PROBLEM SOLVING & SEARCH STRATEGY
Part 3

Dr. Emad Natsheh

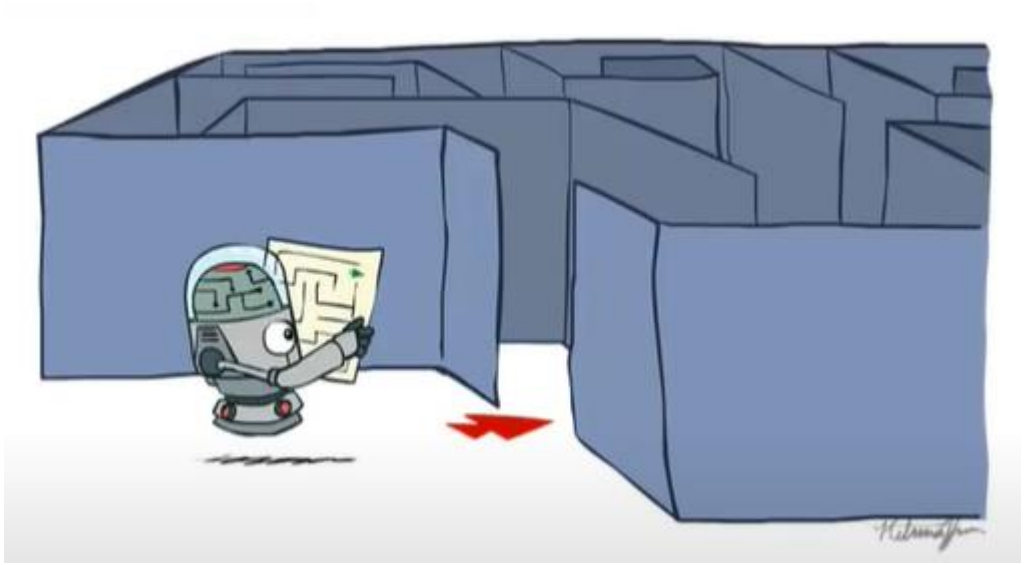
Informed (Heuristic) Search Technique

Informed search

- Heuristic
- Best First Search (Greedy search)
- A*

Recap: Search

- Search problem
  - State
  - Action or cost
  - Successor function
  - Start goal and end goal
- Represent problem as tree/graph
- Search algorithm
  - Choose node for expanding
    - Blind search
    - Inform search



Informed Search

- The assumption behind **blind search** is that we have no way of telling whether a particular search direction is likely to lead us to the goal or not

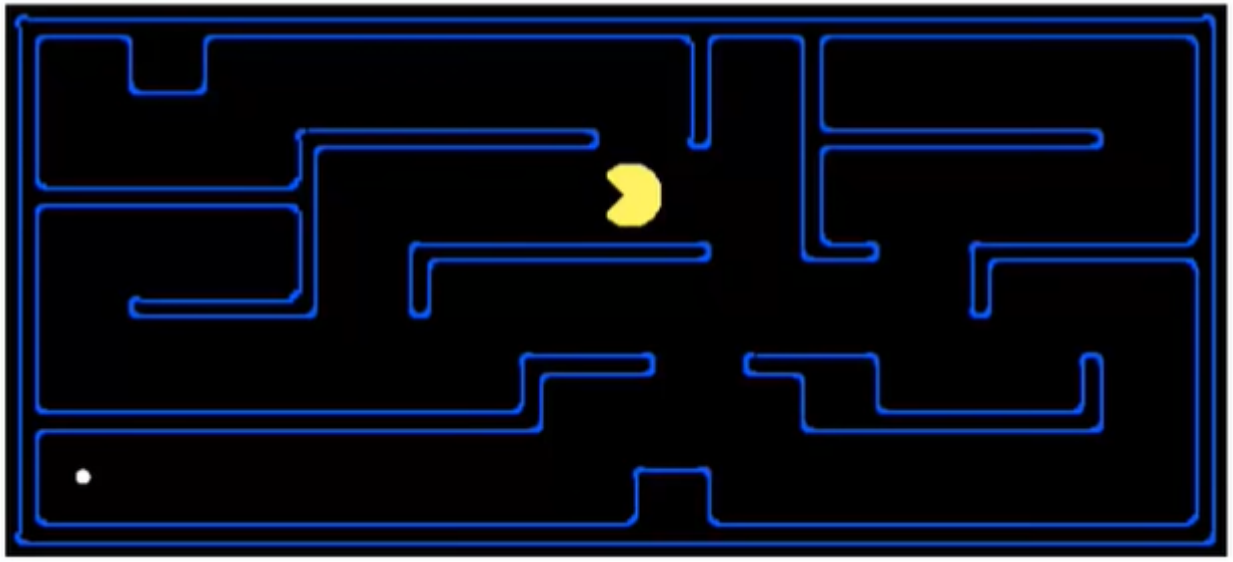
- The key idea behind **informed** or **heuristic** search algorithms is to exploit a task specific measure of goodness to try to either reach the goal more quickly or find a more desirable goal state.

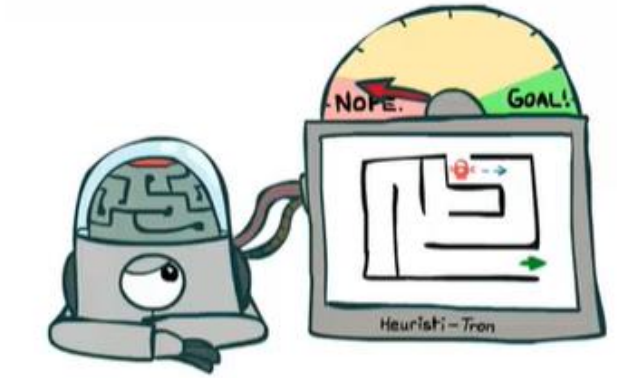
- Heuristic: From the Greek for “find”, “discover”.
  - Heuristics are criteria, methods, or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal”.

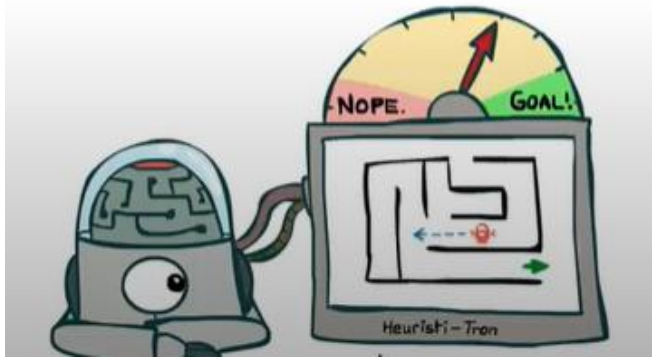
    Judea Pearl “ Heuristics” 1984

Heuristic Function h(n)

- **Heuristic h(n) is** a function that estimates how close a state is to a goal
- Designed for a particular search problem
- We assume that h(n) is non-negative,
  and that h(n)=0 if n is a goal node.

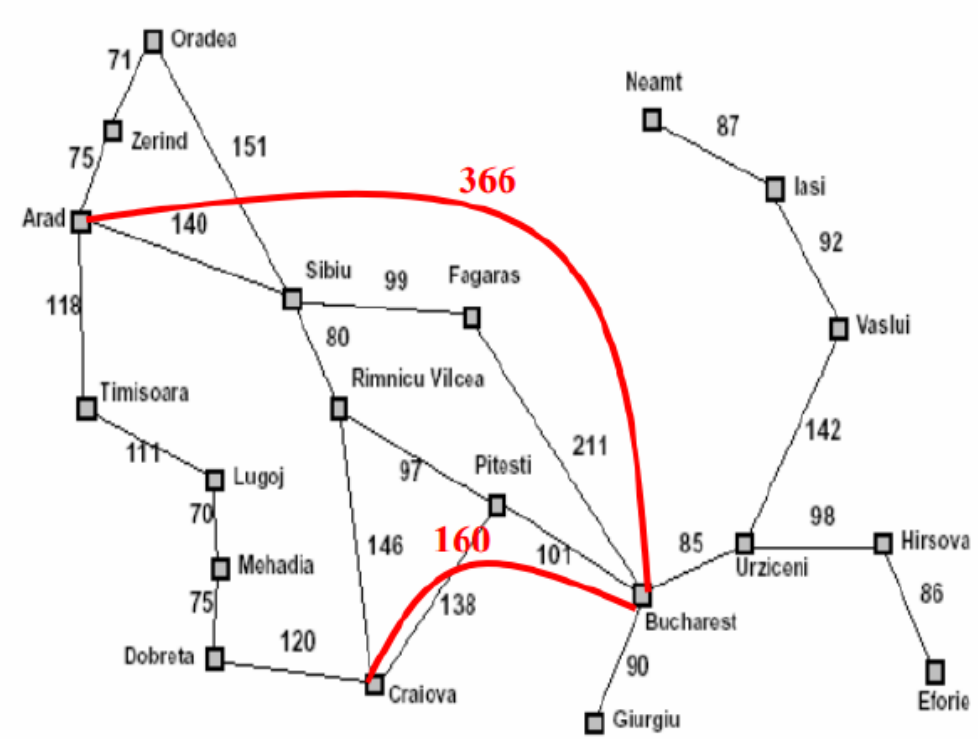






Heuristic Function (Example - Travel Planning)

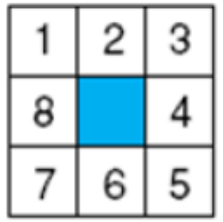
- h'(n) = straight-line distance between n and goal node



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

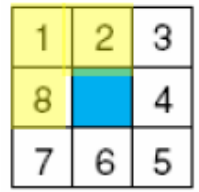
Heuristic Function (Example- 8Puzzle)

- Number of tiles out of places
  - Compared with the goal state
  - How many tiles are not in the right place?
- Sum of taxicab distances
  - For each tile, how far is it from the goal position? How many squares away? What is its distance?
- The objective is to minimize this heuristic value as you search. As it approaches 0, you are closer to the goal.

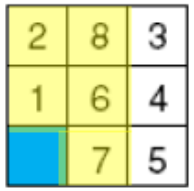
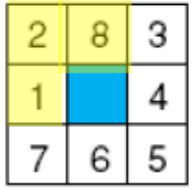
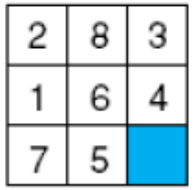


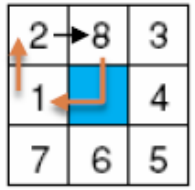
Goal

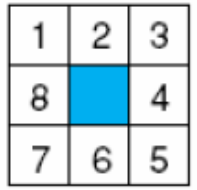
Heuristic Function (Example- 8Puzzle)



Goal

n	$h_1(n)$	$h_2(n)$
	5	6
	3	4
	5	6
	Tiles out of place	Sum of distances out of place



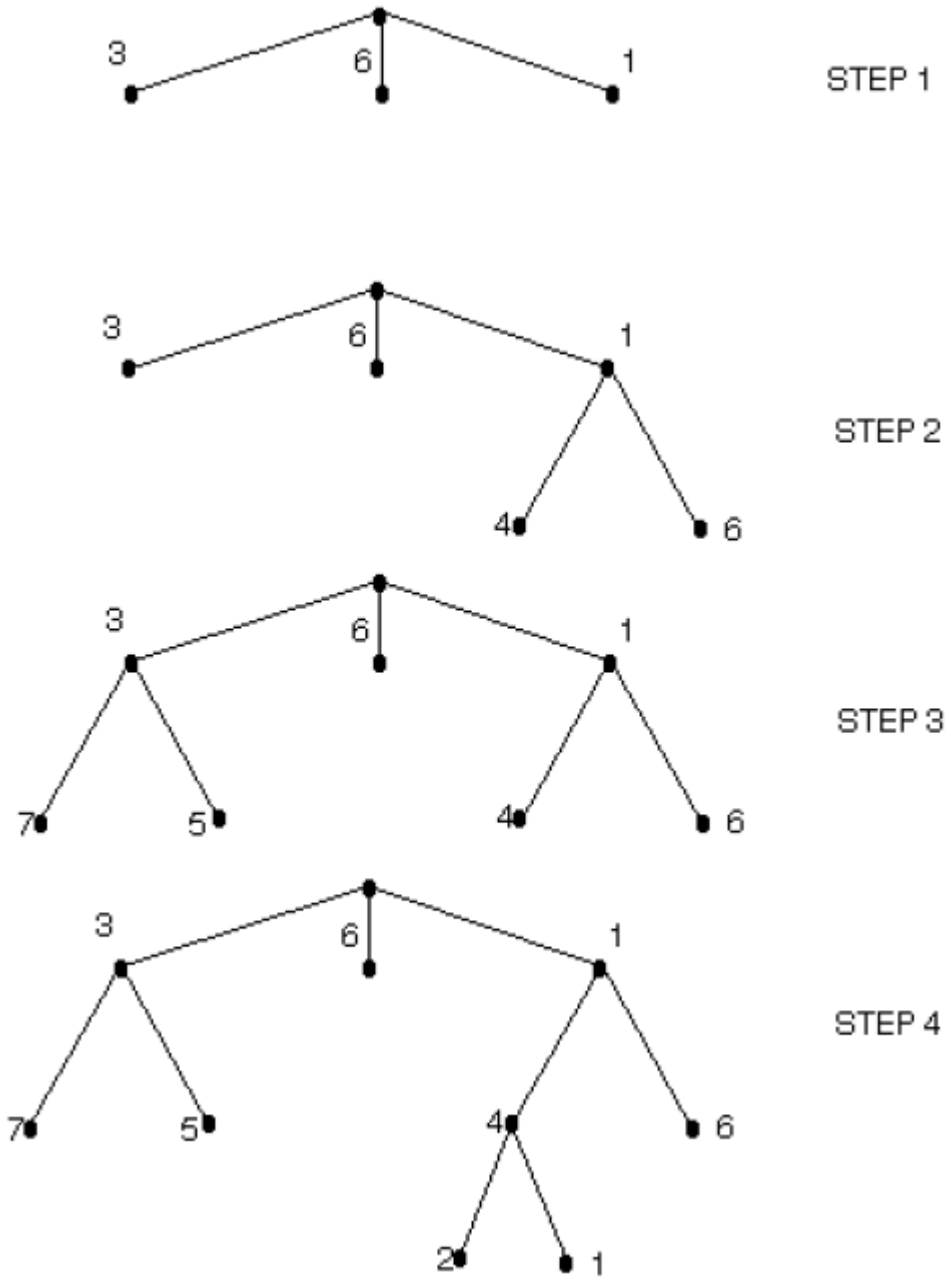


Goal

Best First Search (Greedy search)

Best First Search

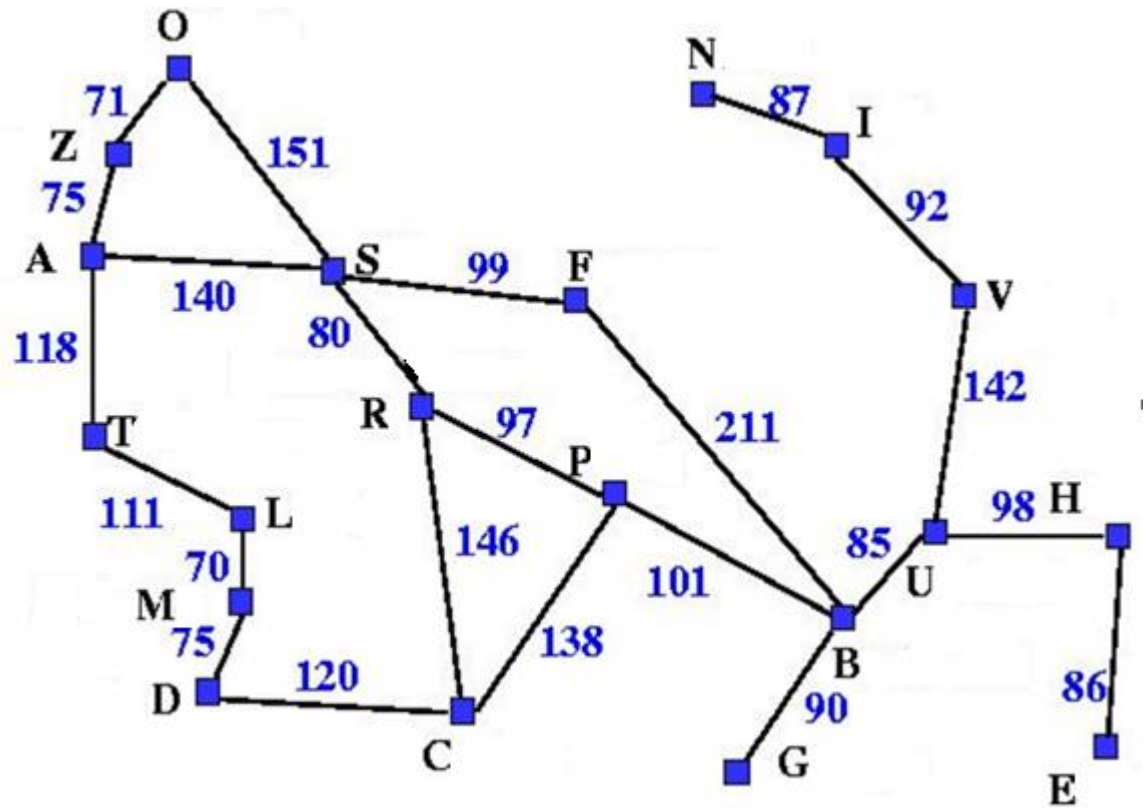
- Implementation: priority queue
- Very similar to UCS but sort the queue regarding to the h(n)
- Strategy: expand the node that you think is closest to the goal



```
Best-first search {

closed list = [ ]
open list = [start node]

    do {
        if open list is empty then{
            return no solution
        }
        n = heuristic best node
        if n == final node then {
            return path from start to goal node
        }
        foreach direct available node do{

            add current node to open list and calculate heuristic
            set n as his parent node

        }
        delete n from open list
        add n to closed list
    } while (open list is not empty)
}
```

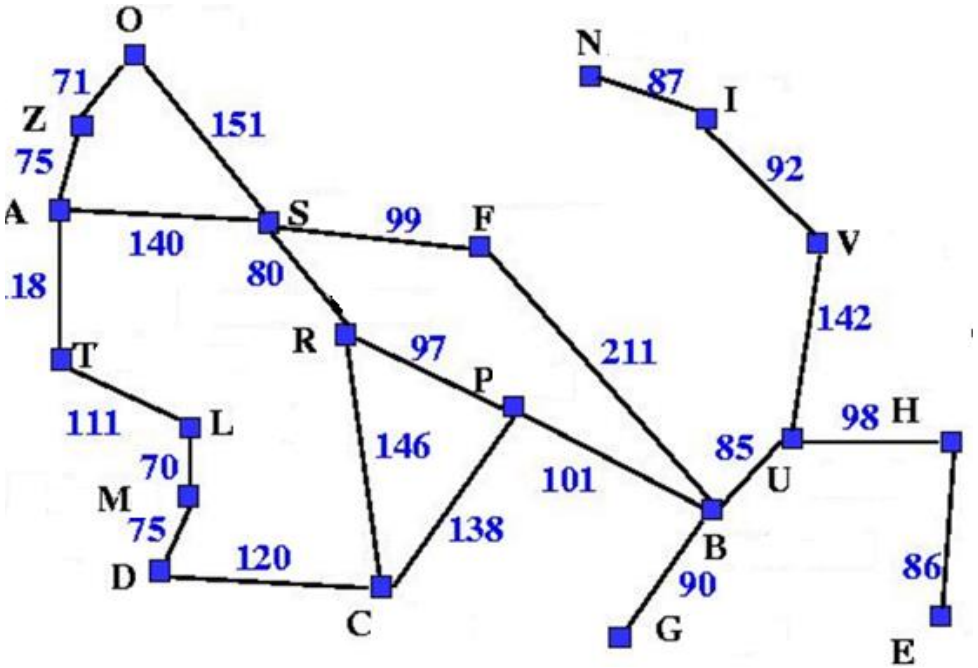
BFS Pseudocode

Best First Search (Example)

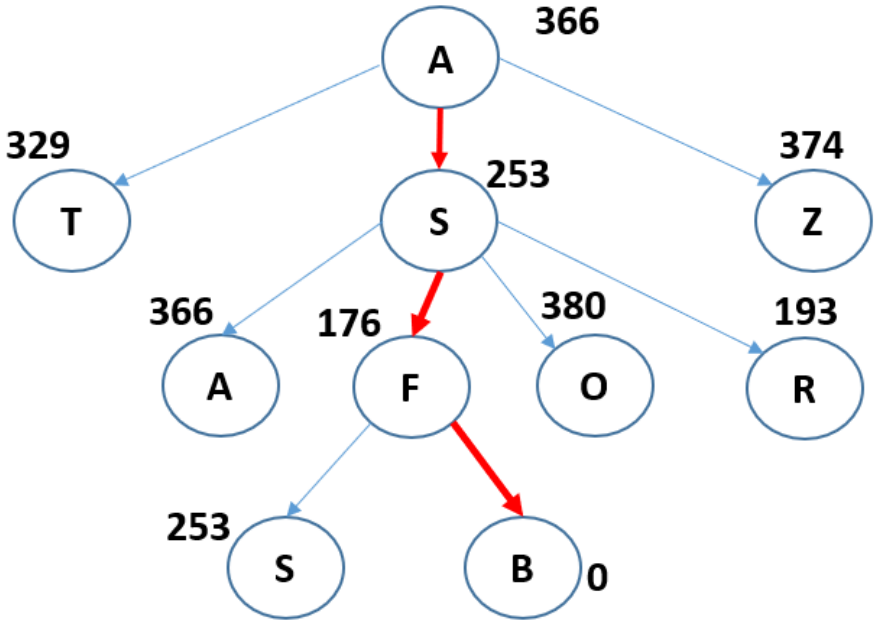


Town	h(n)
A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
R	193
S	253
T	329
U	80
V	199
Z	374

Town	h(n)
A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
R	193
S	253
T	329
U	80
V	199
Z	374



Best First Search (Example) cont.

