# Informed Problem Solving and Search

Presented by Yasin Ceran

September 24, 2024

# Learning Objectives

At the end of the class you should be able to:

- devise an useful heuristic function for a problem
- demonstrate how best-first and $A^*$ search will work on a graph
- predict the space and time requirements for best-first and $A^*$ search

# Graph Search Algorithm

**Input:** a graph,
a set of start nodes,
Boolean procedure $goal(n)$ that tests if $n$ is a goal node.

$frontier := \{\langle s \rangle : s \text{ is a start node}\}$
**while** *frontier* is not empty:
  **select** and **remove** path $\langle n_0, \ldots, n_k \rangle$ from *frontier*
  **if** $goal(n_k)$
    **return** $\langle n_0, \ldots, n_k \rangle$
  **for every** neighbor $n$ of $n_k$
    **add** $\langle n_0, \ldots, n_k, n \rangle$ to *frontier*
**end while**

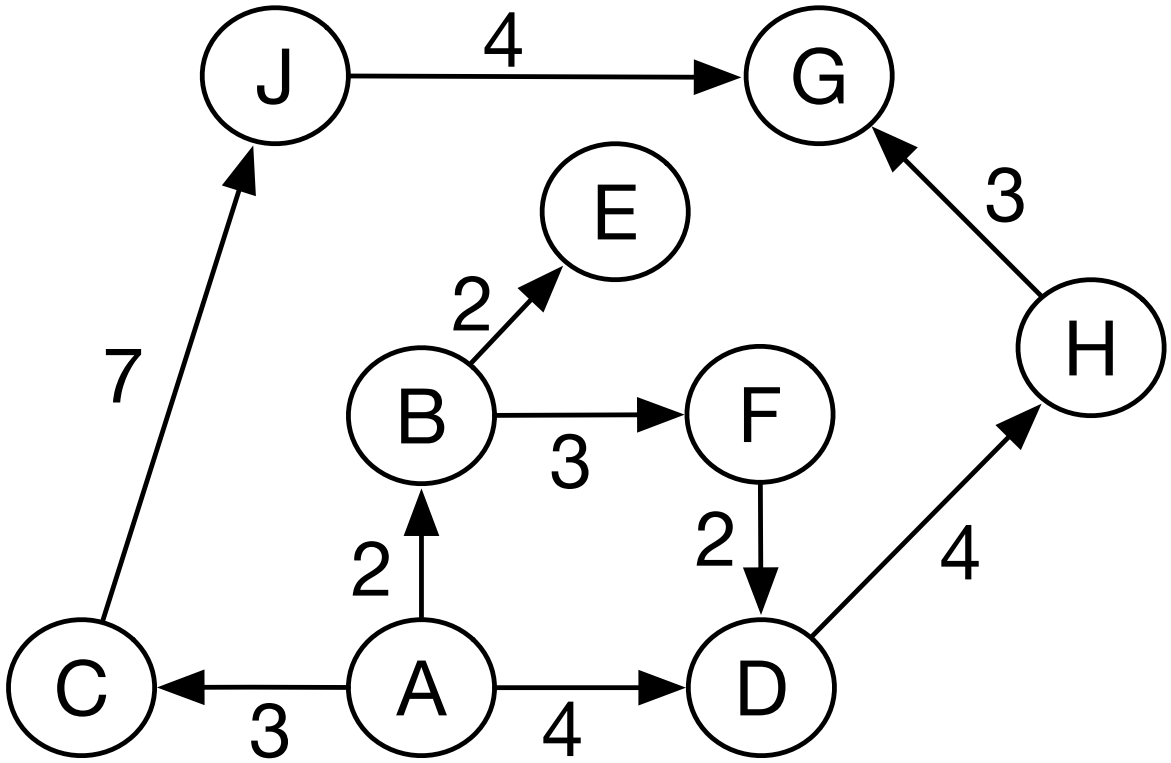
# Heuristic Search

- Idea: don't ignore the goal when selecting paths.
- Often there is extra knowledge that can be used to guide the search: heuristics.
- A heuristic function $h(n)$ is a nonnegative estimate of the cost of the least-cost path from node $n$ to a goal node.
- $h(n)$ needs to be efficient to compute.
- $h$ can be extended to paths: $h(\langle n_0, \ldots, n_k \rangle) = h(n_k)$.
- $h(n)$ is an underestimate if there is no path from $n$ to a goal with cost less than $h(n)$.
- An admissible heuristic is a heuristic function that is an underestimate of the actual cost of a path to a goal.

