Informed Search

Search

Chapter 4 (a)

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. \hy*u*- ˈris-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: (computer science) 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 the best path along which to continue searching

- Define a heuristic function, **h(n)**, that estimates the "goodness" of a node n.

- Specifically, h(n) = **estimated cost** (or distance) of minimal cost path from n **to a goal state**.

- The heuristic function is an estimate, based on domain-specific information that is computable from the current state description, of how close we are to a goal

Heuristics

- **All domain knowledge** used in the search is encoded in the **heuristic function, h()**.

- An example of a "**weak method**" due to limited way domain-specific information is 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) >= 0 for all nodes n
  - h(n) = 0 implies that n is a goal node
  - h(n) = ∞ 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 of weak methods include
  - **Means-ends analysis**: represent current situation & goal and then seek ways to shrink the differences between them
  - **Space splitting:** list the possible solutions to a problem and then try to rule out classes of these possibilities
  - **Subgoaling**: split a large problem into several smaller ones that can be solved one at a time
- Called "weak" methods 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	

1	2	3
4	5	6
7	8	8

N	N	N
N	N	N
N	Y	

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

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

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	

2 spaces

3 spaces

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.

h(n)
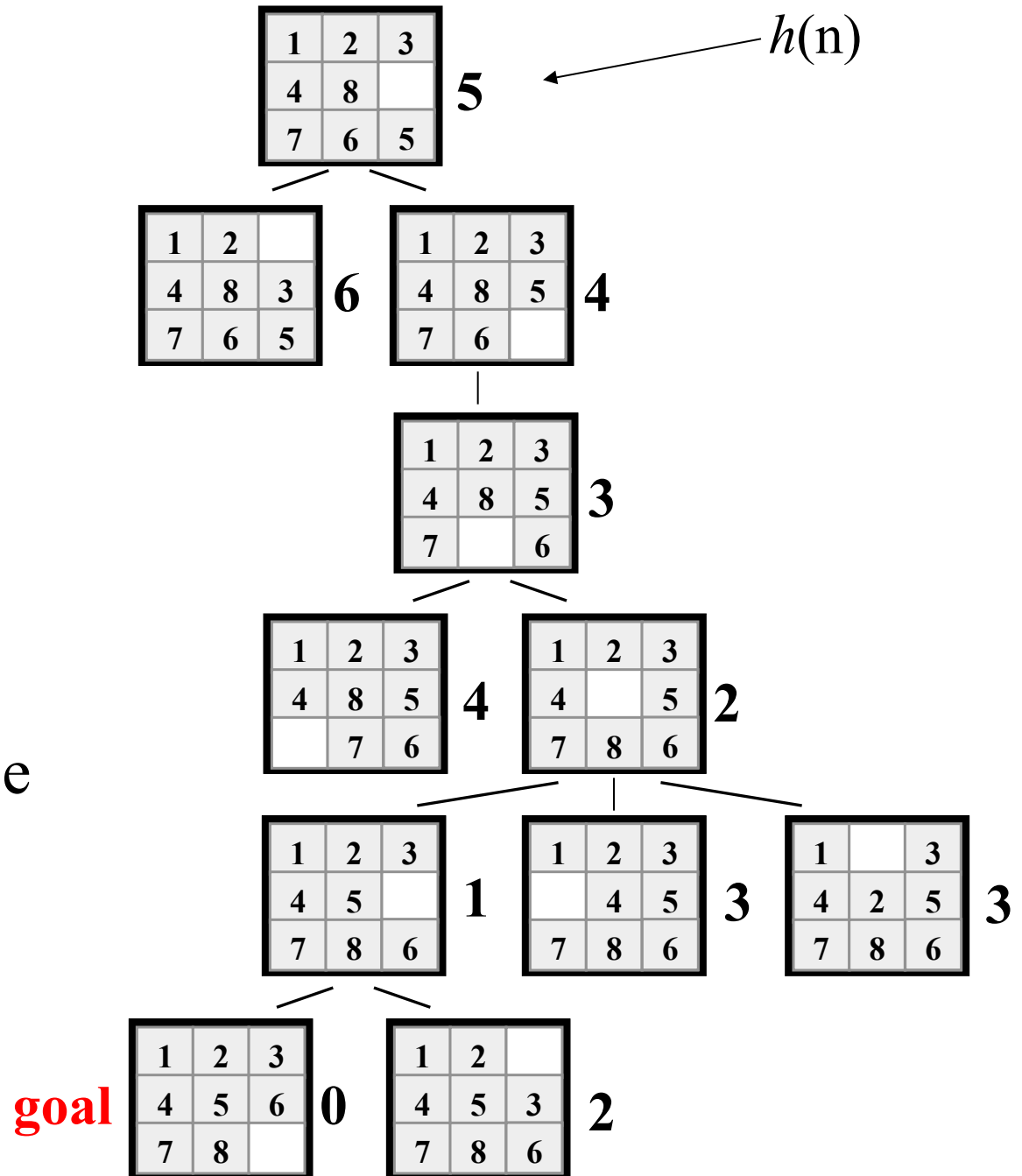
In this example, hill climbing does not work!