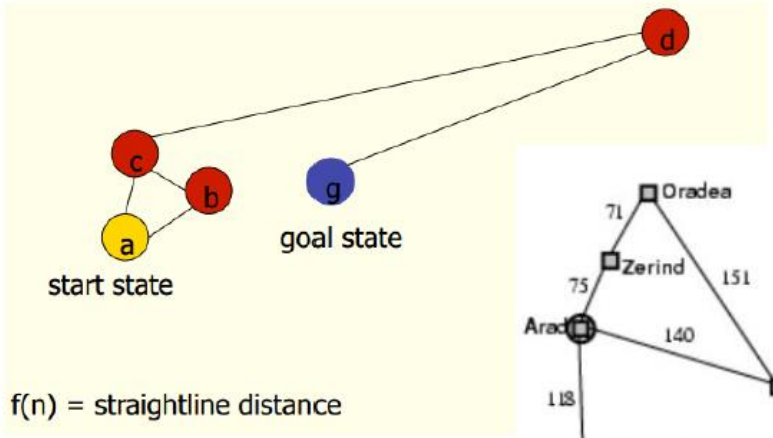


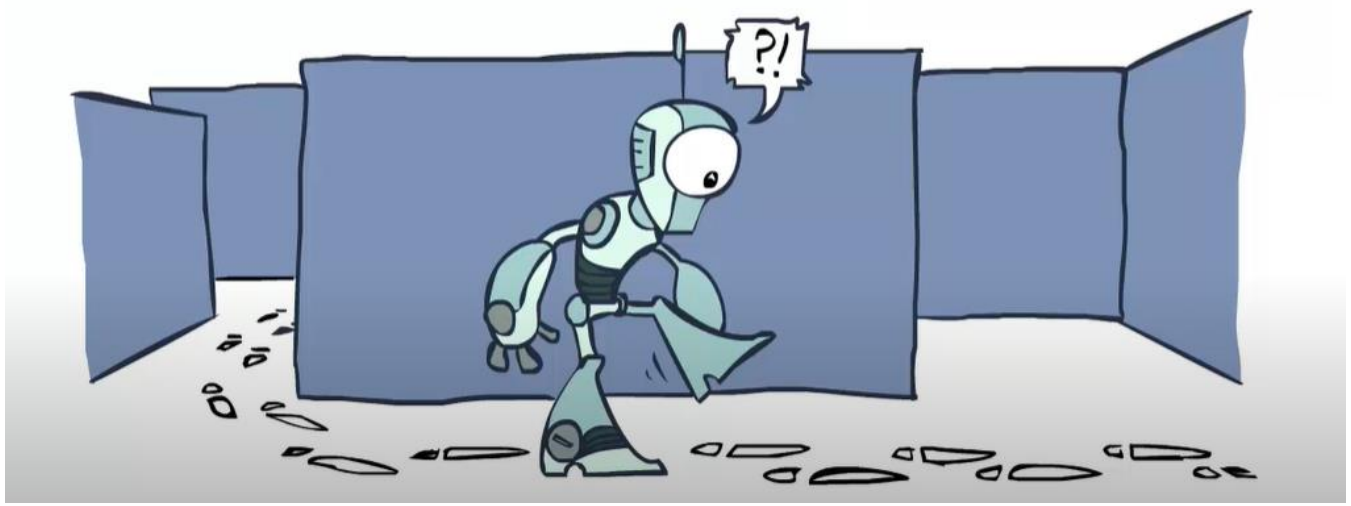
Path find cost =450

Optimal path cost = 418

Best First Search Analysis

- Complete?



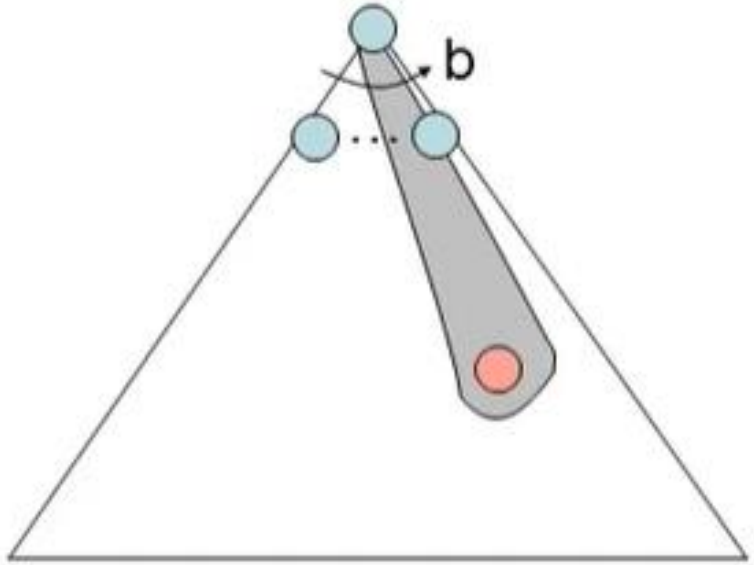


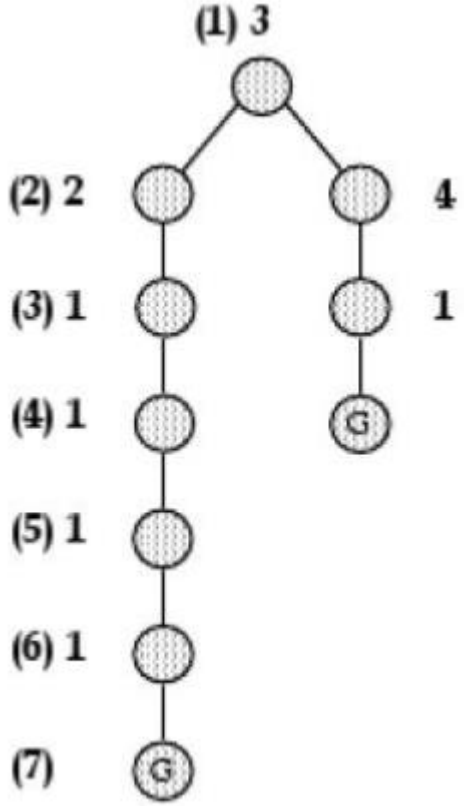
BFS Pseudocode 2

```
Best-first search {

closed list = [ ]
open list = [start node]

    do {
        if open list is empty then{
            return no solution
        }
        n = heuristic best node
        if n == final node then {
            return path from start to goal node
        }
        foreach direct available node do{
            if node not in open and not in closed list do {
                add node to open list
                set n as his parent node
        }
        delete n from open list
        add n to closed list
    } while (open list is not empty)
}
```

Best First Search Analysis

- Optimal?
- A Common case
  - Best first takes you straight to the (wrong) goal

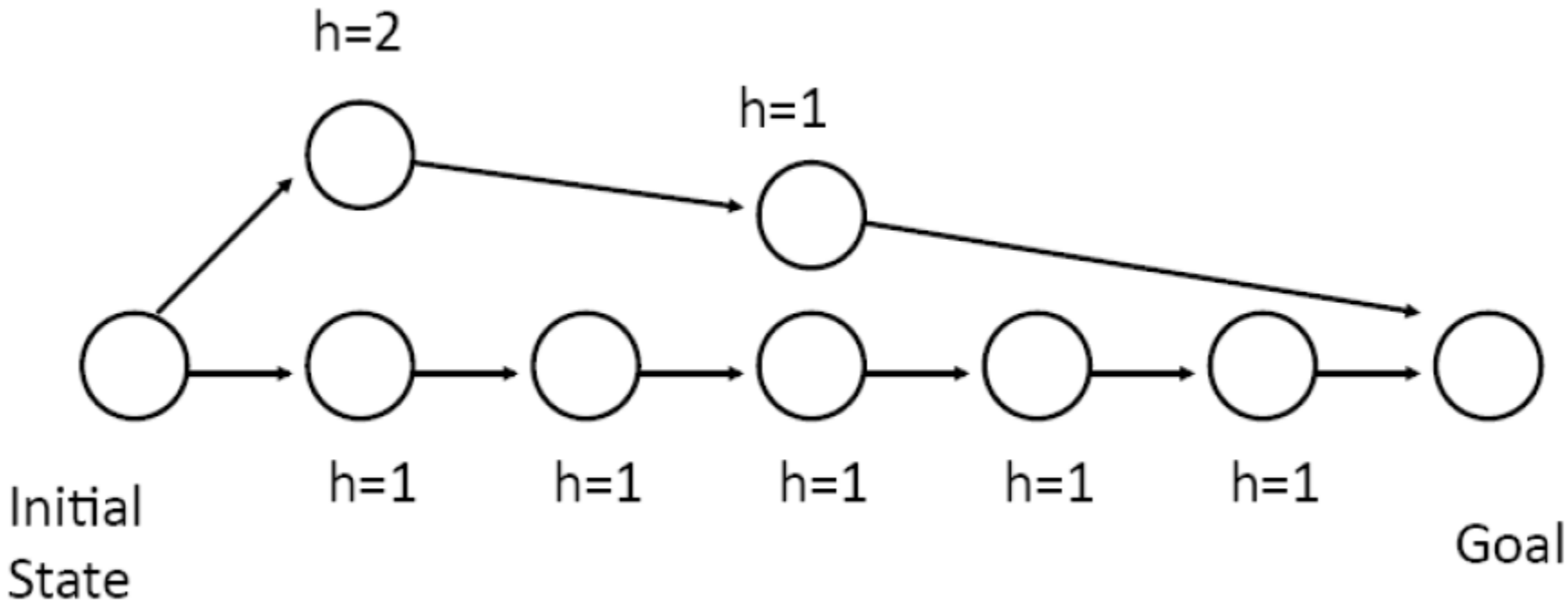




Best First Search Analysis

- Time Complexity → O(b^d)
- Space Complexity → O(b^d)
  - Where b is branching factor and d is depth of solution
  - But a good heuristic can give dramatic improvement

How can we fix the greedy problem?



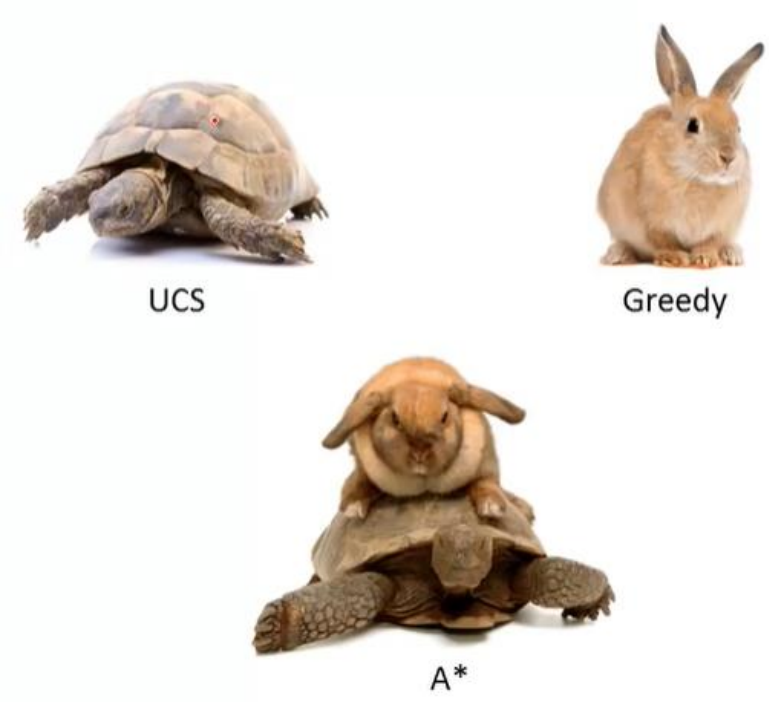
A* Search

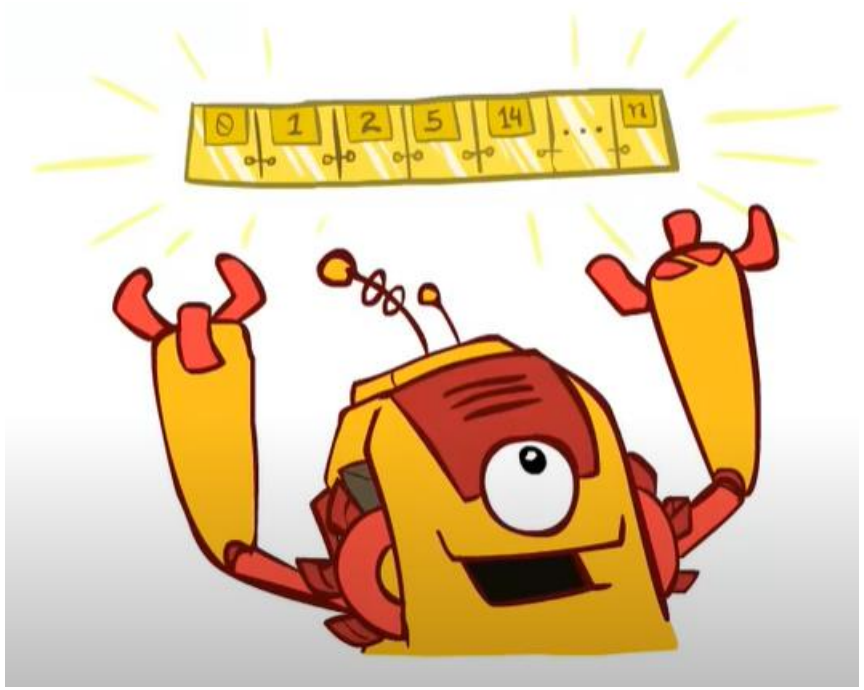
A* was created as part of the Shakey project

IEEE Milestones Award

A*

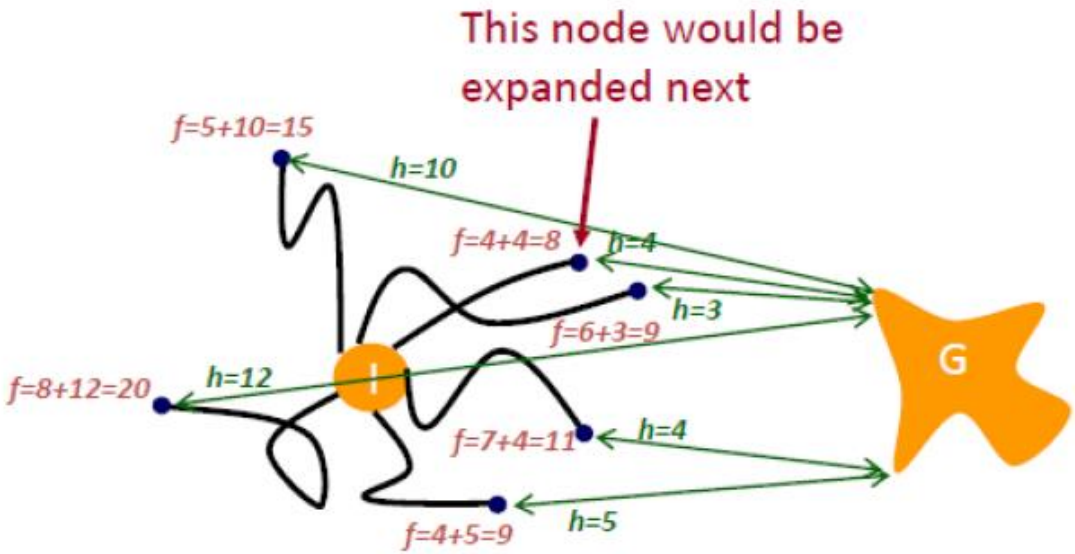
- Optimize form Best first search
- Implementation: priority queue (cost + heuristic)
- F(n) =g(n)+h(n)





A*

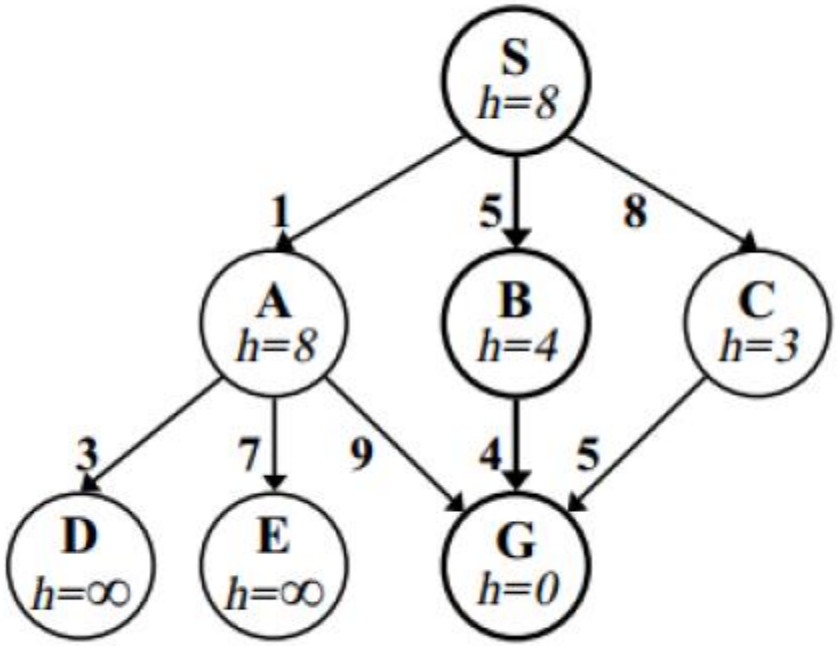
- f(*n*) = *g*(*n*) + *h*(*n*) : *Expand node that appears to* be on cheapest paths to the goal



```
A* search {

closed list = [ ]
open list = [start node]

    do {
        if open list is empty then {
            return no solution
        }
        n = heuristic best node
        if n == final node then {
            return path from start to goal node
        }
        foreach direct available node do{
            if current node not in open and not in closed list do {
                add current node to open list and calculate heuristic
                set n as his parent node
            }
            else{
                check if path from star node to current node is
                better;
                if it is better calculate heuristics and transfer
                current node from closed list to open list
                set n as his parrent node
            }
        delete n from open list
        add n to closed list
    } while (open list is not empty)

}
```

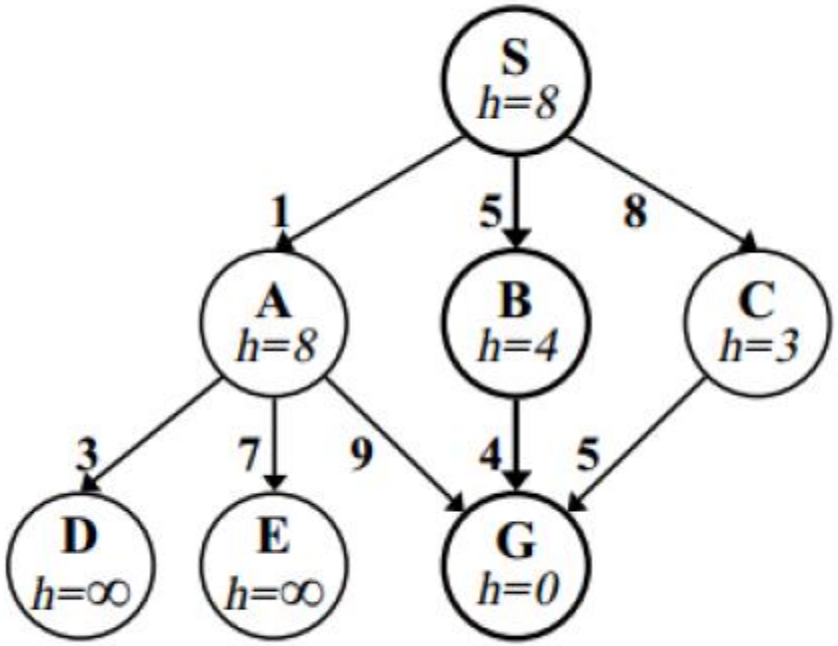
A* Pseudocode

A* (Example1-Table)

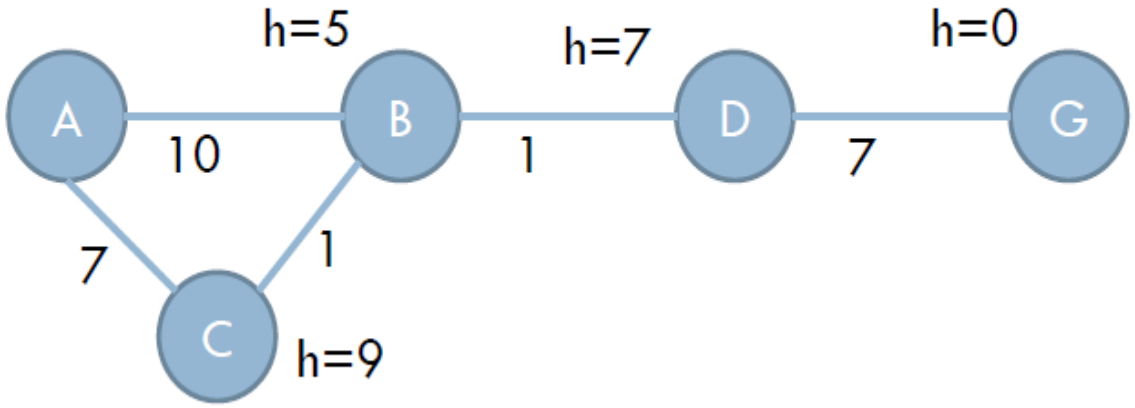


Close	Open
	S

A* (Example1-Tree)

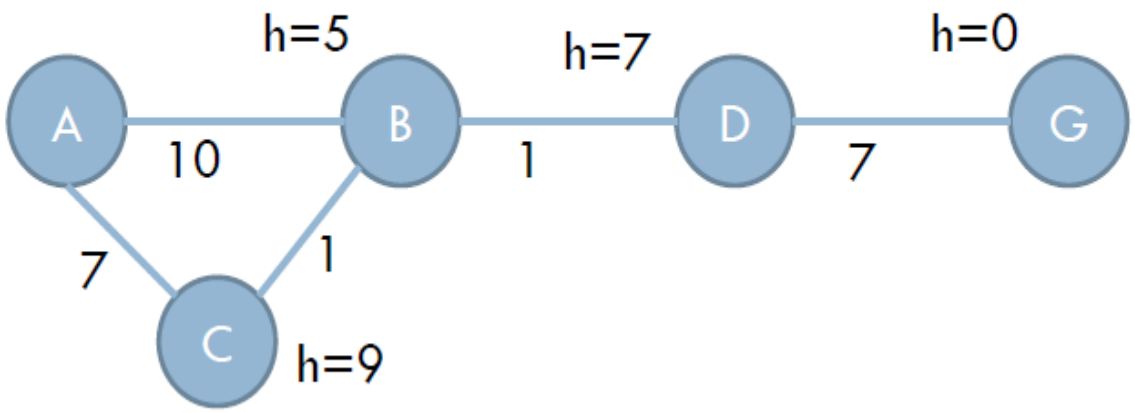


A* (Example2-Table)

