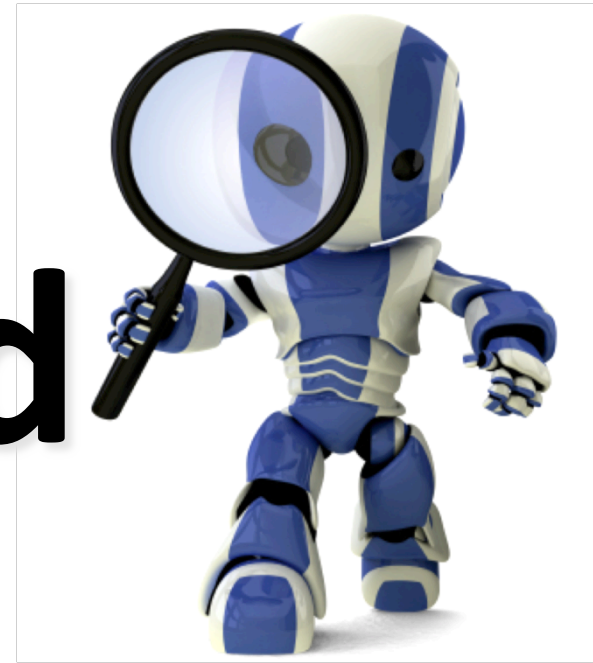


# Informed Search Chapter 4 (a)



Some material adopted from notes  
by Charles R. Dyer, University of  
Wisconsin-Madison

# Today's class

- Heuristic search
- Best-first search
  - Greedy search
  - Beam search
  - A, A\*
  - Examples
- Memory-conserving variations of A\*
- Heuristic functions

# Big idea: heuristic

## Merriam-Webster's Online Dictionary

Heuristic (pron. \hyu-'ris-tik\): adj. [from Greek *heuriskein* to discover] involving or serving as an aid to learning, discovery, or problem-solving by experimental and especially trial-and-error methods

## The Free On-line Dictionary of Computing (15Feb98)

heuristic 1. <programming> A rule of thumb, simplification or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood. Unlike algorithms, heuristics do not guarantee feasible solutions and are often used with no theoretical guarantee. 2. <algorithm> approximation algorithm.

## From WordNet (r) 1.6

heuristic adj 1: (CS) relating to or using a heuristic rule 2: of or relating to a general formulation that serves to guide investigation [ant: algorithmic]  
n : a commonsense rule (or set of rules) intended to increase the probability of solving some problem [syn: heuristic rule, heuristic program]

# Informed methods add domain-specific information

- Add domain-specific information to select best path along which to continue searching
- Define heuristic function,  $h(n)$ , that estimates *goodness* of node  $n$
- $h(n)$  = **estimated cost** (or distance) of minimal cost path from  $n$  **to a goal state**.
- Heuristic function is an estimate, based on domain-specific information, computable from current state description, of how close we are to a goal

# Heuristics

- **All domain knowledge** used in search is encoded in the **heuristic function,  $h()$**
- A **weak method** due to limited way domain-specific information used to solve problem
- Examples
  - Missionaries & Cannibals: # people on starting river bank
  - 8-puzzle: number of tiles out of place
  - 8-puzzle: sum of distances each tile is from its goal position
- In general
  - $h(n) \geq 0$  for all nodes  $n$
  - $h(n) = 0$  implies that  $n$  is a goal node
  - $h(n) = \infty$  implies  $n$  is a dead-end that can't lead to a goal

# Weak vs. strong methods

- *Weak methods* are extremely general methods not tailored to a specific situation or domain
- Examples include
  - **Generate and test:** generate solution candidates and test until you find one
  - **Means-ends analysis:** represent current situation & goal, then seek ways to shrink the differences between them
  - **Space splitting:** list the possible solutions to a problem, then try to rule out classes of these possibilities
  - **Subgoaling:** split a large problem into smaller ones that can be solved one at a time
- *Weak* because they don't use more powerful domain-specific heuristics

# Heuristics for 8-puzzle

Current State

1	2	3
4	5	6
7		8

Goal State

1	2	3
4	5	6
7	8	

*The number of misplaced tiles (not including the blank)*

1	2	3
4	5	6
7	8	8

In this case, only “8” is misplaced, so the heuristic function evaluates to 1.

