# Chapter 3
# Solving Problems by Searching

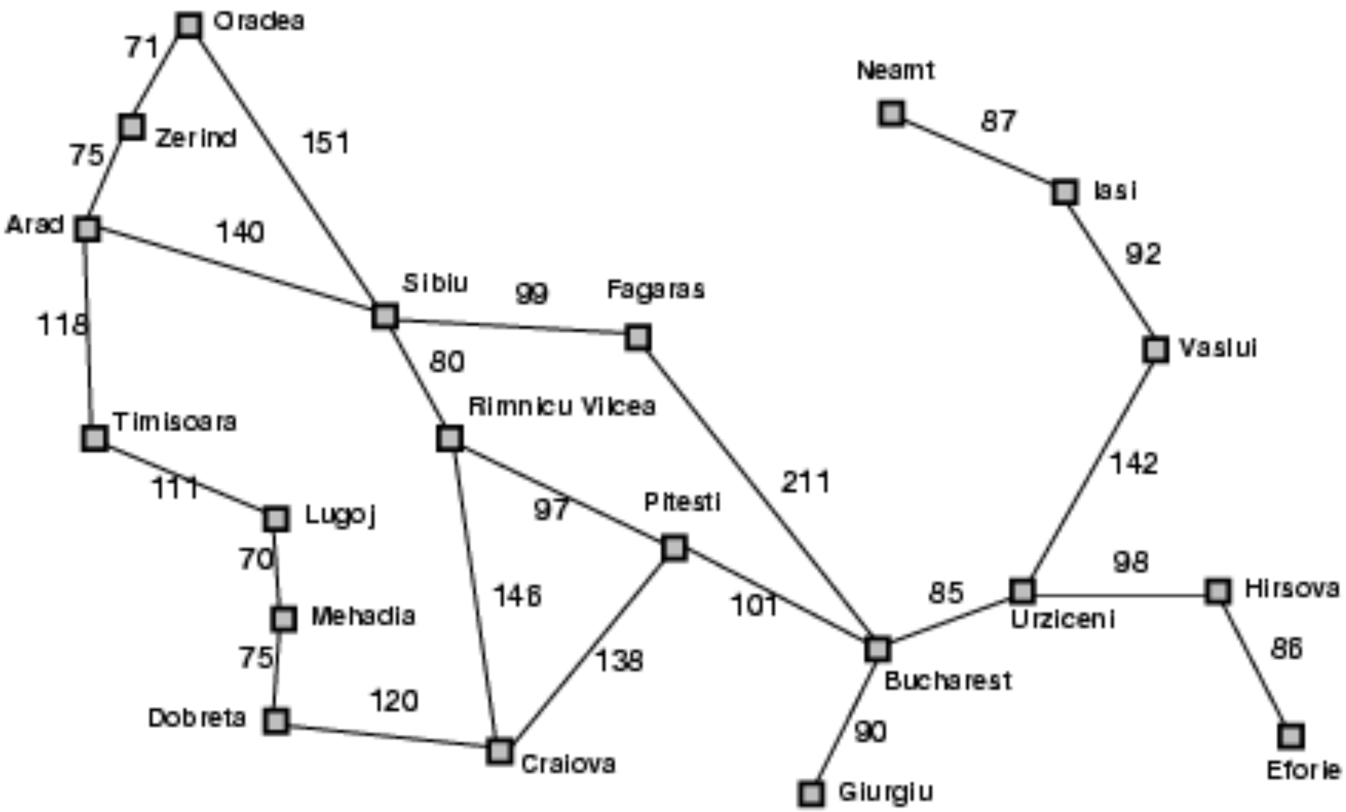
Jane Hsu
National Taiwan University

Acknowledgements: This presentation is created by Jane hsu based on the lecture slides from *The Artificial Intelligence: A Modern Approach* by Russell & Norvig, a PowerPoint version by Min-Yen Kan, as well as various materials from the web.

# Outline

- Informed (Heuristic) Search Strategies
  - Best-first search
  - Greedy best-first search
  - A* search
- Heuristic Functions
  - Relaxed problems
  - Pattern database

# Romania with Step Costs




Straight–line distance to Bucharest

| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Eforie** | 161 |
| **Fagaras** | 176 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Iasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 10 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

# Best-First Search

- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"
  - Expand most desirable unexpanded node

- Implementation:
- Order the nodes in the fringe in decreasing order of desirability

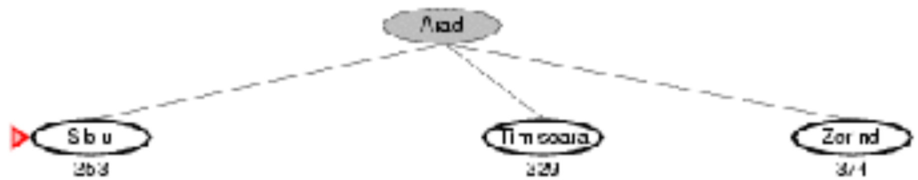
- Special cases:
  - greedy best-first search
  - A* search

# Greedy Best-First Search

- Evaluation function $f(n) = h(n)$ (heuristic) = estimate of cost from $n$ to *goal*
- e.g., $h_{SLD}(n)$ = straight-line distance from $n$ to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

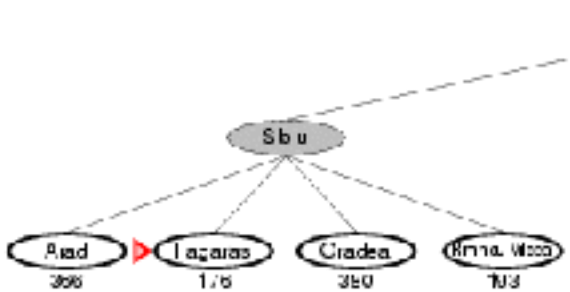
# Greedy Best-First Search Example

# Greedy Best-First Search Example



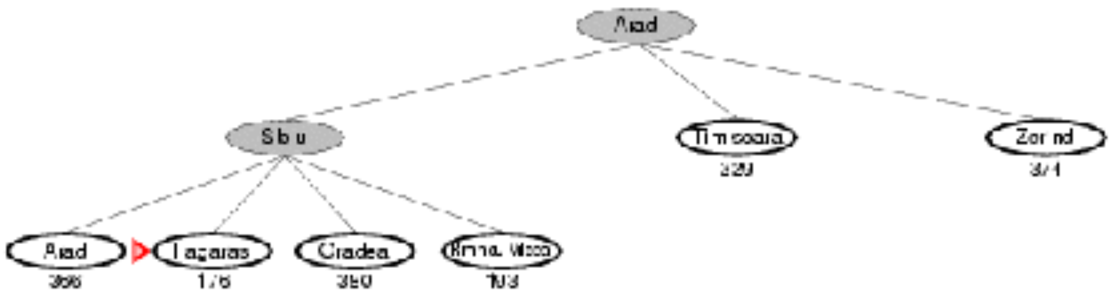
# Greedy Best-First Search Example


# Greedy Best-First Search Example



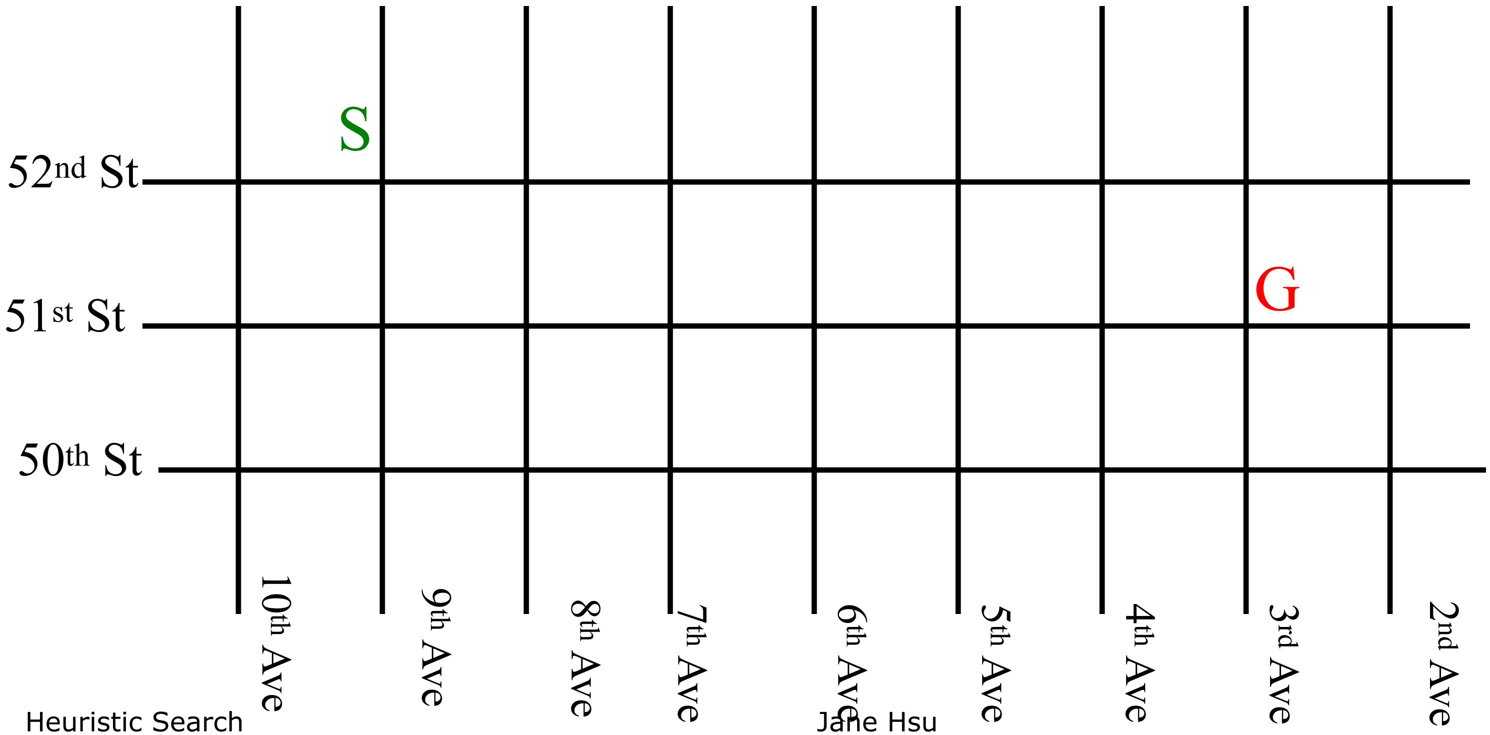
# Greedy Best-First Search Example

# Greedy Best-First Search Example




# Map of Manhattan

- How would you find a path from S to G?