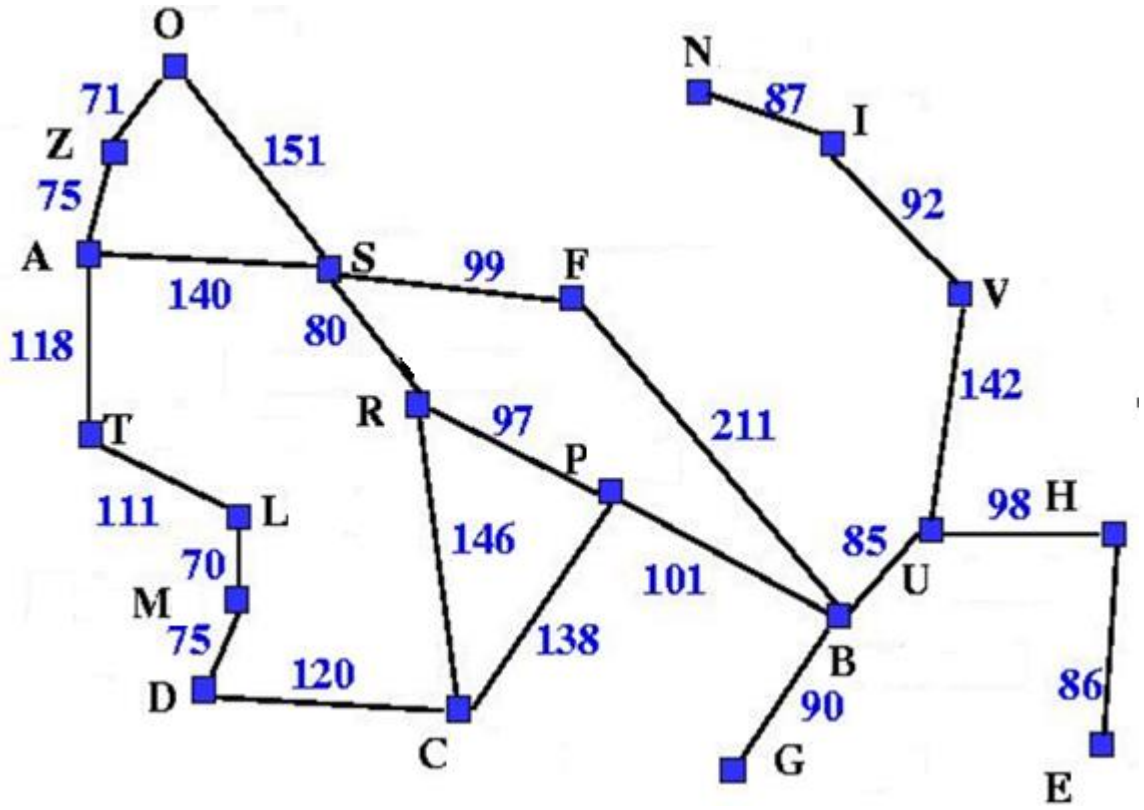


Close	Open
	A

A* (Example2-Tree)

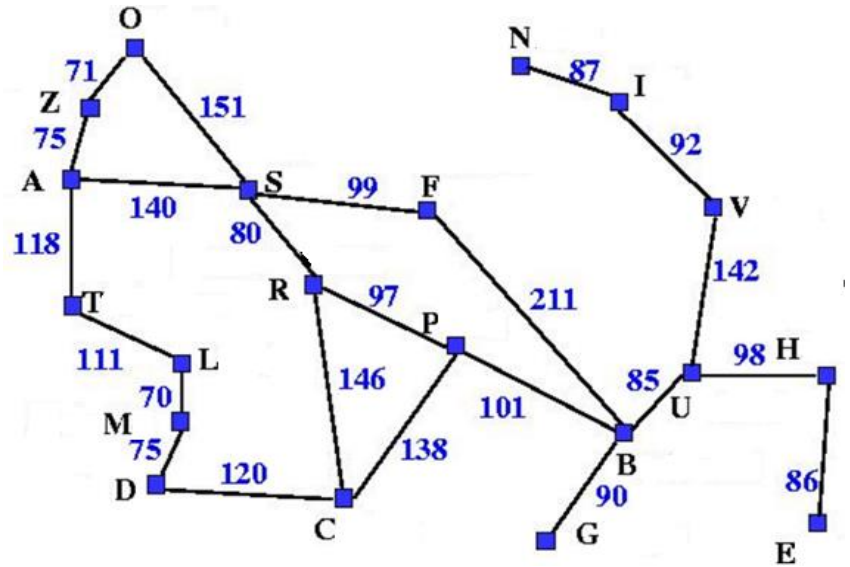


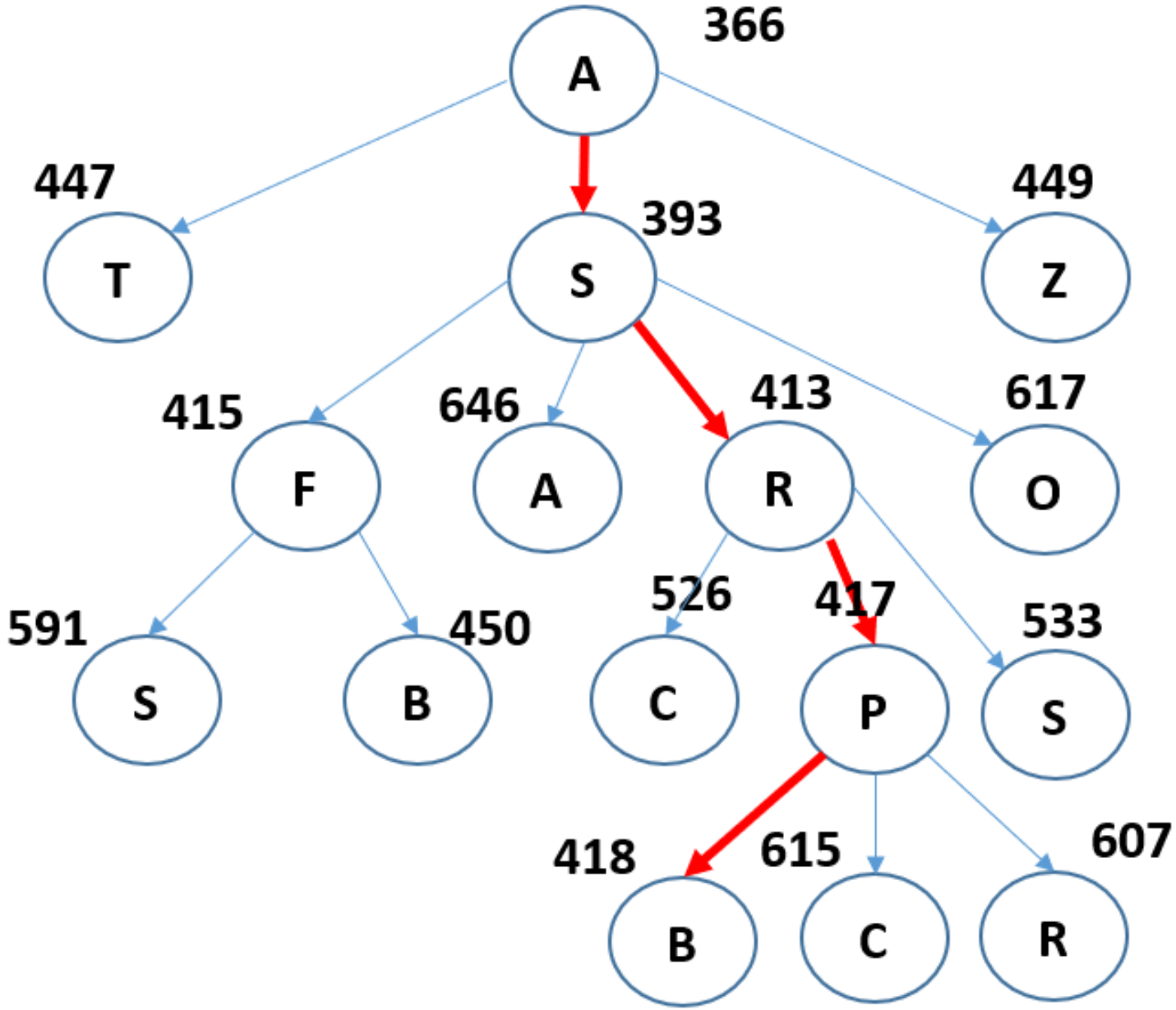
A* (Path Finding Example)



Town	h(n)
A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
R	193
S	253
T	329
U	80
V	199
Z	374

Town	h(n)
A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
R	193
S	253
T	329
U	80
V	199
Z	374




366
A
447
T
393
S
449
Z
415
F
646
A
413
R
617
O
591
S
450
B
526
C
417
P
533
S
418
B
615
C
607
R

A* (8 puzzle Example)

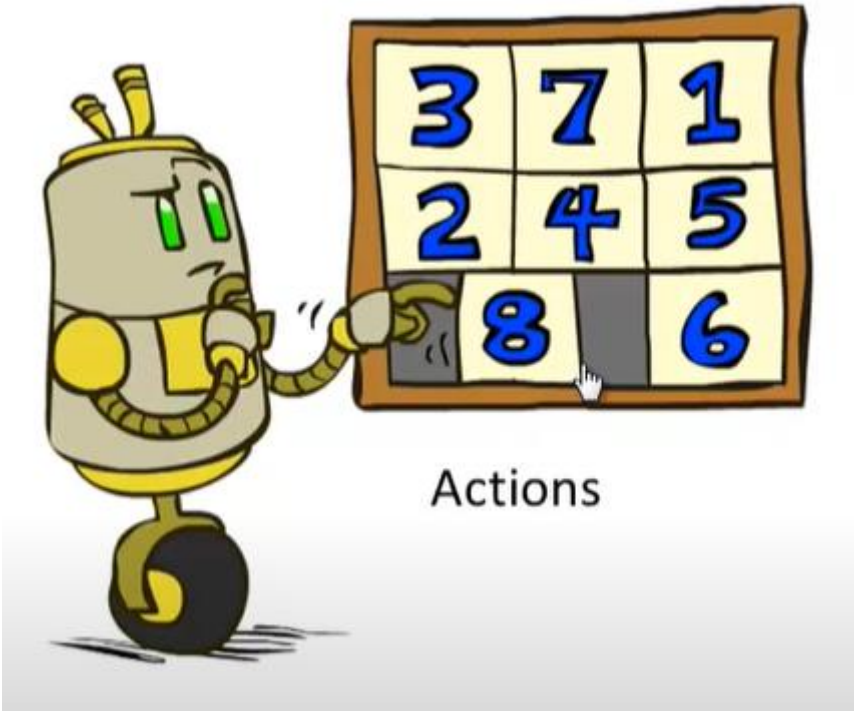
Start state

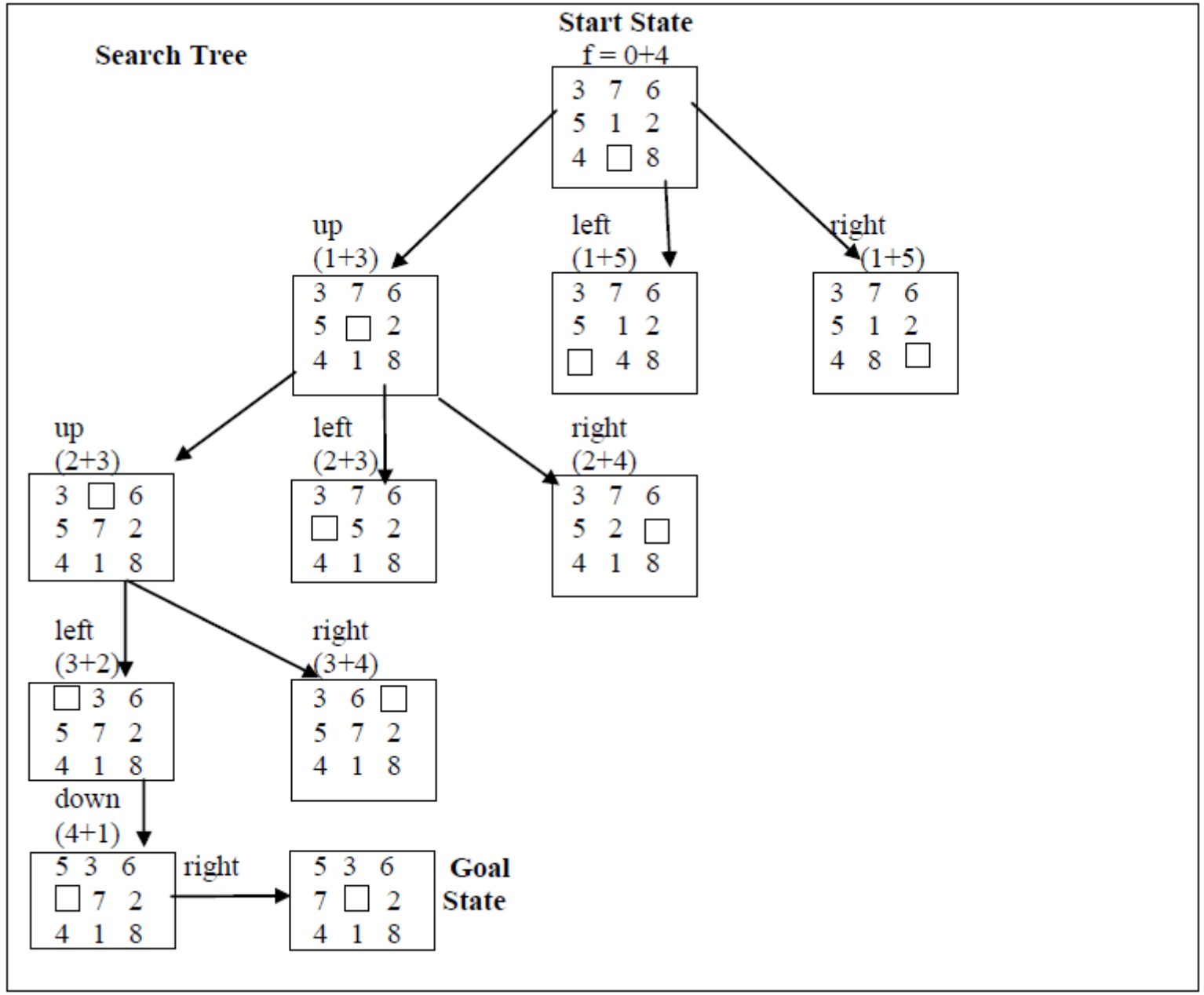
3	7	6
5	1	2
4	□	8

Goal state

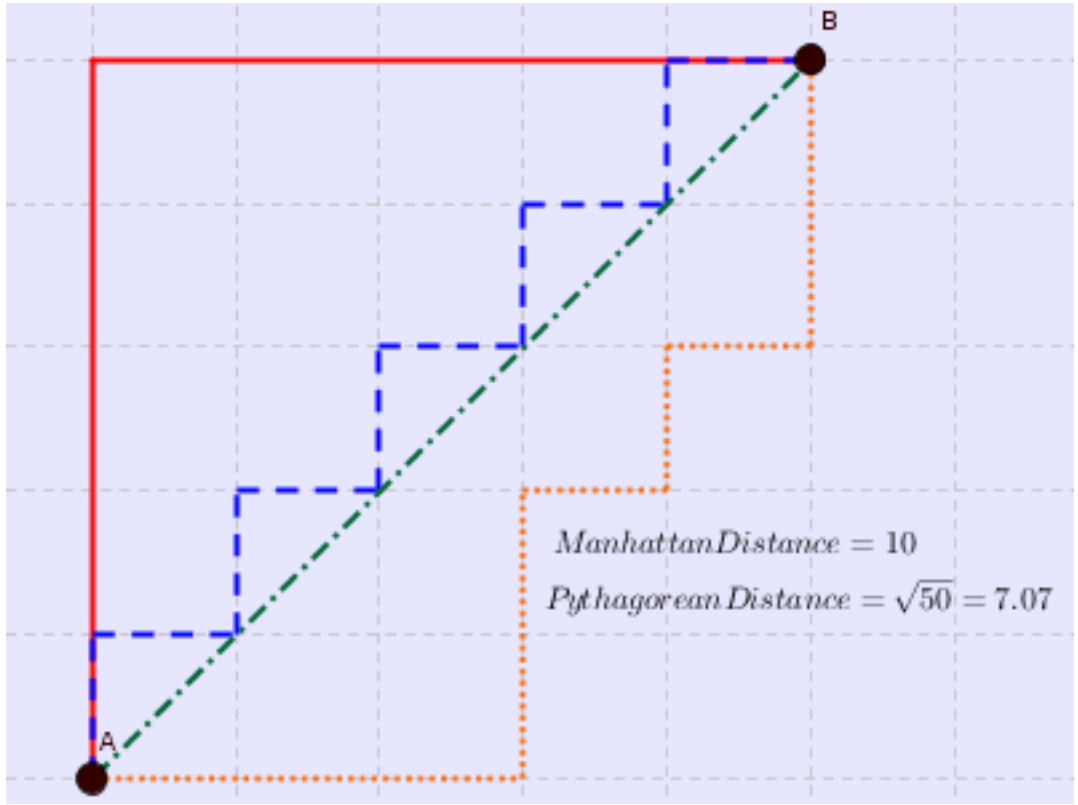
5	3	6
7	□	2
4	1	8

- The choice of evaluation function critically determines search results.
- Consider Evaluation function **f (X) = g (X) + h(X)**
  - h (X)= the number of tiles not in their goal position in a given state X
  - g(X)=depth of node X in the search tree
- For Initial node
  - **f(s)** = 4
- Apply A* algorithm to solve it.




Search Tree
Start State
f = 0+4
3 7 6
5 1 2
4 □ 8
up
(1+3)
3 7 6
5 □ 2
4 1 8
left
(1+5)
3 7 6
5 1 2
□ 4 8
right
(1+5)
3 7 6
5 1 2
4 8 □
up
(2+3)
3 □ 6
5 7 2
4 1 8
left
(2+3)
3 7 6
□ 5 2
4 1 8
right
(2+4)
3 7 6
5 2 □
4 1 8
left
(3+2)
□ 3 6
5 7 2
4 1 8
right
(3+4)
3 6 □
5 7 2
4 1 8
down
(4+1)
5 3 6
□ 7 2
4 1 8
right
5 3 6
7 □ 2
4 1 8
Goal
State

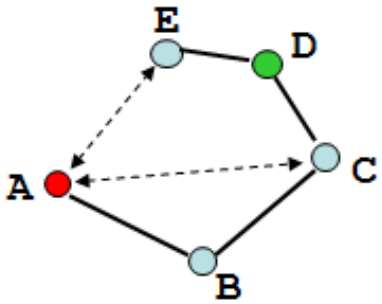
Manhattan Distance



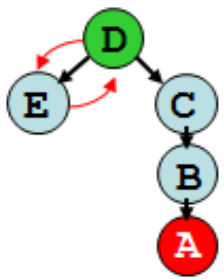
$$|x1 - x2| + |y1 - y2|$$

Straight-line distance

- The distance between two locations on a map can be known without knowing how they are linked by roads (i.e. the absolute path to the goal).



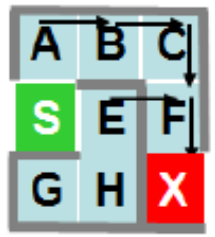
Problem Space



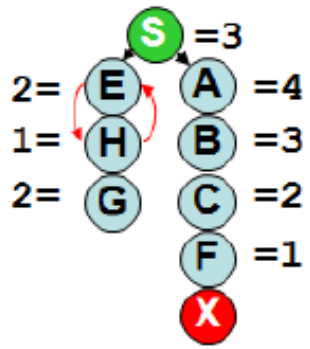
Search Tree

Manhattan Distance

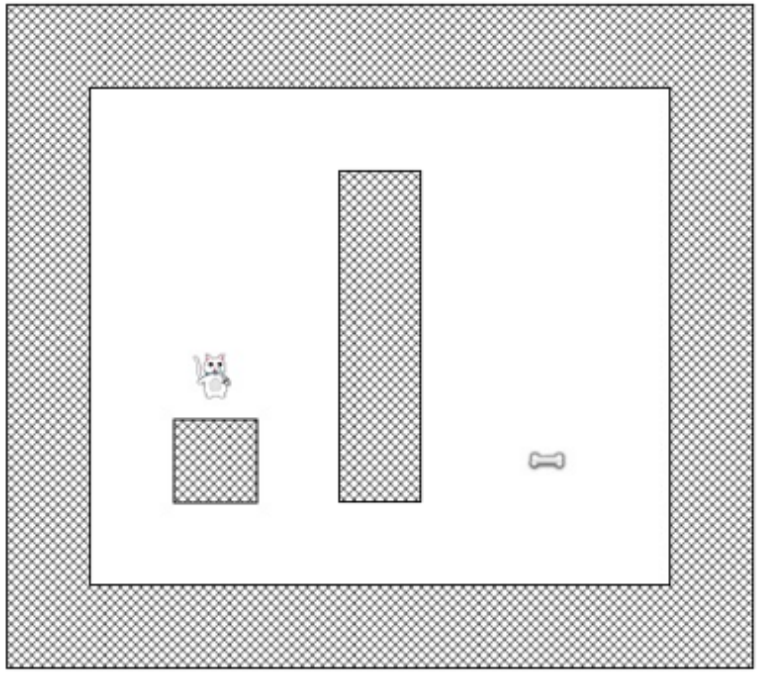
- The smallest number of vertical and horizontal moves needed to get to the goal (ignoring obstacles).

