

## Chapter 3

# Informed Searching

Mustafa Jarrar

[University of Birzeit](#)

# Watch this lecture and download the slides



Course Page: <http://www.jarrar.info/courses/AI/>

More Online Courses at: <http://www.jarrar.info>

Acknowledgement:

This lecture is based on (but not limited to) chapter 4 in "S. Russell and P. Norvig: *Artificial Intelligence: A Modern Approach*".

# Discussion and Motivation



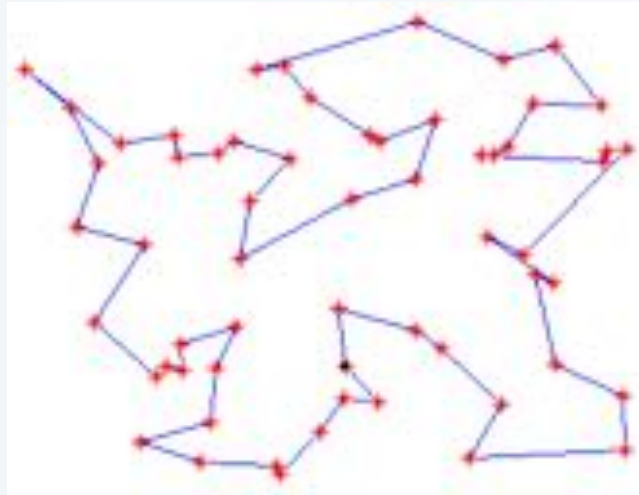
How to determine the minimum number of coins to give while making change?

➔ The coin of the highest value first ?

# Discussion and Motivation

## Travel Salesperson Problem

Given a list of cities and their pair wise distances, the task is to find a shortest possible tour that visits each city exactly once.



- Any idea how to improve this type of search?
- What type of information we may use to improve our search?
- Do you think this idea is useful: (At each stage visit the unvisited city nearest to the current city)?

# Best-first search

Idea: use an **evaluation function**  $f(n)$  for each node

- family of search methods with various evaluation functions (estimate of "desirability")
- usually gives an estimate of the distance to the goal
- often referred to as *heuristics* in this context
- Expand most desirable unexpanded node.
- A **heuristic function** ranks alternatives at each branching step based on the available information (heuristically) in order to make a decision about which branch to follow during a search.

## Implementation:

Order the nodes in fringe in decreasing order of desirability.

## Special cases:

- greedy best-first search

- A search

# Romania with step costs in km



Suppose we can have this info (SLD)  
 → How can we use it to improve our search?

# Greedy best-first search

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

Utilizes a **h** heuristic function as evaluation function

- $f(n) = h(n)$  = estimated cost from the current node to a goal.
- Heuristic functions are **problem-specific**.
- Often straight-line distance for route-finding and similar problems.
- Often better than depth-first, although worst-time complexities are equal or worse (space).

# Greedy best-first search example

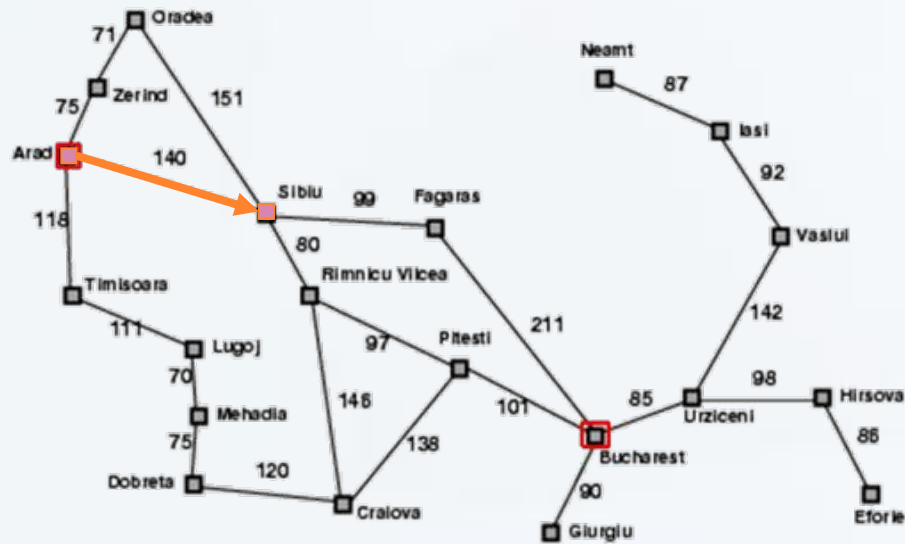
Example from [1]



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



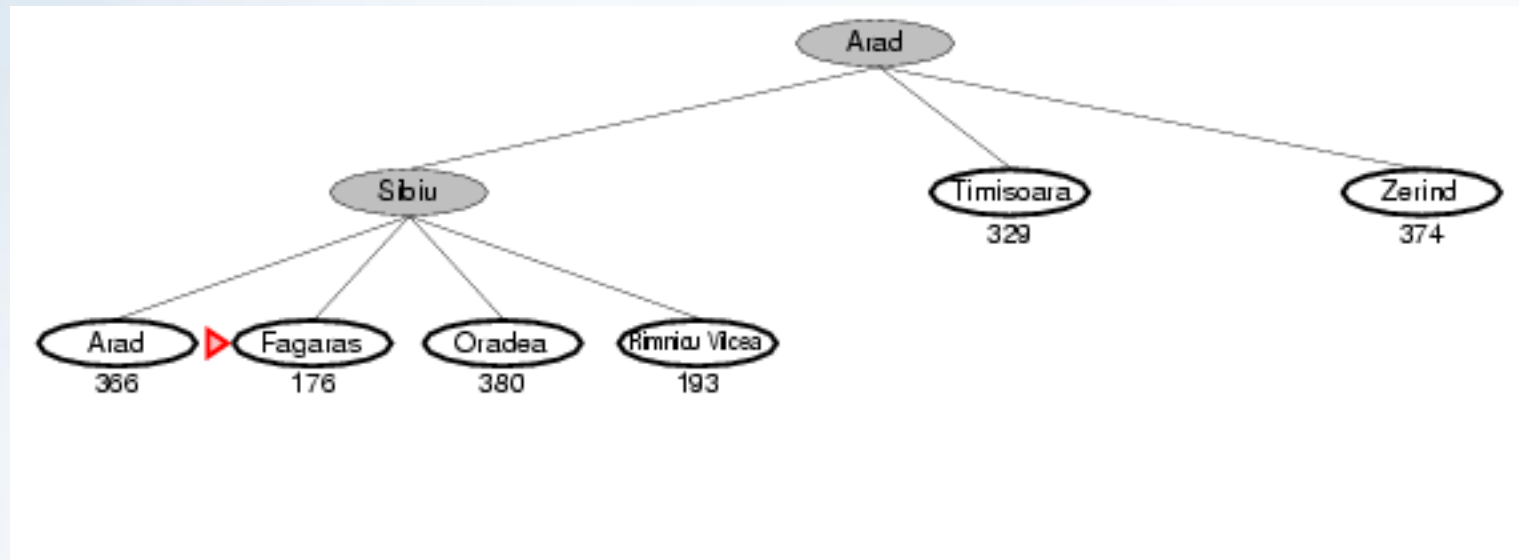
# Greedy best-first search example



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

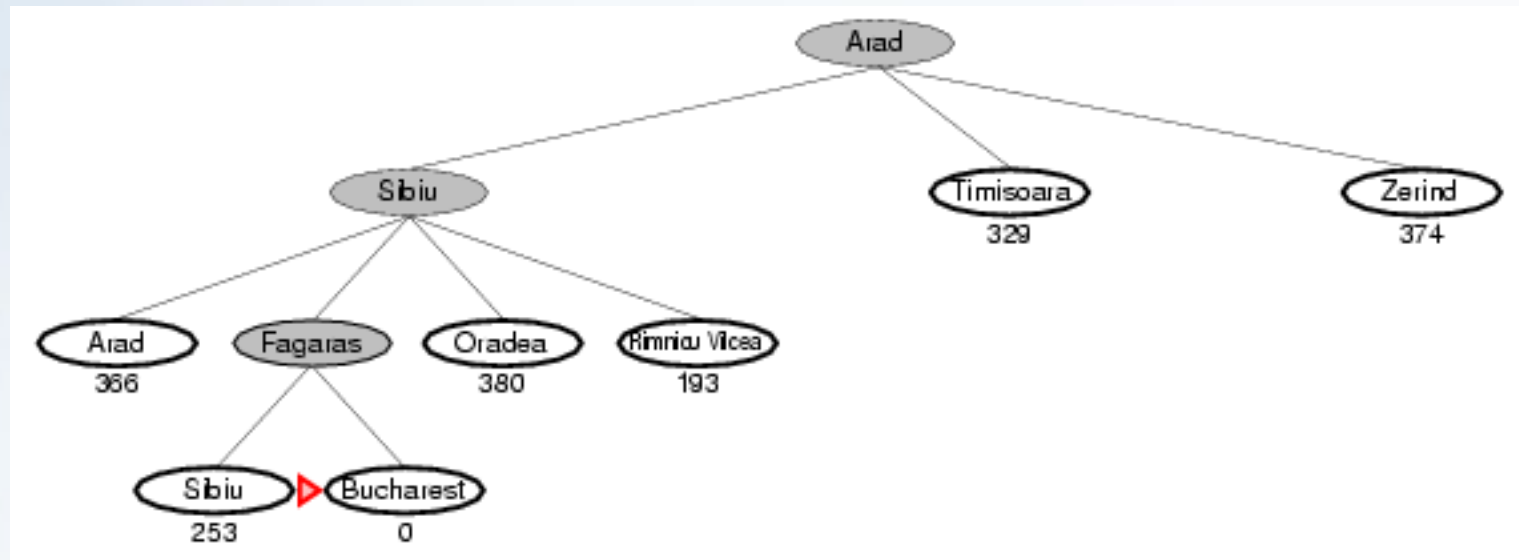
# Greedy best-first search example



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

# Greedy best-first search example



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	380
Oradea	234
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

# Greedy best-first search

```
function GREEDY-BEST-FIRST-SEARCH(initialState, goalTest)
  returns SUCCESS or FAILURE : /* Cost  $f(n) = h(n)$  */

  frontier = Heap.new(initialState)
  explored = Set.new()

  while not frontier.isEmpty():
    state = frontier.deleteMin()
    explored.add(state)

    if goalTest(state):
      return SUCCESS(state)

    for neighbor in state.neighbors():
      if neighbor not in frontier  $\cup$  explored:
        frontier.insert(neighbor)
      else if neighbor in frontier:
        frontier.decreaseKey(neighbor)

  return FAILURE
```

# Properties of greedy best-first search

Complete: No – can get stuck in loops (e.g., lasi → Neamt → lasi → Neamt → ....)