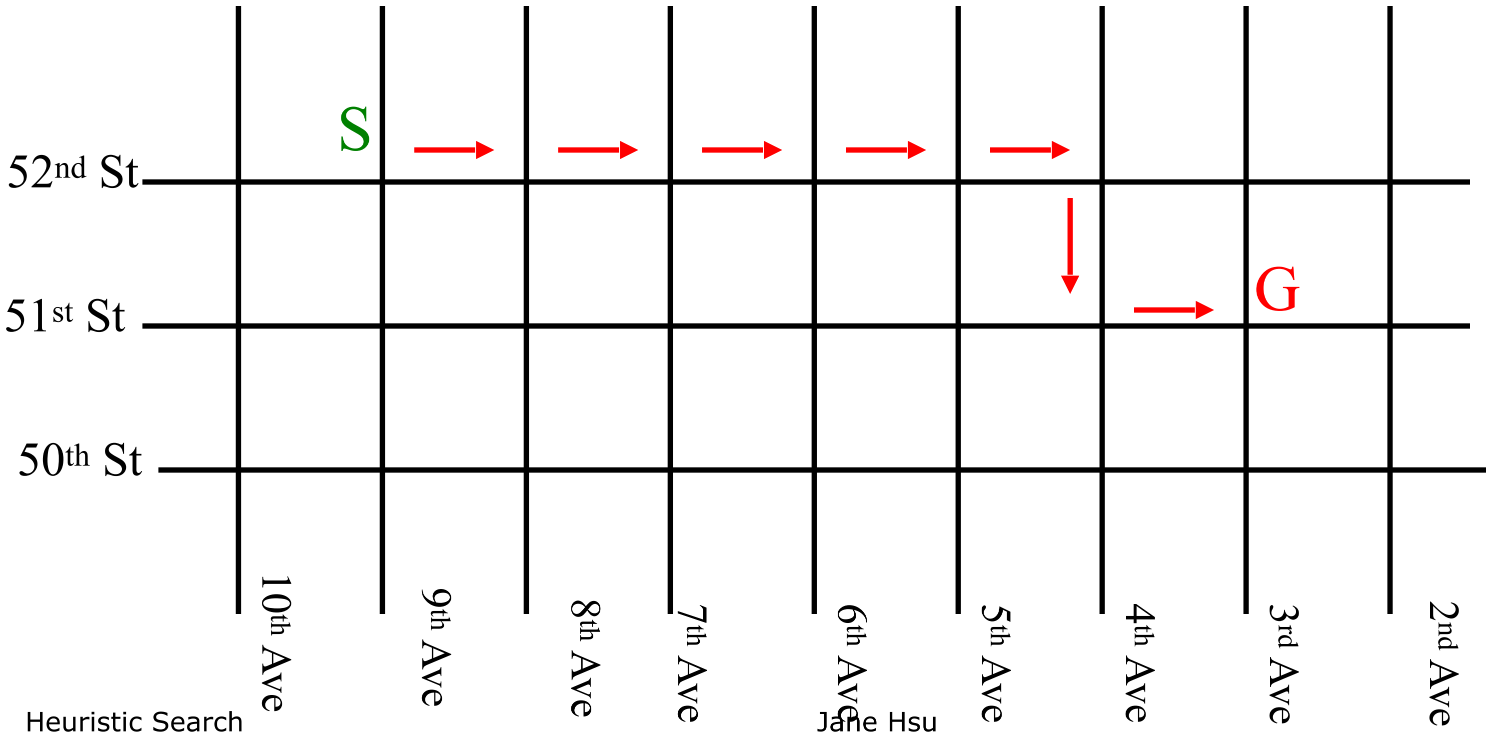


# Best-First Search

- The *Manhattan distance* ($\Delta$ x+ $\Delta$ y) is an estimate of the distance to the goal
  - It is a heuristic function
- Best-First Search
  - Order nodes in priority queue to minimize estimated distance to the goal h(n)
- Compare w/ Dijkstra
  - Order nodes in priority queue to minimize actual distance from the start

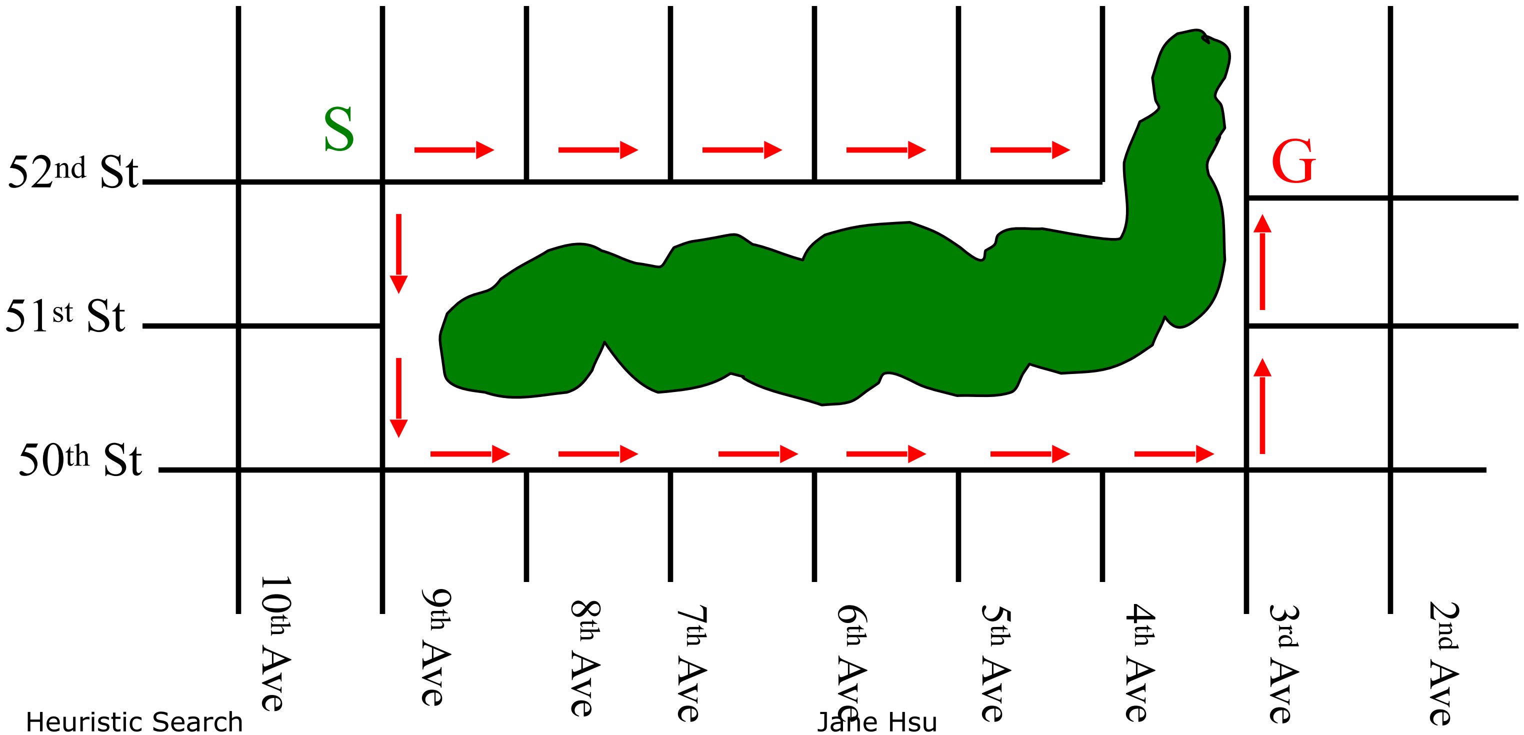
# Best First in Action

- How would you find a path from S to G?



# Problem 1: Led Astray

- Eventually will expand vertex to get back on the right track

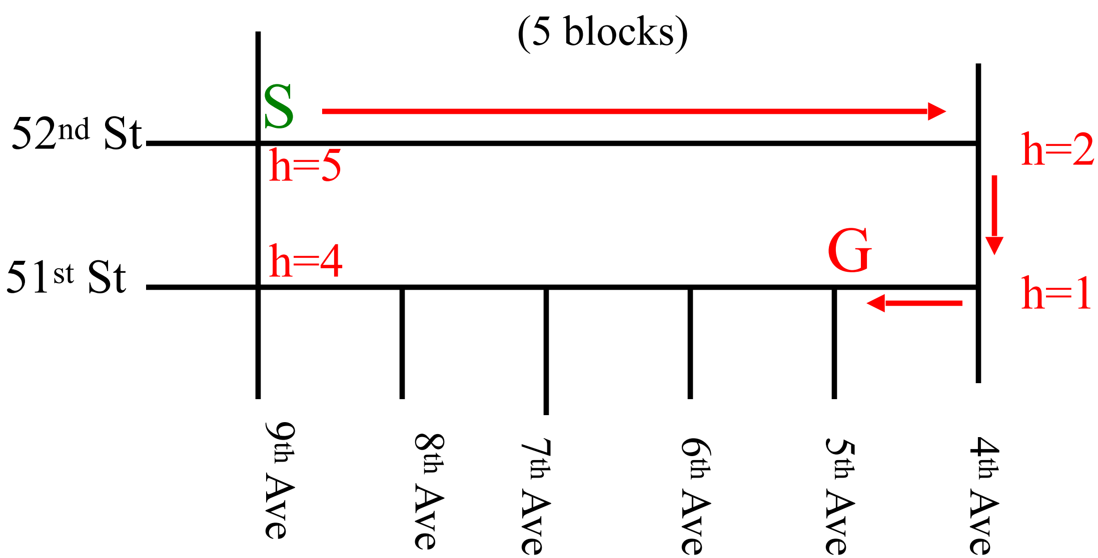


# Problem 2: Optimality

- With Best-First Search, are you *guaranteed* a shortest path is found when
  - goal is first seen?
  - when goal is removed from priority queue (as with Dijkstra?)

