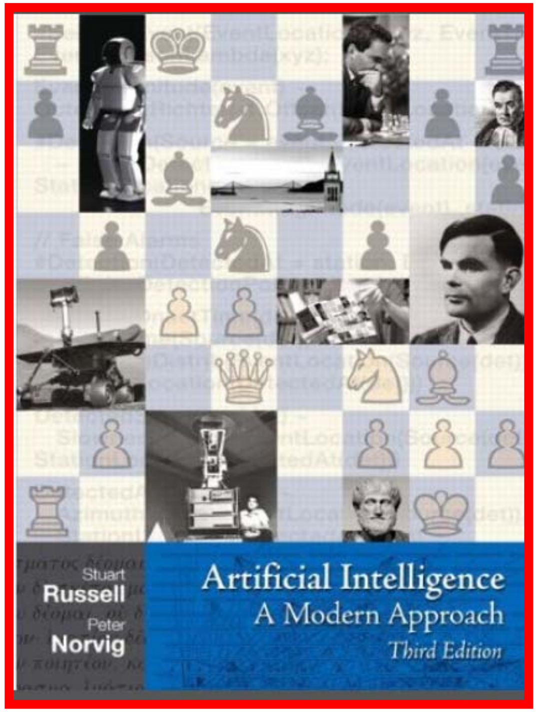


(Chapter-3)

Problem Solving and Search (cont..)

Yanmei Zheng

Searching algorithm

Uninformed Search Algorithms(Blind Search)

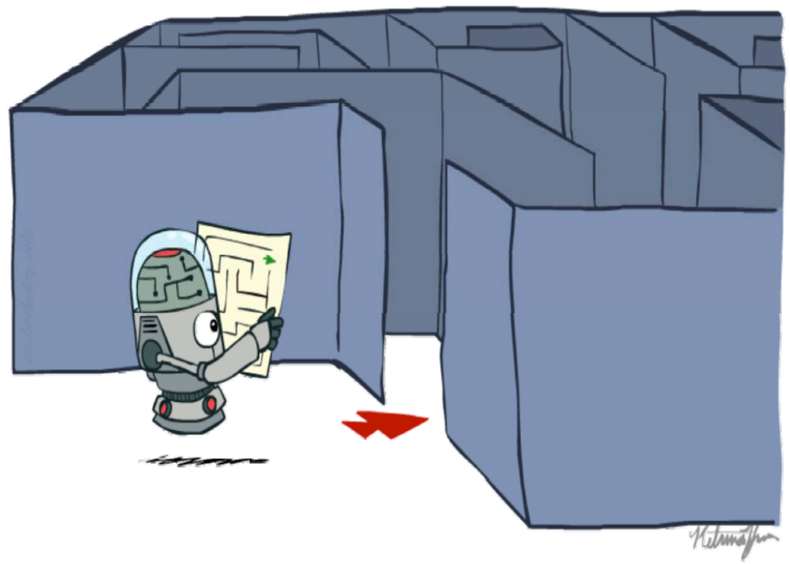
3.1 Breadth first Search
3.2 Depth First Search
3.3 Depth limited Search
3.4 Iterative Deeping Search
3. 5 Bidirectional Search

Informed Search (Heuristic Search)

Best First Search
Greedy Search
A* Search

Recap: Search

- **Search problem:**
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test

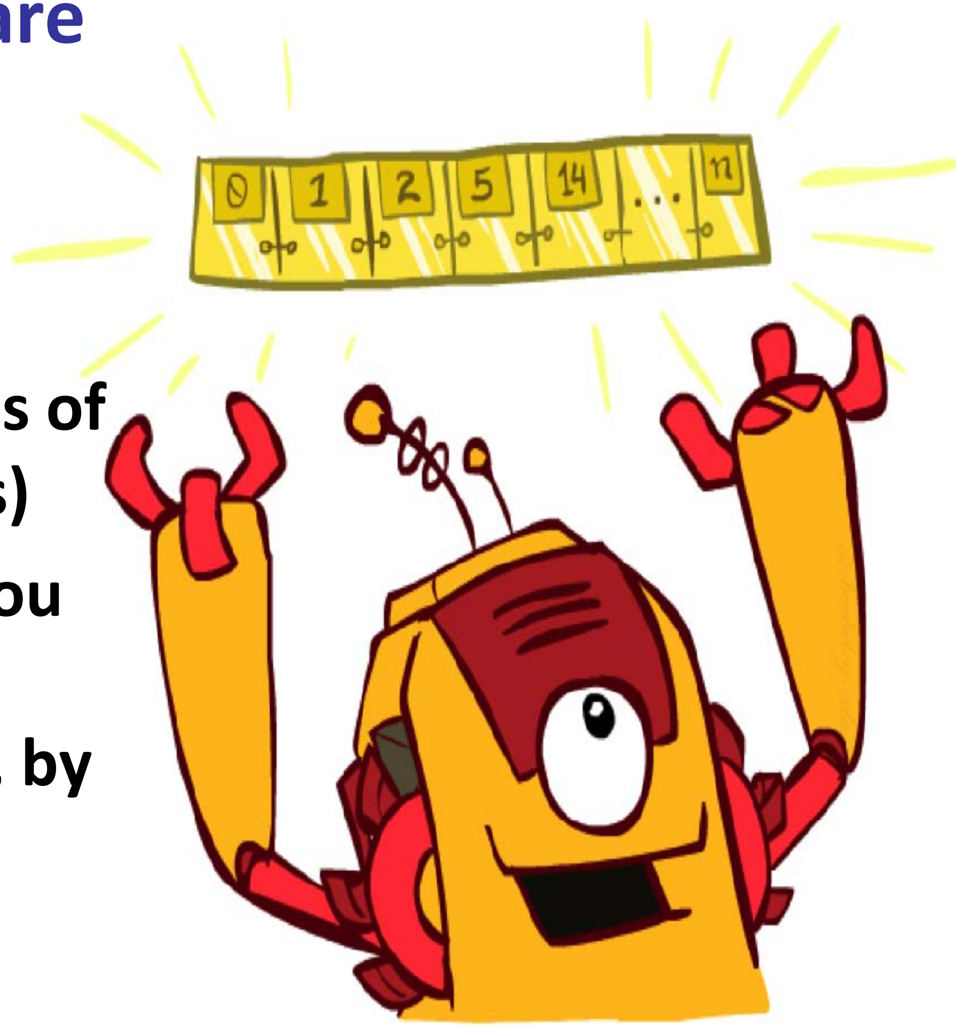


- **Search tree:**
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)

- **Search algorithm:**
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans

The One Queue

- All these search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that takes a variable queuing object



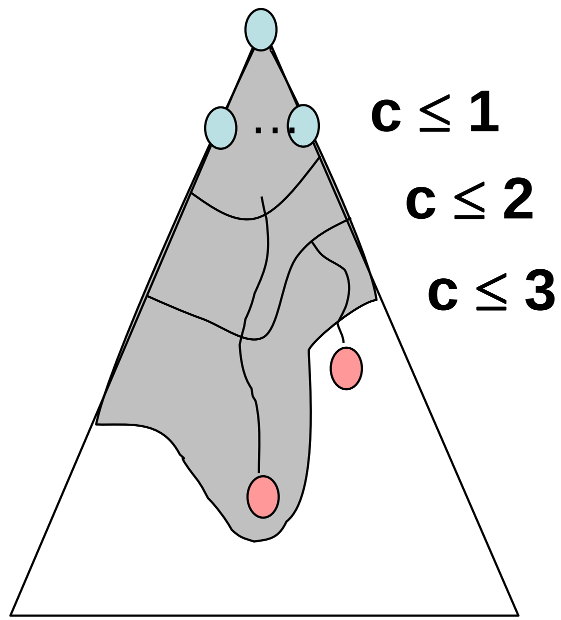
Uninformed Search

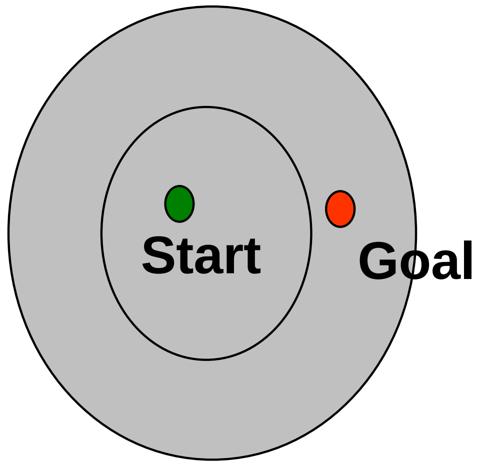
Uniform Cost Search

- Strategy: expand lowest path cost

- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every "direction"
  - No information about goal location



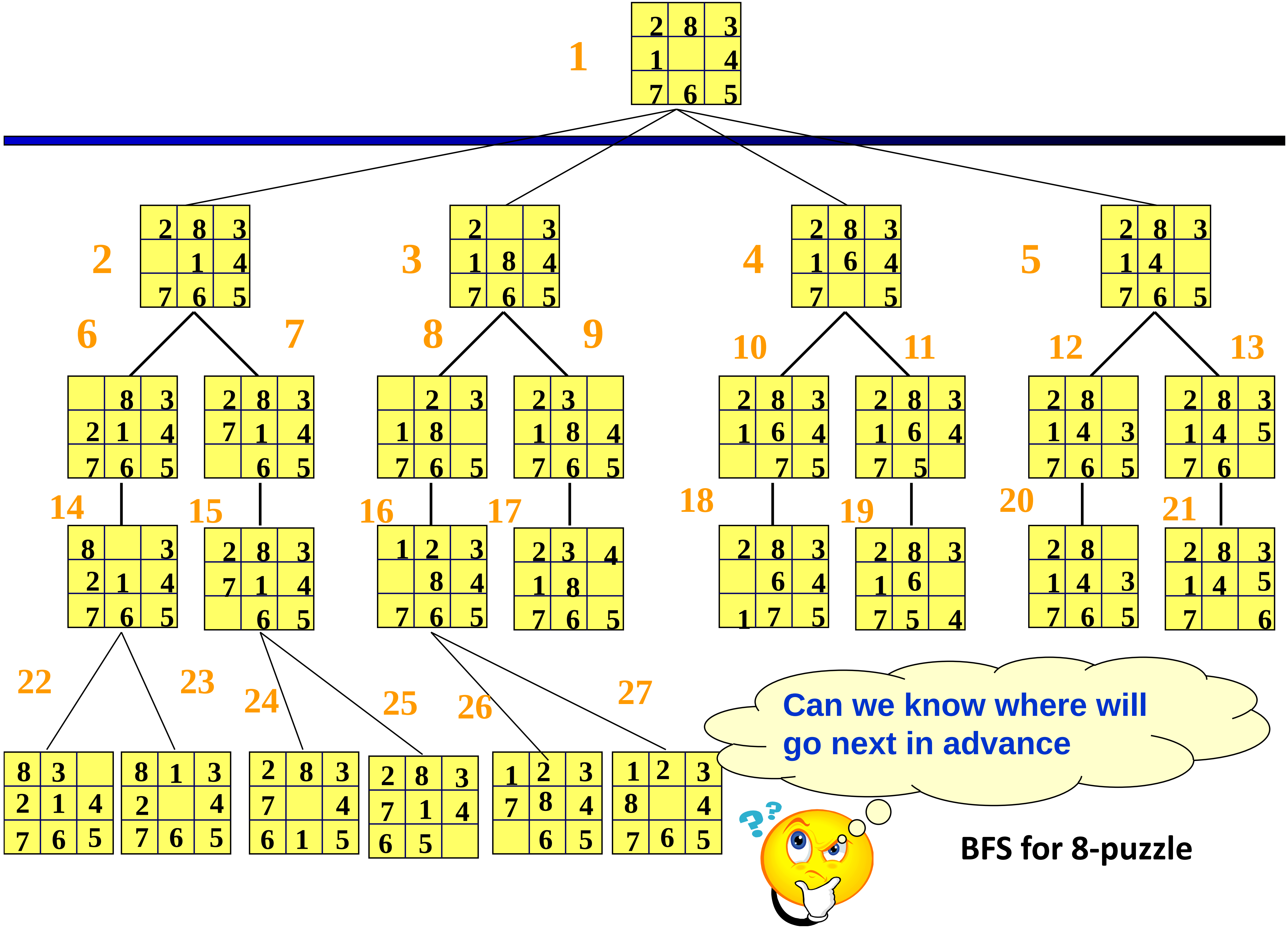


Uninformed versus Informed

Uninformed search	Informed search
➢ does not have any additional information on the **quality** of states. ➢ So, it is impossible to determine **which state is the better than others**. As a result, search efficiency depends only on the structure of a state space	➢ heuristically informed search uses a certain kind of information about states in order to guide search along **promising** branches within a state space. ➢ Using problem specific knowledge as hints to guide the search.

Uninformed versus Informed(cont)

Uninformed search	Informed search
➢ look for solutions by **systematically** generating new states and checking each of them against the goal. 1. It is very **inefficient** in most cases. 2. Most successor states are “obviously” a bad choice. 3. Such strategies do not use problem-specific knowledge	1. They are almost always **more efficient** than uninformed strategies. 2. May reduce time and space complexities. 3. Evaluation function f(n) measures distance to the goal. 4. Order nodes in **Frontier** according to f(n) and decide which node to expand next.



BFS for 8-puzzle

What affects the efficiency of search?

- **8-puzzle**

2	8	3
1		4
7	6	5

Initial state

1	2	3
8		4
7	6	5

Target state

1. **BFS,DFS according to the specified route search, too much computing space and time**

2. **Select the most promising nodes and extend, the search efficiency will be greatly improved.**

Informed search & Exploration

- ***Modified version from blind search algorithm***

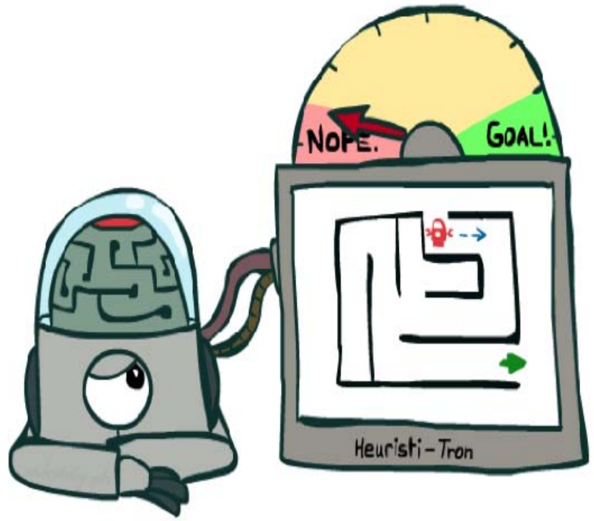
  1. ***Greedy best first search***
  2. ***A* and its relatives***

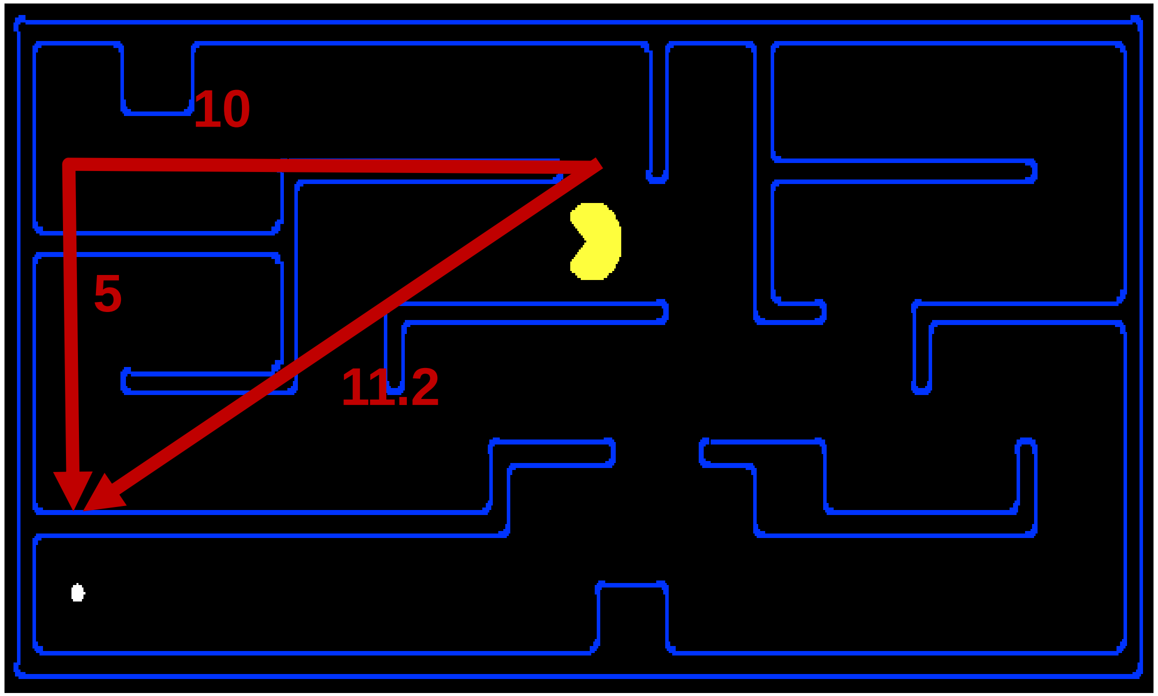
- ***The family of local search includes methods***

