Problem Solving as Search: Informed (Heuristic) Search

Department of Electrical and Electronics Engineering
Spring 2005
Dr. Afşar Saranlı

Thanks to Professor Andrew W. Moore (Carnegie Mellon University) http://www.cs.cmu.edu/~awm/tutorials
Also: Artificial Intelligence: A Modern Approach, 2nd Ed., Russel & Norvig

Overview

- Best First “Greedy” Search -> “Heuristic”?
- Problems with “Best First Greedy” search.
- Good trick: take account of your cost of getting to the current state: A* Search!
- When should the search stop?
- “Admissible” heuristics
- A* : Completeness
- A* : Termination
- A* : The Dark Side
- Saving masses of memory with IDA* (Iterative Deepening A*)

Informed (Heuristic) Search

Basic Idea:

- We are "informed" about the structure of the state space.
- We do not need to expand everything blindly.
- ***MORE:*** It may not be possible to expand everything!!

Informed (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 lowest 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.

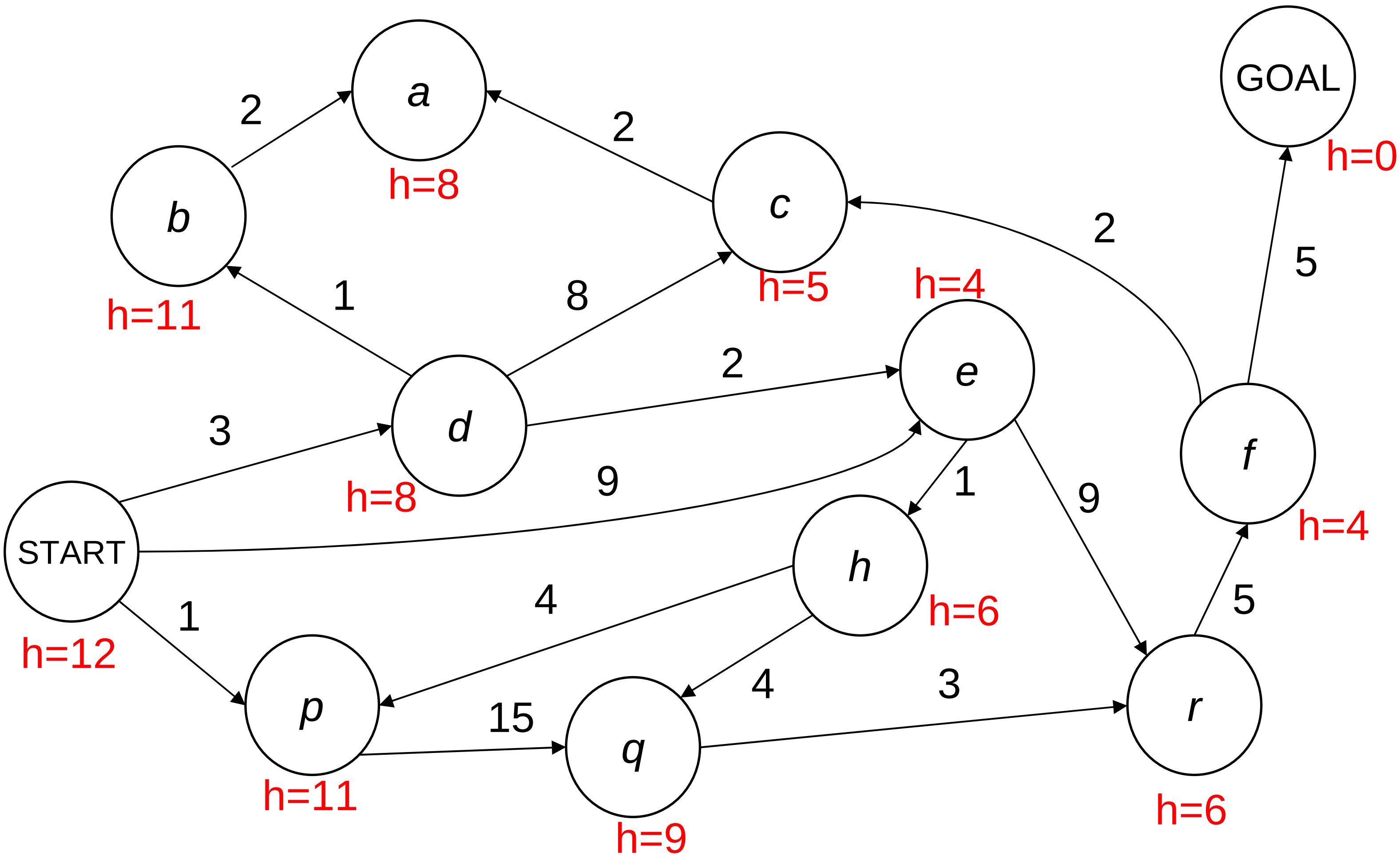
Normally, the expansion of a node is based on the ***Evaluation Function f(n)***

