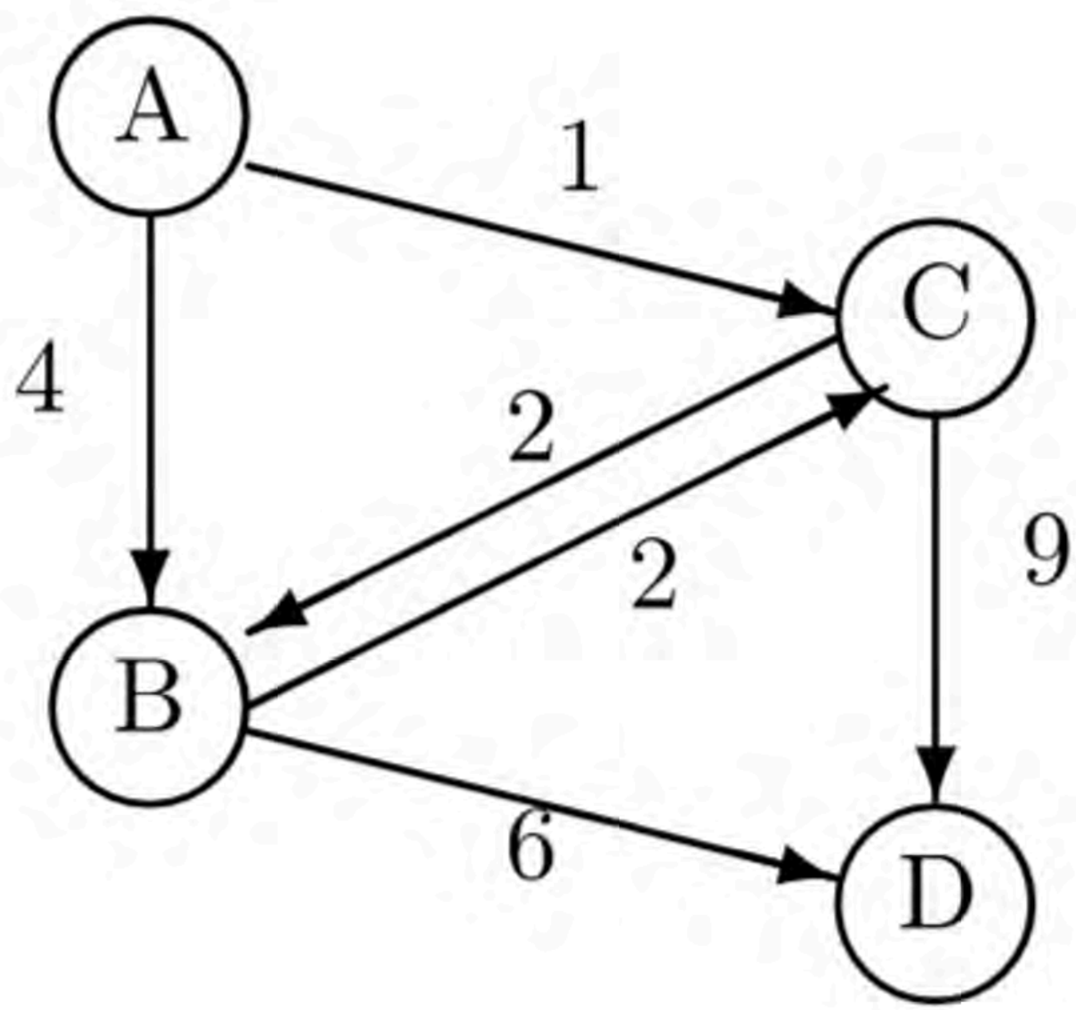
Heuristic Search (Part 2)

- Reading note: Chapter 4 covers heuristic search.

Search animations: Pac Man

https://www.youtube.com/watch?v=2XjzjAfGWzY

Problem!



$h(A) = 8$

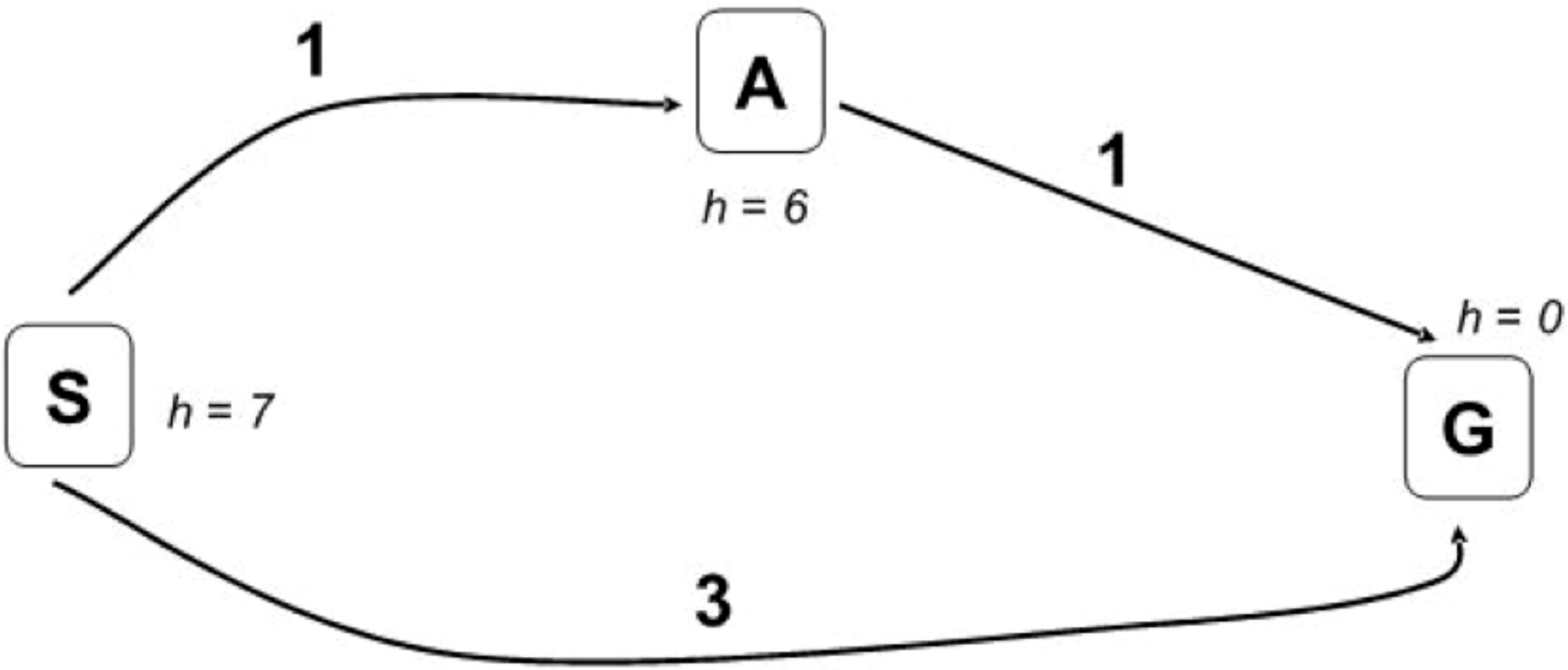
$h(B) = 3$

$h(C) = 7$

$h(D) = 0$

START = A
GOAL = D

Back to A*: is it Optimal?