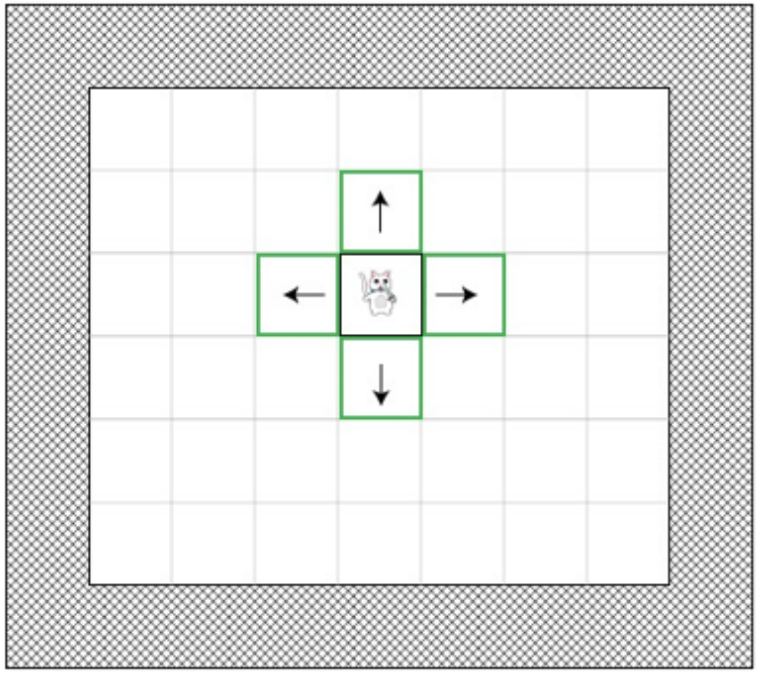


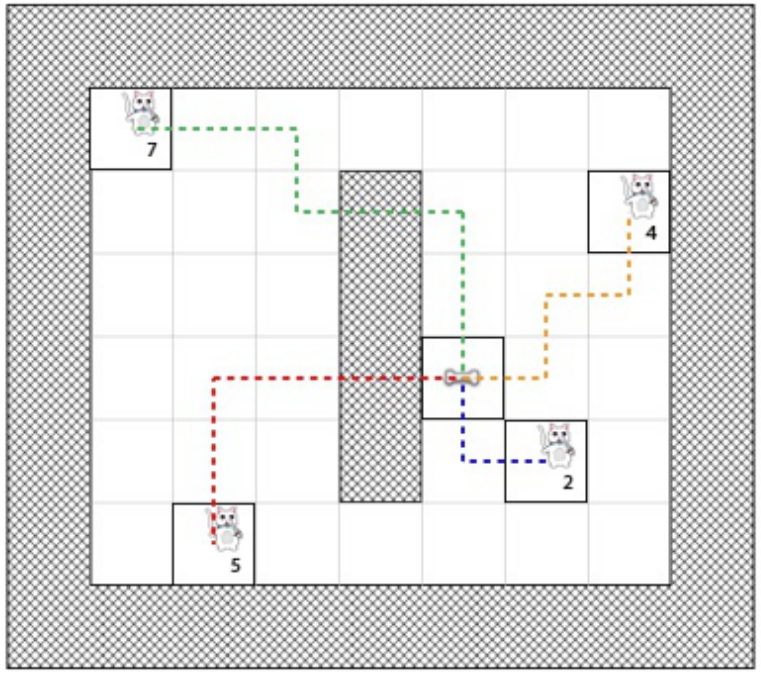
Manhattan Distance
A = 4
E = 2

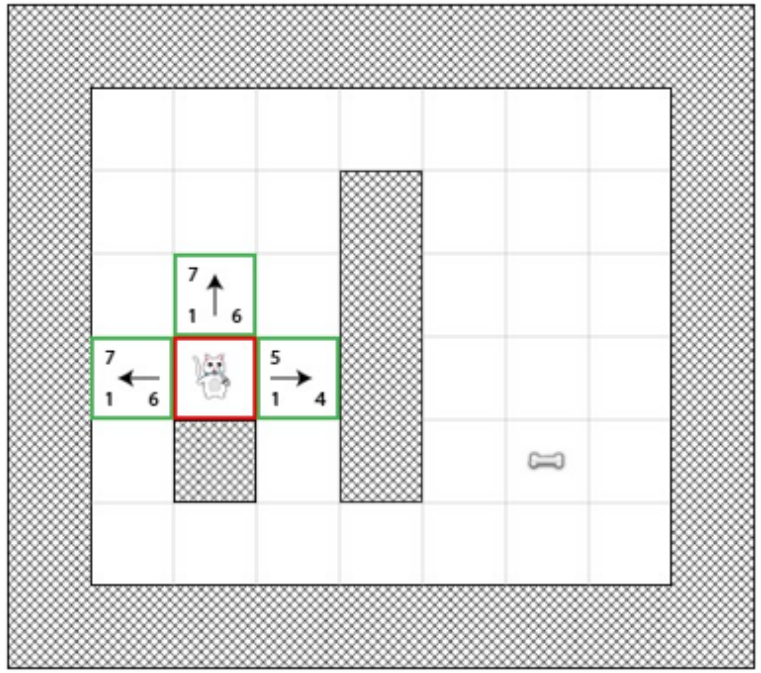


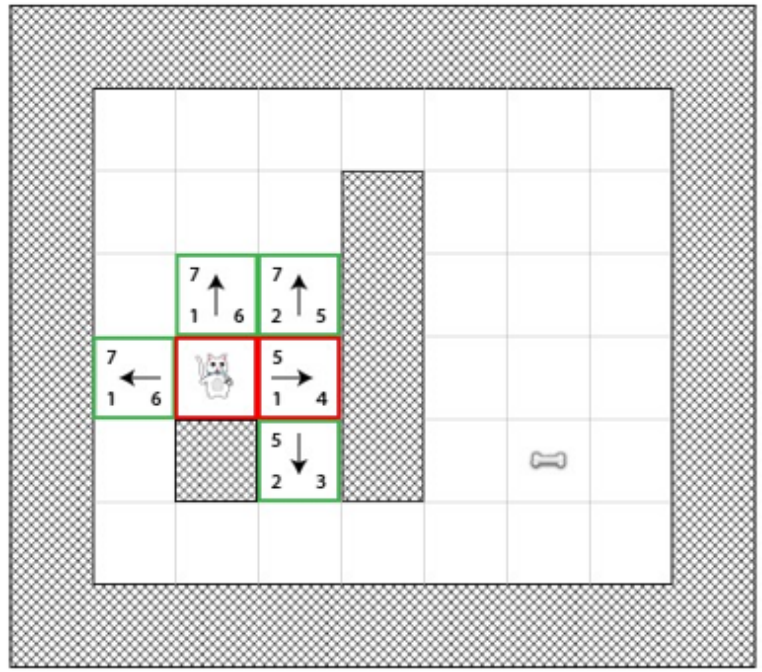
Maze World

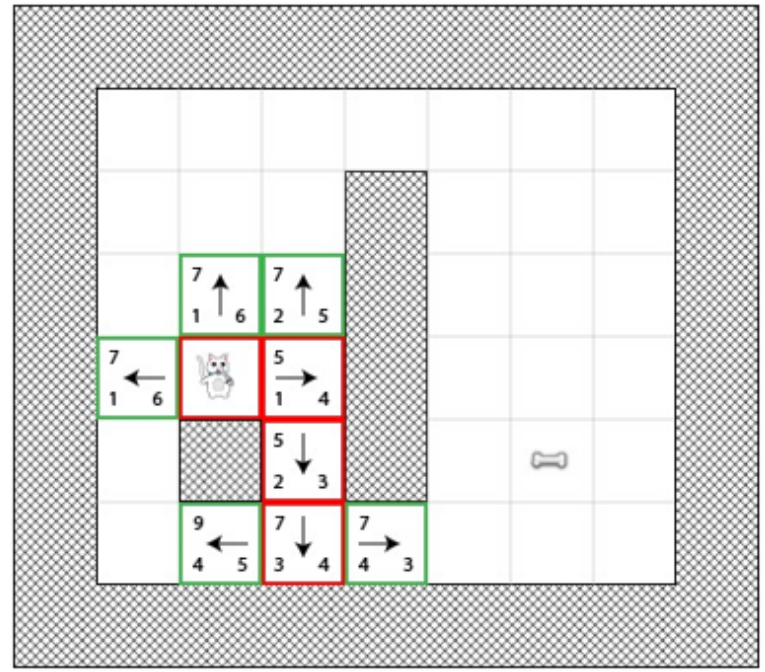


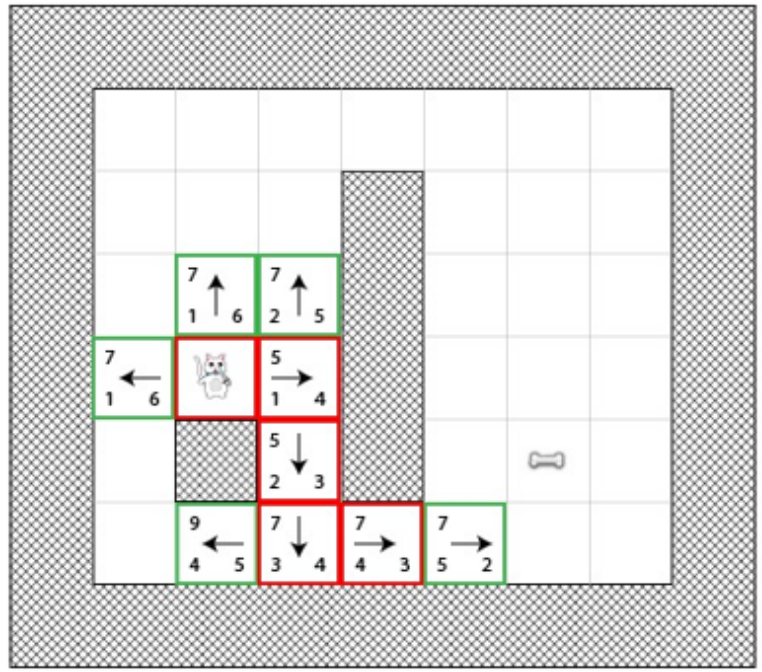


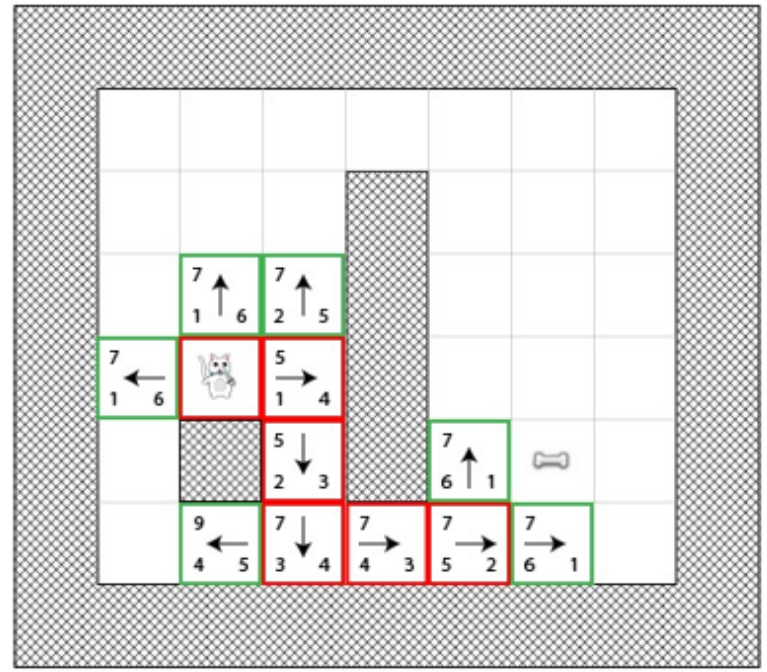





7 1 6
7 2 5
7 1 6
5 1 4
5 2 3


7 1 6
7 2 5
7 1 6
5 1 4
5 2 3
9 4 5
7 3 4
7 4 3

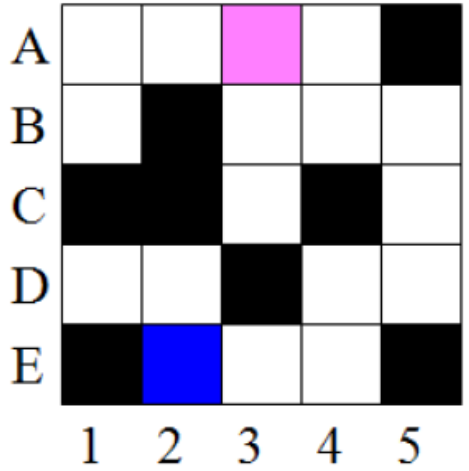

7 1 6
7 2 5
7 1 6
5 1 4
5 2 3
9 4 5
7 3 4
7 4 3
7 5 2


7 1 6
7 2 5
7 1 6
5 1 4
5 2 3
7 6 1
9 4 5
7 3 4
7 4 3
7 5 2
7 6 1

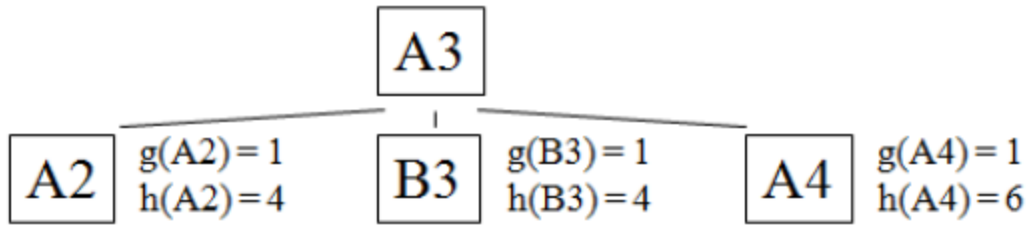
A* (Maze Example)

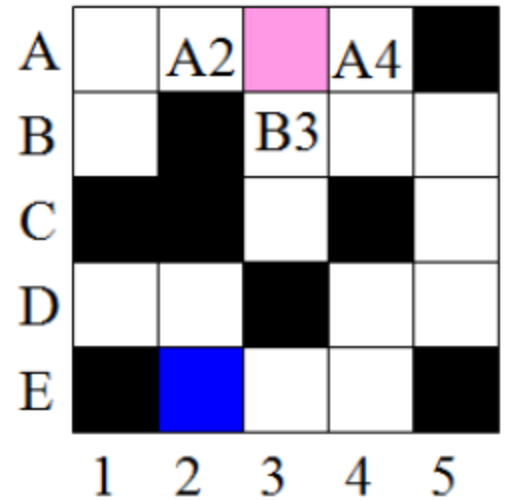
- **Problem**: To get from square **A3** to square **E2**, one step at a time, avoiding obstacles (black squares).

- **Operators**: (in order)
  - **go_left**(n)
  - **go_down**(n)
  - **go_right**(n)
- each operator costs 1.



- **Heuristic**: Manhattan distance

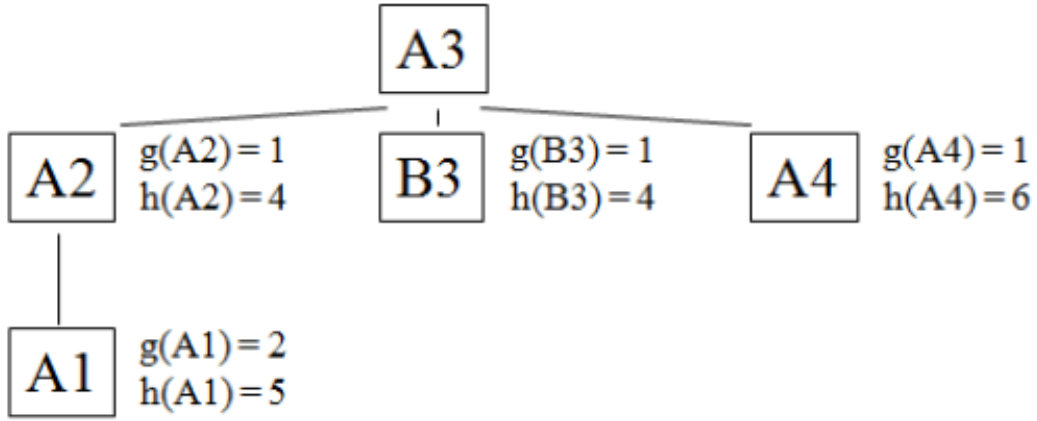


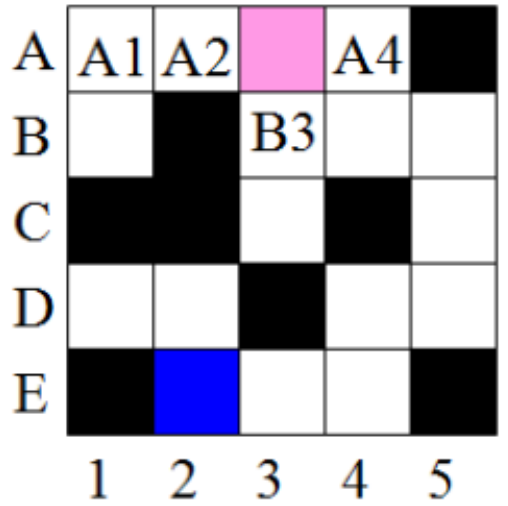


Operators: (in order)

- `go_left`(n)
- `go_down`(n)
- `go_right`(n)

each operator costs 1.

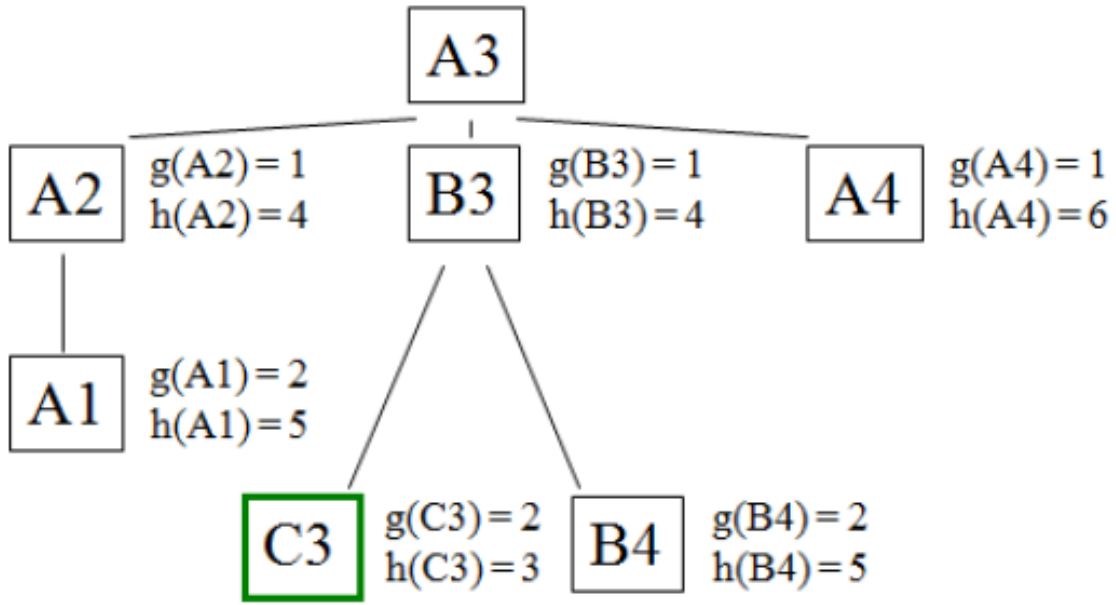


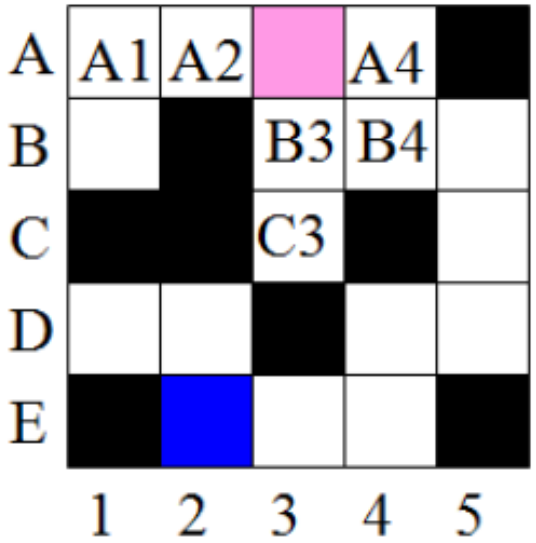


Operators: (in order)

- `go_left`(n)
- `go_down`(n)
- `go_right`(n)

each operator costs 1.



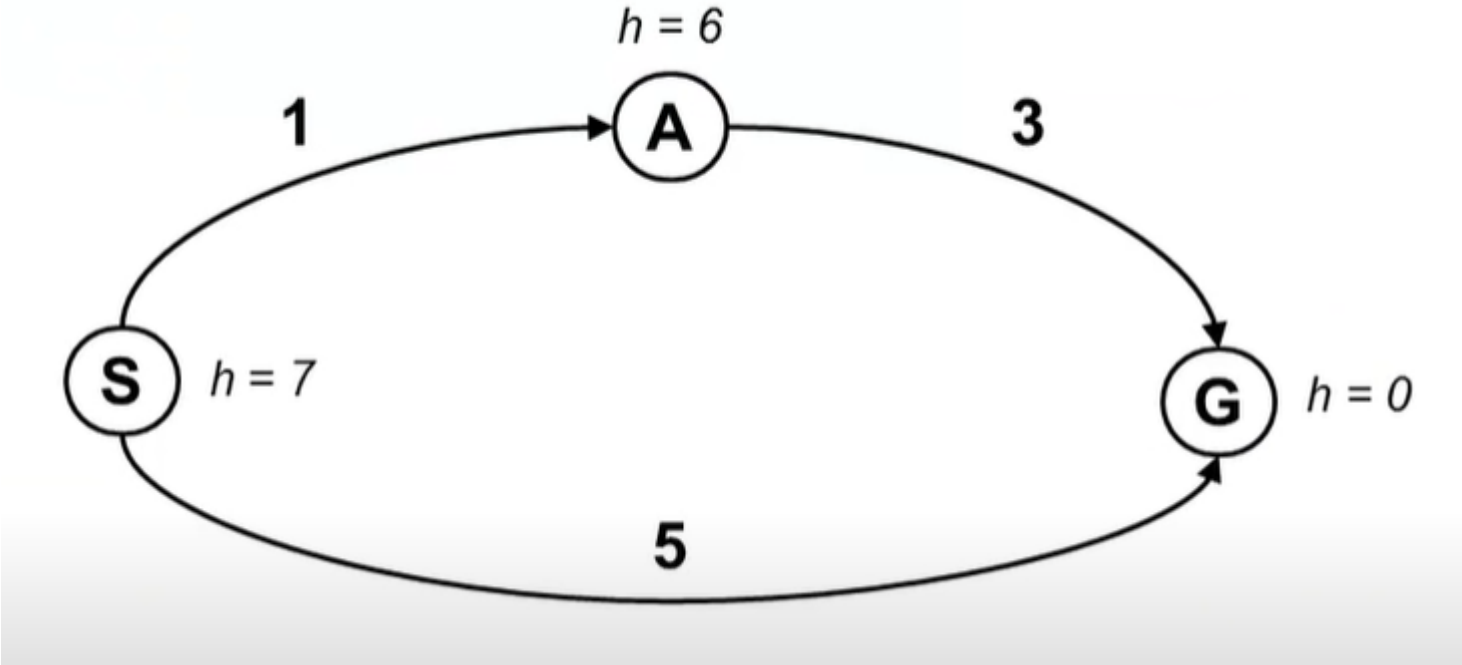


Operators: (in order)

- `go_left`(n)
- `go_down`(n)
- `go_right`(n)

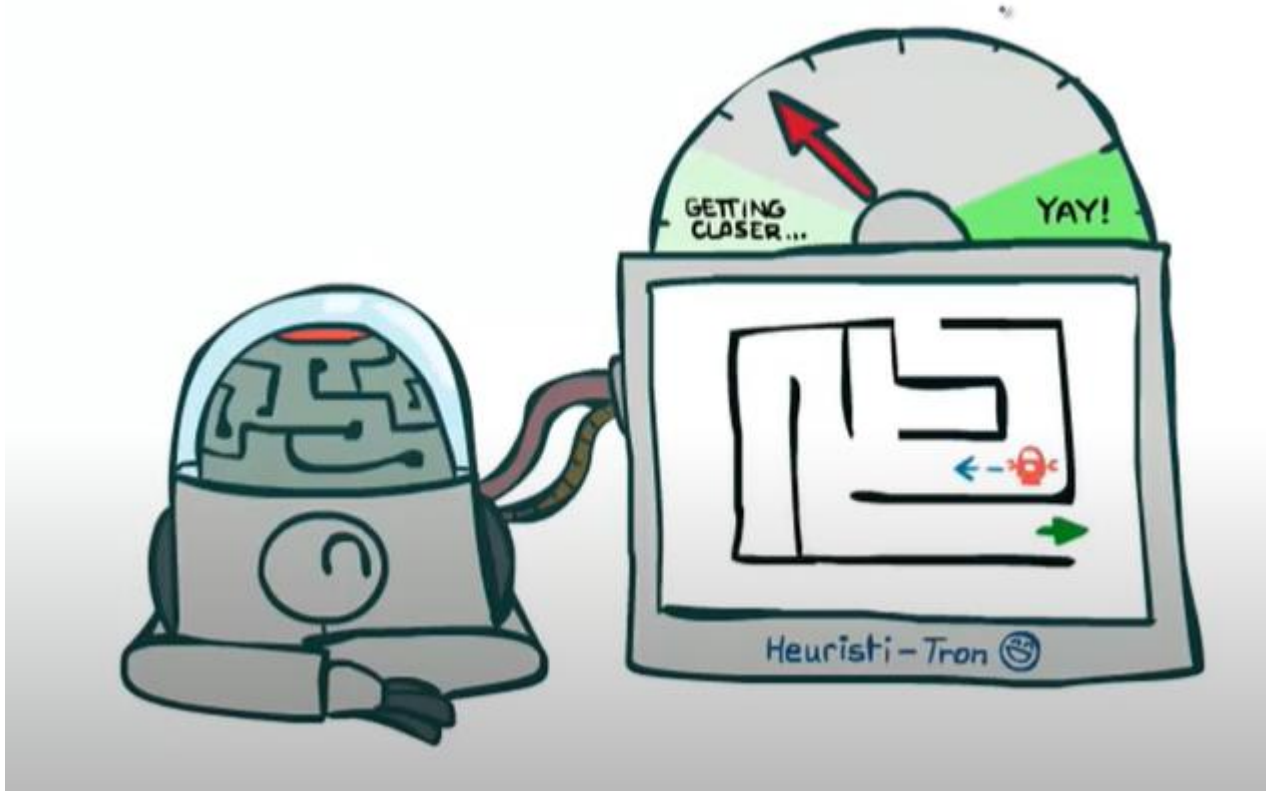
each operator costs 1.

