

# **Solving Problems by Searching**

## **Informed Search**

**COMP3411/9814: Artificial Intelligence**

# Overview

- Heuristics
- Informed Search Methods
  - Best-first search
  - Greedy best-first search
  - $A^*$  search
  - Iterative Deepening  $A^*$  Search



# Informed (Heuristic) Search

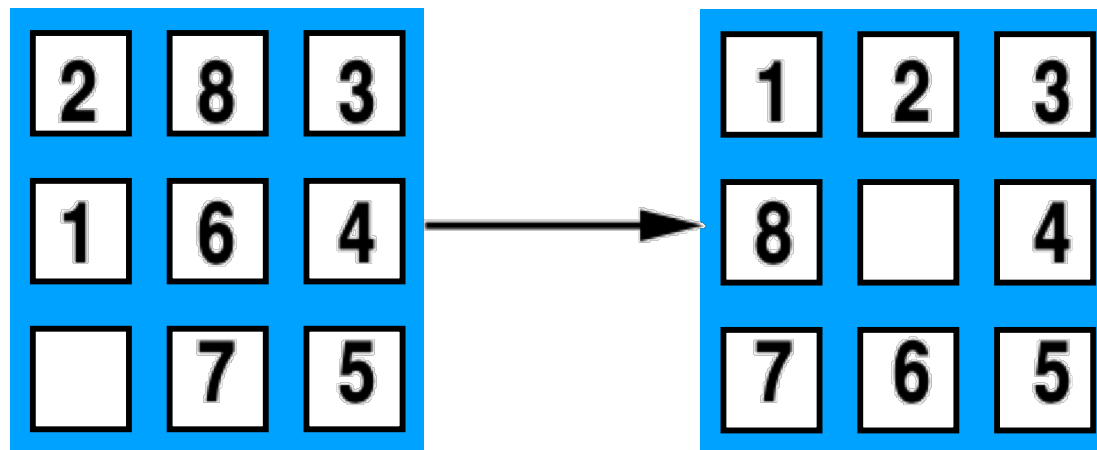
- Informed search strategy
  - use problem-specific knowledge more efficiently than uninformed search
- Uninformed search algorithms have no information about problem other than its definition.
  - some can solve any solvable problem, none of them can do it efficiently
- Informed search algorithms can do well given guidance on where to look for solutions.
- Implemented using a **priority queue** to store frontier nodes

# Heuristics

- Heuristics are “rules of thumb” for deciding which alternative is best
- Heuristic must ***underestimate*** actual cost to get from current node to goal
  - Called an **admissible heuristic**
- Denoted  ***$h(n)$*** 
  - $h(n) = 0$  when ever  $n$  is a **goal** node

# Heuristics — Example

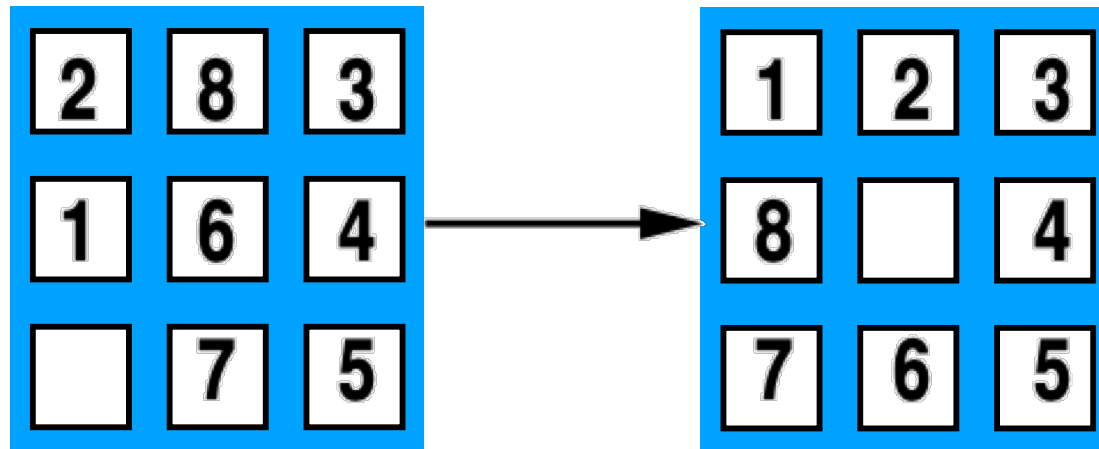
8-Puzzle — number of tiles out of place



$$h(n) = 5$$

# Heuristics — Example

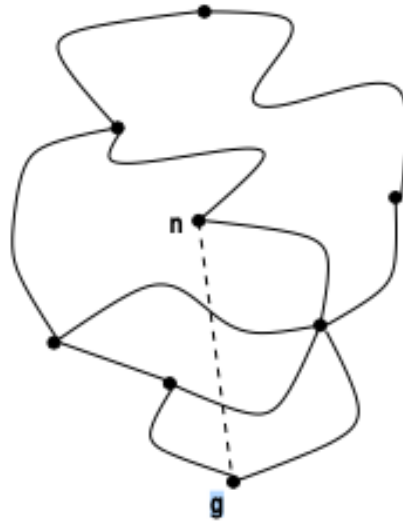
8-Puzzle — Manhattan distance (distance tile is out of place)



$$h(n) = 1 + 1 + 0 + 0 + 0 + 1 + 1 + 2 = 6$$

# Heuristics – Example

Another common heuristic is the straight-line distance (“as the crow flies”) from node to goal



Therefore  $h(n) = \text{distance from } n \text{ to } g$

# Uniformed vs Informed Search

- Uninformed - keeps searching until it stumbles on goal
  - No domain knowledge
- Informed - searches in direction of best guess to goal
  - Uses domain knowledge



# Example Heuristic Functions

- If nodes are points on a Euclidean plane and cost is distance,  $h(n)$  can be straight-line distance from  $n$  to closest goal.
- If nodes are locations and cost is time, can use distance to goal divided by maximum speed.
- If goal is to collect a bunch of coins and not run out of fuel, cost is an estimate of how many steps to collect rest of the coins, refuel when necessary, and return to goal.
- Heuristic function can be found by simplifying calculation of true cost

# Search Strategies

General Search algorithm:

- add initial state to queue
- repeat:
  - take node from front of queue
  - test if it is a goal state; if so, terminate
  - “expand” it, i.e. generate successor nodes and add them to the queue

Search strategies are distinguished by the order in which new nodes are added to the queue of nodes awaiting expansion.