Time:  $O(b^m)$ , but a good heuristic can give significant improvement

Space:  $O(b^m)$  -- keeps all nodes in memory

Optimal: No

b	branching factor
m	maximum depth of the search tree

# Discussion

Do you think  $h_{\text{SLD}}(n)$  is admissible?

Would you use  $h_{\text{SLD}}(n)$  in Palestine? How? Why?

Did you find the Greedy idea useful?

→ Ideas to improve it?

# A\* search

Idea: avoid expanding paths that are already expensive.

Evaluation function = path cost + estimated cost to the goal

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

- $g(n)$  = 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

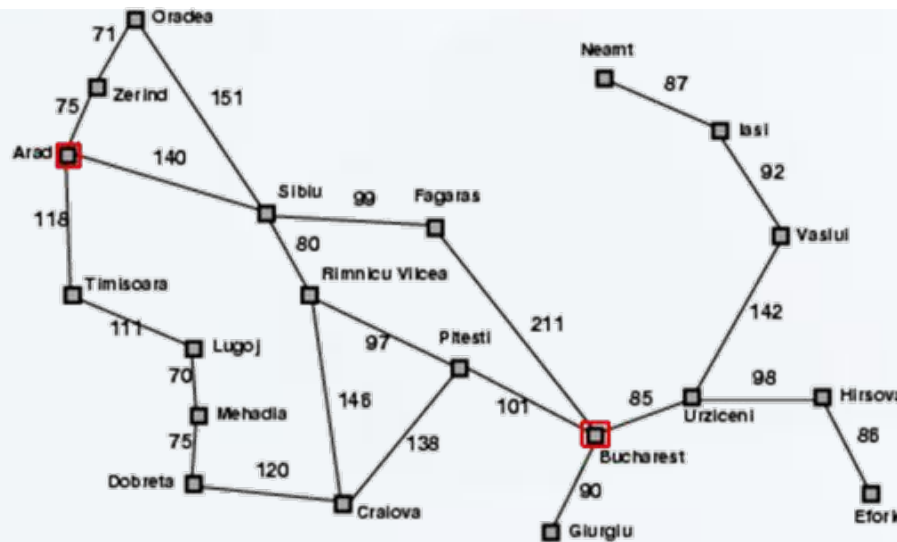
Combines greedy and uniform-cost search to find the (estimated) cheapest path through the current node

- Heuristics must be admissible
  - Never overestimate the cost to reach the goal
- Very good search method, but with complexity problems

# A\* search example

Example from [1]

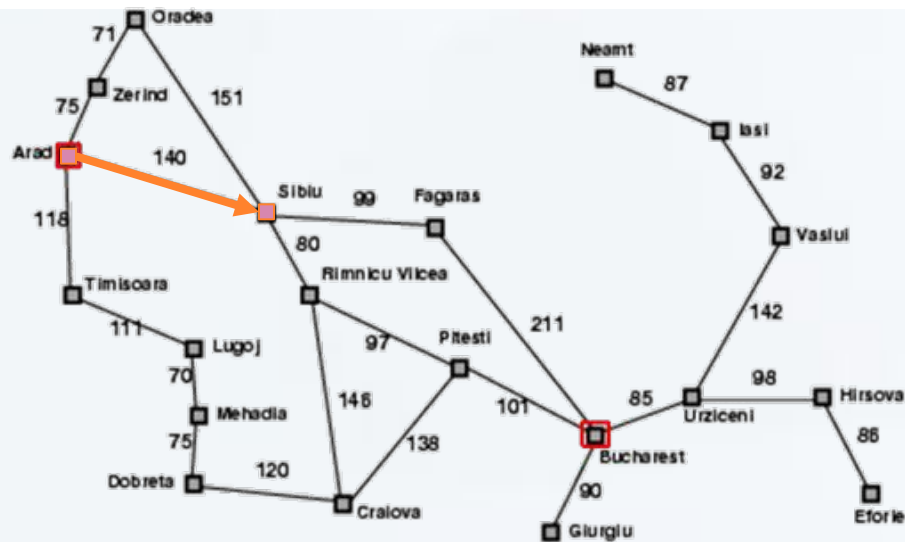
▶ Arad  
366=0+366



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



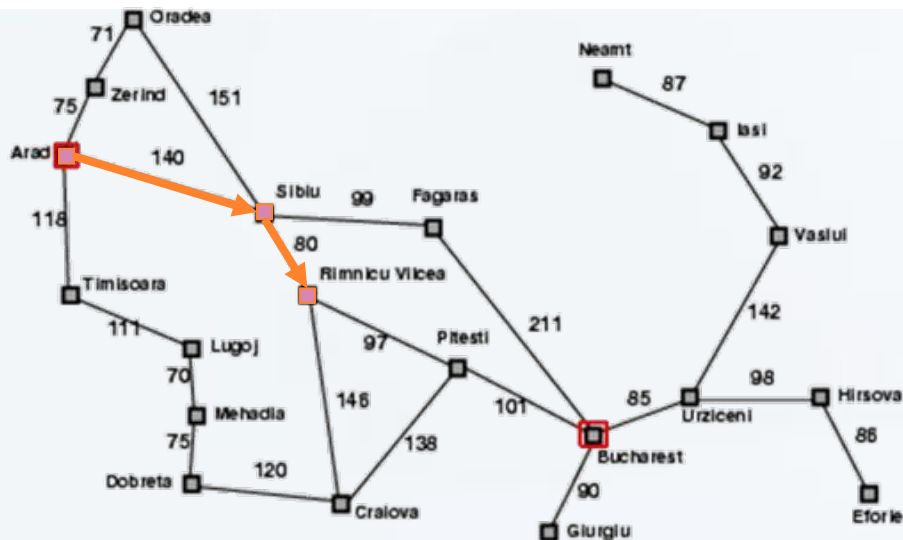
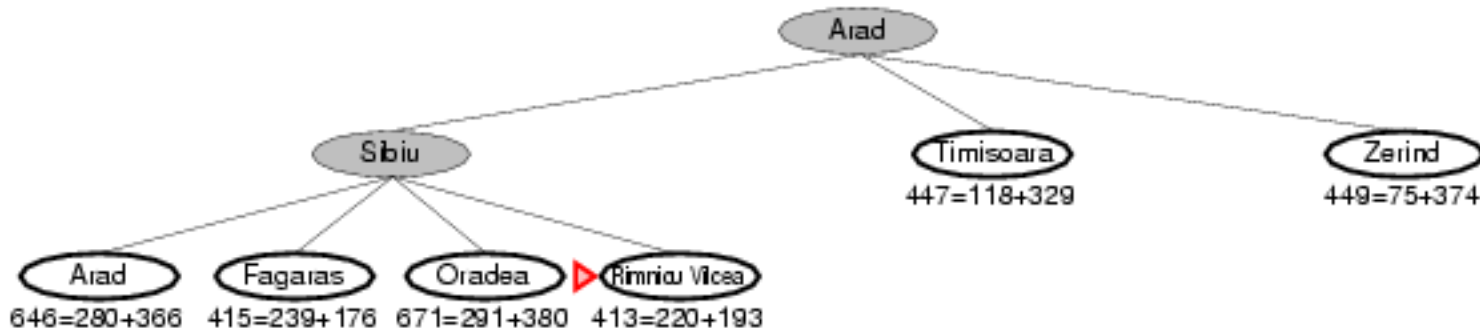
# A\* search example