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

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

h(n)

1	2	3
4	5	8
6	7	

6

1	2	3
4	5	8
6		7

7

1	2	3
4	5	
6	7	8

5

1	2	3
4		5
6	7	8

6

1	2	
4	5	3
6	7	8

6
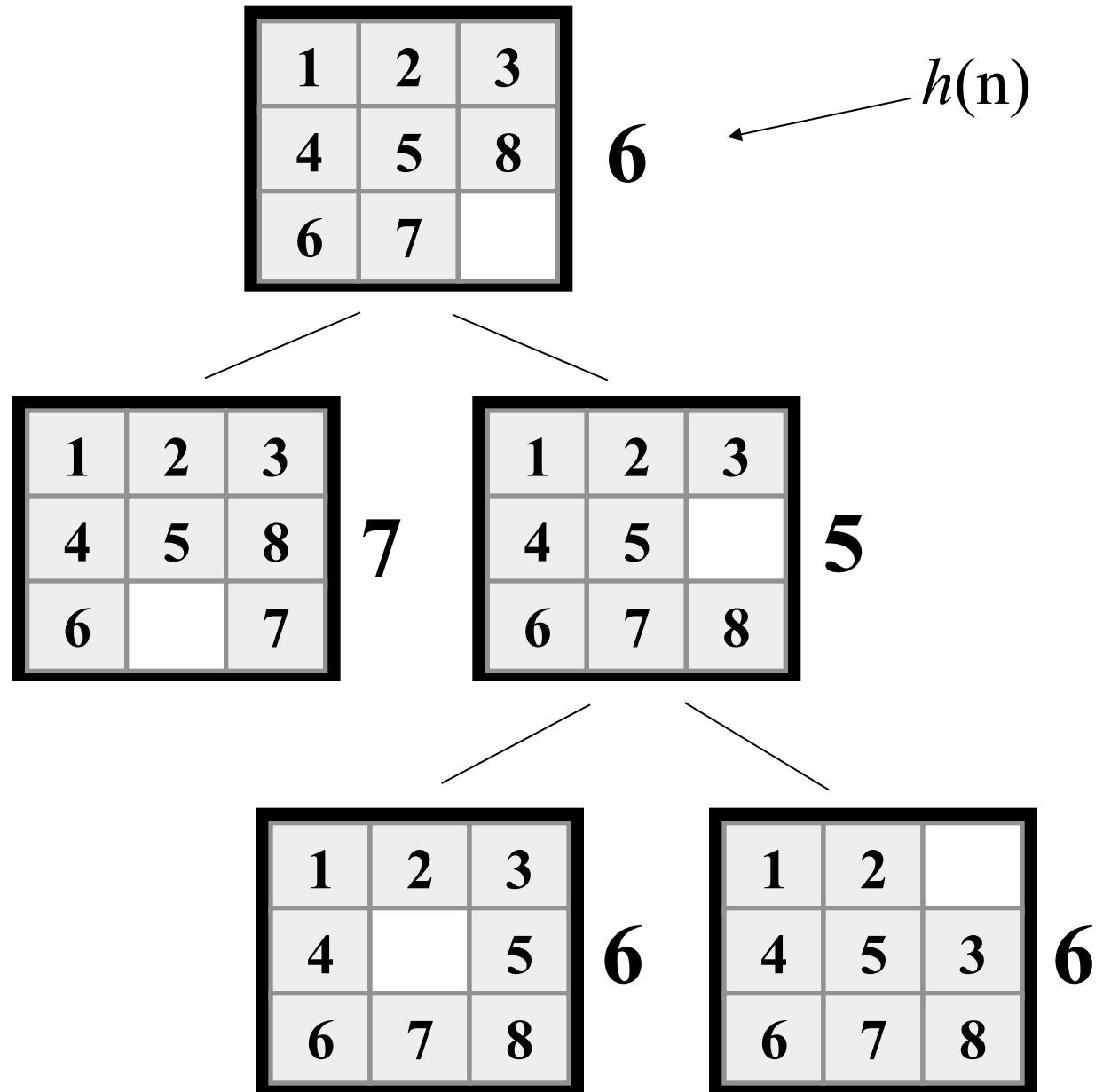
Best-first search

- A search algorithm that optimizes depth-first search by expanding most promising node chosen according to some rule