# Sub-Optimal Solution

- No!  Goal is by definition at distance 0: will be removed from priority queue immediately, even if a shorter path exists!



# Properties of Greedy Best-First Search

- Complete? No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ -- keeps all nodes in memory
- Optimal? No

# Synergy?

- Dijkstra / Breadth First guaranteed to find *optimal* solution
- Best First often visits *far fewer* vertices, but may not provide optimal solution

  - *Can we get the best of both?*

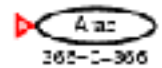
# $A^*$ Search

- Idea: avoid expanding paths that are already expensive

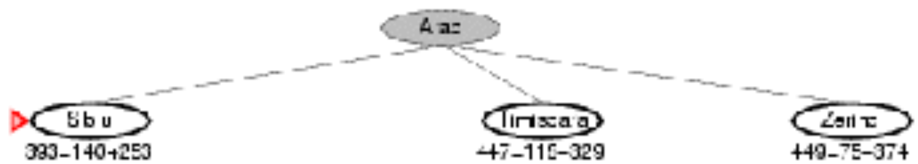
- Evaluation function $f(n) = g(n) + h(n)$

  - $g(n)$ = cost so far to reach $n$
  - $h(n)$ = estimated cost from $n$ to goal
  - $f(n)$ = estimated total cost of path through $n$ to goal