Straight-line distance to Bucharest

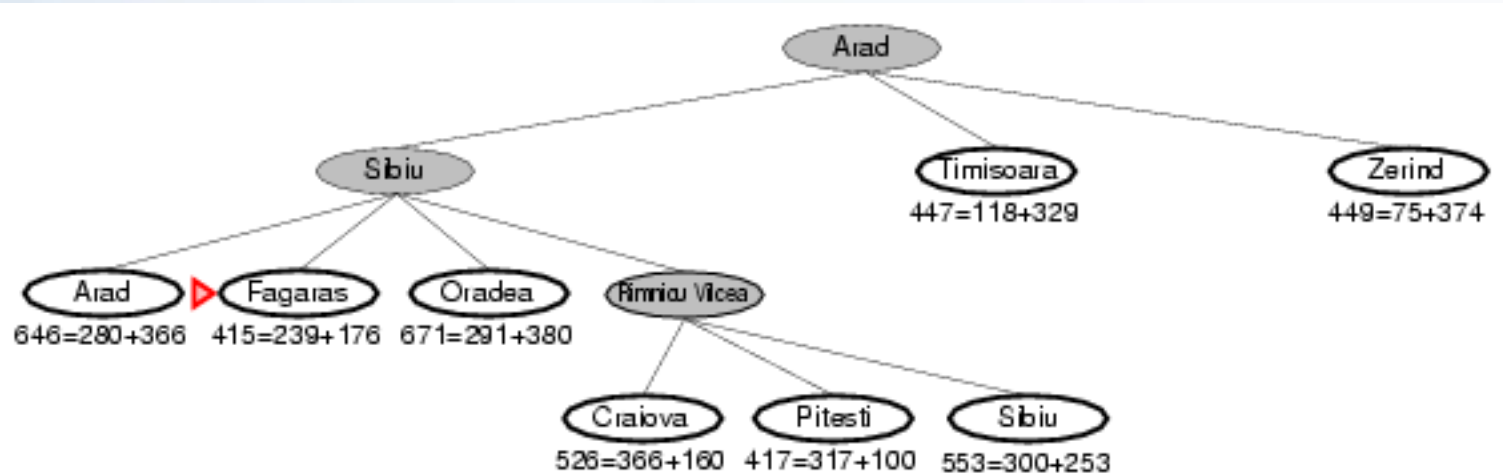
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

# A\* search example



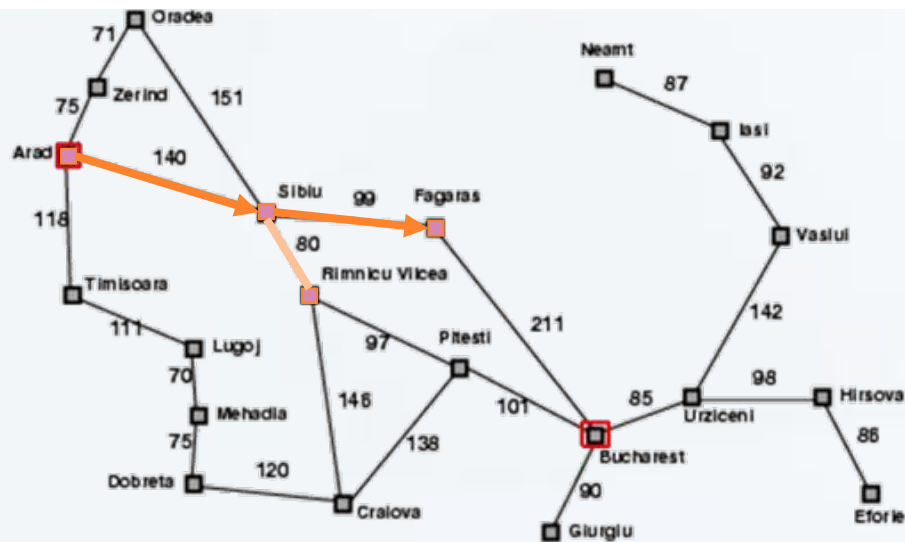
City	Straight-line distance to Bucharest
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	234
Neamt	241
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

