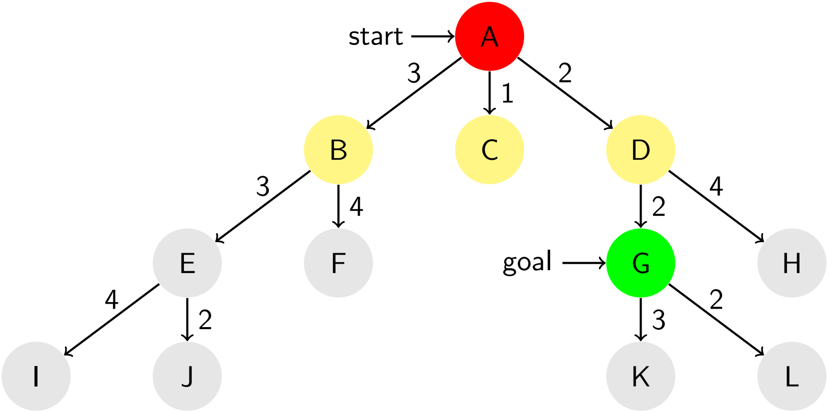
```
Q ← {s_0};
costToCome = 0;
Parent(s_0) ← null;
while Q is not empty do
    Take the minimum costToCome element s from Q;
    if s ∈ S_goal then
        return σ;
    for all s, s' such that s --a--> s' do
        newCostToCome ← costToCome + w(s, a, s');
        if newCostToCome < costToCome(s') then
            costToCome(s') ← newCostToCome;
            Parent(s') ← s;
            update s' in Q;
return failure;
```

Uniform-Cost Search: Step 1



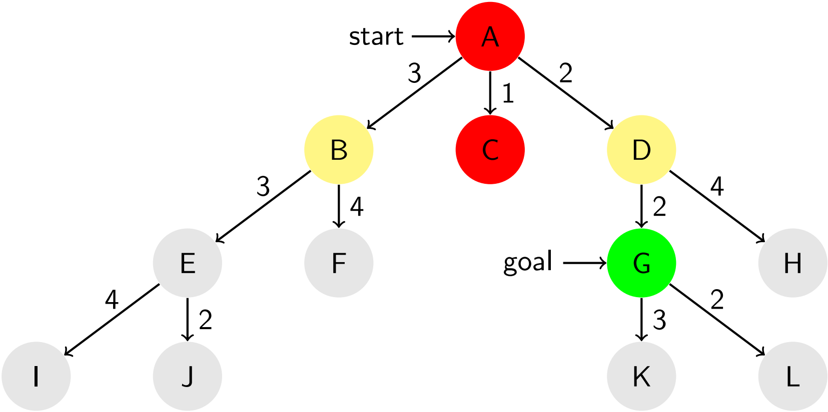
- $Q = \{A\}$
- $\text{costToCome} = \{A : 0\}$

Uniform-Cost Search: Step 2



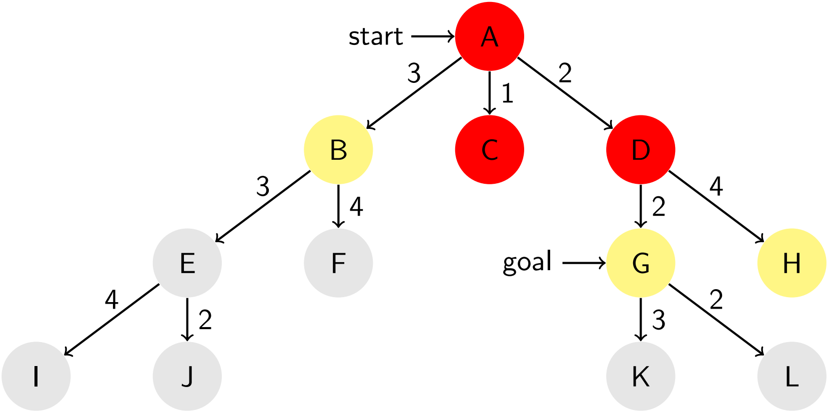
- $Q = \{B, C, D\}$
- costToCome $= \{A : 0, C : 1; D : 2; B : 3\}$