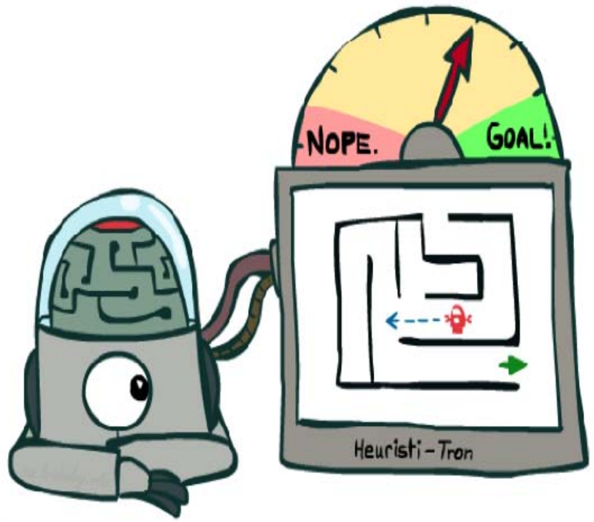
  1. **inspired by statistical physics [simulated annealing]**
  2. **evolutionary biology [genetic algorithms]**
  3. **online search [in which agent is faced with state space that is completely unknown]**

Search Heuristics

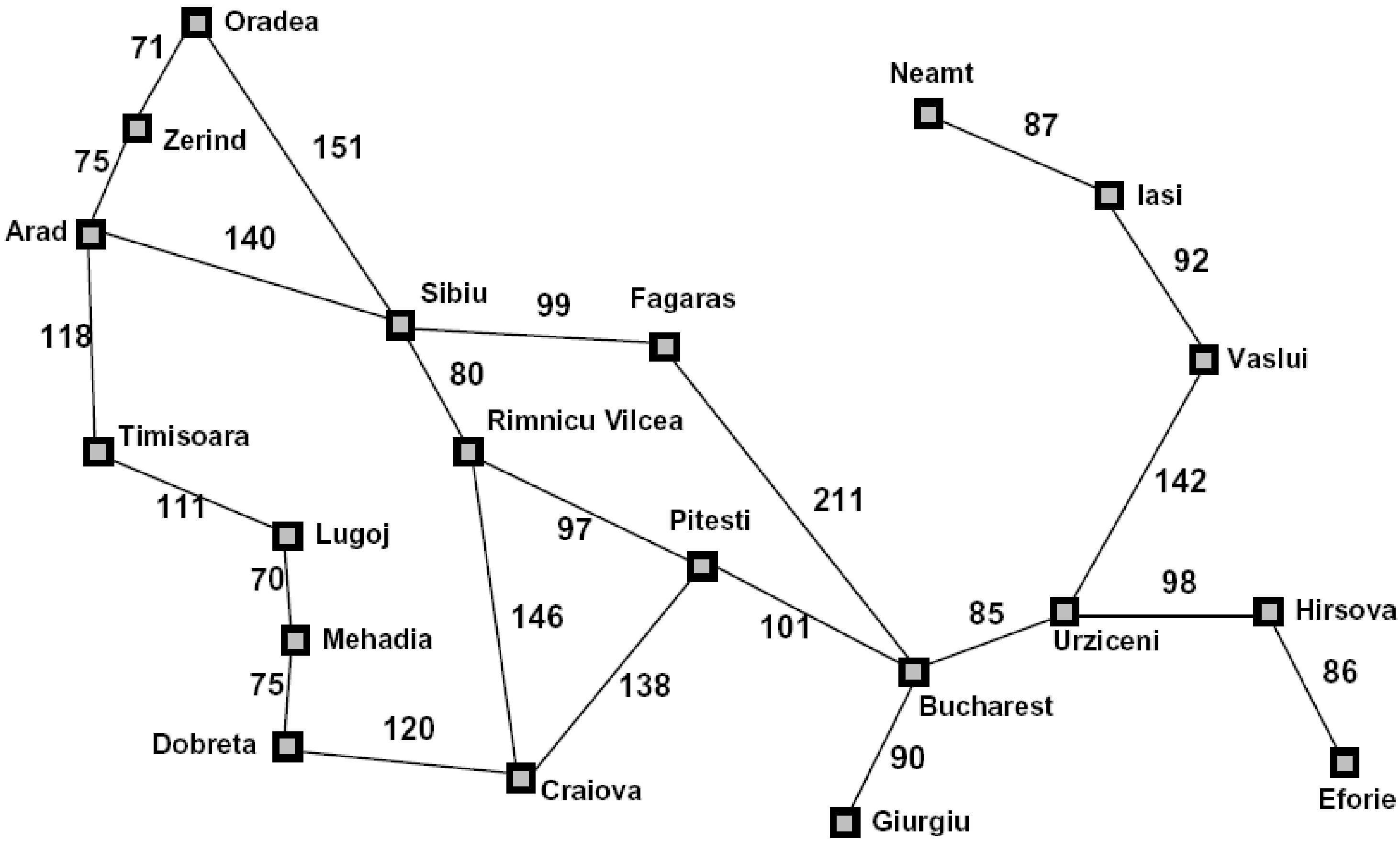
- **A heuristic is:**
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing







Example: Heuristic Function



Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Informed search & Exploration

Best first search

Main idea: use an **evaluation function f(n)** for each node

Implementation:

- Order the nodes in **Frontier** in decreasing order of desirability (from low f(n) which means high desirability to high f(n) which means low desirability.)

- There is a whole family of best-first search strategies, each with a different evaluation function.

Special cases:

- Greedy best-first search.
- A* search.

Greedy Search

Greedy best-first search==Best-first search

Best-first search Algorithm

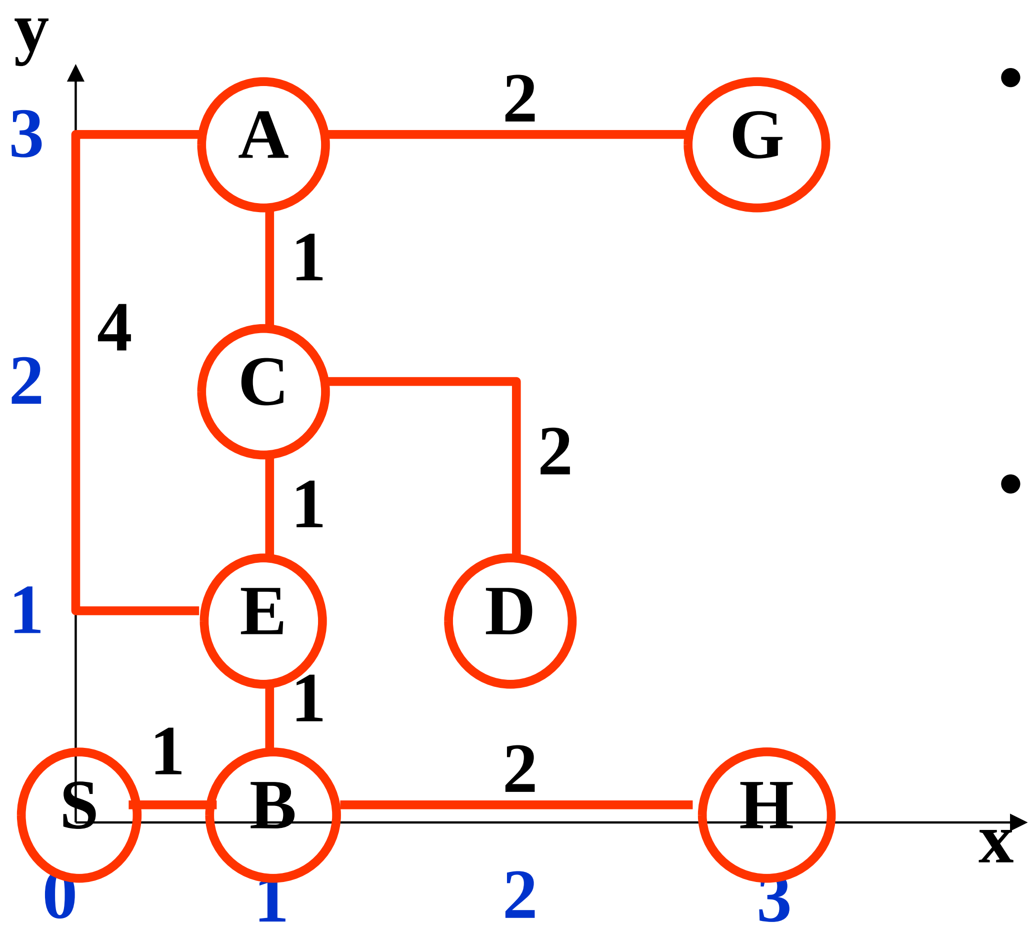
Greedy best first search

- **Tries to expand the node that is closest to the goal**
  - **Use straight line distance** **ex : $h_{SLD}(IN(Arad))=366$**

[note that the values of fn h_{SLD} cannot be computed from the problem description itself]

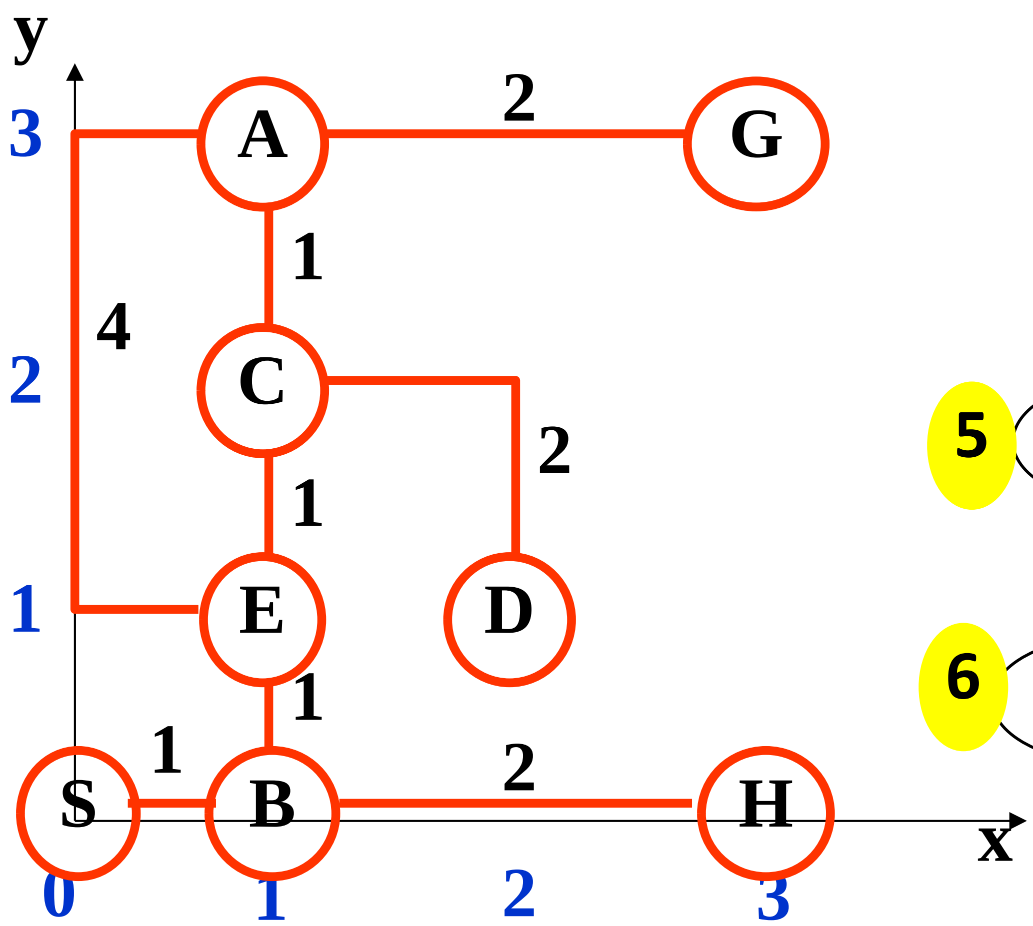
Example: the maze problem

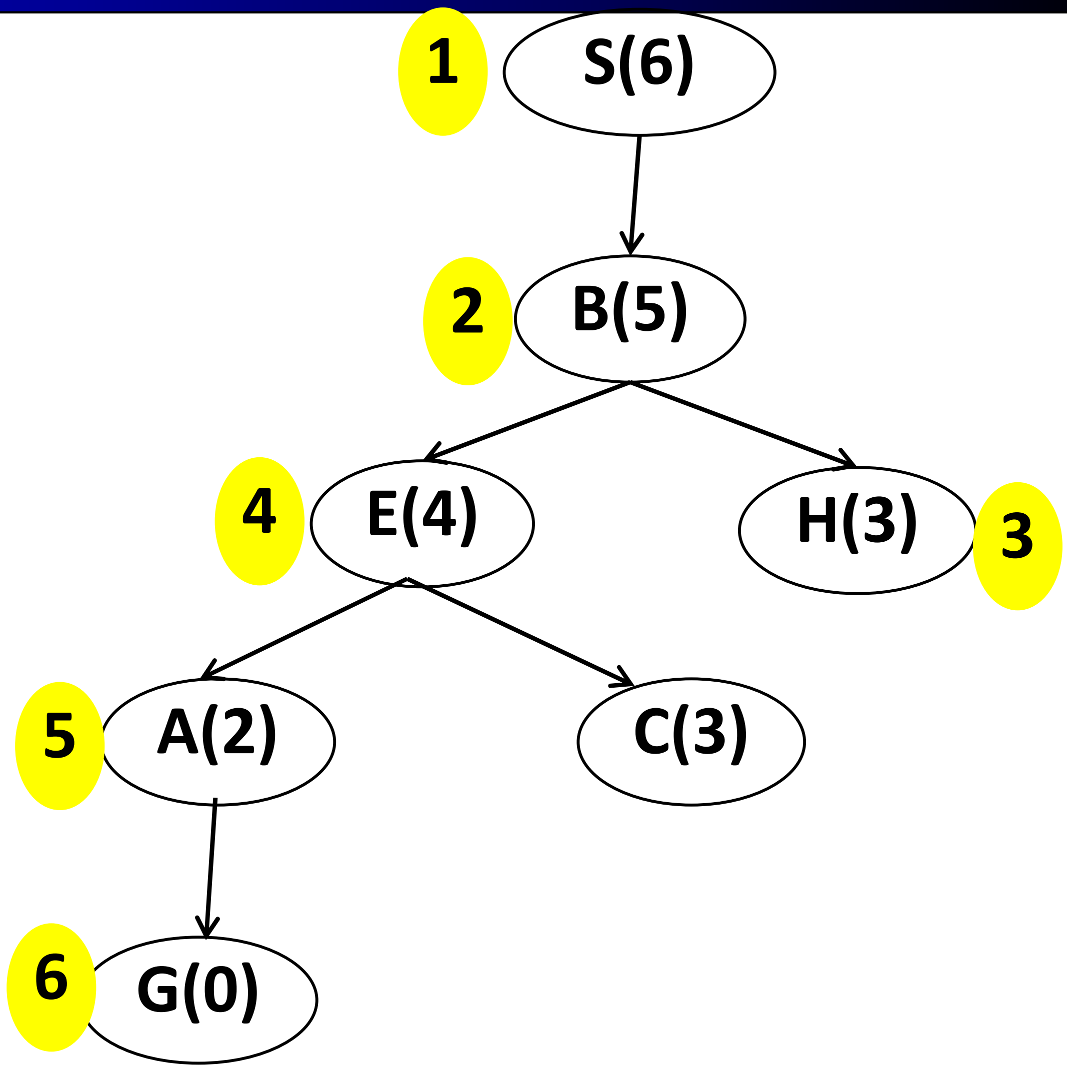
S is the entrance, G is the exit, try to use Best-first search Algorithm to solve it.