# A\* search example

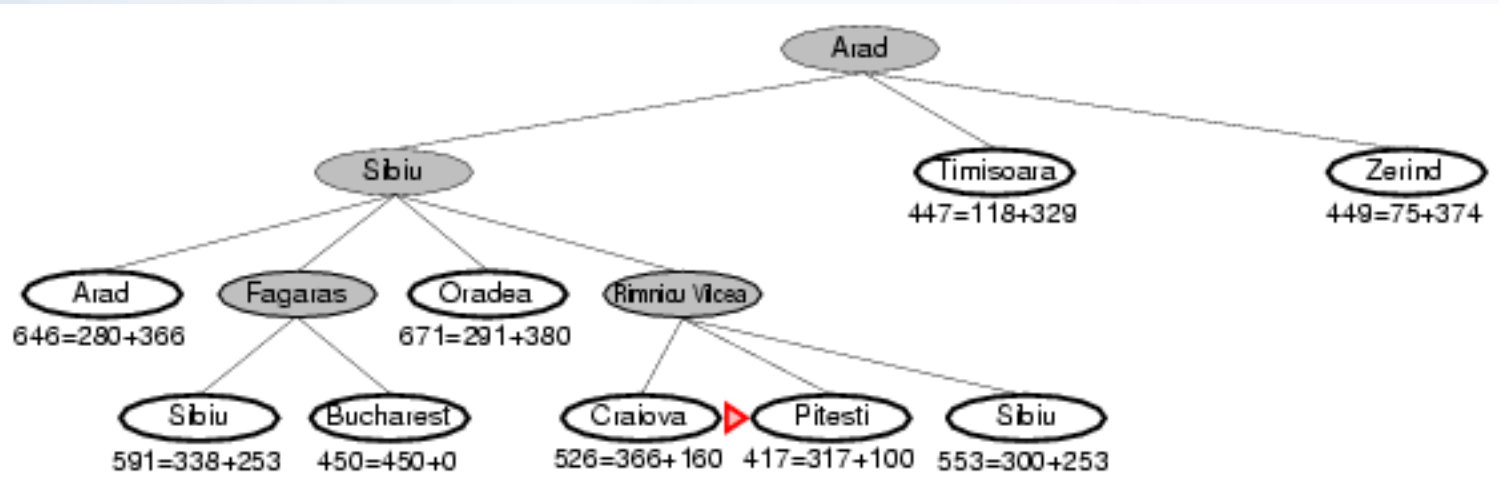


Straight-line distance to Bucharest

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

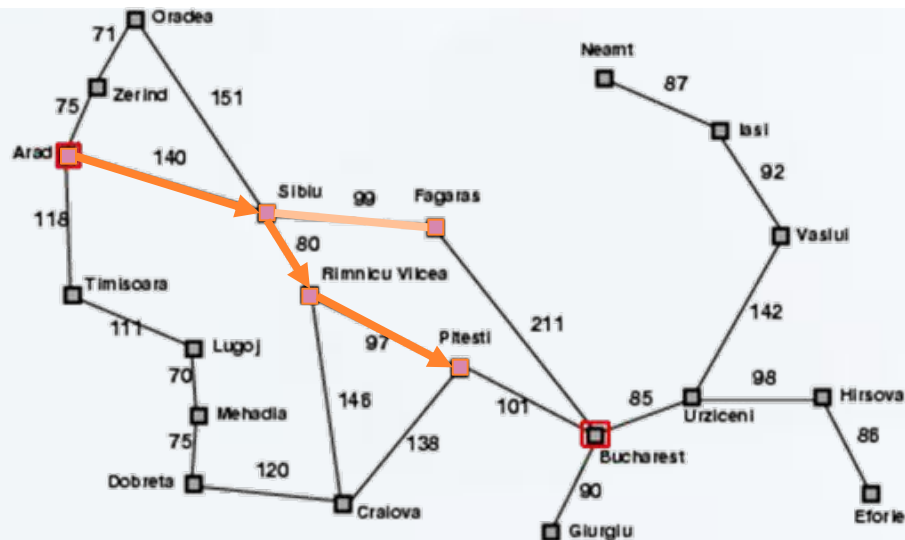


# A\* search example

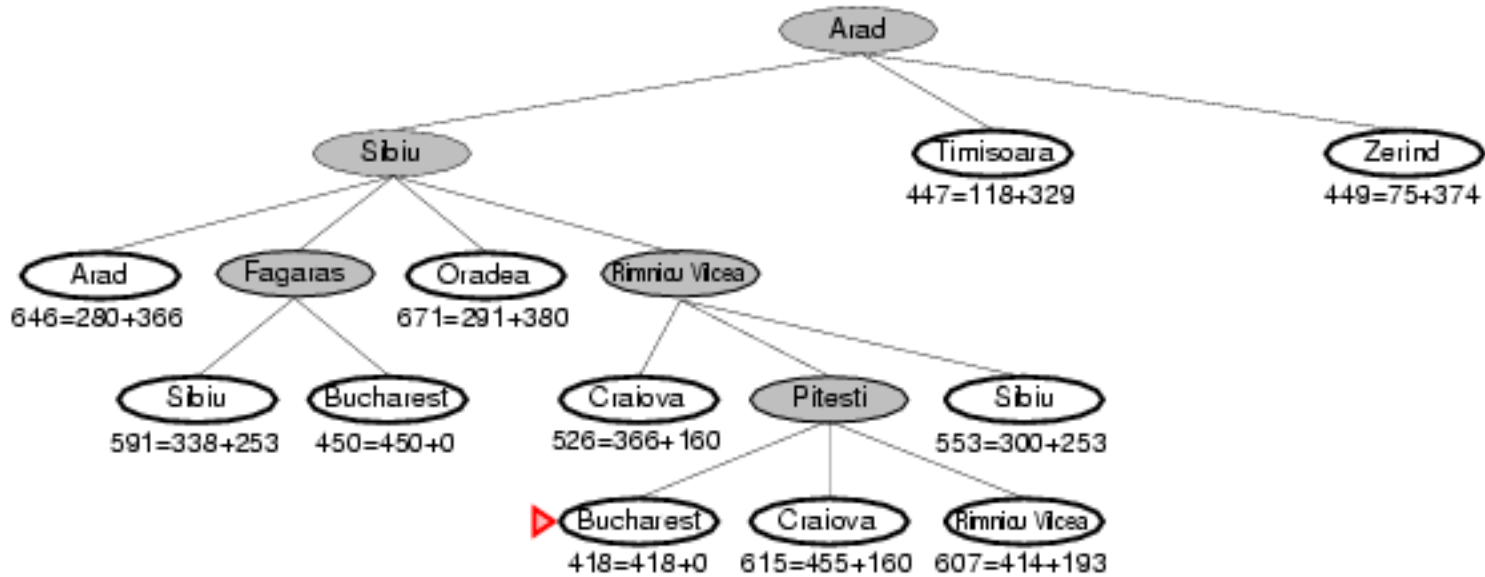


Straight-line distance to Bucharest

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

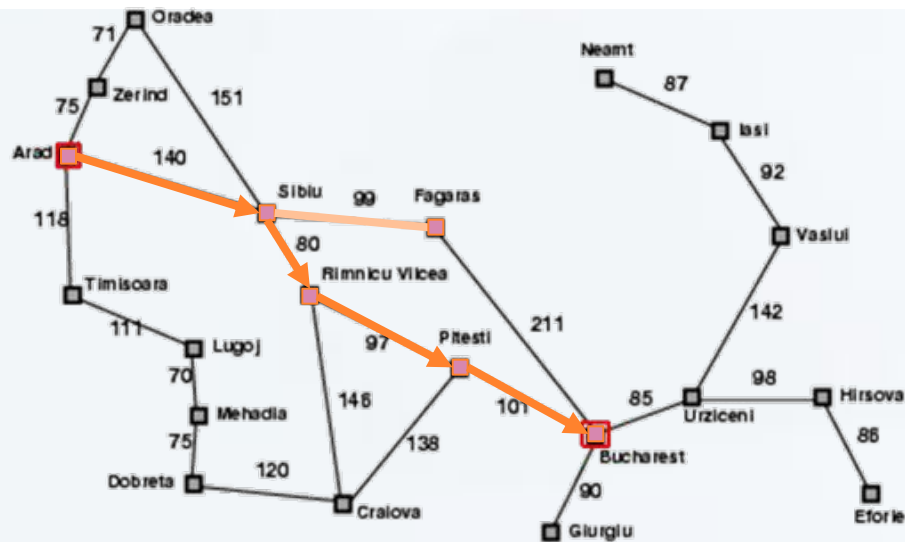


# A\* search example



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	234
Neamt	241
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