Uniform-Cost Search: Step 3



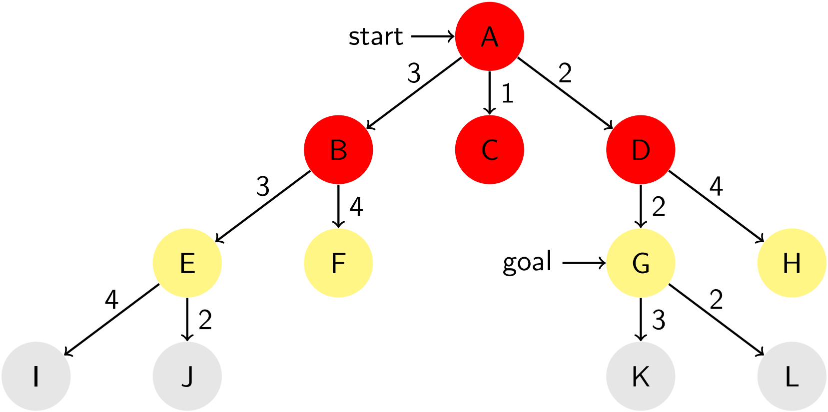
- $Q = \{B, D\}$
- $\text{costToCome} = \{A : 0, C : 1; D : 2; B : 3\}$

Uniform-Cost Search: Step 4



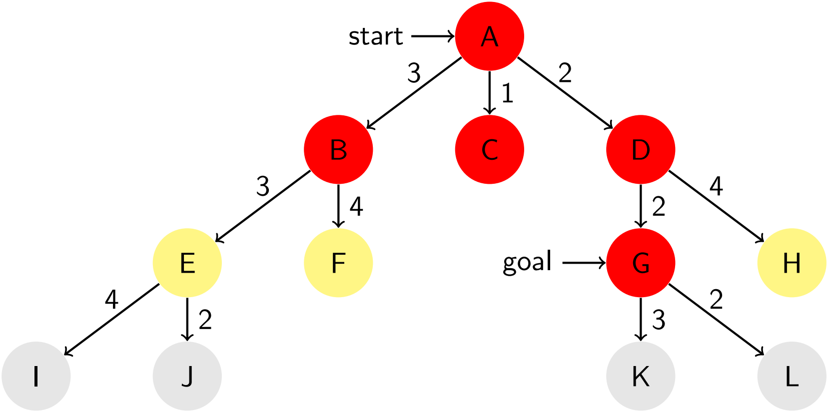
- $Q = \{B, G, H\}$
- costToCome $= \{A : 0, C : 1; D : 2; B : 3, G : 4, H : 6\}$

Uniform-Cost Search: Step 5



- $Q = \{G, H, E, F\}$
- costToCome $= \{A : 0, C : 1; D : 2; B : 3, G : 4, H : 6, E : 6, F : 7\}$

Uniform-Cost Search: Step 5



- $Q = \{H, E, F\}$
- costToCome $= \{A : 0, C : 1; D : 2; B : 3, G : 4, H : 6, E : 6, F : 7\}$
- Reruen {A, D, G}

Uniform-Cost Search

- Extension of BFS to the weighted graph case
- Complete
- Guided by path cost rather than path depth
- Optimal (How about BFS & DFS?)

Uniform-Cost Search

- Extension of BFS to the weighted graph case
- Complete
- Guided by path cost rather than path depth
- Optimal (How about BFS & DFS?)
- For finding the shortest path, BFS is optimal (only to the unweighted graph case) but DFS is not optimal (Such as Cycle Path)