N	N	N
N	N	N
N	Y	

In other words, the heuristic is *telling* us, that it *thinks* a solution might be available in just 1 more move.

# Heuristics for 8-puzzle

**Manhattan Distance** (not including the blank)

Current State

3	2	8
4	5	6
7	1	

Goal State

1	2	3
4	5	6
7	8	

3	→	<u>3</u>

2 spaces

	←	8
	↓	
	<u>8</u>	

3 spaces

<u>1</u>	←	
	↑	
	1	

3 spaces

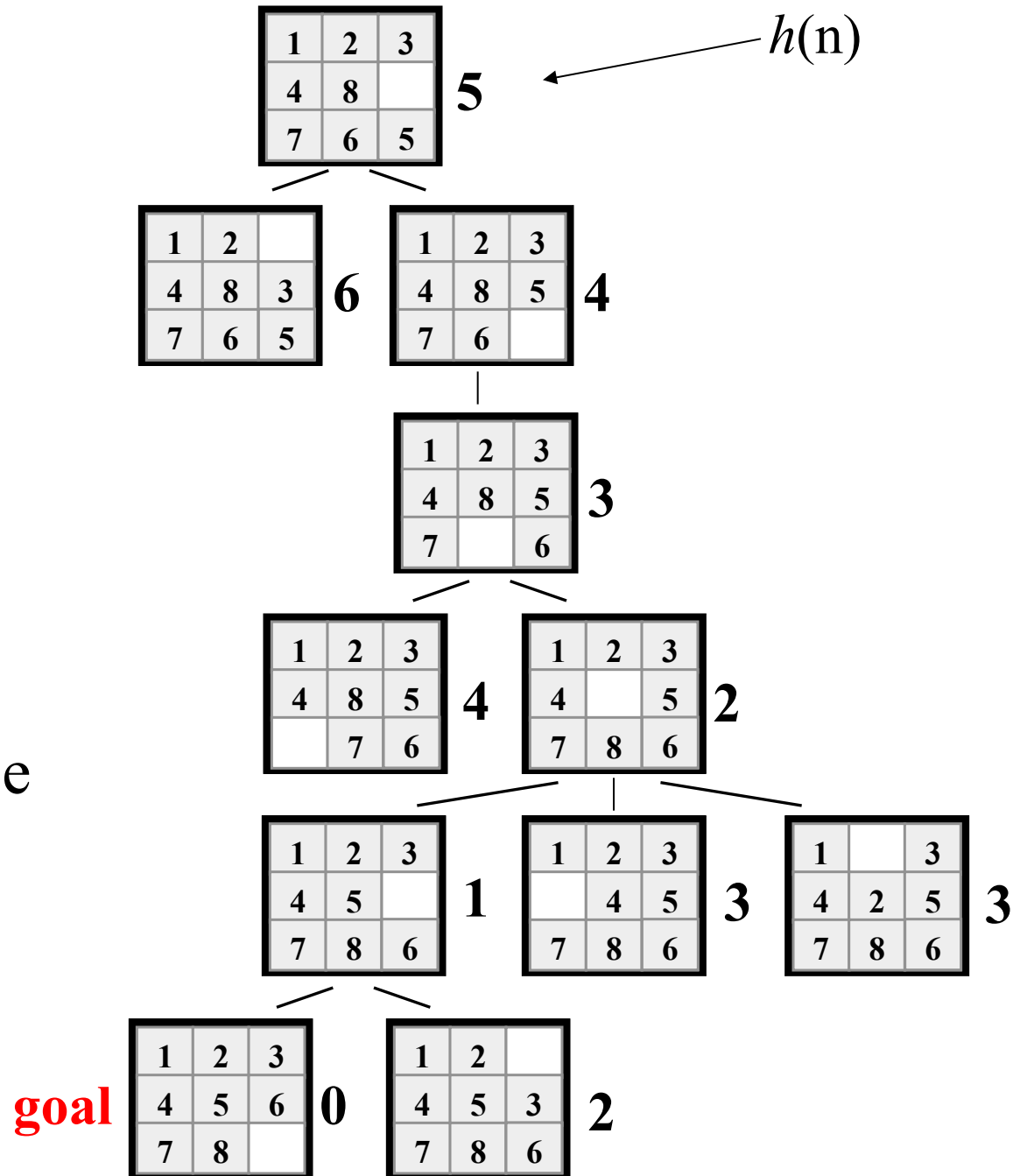
- In this case, only the “3”, “8” and “1” tiles are misplaced, by 2, 3, and 3 squares respectively, so the heuristic function evaluates to 8.
- In other words, the heuristic is *telling* us, that it *thinks* a solution is available in just 8 more moves.
- The misplaced heuristic’s value is 3.