# A\* Algorithm

```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost  $f(n) = g(n) + h(n)$  */

    frontier = Heap.new(initialState)
    explored = Set.new()

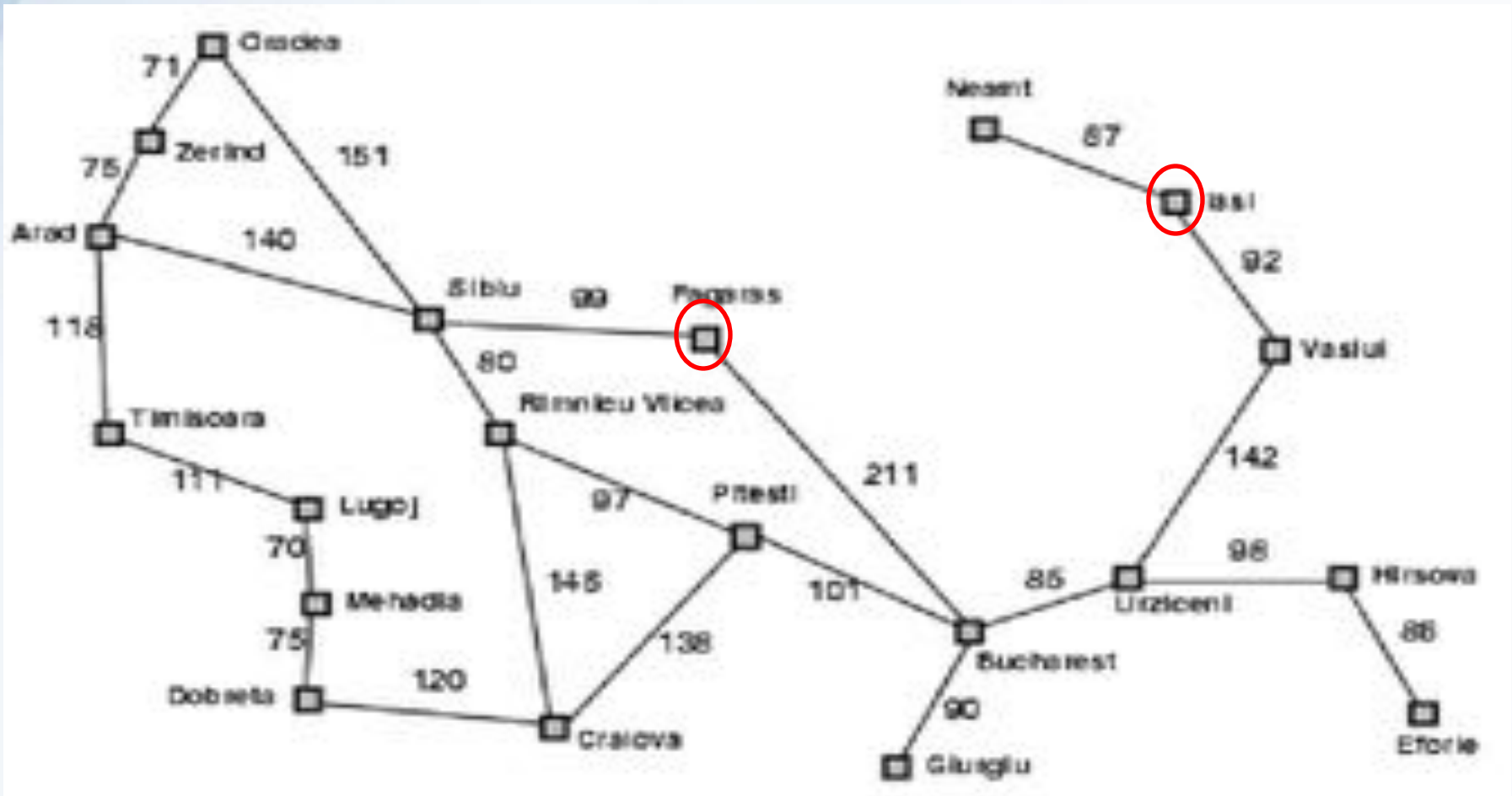
    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier  $\cup$  explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

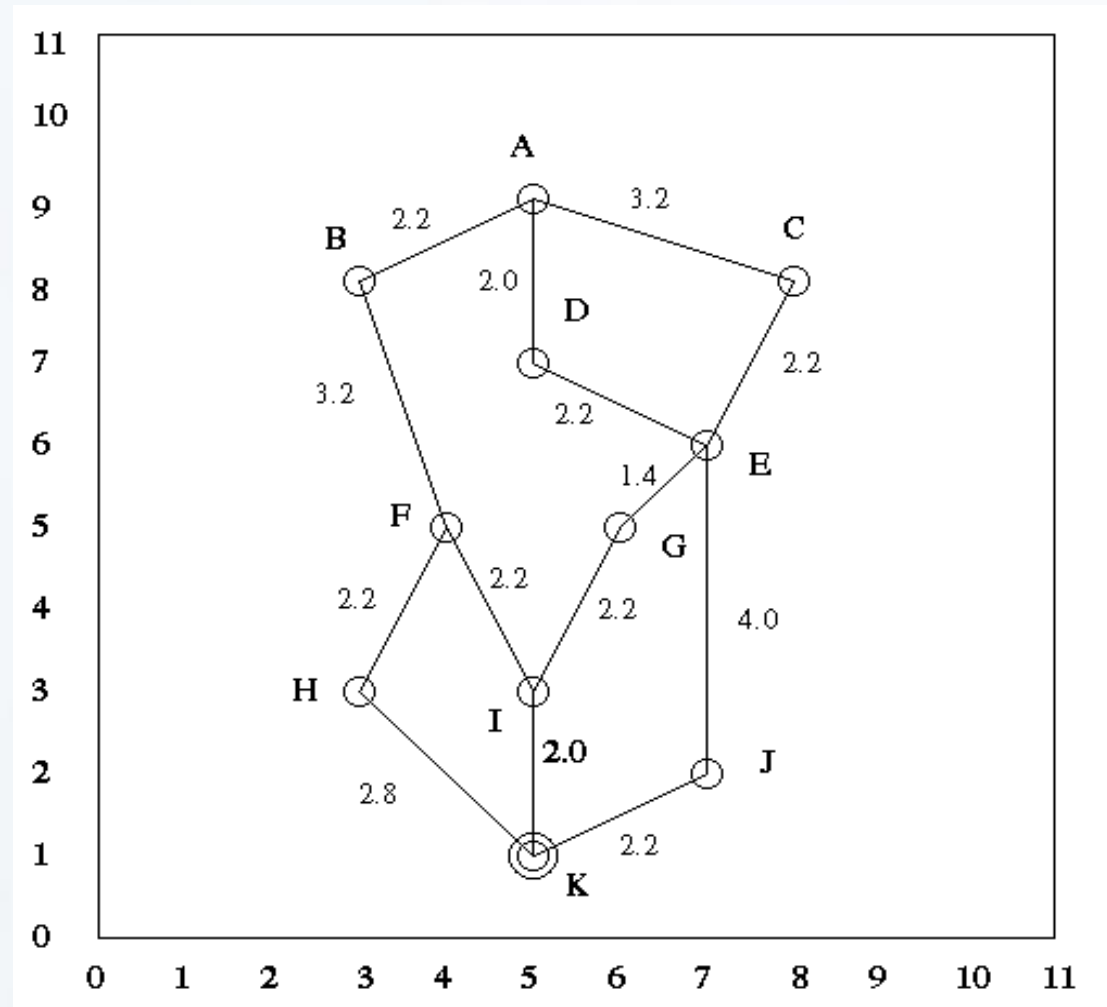
# A\* Exercise



How will A\* get from Iasi to Fagaras?

# A\* Exercise

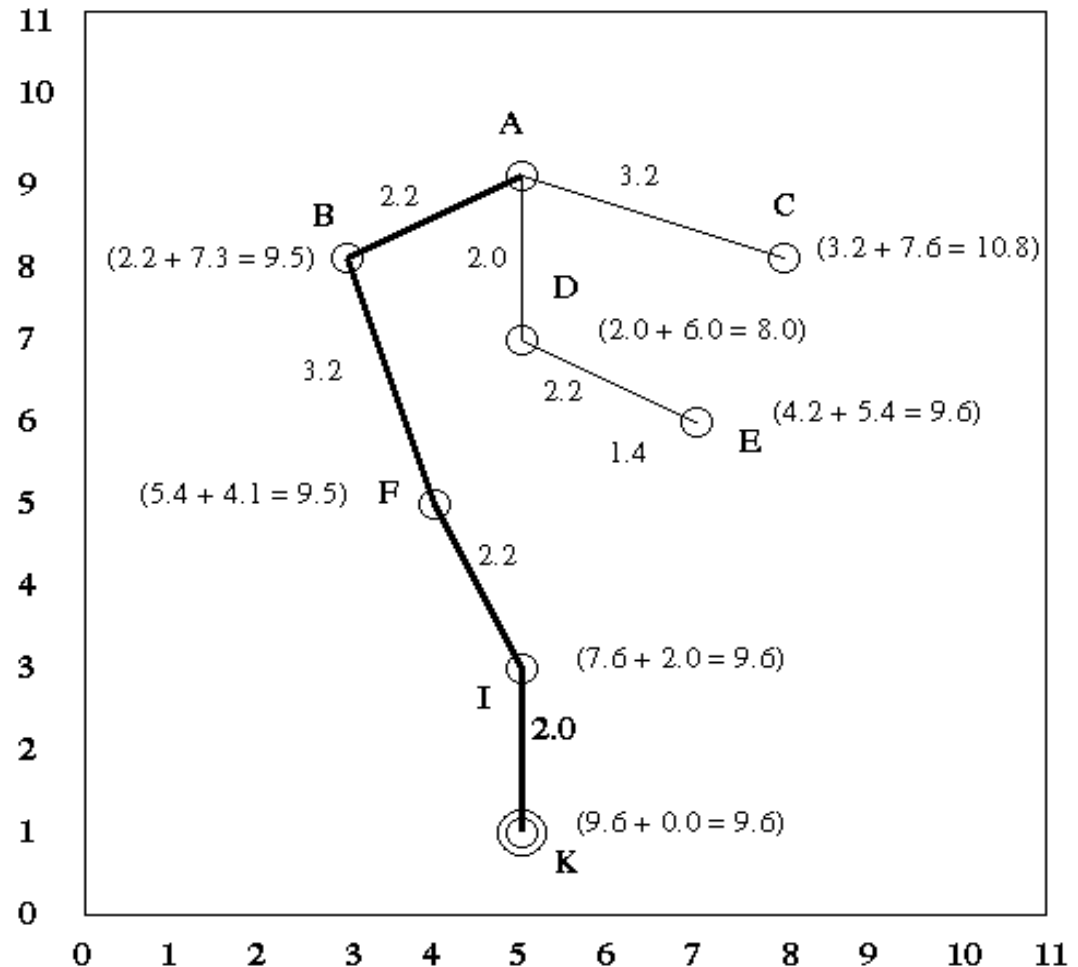
<u>Node</u>	<u>Coordinates</u>	<u>SL Distance</u>
A	(5,9)	8.0
B	(3,8)	7.3
C	(8,8)	7.6
D	(5,7)	6.0
E	(7,6)	5.4
F	(4,5)	4.1
G	(6,5)	4.1
H	(3,3)	2.8
I	(5,3)	2.0
J	(7,2)	2.2
K	(5,1)	0.0





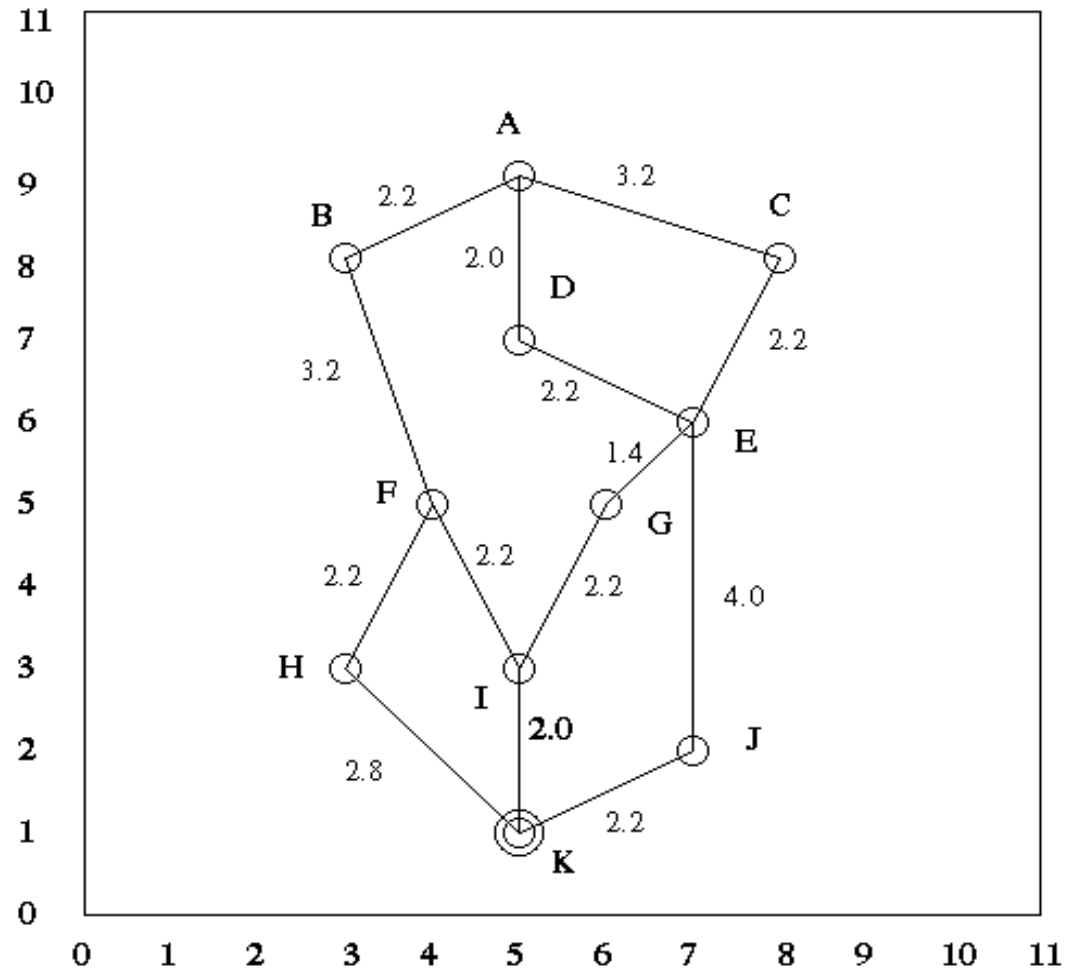
# Solution to A\* Exercise

Node	Coordinates	SL Distance
A	(5,9)	8.0
B	(3,8)	7.3
C	(8,8)	7.6
D	(5,7)	6.0
E	(7,6)	5.4
F	(4,5)	4.1
G	(6,5)	4.1
H	(3,3)	2.8
I	(5,3)	2.0
J	(7,2)	2.2
K	(5,1)	0.0



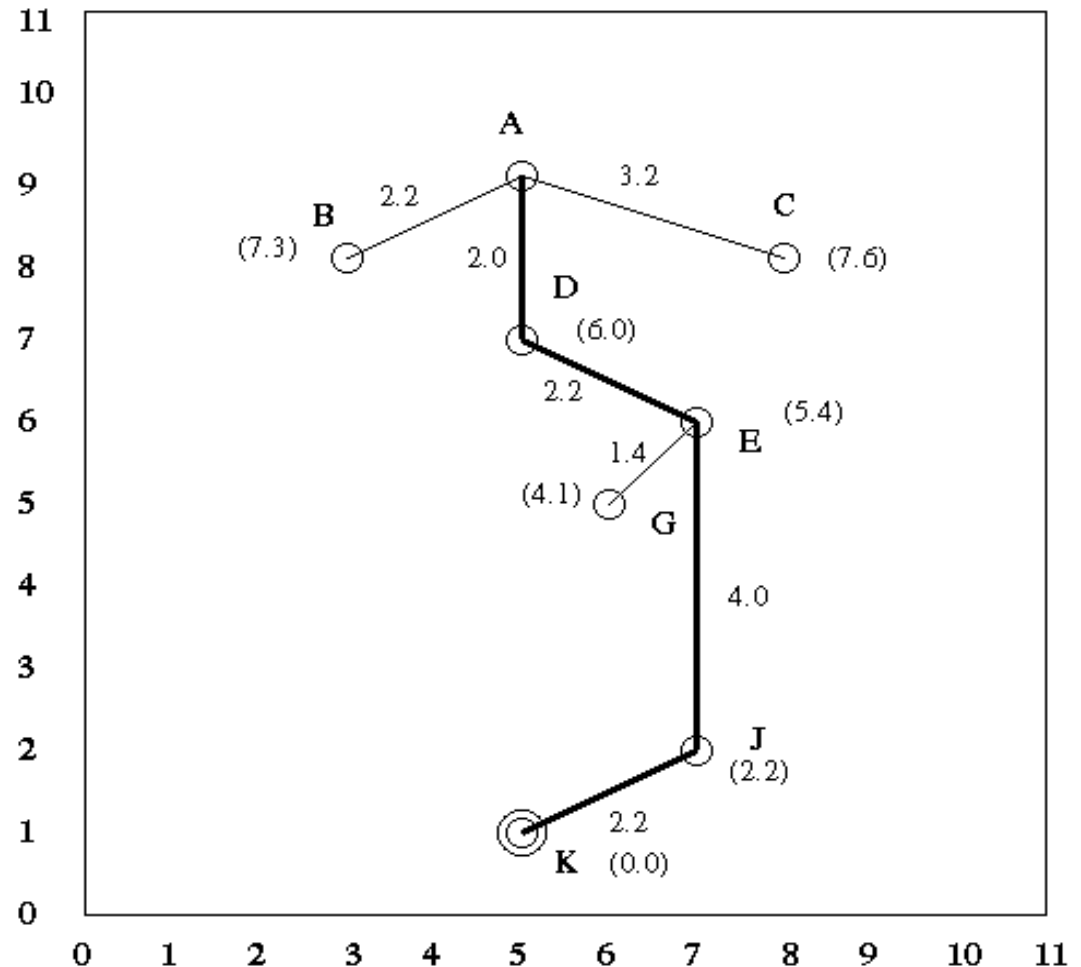
# Greedy Best-First Exercise

<u>Node</u>	<u>Coordinates</u>	<u>Distance</u>
A	(5,9)	8.0
B	(3,8)	7.3
C	(8,8)	7.6
D	(5,7)	6.0
E	(7,6)	5.4
F	(4,5)	4.1
G	(6,5)	4.1
H	(3,3)	2.8
I	(5,3)	2.0
J	(7,2)	2.2
K	(5,1)	0.0



# Solution to Greedy Best-First Exercise

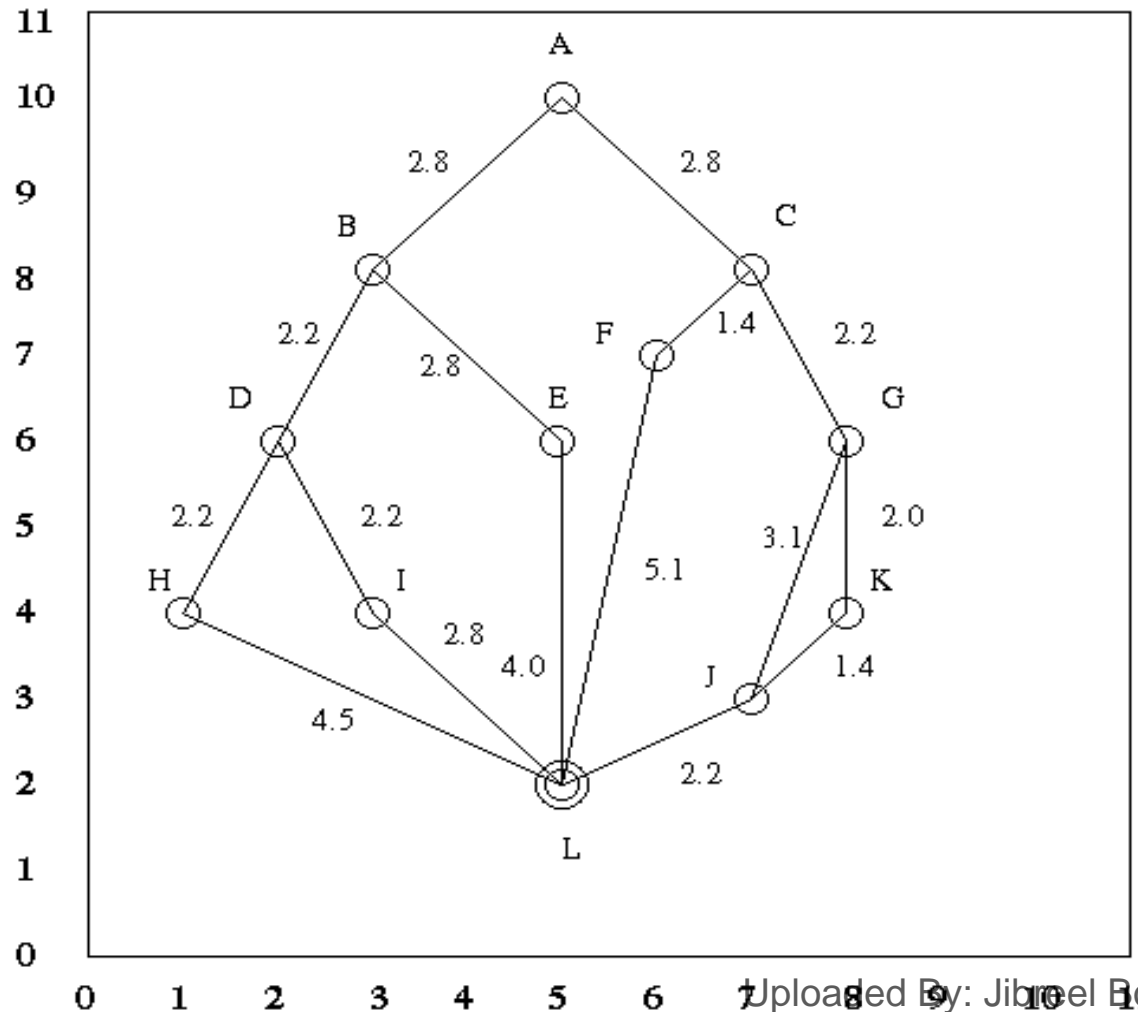
Node	Coordinates	Distance
A	(5,9)	8.0
B	(3,8)	7.3
C	(8,8)	7.6
D	(5,7)	6.0
E	(7,6)	5.4
F	(7,6)	5.4
G	(6,5)	4.1
H	(3,3)	2.8
I	(5,3)	2.0
J	(7,2)	2.2
K	(5,1)	0.0



# Another Exercise

Do 1) A\* Search and 2) Greedy Best-Fit Search

Node	C	$g(n)$	$h(n)$
A	(5,10)	0.0	8.0
B	(3,8)	2.8	6.3
C	(7,8)	2.8	6.3
D	(2,6)	5.0	5.0
E	(5,6)	5.6	4.0
F	(6,7)	4.2	5.1
G	(8,6)	5.0	5.0
H	(1,4)	7.2	4.5