# Search Strategies

- **BFS** and **DFS** treat all new nodes the same way:
  - **BFS** add all new nodes to the back of the queue
  - **DFS** add all new nodes to the front of the queue
- **Best First Search** uses an evaluation function  $f()$  to order the nodes in the queue
  - Similar to uniform cost search
- **Informed** or **Heuristic**:
  - Greedy Search  $f(n) = h(n)$  (estimates cost from node  $n$  to goal)
  - A\* Search  $f(n) = g(n) + h(n)$  (cost from start to  $n$  plus estimated cost to goal)

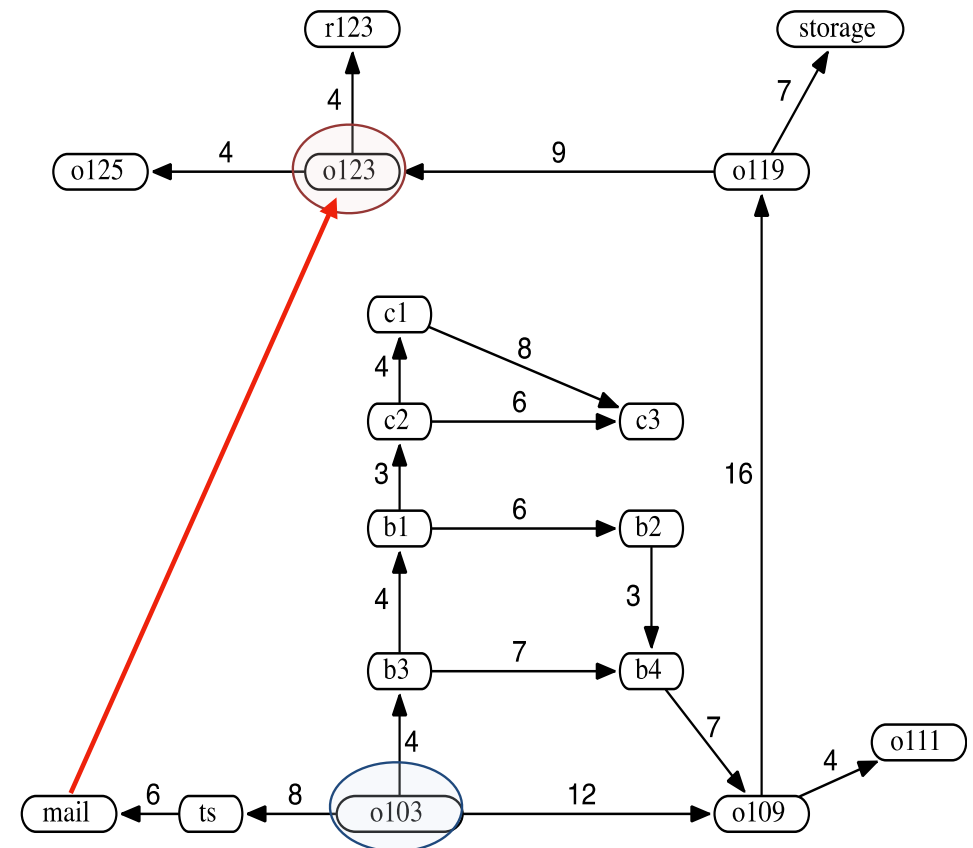
# Delivery Robot Heuristic Function

Use straight-line distance as heuristic, and assume these values:

$h(mail) = 26$	$h(ts) = 23$	$h(o103) = 21$
$h(o109) = 24$	$h(o111) = 27$	$h(o119) = 11$
$h(o123) = 4$	$h(o125) = 6$	$h(r123) = 0$
$h(b1) = 13$	$h(b2) = 15$	$h(b3) = 17$
$h(b4) = 18$	$h(c1) = 6$	$h(c2) = 10$
$h(c3) = 12$	$h(storage) = 12$	

$h(loc) = \text{distance from } loc \text{ to goal}$

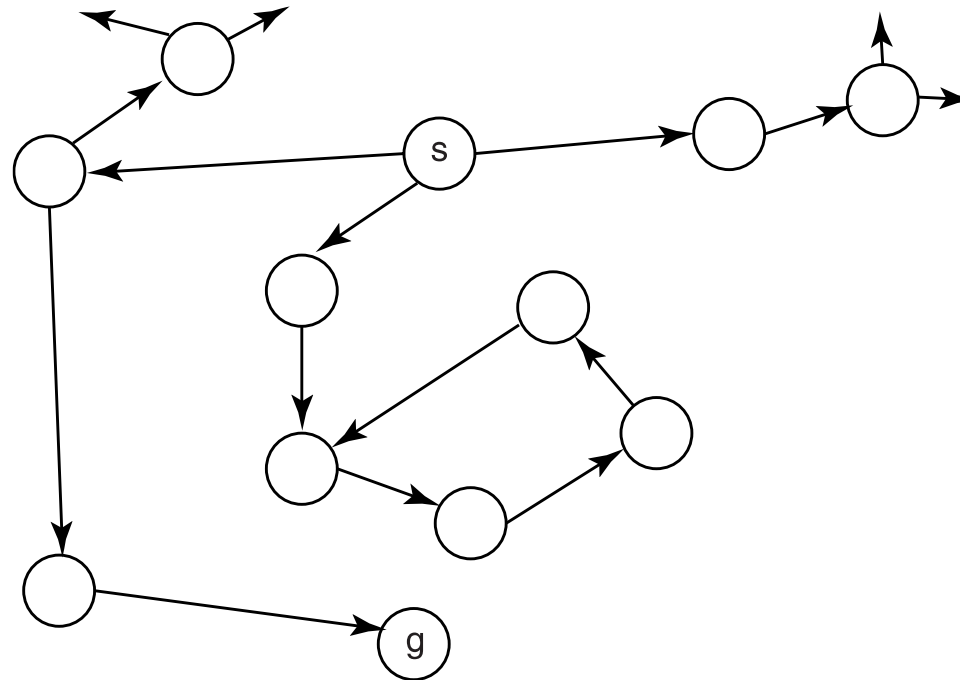
Heuristic function can be extended to paths by making heuristic value of path equal to heuristic value of node at the end of the path:  $h(\langle n_0, \dots, n_k \rangle) = h(n_k)$ .