Is A* Optimal?

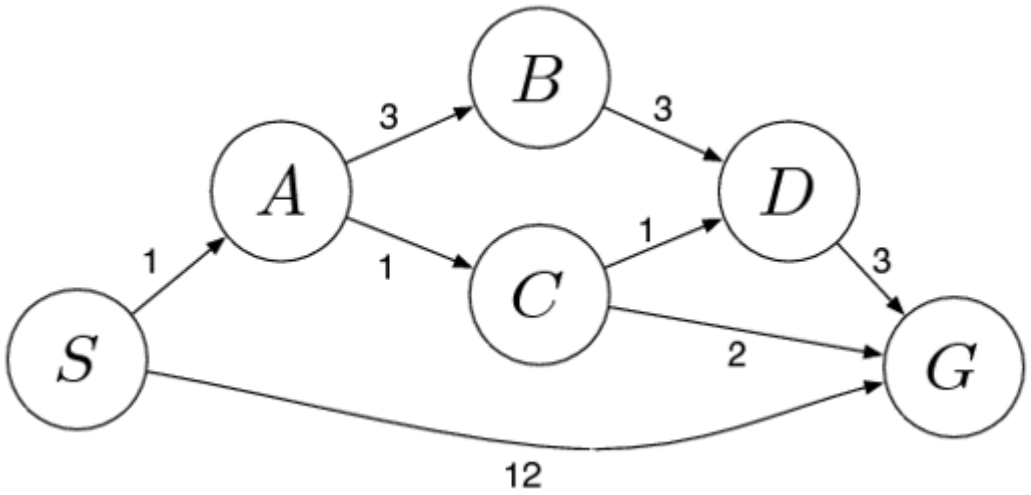


- What went wrong?

Admissible Heuristic



Admissible Heuristic (Example)

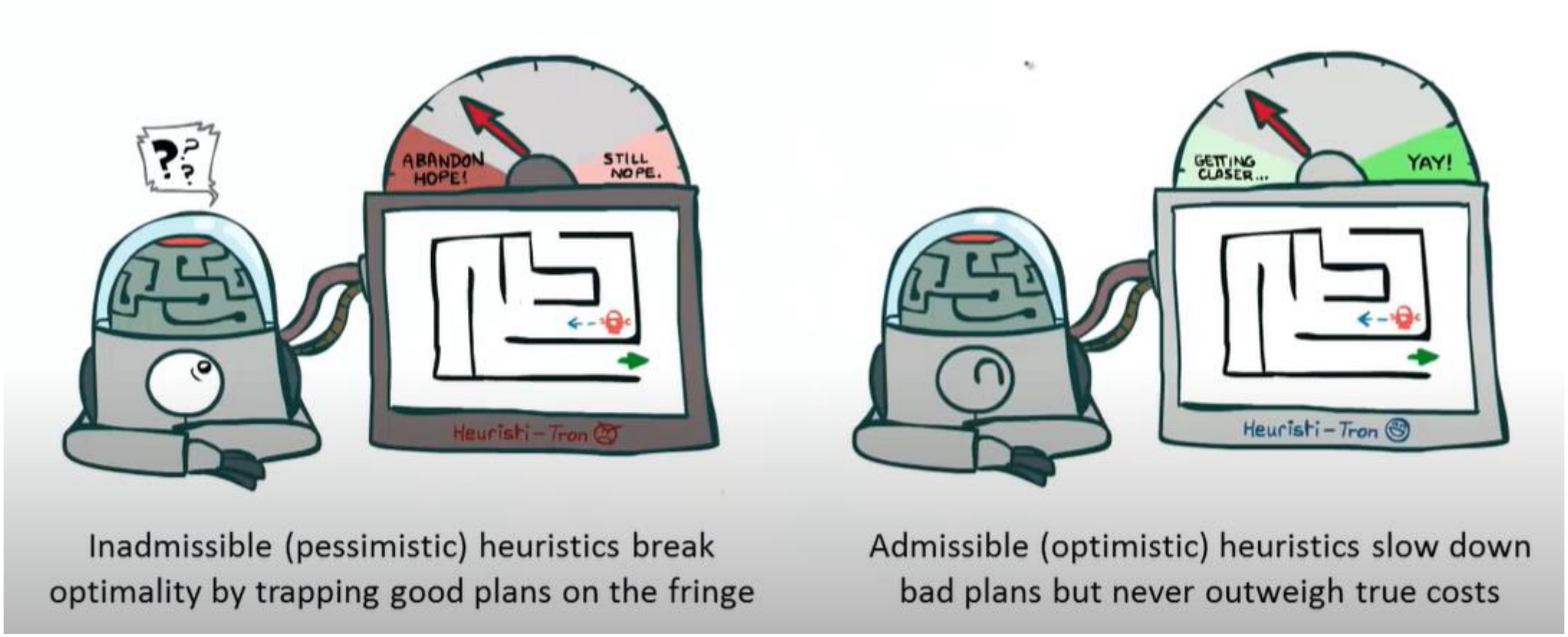


State (n)	H(n)
S	5
A	3
B	6
C	2
D	3
G	0

H(n) <= distance to goal(n)

State (n)	H(n)	distance to goal(n)
S		
A		
B		
C		
D		
G		

Admissibility

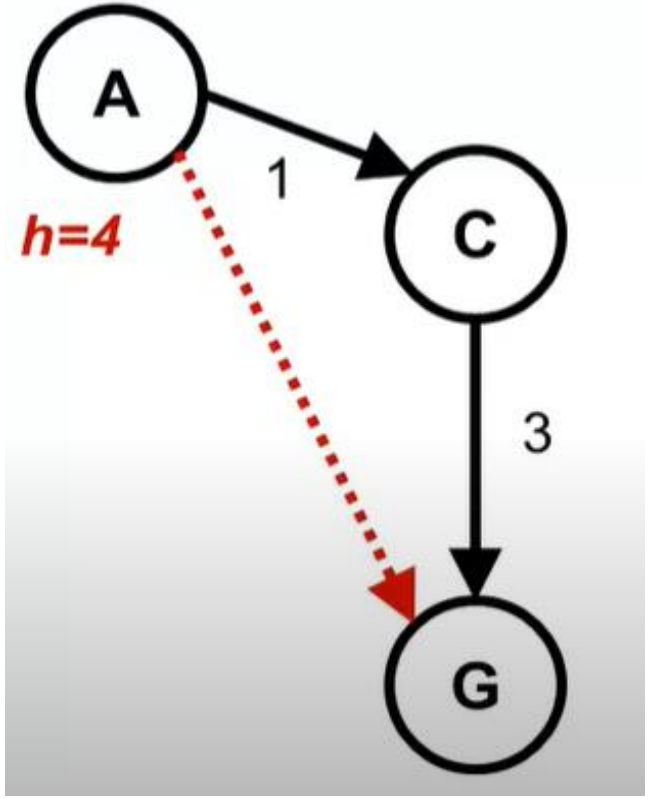


Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe

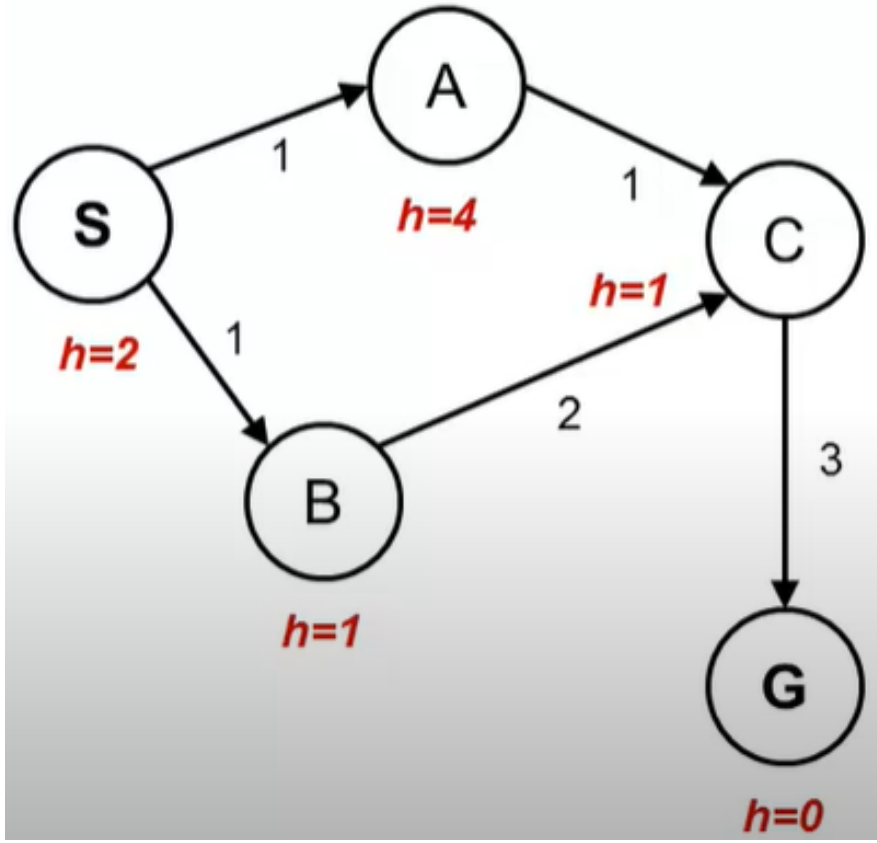
Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

Consistency of Heuristic (Monotonicity)

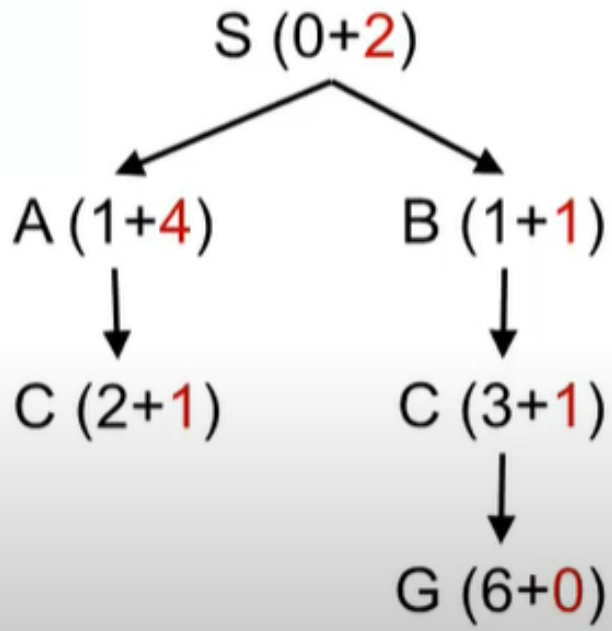
- Main idea: estimated heuristic costs <= actual costs
  - Admissibility: heuristic cost <=actual cost to goal
    - H(A) <= actual cost from A to G
  - Consistency: heuristic “arc’’ cost <= actual cost for each arc
    - H(A)-H(C) <= cost(A to C)
- Consequences of consistency
  - The f value along a path never decreases



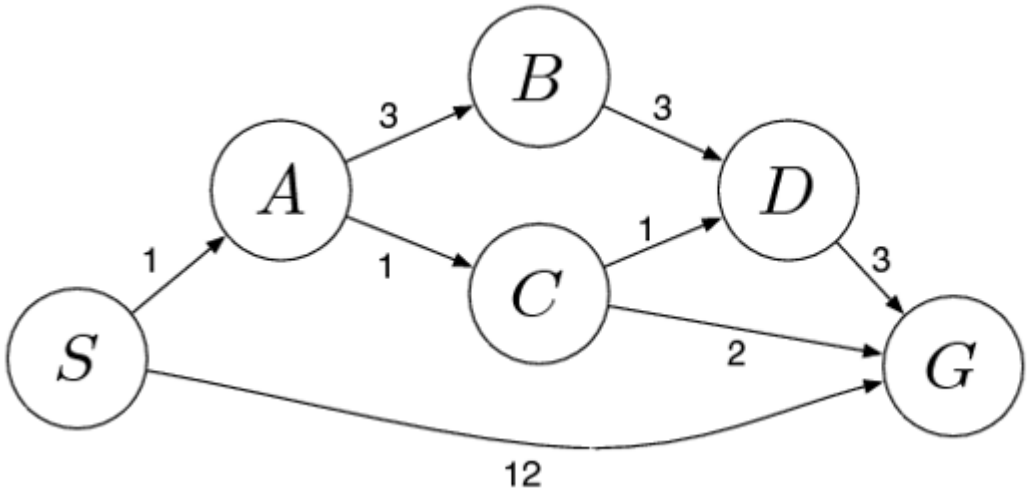
State space graph



Search tree



Monotonic Heuristic (Example)



State (n)	H(n)
S	5
A	3
B	6
C	2
D	3
G	0

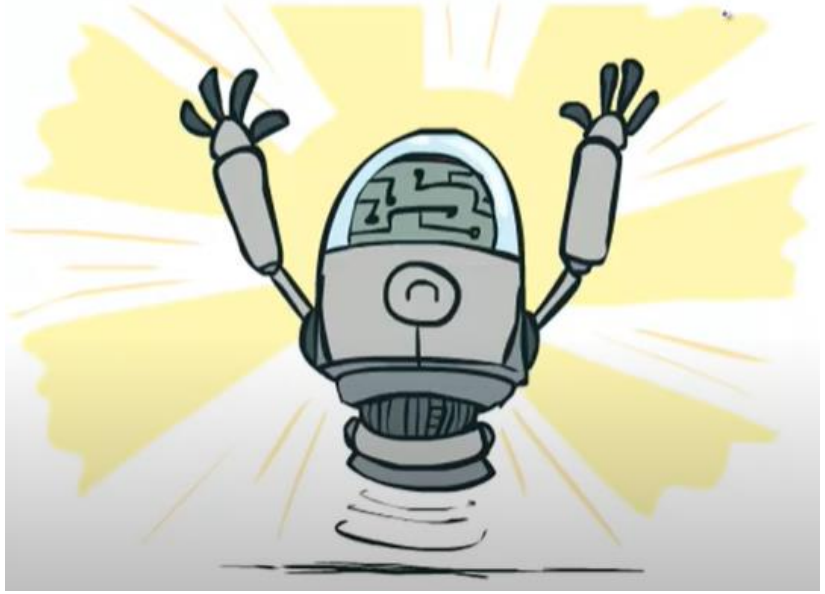
If it's monotonic then it's admissible

H(n) <= h(n')+c(n-n')

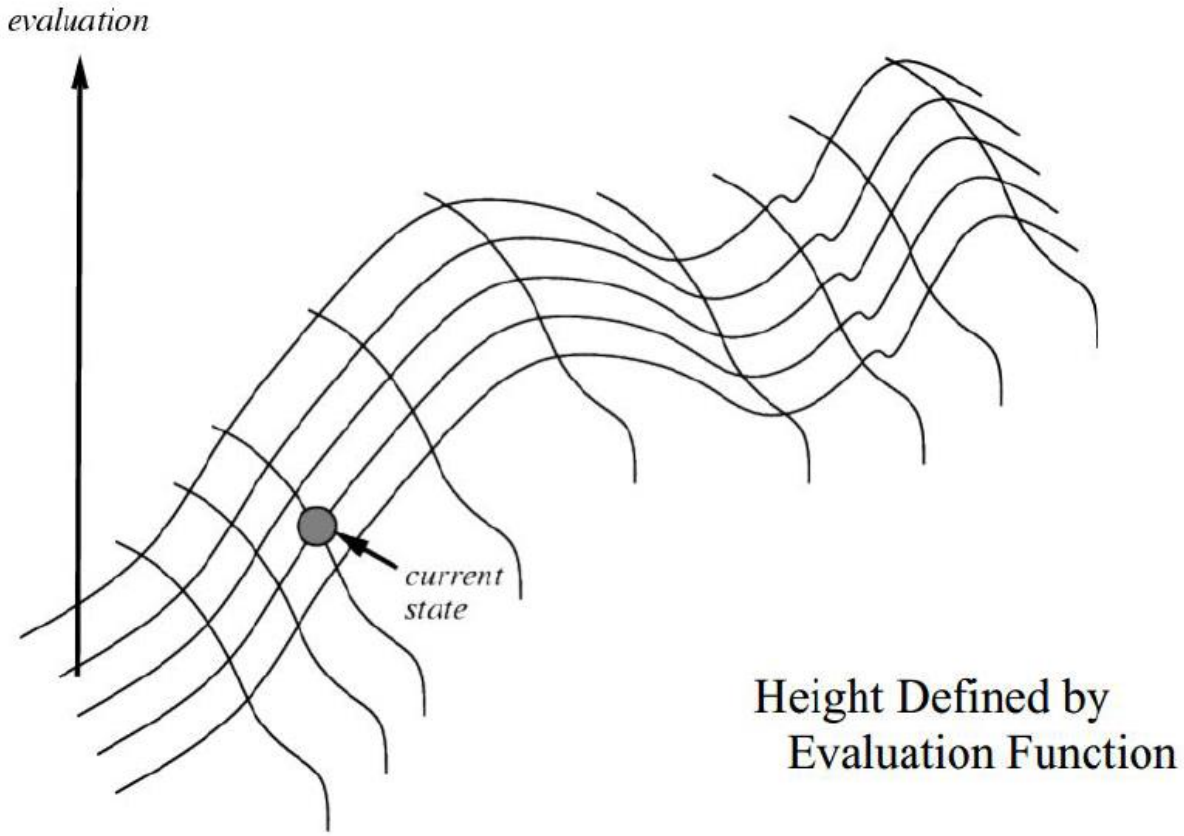
n-n'	H(n)	H(n')	C(n-n')	H(n')+c(n-n')
S-A				
S-G				
A-B				
A-C				
B-D				
C-D				
C-G				
D-G				

Optimality

- Tree Search
  - A* is optimal if heuristic is admissible
- Graph Search
  - A* is optimal if heuristic is consistent
- Consistency implies admissibility



Hill Climbing on a surface of states

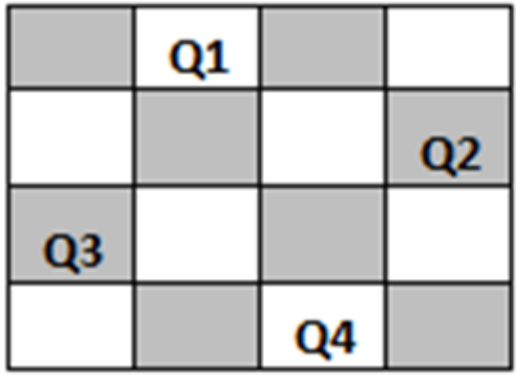


Hill Climbing (Local Search)

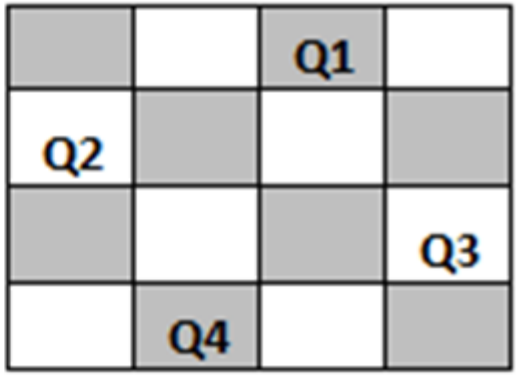
- It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.
- In many optimization problems, the path to the goal I irrelevant; the goal state itself is the solution
- In such cases, we can use local search algorithms
- Keep single “current” state, try to improve it

N Queen Problem

- The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other
- Example 4 Queen



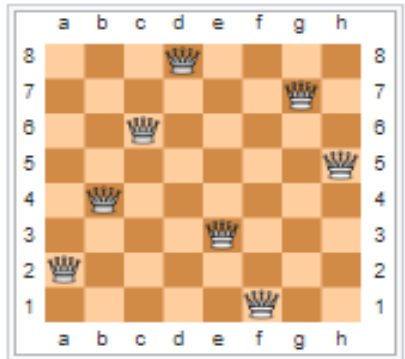
Solution 1

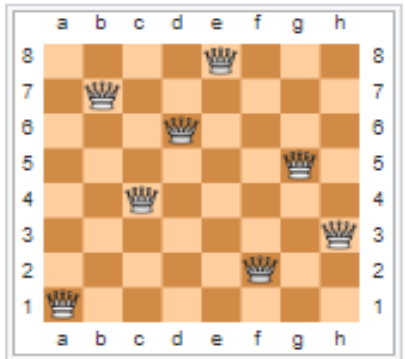


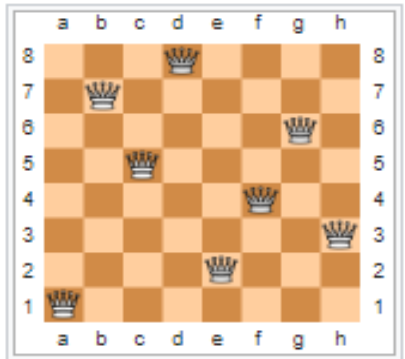
Solution 2

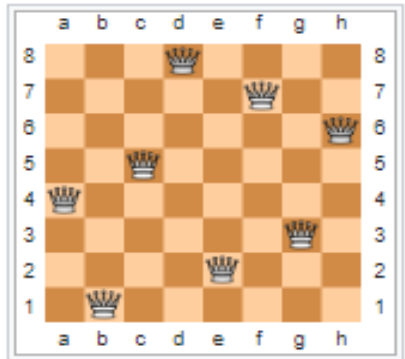
.