I	(3,4)	7.2	2.8
J	(7,3)	8.1	2.2
K	(8,4)	7.0	3.6
L	(5,2)	9.6	0.0



# Admissible Heuristics

Based on [4]

A heuristic  $h(n)$  is **admissible** if for every node  $n$ ,  $h(n) \leq h^*(n)$ , 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**.

The heuristic function  $h_{SLD}(n)$  is admissible because it never overestimates the actual road distance)

Theorem-1: If  $h(n)$  is admissible,  $A^*$  using TREE-SEARCH is optimal.

# Optimality of A\* (proof)

Based on [4]

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

Now, 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  $G$ .

**We want to prove:**

$$f(n) < f(G_2)$$

**(then A\* will prefer  $n$  over  $G_2$ )**

$$f(G_2) = g(G_2) \quad \text{since } h(G_2) = 0$$

$$g(G_2) > g(G) \quad \text{since } G_2 \text{ is suboptimal}$$

$$f(G) = g(G) \quad \text{since } h(G) = 0$$

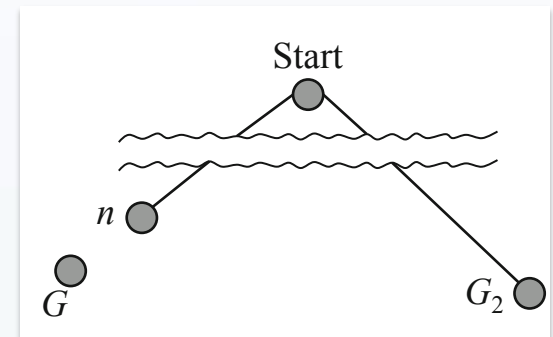
$$\text{Then } f(G_2) > f(G) \quad \text{from above}$$

$$h(n) \leq h^*(n) \quad \text{since } h \text{ is admissible}$$

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

$$\text{Then } f(n) \leq f(G)$$

Thus, A\* will never select  $G_2$  for expansion



# Optimality of A\* (proof)

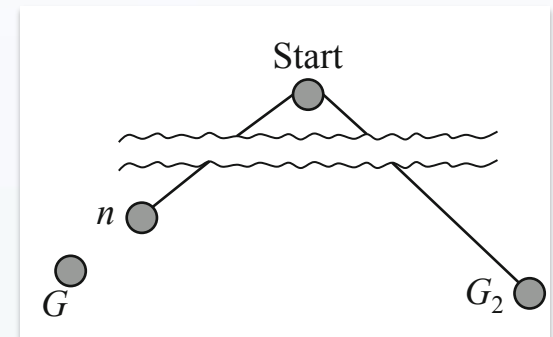
Recall that  $f(n) = g(n) + h(n)$

Now, 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  $G$ .

**We want to prove:**

$$f(n) < f(G_2)$$

**(then A\* will prefer  $n$  over  $G_2$ )**



In other words:

$$f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*,$$

since  $G_2$  is a goal on a non-optimal path ( $C^*$  is the optimal cost)

$$f(n) = g(n) + h(n) \leq C^*, \text{ since } h \text{ is admissible}$$