# Greedy Best-First Search

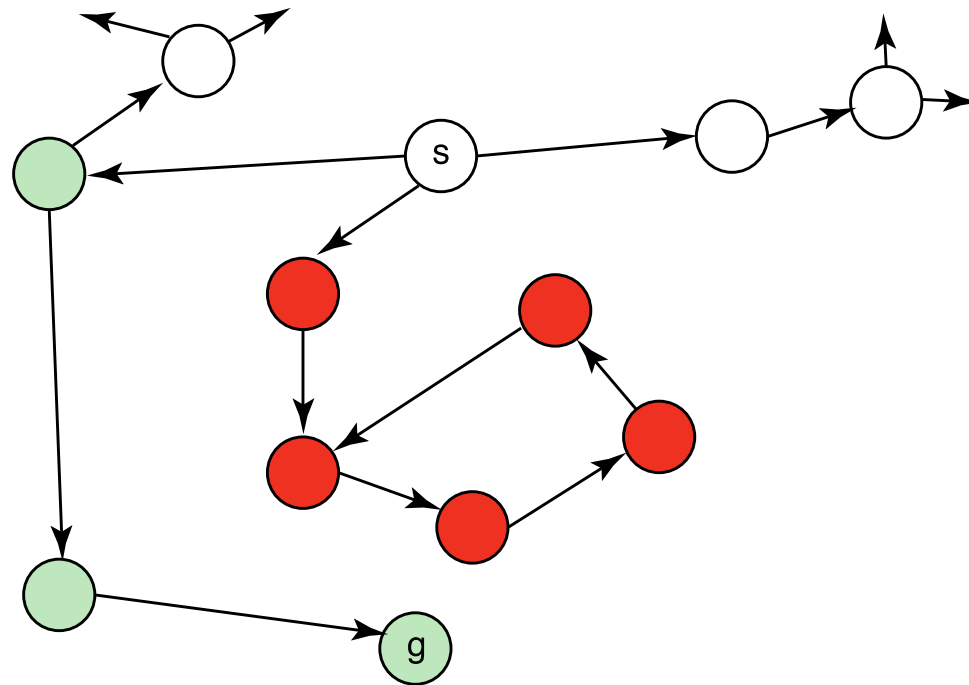
- Always select node closest to goal according to heuristic function
- $h(n)$  is estimated cost to goal
  - $h(n) = 0$  if  $n$  is a goal state
- Frontier is a priority queue ordered by  $h$ .
- “Greedy” algorithm takes “best” node first.
- Like depth-first search, except pick next node by  $h(n)$

# Greedy Best-first Search Example



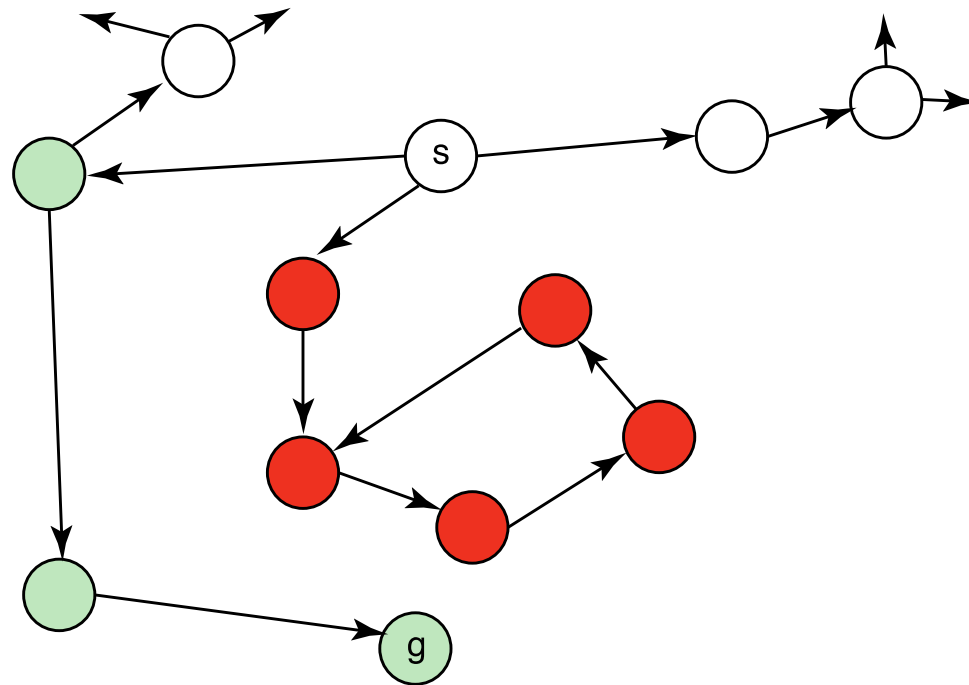
- Graph drawn to scale – cost of arc is its length
- Aim is to find shortest path from **s** to **g**.

# Greedy Best-first Search Example



Straight-line distance to goal **g** is used as heuristic function.