**Total 8**



We can use heuristics to guide search.

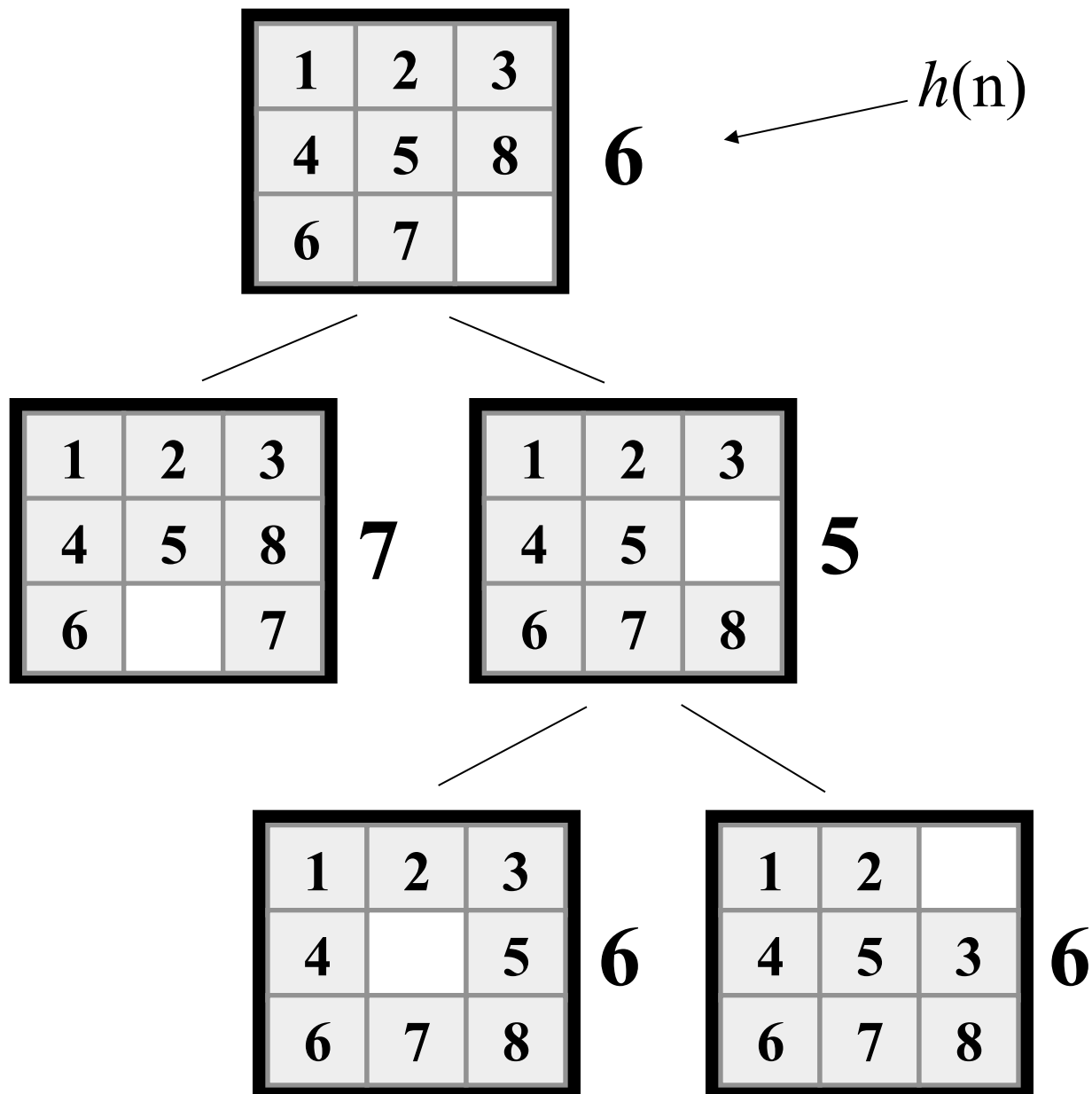
In this [hill climbing](#) example, the Manhattan Distance heuristic helps us quickly find a solution to the 8-puzzle.