$f(n) \leq C^* < f(G_2)$ , so  $G_2$  will never be expanded

→ A\* will not expand goals on sub-optimal paths

# Properties of A\*

- **Complete: Yes**

unless there are infinitely many nodes with  $f \leq f(G)$

- **Time: Exponential**

because all nodes such that  $f(n) \leq C^*$  are expanded!

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

fringe is exponentially large

- **Optimal: Yes**



# Memory Bounded Heuristic Search

How can we solve the memory problem for A\* search?

Idea: Try something like iterative deepening search, but the cutoff is  $f$ -cost ( $g+h$ ) at each iteration, rather than depth first.

Two types of memory bounded heuristic searches:

- Recursive BFS
- SMA\*

# Recursive Best First Search (RBFS)

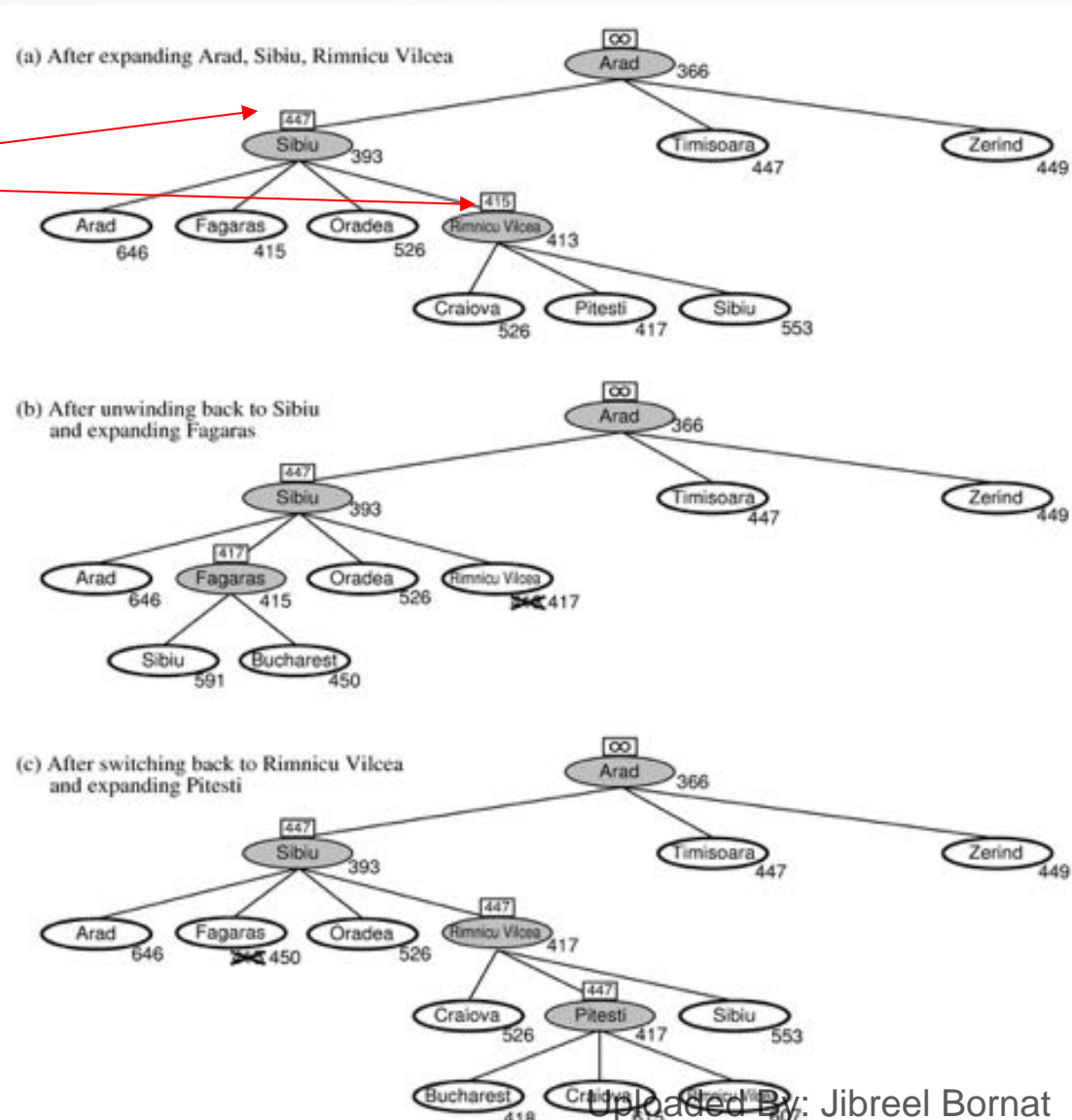
best alternative  
over fringe nodes,  
which are not children:  
*do I want to back up?*

RBFS changes its mind very often in practice.

This is because the  $f=g+h$  become more accurate (less optimistic) as we approach the goal. Hence, higher level nodes have smaller  $f$ -values and will be explored first.

**Problem?** If we have more memory we cannot make use of it.

Any idea to improve this?



# Simple Memory Bounded A\* (SMA\*)

- This is like A\*, but when memory is full we delete the worst node (largest  $f$ -value).
- Like RBFS, we remember the best descendent in the branch we delete.
- If there is a tie (equal  $f$ -values) we first delete the oldest nodes first.
- SMA\* finds the optimal *reachable* solution given the memory constraint.
- But time can still be exponential.

# SMA\* pseudocode

Based on [2]