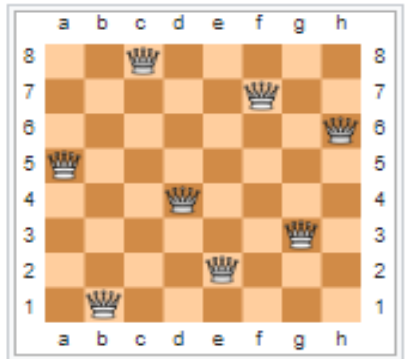
N (problem size)	*Number of solutions*	*Search space (N!)*	*Solution density* (per 10^6)
4	2	24	83,333
5	10	120	83,333
6	4	720	5,555
7	40	5,040	7,936
8	92	40,320	2,281
9	352	362,880	970
10	724	3,628,800	199
11	2,680	39,916,800	67
12	14,032	479,001,600	29

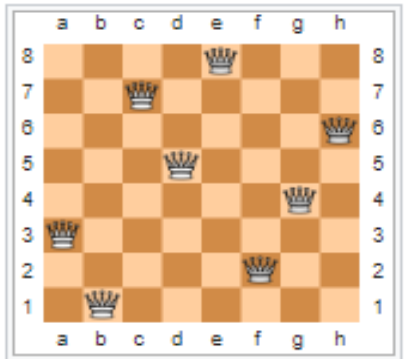

Solution 1

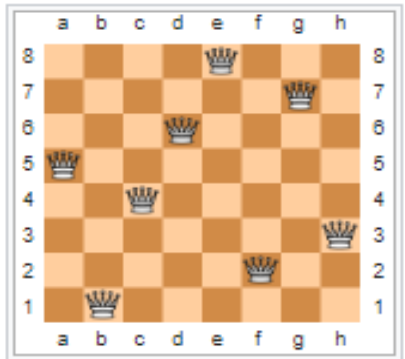

Solution 2

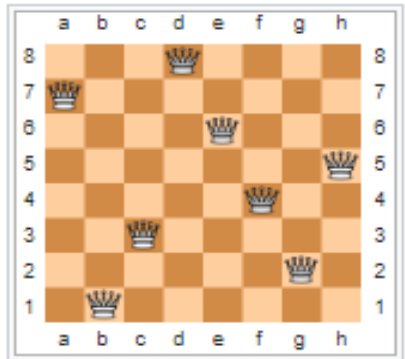

Solution 3

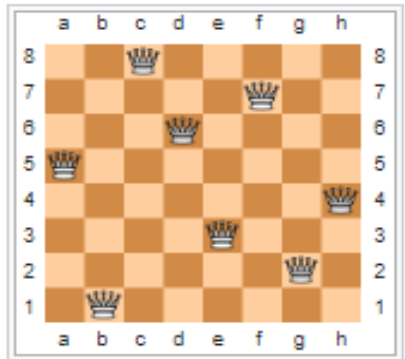

Solution 4

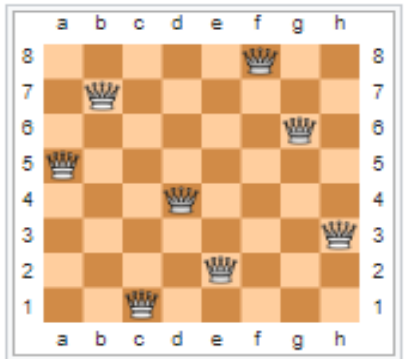

Solution 5

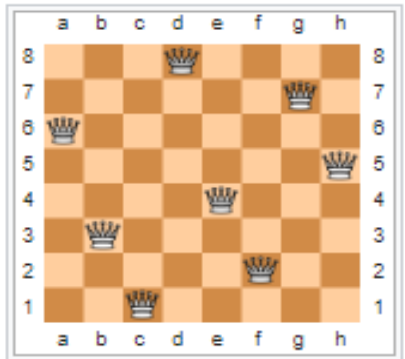

Solution 6

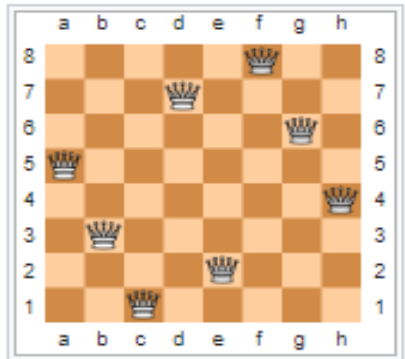

Solution 7


Solution 8


Solution 9


Solution 10


Solution 11


Solution 12

Hill Climbing

- Looks one step ahead to determine if any successor is better than the current state; if there is, move to the best successor.

Rule: If there exists a successor s for the current state n such that

  - $h(s) < h(n)$ and
  - $h(s) \leq h(t)$ for all the successors t of n,

then move from n to s. Otherwise, halt at n.

- Similar to Greedy search in that it uses h(), but does not allow backtracking or jumping to an alternative path since it doesn't "remember" where it has been.
- Appropriate for problems in which we need only a solution. The path to the solution is immaterial
  - E.g: 8- Queen problem, Graph colouring problem