Outline

Greedy (Best-First) Search

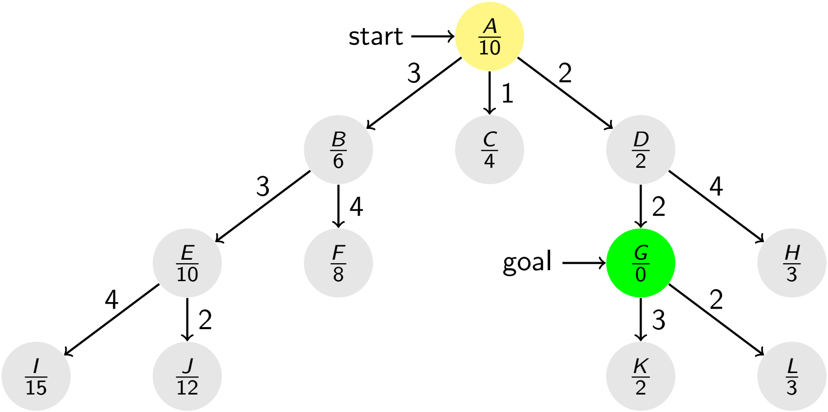
- BFS can find the “minimum-depth” path
- UCS can find the “minimum cost” path
- Forward exploration in all directions
- What if we can get information from the goal
- Heuristic function: minimize the estimated distance to the goal

Greedy (Best-First) Search

Algorithm 4: Greedy (Best-First) Search

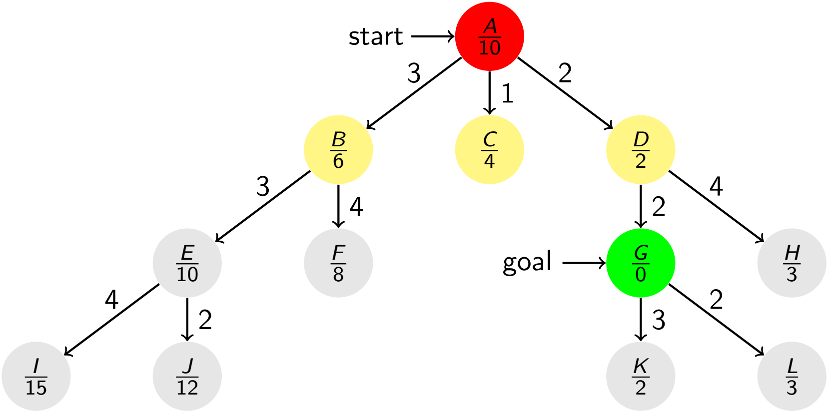
$Q \leftarrow \{s_0\}$;
costToCome $= 0$;
Parent(s_0) $\leftarrow$ *null*;
while Q *is not empty* **do**
  Take the minimum heuristic cost element s from Q;
  if $s \in \mathcal{S}_{goal}$ **then**
    return σ;
  for *all* s, s' *such that* $s \xrightarrow{a} s'$ **do**
    if $s' \notin V$ **then**
      insert s' into Q;
      add s' to V;
      Parent(s') $\leftarrow s$;
return *failure*;

