Searching: Informed Search

CSIS 4463

Overview

- "Best First Greedy Search"
  - The inadequacies of "Best First Greedy" heuristic search.
- A* Search
  - When should the search stop?
  - Admissible heuristics
  - A* search is complete
  - A* search will always terminate
  - A*'s dark secret
- Saving masses of memory with IDA* (Iterative Deepening A*)

Heuristic Search

Suppose in addition to the standard search specification we also have a *heuristic*.

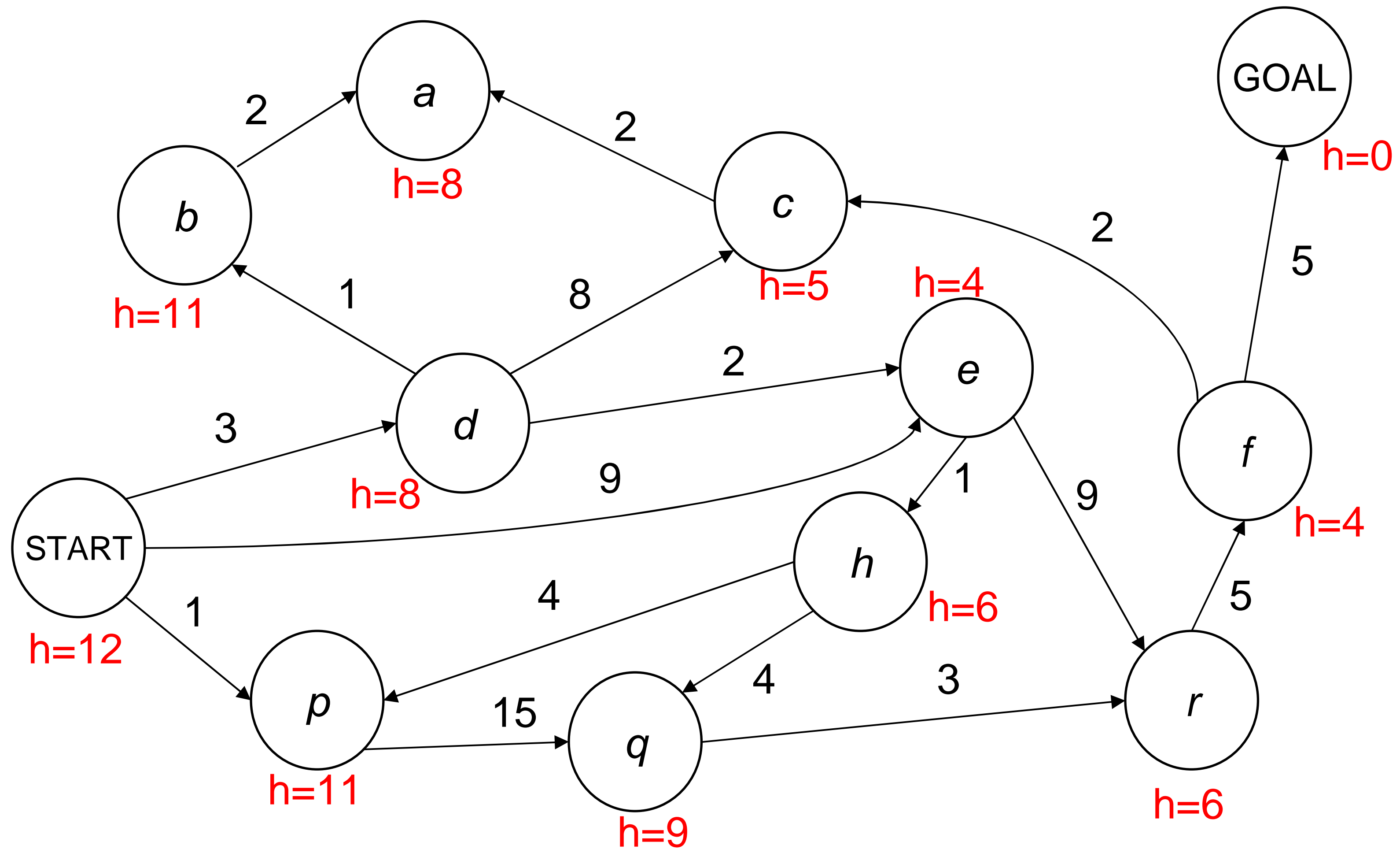
A heuristic function maps a state onto an estimate of the cost to the goal from that state.

Can you think of examples of heuristics?

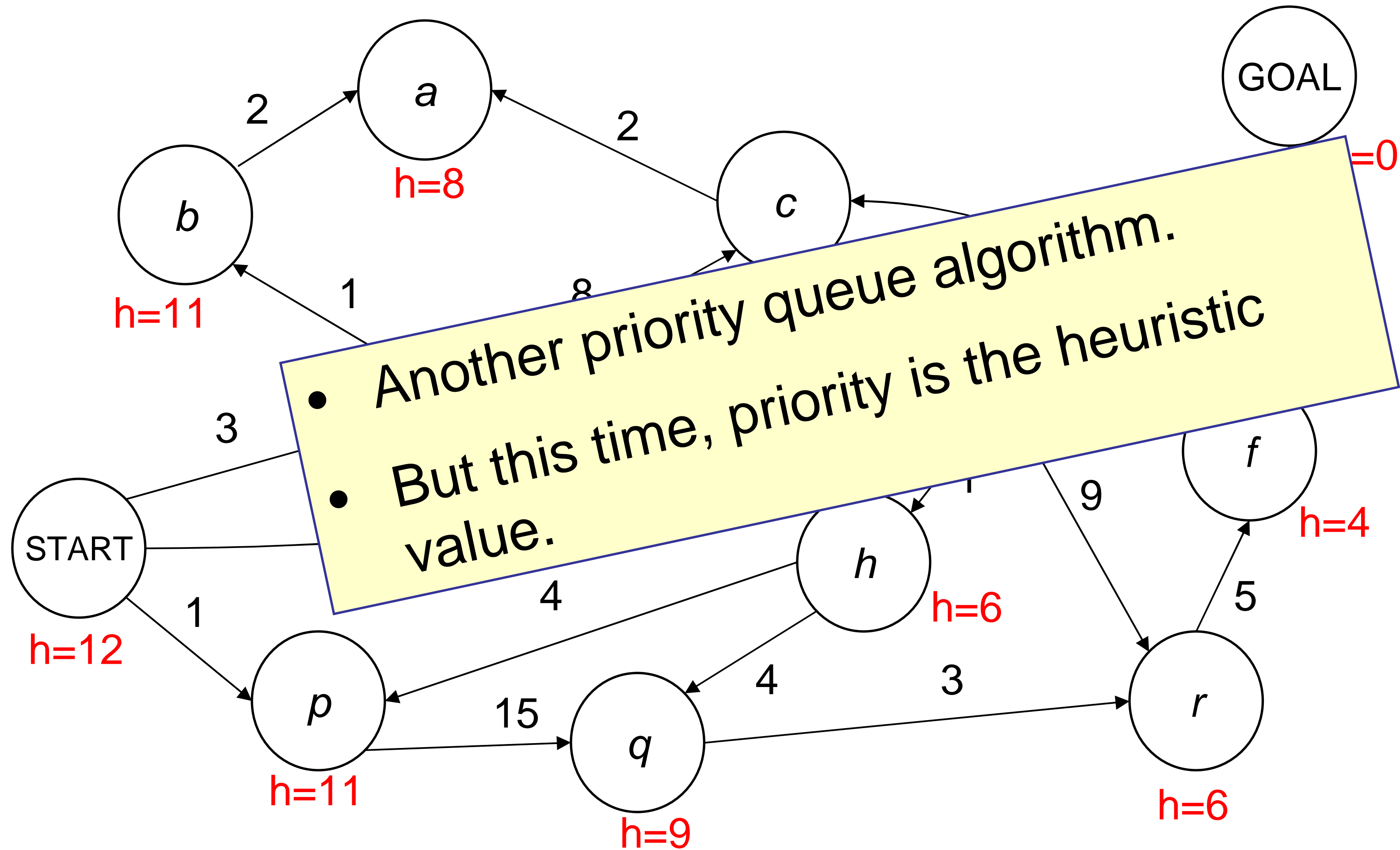
- E.G. for the 8-puzzle?
- E.G. for planning a path through a maze?