In this example,  
hill climbing does  
not work!

All the nodes on  
the fringe are  
taking a step  
“backwards”  
(local minima)

Note that this  
puzzle *is* solvable  
in just 12 more  
steps.

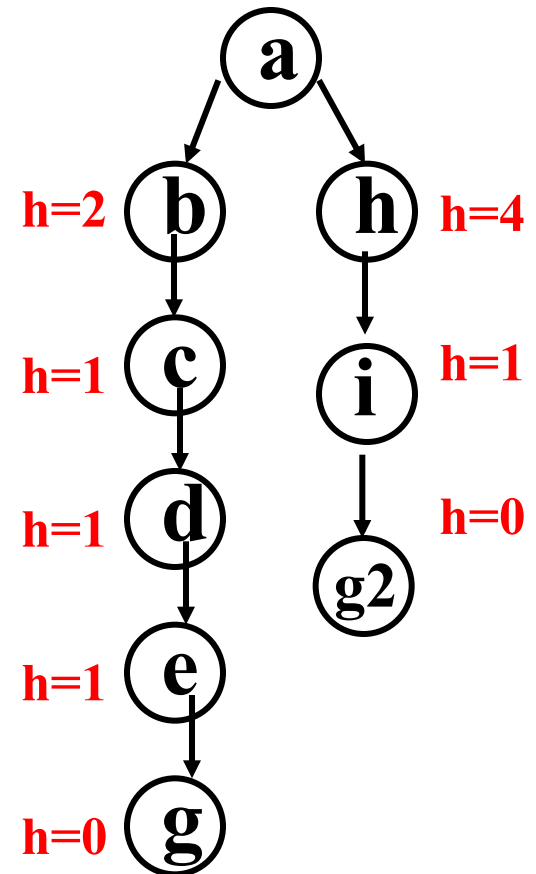


# Best-first search

- Search algorithm that optimizes depth-first search by expanding most promising node chosen according to heuristic rule
- Order nodes on nodes list by increasing value of an evaluation function,  $f(n)$ , incorporating domain-specific information
- This is a generic way of referring to the class of informed methods

# Greedy best first search search

- Uses evaluation function  $f(n) = h(n)$ , sorting nodes by increasing values of  $f$
- Selects node to expand appearing **closest** (i.e., “greedy”) to a goal (i.e., select node with smallest  $f$  value)
- Not complete
- Not admissable, as in the example
  - Assuming all arc costs are one, then greedy search finds goal  $g$ , which has solution cost of five
  - However, optimal solution is path to goal with cost three



# Beam search

- Use evaluation function  $f(n)$ , but maximum size of the nodes list is  $k$ , a fixed constant
- Only keep  $k$  best nodes as candidates for expansion, discard rest
- $k$  is the *beam width*
- More space efficient than greedy search, but may discard nodes on a solution path
- As  $k$  increases, approaches best first search
- Not complete
- Not admissible (optimal)



# Algorithm A

- 1 Put the start node  $S$  on the nodes list, called OPEN
- 2 If OPEN is empty, exit with failure
- 3 Select node in OPEN with minimal  $f(n)$  and place on CLOSED
- 4 If  $n$  is a goal node, collect path back to start and stop
- 5 Expand  $n$ , generating all its successors and attach to them pointers back to  $n$ . For each successor  $n'$  of  $n$ 
  - 1 If  $n'$  is not already on OPEN or CLOSED
    - put  $n'$  on OPEN
    - compute  $h(n')$ ,  $g(n')=g(n)+c(n,n')$ ,  $f(n')=g(n')+h(n')$
  - 2 If  $n'$  is already on OPEN or CLOSED and if  $g(n')$  is lower for the new version of  $n'$ , then:
    - Redirect pointers backward from  $n'$  on path with lower  $g(n')$
    - Put  $n'$  on OPEN