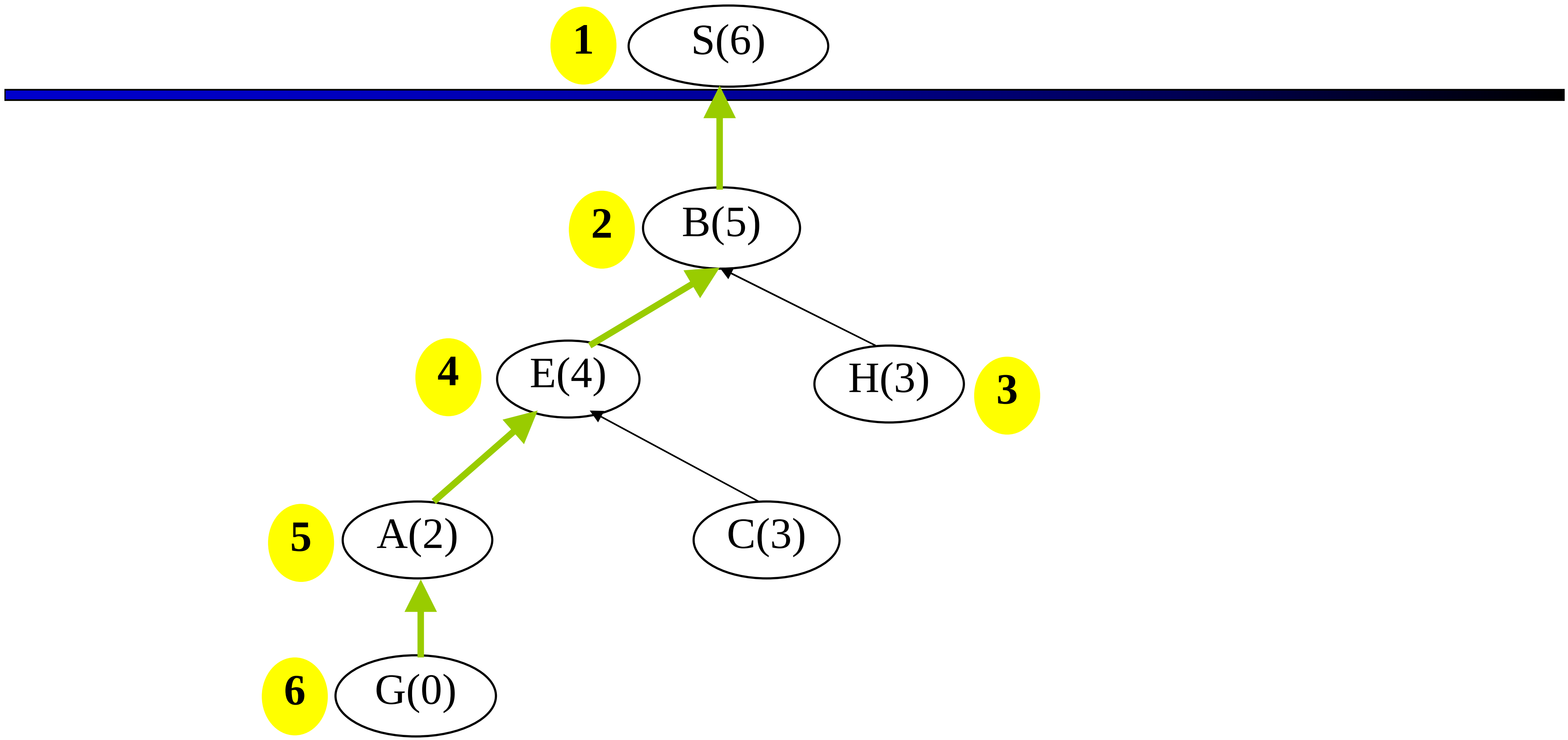


- **f(n)=The distance between each node and the target node in the coordinate system**
- **f(E)=2+2=4**

Example: the maze problem





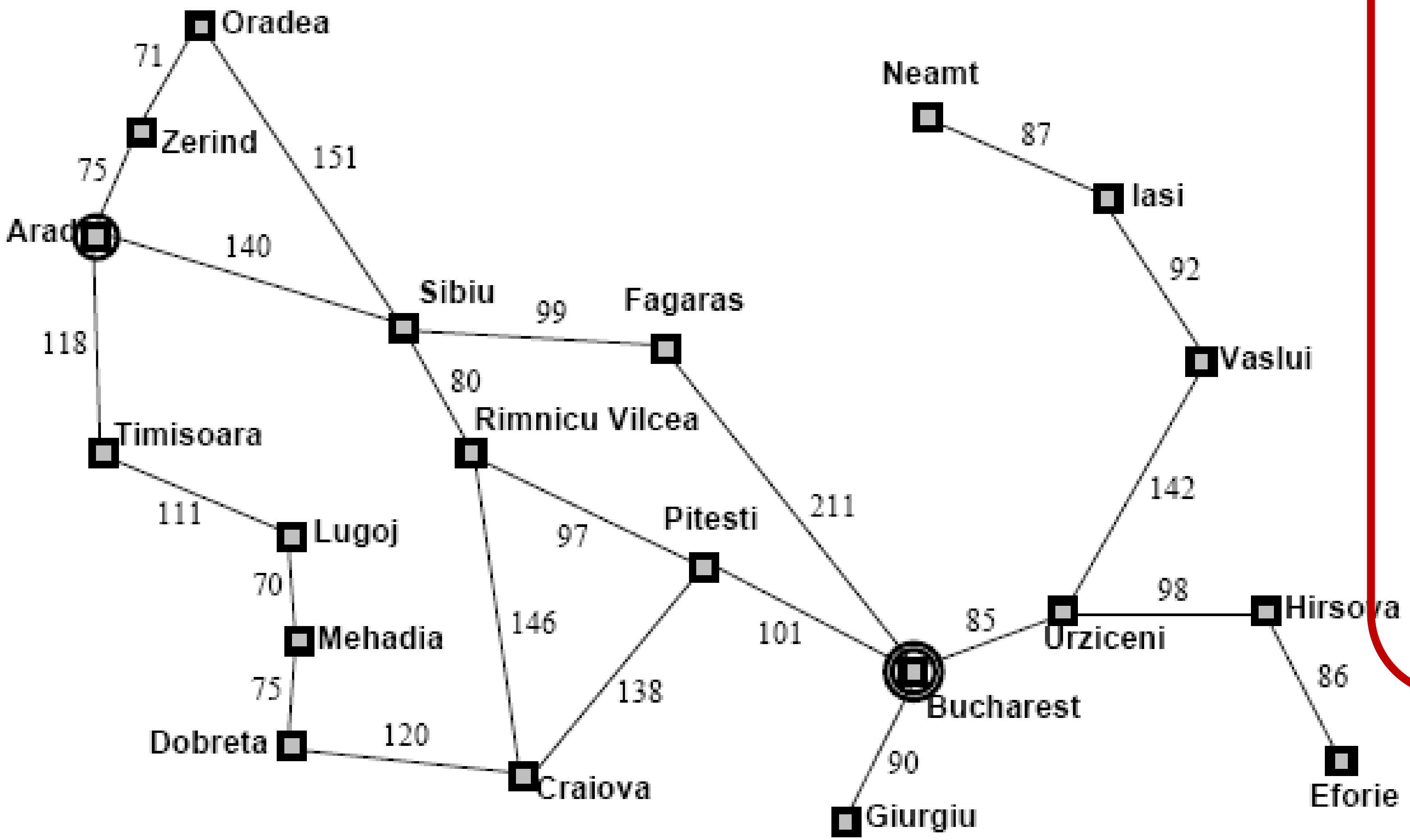


Note: the value in parentheses of each node represents the spatial distance from the node to the target, that is, the value of the evaluation function of this node.

The search results in the path shown with the yellow line.

Greedy Search Example 1

Straight line distances between cities which are additionally provided

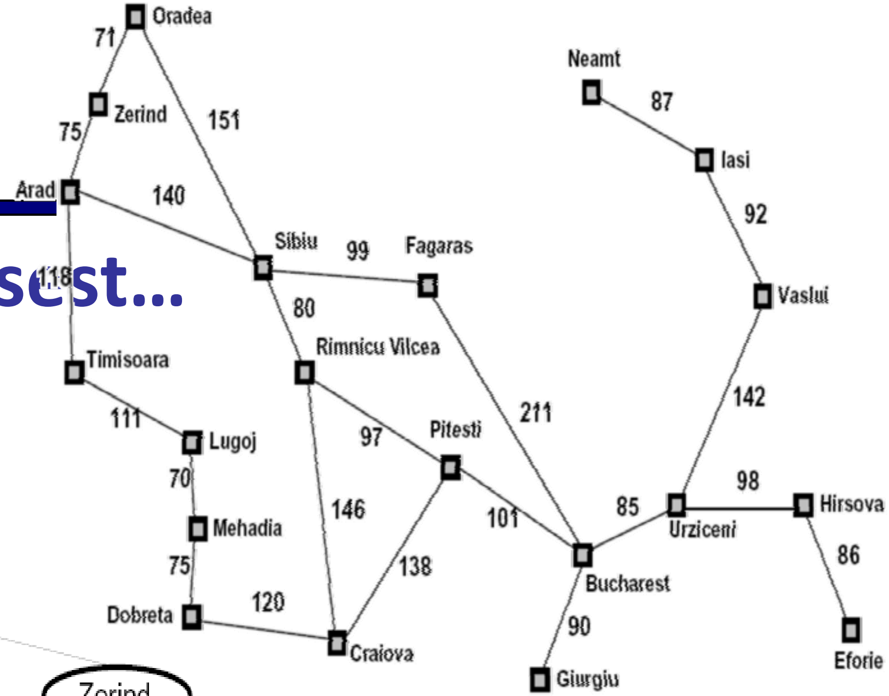


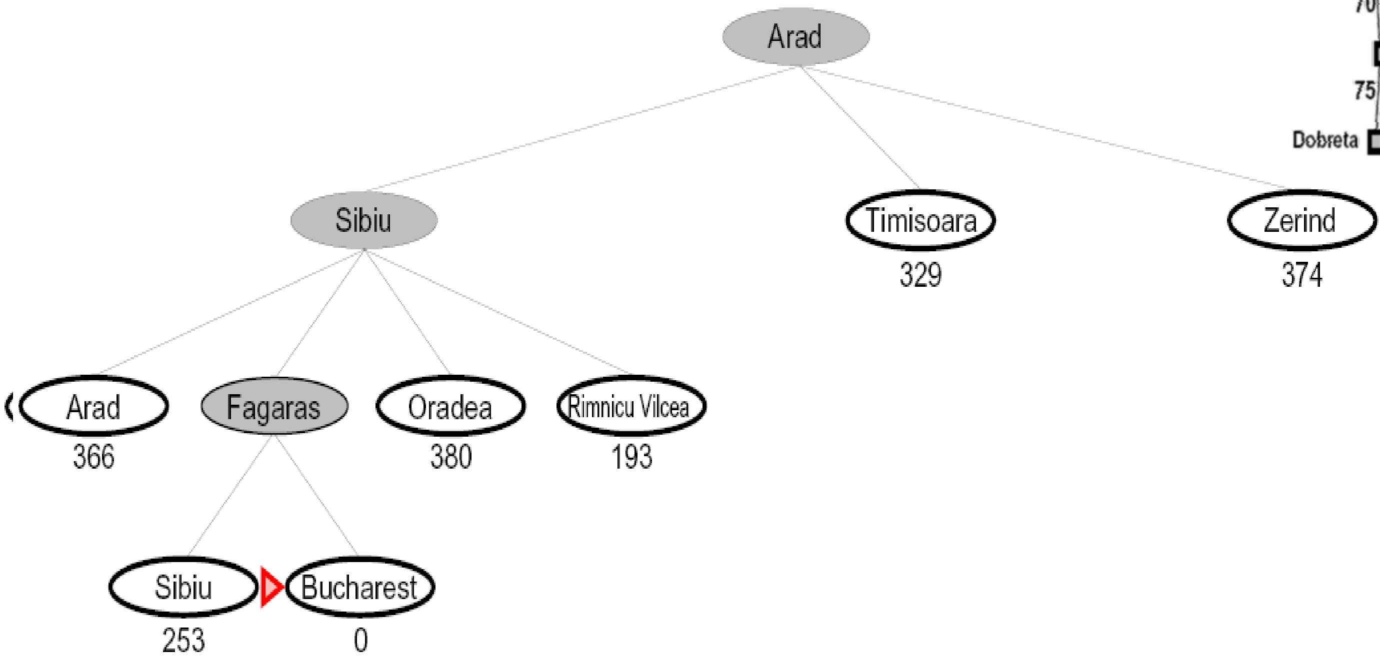
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