Properties of A* depend on conditions on h(n)

- To achieve completeness, optimality, and desirably time and space complexity with A* search, we must put some conditions on the heuristic function h(n) and the search space.

Condition on h(n): Admissible

- Assume each transition due to an action a has cost $\geq \varepsilon > 0$.
- Let h*(n) be the **cost of an optimal path** from n to a goal node (∞ if there is no path). Then an admissible heuristic satisfies the condition:

$$h(n) \leq h^*(n)$$

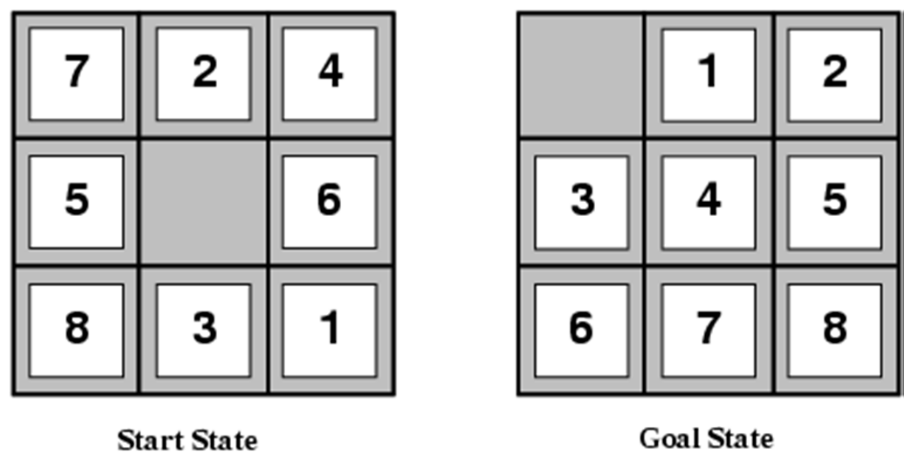
an admissible heuristic never over-estimates the cost to reach the goal, i.e., it is optimistic

- Hence $h(g) = 0$, for any goal node g
- Also $h^*(n) = \infty$ if there is no path from n to a goal node

Admissible heuristics

Which heuristics are admissible for the 8 puzzle?

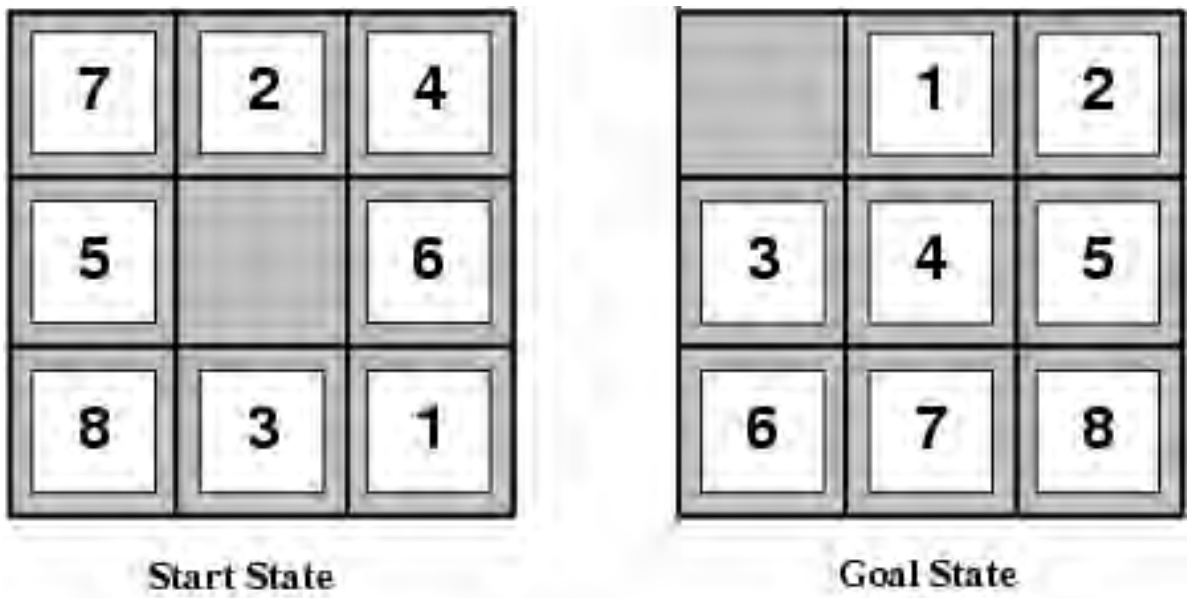
- *h(n)* = number of misplaced tiles
- *h(n)* = total Manhattan distance between tile locations in S and goal locations in G
- $h(n) = \min(2, h^*[n])$
- $h(n) = h^*(n)$
- $h(n) = \max(2, h^*[n])$
- $h(n) = 0$



Admissible heuristics

Say for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



- $h_1(S)$ = ? 8
- $h_2(S)$ = ? 3+1+2+2+2+3+3+2 = 18

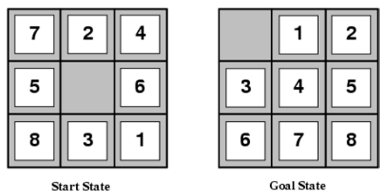
How to build a heuristic?

A useful technique is to simplify a problem when building heuristics, and to let h(n) be the cost of reaching the goal in the easier problem.

For example, in the 8-Puzzle you can only move a tile from square A to B if A is adjacent (left, right, above, below) to B and B is blank

We can relax some of these conditions and:

1. allow a move from A to B if A is adjacent to B (i.e. we can ignore whether or not position is blank),
2. allow a move from A to B if B is blank (i.e. we can ignore adjacency),
3. allow all moves from A to B (ignore both conditions).



How to build a heuristic?

- #3 leads to the misplaced tiles heuristic.
  - To solve the puzzle, we need to move each tile into its final position.
  - Number of moves = number of misplaced tiles.
  - Clearly h(n) = number of misplaced tiles ≤ the h*(n) the cost of an optimal sequence of moves from n.
- #1 leads to the Manhattan distance heuristic.
  - To solve the puzzle we need to slide each tile into its final position.
  - We can move vertically or horizontally.
  - Number of moves = sum over all of the tiles of the number of vertical and horizontal slides we need to move that tile into place.
  - Again h(n) = sum of the Manhattan distances ≤ h*(n)
    - in a real solution we need to move each tile at least that that far and we can only move one tile at a time.