# Greedy Best-first Search Example

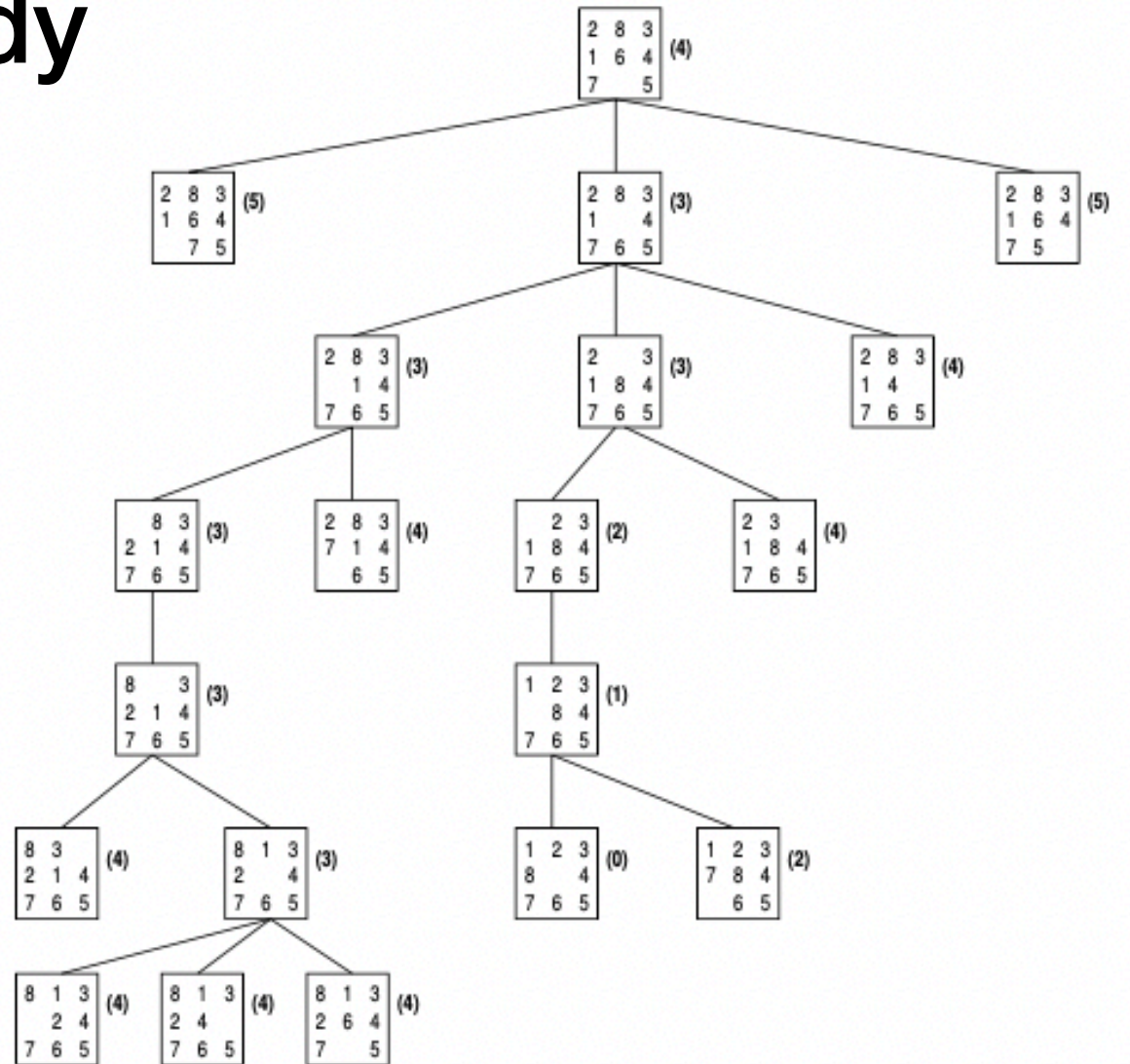


- Greedy depth-first search selects closest node to  $s$  and never terminates
- All nodes below  $s$  look good. Greedy best-first search cycles between them, never trying alternate route.