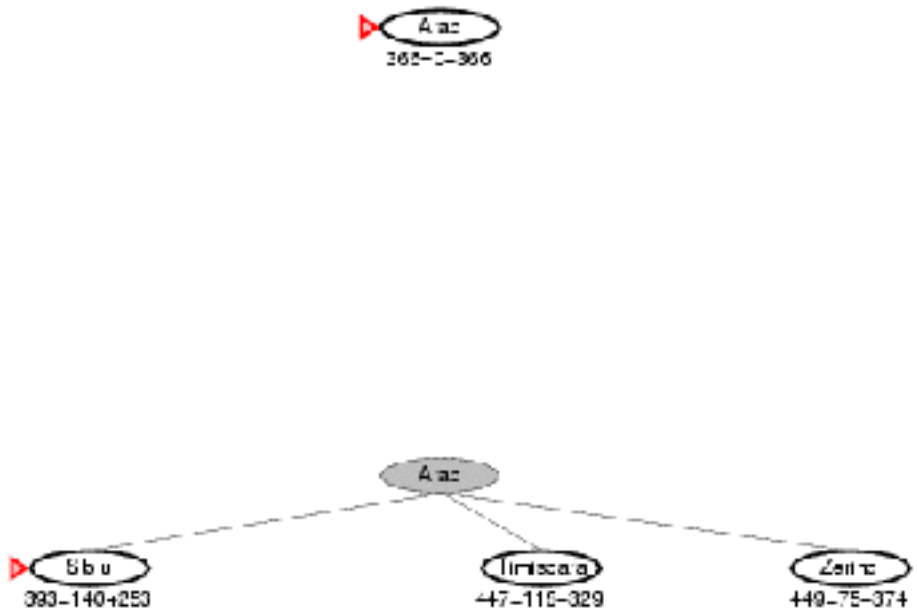
# A* Search Example



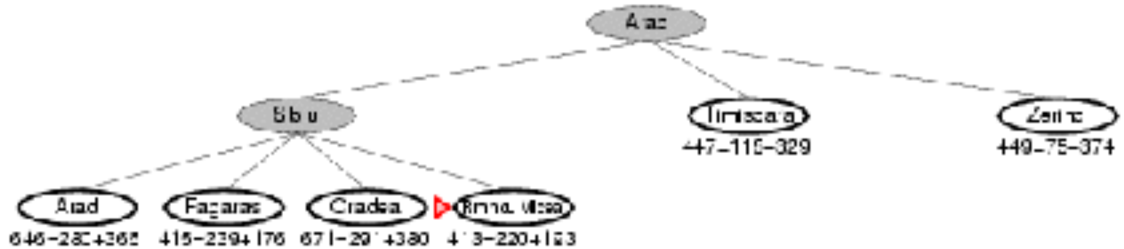
# A* Search Example



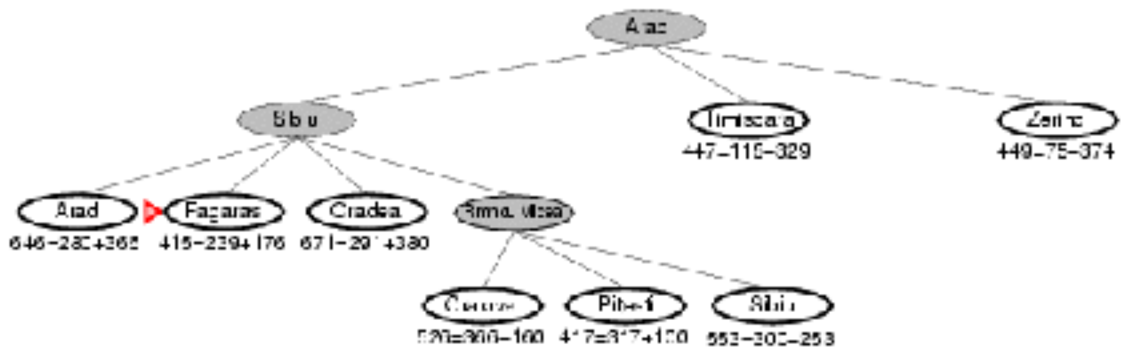
# A* Search Example



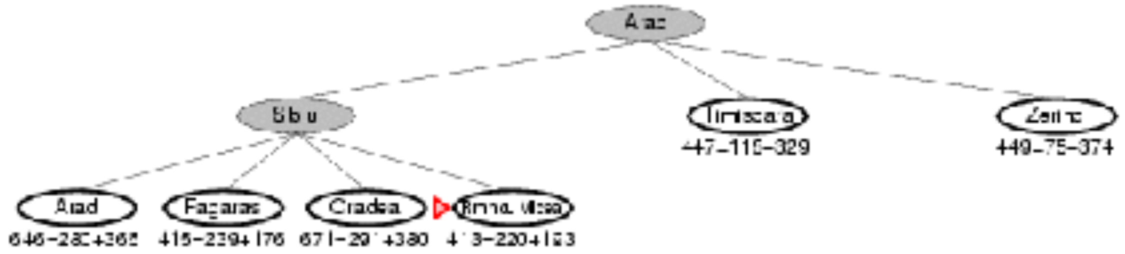
# A* Search Example

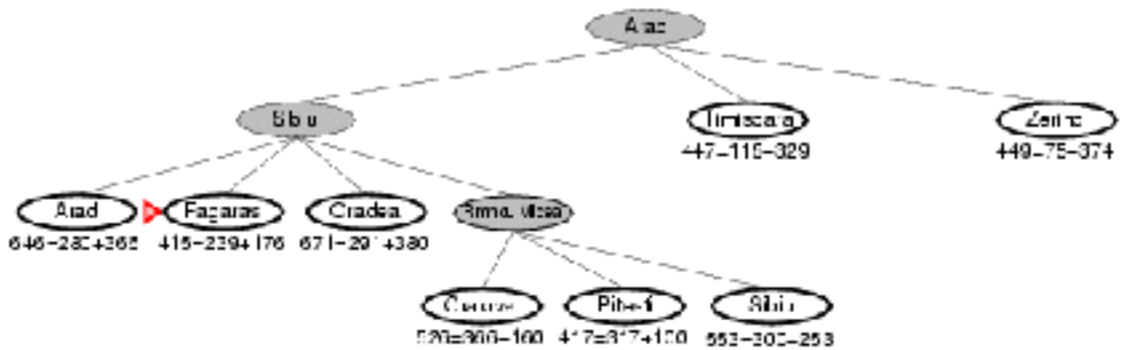


# A* Search Example

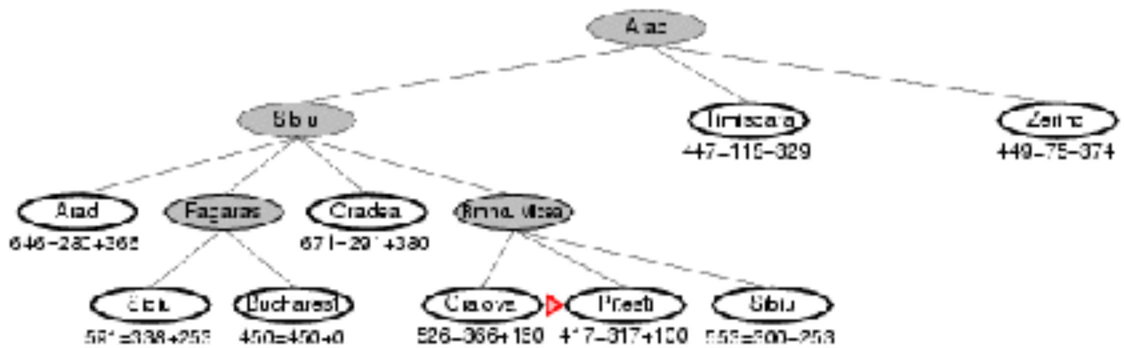


# A* Search Example

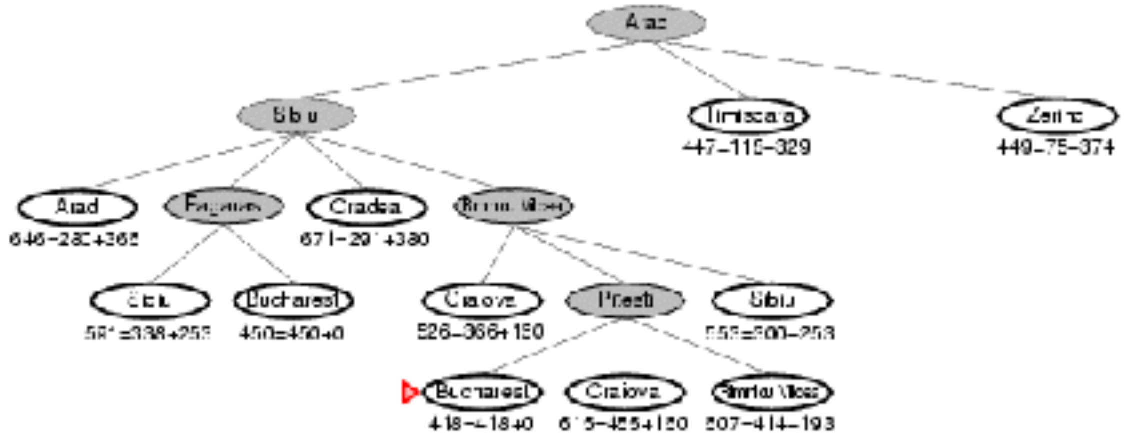


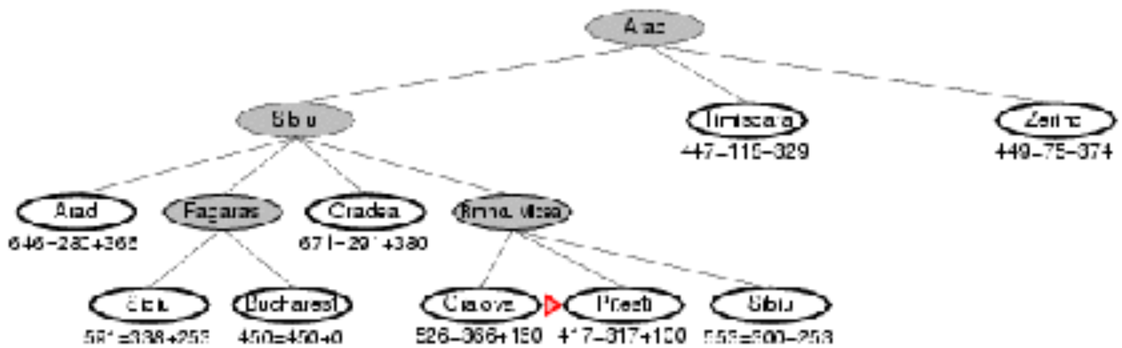
# A* Search Example

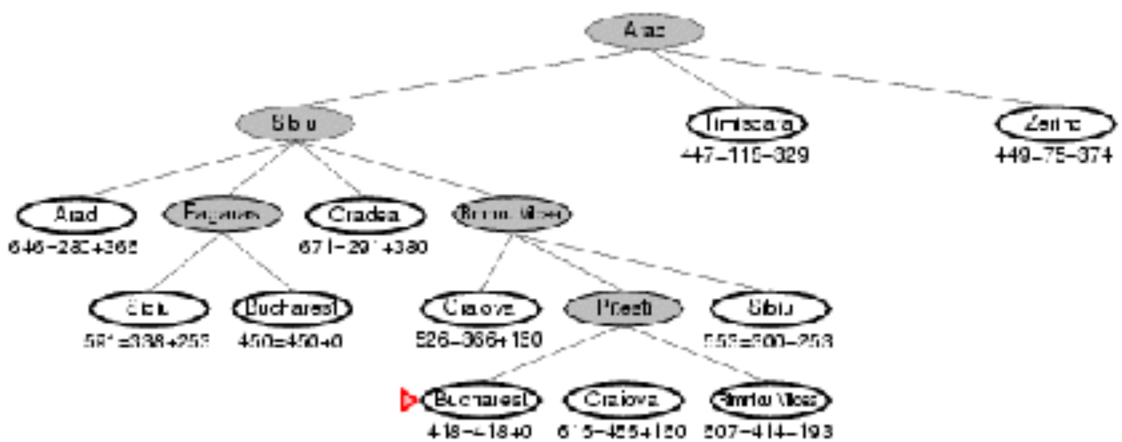


# A* Search Example

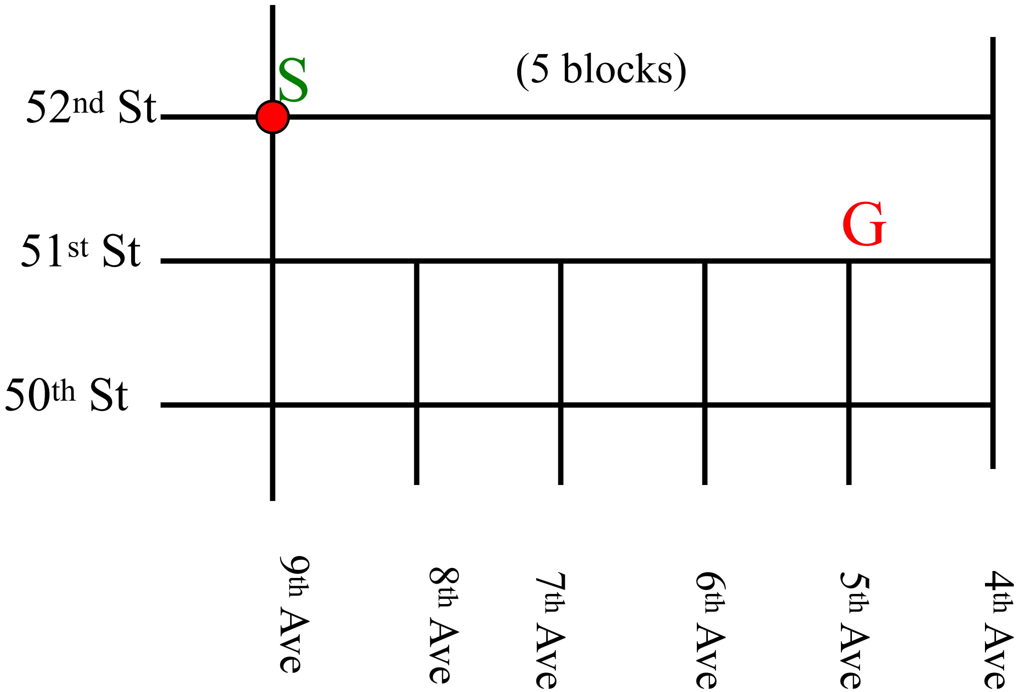


# A* Search Example



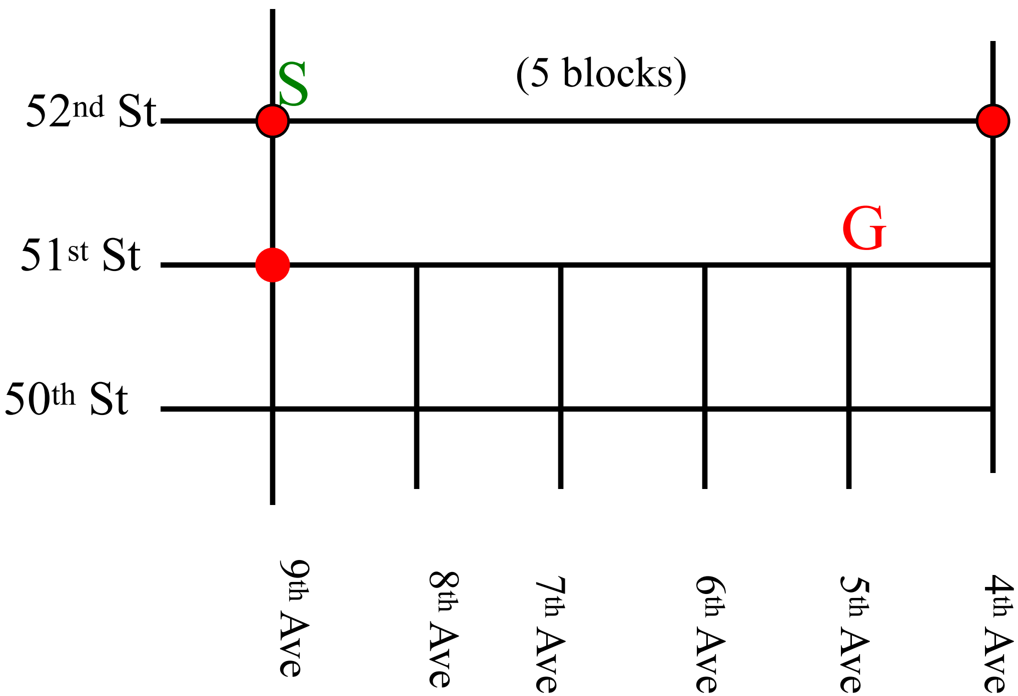



# A* in Action



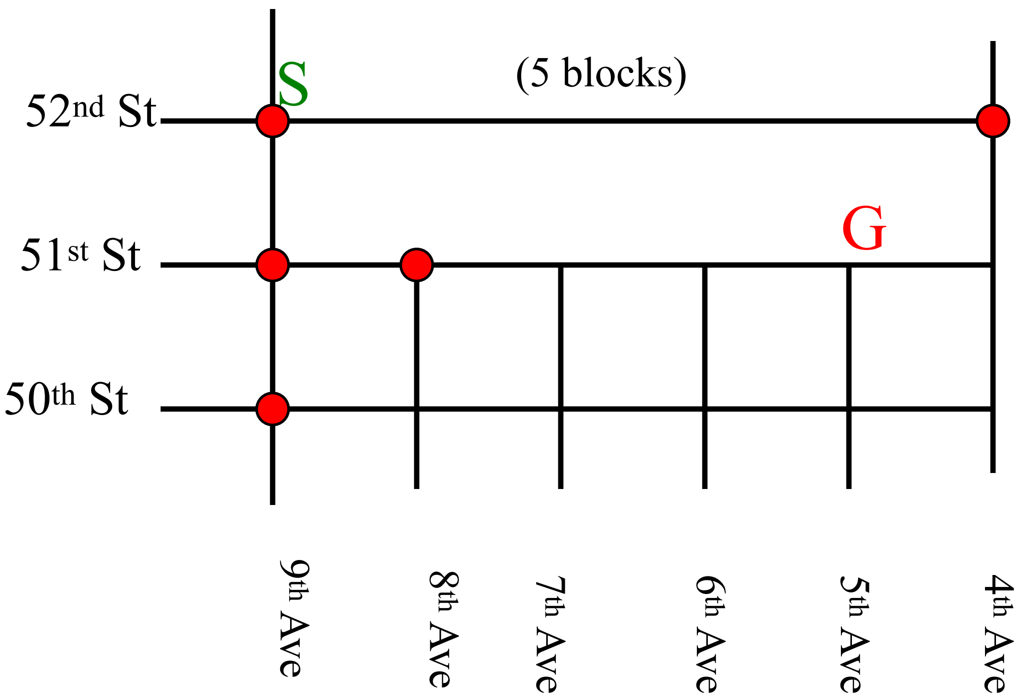

| vertex | g(n) | h(n) | f(n) |
|---|---|---|---|
| 52nd & 9th | 0 | 5 | 5 |
| | | | |
| | | | |