Admissible heuristics make for optimal search

Why?

- Say we have an optimal path to n_{goal} with cost $g(n_{goal})$.
- Let n'_{goal} be a sub-optimal path, meaning $g(n'_{goal}) > g(n_{goal})$.
- Let n" be any sub-path of the optimal path on the Frontier.

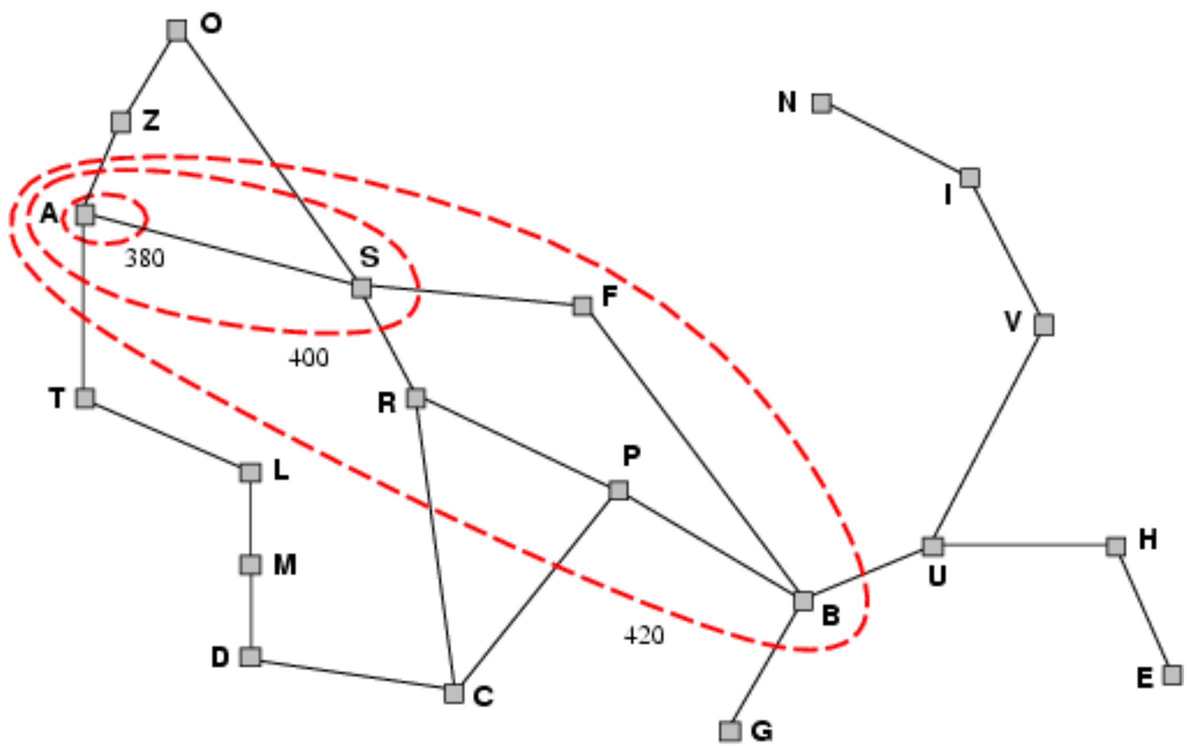
Is it possible for the path to n'_{goal} to be explored before the path to n_{goal}?

- No! Because $f(n_{goal}) < f(n'_{goal})$
- Also $f(n'') <= f(n_{goal})$, because our heuristic is admissible.
- So, $f(n'') < f(n'_{goal})$

Meaning sub-paths on the optimal path to n_{goal} will be explored before any sub-optimal path to the goal!

Admissible heuristics make for optimal search

- A* expands nodes, or paths, in order of increasing *f* value
- Gradually adds f contours
- Each contour contains all paths with $f=f_i$, where $f_i < f_{i+1}$



Stronger condition on h(n): Monotonic (or consistent)

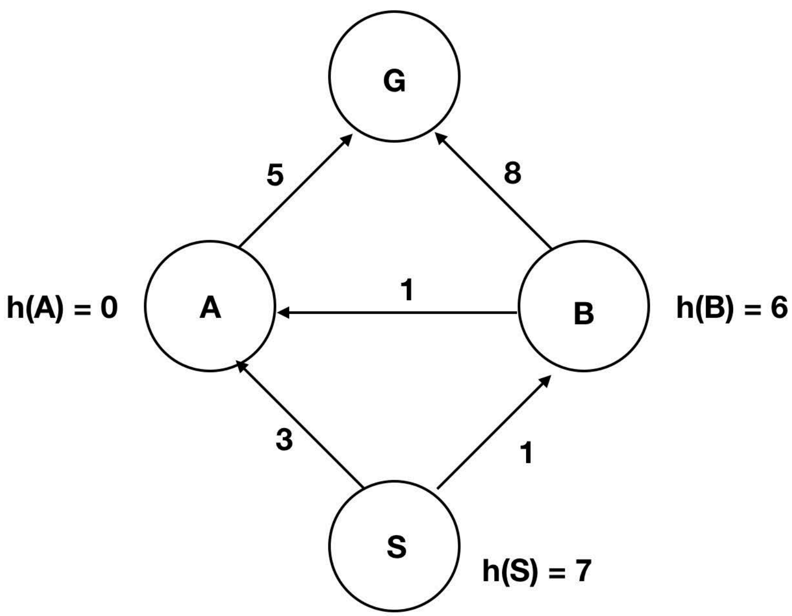
- Stronger condition than admissibility

- A monotone heuristic satisfies the condition

  $$h(n1) \leq c(n1, a, n2) + h(n2)$$

- Note that there might more than one transition (action) that joins n1 and n2, and the inequality must hold for all of them.

- If h(n) is admissible and monotonic, search will be both optimal *and* not "locally" mislead.



Consistency implies Admissibility

Assume consistency: $h(n1) \leq c(n1,a,n2) + h(n2)$

Prove admissible: $h(n) \leq h^*(n)$

If no path exists from n to a goal, $h^*(n) = \infty$ and $h(n) \leq h^*(n)$.
Let the path to from n to n_{goal} be an OPTIMAL path from n to a goal. Call the cost of this path $h^*(n)$, and call the cost of each sub-path from ni to n_{goal}, $h^*(ni)$.
We will prove $h(n) \leq h^*(n)$ by induction on the length of this optimal path.

