Graphs: A search*

Jordi Cortadella and Jordi Petit
Department of Computer Science

Shortest path between two nodes

How to find the shortest path from Barcelona to Girona?

Easy: run Dijkstra from Barcelona

Dijkstra will find ALL shortest paths from Barcelona

Do we really need to waste computations exploring roads that go to Lleida, Amposta or Vielha?

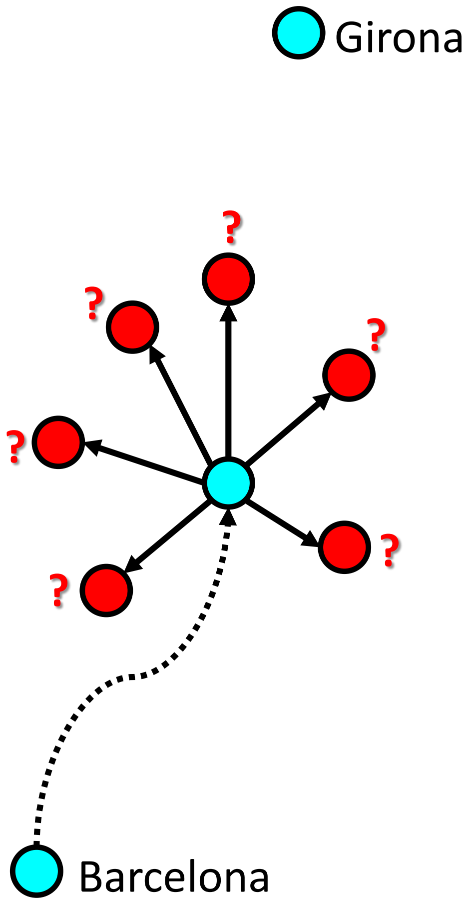
A* search algorithm

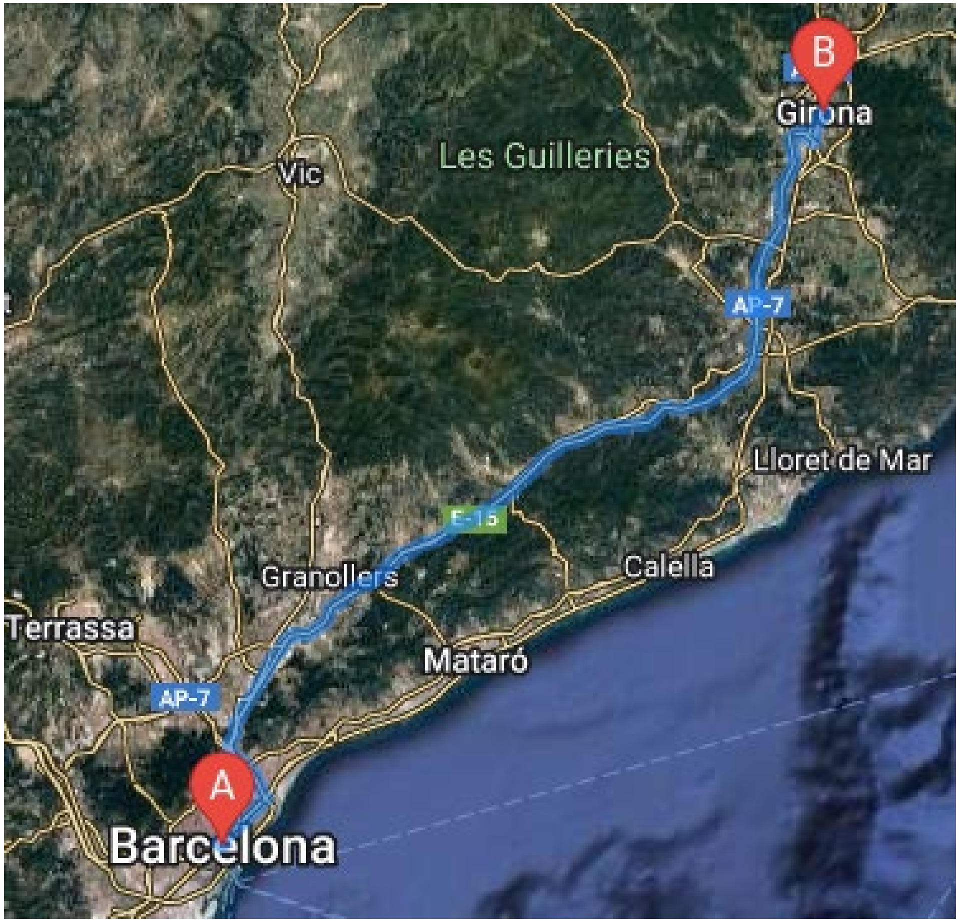
- Original paper:

  P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968.

- A* is a class of graph searching strategies using ad hoc heuristic information. A* guarantees optimal solutions when the heuristic information meets certain properties.

A* search: heuristic guidance

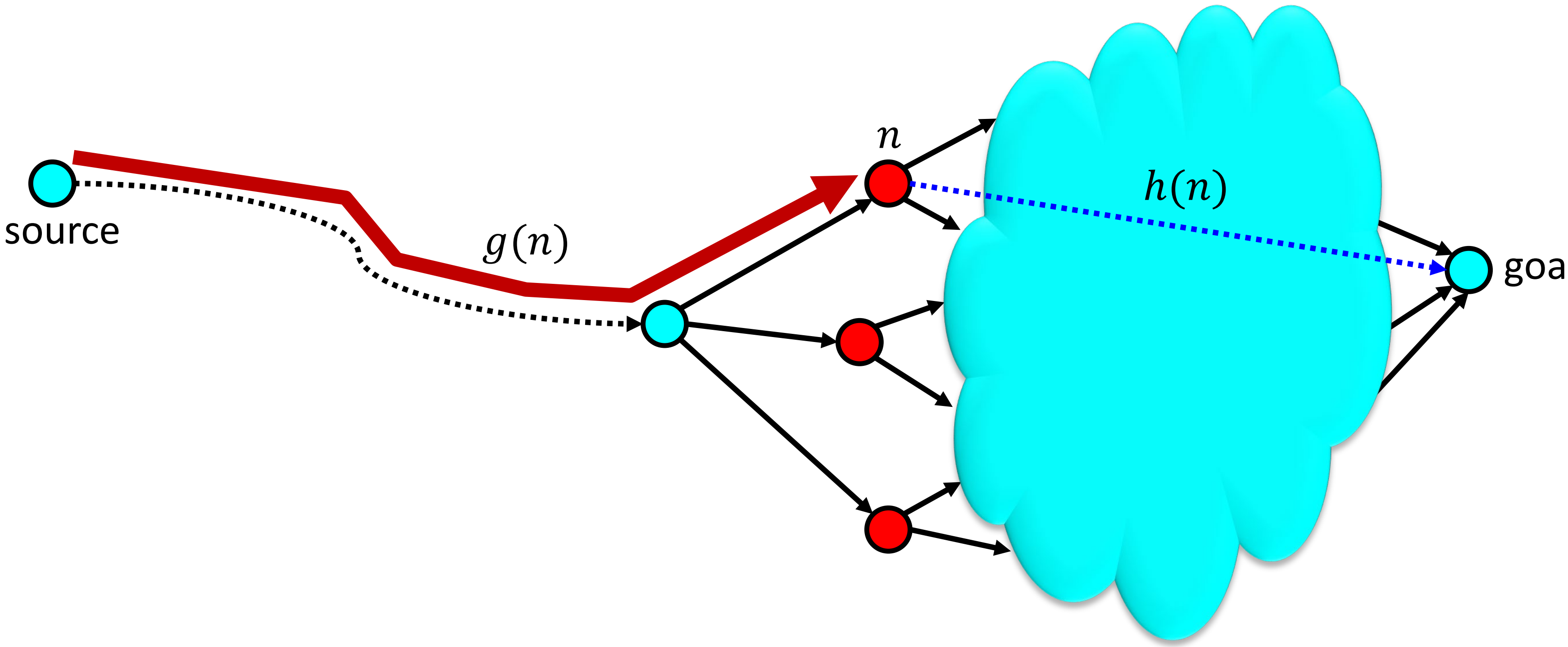




What is the most promising node to explore?