# A* in Action



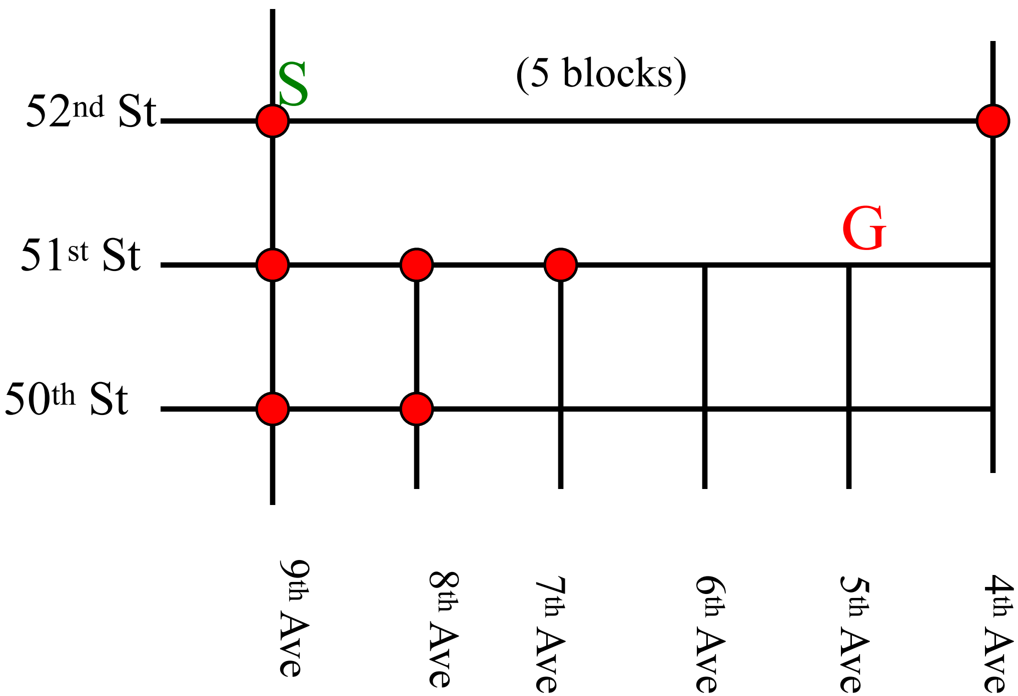

| vertex | g(n) | h(n) | f(n) |
|---|---|---|---|
| 52nd & 4th | 5 | 2 | 7 |
| 51st & 9th | 1 | 4 | 5 |
| | | | |

# A* in Action



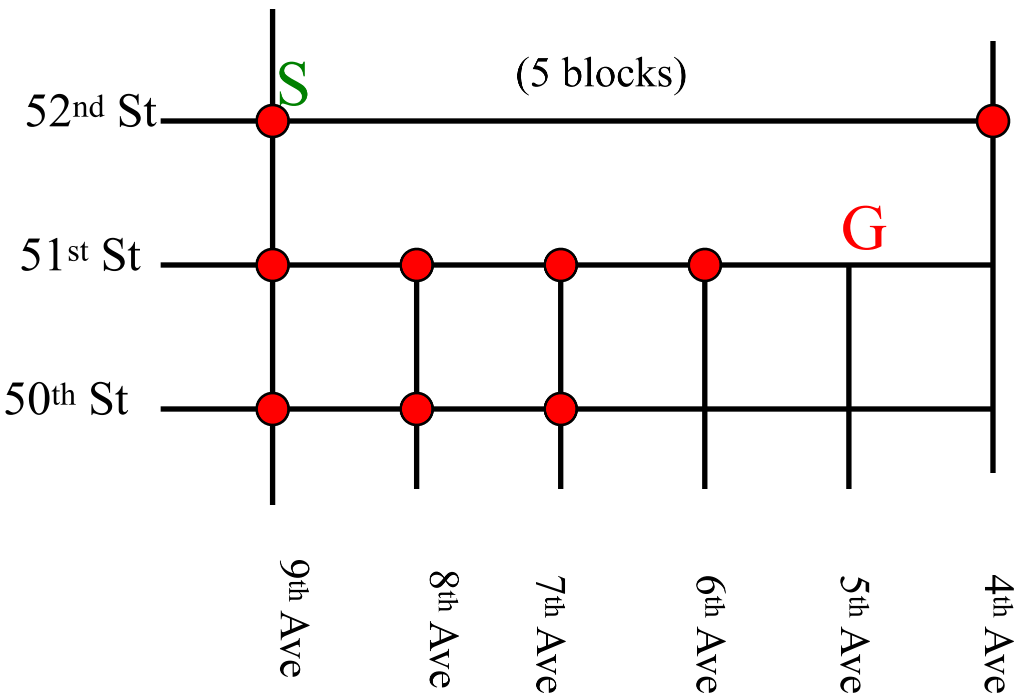

| vertex | g(n) | h(n) | f(n) |
|---|---|---|---|
| 52nd & 4th | 5 | 2 | 7 |
| 51st & 8th | 2 | 3 | 5 |
| 50th & 9th | 2 | 5 | 7 |

# A* in Action



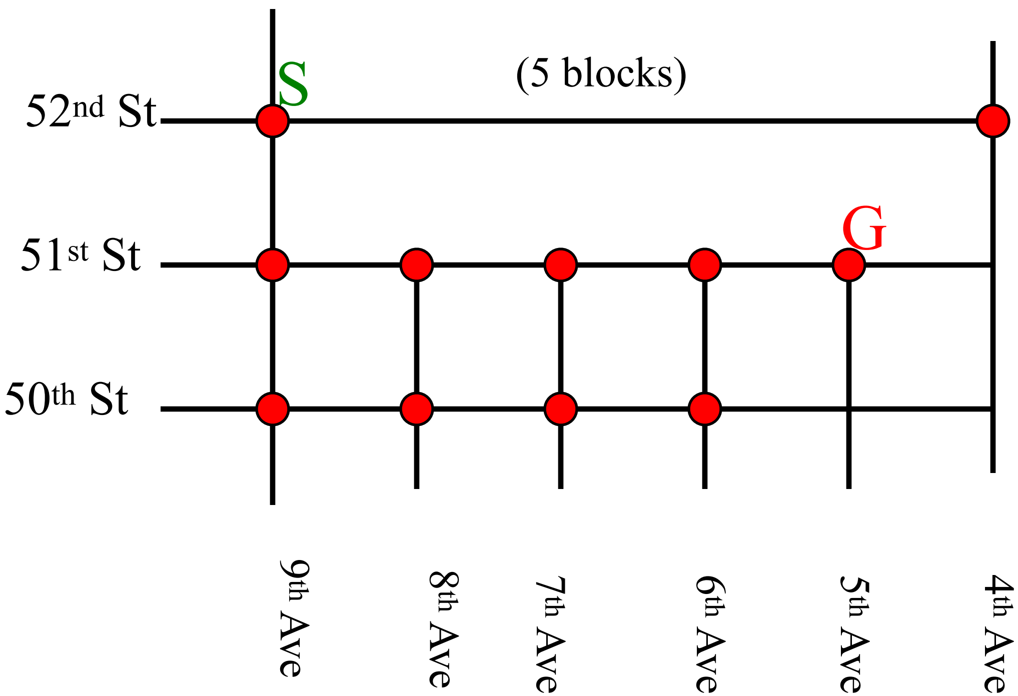

| vertex | g(n) | h(n) | f(n) |
|---|---|---|---|
| 52nd & 4th | 5 | 2 | 7 |
| 51st & 7th | 3 | 2 | 5 |
| 50th & 9th | 2 | 5 | 7 |
| 50th & 8th | 3 | 4 | 7 |

# A* in Action



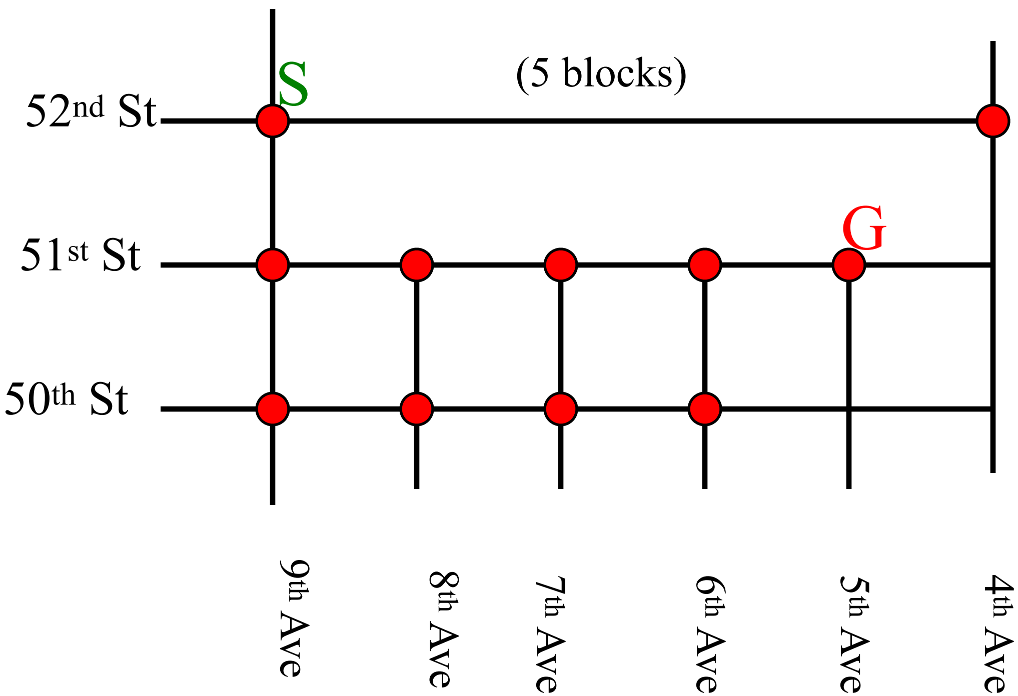

| vertex | g(n) | h(n) | f(n) |
|---|---|---|---|
| 52nd & 4th | 5 | 2 | 7 |
| 51st & 6th | 4 | 1 | 5 |
| 50th & 9th | 2 | 5 | 7 |
| 50th & 8th | 3 | 4 | 7 |
| 50th & 7th | 4 | 3 | 7 |