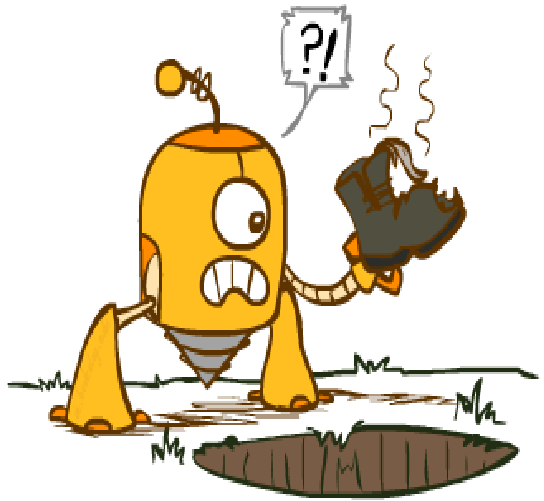
Greedy Search

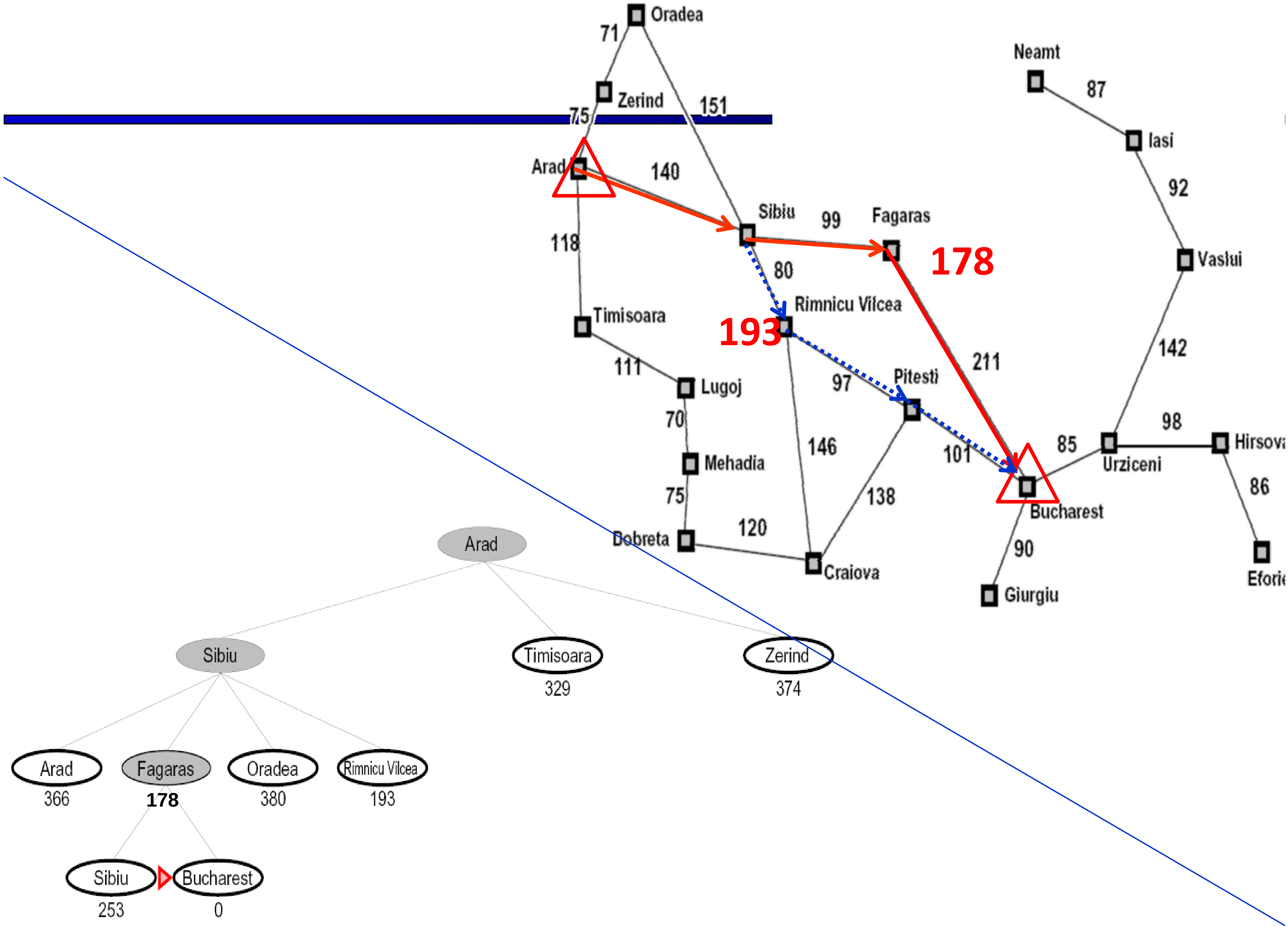
- Expand the node that seems closest…





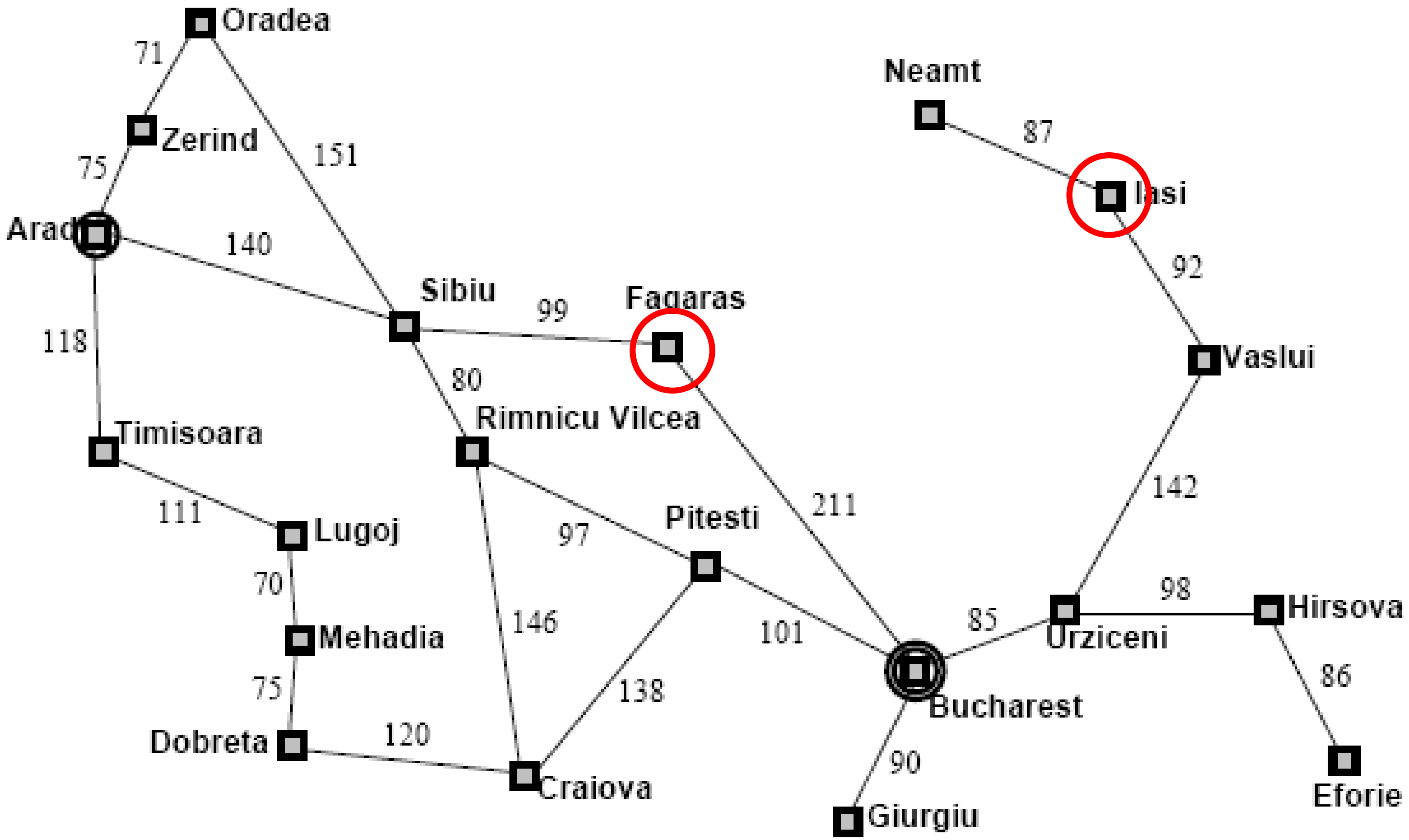
- What can go wrong?




Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
178
118
80
Rimnicu Vilcea
193
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Dobreta
120
Craiova
85
Bucharest
90
Giurgiu
Neamt
87
Iasi
92
Vaslui
142
98
Urziceni
86
Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
178
Oradea
380
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0

Greedy Search Example 2

Straight line distances between cities which are additionally provided



Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- Consider the problem of getting from Iasi to Fagras
- The heuristic suggests that Neamt be expanded first because it is closest to Fagaras but it is like dead end
- The solution is to go first to Vaslui a step that is actually farther from the goal according to the heuristic & then continue to Urzicent, Bucharest and Fagaras.
- In this case , then heuristic causes unnecessary needs to be expanded

Greedy best first search

- *Resembles depth first search in the way it prefers to follow a single path all the way to goal but it will back up when it hits a dead end*
- *It is not optimal (greedy) and incomplete (because of backtracking)*

8-puzzle problem

f(n)= number of misplaced tiles[tiles in wrong places)