```
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: Queue, a queue of nodes ordered by f-cost

  Queue  $\leftarrow$  MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
    if Queue is empty then return failure
    n  $\leftarrow$  deepest least-f-cost node in Queue
    if GOAL-TEST(n) then return success
    s  $\leftarrow$  NEXT-SUCCESSOR(n)
    if s is not a goal and is at maximum depth then
      f(s)  $\leftarrow$   $\infty$ 
    else
      f(s)  $\leftarrow$  MAX(f(n),g(s)+h(s))
    if all of n's successors have been generated then
      update n's f-cost and those of its ancestors if necessary
    if SUCCESSORS(n) all in memory then remove n from Queue
    if memory is full then
      delete shallowest, highest-f-cost node in Queue
      remove it from its parent's successor list
      insert its parent on Queue if necessary
    insert s in Queue
  end
```

# Simple Memory-bounded A\* (SMA\*)

SMA\* is a shortest path algorithm based on the A\* algorithm.

The advantage of SMA\* is that it uses a bounded memory, while the A\* algorithm might need exponential memory.

All other characteristics of SMA\* are inherited from A\*.

## How it works:

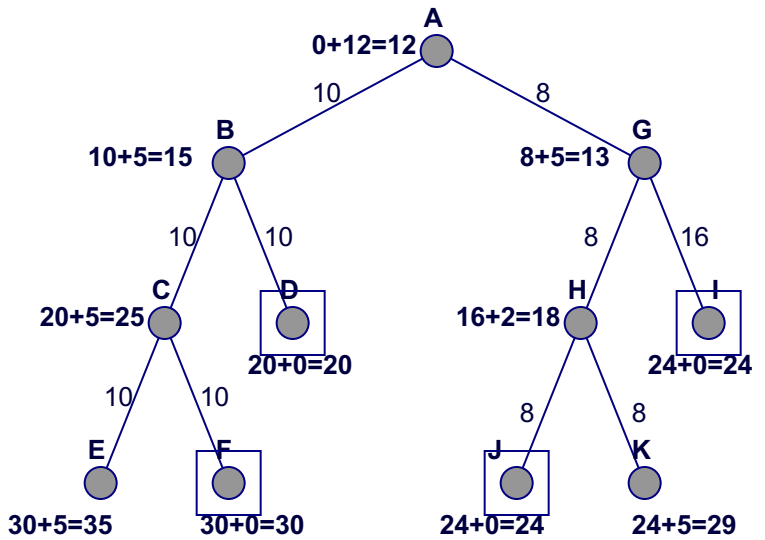
- Like A\*, it expands the best leaf until memory is full.
- Drops the worst leaf node- the one with the highest f-value.
- Like RBFS, SMA\* then backs up the value of the forgotten node to its parent.

# Simple Memory-bounded A\* (SMA\*)

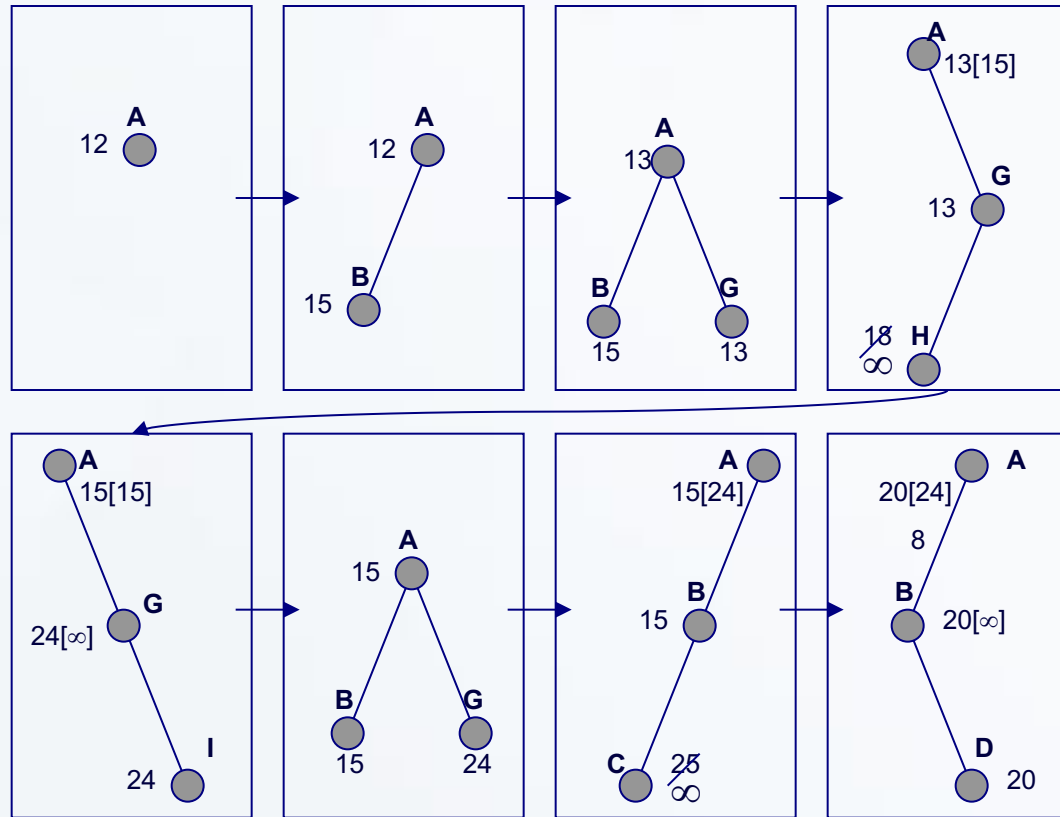
(Example with 3-node memory)

Search space

$f = g + h$       □ = goal



Progress of SMA\*. Each node is labeled with its *current*  $f$ -cost. Values in parentheses show the value of the best forgotten descendant.



$\infty$  is given to nodes that the path up to it uses all available memory.

Can tell when best solution found within memory constraint is optimal or not.

# The Algorithm proceeds as follows [3]

1. At each stage, one successor is added to the deepest lowest- $f$ -cost node that has some successors not currently in the tree. The left child B is added to the root A.
2. Now  $f(A)$  is still 12, so we add the right child G ( $f = 13$ ). Now that we have seen all the children of A, we can update its  $f$ -cost to the minimum of its children, that is, 13. The memory is now full.
3. G is now designated for expansion, but we must first drop a node to make room. We drop the shallowest highest- $f$ -cost leaf, that is, B. When we have done this, we note that A's best forgotten descendant has  $f = 15$ , as shown in parentheses. We then add H, with  $f(H) = 18$ . Unfortunately, H is not a goal node, but the path to H uses up all the available memory. Hence, there is no way to find a solution through H, so we set  $f(H) = \infty$ .
4. G is expanded again. We drop H, and add I, with  $f(I) = 24$ . Now we have seen both successors of G, with values of  $\infty$  and 24, so  $f(G)$  becomes 24.  $f(A)$  becomes 15, the minimum of 15 (forgotten successor value) and 24. Notice that I is a goal node, but it might not be the best solution because A's  $f$ -cost is only 15.
5. A is once again the most promising node, so B is generated for the second time. We have found that the path through G was not so great after all.
6. C, the first successor of B, is a nongoal node at the maximum depth, so  $f(C) = \infty$ .
7. To look at the second successor, D, we first drop C. Then  $f(D) = 20$ , and this value is inherited by B and A.
8. Now the deepest, lowest- $f$ -cost node is D. D is therefore selected, and because it is a goal node, the search terminates.

# SMA\* Properties [2]

- It works with a heuristic, just as A\*
- It is **complete** if the allowed memory is high enough to store the shallowest solution.
- It is **optimal** if the allowed memory is high enough to store the shallowest optimal solution, otherwise it will return the best solution that fits in the allowed memory.
- It avoids repeated states as long as the memory bound allows it
- It will **use all memory** available.
- **Enlarging the memory** bound of the algorithm will only speed up the calculation.
- When enough memory is available to contain the entire search tree, then calculation has an optimal speed



# Admissible Heuristics

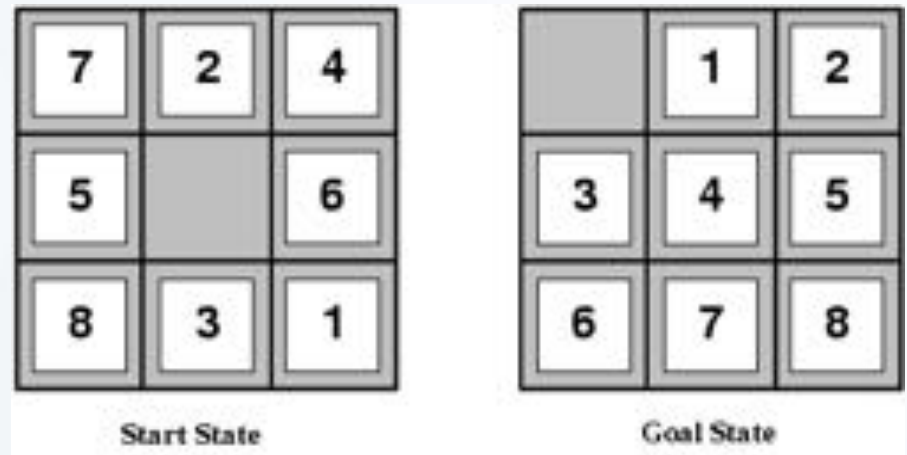
**How can you invent a good admissible heuristic function?  
E.g., for the 8-puzzle**

# 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)



$h_1(S) = ?$

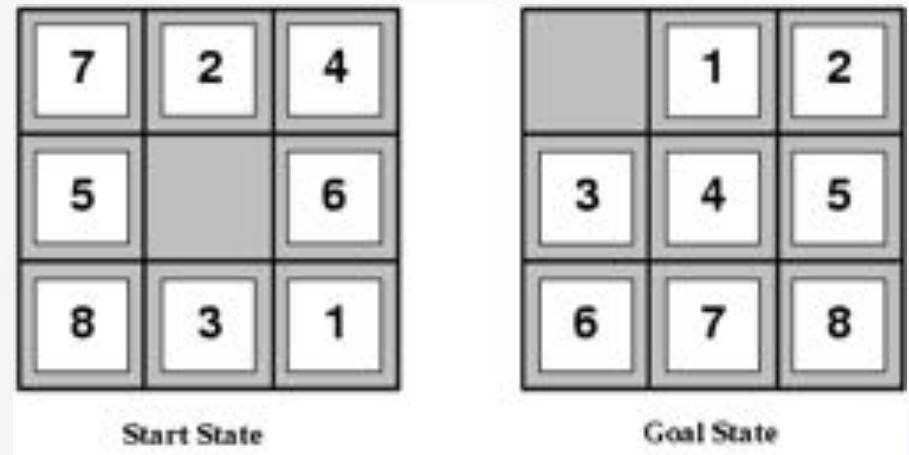
$h_2(S) = ?$

# 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)



$$h_1(S) = 8$$

$$h_2(S) = 3+1+2+2+2+3+3+2 = 18$$

# Dominance

If  $h_2(n) \geq h_1(n)$  for all  $n$ , and both are admissible.

then  $h_2$  **dominates**  $h_1$

$h_2$  is better for search: it is guaranteed to expand less nodes.

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

What to do If we have  $h_1 \dots h_m$ , but none dominates the other?

$$\rightarrow h(n) = \max\{h_1(n), \dots, h_m(n)\}$$

# 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.

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 near square**, then  $h_2(n)$  gives the shortest solution.

# Admissible Heuristics

## How can you invent a good admissible heuristic function?

- Try to relax the problem, from which an optimal solution can be found easily.
- Learn from experience.

➔ Can machines invent an admissible heuristic automatically?

# References

[1] S. Russell and P. Norvig: *Artificial Intelligence: A Modern Approach* Prentice Hall, 2003, *Second Edition*

[2] [http://en.wikipedia.org/wiki/SMA\\*](http://en.wikipedia.org/wiki/SMA*)

[3] Moonis Ali: Lecture Notes on Artificial Intelligence  
<http://cs.txstate.edu/~ma04/files/CS5346/SMA%20search.pdf>

[4] Max Welling: Lecture Notes on Artificial Intelligence  
<https://www.ics.uci.edu/~welling/teaching/ICS175winter12/A-starSearch.pdf>

[5] Kathleen McKeown: Lecture Notes on Artificial Intelligence  
<http://www.cs.columbia.edu/~kathy/cs4701/documents/InformedSearch-AR-print.ppt>

[6] Franz Kurfess: Lecture Notes on Artificial Intelligence  
<http://users.csc.calpoly.edu/~fkurfess/Courses/Artificial-Intelligence/F09/Slides/3-Search.ppt>