Best First "Greedy" Search

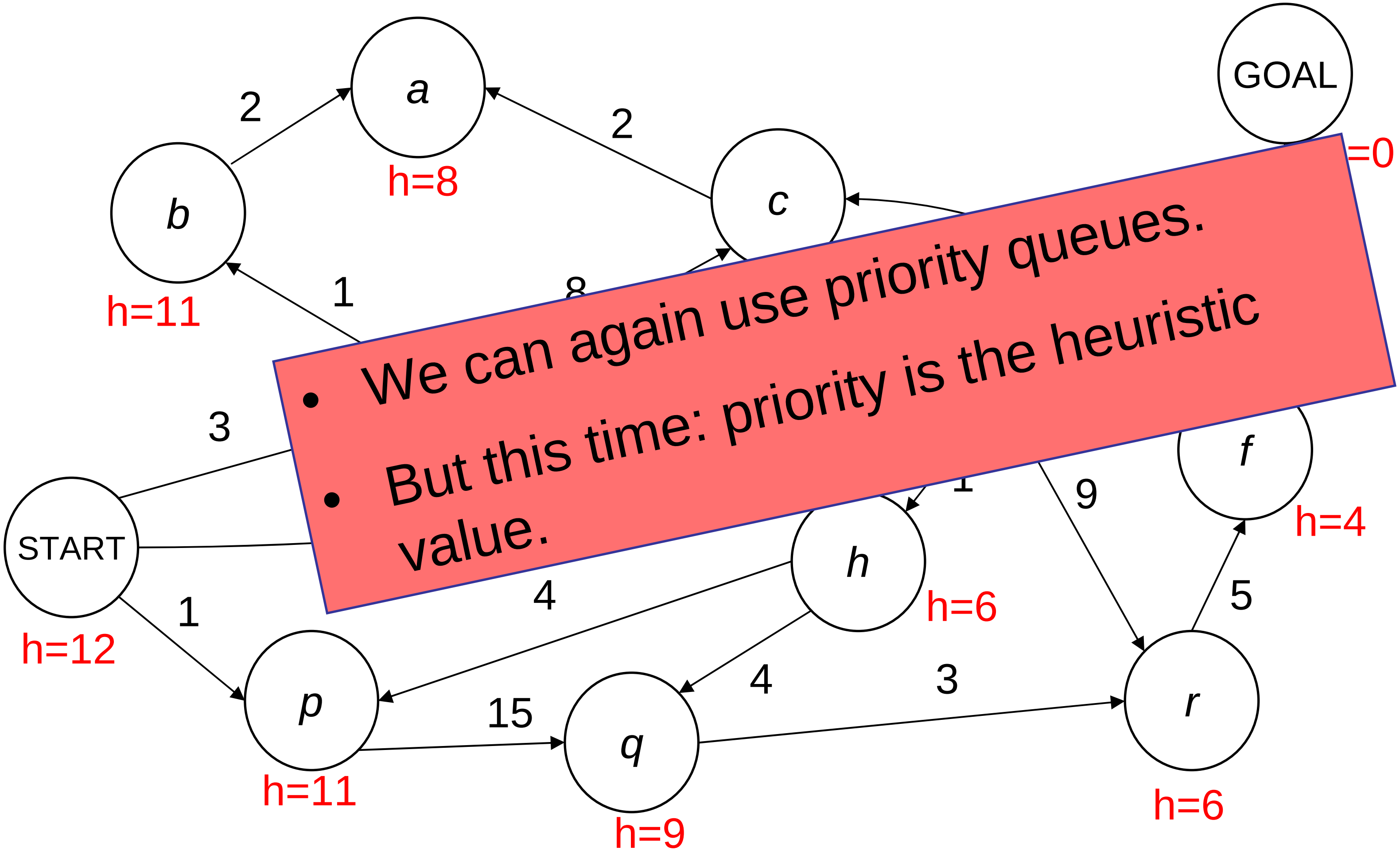
- Simplest Heuristic search
- Needs a Heuristic $g(n)$
- "Expand the Node with the Minimum Heuristic value first"

I.e., we have: $\boldsymbol{f(n) = g(n)}$

Example: Euclidian Heuristic



Example: Euclidian Heuristic



- We can again use priority queues.
- But this time: priority is the heuristic value.