Denote the heuristic by a function $h(s)$ from states to a cost value.

Euclidian Heuristic



Euclidian Heuristic



Best First "Greedy" Search

```
Init-PriQueue(PQ)
Insert-PriQueue(PQ,START,h(START))
while (PQ is not empty and PQ does not contain a goal state)
        (s , h ) := Pop-least(PQ)
        foreach s' in succs(s)
        if s' is not already in PQ and s' never previously been visited
                Insert-PriQueue(PQ,s',h(s'))
```

Algorithm		Complete	Optimal	Time	Space
BestFS	Best First Search				

Best First "Greedy" Search

```
Init-PriQueue(PQ)
Insert-PriQueue(PQ,START,h(START))
while (PQ is not empty and PQ does not contain a goal state)
        (s , h ) := Pop-least(PQ)
        foreach s' in succs(s)
        if s' is not already in PQ and s' never previously been visited
                Insert-PriQueue(PQ,s',h(s'))
```

Algorithm		Complete	Optimal	Time	Space
BestFS	Best First Search	Y			

Best First "Greedy" Search

```
Init-PriQueue(PQ)
Insert-PriQueue(PQ,START,h(START))
while (PQ is not empty and PQ does not contain a goal state)
        (s , h ) := Pop-least(PQ)
        foreach s' in succs(s)
        if s' is not already in PQ and s' never previously been visited
                Insert-PriQueue(PQ,s',h(s'))
```

Algorithm		Complete	Optimal	Time	Space
BestFS	Best First Search	Y	**N**		

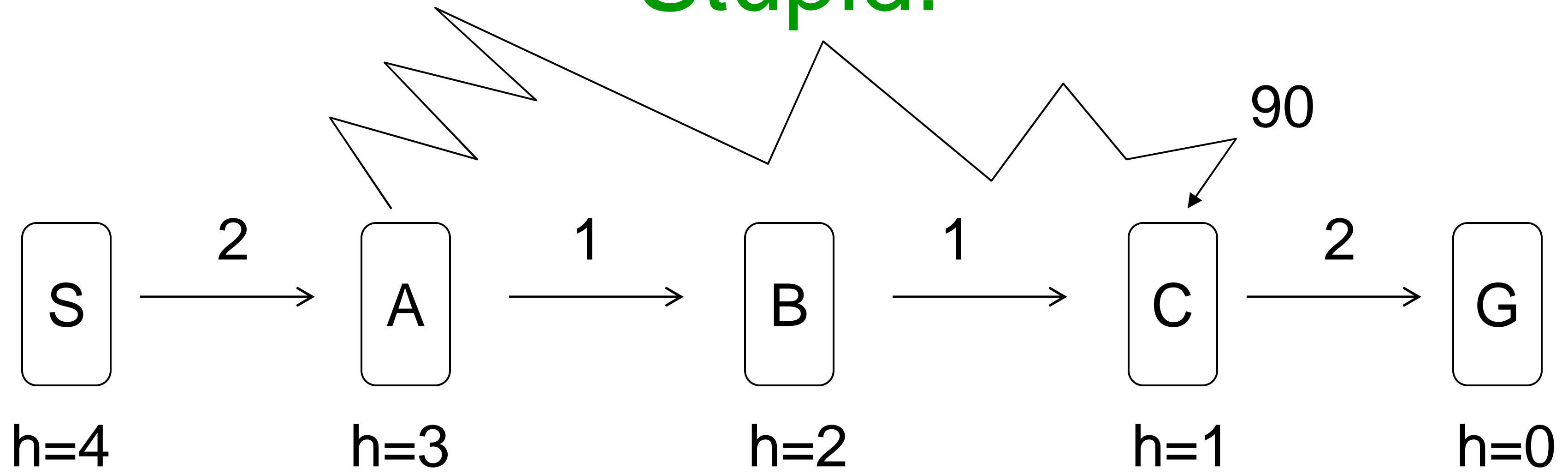
Best First "Greedy" Search

```
Init-PriQueue(PQ)
Insert-PriQueue(PQ,START,h(START))
while (PQ is not empty and PQ does not contain a goal state)
        (s , h ) := Pop-least(PQ)
        foreach s' in succs(s)
        if s' is not already in PQ and s' never previously been visited
                Insert-PriQueue(PQ,s',h(s'))
```

Algorithm		Complete	Optimal	Time	Space
BestFS	Best First Search	Y	**N**	$O(min(N,B^{LMAX}))$	$O(min(N,B^{LMAX}))$