# Example Heuristic Functions

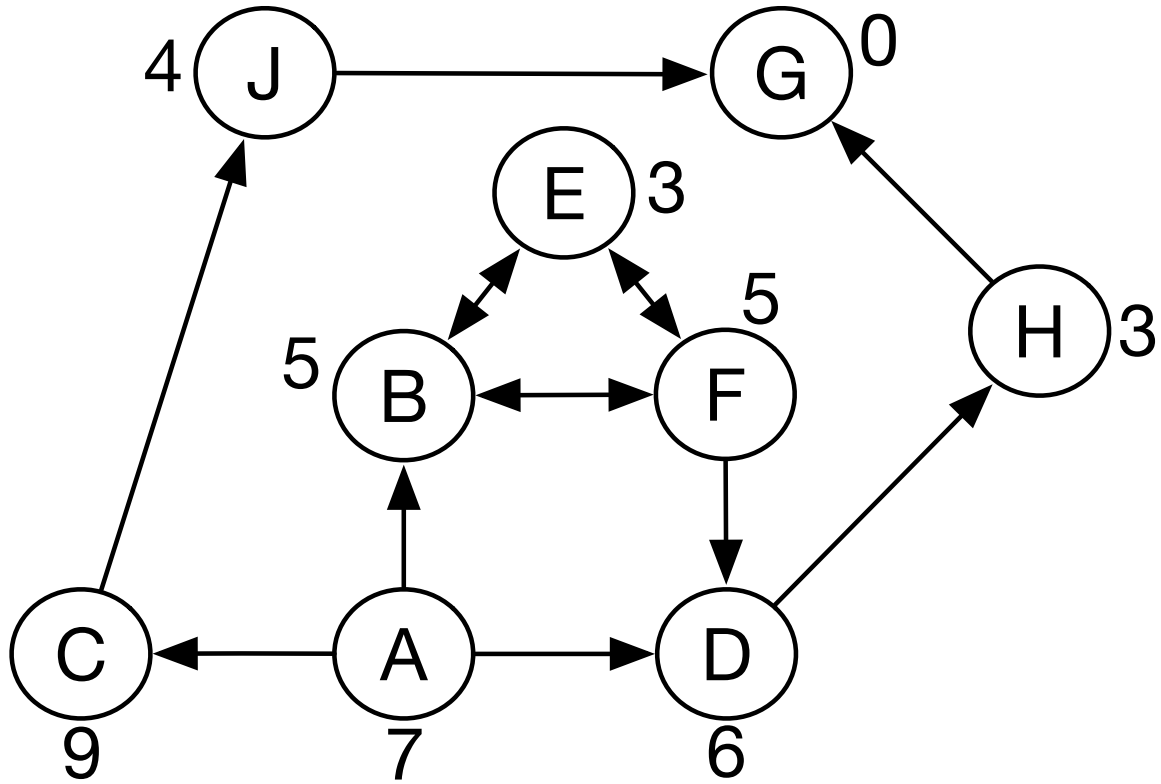
- If the nodes are points on a Euclidean plane and the cost is the distance, $h(n)$ can be the straight-line distance from $n$ to the closest goal.
- If the nodes are locations and cost is time, we can use the distance to a goal divided by the maximum speed.
- If the goal is to collect all of the coins and not run out of fuel, the cost is an estimate of how many steps it will take to collect the rest of the coins, refuel when necessary, and return to goal position.
- A heuristic function can be found by solving a simpler (less constrained) version of the problem.

# Reminder: State-Space Graph for the Delivery Robot (Acyclic)



# Illustrative Graph — Heuristic Search

From $A$ get to $G$:



# Heuristic depth-first Search

- Idea: in depth-first search select a neighbor that is closest to a goal according to the heuristic function.
- It inherits all of the advantages/disadvantages of depth-first search, but locally heads towards a goal.
- The neighbors can be added to the frontier so that the best neighbor is selected first
- This search selects the locally best path, but it explores all paths from the selected path before it selects another path.
- Although it is often used, it suffers from the problems of depth-first search, is not guaranteed to find a solution, and may not find an optimal solution.

# Best-first Search

- Idea: select a path whose end is closest to a goal according to the heuristic function.
- Best-first search selects a path on the frontier with minimal $h$-value.
- It treats the frontier as a priority queue ordered by $h$.