Heuristic: select nodes that reduce the straight-line distance to the target

A* search: intuition



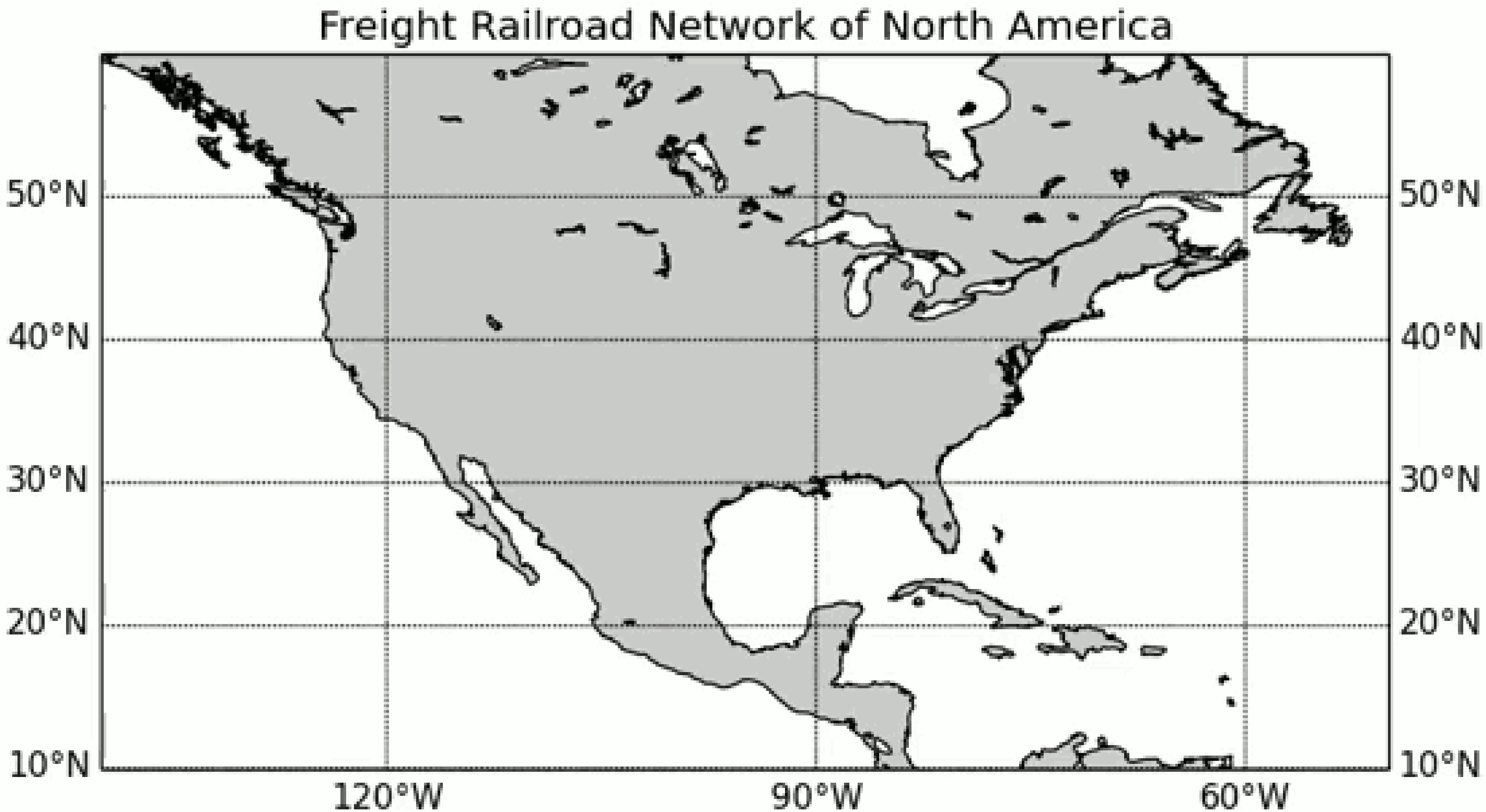
How to extend the path and find the next node n?

For each node n calculate $f(n) = g(n) + h(n)$

- $g(n)$ is the cost of the path from the source to n
- $h(n)$ is the estimated cost of the cheapest path from n to the goal

Select the node with minimum $f(n)$

Example: train network



Finding the optimum path from Washington, D.C. and Los Angeles

$h(x)$ is the great-circle distance (the shortest possible distance on a sphere) to the target

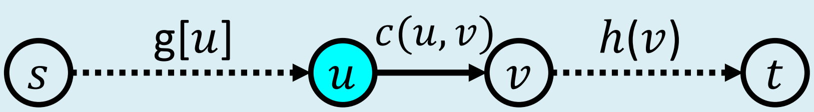
Source: https://en.wikipedia.org/wiki/A*_search_algorithm