Greedy (Best-First) Search: Step 1



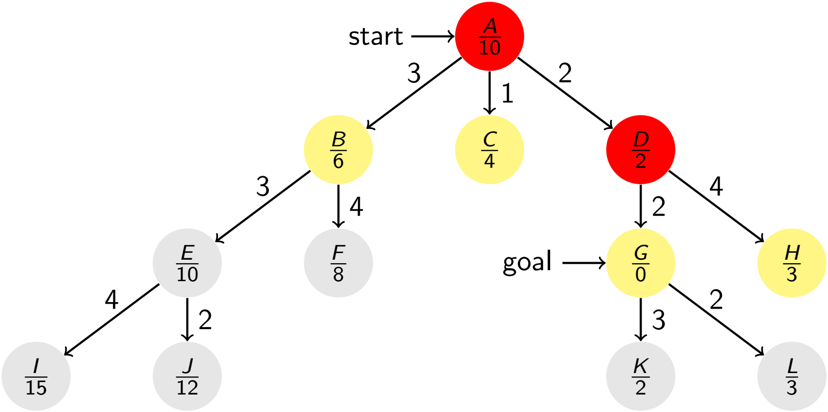
- $Q = \{A\}$
- $V = \{A\}$

Greedy (Best-First) Search: Step 2



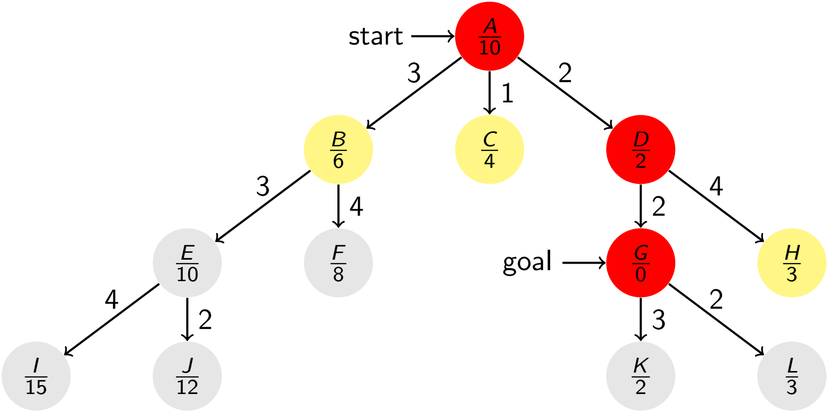
- $Q = \{B:6, C:4, D:2\}$
- costToCome $= \{A, B, C, D\}$

Greedy (Best-First) Search: Step 3



- $Q = \{B : 6, C : 4, G : 0, H : 3\}$
- $\text{costToCome} = \{A, B, C, D, G, H\}$

Greedy (Best-First) Search: Step 4



- $Q = \{B : 6, C : 4, H : 3\}$
- costToCome $= \{A, B, C, D, G, H\}$
- Return $\{A, D, G\}$

Greedy (Best-First) Search

- Similar to DFS, keep exploring until a dead end
- DFS $\rightarrow$ Greedy, depth $\rightarrow$ heuristic function
- BFS $\rightarrow$ UCS, depth $\rightarrow$ uniform cost
- We know DFS + BFS $\rightarrow$ Iterative Deepening
- Greedy Search + UCS $\rightarrow$ **?**

Outline

A Search

- UCS is optimal, but not efficient
- Greedy search is not optimal, but sometimes efficient
- Idea: Utilize the cost from the start to a state, $c(s)$, and the heuristic function that estimates the cost from s to the goal, $h(s)$

$$f(s) = c(s) + h(s)$$

A Search

Algorithm 5: A Search

$Q \leftarrow \{s_0\}$;
$c(s_0) = 0$;
Parent(s_0) $\leftarrow$ *null*;
while *Q is not empty* **do**
    Take the minimum $f(s)$ element s from Q;
    if $s \in \mathcal{S}_{goal}$ **then**
        return σ;
    for *all s, s' such that $s \xrightarrow{a} s'$* **do**
        newCostToCome $\leftarrow c(s) + w(s, a, s')$;
        if *newCostToCome* $< c(s')$ **then**
            $c(s') \leftarrow$ newCostToCome;
            Parent(s') $\leftarrow s$;
            update s' in Q;
return *failure*;

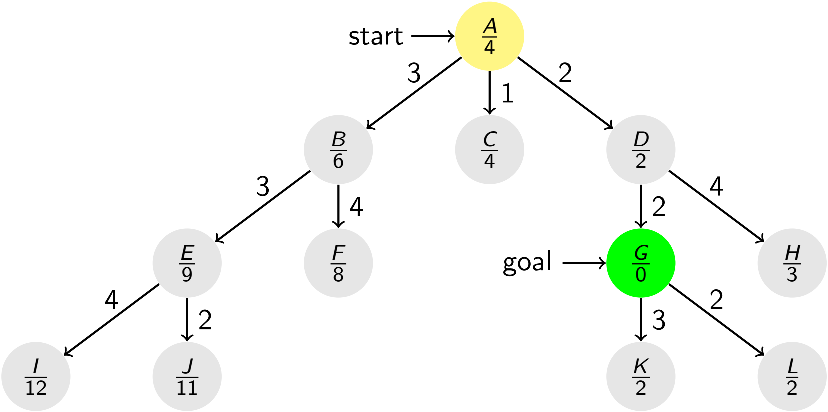
A* Search

- A search is complete, but not optimal
  - $h = 0$, same as UCS
  - h is too large for some “good” states, then it steers the search away
  - balance: h is informative, but not misleading
- Idea: Choose an admissible heuristic, such that $h(s) \leq h^*(s)$ for all states s, where $h^*(s)$ is the “true” optimal cost from s to the goal