Proof by Induction

Assume consistency: $h(n1) \leq c(n1,a,n2) + h(n2)$

Prove admissible: $h(n) \leq h^*(n)$

Base Case:

$h(n_{goal}) = 0 \leq h^*(n_{goal}) = 0$

$h(n_1) \leq c(n_1,a_1,n_{goal}) + h(n_{goal}) \leq c(n_1,a_1,n_{goal}) + h^*(n_{goal}) = h^*(n_1)$

Induction:

Assume $h(n_i) \leq h^*(n_i)$

$h(n_{i-1}) \leq c(n_{i-1},a_{i-1},n_i) + h(n_i) \leq c(n_{i-1},a_{i-1},n_i) + h^*(n_i) = h^*(n_{i-1})$

Some consequences of Monotonicity

1. f-values of states in a path are non-decreasing.

   i.e. if n1 and n2 are nodes along a path, then $f(n1) \leq f(n2)$

   Proof: $f(n1) = g(n1) + h(n1) = \text{cost(path to n1)} + h(n1)$

   $\leq g(n1) + c(n1, a, n2) + h(n2)$

   But $g(n1) + c(n1, a, n2) + h(n2) = g(n2) + h(n2) = f(n2)$

Some consequences of Monotonicity

1. f-values of states in a path are non-decreasing.

   i.e. if n1 and n2 are nodes along a path, then $f(n1) \leq f(n2)$

   Proof: $f(n1) = g(n1) + h(n1) = cost(path\ to\ n1) + h(n1)$

   $\leq g(n1) + c(n1, a, n2) + h(n2)$

   But $g(n1) + c(n1, a, n2) + h(n2) = g(n2) + h(n2) = f(n2)$

   So $f(n1) \leq f(n2)$

Some consequences of Monotonicity

2. If n2 is expanded after n1, then f(n1) ≤ f(n2).

   i.e. f-values of nodes that are expanded cannot decrease during the search.

Why? When n1 was selected for expansion, n2 was either:

1. Already on the frontier, meaning f(n1) ≤ f(n2). Otherwise we would have expanded n2 before n1.
2. Added to the frontier as a result of n1's expansion, meaning n2 and n1 lie along the same path. If this is the case, as we demonstrated on the prior slide, f(n1) ≤ f(n2).

Some consequences of Monotonicity

3. If node n has been expanded, every path with a lower f-value than n has already been expanded.
   - Say we just expanded node ni on a path to node nk, and that f(nk) < f(n).
   - This means ni+1 is on the frontier and f(ni+1) ≤ f(nk), because they are both on the same path.
   - BUT if ni+1 were on the frontier at the same time as node n, it would have been expanded before n because f(ni+1) ≤ f(nk) < f(n).
   - Thus, n *can't* have been expanded before every path with a lower f-value has been expanded.

Some consequences of Monotonicity

4. The first time A* expands a node, it has found the minimum cost path to that node.

f(of the first discovered path to n) = cost(of the first discovered path to n) + h(n).

Likewise,

f(of any other path to n) = cost(of any other path to n) + h(n).

From the prior slide we know:

f(of the first discovered path to n) ≤ f(of any other path to n).

This means, by substitution:

cost(of 1st discovered path to n) ≤ cost(of any other path to n)

Hence, the first discovered path is the optimal one!

Monotonic, Admissible A*

Complete?

YES. Consider a least cost path to a goal node

–SolutionPath = <Start→ n1→ …→ G> with cost c(SolutionPath).

–Since each action has a cost ≥ ε > 0, there are only a finite number of paths that have f-value < c(SolutionPath). None of these paths lead to a goal node since SolutionPath is a least cost path to the goal.

–So eventually SolutionPath, or some equal cost path to a goal must be expanded.

Time and Space complexity?

–When h(n) = 0 for all n, h is monotone (A* becomes uniform-cost search)!

–When h(n) > 0 for some n and still admissible, the number of nodes expanded will be no larger than uniform-cost.

–Hence the same bounds as uniform-cost apply. (These are worst case bounds). Still exponential complexity unless we have a very good *h*!

–In real world problems, we sometimes run out of time and memory. We will introduce IDA* to address some memory issues, but IDA* isn't very good when many cycles are present.

Monotonic, Admissible A*

Optimal?

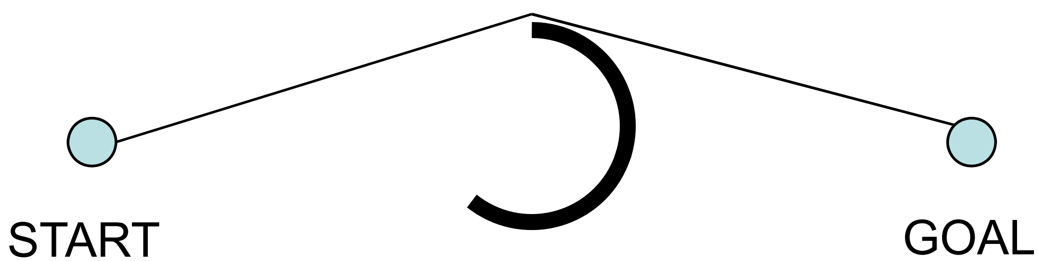
YES. As we saw, the first path to a goal node must be optimal.

Cycle Checking?

We can use a simple implementation of cycle checking (multiple path checking) - just reject all search nodes that visit a state already visited by a previously expanded node. We need keep only the first path to a state, rejecting all subsequent paths.

Effect of Heuristic Functions

- What portion of the state space will be explored by UCS? A*? Greedy search? Weighted A*?



Limitations of A* Search

- Observation: While A* may expand less of the state space, it is still constrained by speed or memory (many states are explored, on Frontier).
- Tools to address these problems:
  - IDA* (Iterative Deepening A*) - similar to Iterative Deepening Search.
  - Weighted A* - A* with an added weight, to bias exploration toward goal. We looked at this a bit last time!