f(n)=4

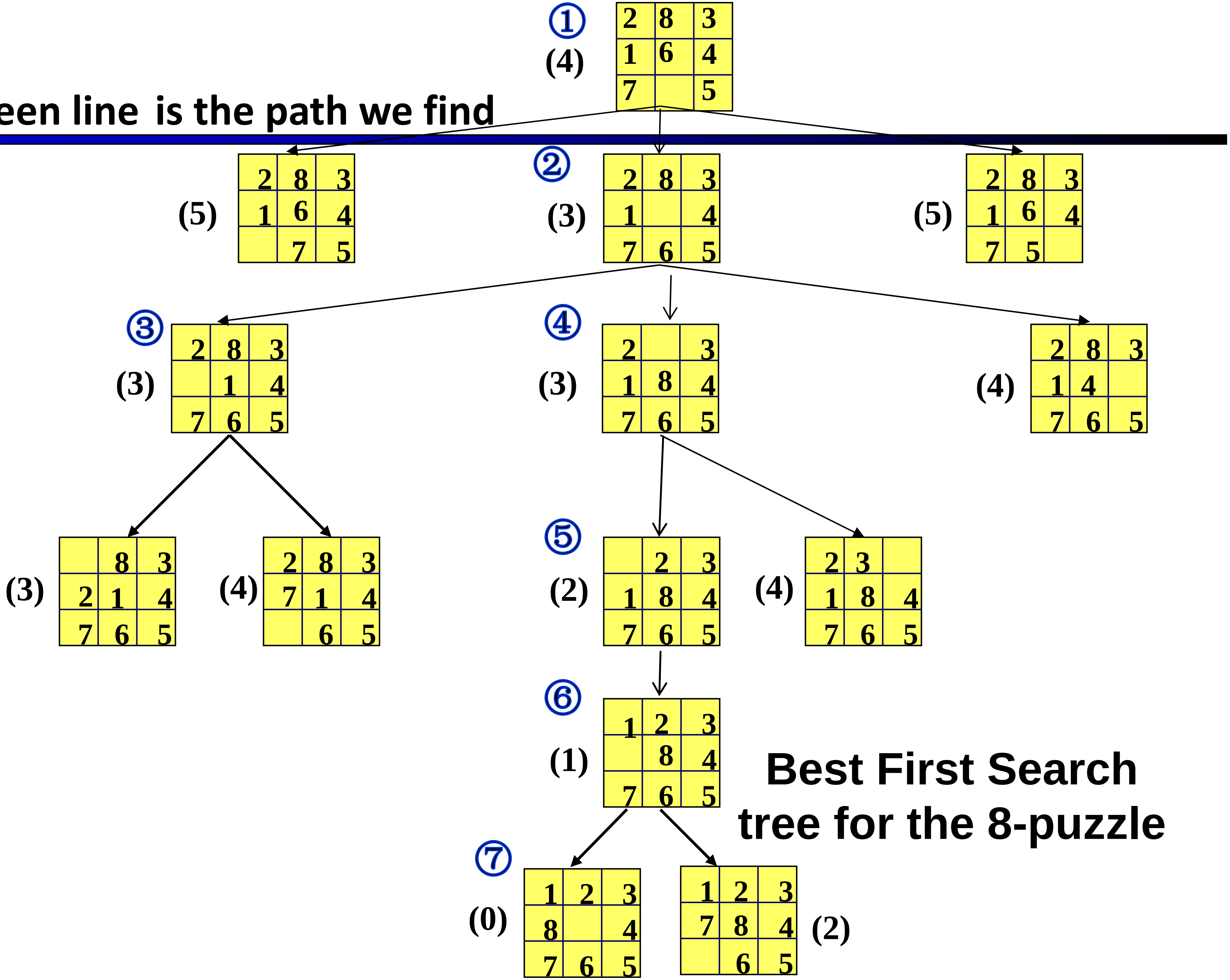
2	8	3
1	6	4
7		5

Initial state

1	2	3
8		4
7	6	5

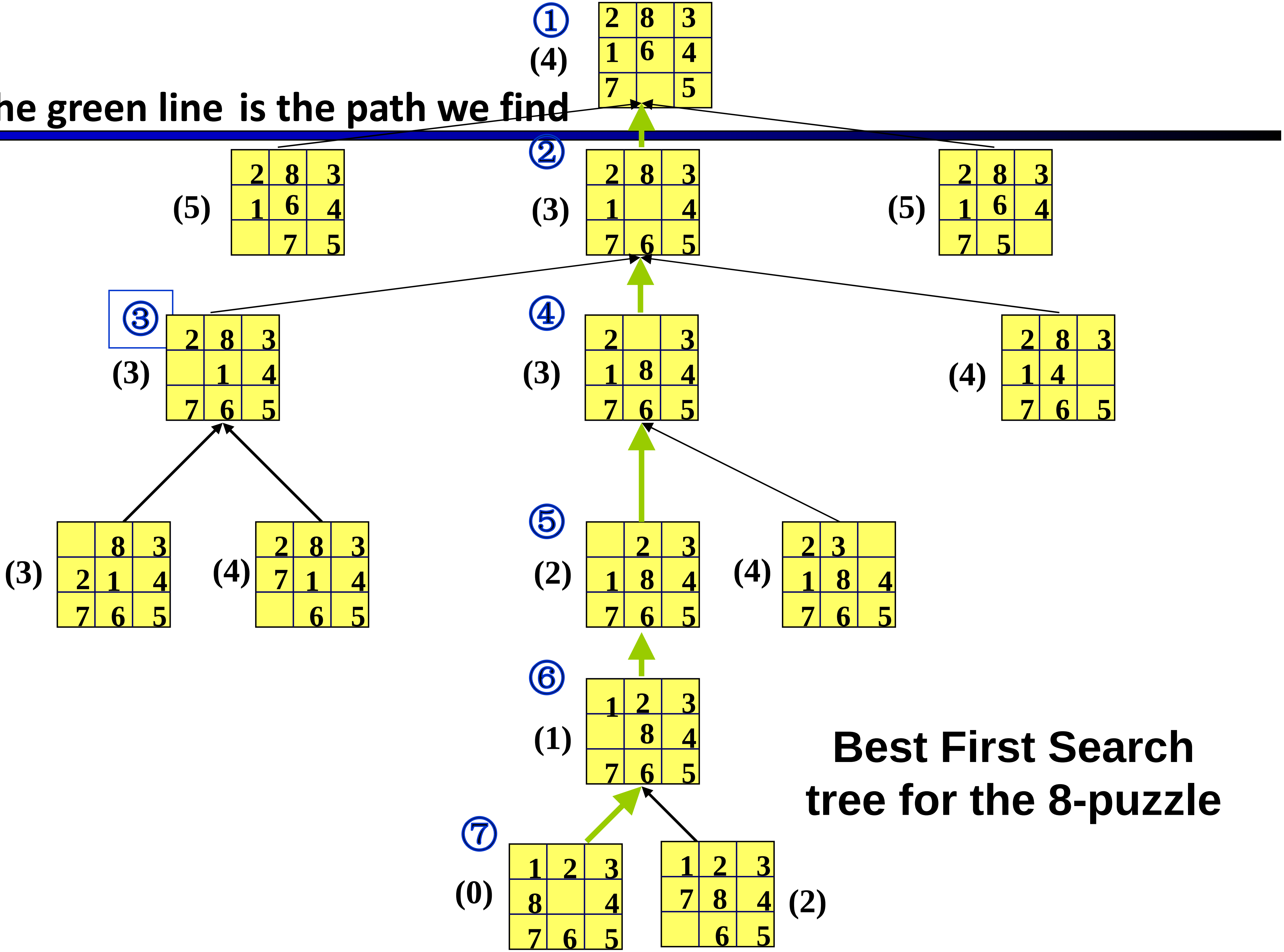
Goal state

The green line is the path we find



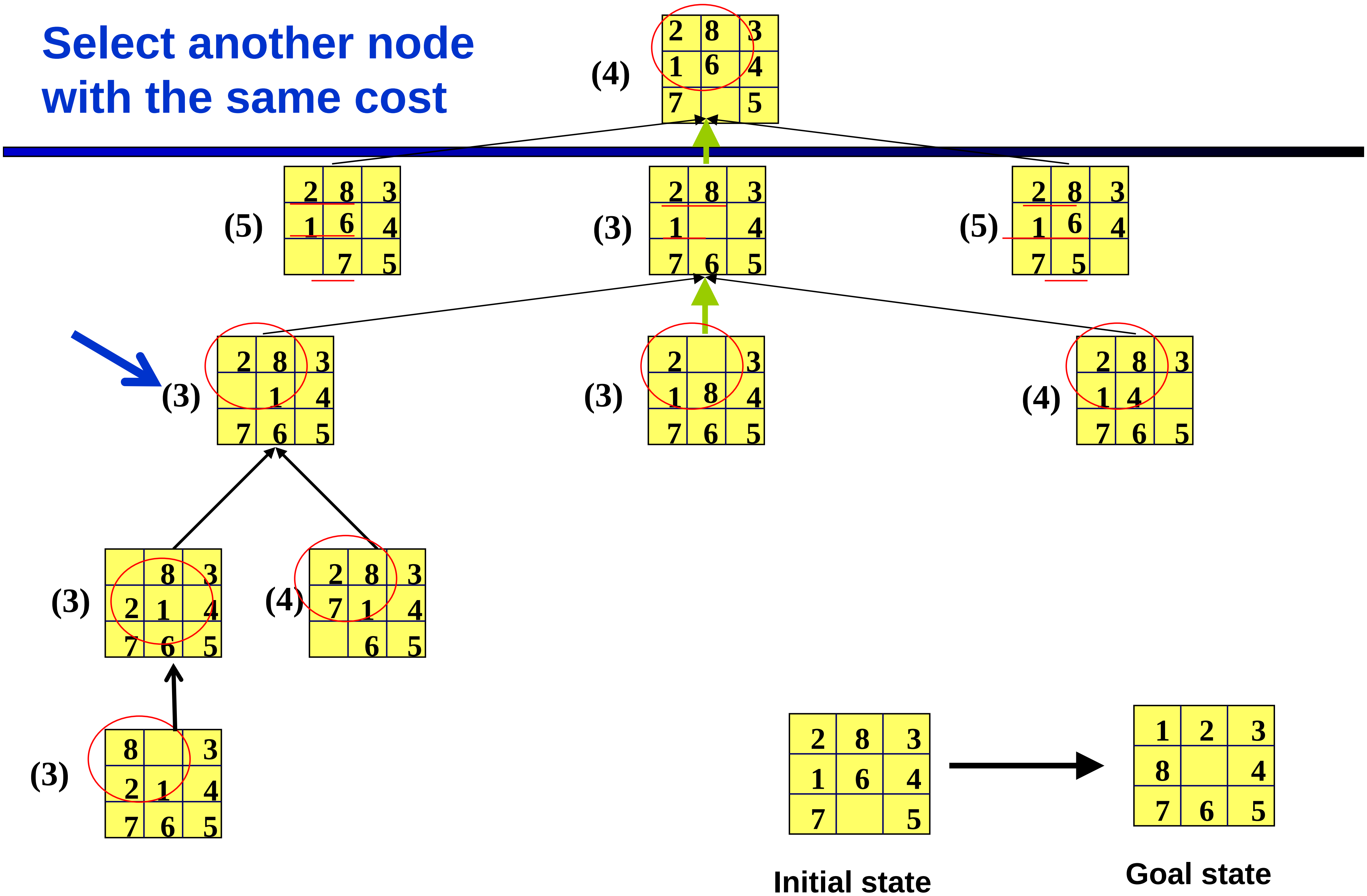
Best First Search tree for the 8-puzzle

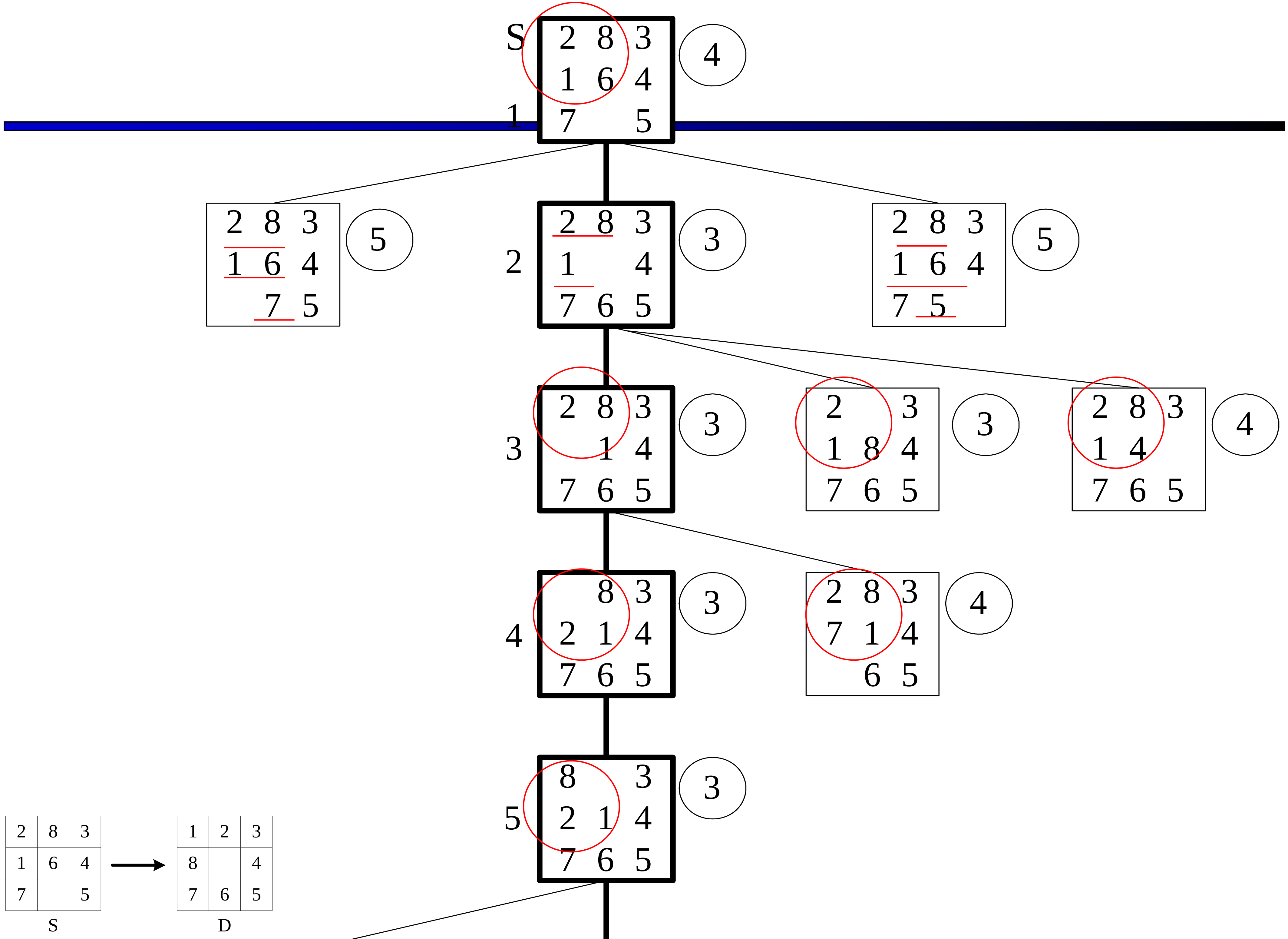
The green line is the path we find



Best First Search tree for the 8-puzzle

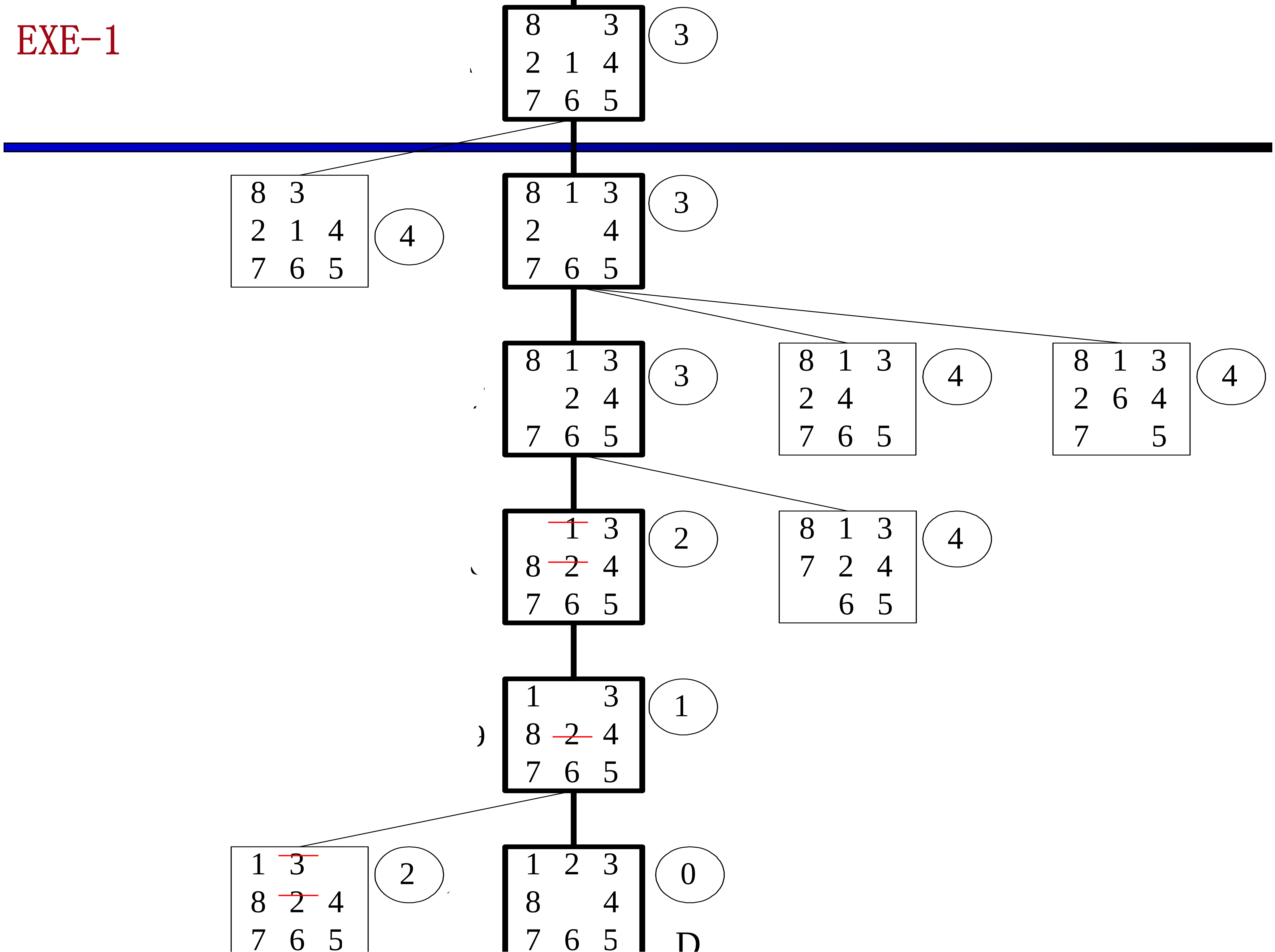
Select another node with the same cost





S
1
2
3
4
5
2 8 3
1 6 4
7 5
D
1 2 3
8 4
7 6 5

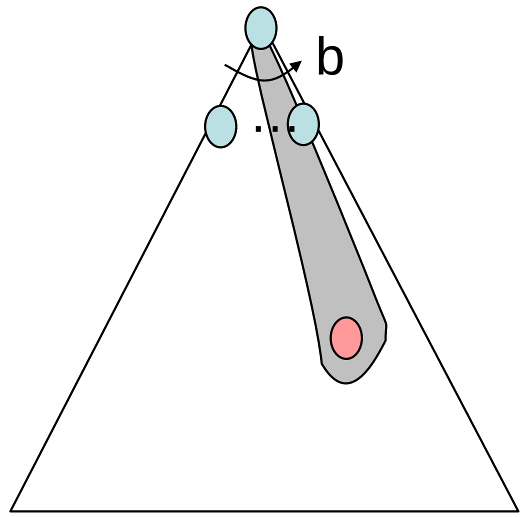
EXE-1



8 3
2 1 4
7 6 5
3
8 3
2 1 4
7 6 5
4
8 1 3
2 4
7 6 5
3
8 1 3
2 4
7 6 5
3
8 1 3
2 4
7 6 5
4
8 1 3
2 6 4
7 5
4
1 3
8 2 4
7 6 5
2
8 1 3
7 2 4
6 5
4
1 3
8 2 4
7 6 5
1
1 3
8 2 4
7 6 5
2
1 2 3
8 4
7 6 5
0

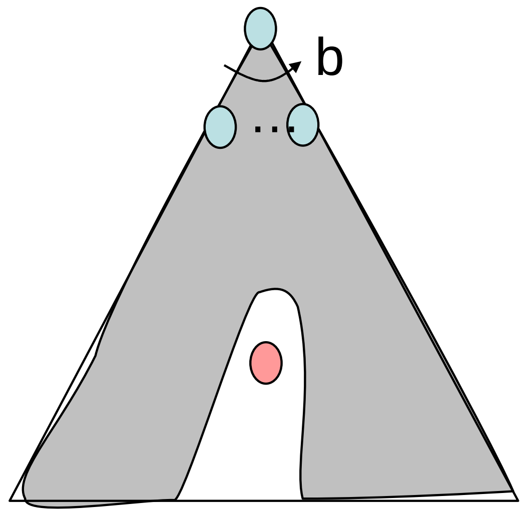
Greedy Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state

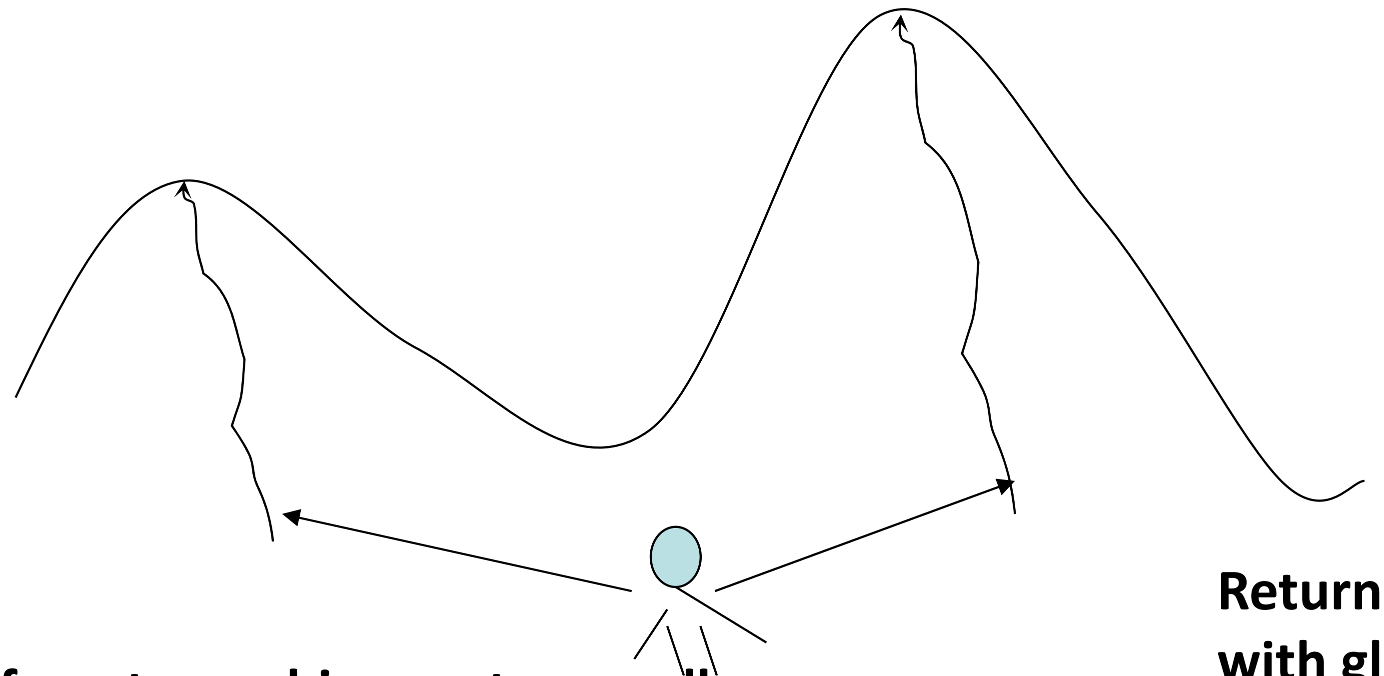


- A common case:
  - Best-first takes you straight to the (wrong) goal

- Worst-case: like a badly-guided DFS

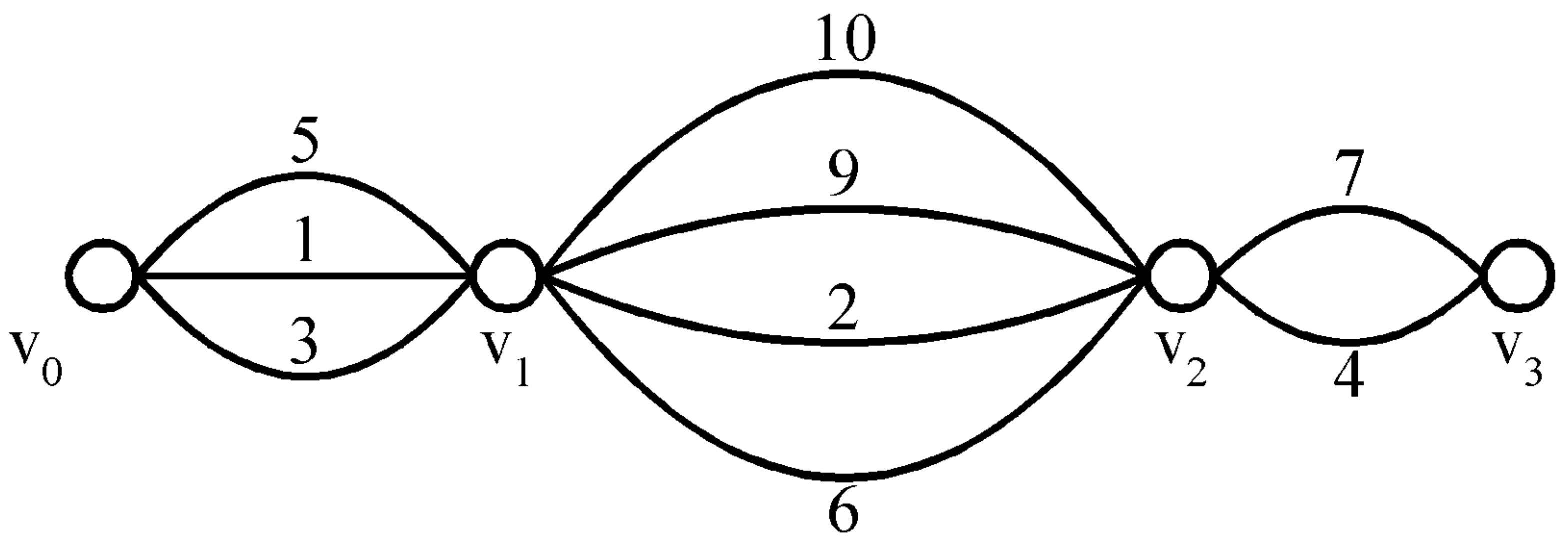


Think There's no optimal solution? Maybe make the left one as the first choice



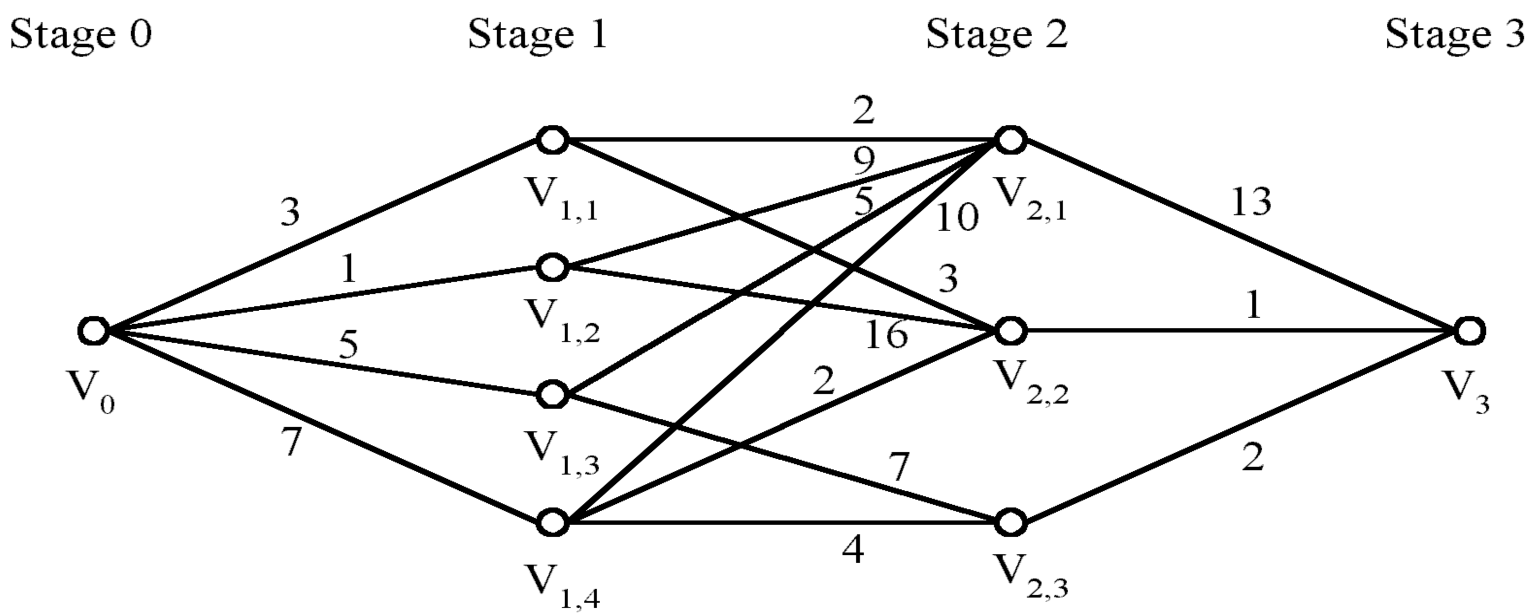
Shortest paths

- Find a shortest path from v_0 to v_3???
- Can the **greedy** method solve this problem???
- The shortest path: 1 + 2 + 4 = 7.