- Order nodes on the nodes list by increasing value of an evaluation function, **f(n)**, that incorporates domain-specific information in some way

- 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 admissible, as in the example
  - Assuming all arc costs are one, then greedy search will find goal g, which has a solution cost of five
  - However, the optimal solution is the path to goal with cost three

a

h=2 b   h h=4

h=1 c   i h=1

h=1 d   h=0

g2

h=1 e

h=0 g
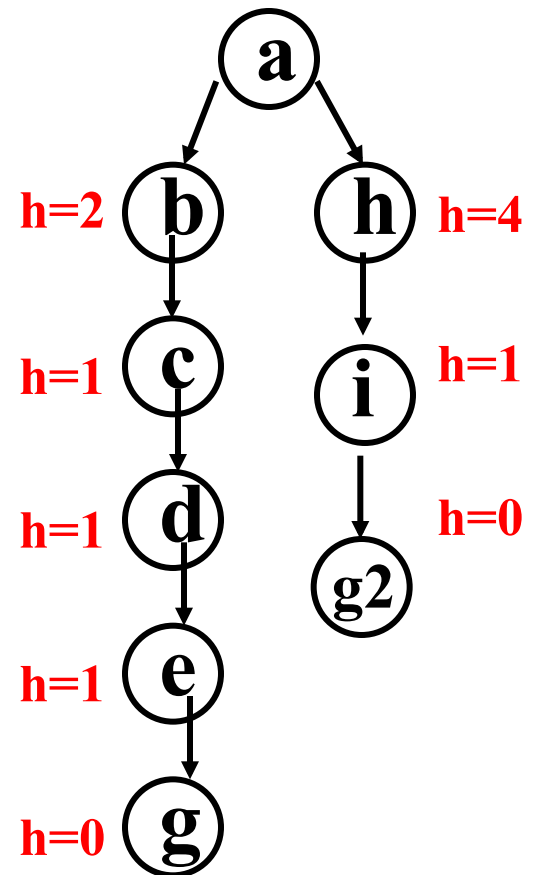
Beam search

- Use an evaluation function f(n), but the maximum size of the nodes list is k, a fixed constant

- Only keeps k best nodes as candidates for expansion, and throws the rest away

- k is the "beam width"

- More space efficient than greedy search, but may throw away a node that is on a solution path

- As k increases, beam search approaches best first search

- Not complete

- Not admissible (optimal)

Algorithm A

- Use as an evaluation function

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

- g(n) = minimal-cost path from the start state to state n

- g(n) term adds a "breadth-first" component to the evaluation function

- Ranks nodes on search frontier by estimated cost of solution from start node *through the given node* to goal

- Not complete if h(n) can equal infinity

- Not admissible (optimal)

$g(d)=4$

$h(d)=9$

C is chosen next to expand

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*

- "A star"
- Described by Hart and Nilsson in 1968
- Algorithm A with constraint that **h(n) <= h*(n)**
- **h*(n)** = *true cost* of the minimal cost path from n to a goal
- h is **admissible** when h(n) <= h*(n) holds
- Using an admissible heuristic guarantees that the first solution found will be an optimal one
- A* is **complete** whenever the branching factor is finite, and every operator has a fixed positive cost
- A* is **admissible**

Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics SSC4* **4** (2): 100–107.

Some observations on A

- **Perfect heuristic:** If h(n) = h*(n) for all n, then only the nodes on the optimal solution path will be expanded. So, no extra work will be performed

- **Null heuristic:** If h(n) = 0 for all n, then this is an admissible heuristic and A* acts like uniform-cost search

- **Better heuristic:** If h1(n) < h2(n) <= h*(n) for all non-goal nodes, then h2 is a *better* heuristic than h1
  - If A1* uses h1, and A2* uses h2, then every node expanded by A2* is also expanded by A1*
  - i.e., A1 expands at least as many nodes as A2*
  - We say that A2* is *better informed* than A1*