IDA* - Iterative Deepening A*

Objective: reduce memory requirements for A*

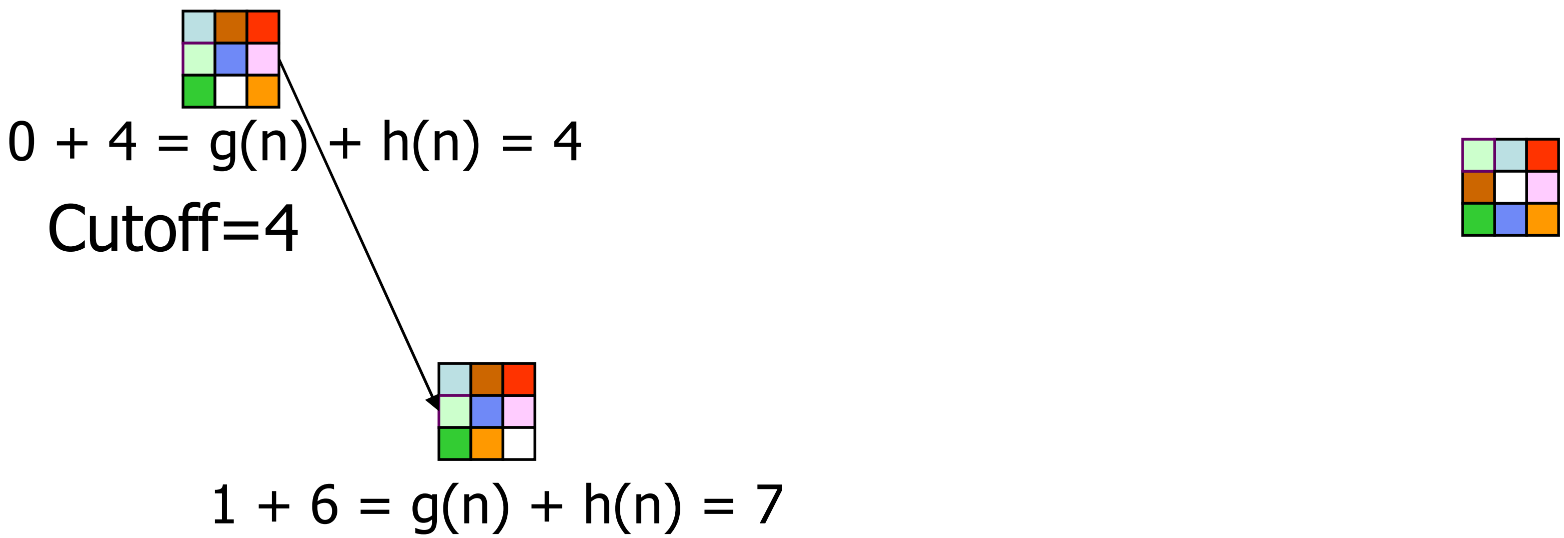
- Like iterative deepening, but now the "cutoff" is the f-value (g+h) rather than the depth
- At each iteration, the cutoff value is the smallest f-value of any node that exceeded the cutoff on the previous iteration
- Avoids overhead associated with keeping a sorted queue of nodes, and the open list occupies only linear space.
- Two new parameters:
  - curBound (any node with a bigger f-value is discarded)
  - smallestNotExplored (the smallest f-value for discarded nodes in a round); when Frontier becomes empty, the search starts a new round with this bound.
  - To compute "smallestNotExplored" most readily, expand all nodes with f-value EQUAL to the f-limit.