A* algorithm for shortest paths

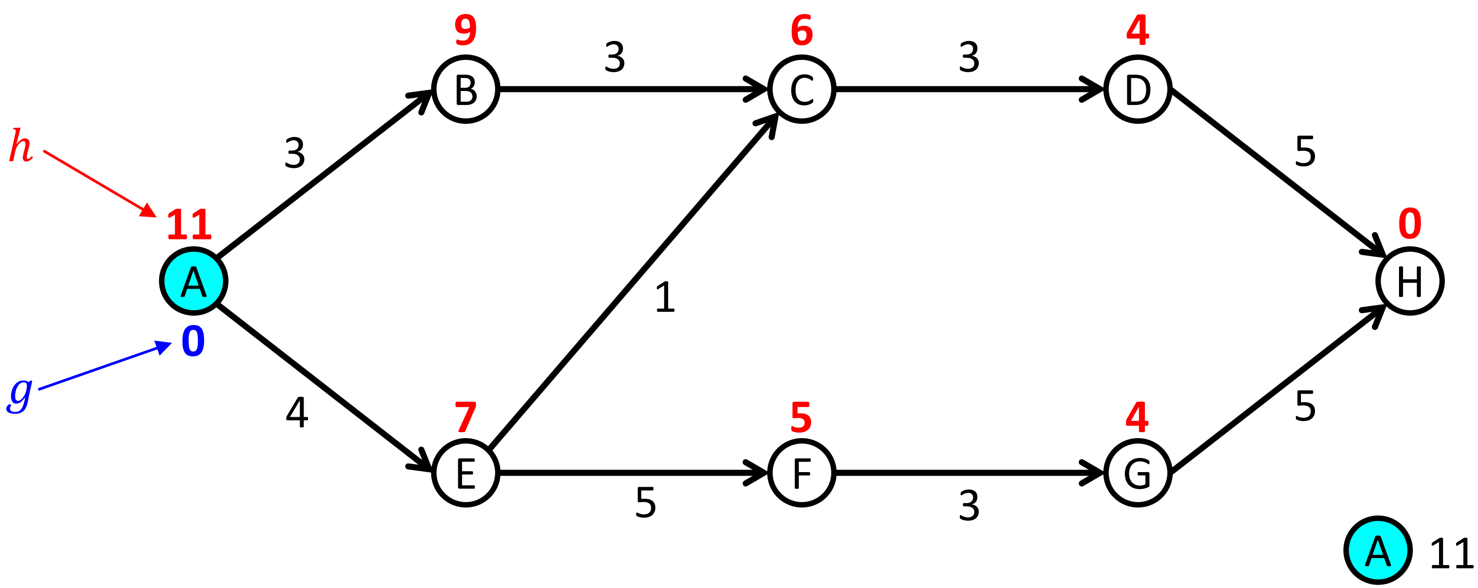
```
def Astar_search(G, s, t, c, h) → pred:
  """Input:  Graph G(V,E), source node s, target node t,
             positive edge costs {c(e): e ∈ E},
             function to estimate the cost to the target {h(v): v ∈ V}
    Output: pred[u] has the predecessor in the shortest path from s to t,
             if t ∉ pred, no path exists from s to t
  """
  f = {}  # dictionary for the f value (∞ if not present)
  g = {}  # dictionary for the g value (∞ if not present)
  pred = {}  # dictionary of predecessors

  g[s] = 0
  f[s] = h(s)
  Q = {s}  # open nodes: priority queue sorted by f

  while not Q.empty():
    u = Q.deletemin()  # get open node with min cost
    if u == t: return
    for all (u,v) ∈ E:
      gv = g[u] + c(u,v)
      if gv < g[v]:  # g[v]=∞ if v ∉ g
        g[v] = gv
        f[v] = gv + h(v)
        pred[v] = u
        Q.add(v)  # new open node (or update if already in Q)
```