# Algorithm A\*

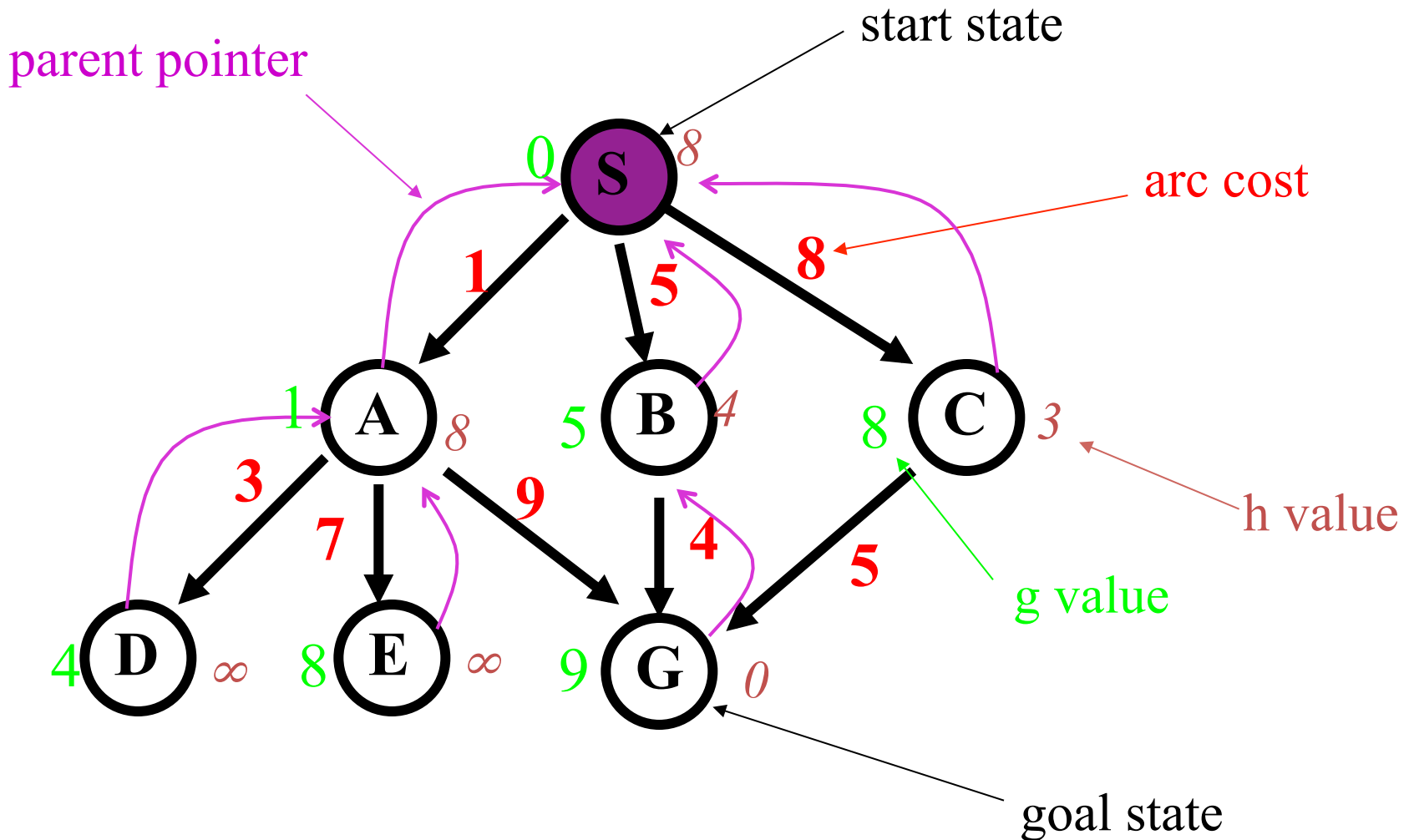
- Pronounced “a star”
- Algorithm A with constraint that  $h(n) \leq h^*(n)$
- $h^*(n) = \textit{true cost of minimal cost path}$  from  $n$  to a goal
- $h$  is **admissible** when  $h(n) \leq h^*(n)$  holds
- Using an admissible heuristic guarantees that 1st solution found will be an optimal one
- A\* is **complete** whenever branching factor is finite and every operator has a fixed positive cost
- A\* is **admissible**