Shortest paths on a multi-stage graph

- Find a shortest path from v_0 to v_3 in the multi-stage graph.



- Greedy method: $v_0v_{1,2}v_{2,1}v_3 = 23$
- Optimal: $v_0v_{1,1}v_{2,2}v_3 = 7$
- The greedy method does not work.

A* Search

Algorithm of A

Evaluation function f:

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

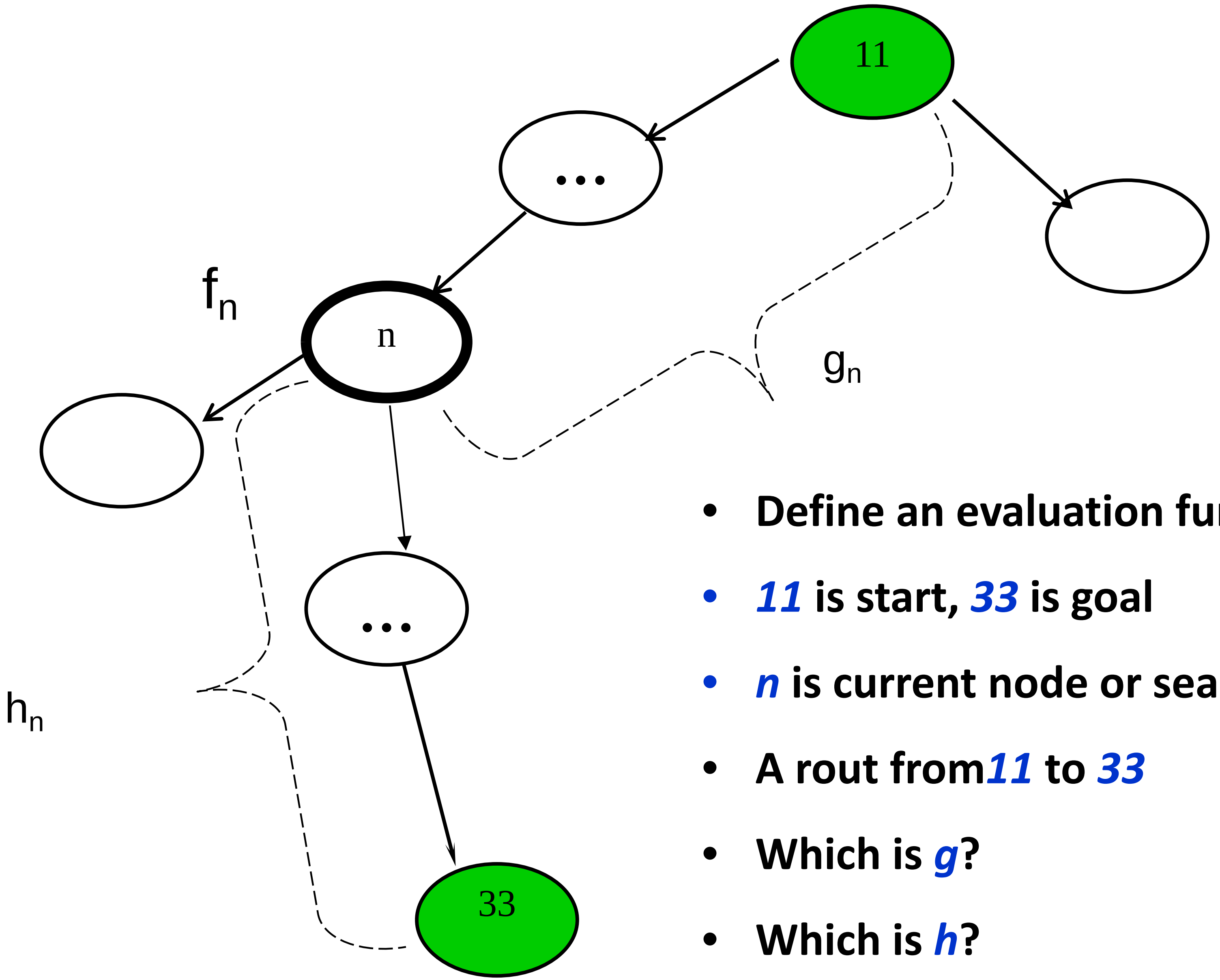
where n is the evaluated node.

g(n)： Cost paying already for searching from the initial node S to n.

h(n)： Cost evaluation in the further searching from node n to the target G.

f(n) ： Cost evaluation in searching along the rout from the initial node S to G across n.

3 part of searching graph



- **Define an evaluation function *f***
- ***11* is start, *33* is goal**
- ***n* is current node or searching *n* now**
- **A rout from*11* to *33***
- **Which is *g*?**
- **Which is *h*?**

Evaluation Function (Cost Function)

- Different f and different algorithms
  - $f(n)=g(n)$ UCS
  - $f(n)=h(n)$ Greedy best first
  - $f(n)=g(n)+h(n)$ A

A* best first search

- ***Main idea*: avoid expanding paths that are already expensive.**
- ***Minimizing the total estimated solution cost***
- ***It evaluate a node by***

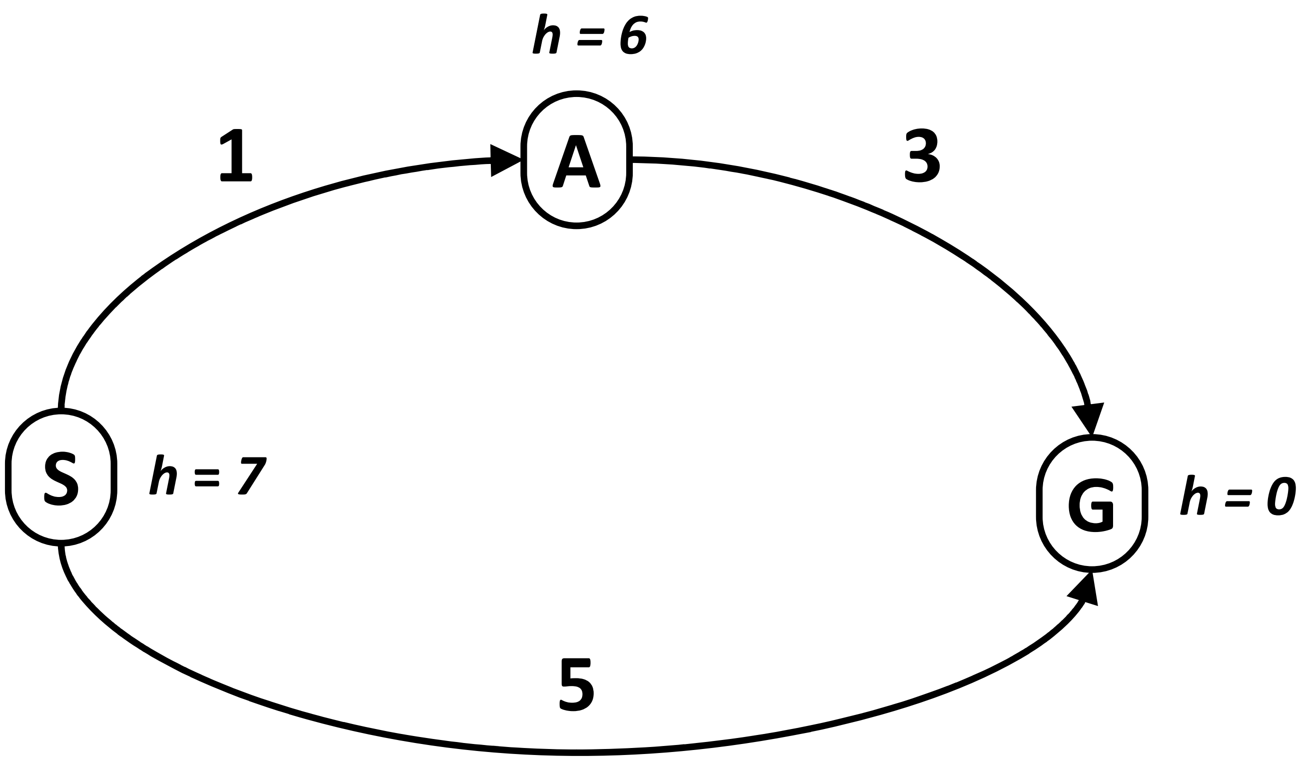
$$F(n) = g(n) + h(n)$$

Cost so far to reach n

Estimated Cost to get from n to goal

- Path cost is g and heuristic function is h
  - $f(\text{state}) = g(\text{state}) + h(\text{state})$
  - Choose smallest overall path cost (known + estimate)

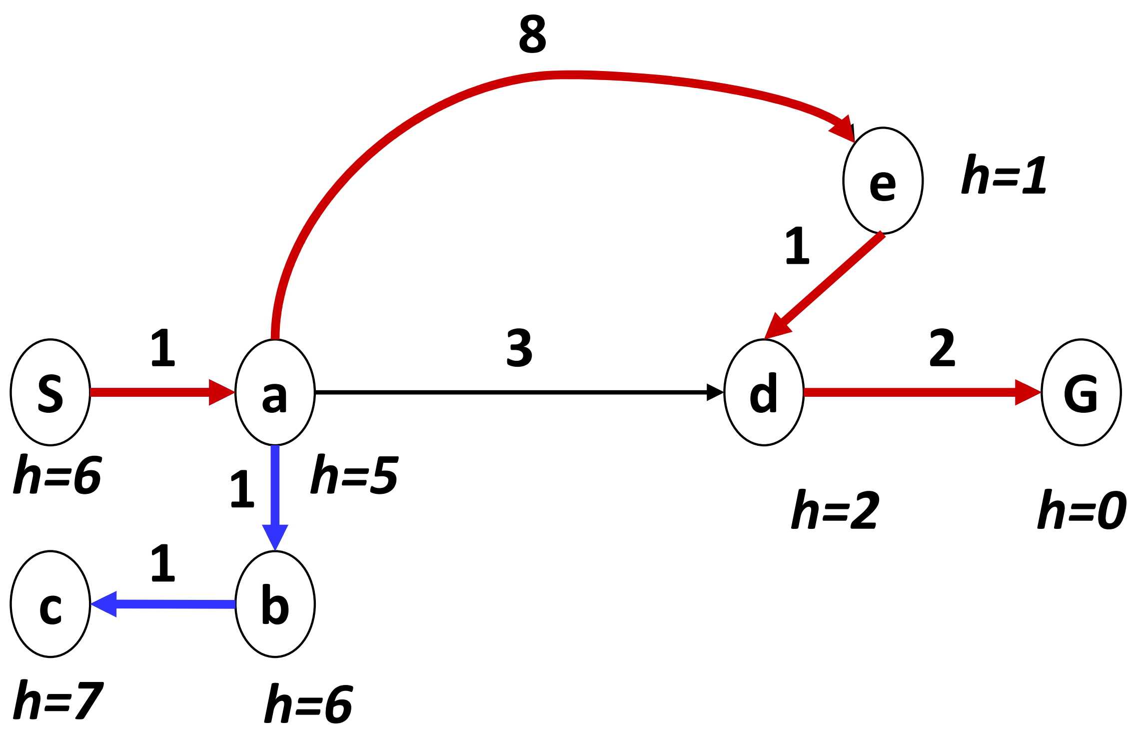
Is A* Optimal?



- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

Combining UCS and Greedy

- **Uniform-cost** orders by path cost, or *backward cost* g(n)
- **Greedy** orders by goal proximity, or *forward cost* h(n)

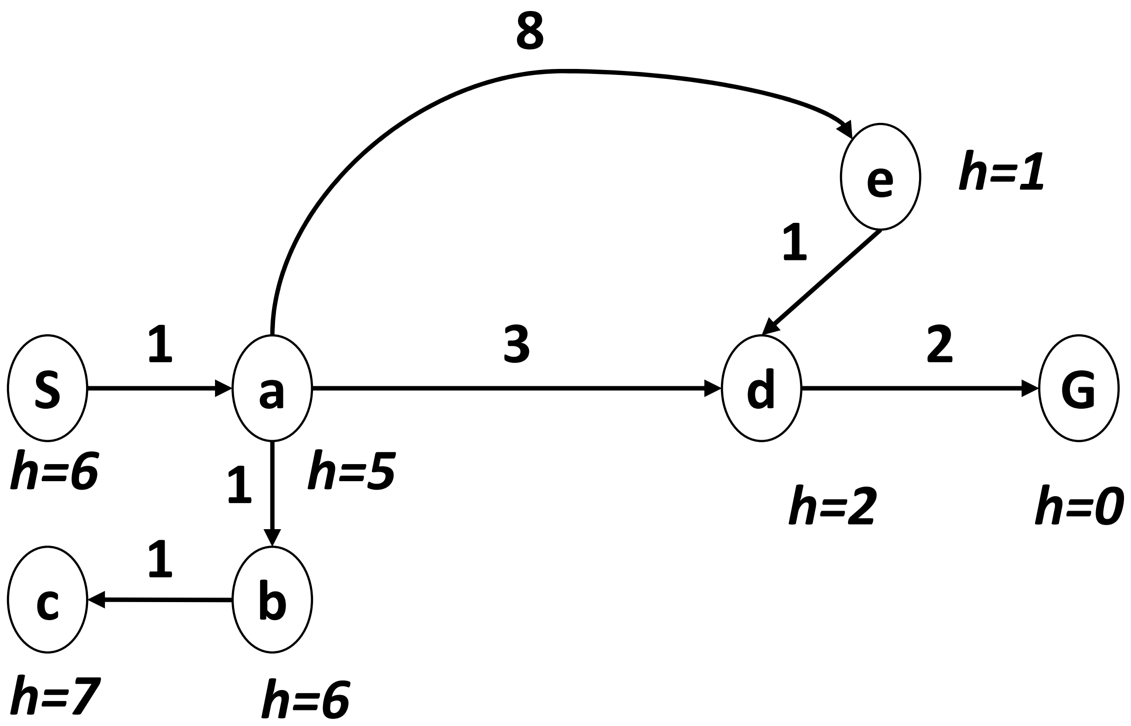


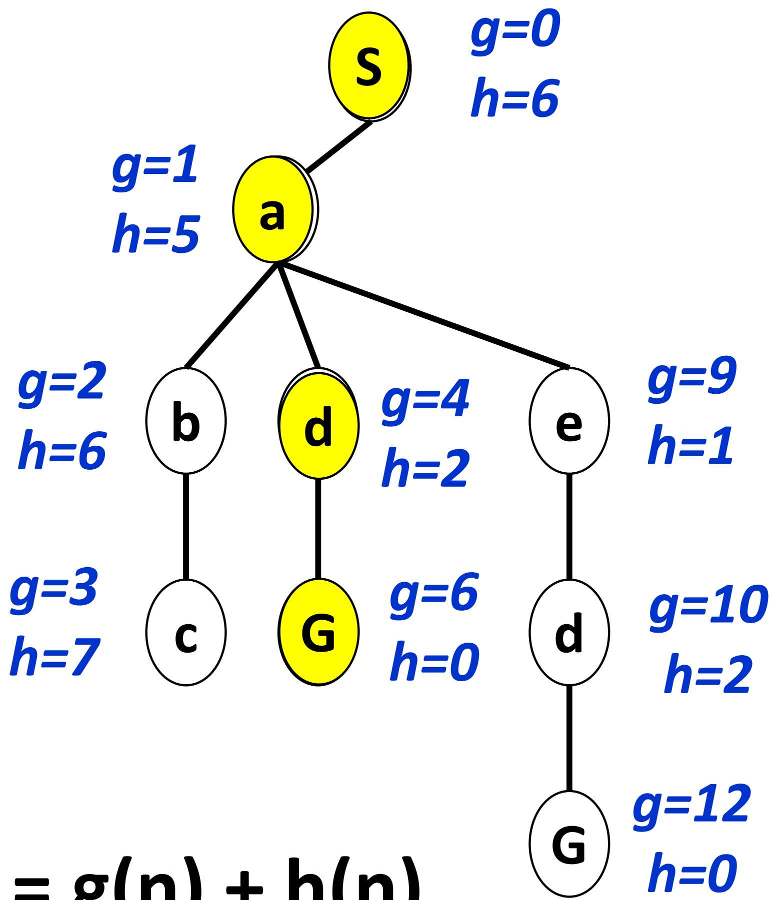
- **A* Search** orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

Combining UCS and Greedy

- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)

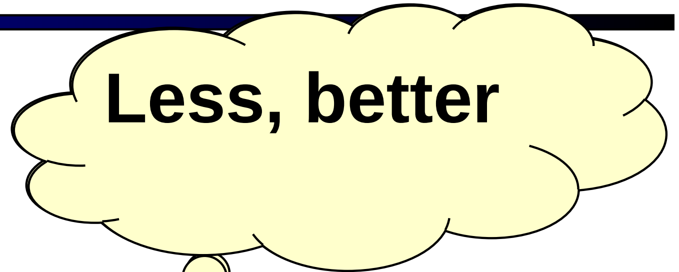




- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

8-puzzle with A*



- $f(x) = g(x) + h(x)$
- $g(x)$: The number of moves from the initial state to x

  =number of misplaced tiles[tiles in wrong places)
- $h(x)$: ?