IDA* Example: 8-Puzzle

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

h(n) = number of misplaced tiles

blank tile is white

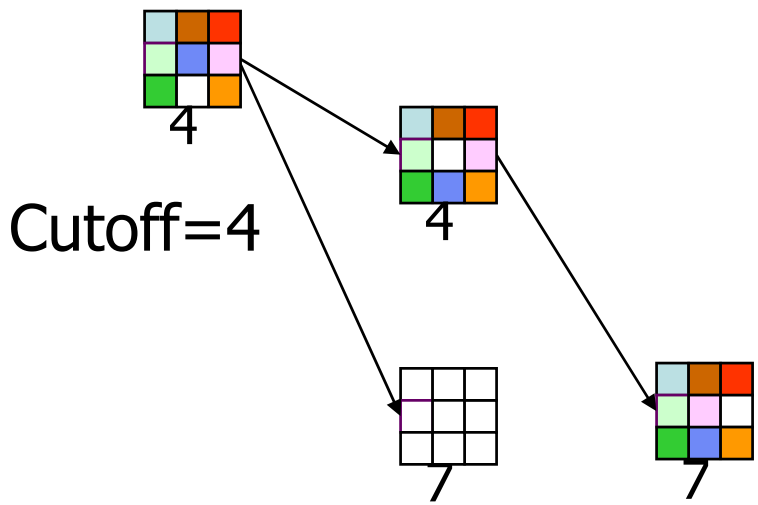


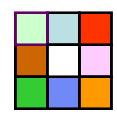
IDA* Example: 8-Puzzle

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

h(n) = number of misplaced tiles

blank tile is white



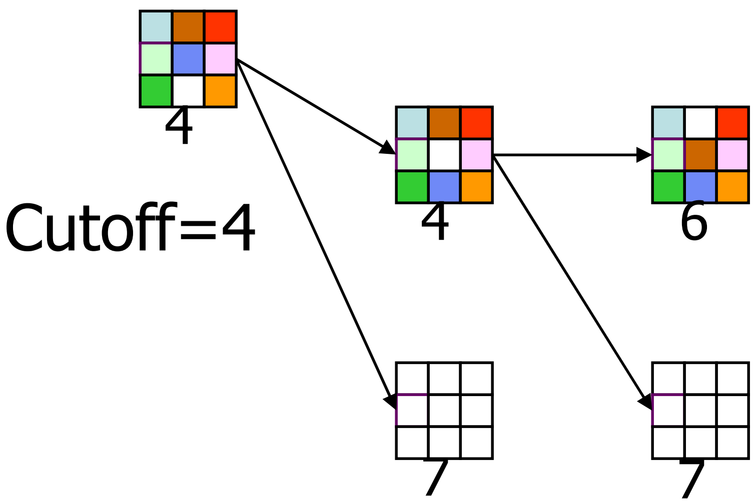


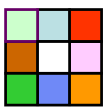
IDA* Example: 8-Puzzle

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

h(n) = number of misplaced tiles

blank tile is white

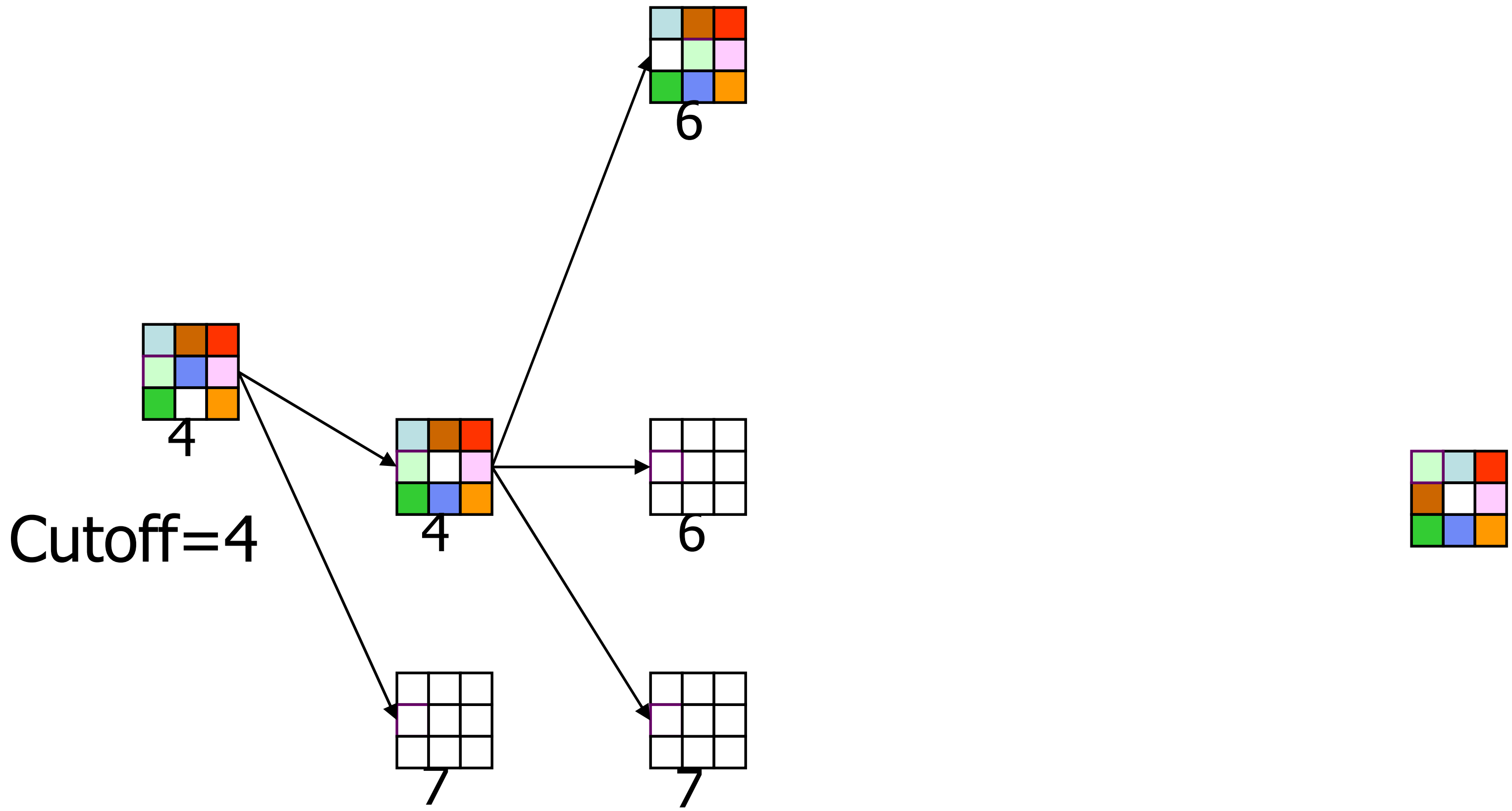




IDA* Example: 8-Puzzle

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