- The closer h to h*, the fewer extra nodes expanded

Example search space
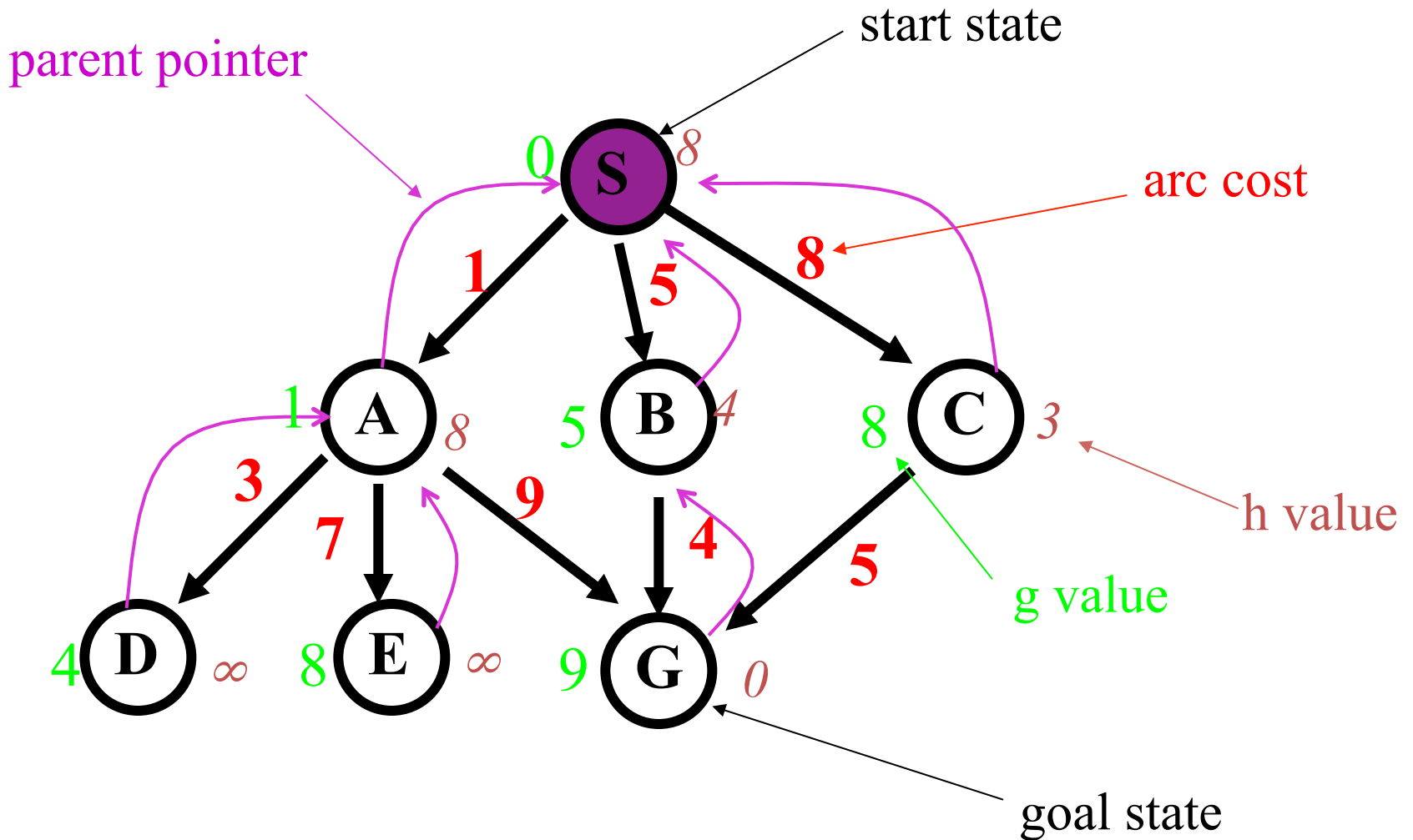
Example

n	g(n)	h(n)	f(n)	h*(n)
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 the (hypothetical) perfect heuristic (an oracle)

- Since h(n) <= h*(n) for all n, h is admissible (optimal)