A*

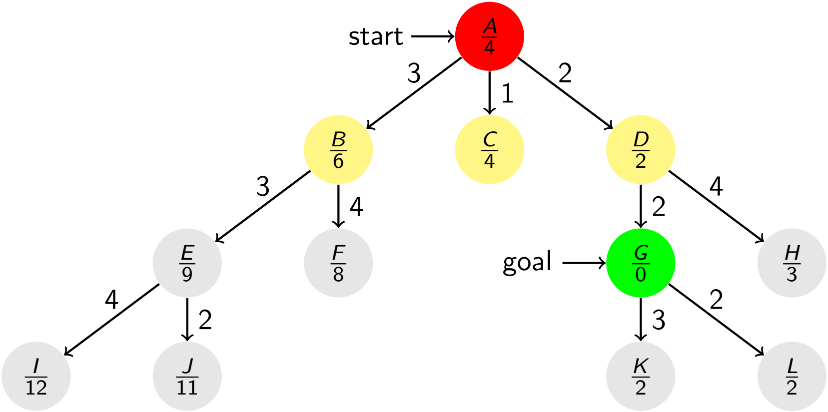
The A search with an admissible heuristic is called A*, and is optimal.

A* Search: Step 1



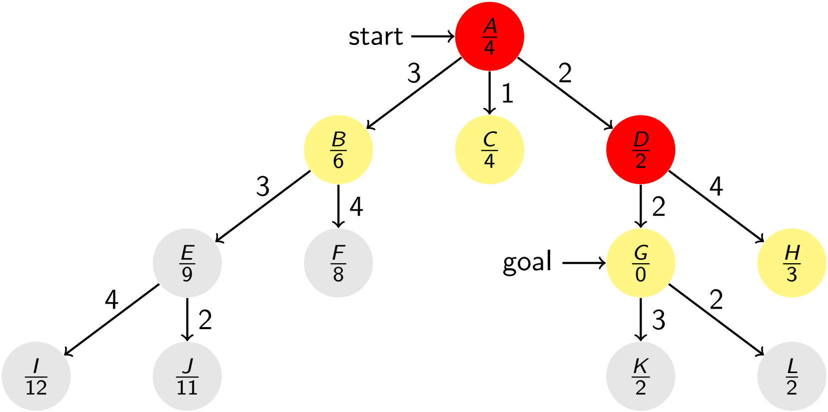
- $Q = \{A(0+4)\}$
- $\text{costToCome} = \{A : 0\}$

A* Search: Step 2



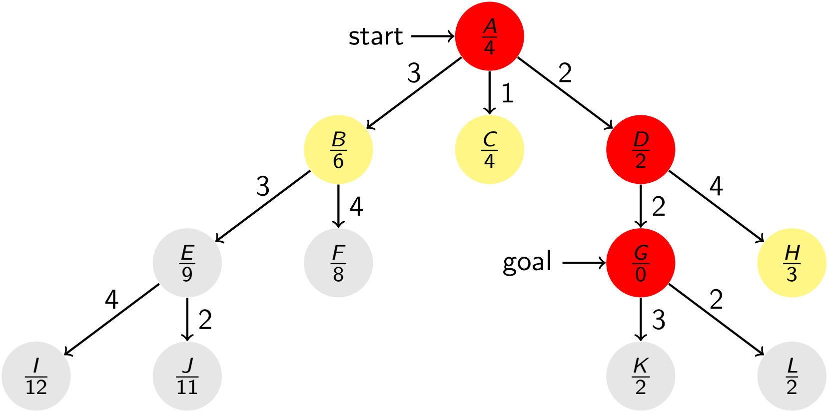
- $Q = \{B(3+6), C(1+4), D(2+2)\}$
- $\text{costToCome} = \{A:0, B:3, C:1, D:2\}$

A* Search: Step 3



- $Q = \{B(3+6), C(1+4), G(4+0), H(6+3)\}$
- $\mathrm{costToCome} = \{A:0, B:3, C:1, D:2, G:4, H:6\}$

A* Search: Step 4



- $Q = \{B(3+6), C(1+4), H(6+3)\}$
- costToCome $= \{A:0, B:3, C:1, D:2, G:4, H:6\}$
- Return $\{A, D, G\}$