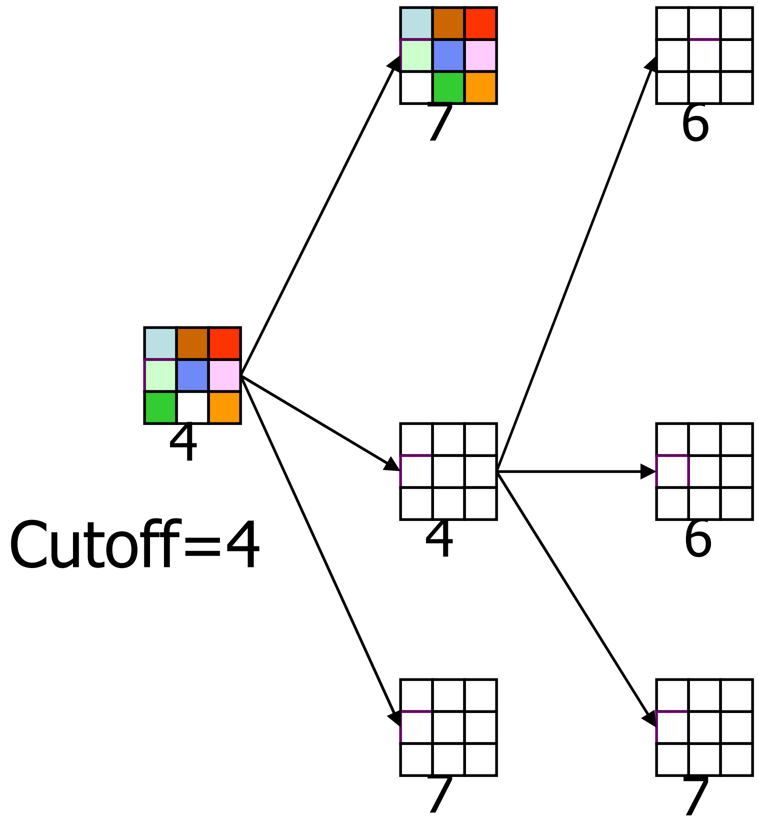
h(n) = number of misplaced tiles

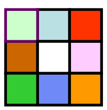


IDA* Example: 8-Puzzle

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

h(n) = number of misplaced tiles





IDA* Example: 8-Puzzle

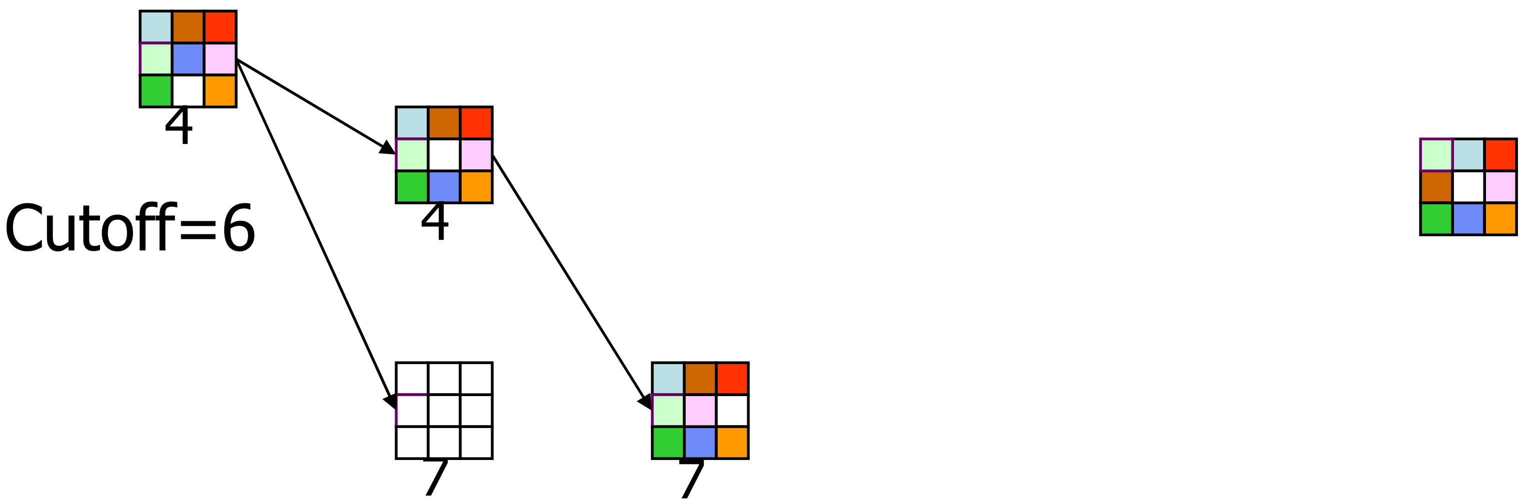
$f(n) = g(n) + h(n)$

$h(n)$ = number of misplaced tiles

IDA* Example: 8-Puzzle

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

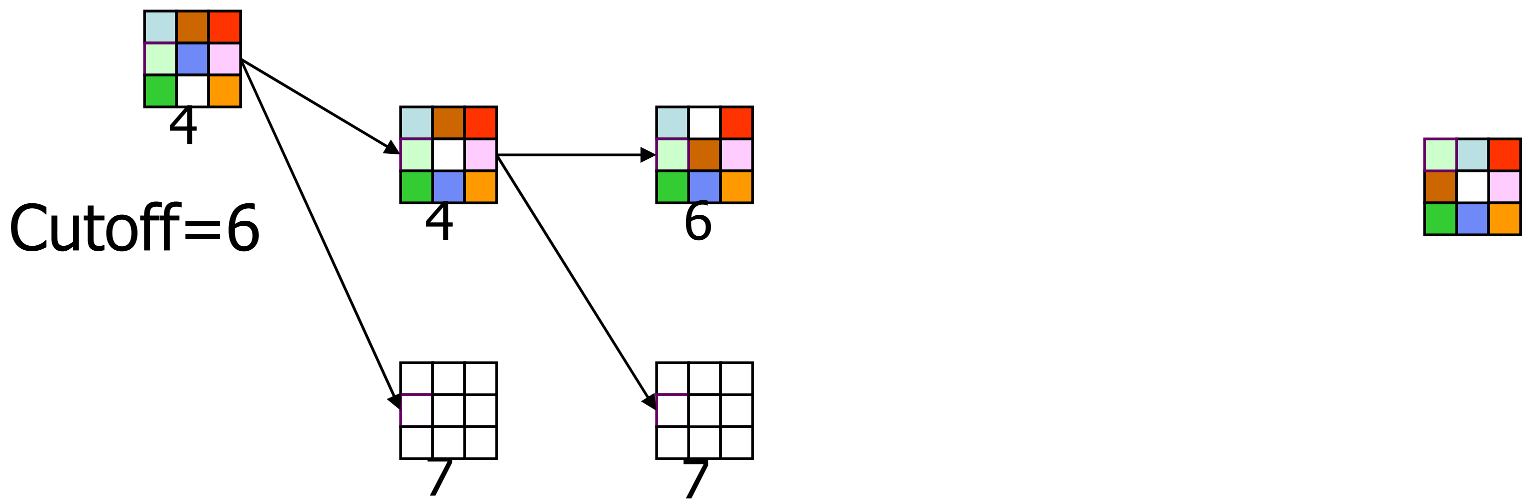
h(n) = number of misplaced tiles



8-Puzzle

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

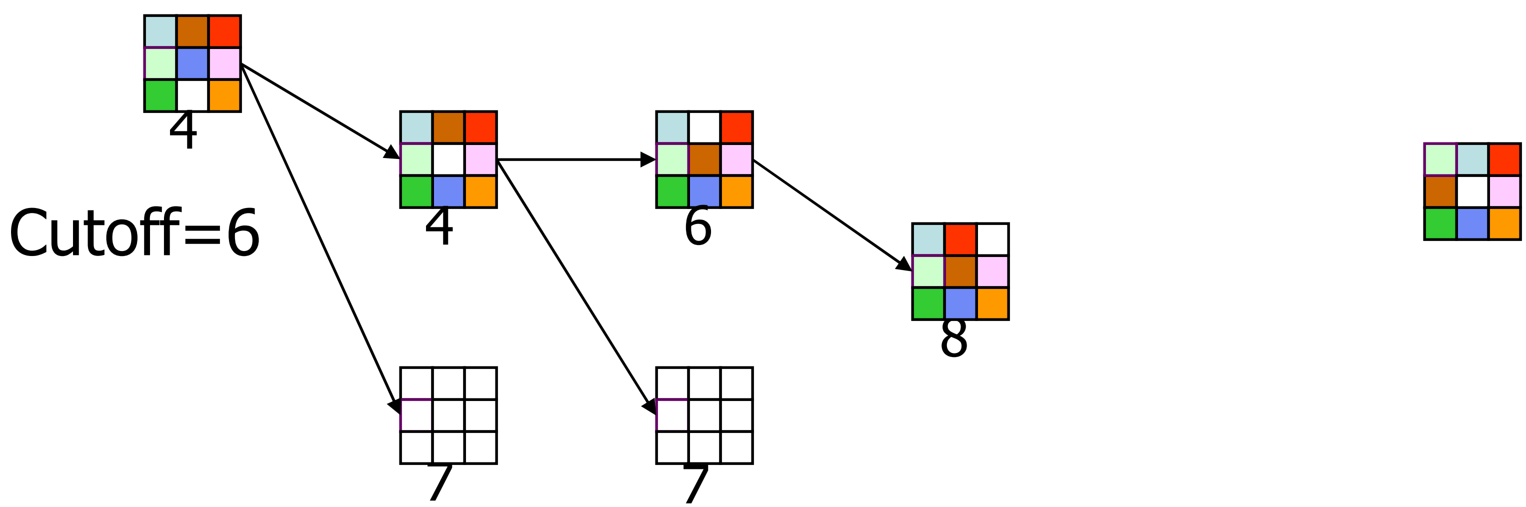
h(n) = number of misplaced tiles



8-Puzzle