# A* in Action




| vertex | g(n) | h(n) | f(n) |
|---|---|---|---|
| 52nd & 4th | 5 | 2 | 7 |
| 51st & 5th | 5 | 0 | 5 |
| 50th & 9th | 2 | 5 | 7 |
| 50th & 8th | 3 | 4 | 7 |
| 50th & 7th | 4 | 3 | 7 |

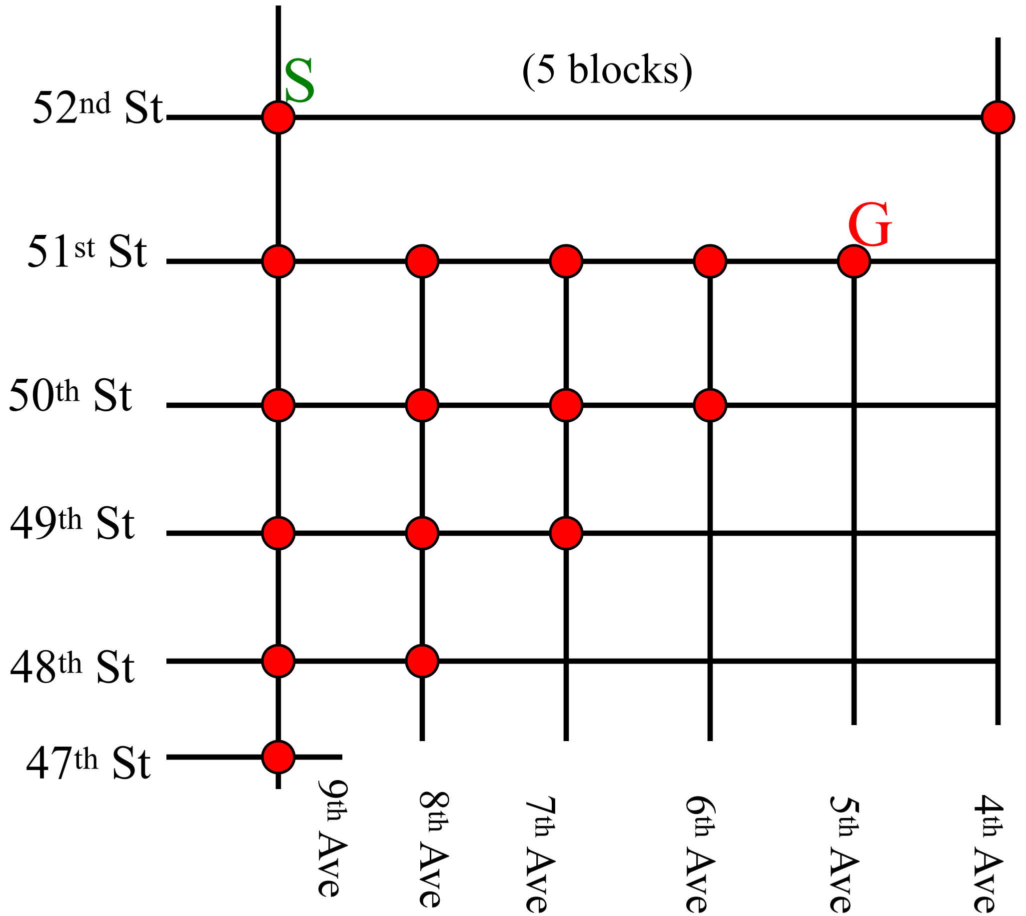
# A* in Action




| vertex | g(n) | h(n) | f(n) |
|---|---|---|---|
| 52nd & 4th | 5 | 2 | 7 |
| 50th & 9th | 2 | 5 | 7 |
| 50th & 8th | 3 | 4 | 7 |
| 50th & 7th | 4 | 3 | 7 |

*DONE!*

# What Would Dijkstra Have Done?

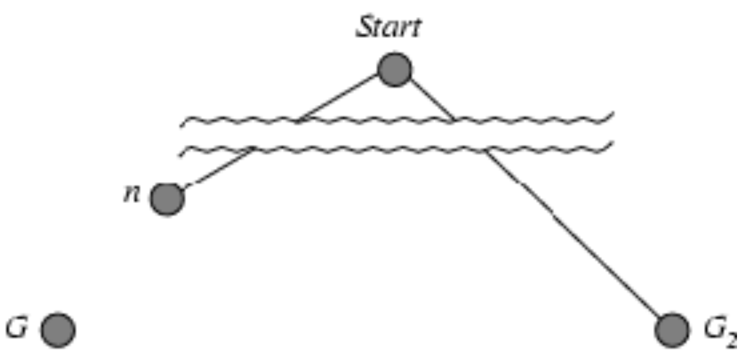


# Admissible Heuristics

- A heuristic $h(n)$ is admissible if for every node $n$, $h(n) \le h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from $n$.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If $h(n)$ is admissible, A* using TREE-SEARCH is optimal.

# Optimality of A* (proof)

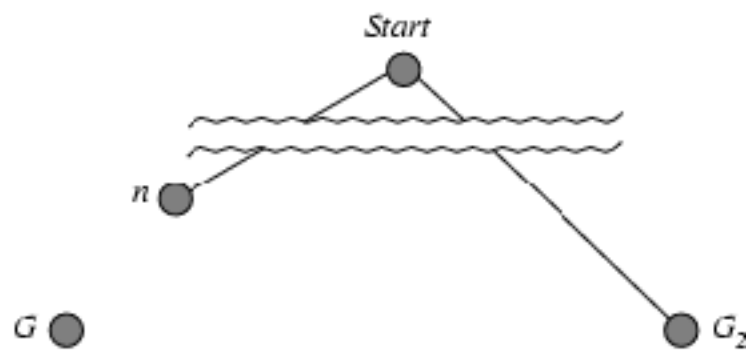
- Suppose some suboptimal goal $G_2$ has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.




- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since $G_2$ is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

# Optimality of A* (proof)

- Suppose some suboptimal goal $G_2$ has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.



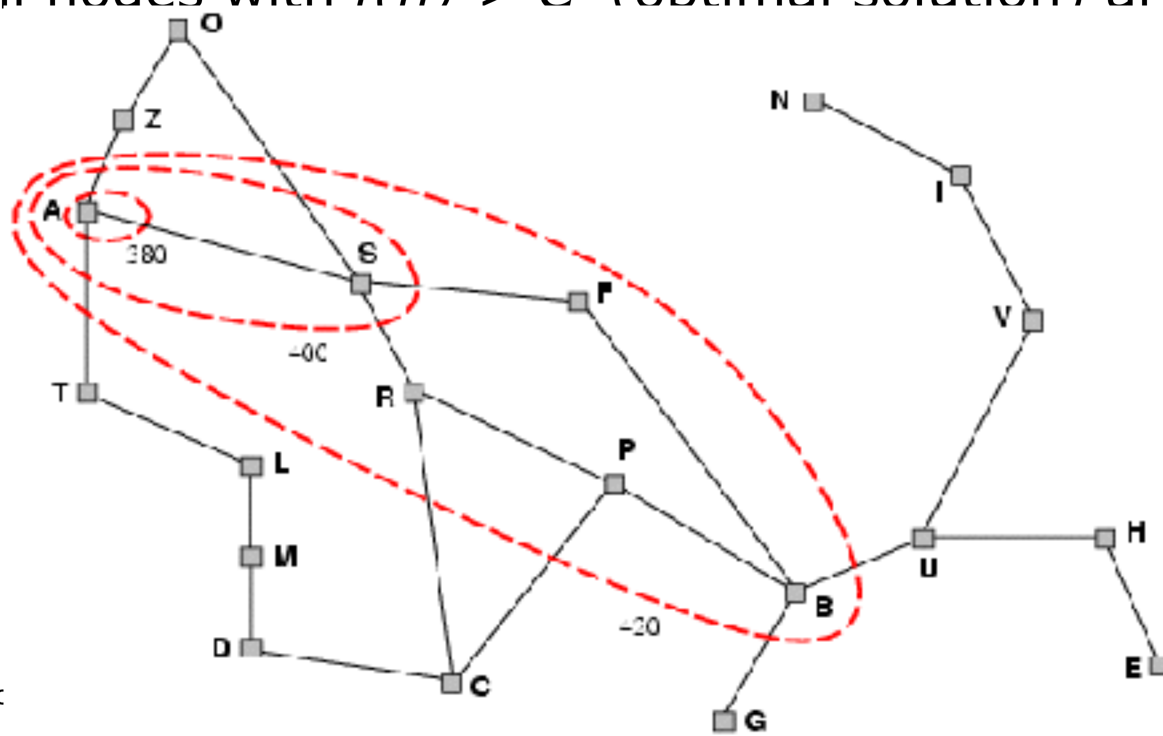

- $f(G_2) > f(G)$ from above
- $h(n) \leq h^*(n)$ since h is admissible
- $g(n) + h(n) \leq g(n) + h^*(n)$

$f(n) \leq f(G)$

- Hence $f(G_2) > f(n)$, and A* will never select $G_2$ for expansion

# Optimality of A*

- A* expands nodes in order of increasing *f* value
  - Gradually adds "*f*-contours" of nodes
  - Contour *i* has all nodes with $f=f_i$, where $f_i < f_{i+1}$
  - All nodes with $f(n) > C^*$ (optimal solution) are pruned.