# Observations on A

- **Perfect heuristic:** If  $h(n) = h^*(n)$  for all  $n$ , then only nodes on an optimal solution path are expanded; no extra work is performed
- **Null heuristic:** If  $h(n) = 0$  for all  $n$ , then it is an admissible heuristic and  $A^*$  acts like uniform-cost search
- **Better heuristic:** If  $h_1(n) < h_2(n) \leq h^*(n)$  for all non-goal nodes, then  $h_2$  is a *better* heuristic than  $h_1$ 
  - If  $A_1^*$  uses  $h_1$ , and  $A_2^*$  uses  $h_2$ , then every node expanded by  $A_2^*$  is also expanded by  $A_1^*$
  - i.e.,  $A_1$  expands at least as many nodes as  $A_2^*$
  - We say that  $A_2^*$  is *better informed* than  $A_1^*$
- The closer  $h$  to  $h^*$ , the fewer extra nodes expanded

# Example search space



# Example

<b>n</b>	<b>g(n)</b>	<b>h(n)</b>	<b>f(n)</b>	<b>h*(n)</b>
S	0	8	8	9
A	1	8	9	9
B	5	4	9	4
C	8	3	11	5
D	4	inf	inf	inf
E	8	inf	inf	inf
G	9	0	9	0

- $h^*(n)$  is (hypothetical) perfect heuristic (an oracle)
- Since  $h(n) \leq h^*(n)$  for all  $n$ ,  $h$  is admissible (optimal)
- Optimal path =  $S B G$  with cost 9

# Greedy search

$$f(n) = h(n)$$

<b>node expanded</b>	<b>nodes list</b>
	{ S(8) }
S	{ C(3) B(4) A(8) }
C	{ G(0) B(4) A(8) }
G	{ B(4) A(8) }

- Solution path found is S C G, 3 nodes expanded.
- See how fast the search is!! But it is NOT optimal.

# A\* search

$$f(n) = g(n) + h(n)$$

node exp.	nodes list
	{ S(8) }
S	{ A(9) B(9) C(11) }
A	{ B(9) G(10) C(11) D(inf) E(inf) }
B	{ G(9) G(10) C(11) D(inf) E(inf) }
G	{ C(11) D(inf) E(inf) }

- Solution path found is S B G, 4 nodes expanded..
- Still pretty fast. And optimal, too.

# Proof of the optimality of $A^*$

- Assume that  $A^*$  has selected  $G_2$ , a goal state with a suboptimal solution, i.e.,  $g(G_2) > f^*$
- Proof by contradiction shows its impossible
  - Choose a node  $n$  on an optimal path to  $G$
  - Because  $h(n)$  is admissible,  $f^* \geq f(n)$
  - If we choose  $G_2$  instead of  $n$  for expansion, then  $f(n) \geq f(G_2)$
  - This implies  $f^* \geq f(G_2)$
  - $G_2$  is a goal state:  $h(G_2) = 0$ ,  $f(G_2) = g(G_2)$ .
  - Therefore  $f^* \geq g(G_2)$
  - Contradiction

# Dealing with hard problems

- For large problems,  $A^*$  may require too much space
- Variations conserve memory: IDA\* and SMA\*
- IDA\*, iterative deepening  $A^*$ , uses successive iteration with growing limits on  $f$ , e.g.
  - $A^*$  but don't consider a node  $n$  where  $f(n) > 10$
  - $A^*$  but don't consider a node  $n$  where  $f(n) > 20$
  - $A^*$  but don't consider a node  $n$  where  $f(n) > 30, \dots$
- SMA\* -- Simplified Memory-Bounded  $A^*$ 
  - Uses queue of restricted size to limit memory use