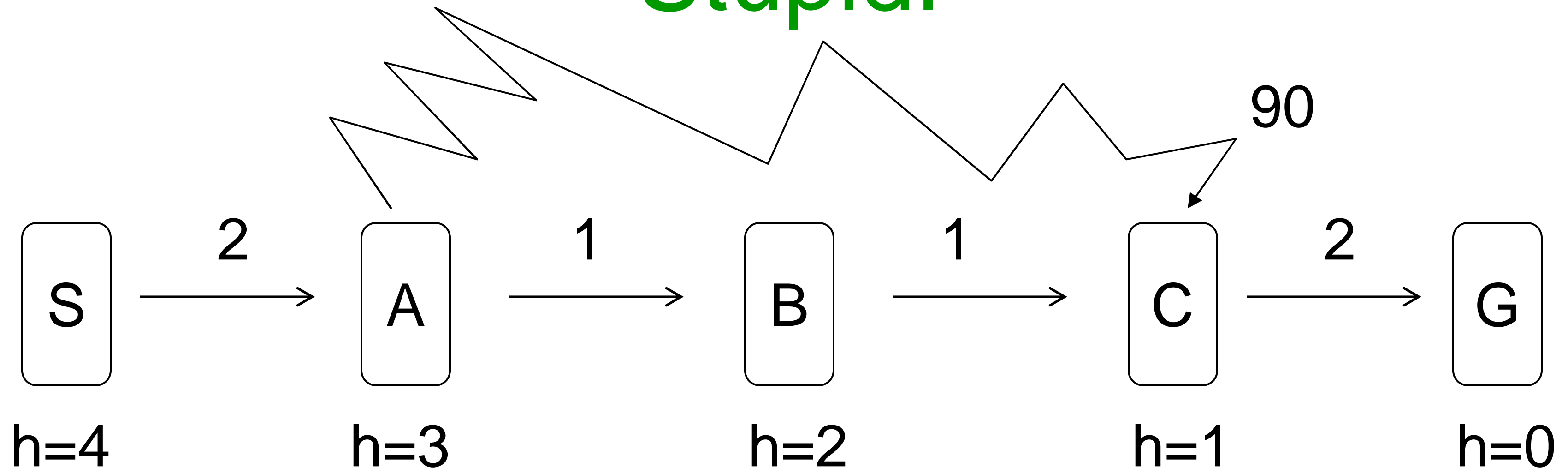
A few improvements to this algorithm can make things much better. It's a little thing we like to call: A*….

Let's Make "Best first Greedy" Look Stupid!



- PQ = {(S,4)}
- Expand S
  - PQ = {(A,3)}
- Expand A
  - PQ = {(C,1), (B,2)}
- Expand C
  - PQ = {(G,0), (B,2)}
- Expand G… found the goal… solution path S,A,C,G has cost 94…
- But a shorter path exists… S,A,B,C,G with cost 6

Let's Make "Best first Greedy" Look Stupid!



- Best –first greedy is clearly not guaranteed to find optimal
- Obvious question: What can we do to avoid the stupid mistake?

A* - The Basic Idea

- Best-first greedy: When you expand a node n, take each successor n' and place it on PriQueue with priority $h(n')$
- A*: When you expand a node n, take each successor n' and place it on PriQueue with priority

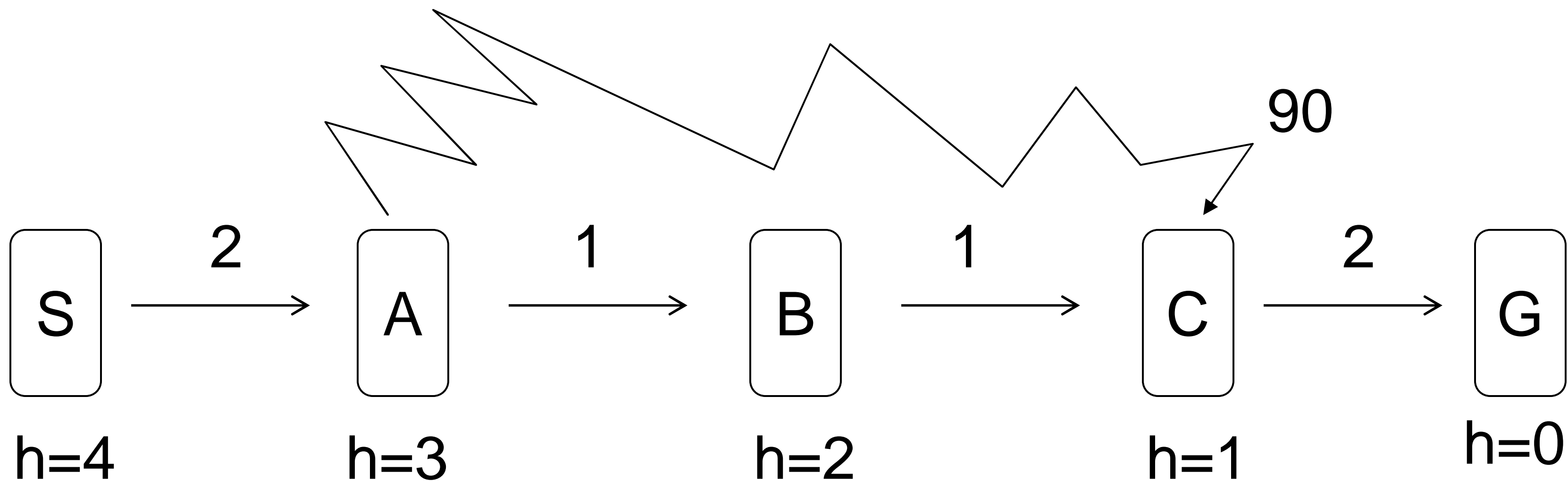
$$(\text{Cost of getting to } n') + h(n') \quad (1)$$

$$\text{Let } g(n) = \text{Cost of getting to } n \quad (2)$$

and then define…

$$f(n) = g(n) + h(n) \quad (3)$$

A* Looking Non-Stupid



When we're at state A…

We have 2 sucessors, B and C…

f(B) = g(B) + h(B) = 3 + 2 = 5

f(C) = g(C) + h(C) = 92 + 1 = 93

PQ = { (B, 5), (C, 93) }

When we're at state B…

One successor, C.