# Properties of A*

- Complete? Yes (unless there are infinitely many nodes with f ≤ *f(G)* )
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

# Properties of A*

- Complete? Yes – assume finitely many nodes with $f(n) \leq f(G)$
- Optimal? Yes
- Efficient?
  - A* is *optimally efficient* for any given heuristic function. i.e. no other optimal algorithm is guaranteed to expand fewer nodes than A*.
- Time? Exponential
- Space? Exponential (all nodes in memory)

- Theorem: The search space of A* grows *exponentially* unless the error in the heuristic function grows no faster than the logarithm of the actual path cost. [p.101]

# Question?

# Memory-Bounded Heuristic Search

- Iterative deepening A*
  - Cutoff: *f*-cost
  - Not suitable for real-valued costs
  - Space:
- Recursive best-first search
  - Keeping track of the *f*-value of the best alternative path from any ancestor of the current node.
  - Linear space

# RBFS (from AIMA)

**function** RECURSIVE-BEST-FIRST-SEARCH(*problem*) **returns** a solution, or failure
  RBFS(*problem*, MAKE-NODE(INITIAL-STATE[*problem*]), $\infty$)

**function** RBFS(*problem*, *node*, *f_limit*) **returns** a solution, or failure and a new $f$-cost limit
  **if** GOAL TEST[*problem*](*state*) **then return** *node*
  *successors* $\leftarrow$ EXPAND(*node*, *problem*)
  **if** *successors* is empty **then return** *failure*, $\infty$
  **for each** s **in** *successors* **do**
    $f[s] \leftarrow \max(g(s) + h(s), f[node])$
  **repeat**
    *best* $\leftarrow$ the lowest $f$ value node in *successors*
    **if** $f[best] > f\_limit$ **then return** *failure*, $f[best]$
    *alternative* $\leftarrow$ the second-lowest $f$-value among *successors*
    *result*, $f[best] \leftarrow$ RBFS(*problem*, *best*, min(*f_limit*, *alternative*))
    **if** *result* $\neq$ *failure* **then return** *result*

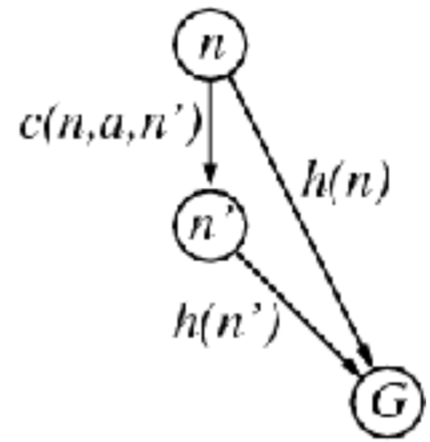
# SMA*
# Simplified Memory-bounded A*

- Expand the (newest) best leaf until memory is full.
- Drop the (oldest) worst leaf node (highest *f*-value)
- Back up the value of the "forgotten" node to its parent.
- Regenerate the sub-tree only when all other paths look worse.
- Complete? If any reachable solution exists.
- Optimal? If any optimal solution is reachable.

# Consistent Heuristics

- A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a*,

  $h(n) \le c(n,a,n') + h(n')$




- If *h* is consistent, we have

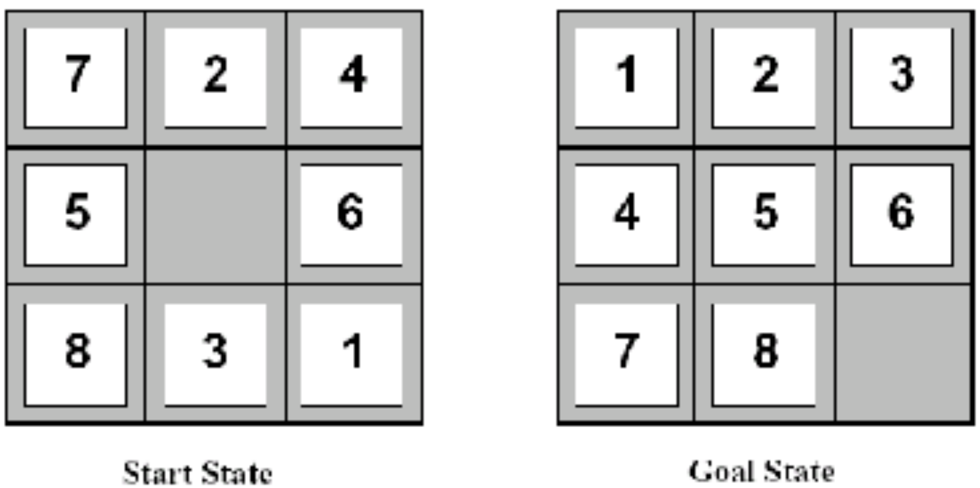
$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&\ge g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

  i.e., *f(n)* is non-decreasing along any path.

- Theorem: If *h(n)* is consistent, A* using GRAPH-SEARCH is optimal.

# 8-Puzzle

- Convert the *initial* configuration into the *goal* configuration by moving the tiles.
- Legal moves:
  - Move any tile to *adjacent empty* square

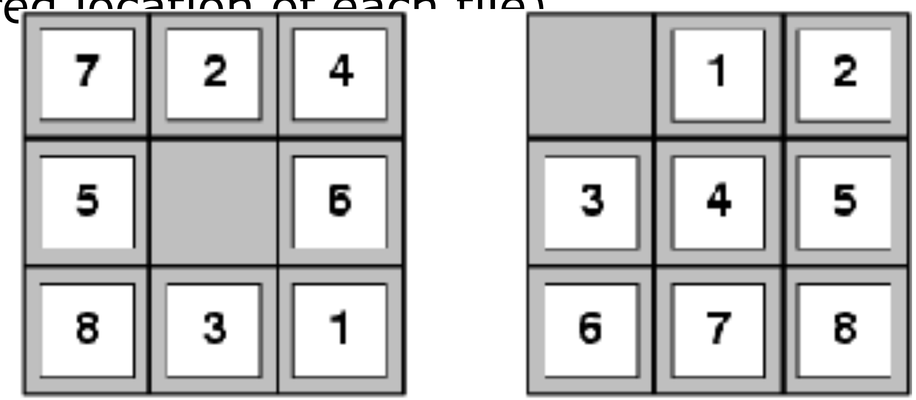


# Admissible Heuristics

E.g., 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)




Start State    Goal State

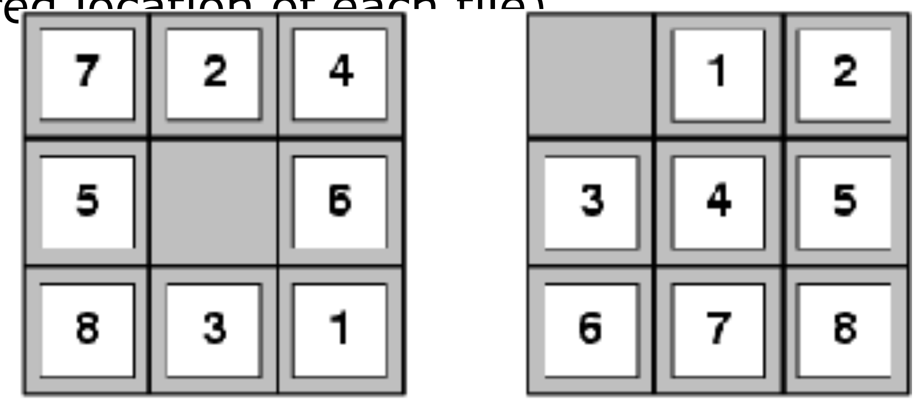
- $h_1(S) = ?$
- $h_2(S) = ?$

# Admissible Heuristics

E.g., 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)



Start State Goal State

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

# Dominance

- If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible)
- then $h_2$ dominates $h_1$
- $h_2$ is better for search

- Typical search costs (average number of nodes expanded):

- *d=12* IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes
- *d=24* IDS = too many nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

# Importance of Heuristics

| | | |
|---|---|---|
| 7 | 2 | 3 |
| 4 | 1 | 6 |
| 8 | 5 | |

- h1 = number of tiles in the wrong place
- h2 = sum of distances of tiles from correct location

| D | IDS | A*(h1) | A*(h2) |
|---|---|---|---|
| 2 | 10 | 6 | 6 |
| 4 | 112 | 13 | 12 |
| 6 | 680 | 20 | 18 |
| 8 | 6384 | 39 | 25 |
| 10 | 47127 | 93 | 39 |
| 12 | 364404 | 227 | 73 |
| 14 | 3473941 | 539 | 113 |
| 18 | | 3056 | 363 |
| 24 | | 39135 | 1641 |

# Inventing Heuristic Functions

- Relaxed problems: the cost of an exact solution to a relaxed problem is often a good heuristic for the original problem. e.g.
  - A if B and C
  - A if B
  - A if C
  - A
- Composite heuristics
  - $h(n) = \max(h_1(n),\ldots,h_m(n))$
- Weighted evaluation function
  - $f_w(n) = (1-w)g(n) + w\,h(n)$
- Learn the coefficients for features of a state
  - $h(n) = c_1x_1(n) + \ldots + c_kx_k(n)$
- Statistical information
- Search cost
  - Good heuristics should be efficiently computable.

# Relaxed Problems

- A problem with fewer restrictions on the actions is called a relaxed problem

- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- Original: move any tile to *adjacent empty* square
- Relaxed problems:
  - Move from A to B, if A is *adjacent* to B.
    - Manhattan distance
  - Move from A to B, if B is *empty*.
    - Gaschnig's heuristic (1979)
  - Move from A to B. e.g. a tile can be moved to anywhere
    - Misplaced tiles

# Composite Heuristics

- Given a collection of admissible heuristics $h_1 \ldots h_n$ for a problem, and none of them dominates any of the others, which should we choose?

# Finding Optimal Solutions

- Input: A random solvable initial state
- Output: A shortest sequence of moves that maps the initial state to the goal state

- *Generalized sliding-tile puzzle is NP Complete (Ratner and Warmuth, 1986)*
  - People can't find optimal solutions.
  - Progress measured by size of problems that can be solved optimally.