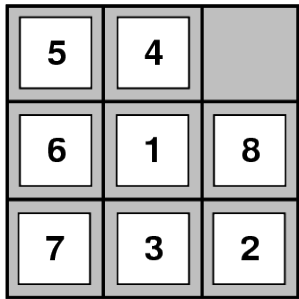
# Finding good heuristics

- If  $h_1(n) < h_2(n) \leq h^*(n)$  for all  $n$ ,  $h_2$  is better than (**dominates**)  $h_1$
- Relaxing problem: remove constraints for easier problem; use its solution cost as heuristic function
- Combining heuristics: max of two admissible heuristics is an admissible heuristic, and it's better!
- Use statistical estimates to compute  $h$ ; may lose admissibility
- Identify good features, then use learning algorithm to find heuristic function; also may lose admissibility

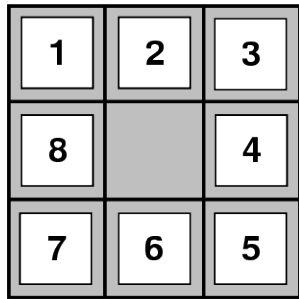


# In-class Exercise: Creating Heuristics

## 8-Puzzle

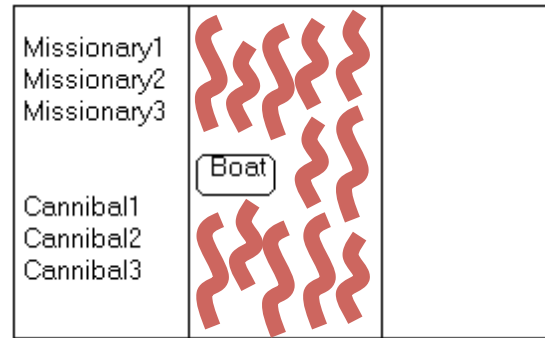


Start State

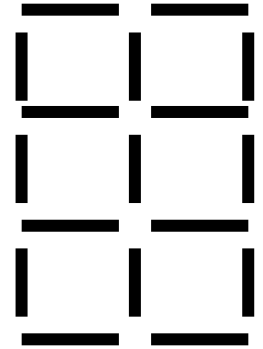


Goal State

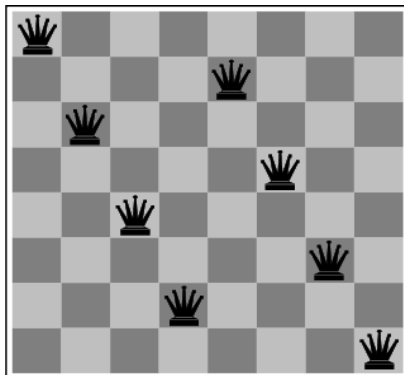
## Missionaries and Cannibals



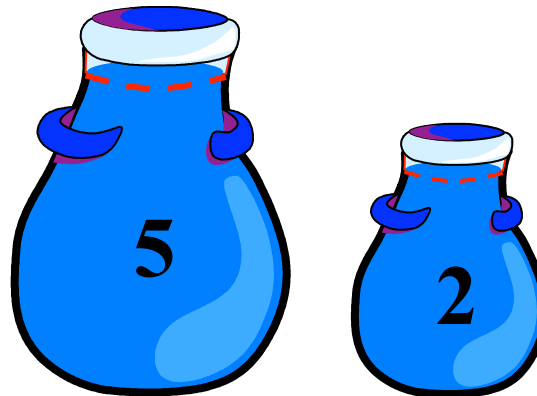
## Remove 5 Sticks



## N-Queens



## Water Jug Problem



## Route Planning



# Summary: Informed search

- **Best-first search** is general search where minimum-cost nodes (wrt some measure) are expanded first
- **Greedy search** uses minimal estimated cost  $h(n)$  to goal state as measure; reduces search time, but is neither complete nor optimal
- **A\* search** combines uniform-cost search and greedy search:  $f(n) = g(n) + h(n)$ . Handles state repetitions and  $h(n)$  never overestimates
  - A\* is complete and optimal, but space complexity high
  - Time complexity depends on quality of heuristic function
  - IDA\* and SMA\* reduce the memory requirements of A\*