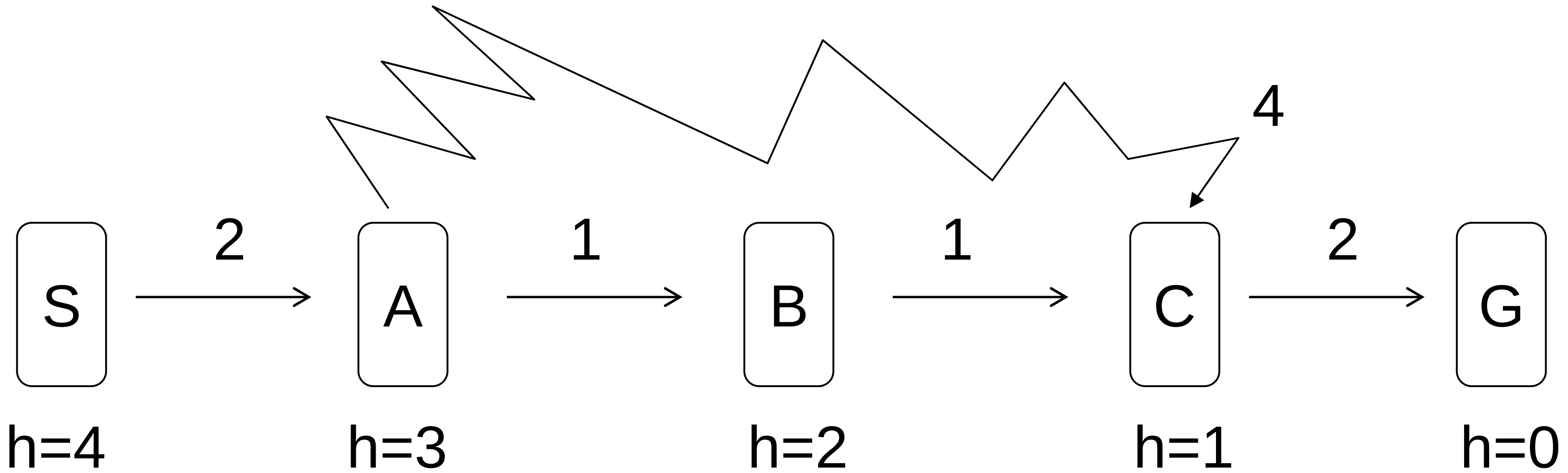
Best First "Greedy" Search

```
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**	$O(min(N,B^{LMAX}))$	$O(min(N,B^{LMAX}))$

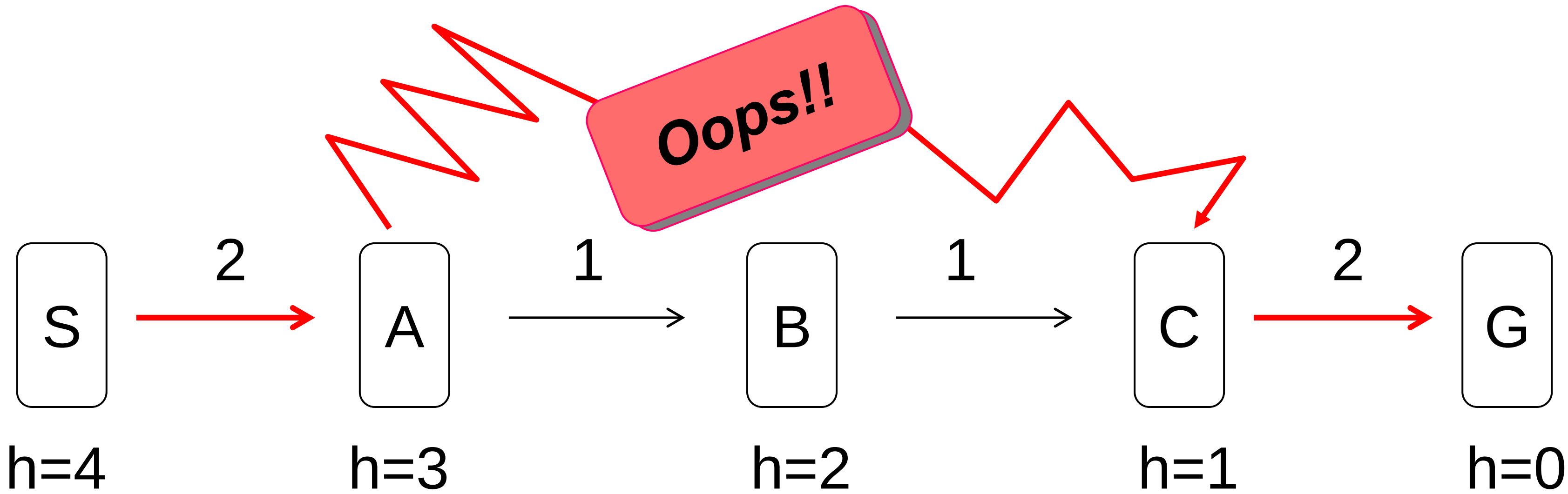
A few improvements to this algorithm can make things much better. It's a little thing we like to call: A*….

Let's make "Best First Greedy" look stupid!



- What would "Best First Greedy" do?

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* Search: 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

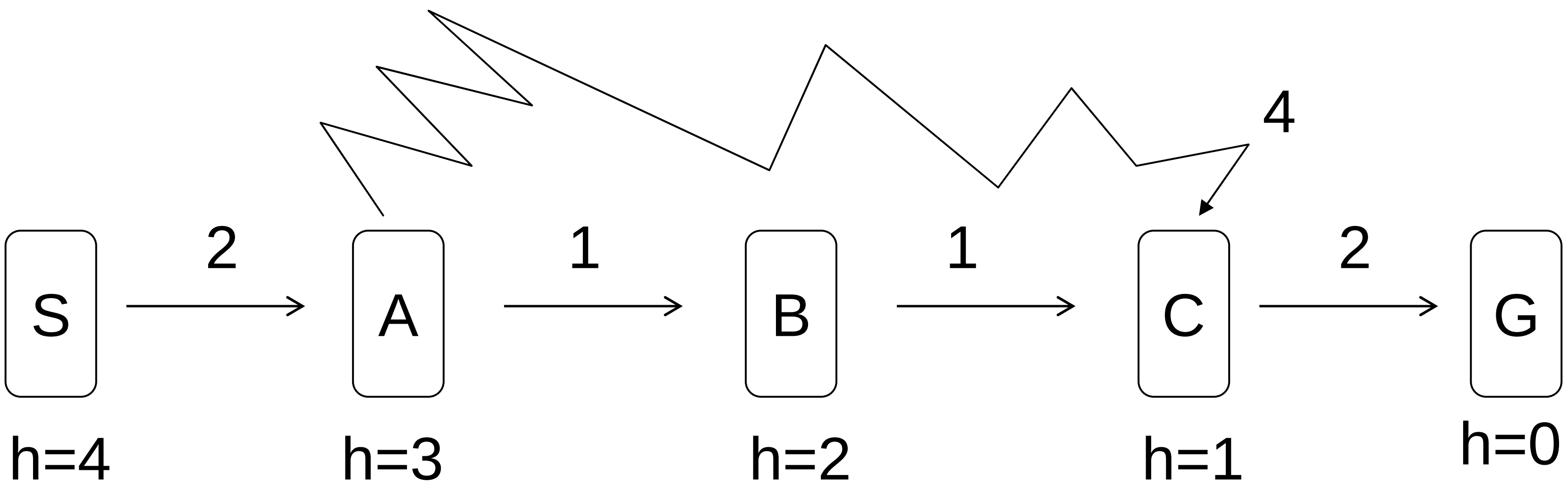
$$\text{(Cost of getting to } n') + h(n') \tag{1}$$

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

and then define…

$$f(n) = g(n) + h(n) \tag{3}$$

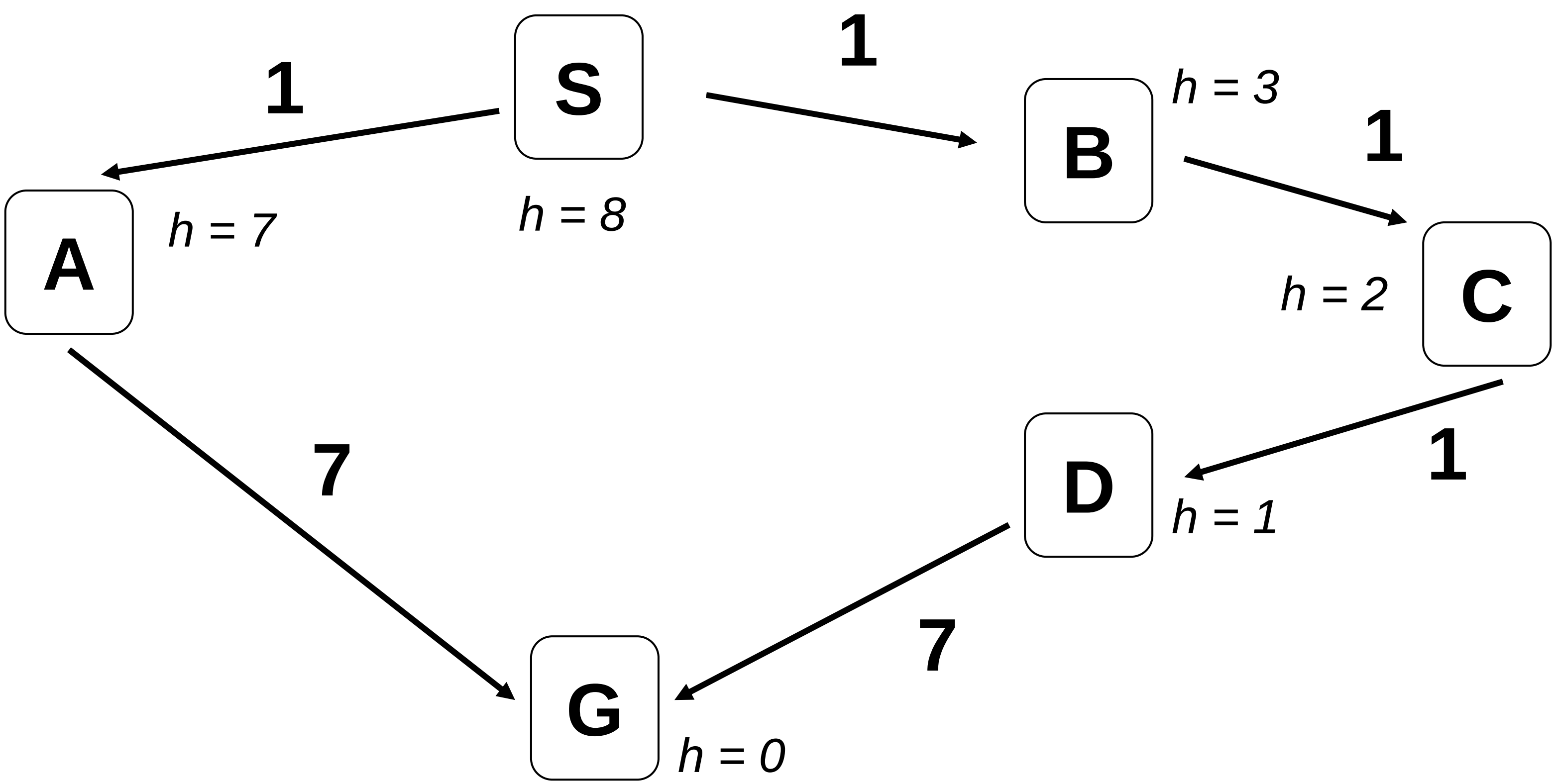
A* Search: Looking Non-Stupid



When should A* Terminate?

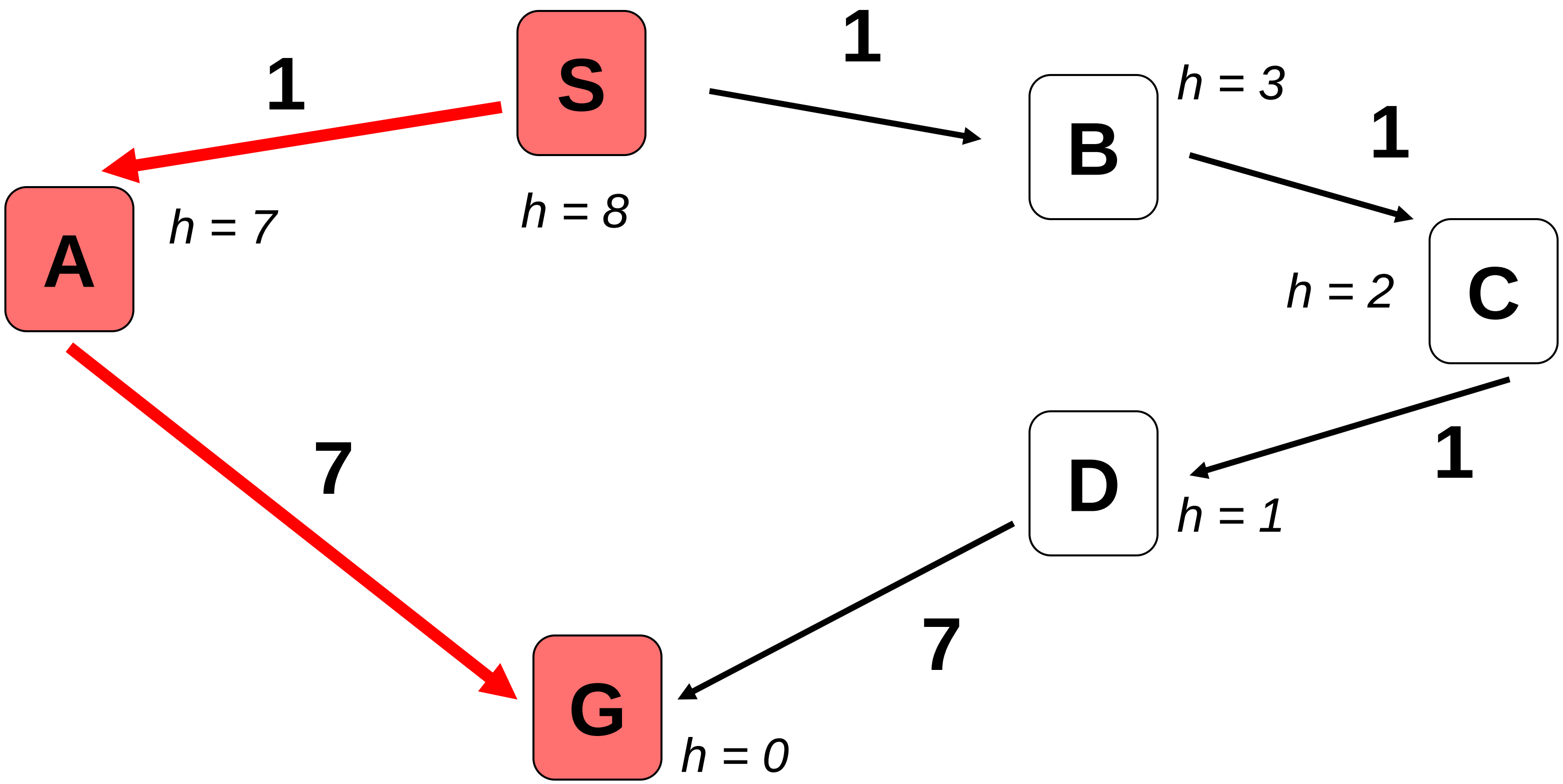
Idea: As soon as it generates a goal state?

Look at this example:



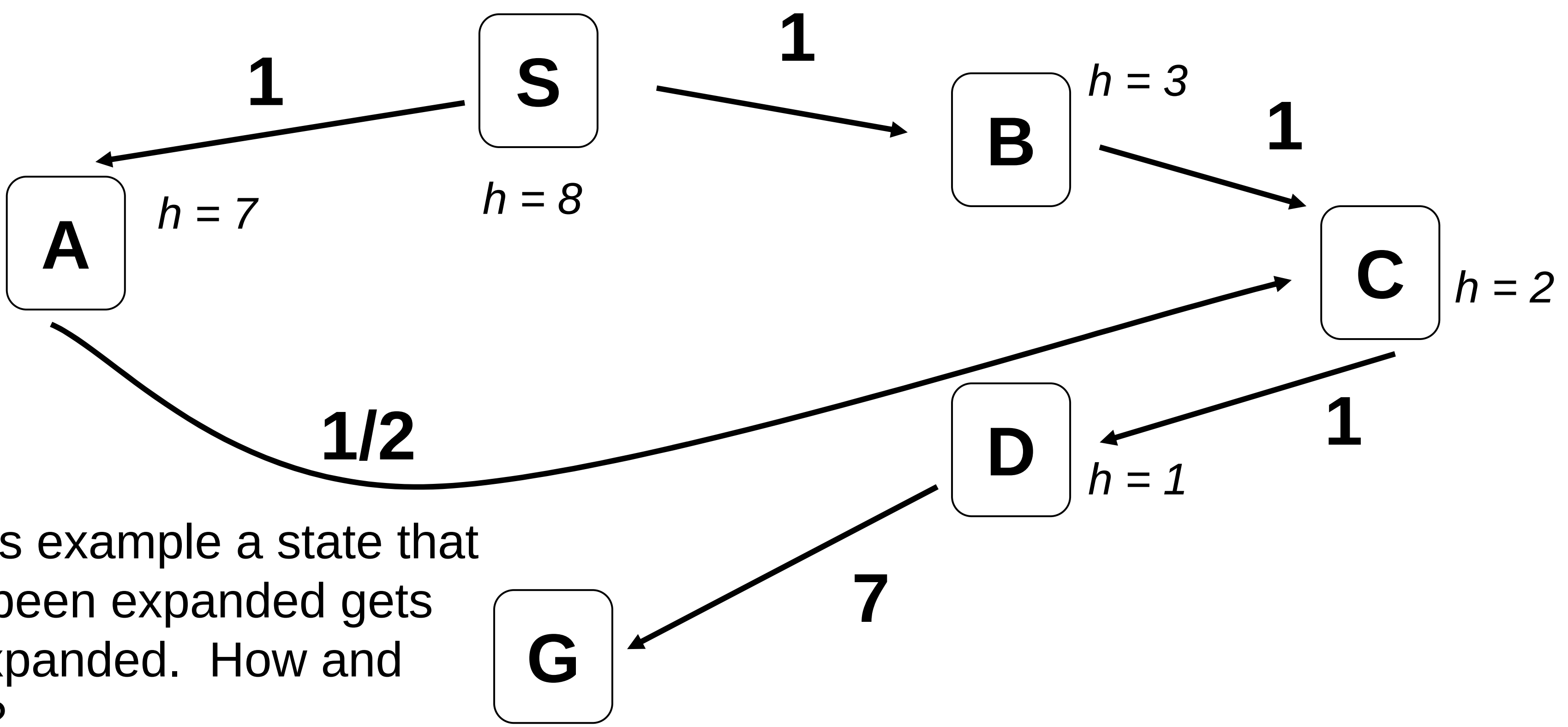
Correct A* Termination Rule

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



A* Revisiting States

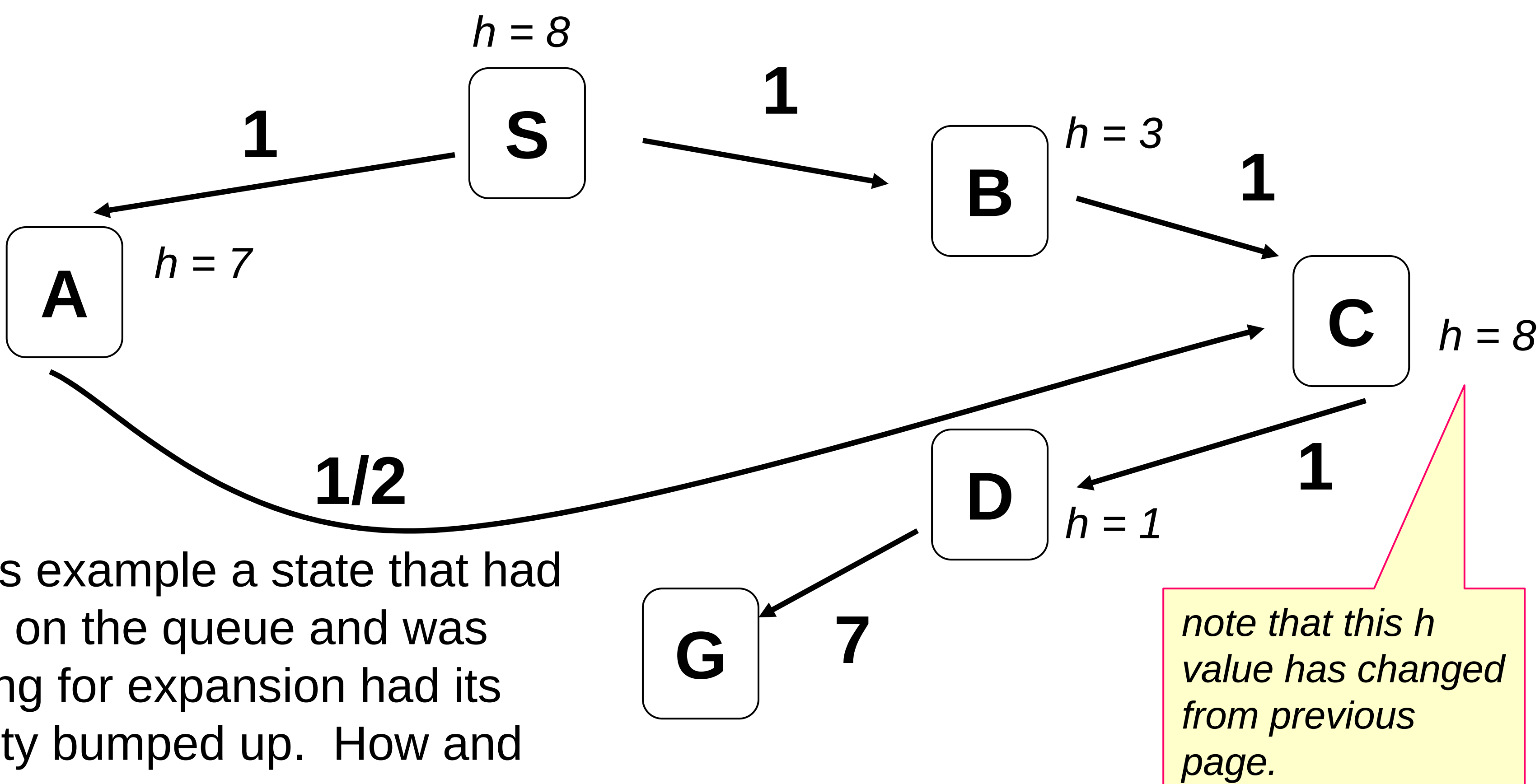
Another question: What if A* revisits a state that was already expanded, and discovers a shorter path?



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?

Finally: The A* Search Algorithm

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

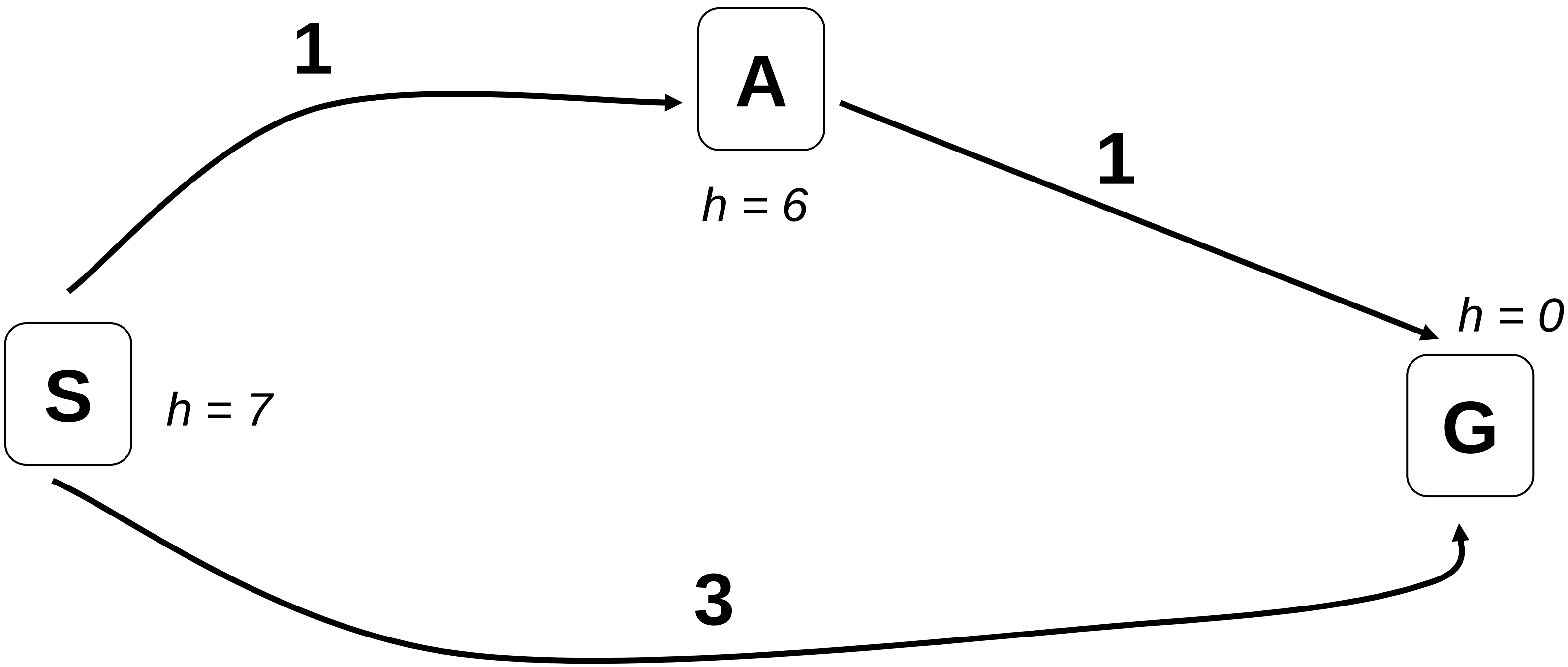
Reminder: *h*(*n*) is a heuristic estimate of cost to a goal from n

- Priority queue *PQ* begins empty.
- *V* (= set of previously visited (*state*,*f*,*backpointer*)-triples) begins empty.
- Put *S* into *PQ* and *V* with priority $f(s) = g(s) + h(s)$

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

- 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'*

Is A* Guaranteed to find the Optimal Path?



Nope. And this example shows why not.

Admissible Heuristic