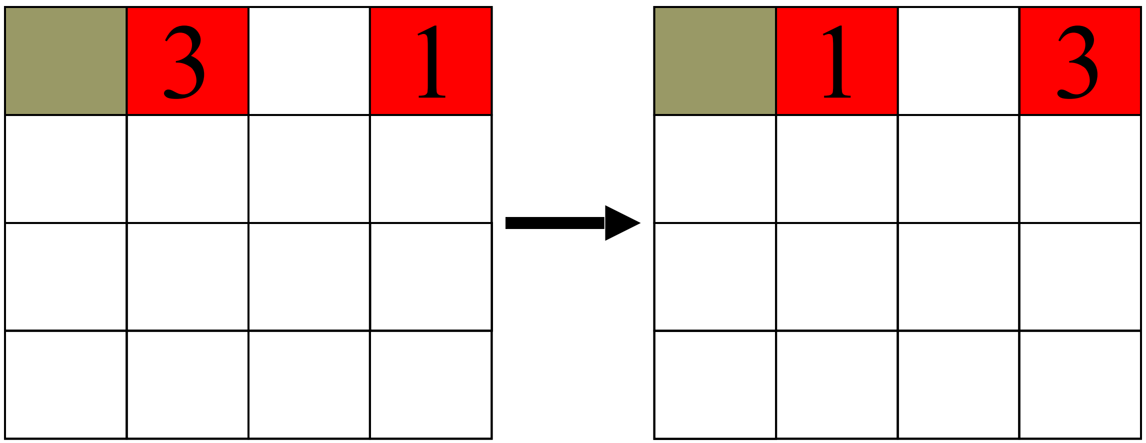
# Fifteen Puzzle

- Invented by Sam Loyd in 1870s
- "...engaged the attention of nine out of ten persons of both sexes and of all ages and conditions of the community."
- $1000 prize to swap positions of two tiles

| | | | |
|---|---|---|---|
| | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 |

- Adapted from "Recent Progress in the Design and Analysis of Admissible Heuristic Functions" by R. Korf.

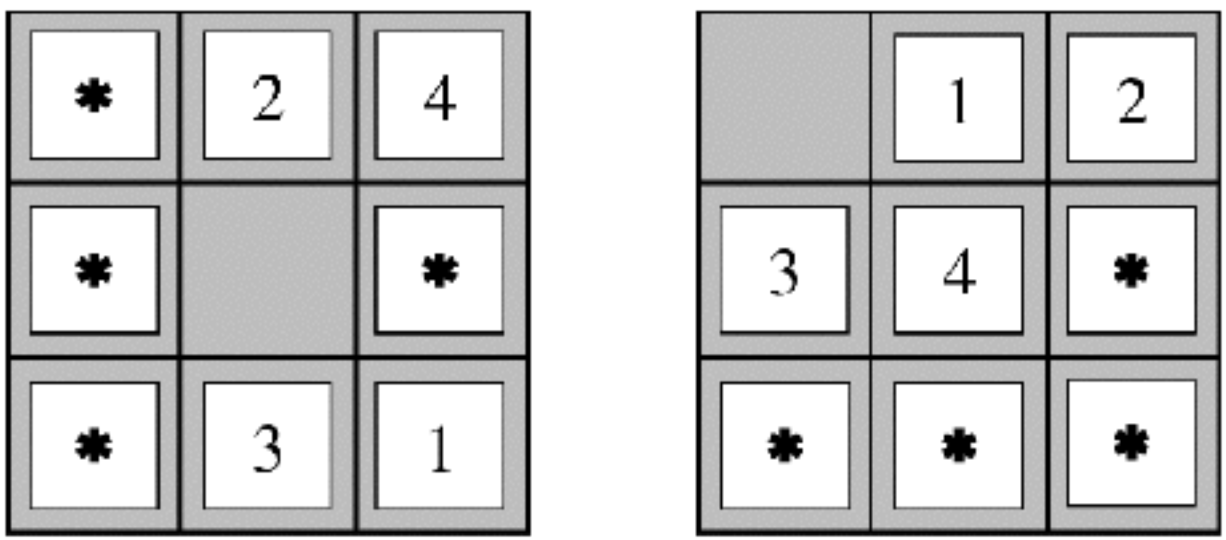
# Linear Conflict Heuristic




- Hansson, Mayer, and Yung, 1991
- Given two tiles in their goal row, but reversed in position, additional vertical moves can be added to Manhattan distance.
- Still not accurate enough to solve 24-Puzzle
- We can generalize this idea further.

# Sub-Problems

- Admissible heuristics can also be derived from the solution cost of a sub-problem of a given problem.

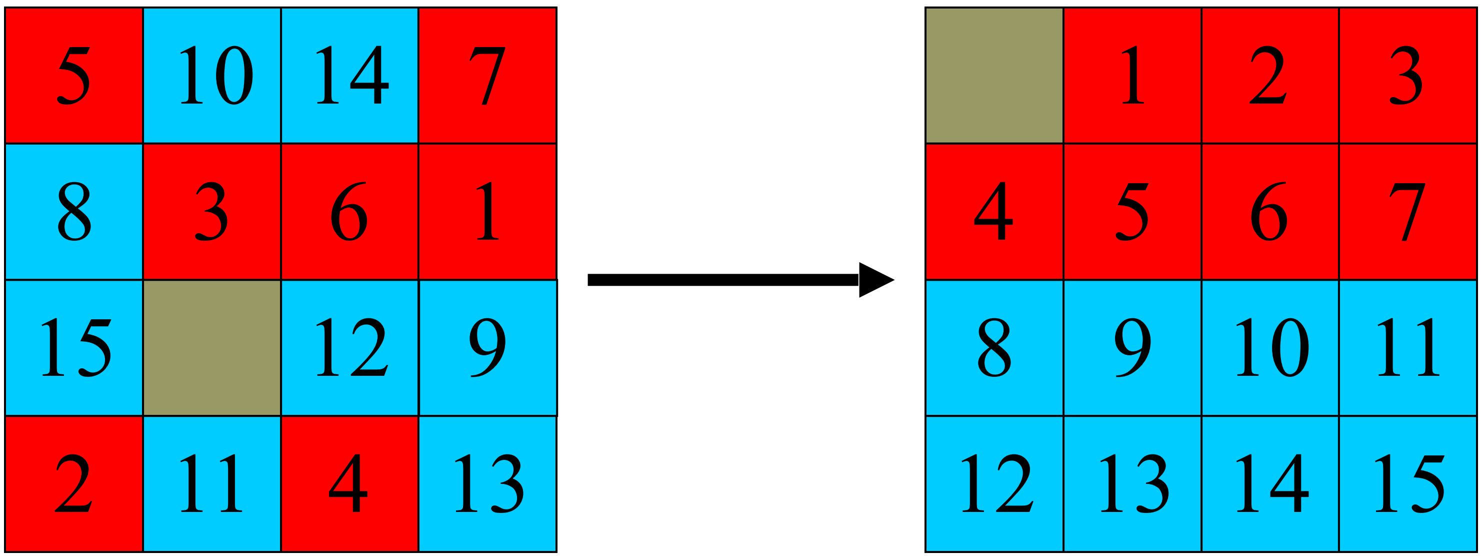



Start State Goal State

# Pattern Database Heuristics

- A pattern database is a complete set of such positions, with associated number of moves.
  - e.g. a 8-tile pattern database for the Fifteen Puzzle contains 519 million entries.
- On 15 puzzle, IDA* with pattern database heuristics is about 10 times faster than with Manhattan distance
  - Culberson and Schaeffer, 1996
- Pattern databases can also be applied to Rubik's Cube.

# Additive Databases




- The 7-tile database contains 58 million entries.
  - 20 moves needed to solve red tiles
- The 8-tile database contains 519 million entries.
  - 25 moves needed to solve blue tiles
- Overall heuristic is 20+25=45 moves

# Swap Two Tiles

| | | | |
|---|---|---|---|
| | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |
| 12 | 13 | 15 | 14 |

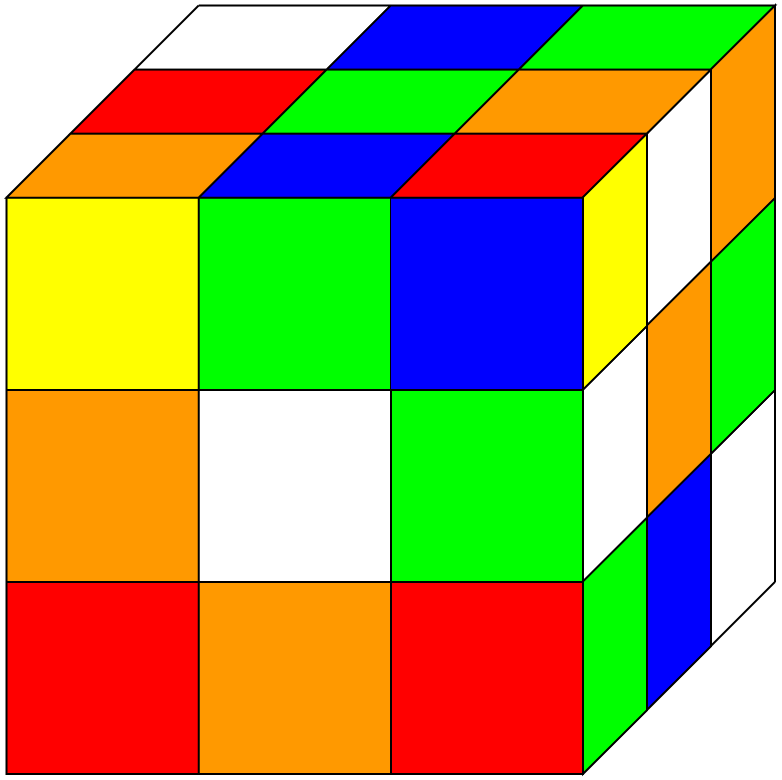
→

| | | | |
|---|---|---|---|
| | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 |

(Johnson & Storey, 1879) proved it's impossible.

# Rubik's Cube

- Invented in 1974 by Erno Rubik of Hungary
- Over 100 million sold worldwide
- Most famous combinatorial puzzle ever



# Sizes of Problem Spaces

| Problem | Nodes | Brute-Force Search Time (10 million nodes/second) |
| --- | --- | --- |
| 8 Puzzle: | $10^5$ | .01 seconds |
| $2^3$ Rubik's Cube: | $10^6$ | .2 seconds |
| 15 Puzzle: | $10^{13}$ | 6 days |
| $3^3$ Rubik's Cube: | $10^{19}$ | 68,000 years |
| 24 Puzzle: | $10^{25}$ | 12 billion years |