# Best-first search

Idea: use an evaluation function for each node
– estimate of "desirability"

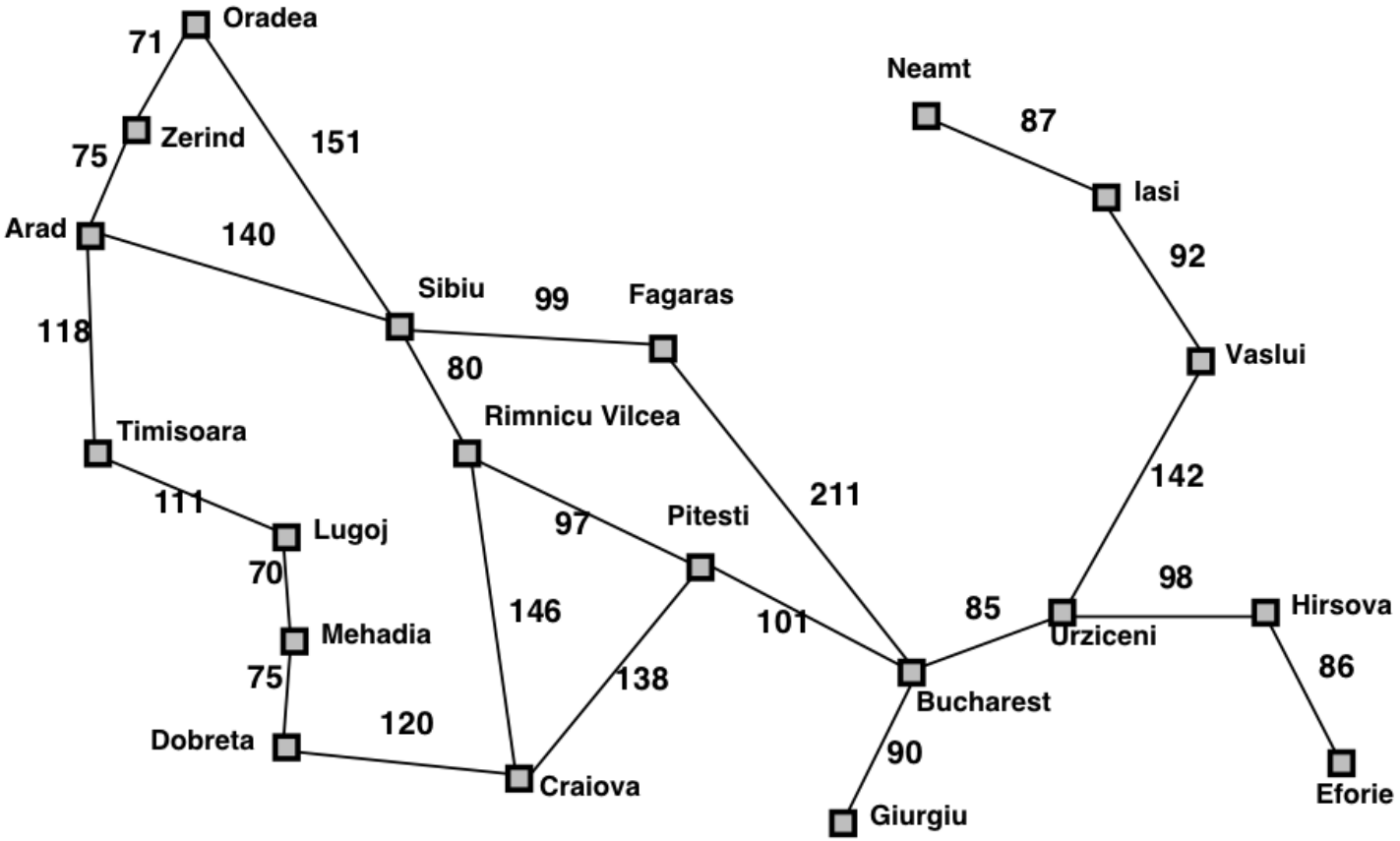
$\Rightarrow$ Expand most desirable unexpanded node

Implementation:

*fringe* is a queue sorted in decreasing order of desirability

Special cases:
greedy search
$A^*$ search

# Romania with step costs in km




| Straight–line distance to Bucharest | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Eforie** | 161 |
| **Fagaras** | 178 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Iasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 98 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