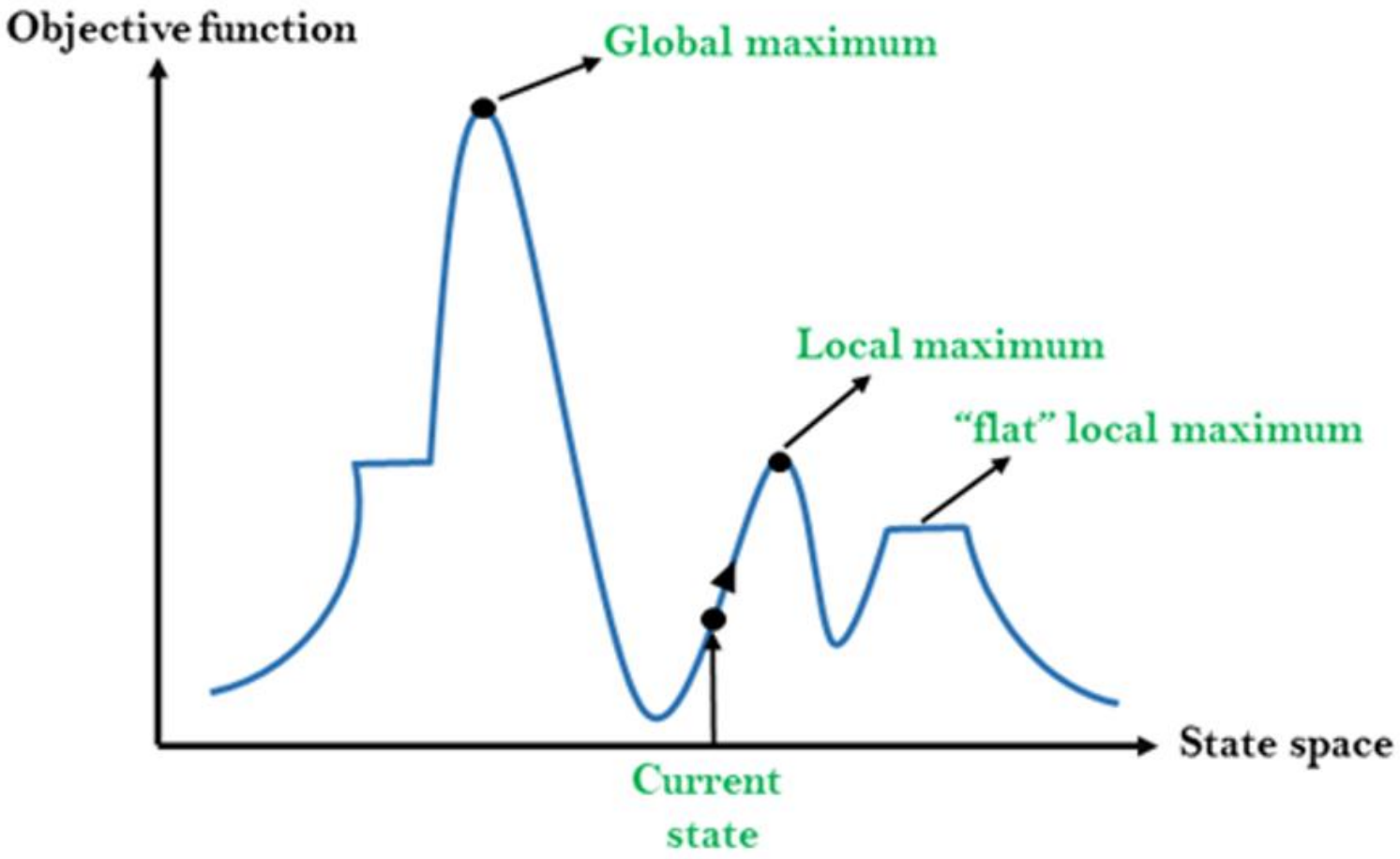
Pseudocode Hill Climbing

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] < VALUE[current] then return STATE[current]
      current ← neighbor
   end
```

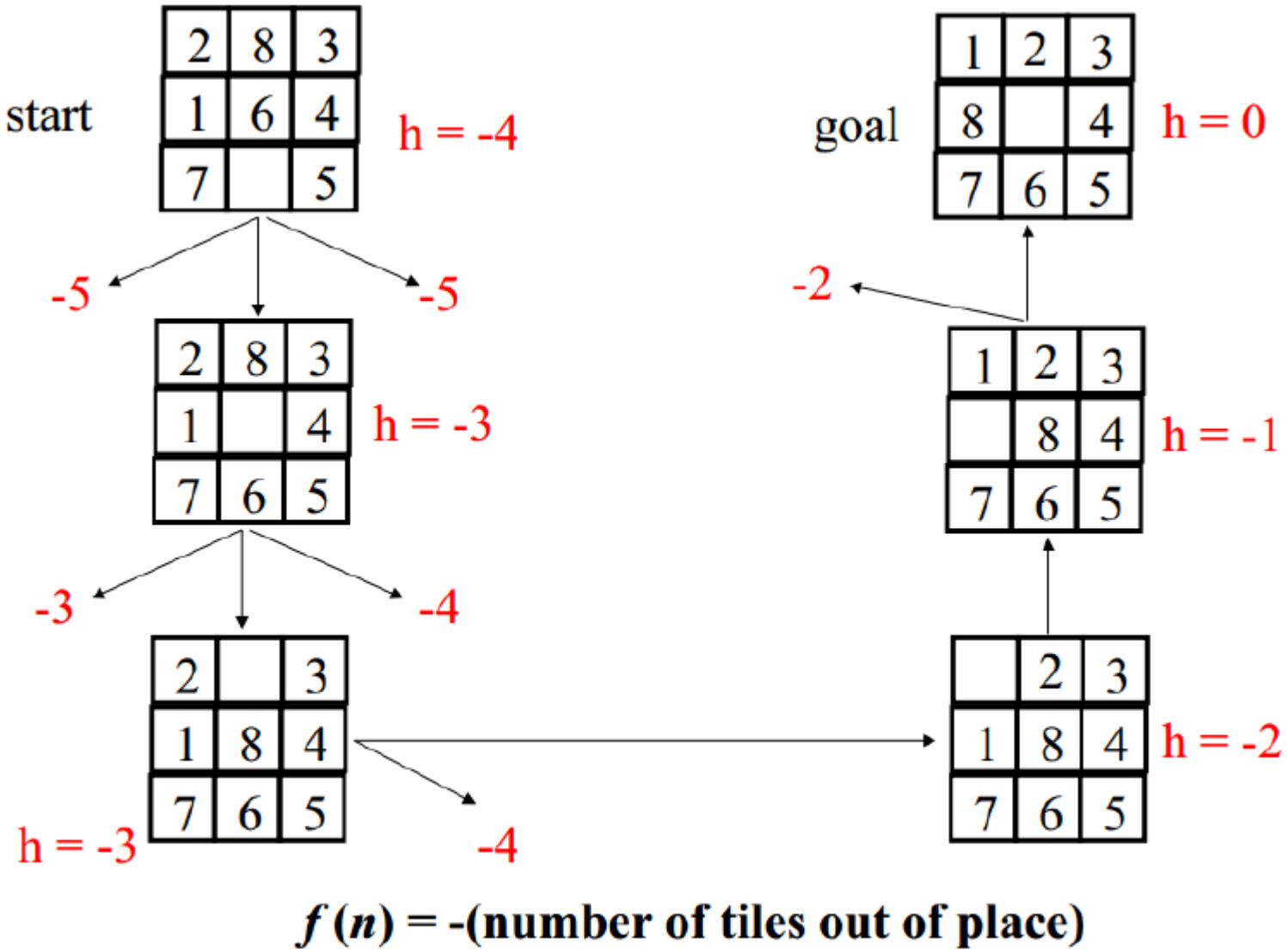
Looks at all successors

Example

Drawbacks of hill climbing

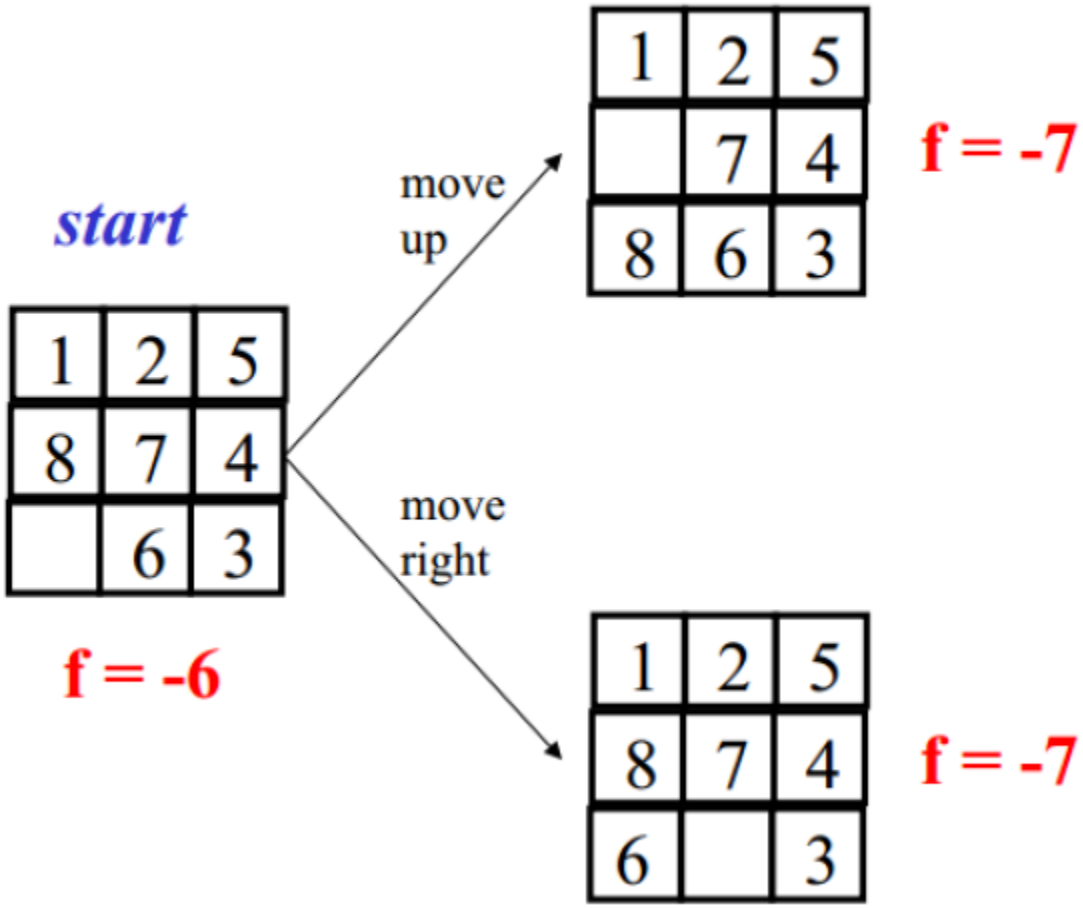


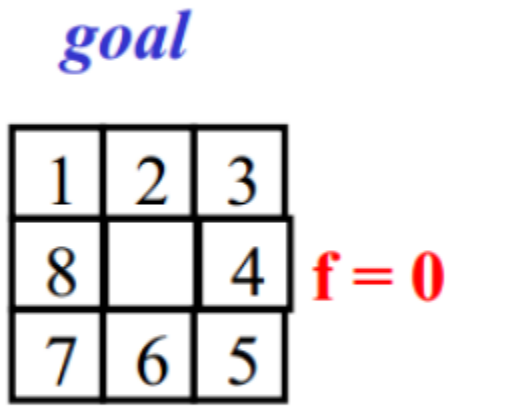
Example 1



***f* (*n*) = -(number of tiles out of place)**

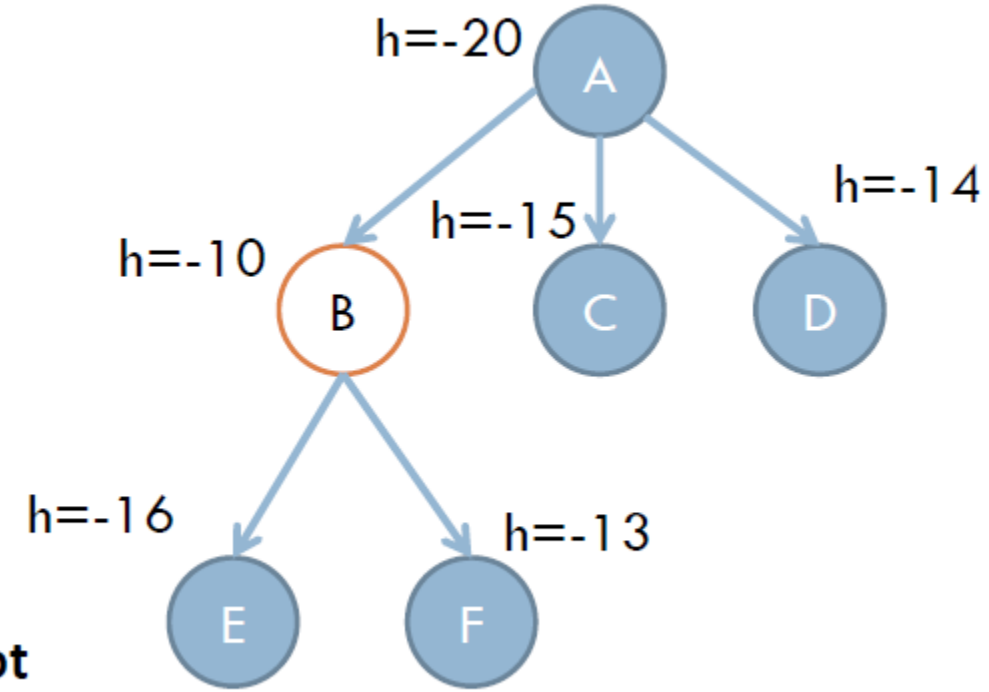
Example 2





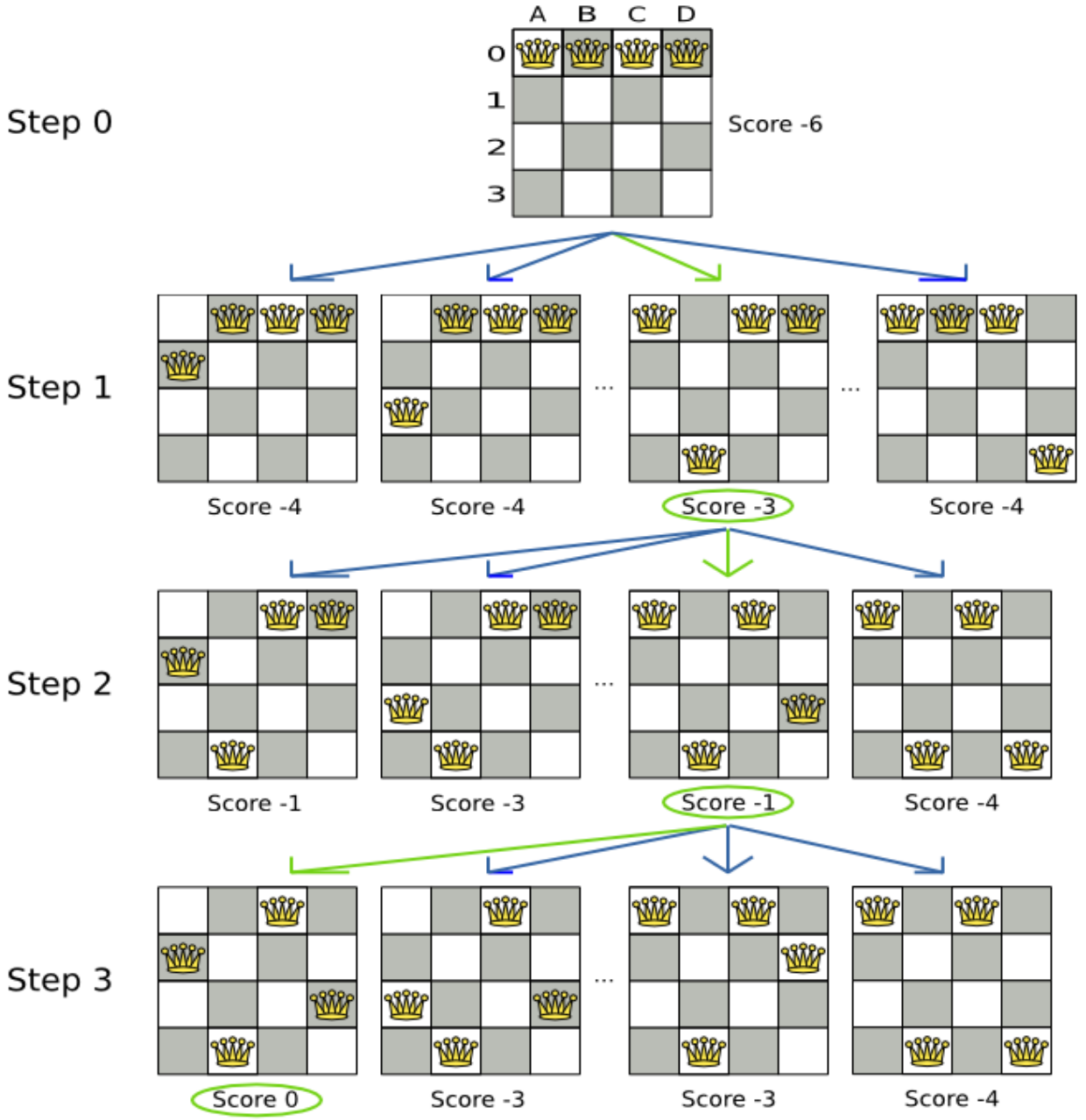
f = -(manhattan distance)

Example 3



Algorithm is not Complete

Example 4


A B C D
0
1
2
3
Step 0
Score -6
Step 1
Score -4
Score -4
Score -3
Score -4
Step 2
Score -1
Score -3
Score -1
Score -4
Step 3
Score 0
Score -3
Score -3
Score -4

Hill Climbing Analysis

- Complete ?
- Optimal ?
- Time Complexity
- Space Complexity

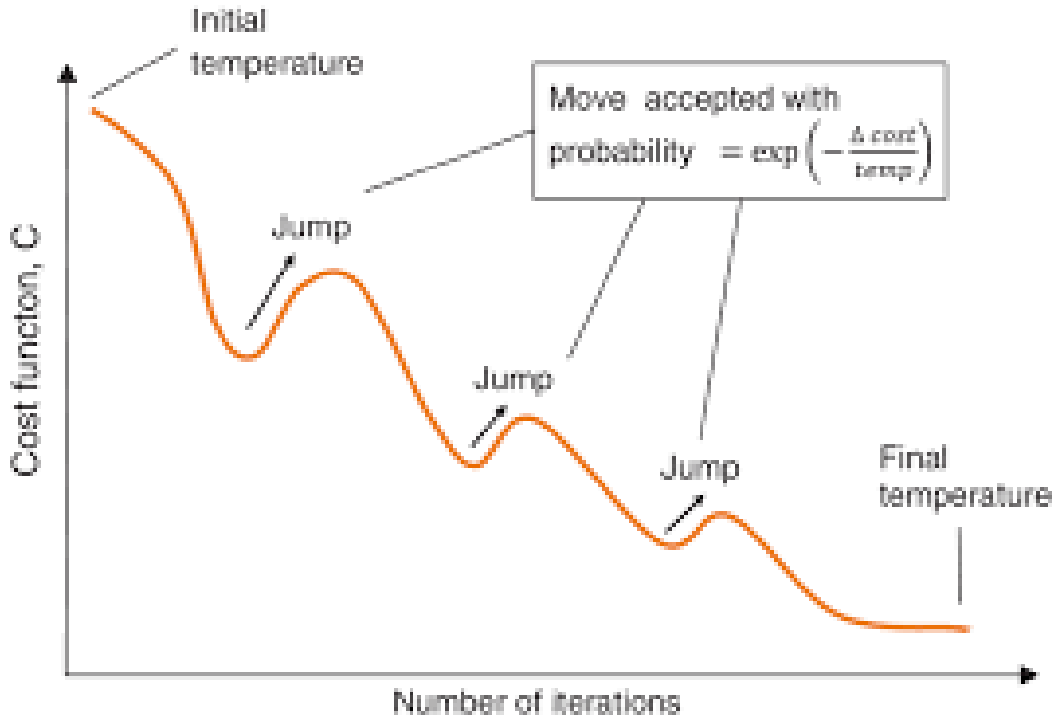
Simulated Annealing

Anneal

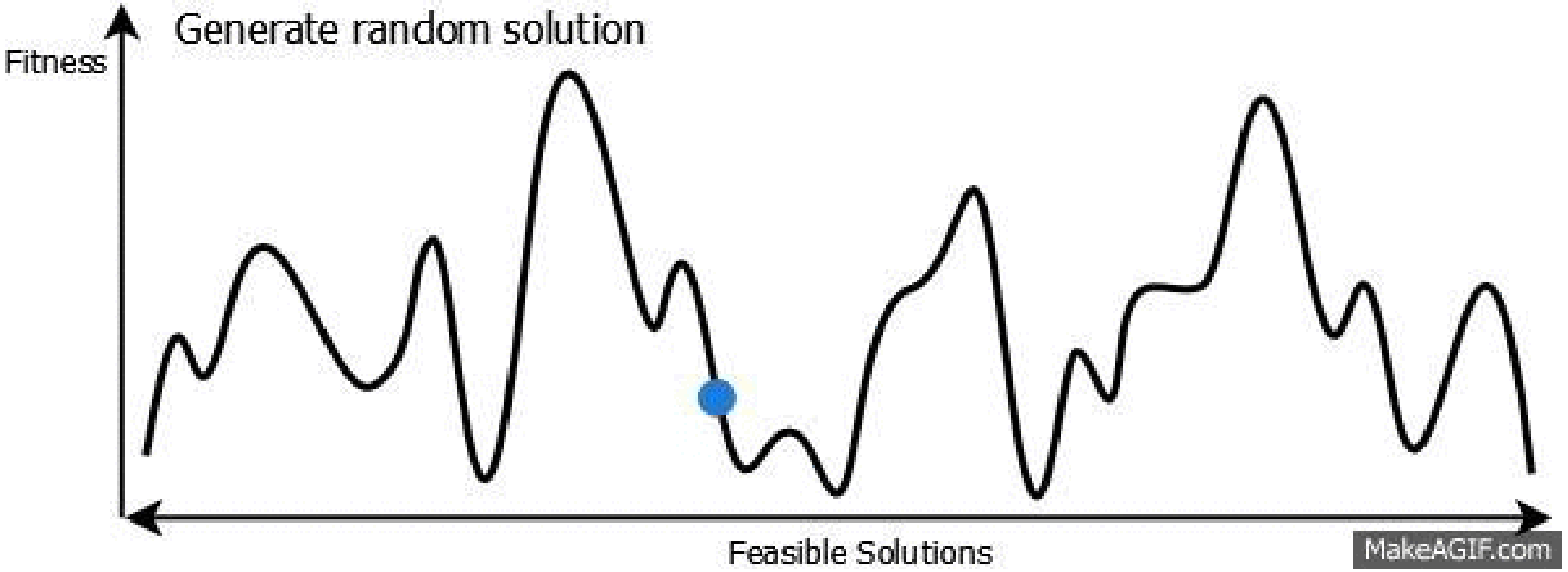
- To subject (glass or metal) to a process of heating and slow cooling in order to toughen and reduce brittleness.

Pseudocode of Simulated Annealing

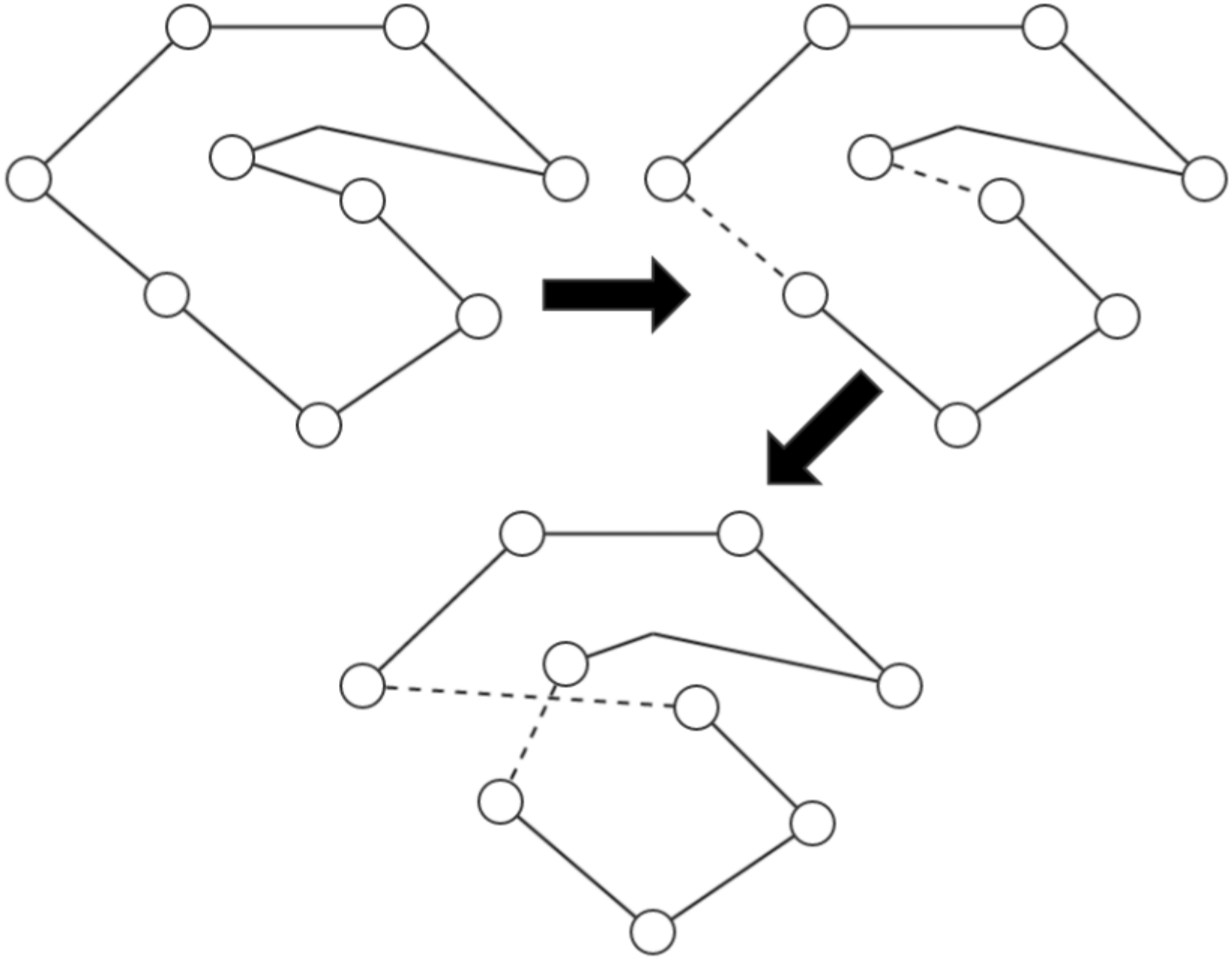
```
Procedure SIM_ANNEAL(m, iter_max, T)
BEGIN
    xcurr ← InitalConfig(m)
    xbest ← xcurr
    for i=1 to iter_max do
    BEGIN
        Tc ←Calc_Temp(i, T)
        xnext ←Random-Successor(x_curr)
        ΔE ← xnext – xcurr
        if (ΔE>0) BEGIN
            xcurr ← xnext
            if (xbest < xcurr) xbest ← xcurr
        END
        else if (e^(−ΔE/T) > Random(0, 1))
        xcurr ← xnext
    END
END
```



Simulated annealing algorithm



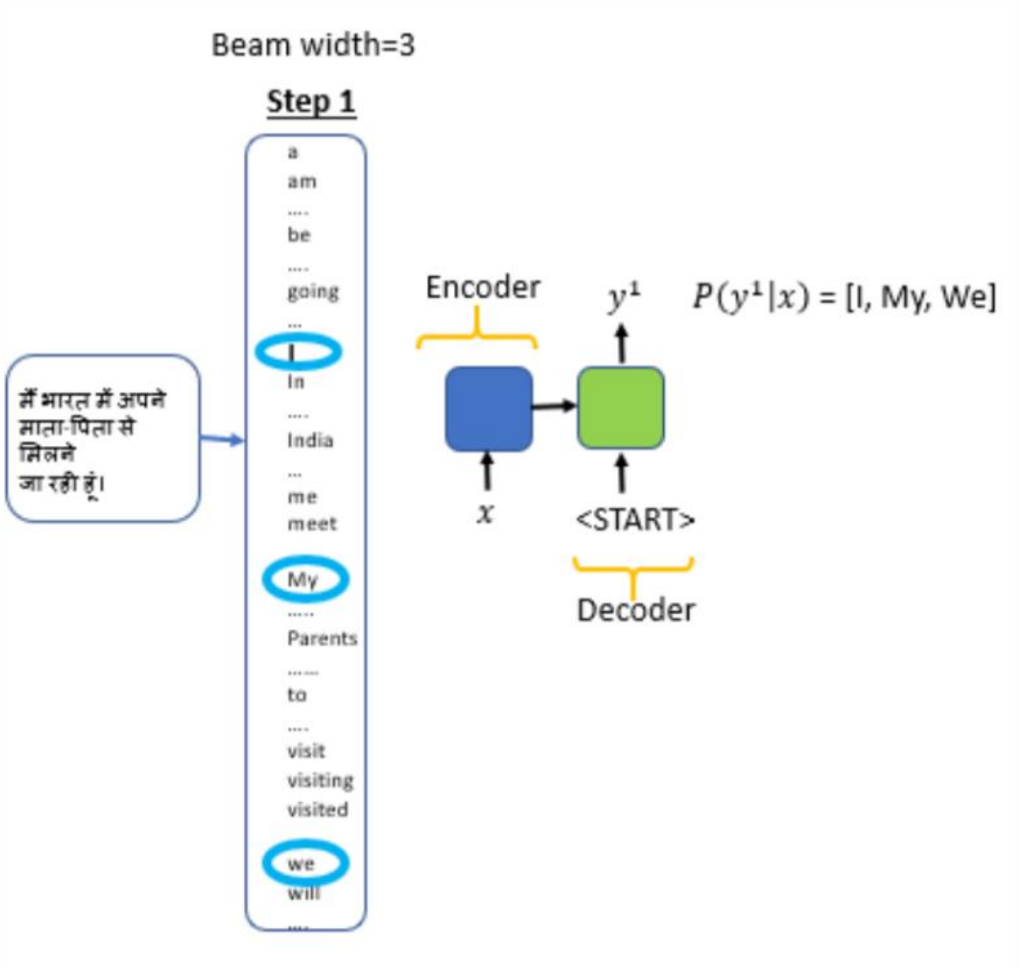
https://toddwschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny/



Beam Search

- Beam search is a heuristic search algorithm that explores a graph by expanding the most promising node in a limited set.
- Beam search is an optimization of breadth-first search that reduces its memory requirements.
- A beam search is most often used to maintain tractability in large systems with insufficient amount of memory to store the entire search tree. For example, it has been used in many machine translation systems.

Machine Translation(Seq to Seq)

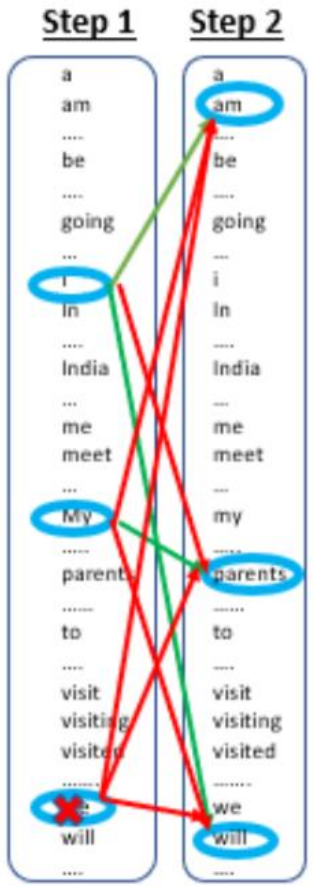


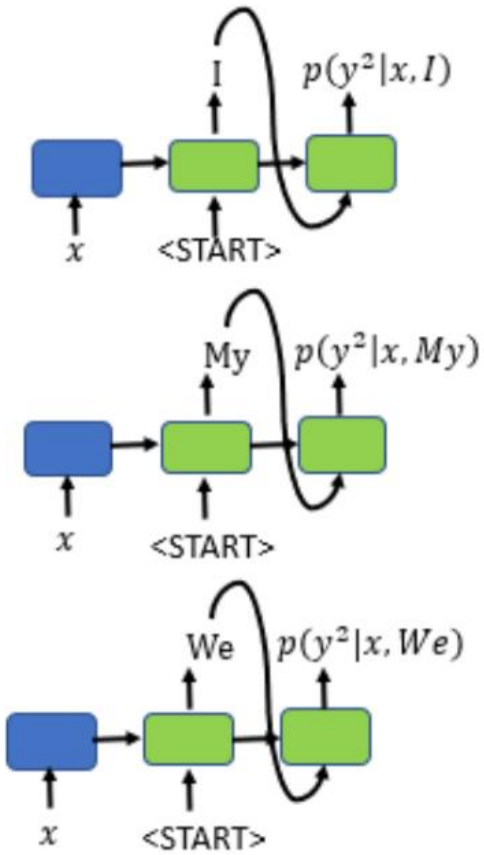
- Input the encoded input sentence to the decoder; the decoder will then apply softmax function to all the 10,000 words in the vocabulary.
- From 10,000 possibilities, we will select only the top 3 words with the highest probability.

Machine Translation(Seq to Seq)

Beam width=3

मैं भारत में अपने माता-पिता से मिलने जा रही हूं।



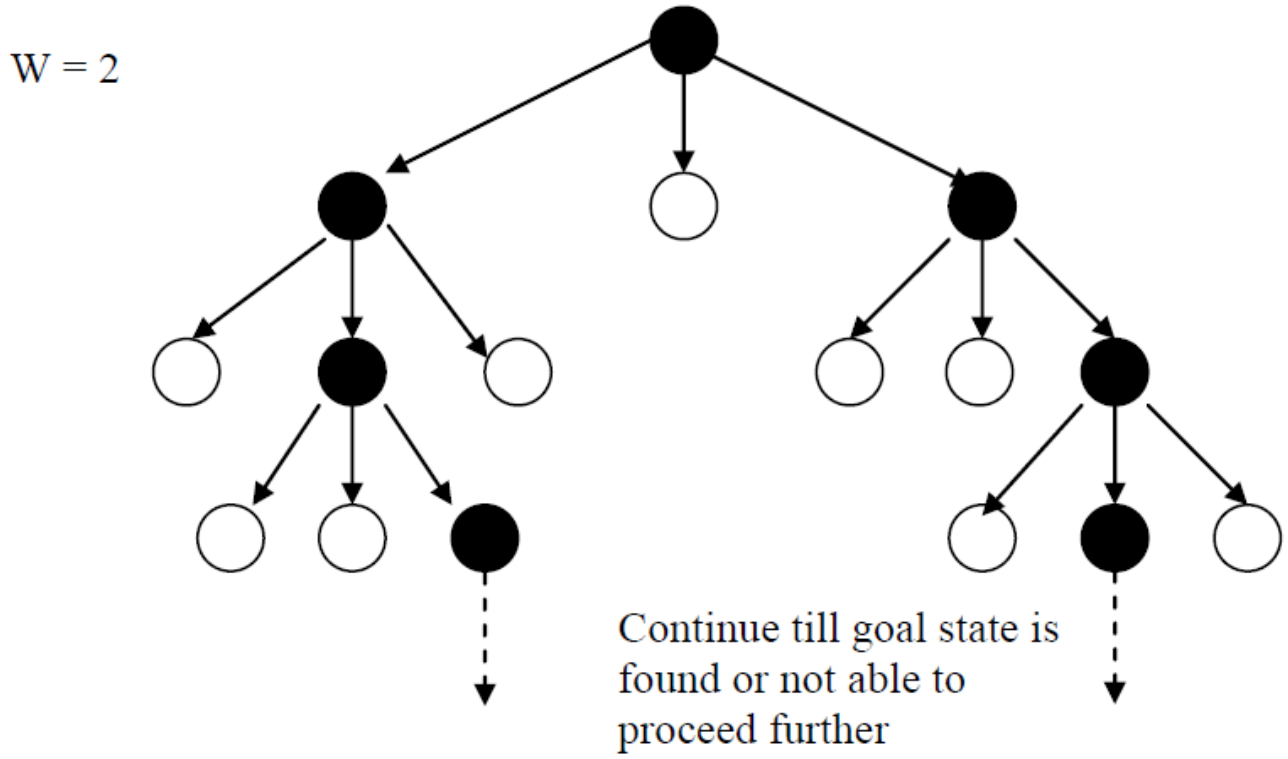


$P(y^1, y^2|x) = p(y^1|x) * p(y^2|x, y^1)$

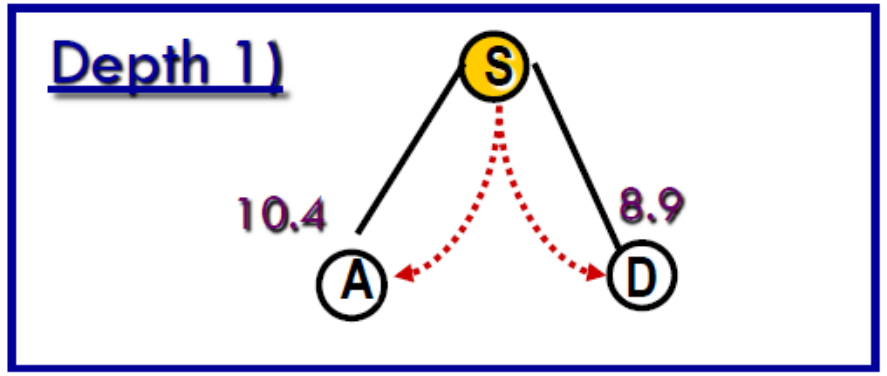
$P(y^1|x)$ = [I, My, We]

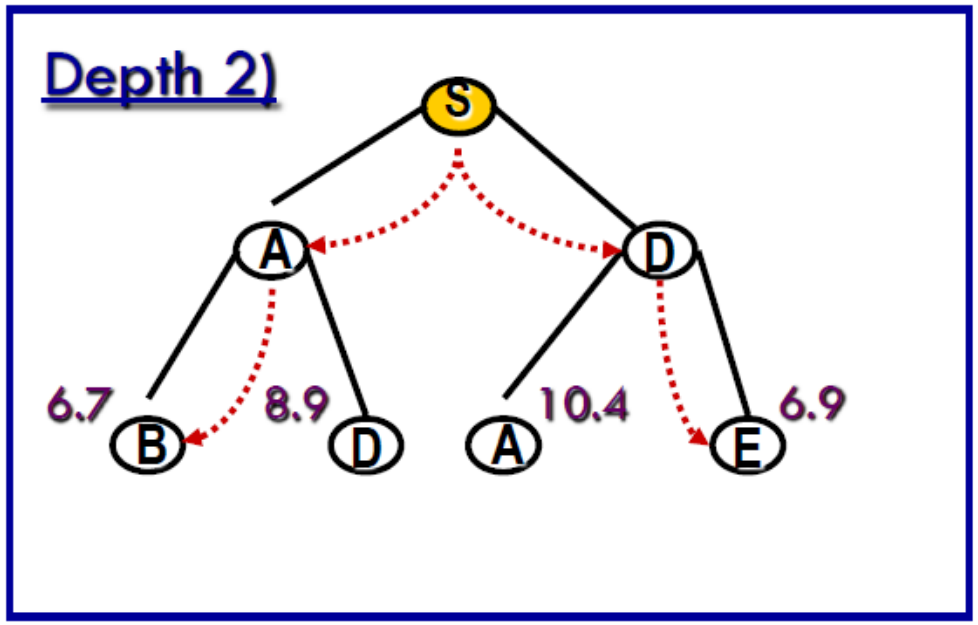