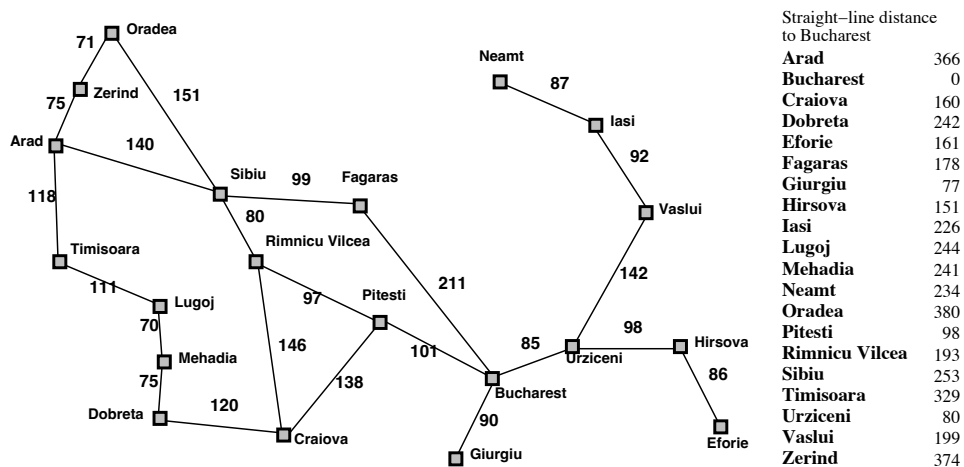


# Examples of Greedy Best-First Search

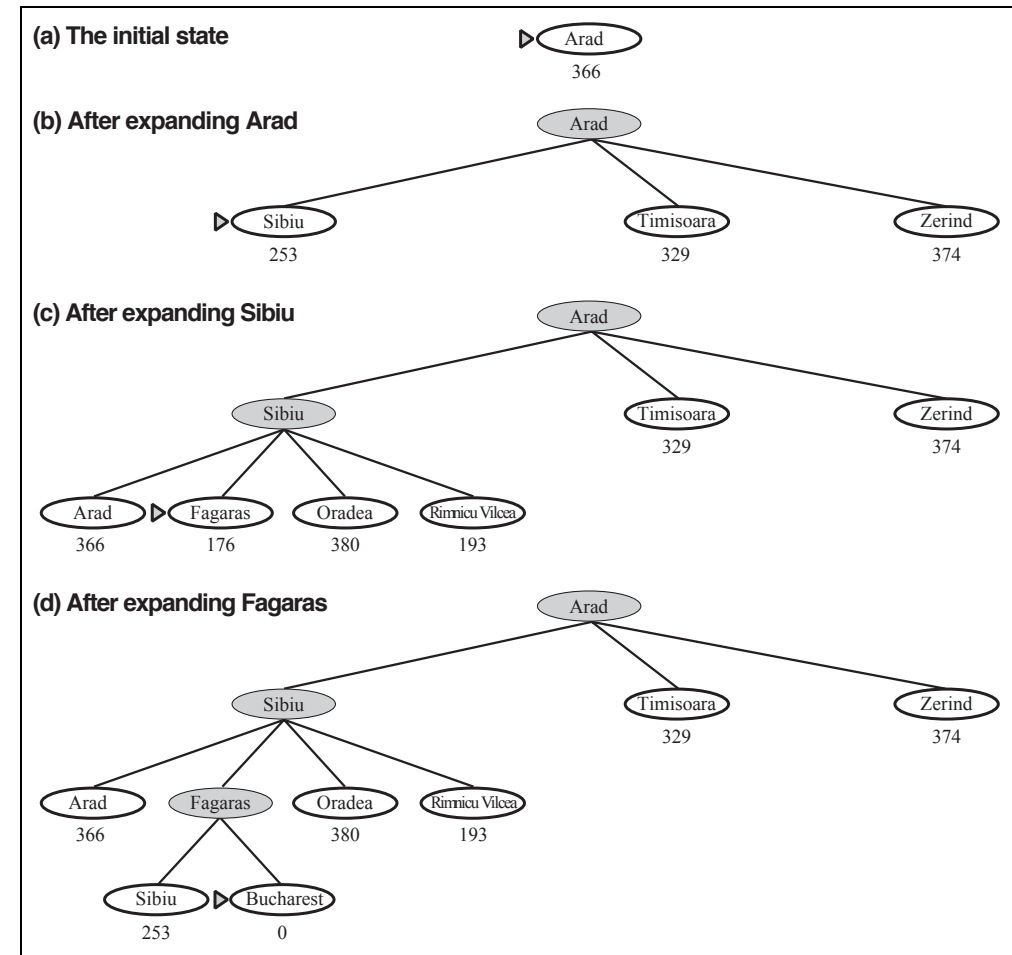
$h$  is number of tiles out of place



# Examples of Greedy Best-First Search



- Stages in a greedy best-first tree search for route from Arad to Bucharest with the straight-line distance heuristic.
- Note that **straight-line distances are less than actual distances** in map.



# Properties of Greedy Best-First Search

**Complete:** No. Can get stuck in loops.  
(Complete in finite space with repeated-state checking)

**Time:**  $O(b^m)$ , where  $m$  is the maximum depth in search space.

**Space:**  $O(b^m)$  (retains all nodes in memory)

**Optimal:** No.

- Greedy Search has the same deficits as Depth-First Search.
- However, a good heuristic can reduce time and memory costs substantially.

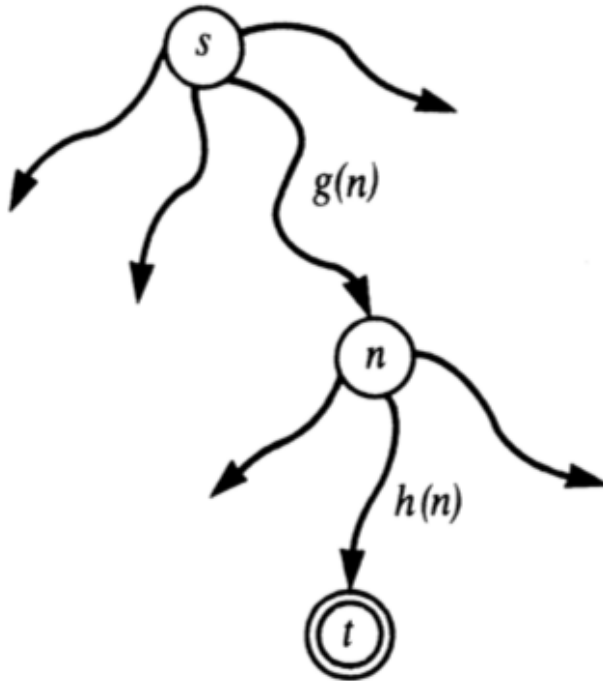
# A\* Search

- Use both cost of path generated and estimate to goal to order nodes on the frontier
  - $g(n)$  = cost of path from start to  $n$
  - $h(n)$  = estimate from  $n$  to goal
- Order priority queue using function  $f(n) = g(n) + h(n)$
- $f(n)$  is the estimated cost of the cheapest solution extending this path

# A\* Search

- Combines uniform-cost search and greedy search
- Greedy Search minimises  $h(n)$ 
  - efficient but not optimal or complete
- Uniform Cost Search minimises  $g(n)$ 
  - optimal and complete but not efficient

# Heuristic Function

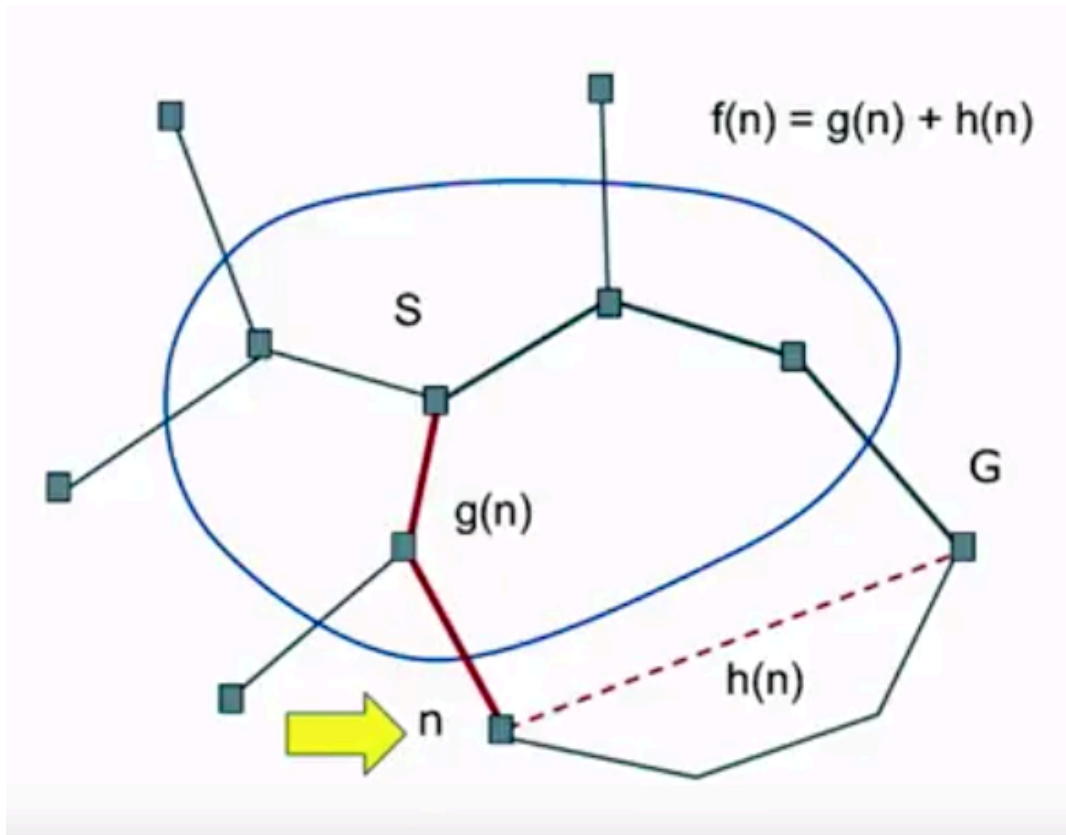


Heuristic estimate  $f(n)$  = cost of the cheapest path from  $s$  to  $t$  via  $n$ :  $f(n) = g(n) + h(n)$