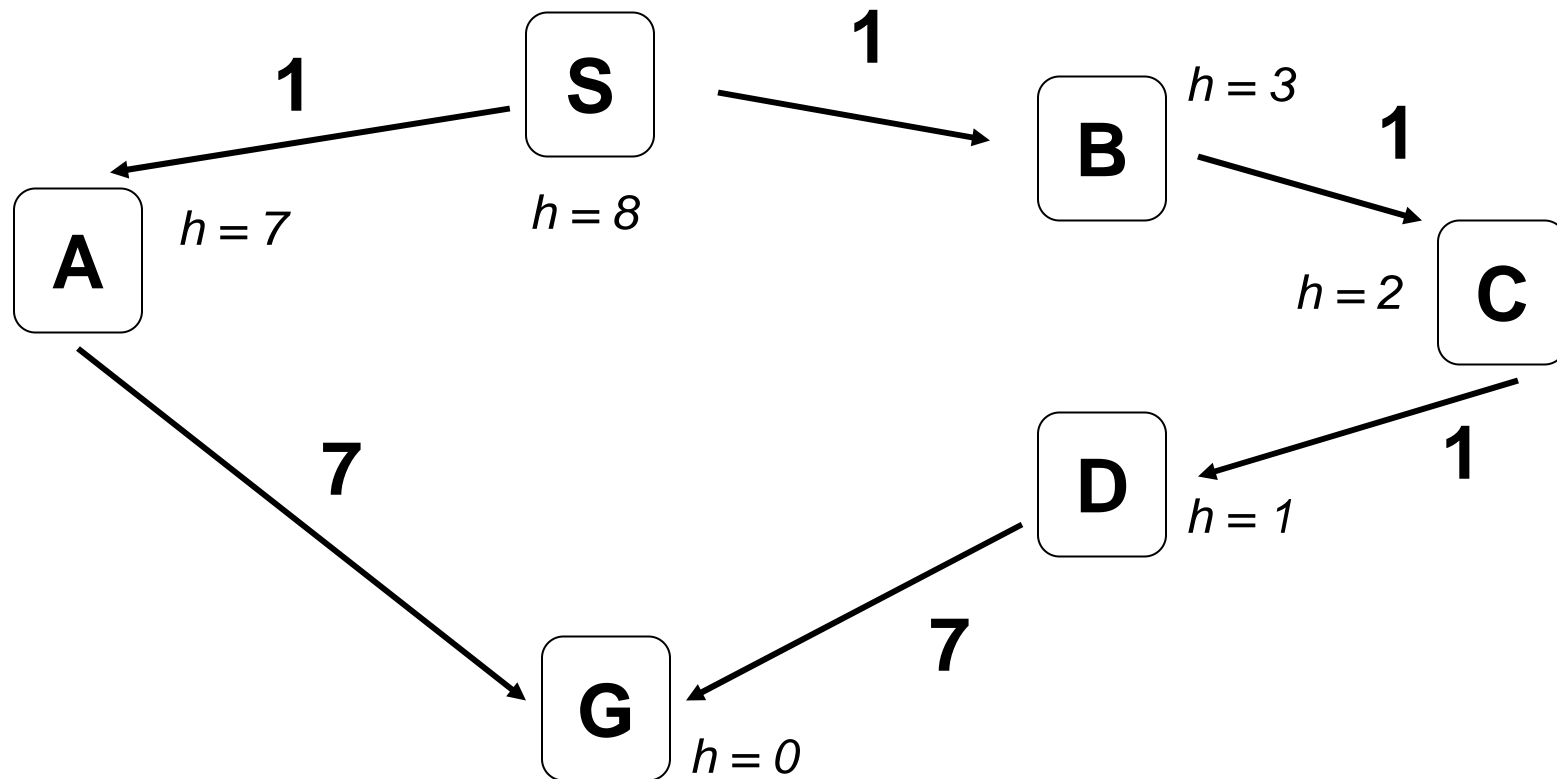
f(C) = g(C) + h(C) = 4 + 1 = 5

PQ = { (C, 5) }

When should A* terminate?

Idea: As soon as it generates a goal state?

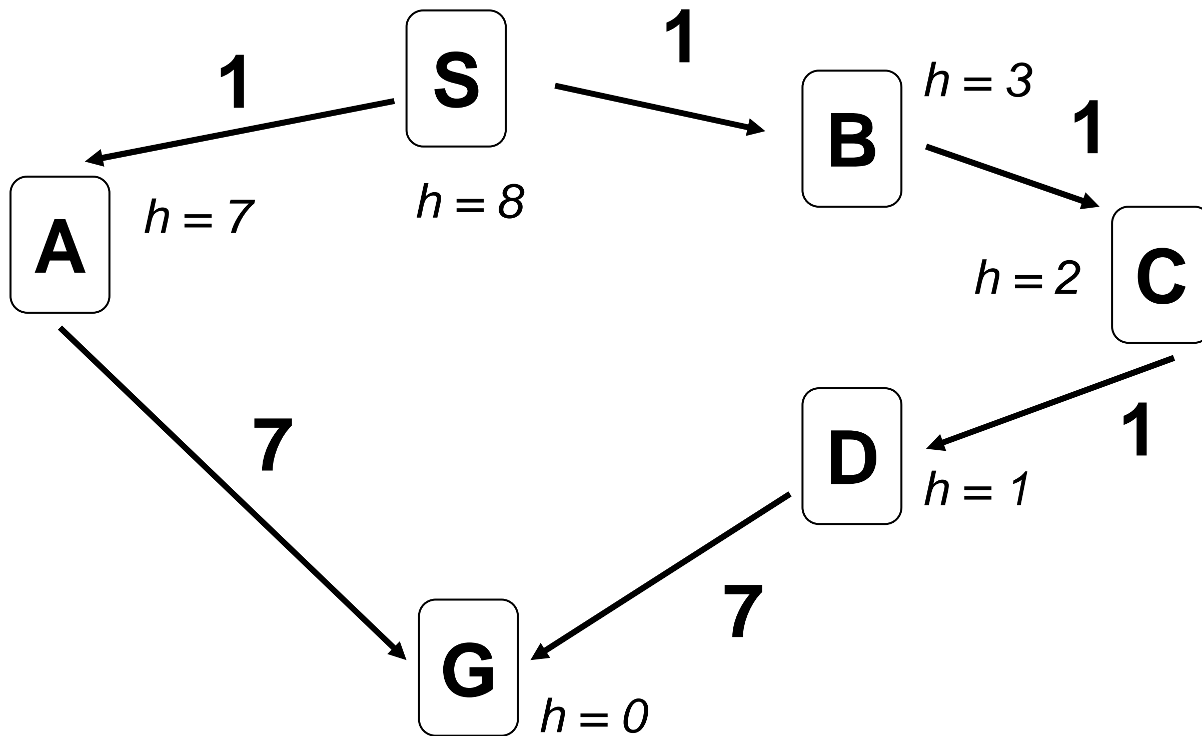
Look at this example:



When should A* terminate?

Idea: As soon as it generates a goal state?