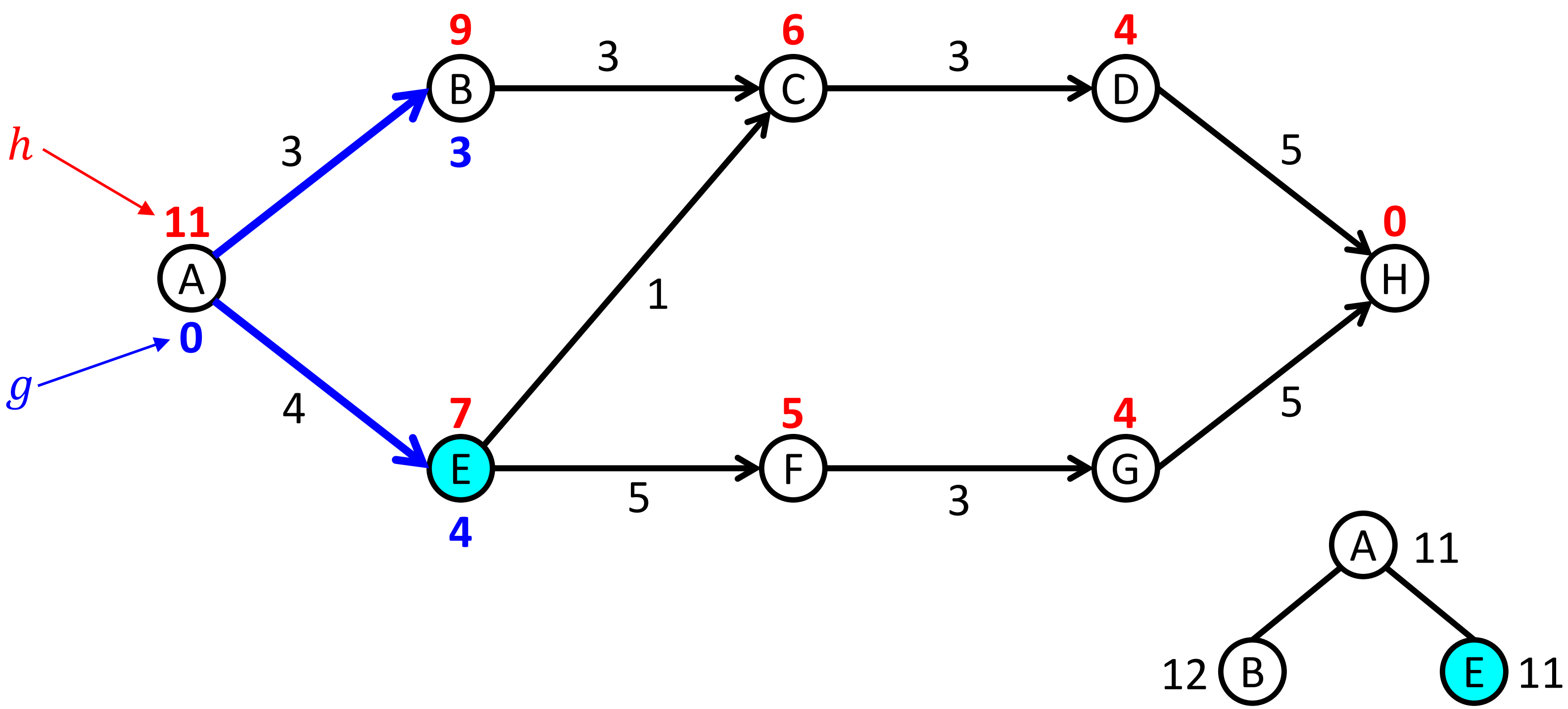


Running A*: example



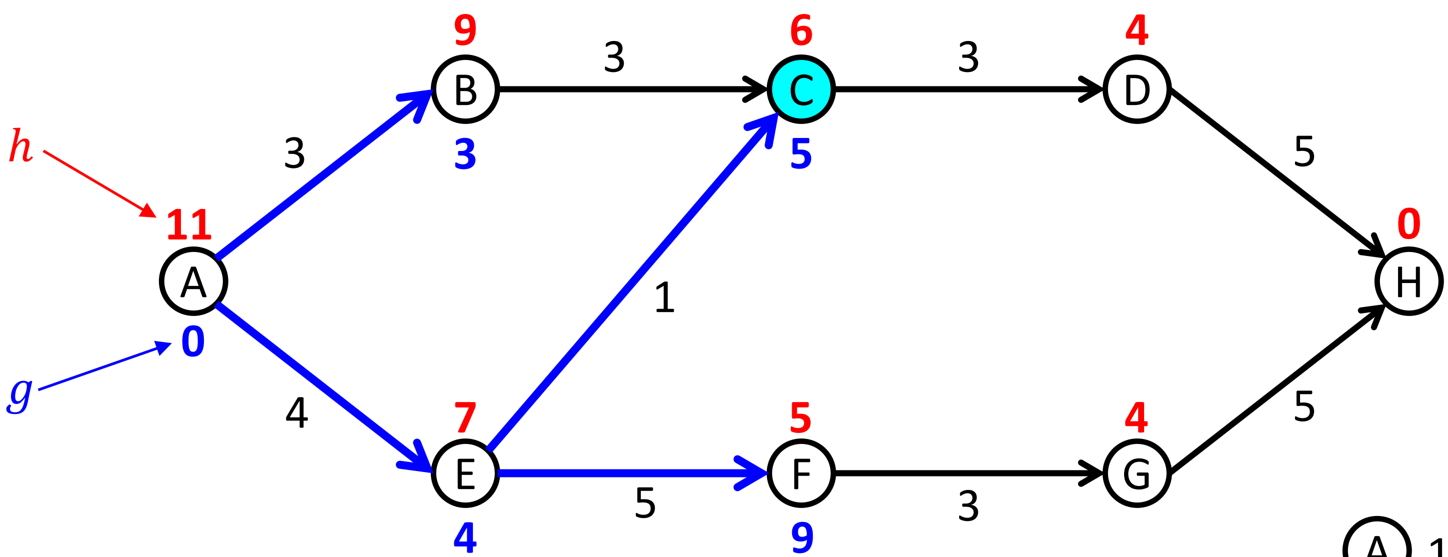
Q = {A(11)}

Running A*: example

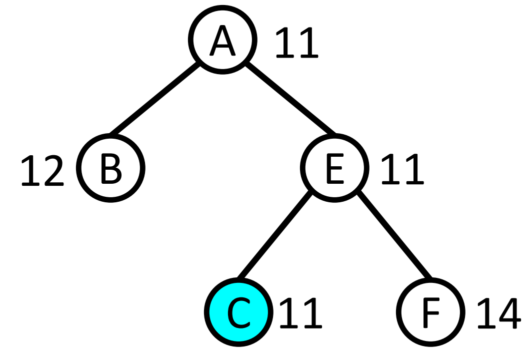


Q = {E(11), B(12)}

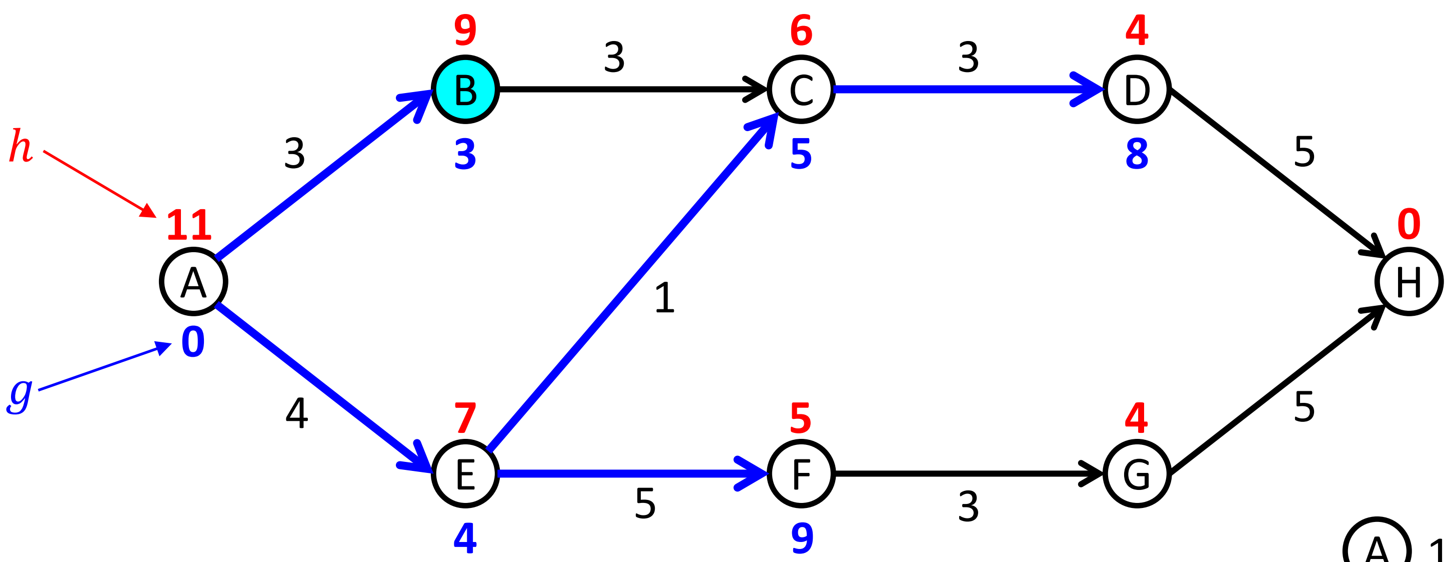
Running A*: example

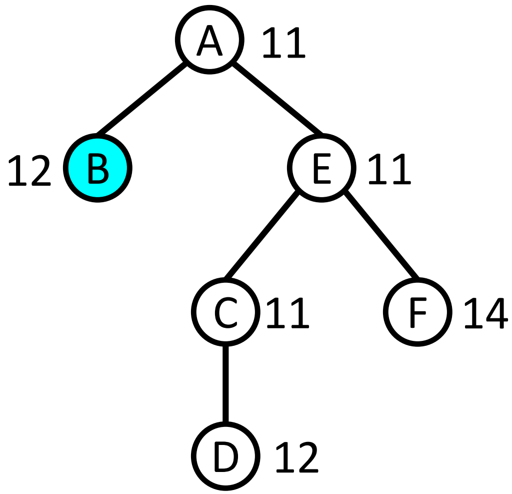


Q = {C(11), B(12), F(14)}



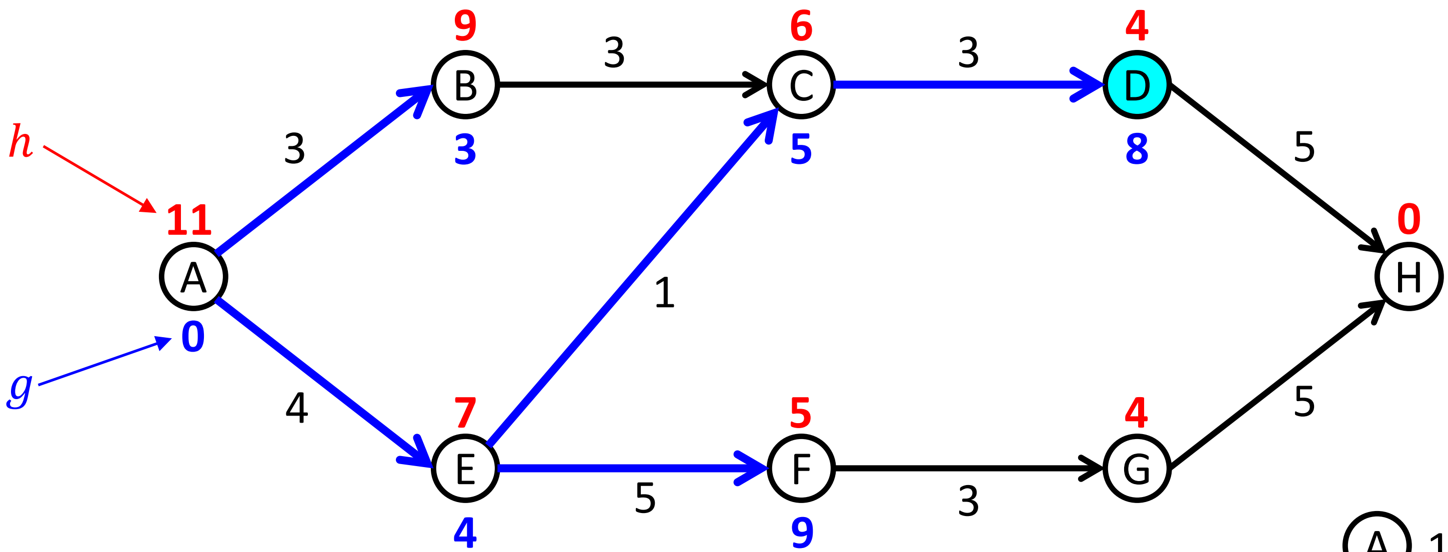
Running A*: example



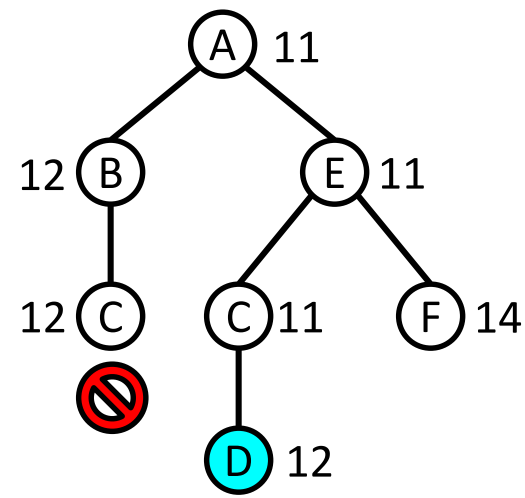


Q = {B(12), D(12), F(14)}

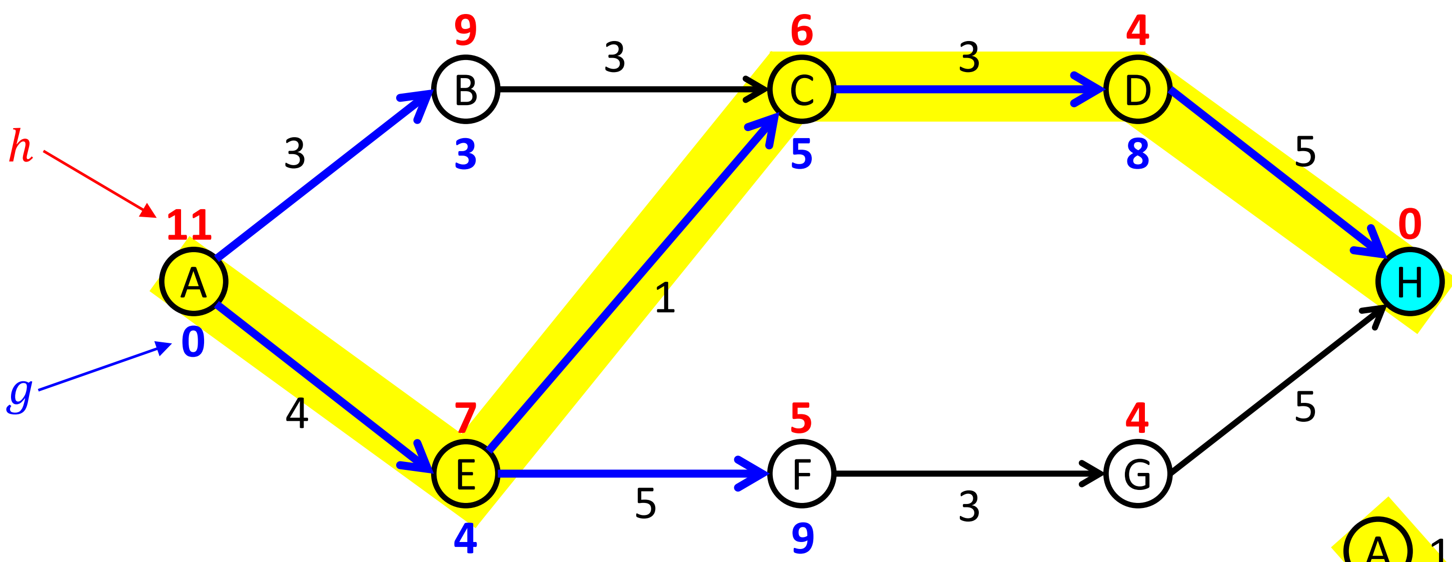
Running A*: example



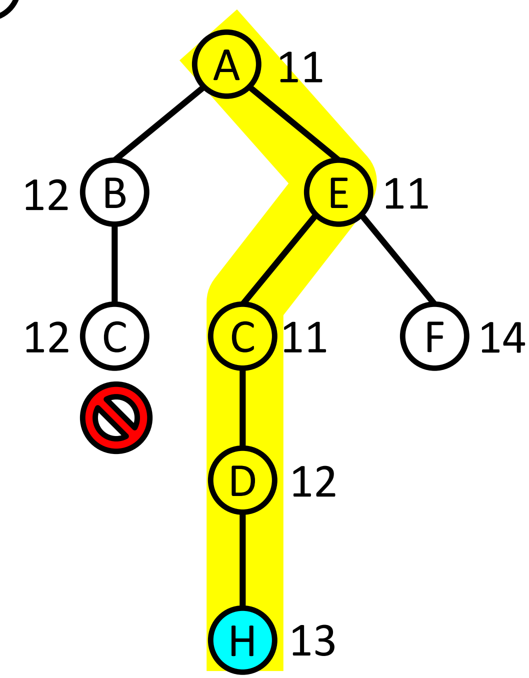
$$Q = \{D(12), F(14)\}$$



Running A*: example



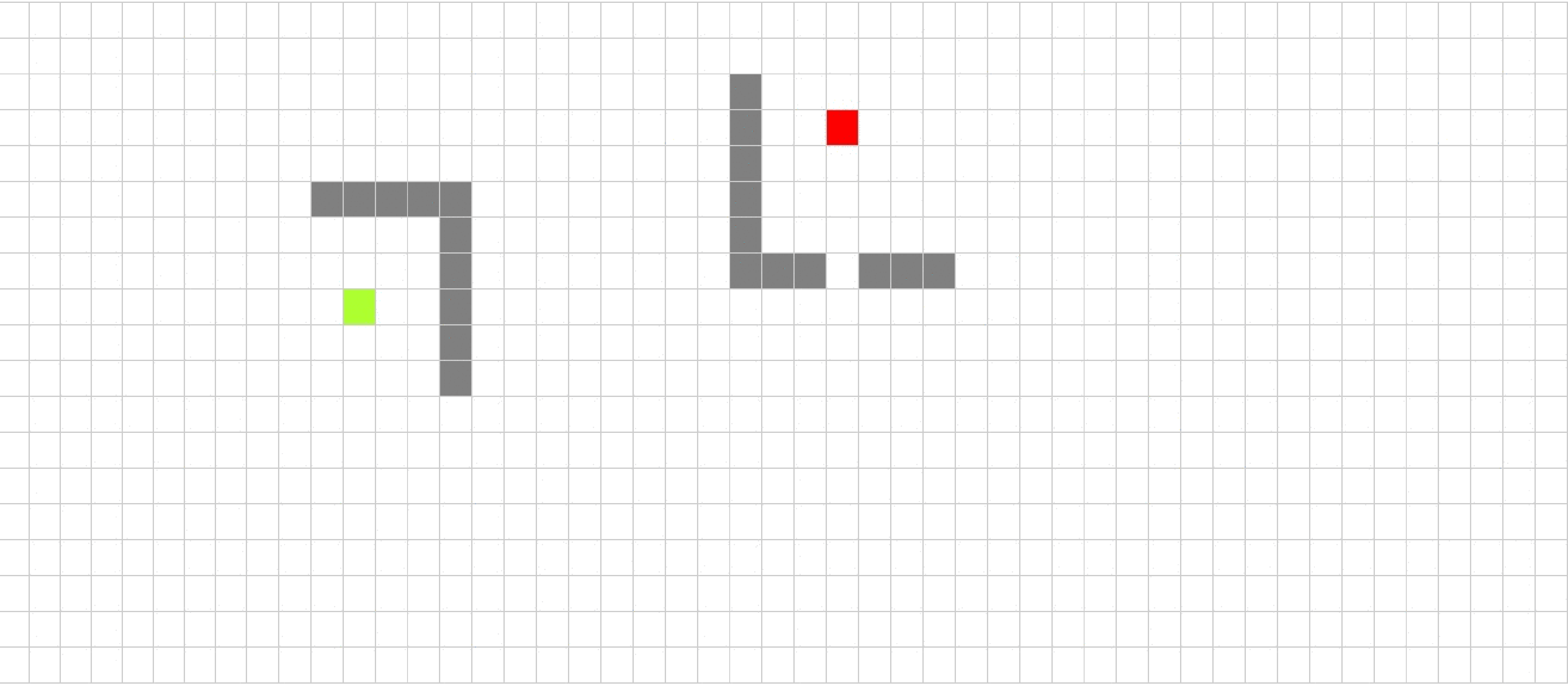
Q = {H(13), F(14)}