2	8	3
7	1	4
	6	5

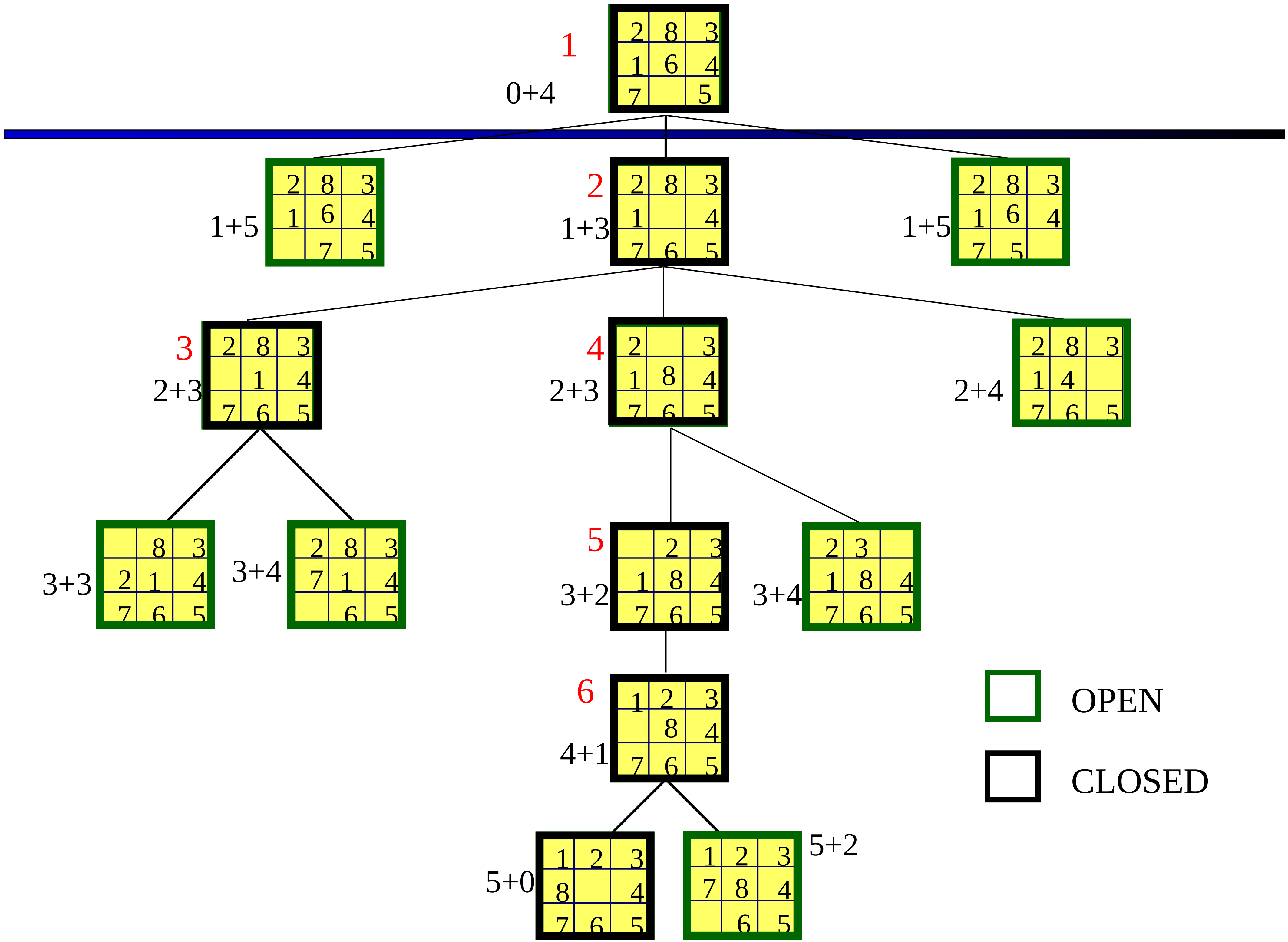
$h(x)=4$

1	2	3
7	8	4
	6	5

$h(x)=2$

1	2	3
	8	4
7	6	5

$h(x)=1$



1
0+4
2 8 3
1 6 4
7 5
1+5
2
1+3
3
2+3
4
2+3
2+4
3+3
3+4
5
3+2
3+4
6
4+1
5+0
5+2
OPEN
CLOSED

Different h(n) for A*

- $h_1(n)$ = number of misplaced tiles[tiles in wrong places)
- $h_2(n)$ = total Manhattan distance [how many moves to reach right place](i.e., no. of squares from desired location of each tile)

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

Tile 1 : 1
Tile 2 : 1
Tile 6 : 1
Tile 8 : 2

Different h(n) for A*

1	2	3
8		4
7	6	5

$h_1(n)$ **= number of misplaced tiles[tiles in wrong places)**
$h_2(n)$ **= total Manhattan distance [how many moves to reach right place](i.e., no. of squares from desired location of each tile)**

1	2	3
7	8	4
	6	5

1	2	7
8	3	4
	6	5

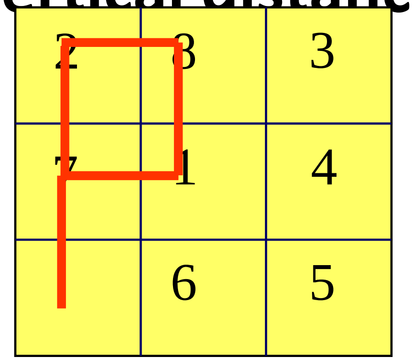
1	2	3
8	4	7
	6	5

$h1(x)=2$ $h1(x)=2$ $h1(x)=2$

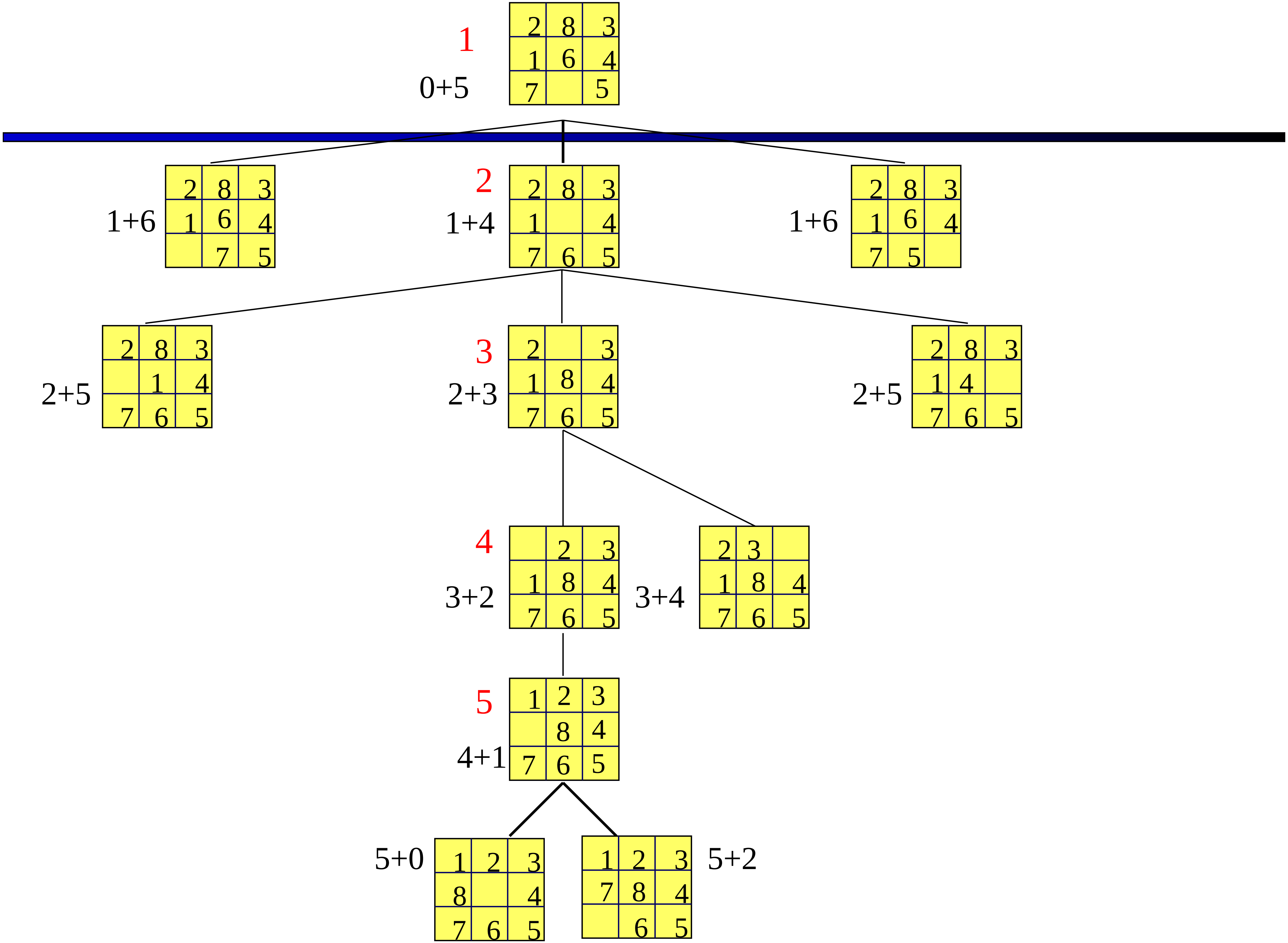
$h2(x)=0+1+1+0=2$ $h2(x)=1+1+2+2=6$ $h2(x)=1+1+2+1=4$

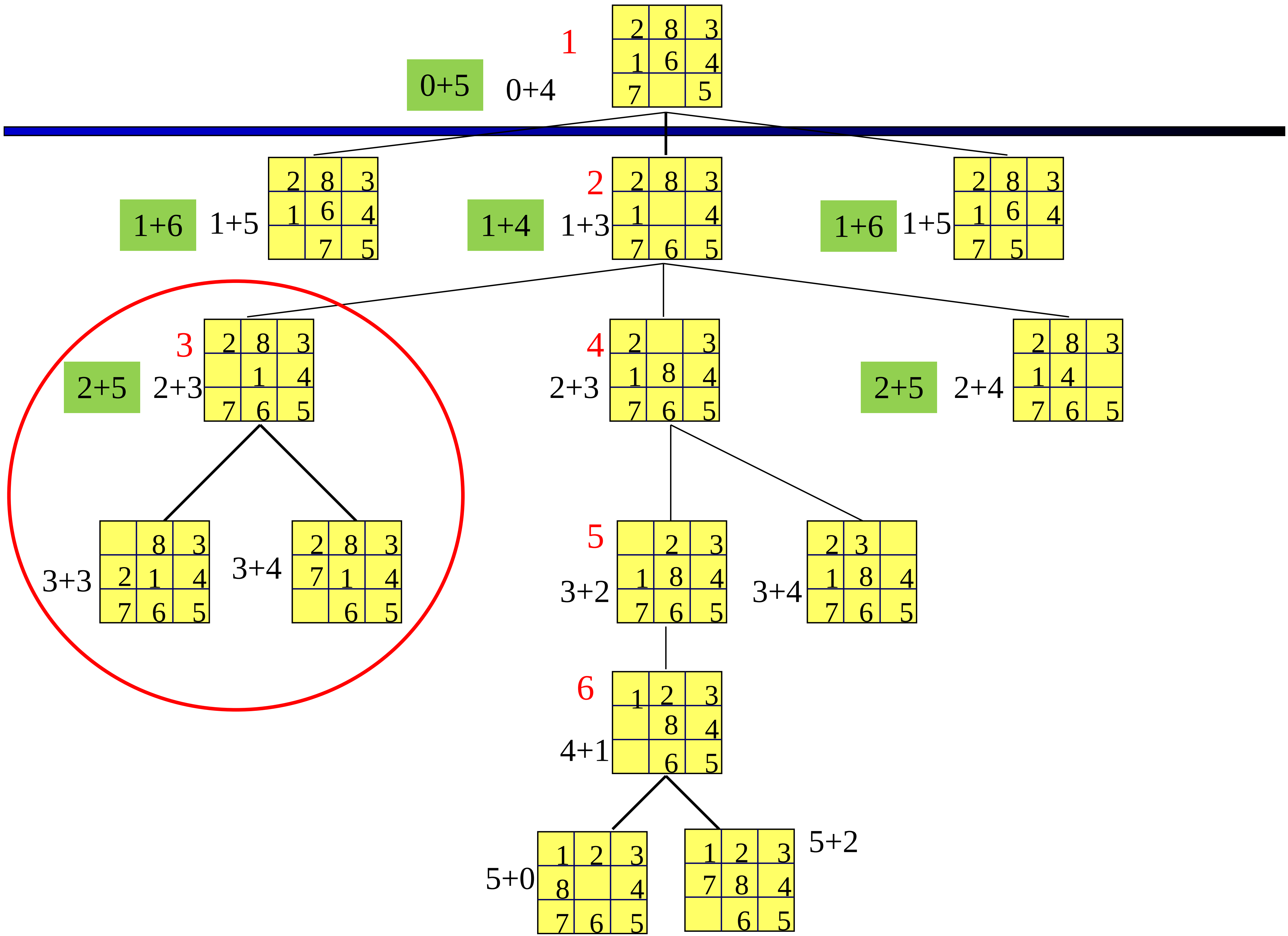
Different h(n) for A*

- $f(x) = g(x) + h(x)$
- $g(x)$:The number of moves from the initial state to x
- $h(x)$:total Manhattan distance [how many moves to reach right place](i.e., no. of squares from desired location of each tile)
- the sum of the distances of the tiles from their goal positions, using city block distance, which is the sum of the horizontal and vertical distances (Manhattan Distance)



$h(x) = 2 + 1 + 1 + 2 = 6$


1
0+5
2
1+6
1+4
1+6
3
2+5
2+3
2+5
4
3+2
3+4
5
4+1
5+0
5+2



1
0+5
0+4
2 8 3
1 6 4
7 5
1+6
1+5
2 8 3
1 6 4
7 5
2
1+4
1+3
2 8 3
1 4
7 6 5
1+6
1+5
2 8 3
1 6 4
7 5
3
2+5
2+3
2 8 3
1 4
7 6 5
4
2+3
2 3
1 8 4
7 6 5
2+5
2+4
2 8 3
1 4
7 6 5
3+3
8 3
2 1 4
7 6 5
3+4
2 8 3
7 1 4
6 5
5
3+2
2 3
1 8 4
7 6 5
3+4
2 3
1 8 4
7 6 5
6
4+1
1 2 3
8 4
7 6 5
5+0
1 2 3
8 4
7 6 5
5+2
1 2 3
7 8 4
6 5