- Optimal path = *S B G* with cost 9

Greedy search

f(n) = h(n)

```
node expanded       nodes list
                    { 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 G2, a goal state with a suboptimal solution, i.e., $g(G2) > f^*$

- We show that this is impossible
  - Choose a node n on the optimal path to G
  - Because $h(n)$ is admissible, $f^* >= f(n)$
  - If we choose G2 instead of n for expansion, then $f(n) >= f(G2)$.
  - This implies $f^* >= f(G2)$.
  - G2 is a goal state: $h(G2) = 0$, $f(G2) = g(G2)$.
  - Therefore $f^* >= g(G2)$
  - Contradiction

Dealing with hard problems

- For large problems, A* may require too much space

- Two 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 any node n where f(n) >10
  - A* but don't consider any node n where f(n) >20
  - A* but don't consider any node n where f(n) >30, …

- SMA* -- Simplified Memory-Bounded A*
 - uses a queue of restricted size to limit memory use

On finding a a good heuristic

- If $h_1(n) < h_2(n) <= h^*(n)$ for all n, h2 is better than (**dominates**) h1

- Relaxing the problem: remove constraints to create a (much) easier problem; use the solution cost for this problem as the heuristic function

- Combining heuristics: take the max of several admissible heuristics: still have an admissible heuristic, and it's better!

- Use statistical estimates to compute g; may lose admissibility

- Identify good features, then use a learning algorithm to find a heuristic function; also may lose admissibility
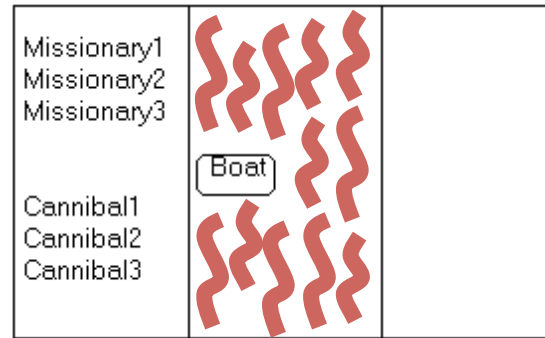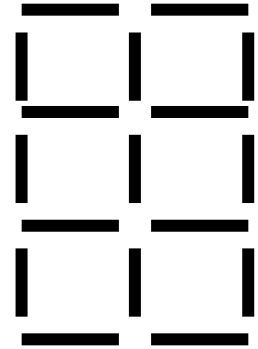
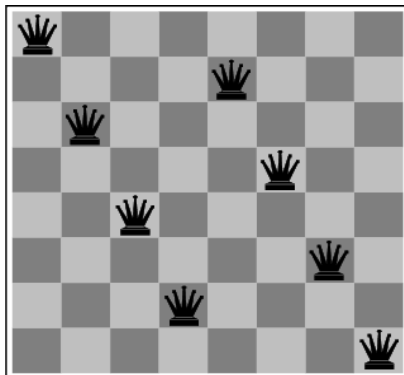# In-class Exercise: Creating Heuristics

8-Puzzle

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5
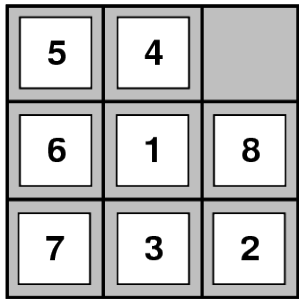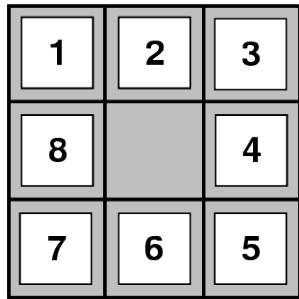
Goal State

Missionaries and Cannibals

Missionary1
Missionary2
Missionary3

Boat
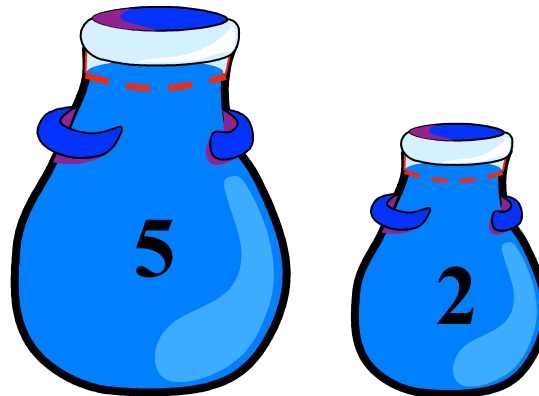
Cannibal1
Cannibal2
Cannibal3

Remove 5 Sticks

N-Queens

Water Jug Problem

5

2

Route Planning

Summary: Informed search

- **Best-first search** is general search where the minimum-cost nodes (w.r.t. some measure) are expanded first

- **Greedy search** uses minimal estimated cost h(n) to the 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). A* handles state repetitions and h(n) never overestimates

  – A* is complete and optimal, but space complexity is high.

  – The time complexity depends on the quality of the heuristic function.

  – IDA* and SMA* reduce the memory requirements of A*.