Finally, H is selected and the algorithm terminates

Backtrack from H to find the selected path

Avoiding obstacles



Source: https://github.com/vittin/A-Star/

The heuristic function: $h(x)$

- A* relies on a good heuristic to estimate the cost to reach the goal

- How about using a "bad" heuristic function?
  - Does the algorithm find the optimum path?
  - Does it run efficiently?

- Let us study the concepts of *admissible* and *consistent* heuristic function

Admissibility

- A heuristic function is said to be *admissible* if it never overestimates the cost of reaching the goal

- Example: the straight-line distance in a map is an *admissible* function
  (no path can be shorter than the straight line)

Admissibility

- Important result:
  - If $h(x)$ is admissible, A* will find the optimum path

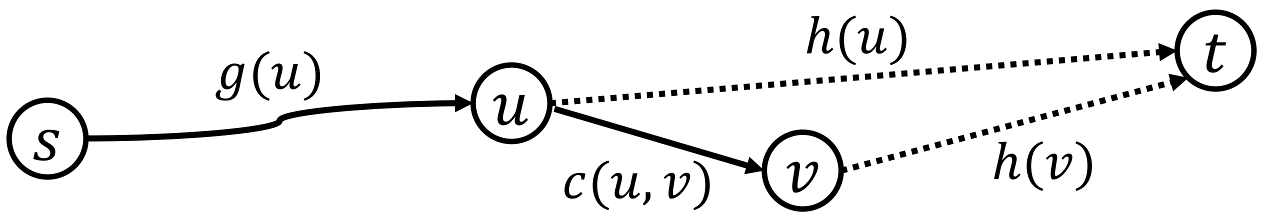
- Proof (informal):
  - A* will never overlook a path with lower cost, since a node v with lower $f(v)$ than the goal will exist in the set of open nodes before the goal is reached.

Consistency

- A heuristic function $h(x)$ is said to be *consistent* (or monotone) if

$$h(u) \leq c(u,v) + h(v)$$

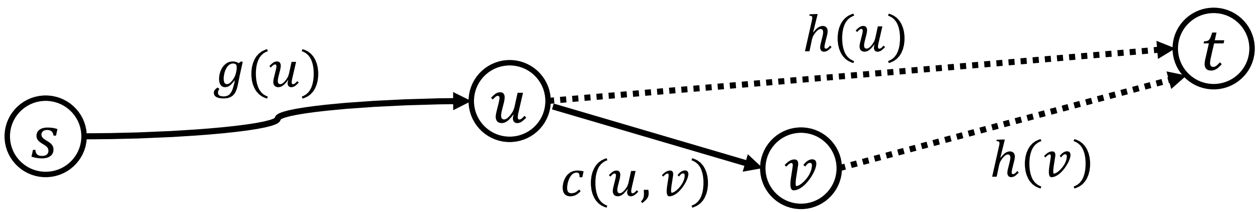
for every edge (u,v) with cost $c(u,v)$



- Important result:
  - If $h(x)$ is consistent, A* is guaranteed to find an optimal path without processing any node more than once