Proof of A* Optimality

1. Assume that A* returns a path σ, but $cost(\sigma) > cost(\sigma^*)$
2. Find the first state on the optimal path σ^* but not on σ, call it s
3. $f(s) > cost(\sigma)$, otherwise we would have included s in σ
4. $f(s) = c(s) + h(s)$ by definition
5. $= c^*(s) + h(s)$ because s is on the optimal path
6. $\leq c^*(s) + h^*(s)$ because h is admissible
7. $= f^*(s) = cost(\sigma^*)$
8. Hence $cost(\sigma^*) \geq f(s) > cost(\sigma)$, which is a contradiction

Admissible Heuristics

- A heuristic that never overestimates the costToGo
- $h = 0$, always works, but not informative
- $h = distance(v, g)$, when the vertices are physical locations
- $h = ||v - g||_p$, when the vertices are points in a normed vector space

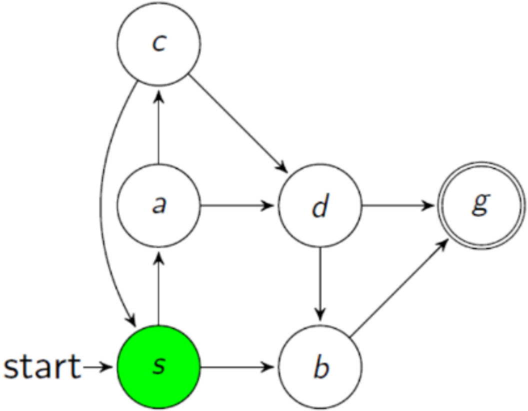
Consistent Heuristics

- Consistency (triangle inequality): $\forall s \xrightarrow{a} s', h(s) \leq w(s, a, s') + h(s')$
- $f(s) = c(s) + h(s)$ is non-decreasing along paths

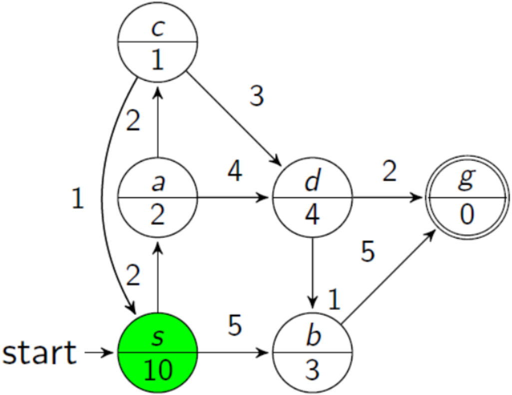
$$f(s') = c(s') + h(s') = c(s) + w(s, a, s') + h(s') \geq c(s) + h(s) = f(s)$$

- The first path found to a state is also the optimal path

Exercises



Exercises



8-puzzle Problem

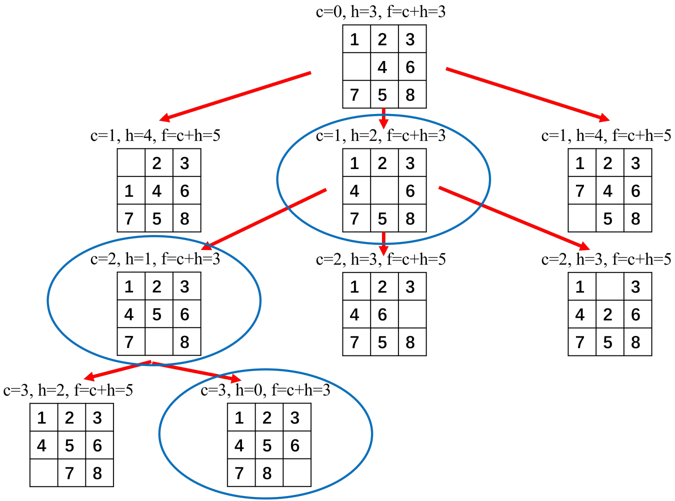
1	2	3
	4	6
7	5	8

Initial State

1	2	3
4	5	6
7	8	

Goal State

8-puzzle Problem



Q & A

1. Shortest Path Problems
2. Uniform-Cost Search
3. Greedy Search
4. Optimal Search