- 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
- $h(n) = 1$

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

Path Optimality

- A* with Admissible heuristic guarantees optimal path.
- Proof?
- Think about it.

Is A* Guaranteed to Terminate?

- 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 worst we could do is to look at every acyclic path in the graph.
- So, it terminates.

i.e. is it complete?

Compare Iterative Deepening with A*

From Russell and Norvig, Page 107, Fig 4.8

	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	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

A* Search: 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.

Optimality Proof: By Contradiction

- 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

- $f(n) >= f(G_1)$ (else search wouldn't have en[illegible])
- Also $f(n) = g(n) + h(n)$

  $= g^*(n) + h(n)$

  $<= g^*(n) + h^*(n)$

  $= f^*$

because it's on optimal path

By the admissibility assumption

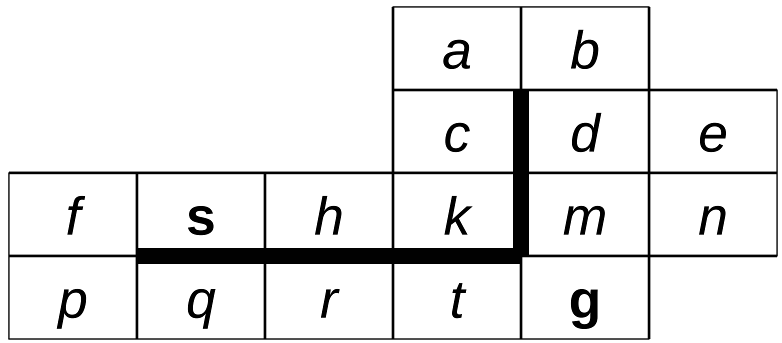
Because n is on the optimal path

So $f^* >= f(n) >= f(G_1)$

contradicting top of slide

Example: Part 1

In the following maze the successors of a cell include any cell directly to the east, south, west or north of the current cell except that no transition may pass through the central barrier. for example *successors*(*m*) = { *d* , *n* , *g* }.

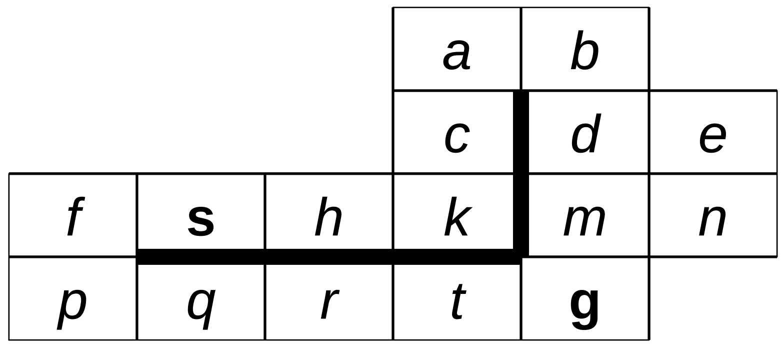


The search problem is to find a path from **s** to **g**. We are going to examine the order in which cells are expanded by various search algorithms. for example, one possible expansion order that breadth first search might use is:

s *h f k p c q a r b t d* **g**

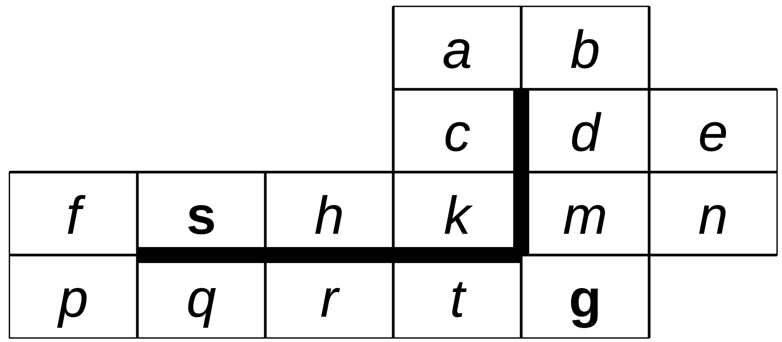
There are other possible orders depending on which of two equal-distance-from-start states happen to be expanded first. For example **s** *f h p k c q r a t b* **g** is another possible answer. *continued->*