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

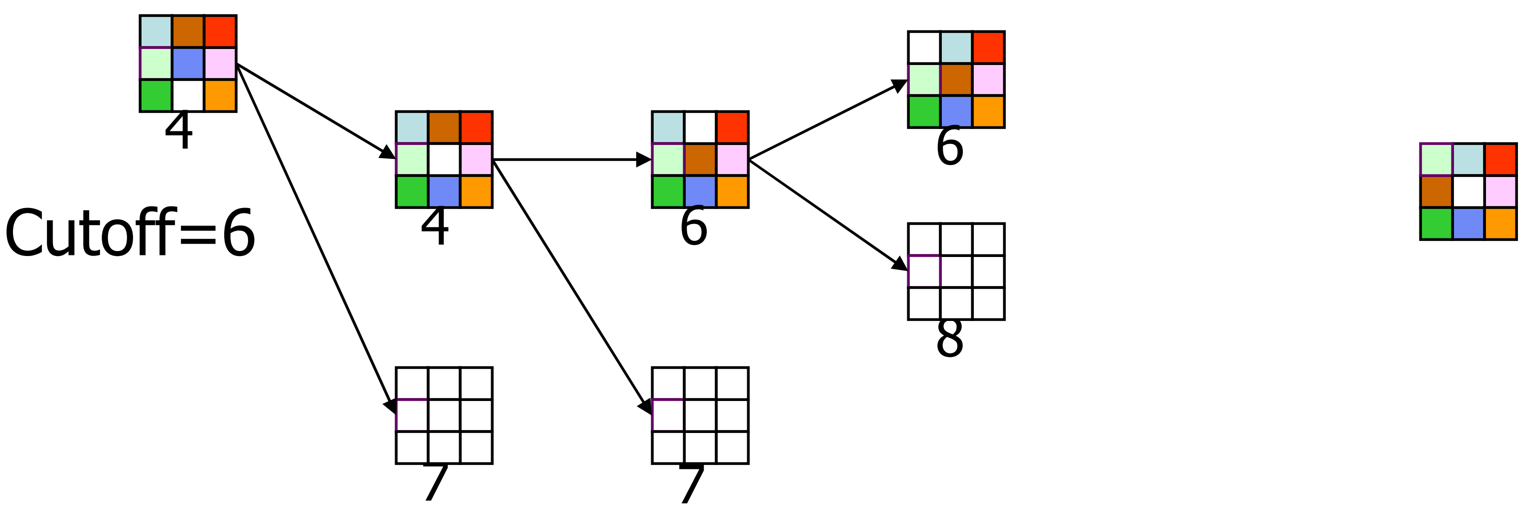
h(n) = number of misplaced tiles



8-Puzzle

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

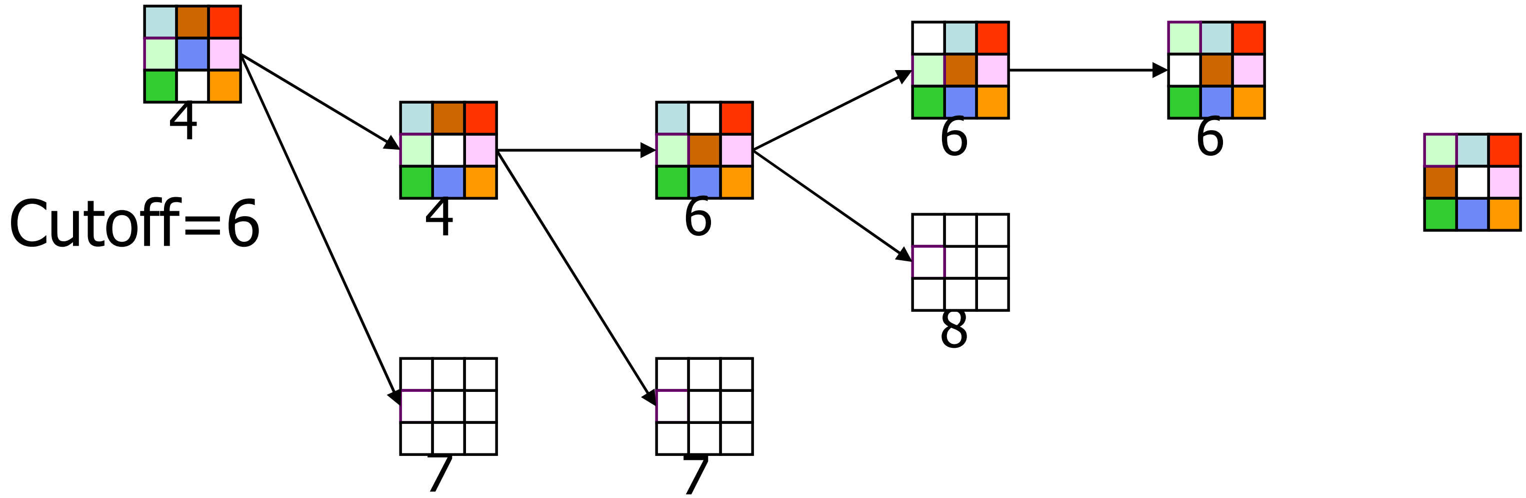
h(n) = number of misplaced tiles



8-Puzzle

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

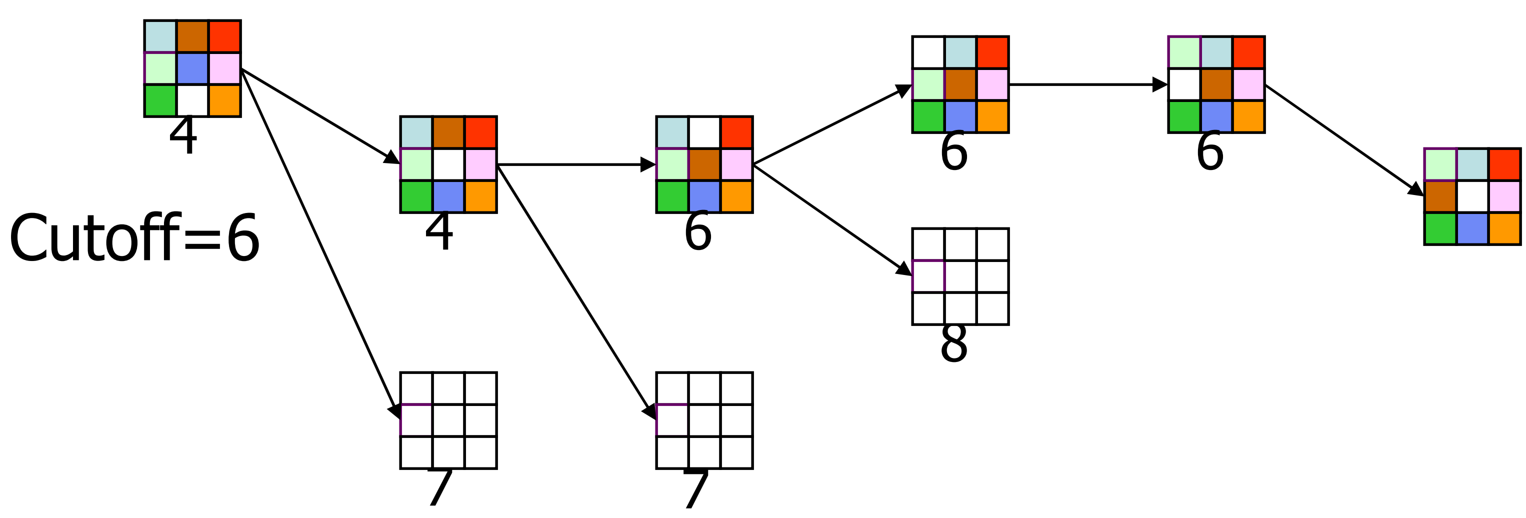
$h(n)$ = number of misplaced tiles



8-Puzzle

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

h(n) = number of misplaced tiles



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

IDA* - Iterative Deepening A*

- Optimal?
- Complete?
- Time and Space Complexity?
- Cycle Checking?