A*
Search
Example

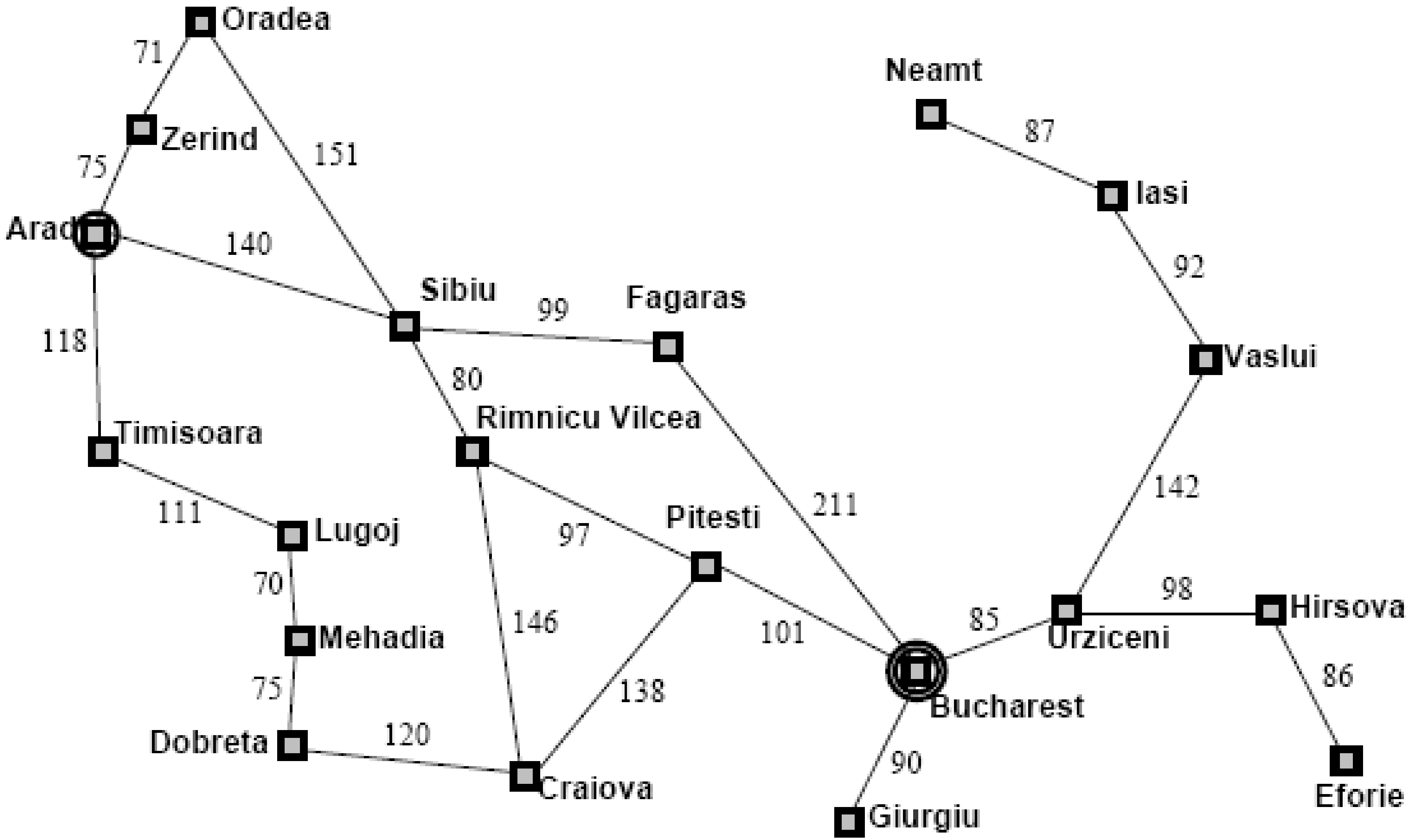
Straight Line Distances to Bucharest

Town	SLD
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244

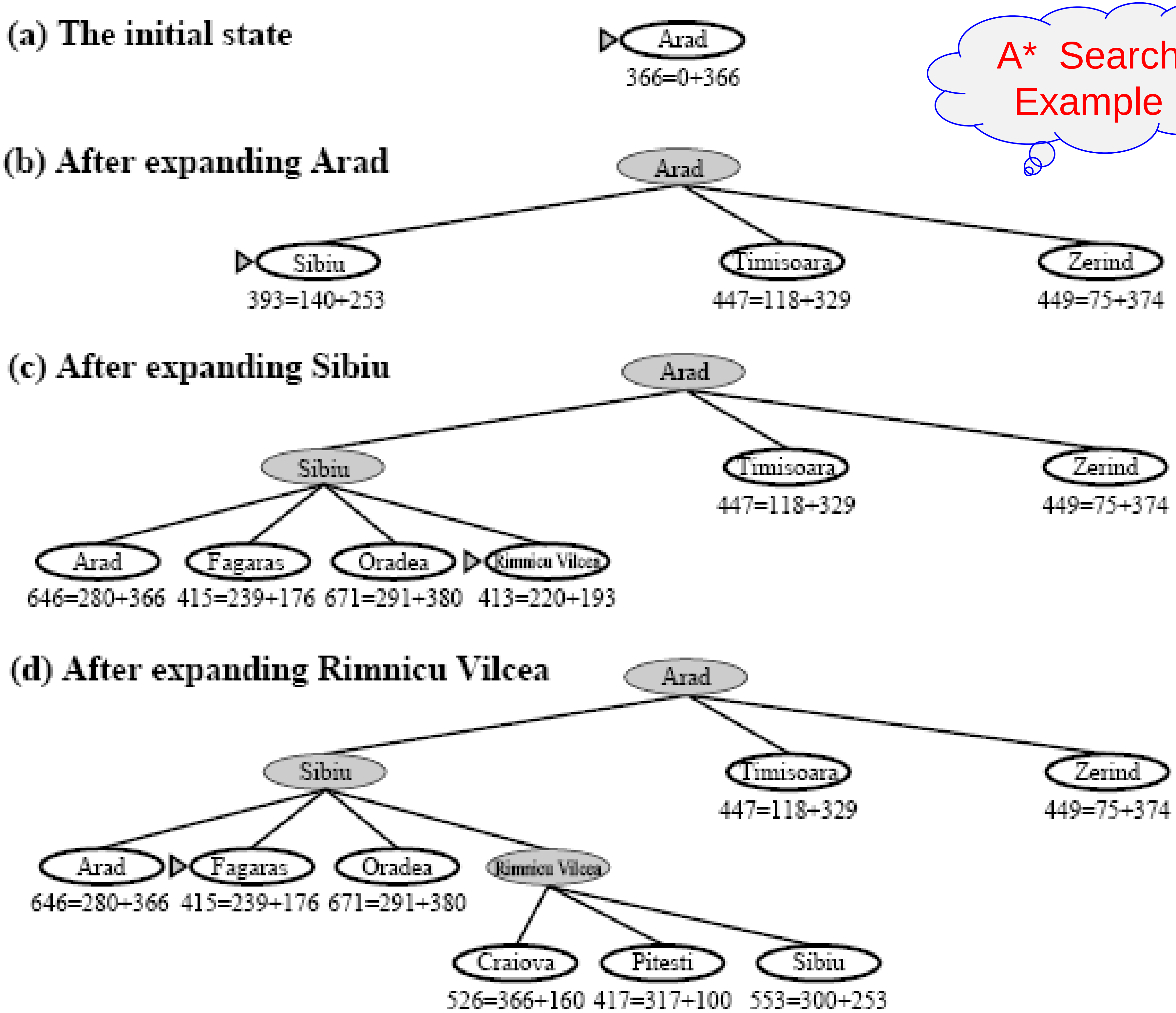
Town	SLD
Mehadai	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

We can use straight line distances as an admissible heuristic as they will never overestimate the cost to the goal. This is because there is no shorter distance between two cities than the straight line distance.

A* Search Example

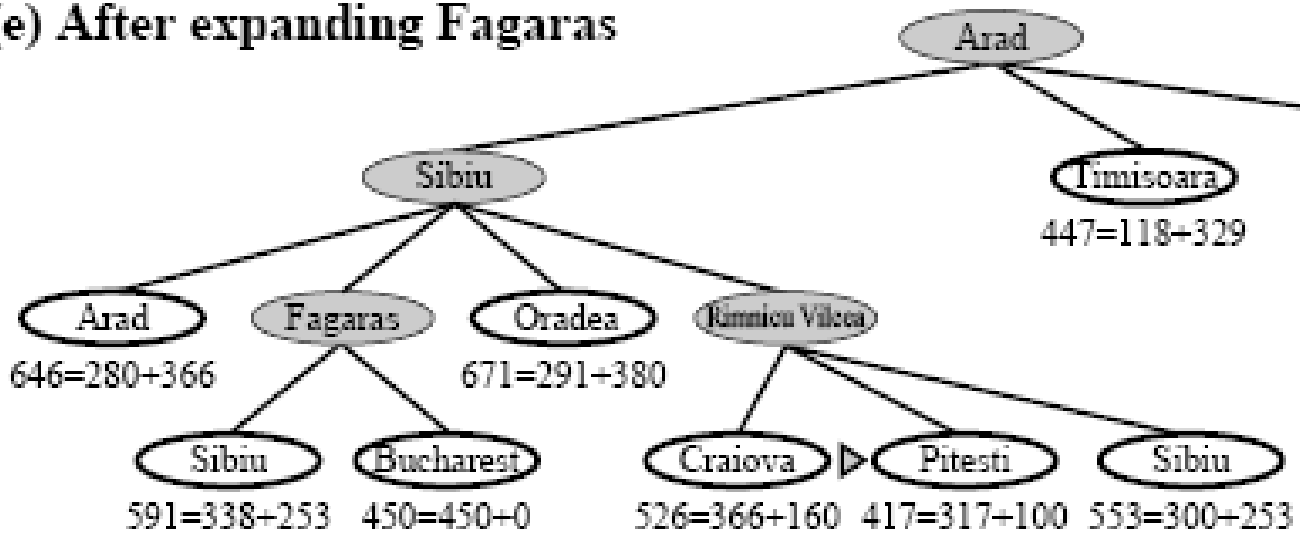


A* Search Example

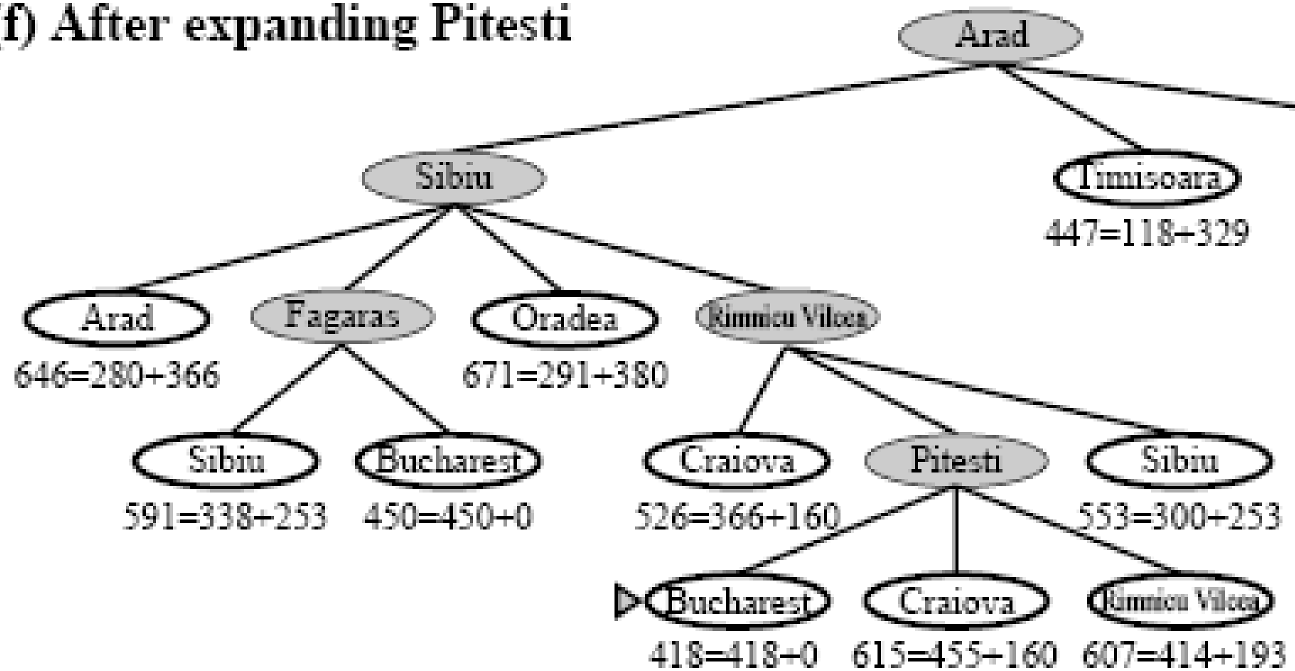


A* Search Example

(e) After expanding Fagaras



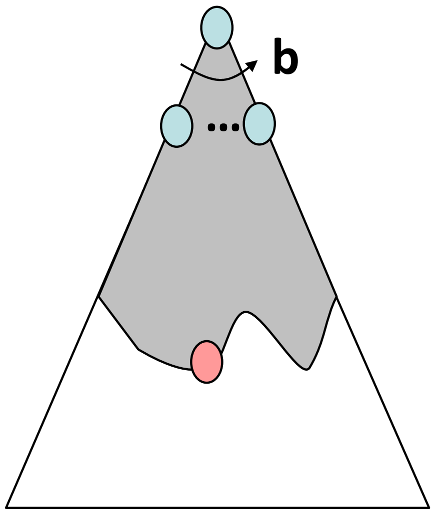
(f) After expanding Pitesti



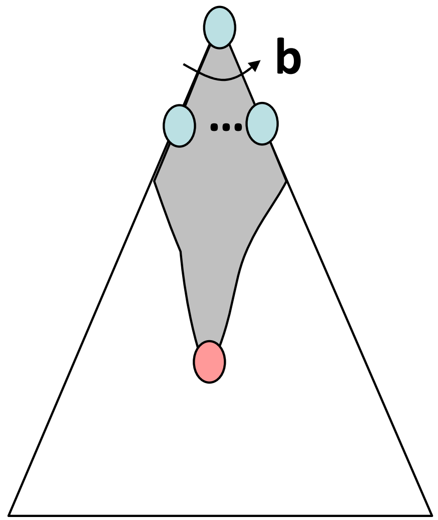
Properties of A*

Properties of A*

Uniform-Cost

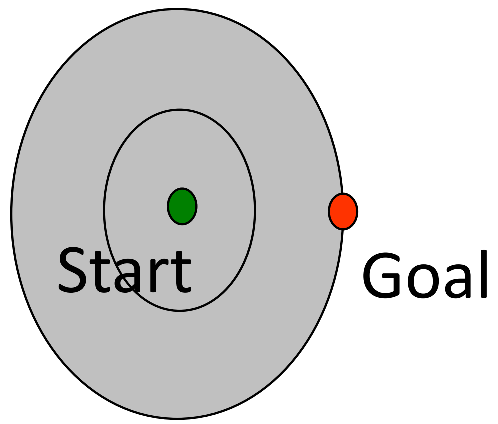


A*

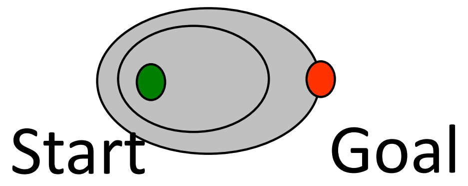


UCS vs A* Contours

- Uniform-cost expands equally in all "directions"



- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



- hedge one's bets

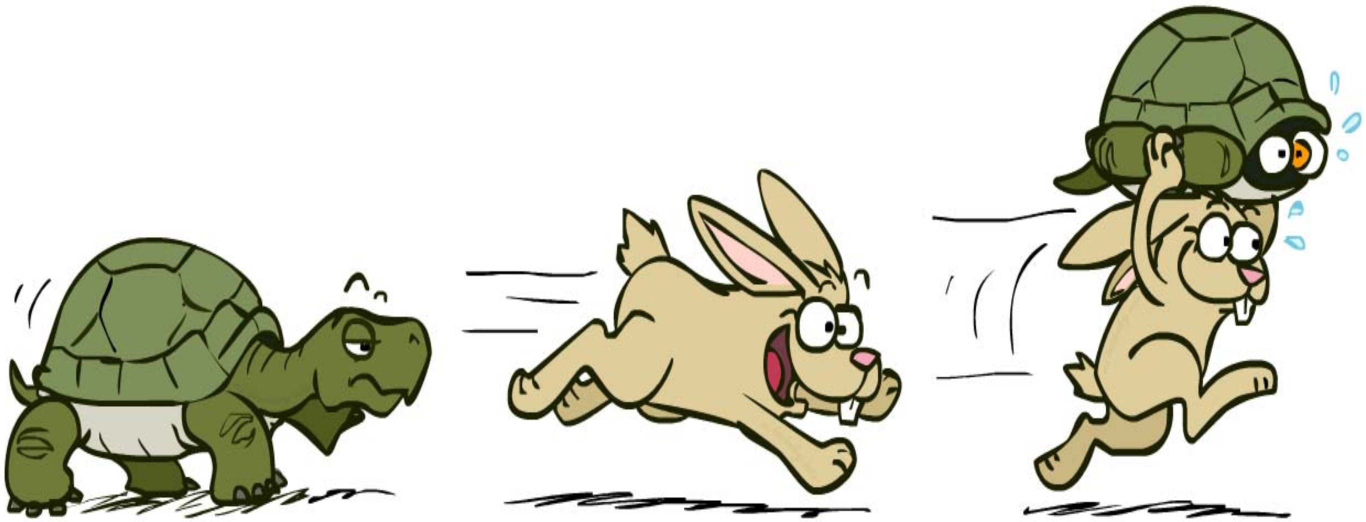
A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

A*: Summary

A*: Summary

- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems



Properties of A*

- <u>Complete?</u> Yes
- <u>Time?</u> Exponential
- <u>Space?</u> Keeps all nodes in memory
- <u>Optimal?</u> Yes

Thank you

End of Chapter 3