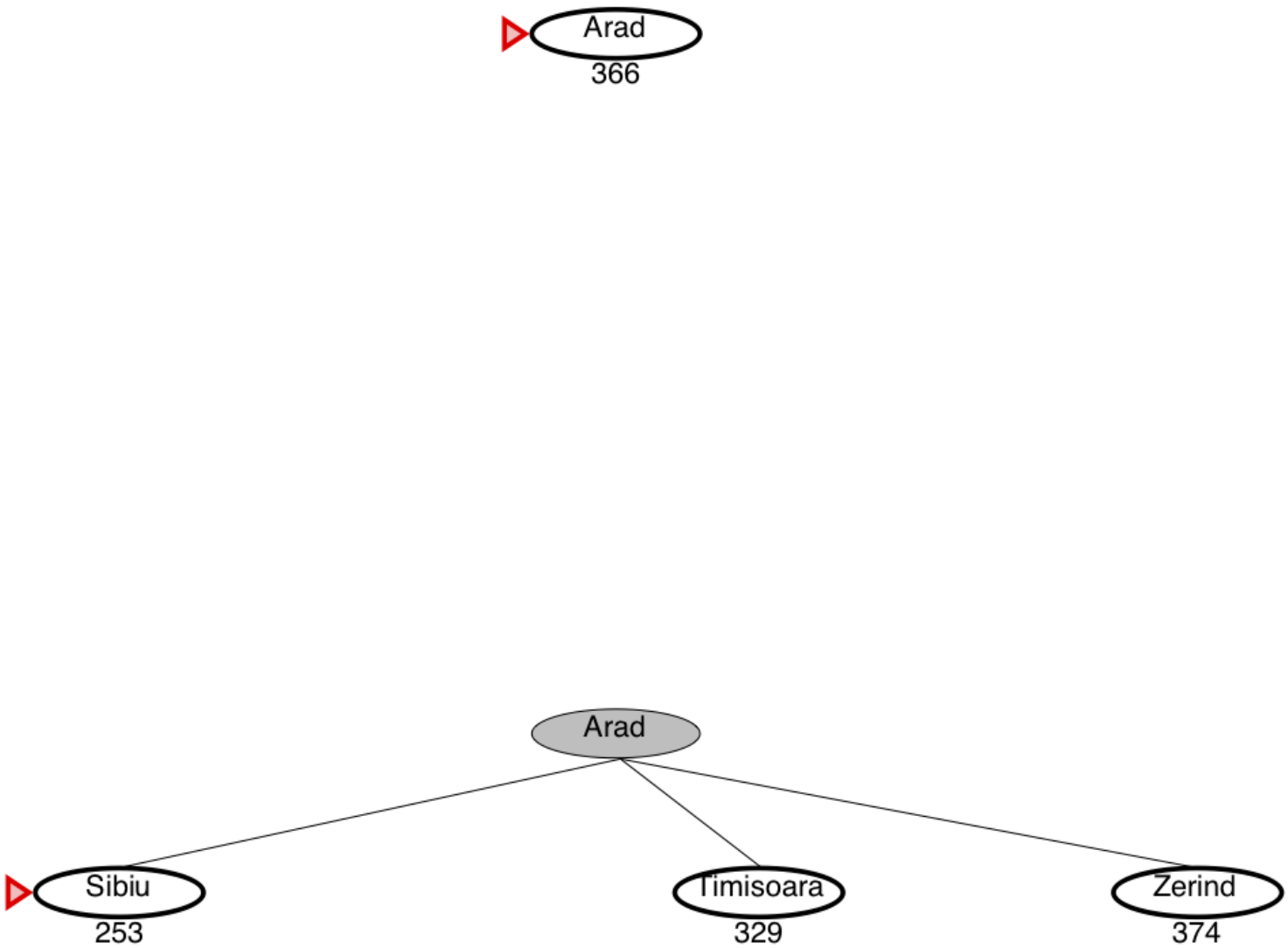
# Greedy search

Evaluation function $h(n)$ (**heuristic**)
= estimate of cost from $n$ to the closest goal

E.g., $h_{\mathrm{SLD}}(n)$ = straight-line distance from $n$ to Bucharest

Greedy search expands the node that **appears** to be closest to goal

# Greedy Search Example



# Properties of Greedy Search

- Complete?? No–can get stuck in loops, e.g.,
  Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
  Complete in finite space with repeated-state checking
- Time?? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space?? $O(b^m)$—keeps all nodes in memory
- Optimal?? No