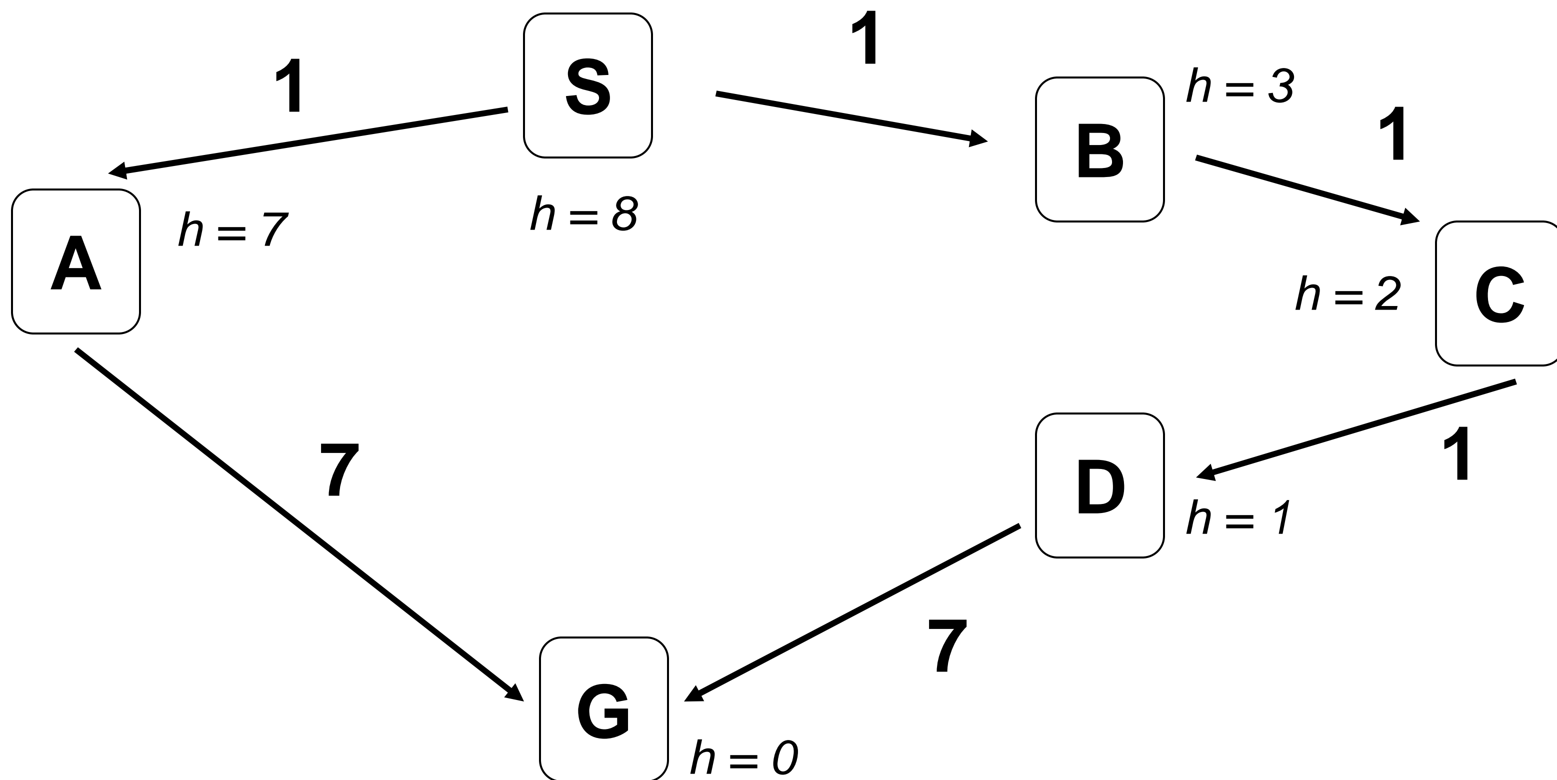
Look at this example:



- Expand S
  - PQ = {(B,4),(A,8)}
- Expand B
  - PQ = {(C,4),(A,8)}
- Expand C
  - PQ = {(D,4),(A,8)}
- Expand D
  - PQ = {(A,8),(G,10)}
- A goal states has been generated

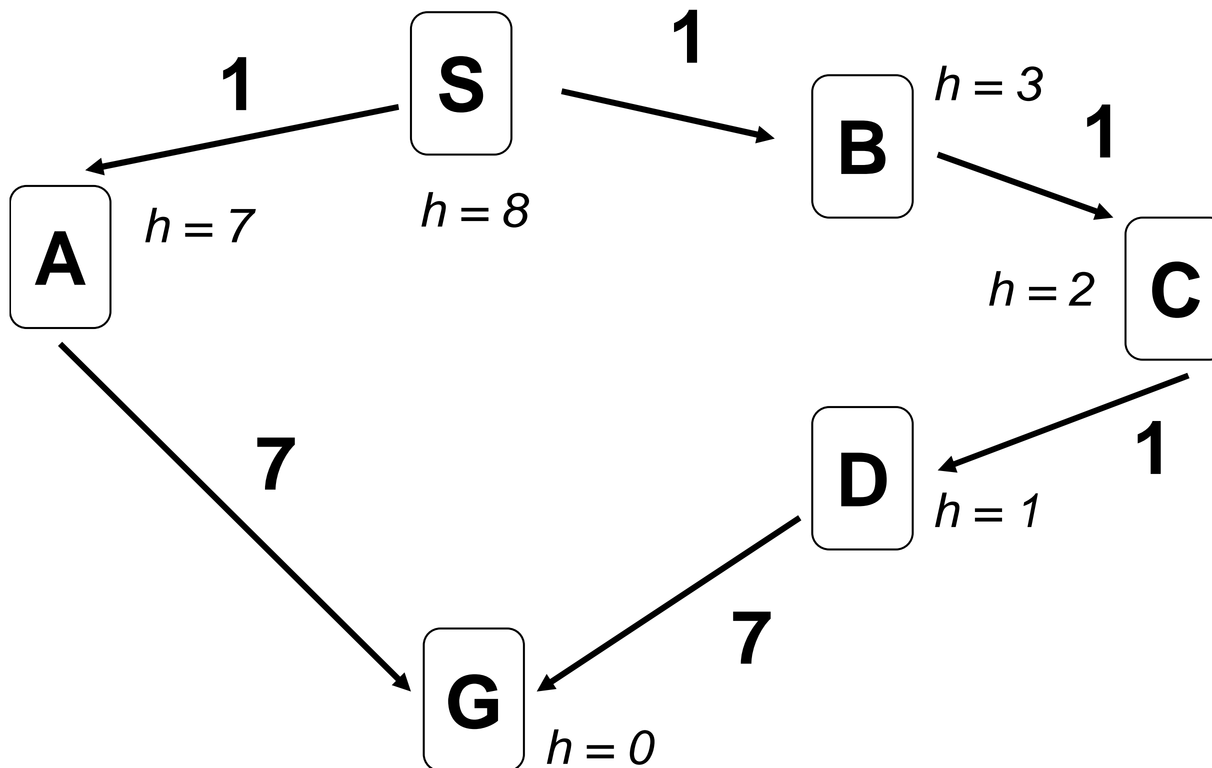
Correct A* termination rule:

A* Terminates Only When a Goal State Is Popped from the Priority Queue



Correct A* termination rule:

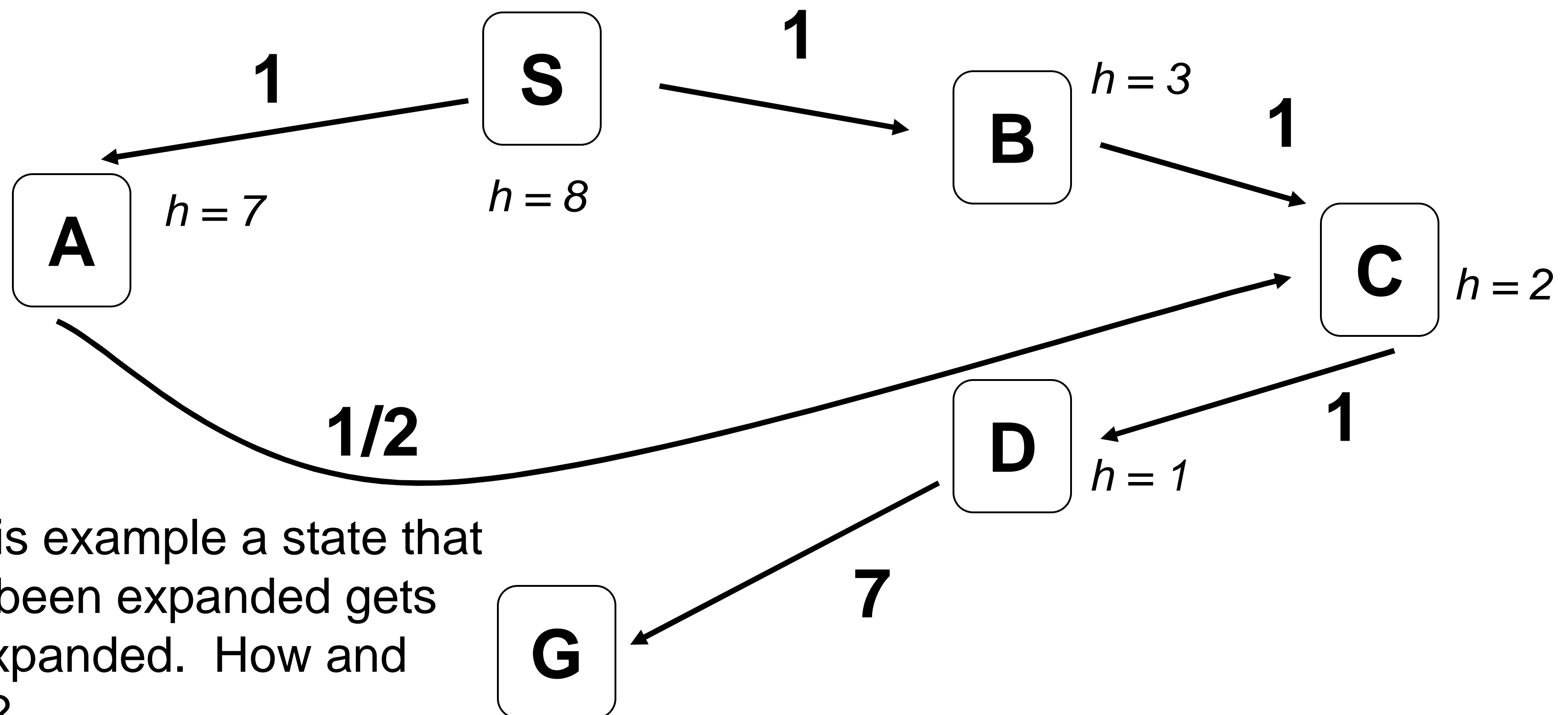
A* Terminates Only When a Goal State Is Popped from the Priority Queue



- Expand S
  - PQ = {(B,4),(A,8)}
- Expand B
  - PQ = {(C,4),(A,8)}
- Expand C
  - PQ = {(D,4),(A,8)}
- Expand D
  - PQ = {(A,8),(G,10)}
- Expand A
  - PQ = { (G,8) }
- Expand G
  - Found shorter path