$g(n)$  is the cost the path from  $s$  to  $n$

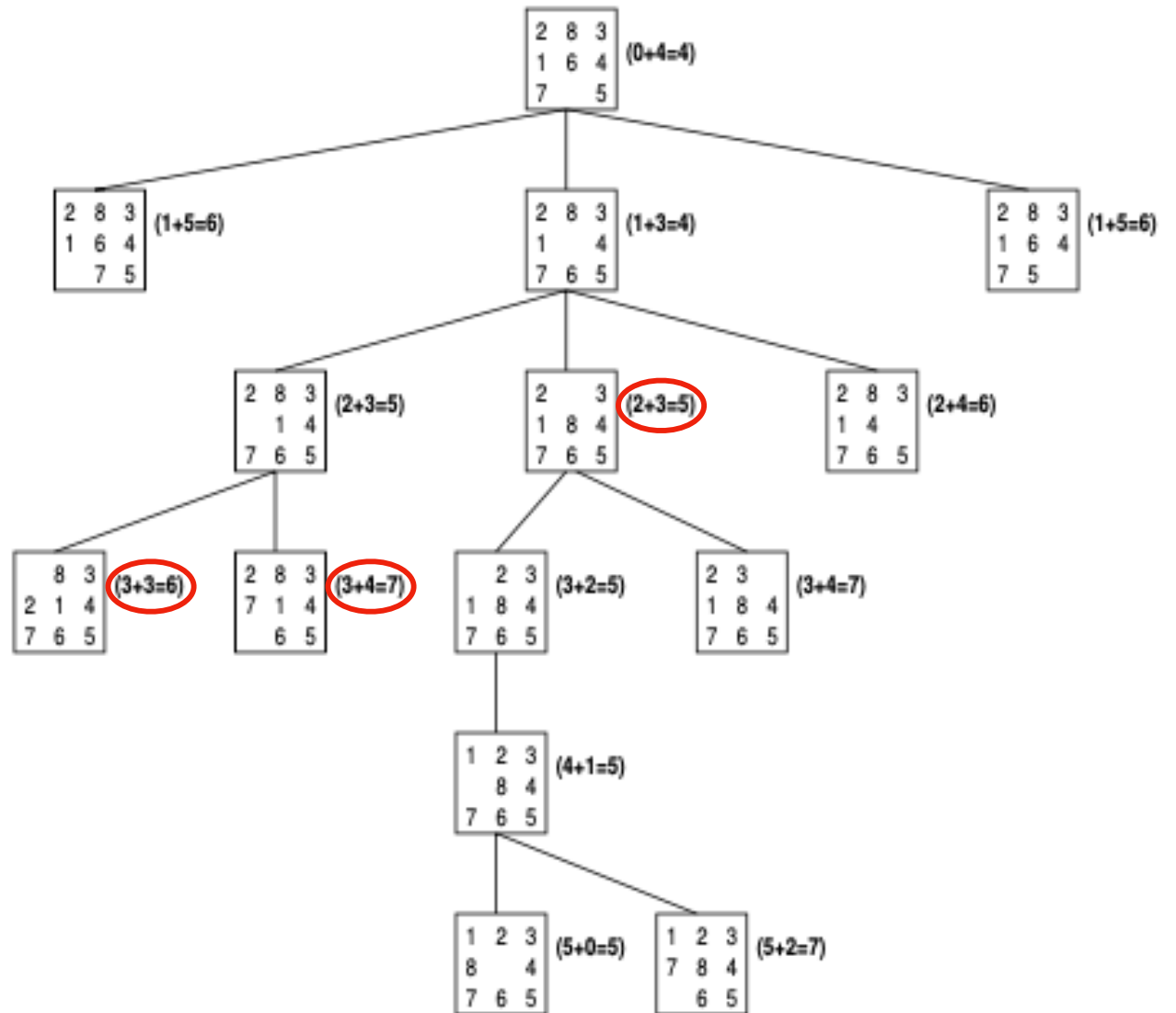
$h(n)$  is an estimate of the cost of an optimal path from  $n$  to  $t$ .

# A\* Search



- **S** = start
- **G** = Goal
- **n** = current node
- $g(n)$  = actual cost from **S** to **n**
- $h(n)$  = estimated distance from **n** to **G**