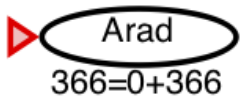
# $A^*$ Search

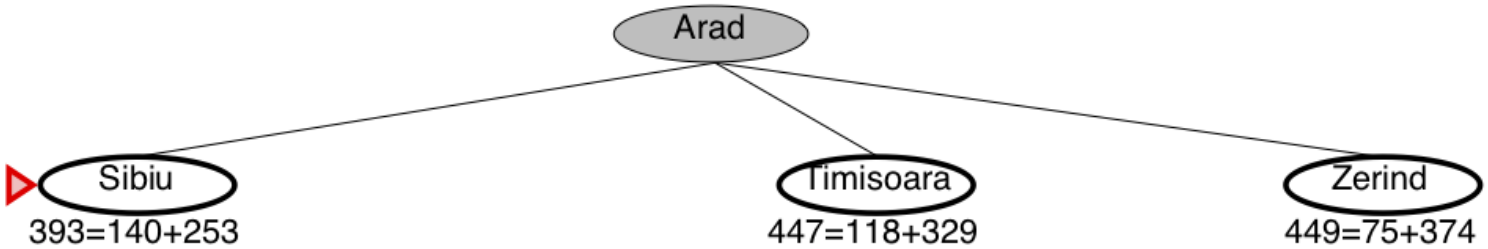
- $A^*$ search uses both path cost and heuristic values
- $cost(p)$ is the cost of path $p$.
- $h(p)$ estimates the cost from the end of $p$ to a goal.
- Let $f(p) = cost(p) + h(p)$.
  $f(p)$ estimates the total path cost of going from a start node to a goal via $p$.

$$\underbrace{\underbrace{start \xrightarrow{\text{path } p} n}_{cost(p)} \underbrace{\xrightarrow{\text{estimate}} goal}_{h(p)}}_{f(p)}$$

- In $A^*$ search, the frontier is a priority queue ordered by $f(p)$.
- It always selects the path on the frontier with the lowest estimated cost from the start to a goal node constrained to go via that path.

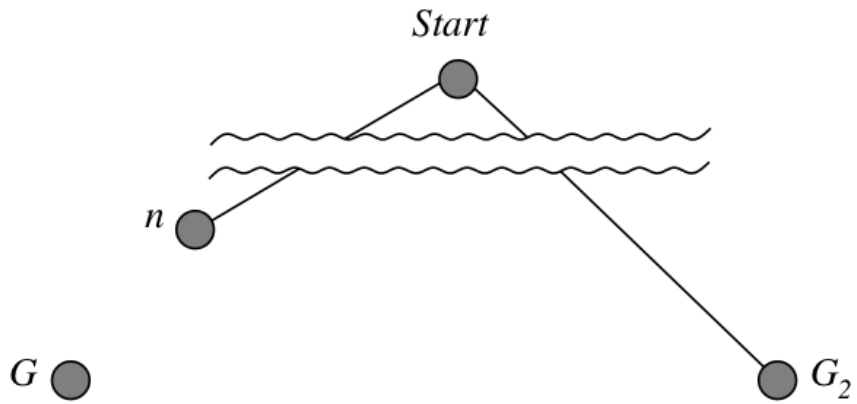
# $A^*$ Example


Arad
366=0+366


Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374

# Optimality of $A^*$ (standard proof)

Suppose some suboptimal goal $G_2$ has been generated and is in the queue. Let $n$ be an unexpanded node on a shortest path to an optimal goal $G_1$.