A* revisiting states

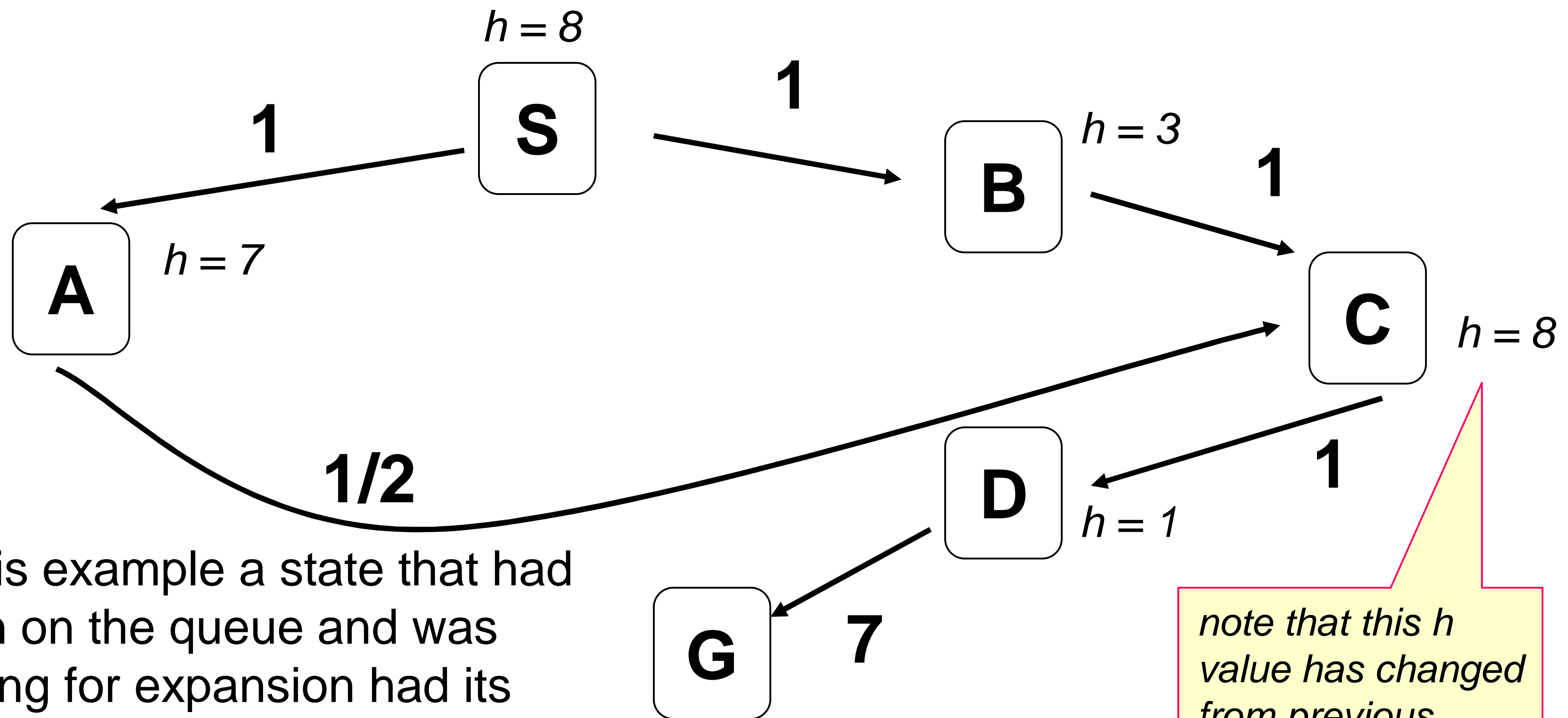
Another question: What if A* revisits a state that was already expanded, and discovers a shorter path?



In this example a state that had been expanded gets re-expanded. How and why?

A* revisiting states

What if A* visits a state that is already on the queue?



In this example a state that had been on the queue and was waiting for expansion had its priority bumped up. How and why?

The A* Algorithm

Reminder: $g(n)$ is cost of shortest known path to n

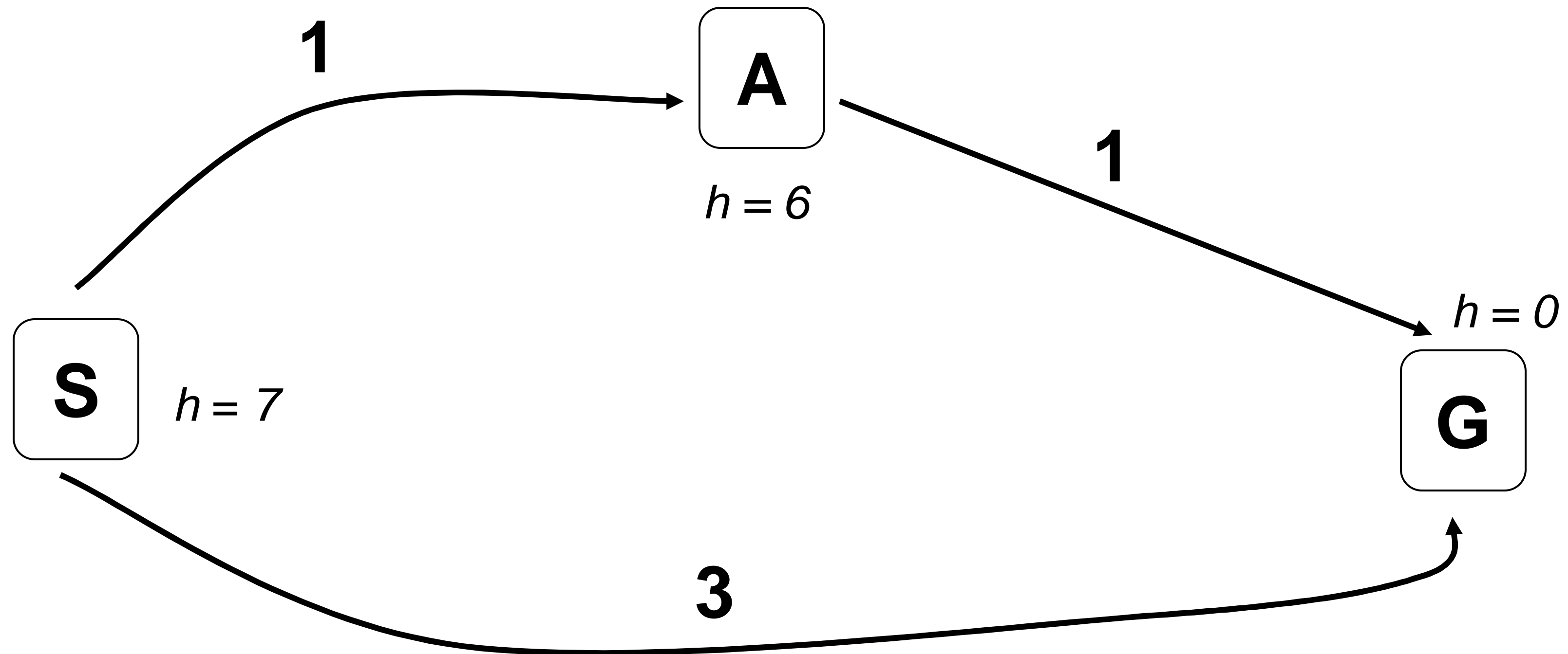
Reminder: $h(n)$ is a heuristic estimate of cost to a goal from n

- Priority queue *PQ* begins empty.
- *V* (= set of previously visited (*state*,*f*,*backpointer*)-triples) begins empty.
- Put *S* into *PQ* and *V* with priority $f(s) = g(s) + h(s)$
- Is *PQ* empty?
  - Yes? Sadly admit there's no solution
  - No? Remove node with lowest $f(n)$ from queue. Call it *n*.
  - If *n* is a goal, stop and report success.
  - "expand" *n* : For each n' in **successors**(n)….
    - Let $f' = g(n') + h(n') = g(n) + cost(n,n') + h(n')$
    - **If** *n'* not seen before, or *n'* previously expanded with $f(n')>f'$, or *n'* currently in *PQ* with $f(n')>f'$
    - **Then** Place/promote *n'* on priority queue with priority *f'* and update *V* to include (*state*=*n'*, *f '*, *BackPtr*=*n*).
    - **Else** Ignore *n'*

= $h(s)$ because $g(\text{start}) = 0$

use sneaky trick to compute $g(n)$

Is A* Guaranteed to Find the Optimal Path?