# A\* Search

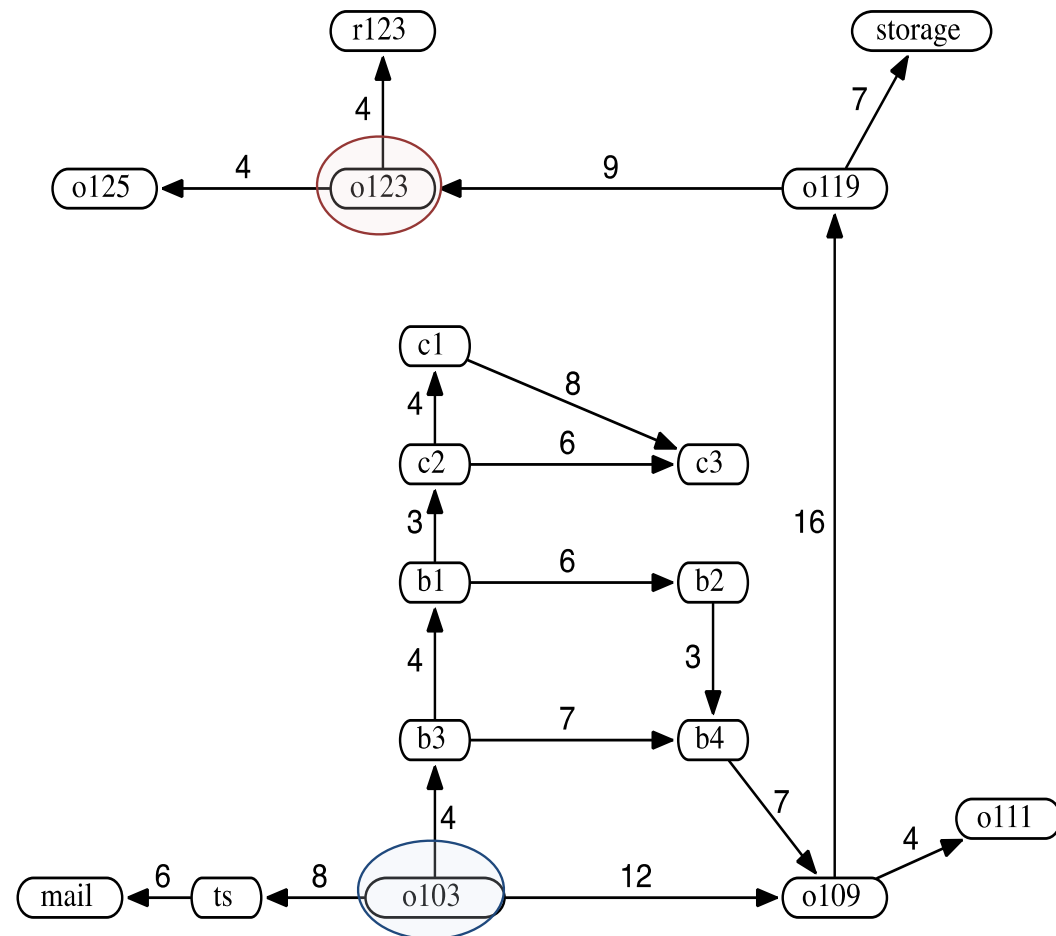




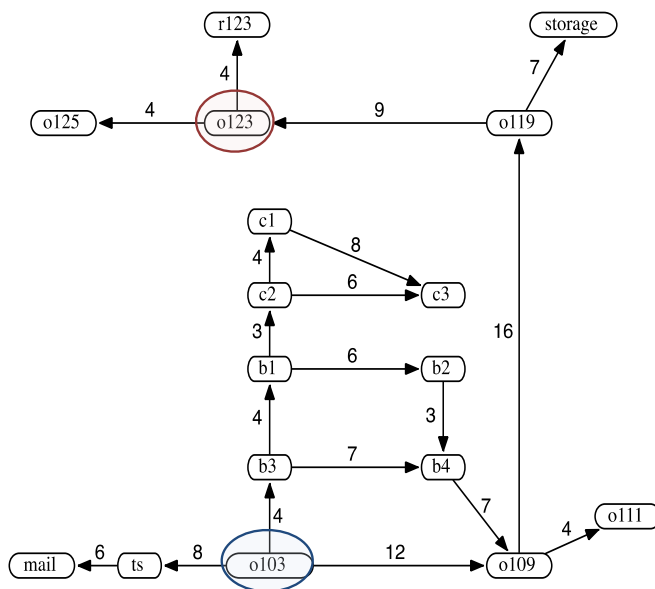
# Delivery Robot Heuristic Function

Use straight-line distance as heuristic, and assume these values:

$$\begin{array}{lll}
 h(mail) = 26 & h(ts) = 23 & h(o103) = 21 \\
 h(o109) = 24 & h(o111) = 27 & h(o119) = 11 \\
 h(o123) = 4 & h(o125) = 6 & h(r123) = 0 \\
 h(b1) = 13 & h(b2) = 15 & h(b3) = 17 \\
 h(b4) = 18 & h(c1) = 6 & h(c2) = 10 \\
 h(c3) = 12 & h(storage) = 12 & 
 \end{array}$$



# A\* Search - The Delivery Robot



$h(mail) = 26$      $h(ts) = 23$      $h(o103) = 21$   
 $h(o109) = 24$      $h(o111) = 27$      $h(o119) = 11$   
 $h(o123) = 4$      $h(o125) = 6$      $h(r123) = 0$   
 $h(b1) = 13$      $h(b2) = 15$      $h(b3) = 17$   
 $h(b4) = 18$      $h(c1) = 6$      $h(c2) = 10$   
 $h(c3) = 12$      $h(storage) = 12$

1.  $[o103_{21}]$      $h(o103) = 21$
2.  $[b3_{21}, ts_{31}, o109_{36}]$      $f(\langle o103, b3 \rangle) = g(\langle o103, b3 \rangle) + h(b3) = 4 + 17 = 21$
3.  $[b1_{21}, b4_{29}, ts_{31}, o109_{36}]$
4.  $[c2_{21}, b2_{29}, b4_{29}, ts_{31}, o109_{36}]$
5.  $[c1_{21}, b2_{29}, b4_{29}, c3_{29}, ts_{31}, o109_{36}]$
6.  $[b2_{29}, b4_{29}, c3_{29}, ts_{31}, c3_{35}, o109_{36}]$
7.  $[b4_{29}, ts_{31}, c3_{35}, o109_{36}]$
8.  $[ts_{31}, c3_{35}, b4_{35}, o109_{36}, o109_{42}]$
- .....

- Lowest-cost path is eventually found.
- Forced to try many different paths, because some temporarily seem to have the lowest cost.
- Still does better than lowest-cost-first search and greedy best-first search.

# Optimality of A\*

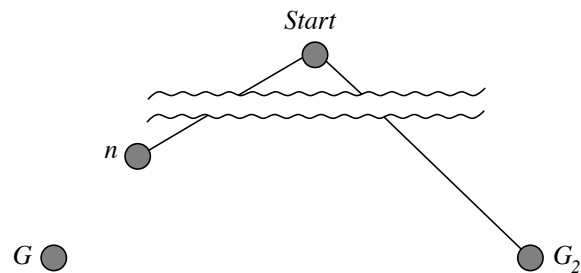
- Heuristic  $h$  is said to be **admissible** if

$\forall n \ h(n) \leq h^*(n)$  where  $h^*(n)$  is the true cost from  $n$  to goal

- If  $h$  is **admissible** then  $f(n)$  never overestimates the actual cost of the best solution through  $n$ .
  - $f(n) = g(n) + h(n)$ , where  $g(n)$  is the actual cost to  $n$  and  $h(n)$  is an underestimate
- Example:  $h = \text{straight line distance}$  is admissible because the shortest path between any two points is a line.
- A\* is optimal if  $h$  is admissible.
- Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is.

# Optimality of A\* Search

Suppose a sub-optimal goal node  $G_2$  has been generated and is in the queue. Let  $n$  be the last unexpanded node on a shortest path to an optimal goal node  $G$ .