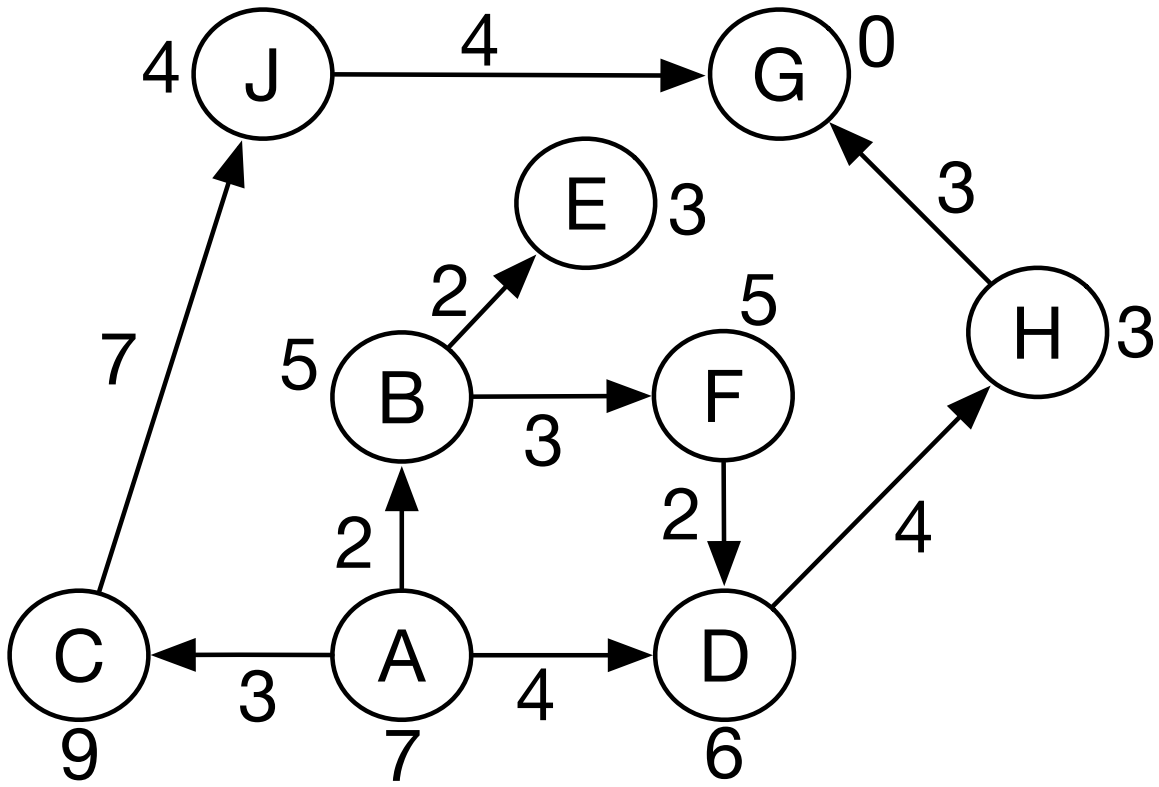



$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible}
\end{aligned}
$$

Since $f(G_2) > f(n)$, $A^*$ will never select $G_2$ for expansion

# Example graph with heuristics (acyclic)

Start: $A$. Goal: $G$.




Heuristic value of a node is shown next to the node.

# $A^*$ Search Algorithm

- $A^*$ is a mix of lowest-cost-first and best-first search.
- It treats the frontier as a priority queue ordered by $f(p)$.
- It always selects the node on the frontier with the lowest estimated distance from the start to a goal node constrained to go via that node.

# Complexity of $A^*$ Search

- Does $A^*$ search guarantee to find the path with fewest arcs?
- Does $A^*$ search guarantee to find the least-cost path?
- What happens on infinite graphs or on graphs with cycles if there is a solution?
- What is the time complexity as a function of length of the path selected?
- What is the space complexity as a function of length of the path selected?
- How does the goal affect the search?

# Admissibility of $A^*$

A search algorithm is admissible if, whenever a solution exists, it returns an optimal solution.

If there is a solution, $A^*$ always finds an optimal solution – -the first path to a goal selected — if

- the branching factor is finite
- arc costs are bounded above zero (there is some $\epsilon > 0$ such that all of the arc costs are greater than $\epsilon$), and
- $h(n)$ is nonnegative and an underestimate of the cost of the shortest path from $n$ to a goal node:

$$0 \leq h(n) \leq \text{cost of shortest path from } n \text{ to a goal}$$

# Why is $A^*$ admissible?

- If a path $p$ to a goal is selected from the frontier, can there be a lower cost path to a goal?
- $h(p) = 0$
- Suppose path $p'$ is on the frontier. Because $p$ was chosen before $p'$, and $h(p) = 0$:

$$cost(p) \leq cost(p') + h(p').$$

- Because $h$ is an underestimate:

$$cost(p') + h(p') \leq cost(p'')$$

for any path $p''$ to a goal that extends $p'$.

- So $cost(p) \leq cost(p'')$ for any other path $p''$ to a goal.

# Summary of Search Strategies

| Strategy | Frontier Selection | Complete | Halts | Space |
|---|---|---|---|---|
| Depth-first | Last node added | No | No | Linear |
| Breadth-first | First node added | Yes | No | Exp |
| Heuristic depth-first | Local min $h(p)$ | No | No | Linear |
| Best-first | Global min $h(p)$ | No | No | Exp |
| Lowest-cost-first | Minimal $cost(p)$ | Yes | No | Exp |
| $A^*$ | Minimal $f(p)$ | Yes | No | Exp |

Complete — if there a path to a goal, it can find one, even on infinite graphs.

Halts — on finite graph (perhaps with cycles).

Space — as a function of the length of current path