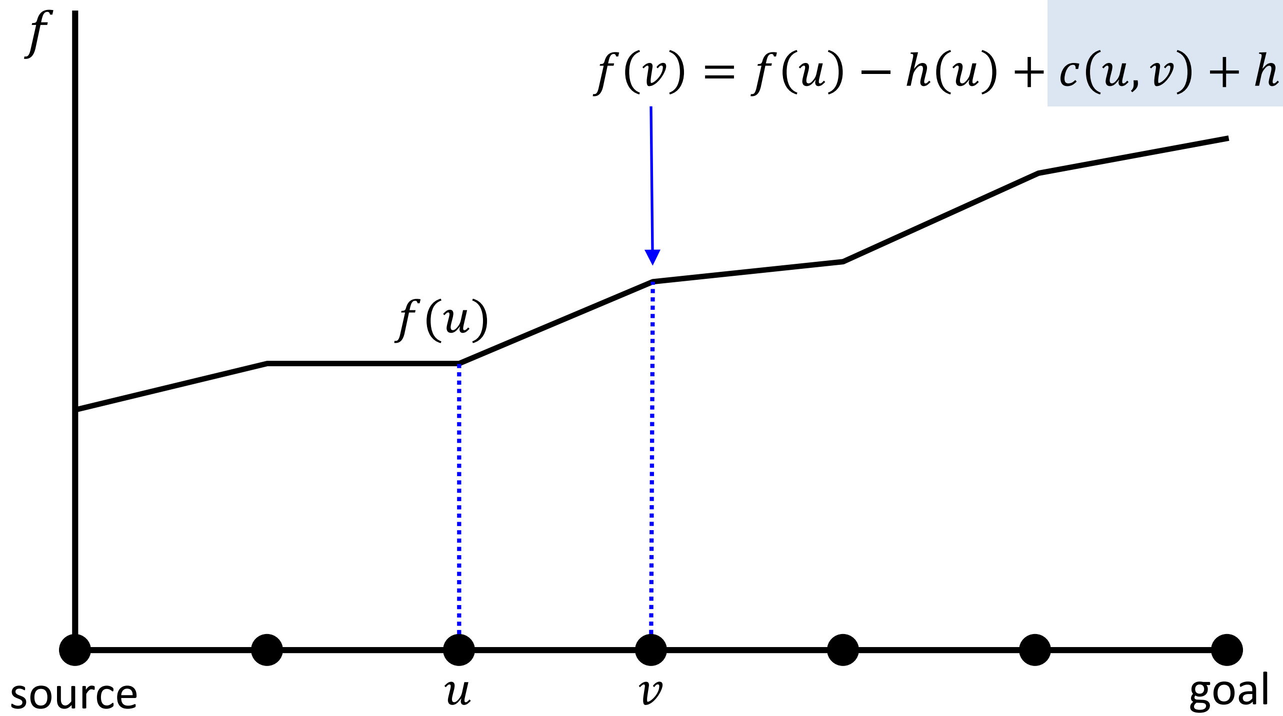
Consistency

If $h(x)$ is consistent then $f(x)$ is an increasing function



$$h(u) \le c(u,v) + h(v)$$

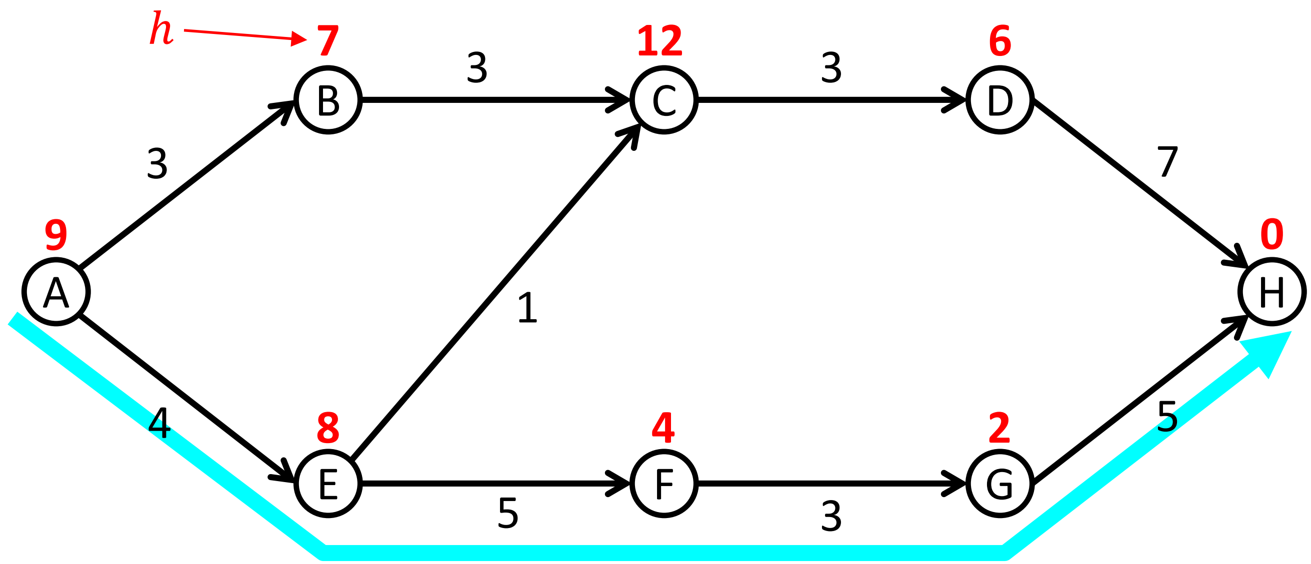
$$f(v) = f(u) - h(u) + c(u,v) + h(v) \ge f(u)$$



Example

Goal: find the shortest path from A to H

- The h function is not admissible (it overestimates the cost to the goal)
- Example: $h(\mathrm{C}) = 12$. The solution may not be optimal

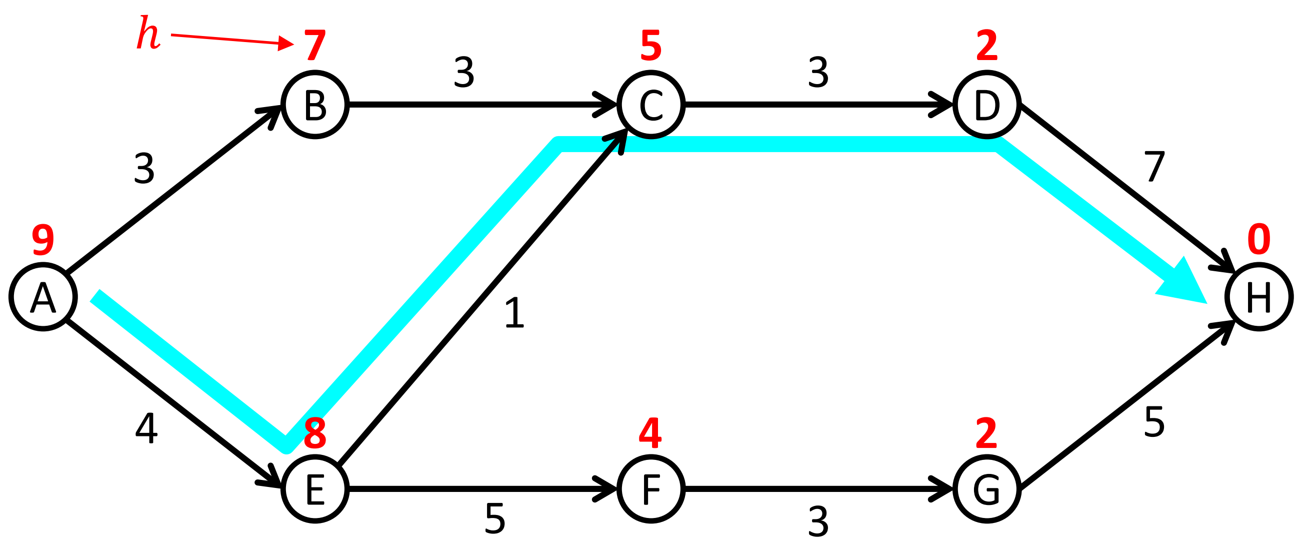


Visited nodes during the A* search: A B E F G H

Example

Goal: find the shortest path from A to H.

- The h function is admissible (it does not overestimate the cost to the goal)
- The h function is not consistent, e.g., $h(\mathrm{E}) > d(\mathrm{E},\mathrm{C}) + h(\mathrm{C})$
- The solution will be optimal. Some nodes may be visited more than once