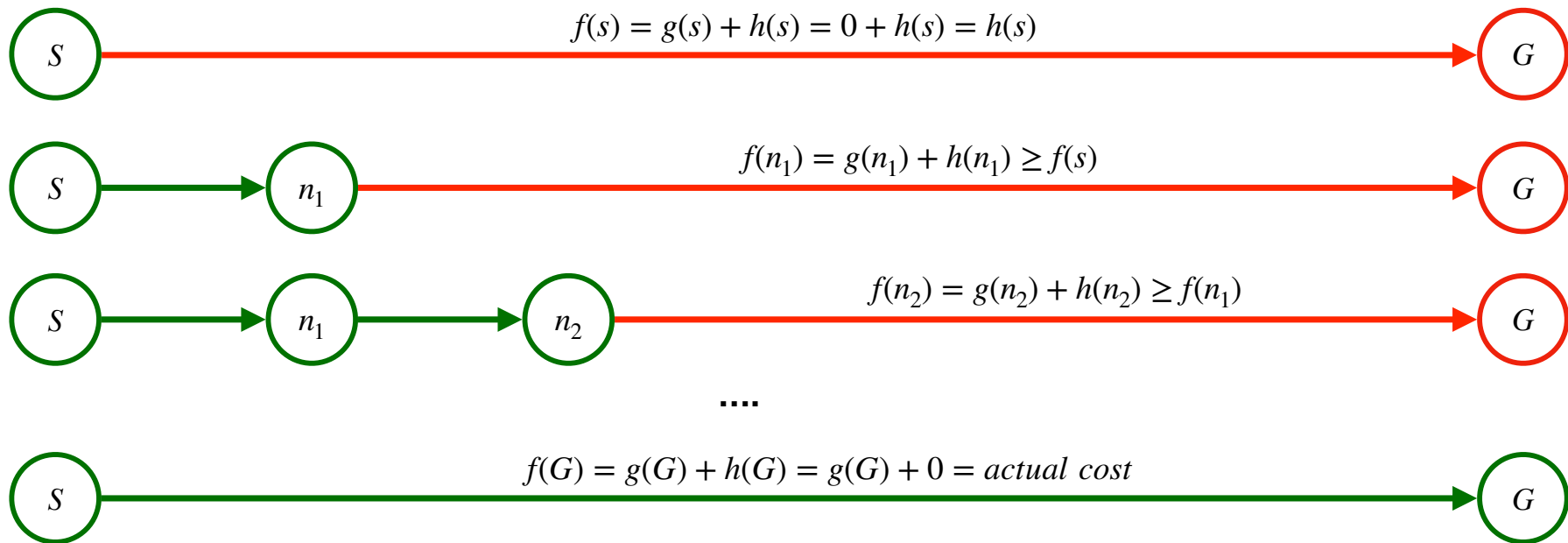


$$\begin{aligned} f(G_2) &= g(G_2) \\ &= g(G) \\ &\geq f(n) \end{aligned}$$

since  $h(G_2) = 0$   
since  $G_2$  is sub-optimal  
since  $h$  is admissible

Hence  $f(G_2) > f(n)$ , and A\* will never select  $G_2$  for expansion because queue is always ordered, e.g.  $[\dots, n, \dots, G_2]$ .

# Consistent Heuristics



If  $h(n)$  is an underestimate of the actual cost,  $h(n)$ , and  $f(n)$ , can never decrease as search heads towards the goal

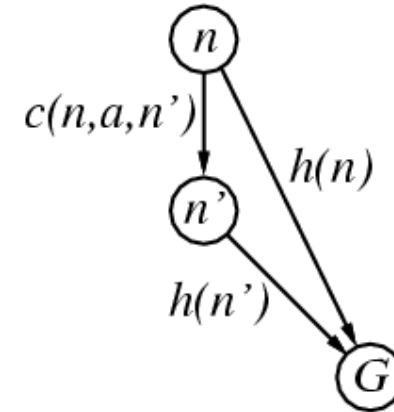
# Consistent Heuristics

- A heuristic is consistent if  $f(n)$  is nondecreasing along any path
- I.e. for every node  $n$ , every successor  $n'$  of  $n$  generated by any action  $a$ ,

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

If  $h$  is consistent, we have

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



# Optimality of A\* Search

**Complete:** Yes, unless infinitely many nodes with  $f \leq$  cost of solution

**Time:** Exponential in *relative error in  $h \times$  length of solution*

**Space:** Keeps all expanded nodes in memory

**Optimal:** Yes (assuming  $h$  is admissible).

# Iterative Deepening A\* Search

- Iterative Deepening A\* is a low-memory variant of A\* that performs a series of depth-first searches but cuts off each search when the  $f$  exceeds current threshold, initially  $f(start)$ .
- The threshold is increased with each successive search.



# Examples of Admissible Heuristics

$h_1(n)$  = total number of misplaced tiles

$h_2(n)$  = total **Manhattan distance** =  $\sum$  distance from goal position

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

$$h_1(Start) = 6$$

$$h_2(Start) = 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14$$

# How to Find Heuristic Functions ?

- Admissible heuristics can often be derived from the exact solution cost of a simplified or “relaxed” version of the problem. (i.e. with some of the constraints weakened or removed)
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then  $h_1(n)$  gives the shortest solution.
- If the rules are relaxed so that a tile can move to any adjacent square, then  $h_2(n)$  gives the shortest solution.

# Dominance

- if  $h_2(n) \geq h_1(n)$  for all  $n$  (both admissible)
  - $h_2$  **dominates**  $h_1$  and is better for search.
  - Try make the heuristic  $h$  as large as possible, without exceeding  $h^*$ .

- typical search costs:

14-puzzle	IDS	= 3,473,941 nodes
	$A^*(h_1)$	= 539 nodes
	$A^*(h_2)$	= 113 nodes
24-puzzle	IDS	$\approx 54 \times 10^9$ nodes
	$A^*(h_1)$	= 39,135 nodes
	$A^*(h_2)$	= 1,641 nodes

# Summary of Informed Search

- Heuristics can be applied to reduce search cost.
- Greedy Search tries to minimise cost from current node  $n$  to the goal.
- $A^*$  combines the advantages of Uniform-Cost Search and Greedy Search
- $A^*$  is complete, optimal and optimally efficient among all optimal search algorithms.
- Memory usage is still a concern for  $A^*$ . IDA\* is a low-memory variant.

# Summary

- Informed search makes use of problem-specific knowledge to guide progress of search
- This can lead to a significant improvement in performance
- Much research has gone into admissible heuristics
  - Even on the automatic generation of admissible heuristics