Nope. And this example shows why not.

Admissible Heuristics

- Write $h^*(n)$ = the true minimal cost to goal from n.
- A heuristic h is admissible if
  $h(n) <= h^*(n)$ for all states n.
- An admissible heuristic is guaranteed never to overestimate cost to goal.
- An admissible heuristic is optimistic.

8-Puzzle Example

Example State

1		5
2	6	3
7	4	8

Goal State

1	2	3
4	5	6
7	8	

Which of the following are admissible heuristics?

- $h(n)$ = Number of tiles in wrong position in state n
- $h(n) = 0$
- $h(n)$ = Sum of Manhattan distances between each tile and its goal location
- $h(n) = 1$

- $h(n) = \min(2, h^*[n])$
- $h(n) = h^*(n)$
- $h(n) = \max(2, h^*[n])$

Proof: A* with Admissible Heuristic Guarantees Optimal Path

- Suppose it finds a suboptimal path, ending in goal state G_1 where $f(G_1) > f^*$ where $f^* = h^*(start)$ = cost of optimal path.
- There must exist a node n which is
  - Unexpanded
  - The path from start to n (stored in the BackPointers(n) values) is the start of a true optimal path

Why must such a node exist? Consider any optimal path s,$n1$,$n2$…goal. If all along it were expanded, the goal would've been reached along the shortest path.

- $f(n) >= f(G_1)$ (else search wouldn't have ended)
- Also $f(n) = g(n) + h(n)$

$= g^*(n) + h(n)$ — because it's on optimal path

$<= g^*(n) + h^*(n)$ — By the admissibility assumption

$= f^*$ — Because n is on the optimal path

So $f^* >= f(n) >= f(G_1)$ — contradicting top of slide

Is A* Guaranteed to Terminate?

i.e. is it complete?

- There are finitely many acyclic paths in the search tree.
- A* only ever considers acyclic paths.
- On each iteration of A* a new acyclic path is generated because:
  - When a node is added the first time, a new path exists.
  - When a node is "promoted", a new path to that node exists. It must be new because it's shorter.
- So the very most work it could do is to look at every acyclic path in the graph.
- So, it terminates.

Comparing Iterative Deepening with A*

From Russell and Norvig

	For 8-puzzle, average number of states expanded over 100 randomly chosen problems in which optimal path is length…		
	…4 steps	…8 steps	…12 steps
Iterative Deepening (see previous slides)	112	6,300	3.6×10^6
A* search using "number of misplaced tiles" as the heuristic	13	39	227
A* using "Sum of Manhattan distances" as the heuristic	12	25	73

Indeed there are only a couple hundred thousand states for the entire eight puzzle

Comments

1. At first sight might look like even “number of misplaced tiles” is a great heuristic. But probably h(state)=0 would also do much much better than ID, so the difference is mainly to do with ID’s big problem of expanding the same state many times, not the use of a heuristic.
2. Judging solely by “number of states expanded” does not account for overhead of maintaining hash tables and priority queue for A*, though it’s pretty clear here that this won’t dramatically change the results.

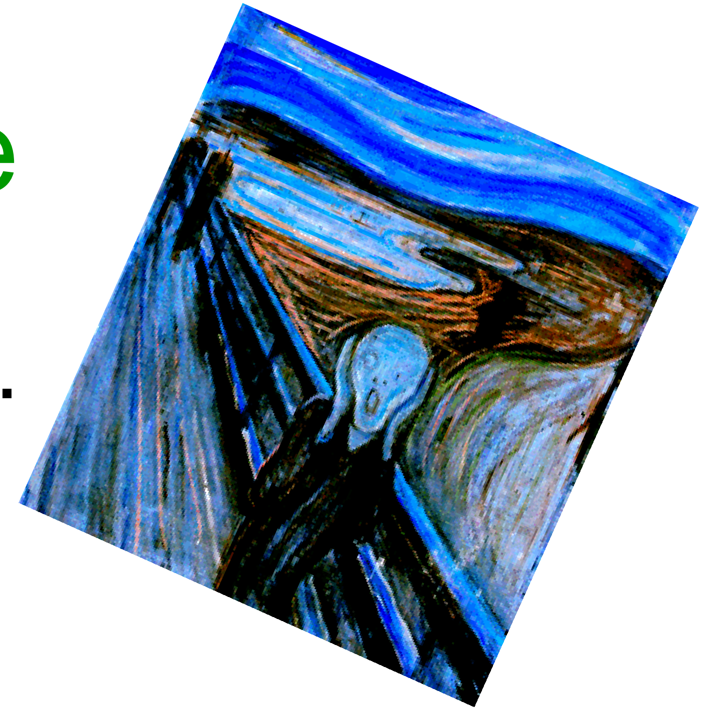
…ig 4.8

…ates …ndomly …ich optimal			
	…	…8 steps	…12 steps
Iterative Deepening (see previous slides)	112	6,300	3.6×10^6
A* search using “number of misplaced tiles” as the heuristic	13	39	227
A* using “Sum of Manhattan distances” as the heuristic	12	25	73

Dominating Heuristics

- Given 2 admissible heuristics for a problem, one might ask whether one is always better?
- **Example:** *Is Manhattan distance always better than number of misplaced tiles for the 8-puzzle?* Yes
- A heuristic h_2 **dominates** h_1 if for all nodes n, $h_2(n) >= h_1(n)$
- Using A* with heuristic h_2 will never expand more nodes than A* with h_1.
- It is always better to use a heuristic function with higher values, provided:
  - it is admissible (never overestimates)
  - and that the computation time for the heuristic is not too large

A* : The Dark Side



- A* can use lots of memory. In principle:

  O(number of states)

- For really big search spaces, A* will run out of memory.

IDA* : Memory Bounded Search

- Iterative deepening A*. Actually, pretty different from A*. Assume costs integer.
  1. Do loop-avoiding DFS, not expanding any node with $f(n) > 0$. Did we find a goal? If so, stop.
  2. Do loop-avoiding DFS, not expanding any node with $f(n) > 1$. Did we find a goal? If so, stop.
  3. Do loop-avoiding DFS, not expanding any node with $f(n) > 2$. Did we find a goal? If so, stop.
  4. Do loop-avoiding DFS, not expanding any node with $f(n) > 3$. Did we find a goal? If so, stop.

  …keep doing this, increasing the $f(n)$ threshold by 1 each time, until we stop.
- This is
  - Complete
  - Guaranteed to find optimal
  - More costly than A* in general.