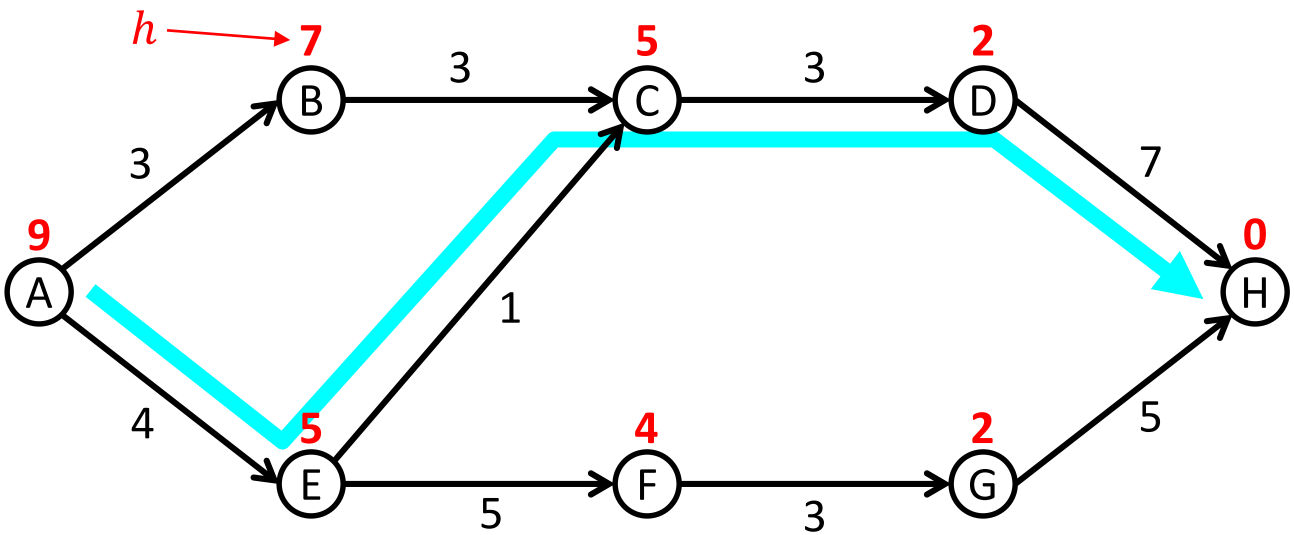


Visited nodes during the A* search: A B **C D** E **C D** F G H

Example

Goal: find the shortest path from A to H.

- The h function is admissible (it does not overestimate the cost to the goal)
- The h function is consistent
- The solution will be optimal and the nodes will be visited once at most



Visited nodes during the A* search: A E B C D F G H

Complexity

- The time complexity of A* highly depends on the heuristic function. The worst-case complexity is $\mathrm{O}(b^d)$, where
  - d is the depth of the shortest path
  - b is the average branching factor (number of successor nodes of each node)

- Each heuristic has an *effective* branching factor b^*, and the number of visited nodes is:

$$1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$

- Good heuristics have small values for b^* (the optimal heuristic has $b^* = 1$)

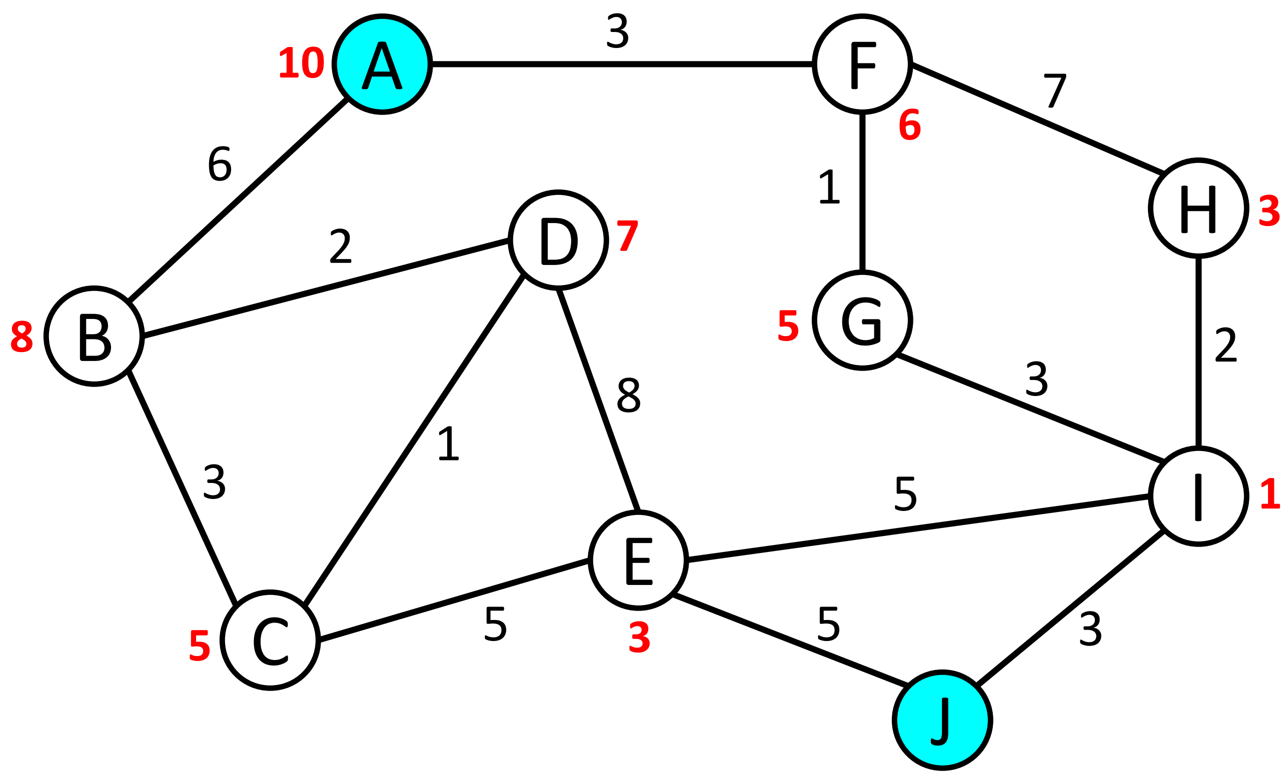
Complexity

- A* may not terminate if the graph is infinite and no path exists to the target

- A* keeps all generated nodes in memory. There are memory-bounded heuristic searches:
  - Iterative deepening A*: guided DFS, no priority queue, nodes may be visited multiple times
  - Simplified Memory-Bounded A* (SMA*): nodes with highest f-cost pruned from the queue
  - and others ...

EXERCISES

Shortest path



Find the shortest path from A to J using the A* algorithm.
The red numbers next to the nodes represent the heuristic value.
Questions:

- Is the heuristic admissible?
- Is the heuristic consistent?
- Report the visited node at each step of the algorithm

8-puzzle problem

Initial state

2	**8**	**3**
1	**6**	**4**
7		**5**

Goal

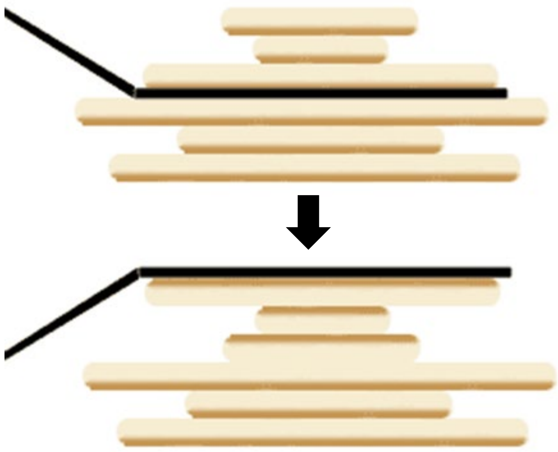
1	**2**	**3**
8		**4**
7	**6**	**5**

- Use the A* algorithm to find the smallest sequence of shifts to reach the goal. Depict the search tree

- Consider:
  - $g(n)$ = depth of the node (number of shifts)
  - $h(n)$ = number of misplaced tiles

Pancake sorting

- You have a disordered stack of pancakes of different sizes. You want to sort this pile (smallest pancake on top, largest one at the bottom) using a spatula by flipping parts of the stack



- Describe how you would use A* to find the shortest sequence of flips that sort the pile
  - Define the initial state, successor states, and the functions $g(n)$ and $h(n)$. Make sure $h(n)$ is admissible