Example: Part 1



Assume you run **depth-first-search** until it expands the goal node. Assume that you always try to expand East first, then South, then West, then North. Assume your version of depth first search avoids loops: it never expands a state on the current path. What is the order of state expansion?

Example Part 2: Exercise



Next, you decide to use a Manhattan Distance Metric heuristic function

h(*state*) = shortest number of steps from *state* to **g** if there were no barriers

So, for example, *h*(*k*) = 2, *h*(**s**) = 4, *h*(**g**) = 0

Assume you now use best-first greedy search using heuristic *h* (a version that never re-explores the same state twice). Again, give all the states expanded, in the order they are expanded, until the algorithm expands the goal node.

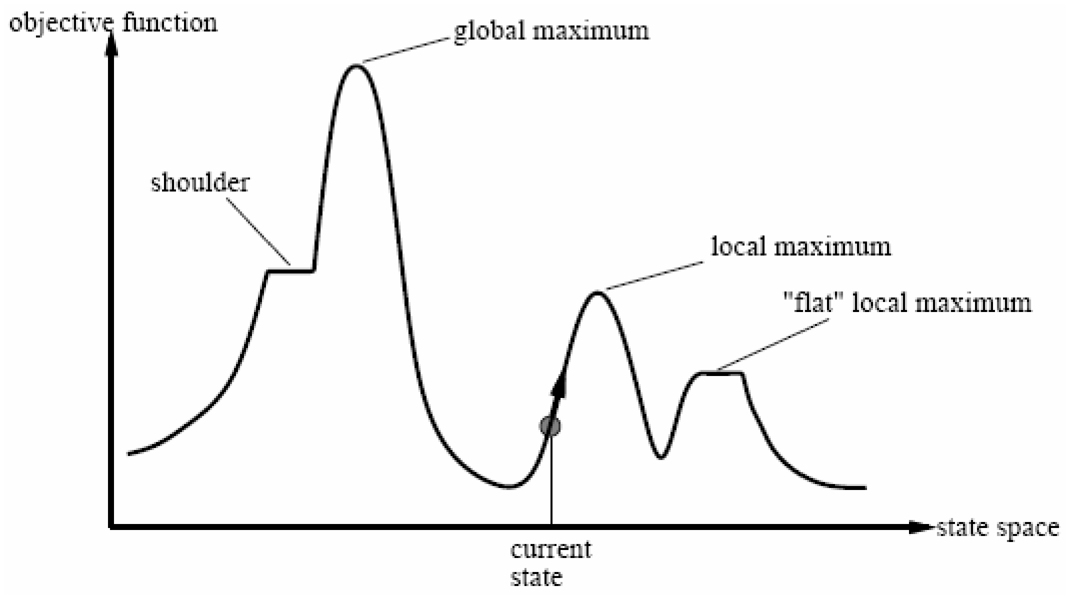
Finally, assume you use A* search with heuristic *h*, and run it until it terminates using the conventional A* termination rule. Again, give all the states expanded, in the order they are expanded. (Note that depending on the method that A* uses to break ties, more than one correct answer is possible).

Local Search Algorithms

- Assume path to Goal does not matter!
- Then: **Local Search** with a single **current state.**
  - Not systematic,
  - Requires very little memory
  - Reasonable solutions in large or infinite (continuous) state spaces
- Also: Solution to pure Optimization Problems!
- Goal defined according to an **Objective Function.**

Local Search Algorithms

- Define: ***State Space Landscape*** or ***Objective Function landscape***



Local Search Algorithms

- Hill Climbing (alternative: Gradient Descent)
- Simulated Annealing Search
- Local Beam Search
- Genetic Algorithms

- Optimization in Continuous Spaces: **Classic Optimization Literature**

Assignment:

- Read Chapter 4 in Russel & Norvig.