$P(y^1, y^2|x)$= [I am, I will, My parents]

Beam Search


W = 2
Continue till goal state is found or not able to proceed further

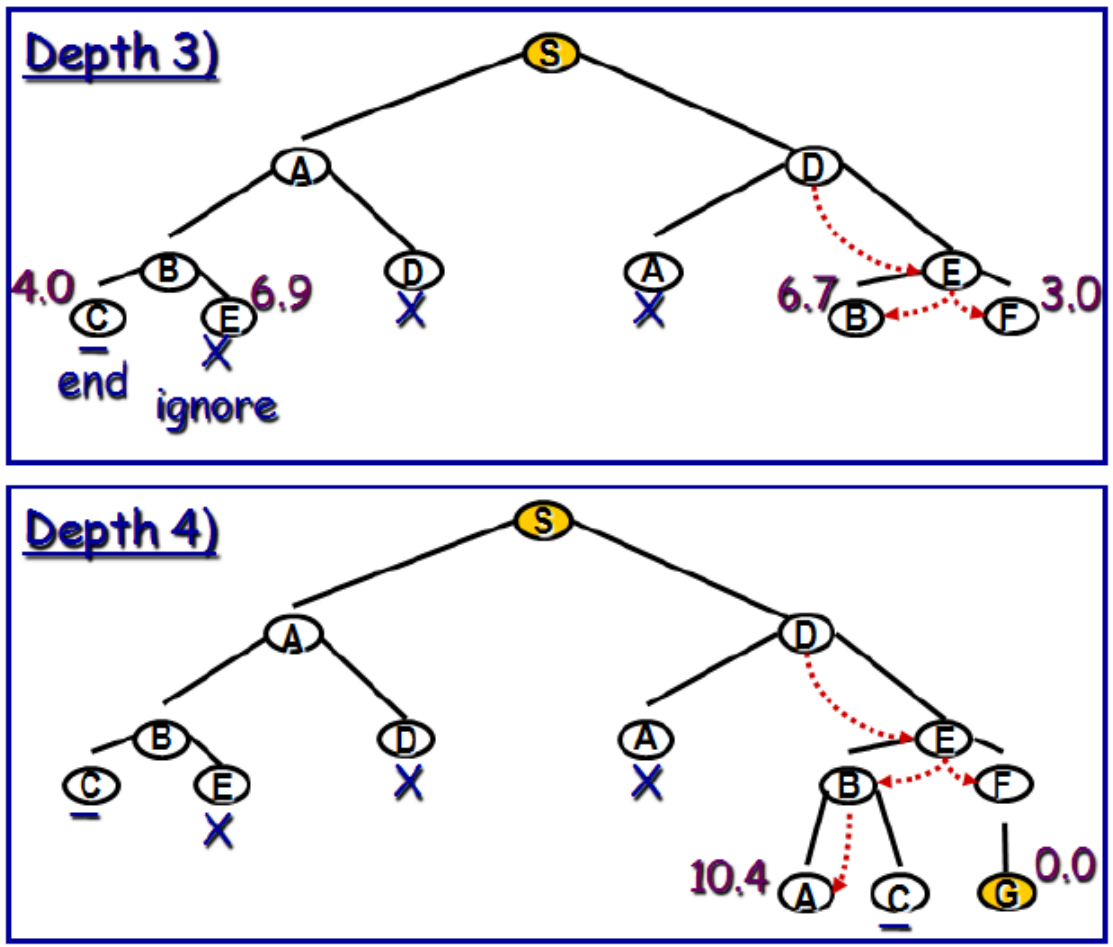
Beam Search Example





- Assume a pre-fixed WIDTH (example : 2)
- Perform breadth-first, BUT Only keep the width best new nodes depending on heuristic at each new level.

Beam Search Example



- Optimization: ignore leafs that are not goal nodes (see C)

Pseudocode of Beam Search

- Found = false;
- NODE = Root_node;
- If NODE is the goal node, then *Found = true* else find SUCCs of NODE, if any with its estimated cost and store in OPEN list;
- While (Found = false and not able to proceed further)

  {
  - Sort OPEN list;
  - Select top W elements from OPEN list and put it in W_OPEN list and empty OPEN list;
  - While (W_OPEN ≠ ϕ and Found = false)

  {
    - Get NODE from W_OPEN;
    - If NODE = Goal state then Found = true else

      {
      - Find SUCCs of NODE, if any with its estimated cost
      - store in OPEN list;

      }

  } // end while

} // end while

- If *Found = true* then return *Yes* otherwise return *No* and Stop

Beam Search Analysis

- Complete ?
- Optimal ?
- Time Complexity
- Space Complexity

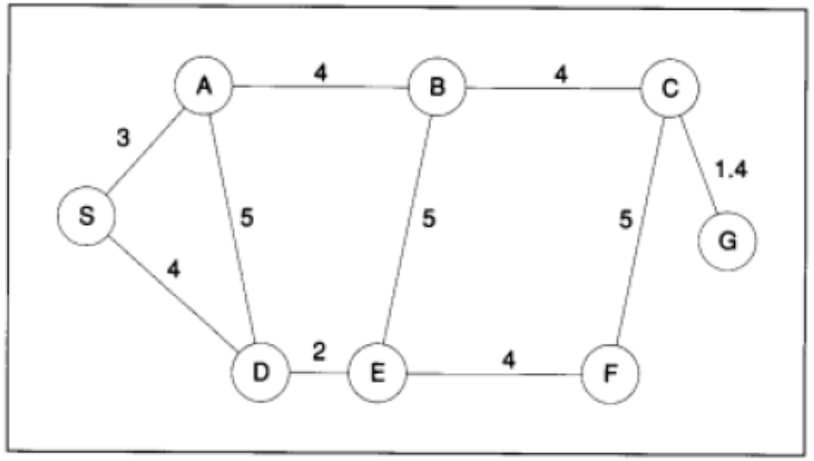
Class Exercise 1

- Consider the following graph. S denotes the starting state and G denotes the goal state. The number attached to each edge in the graph represents the COST of traversing the edge. Assume also that the ESTIMATED distances to the goal are given by the following table:

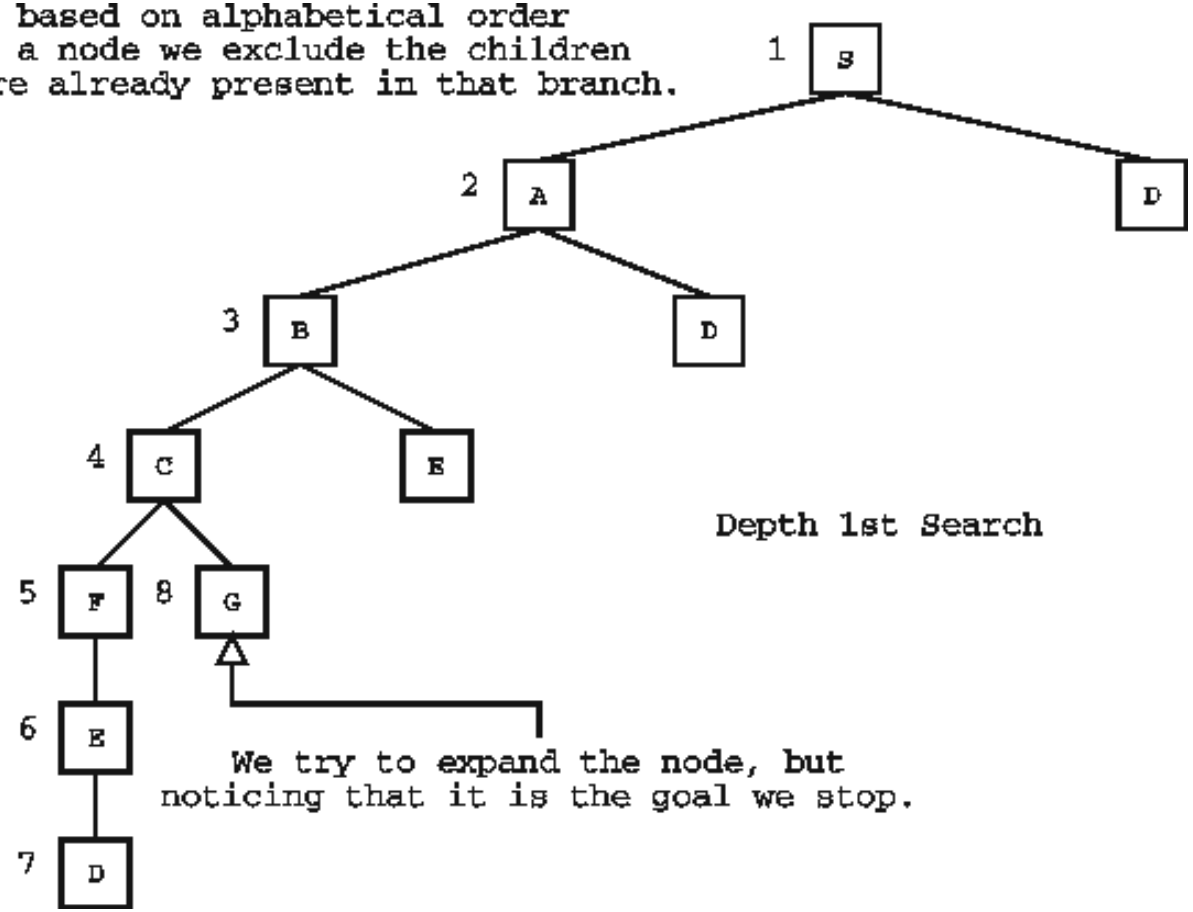
FROM	TO	ESTIMATED DISTANCE
S	G	10
A	G	8
B	G	5
C	G	1.4
D	G	9
E	G	6
F	G	2
G	G	0

Class Exercise 1

- For each of the following search methods show the search tree as it is explored/expanded by the method until it finds a path from S to G. Number the nodes in the order in which they are expanded by each method. **Show your work**.
  - Depth First Search
  - Breadth First Search
  - Uniform Cost Search
  - Best First Search
  - A*
  - Hill-climbing
  - Beam Search (m = 2)



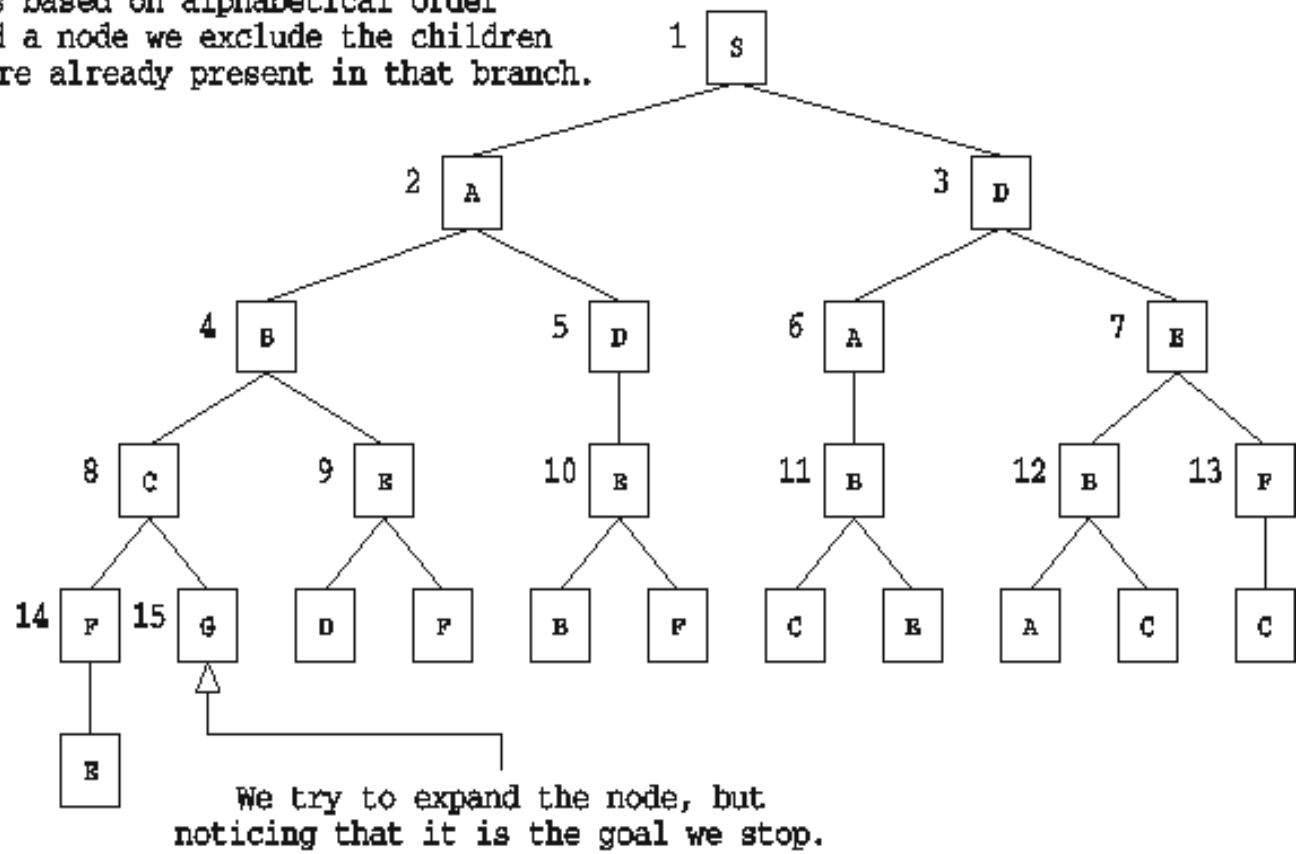
We select nodes based on alphabetical order
and when we expand a node we exclude the children
of that node which are already present in that branch.



Depth 1st Search

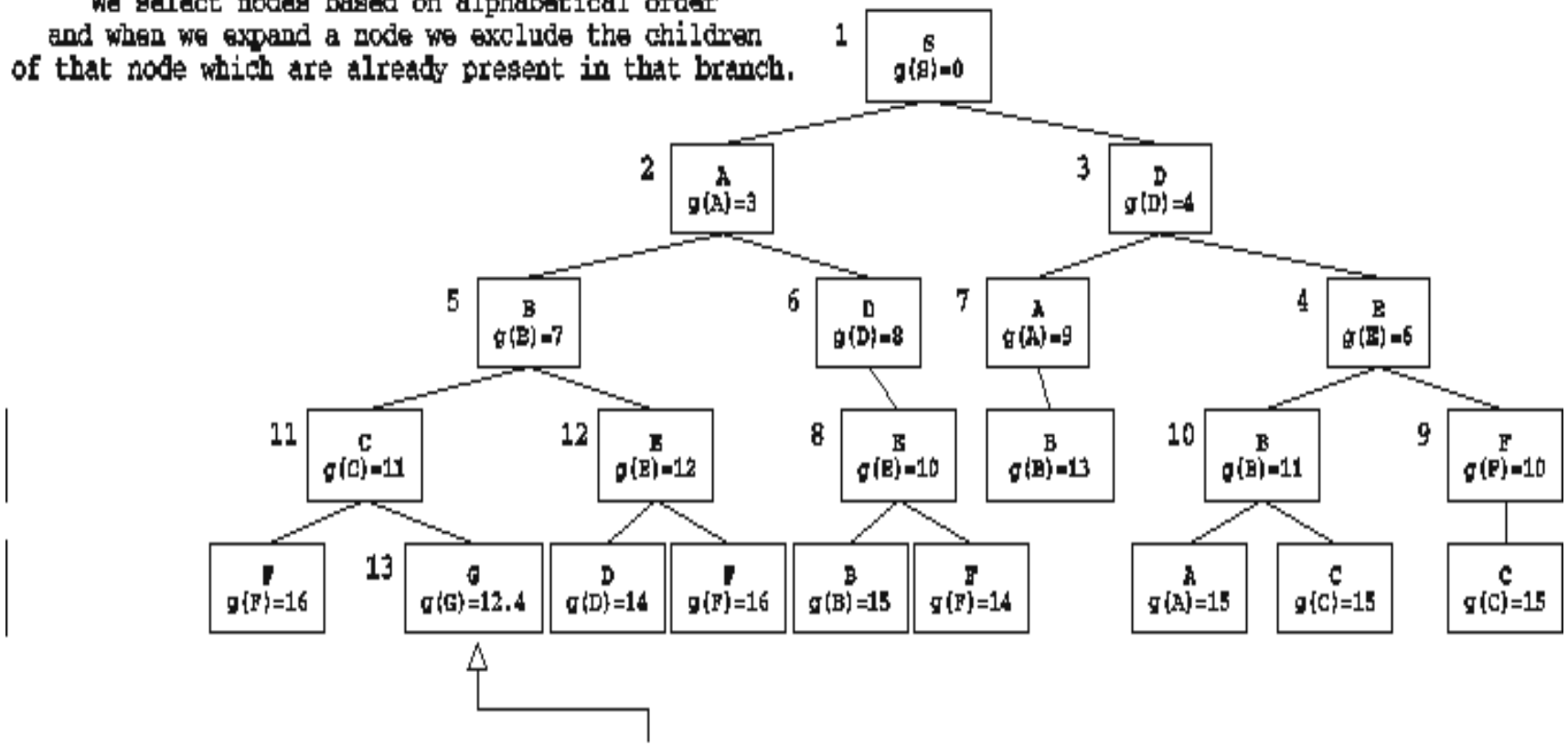
We try to expand the node, but
noticing that it is the goal we stop.

We select nodes based on alphabetical order
and when we expand a node we exclude the children
of that node which are already present in that branch.



Breadth 1st Search

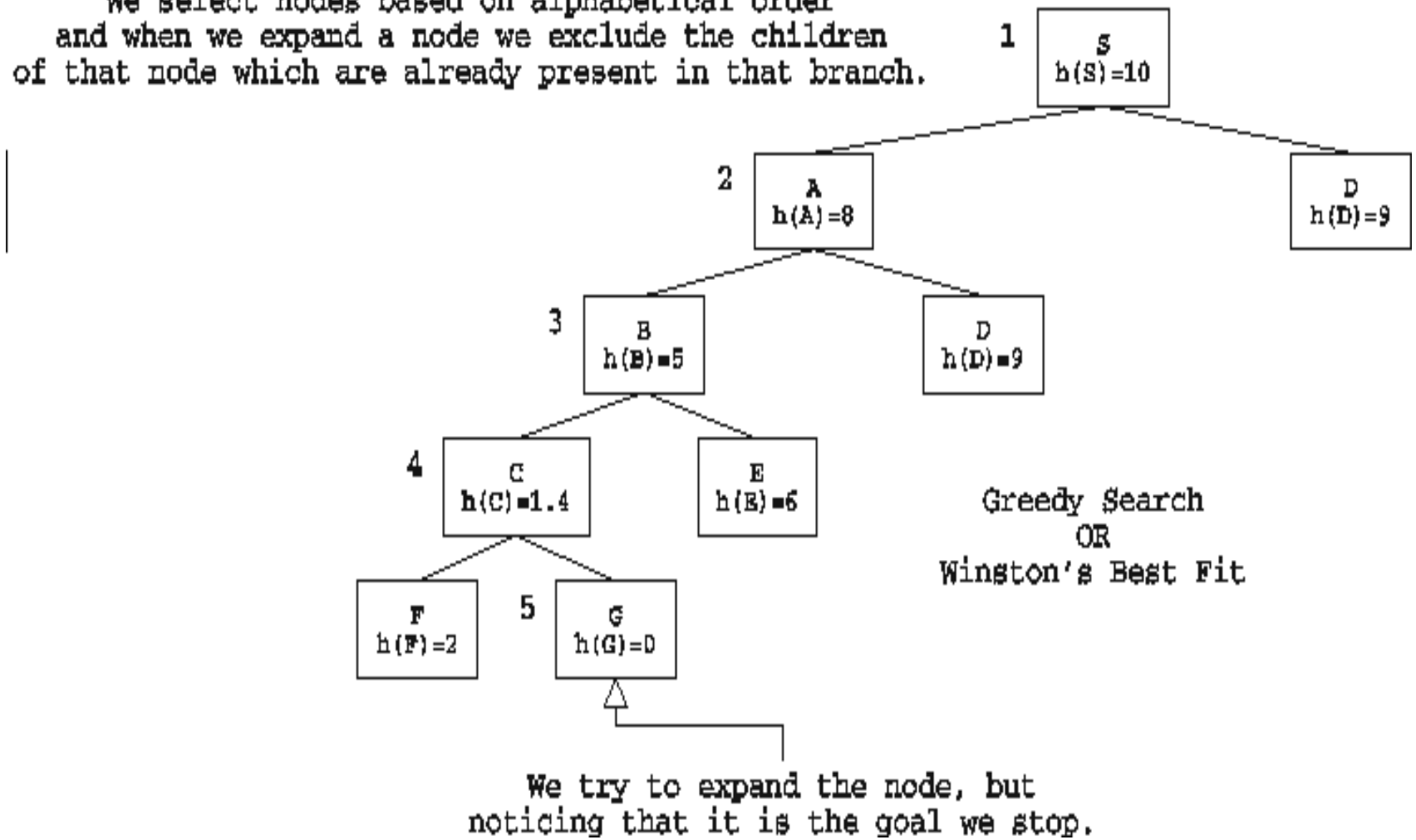
We select nodes based on alphabetical order
and when we expand a node we exclude the children
of that node which are already present in that branch.



We try to expand the node, but
noticing that it is the goal we stop.

Branch and Bound Search OR Uniform-Cost Search

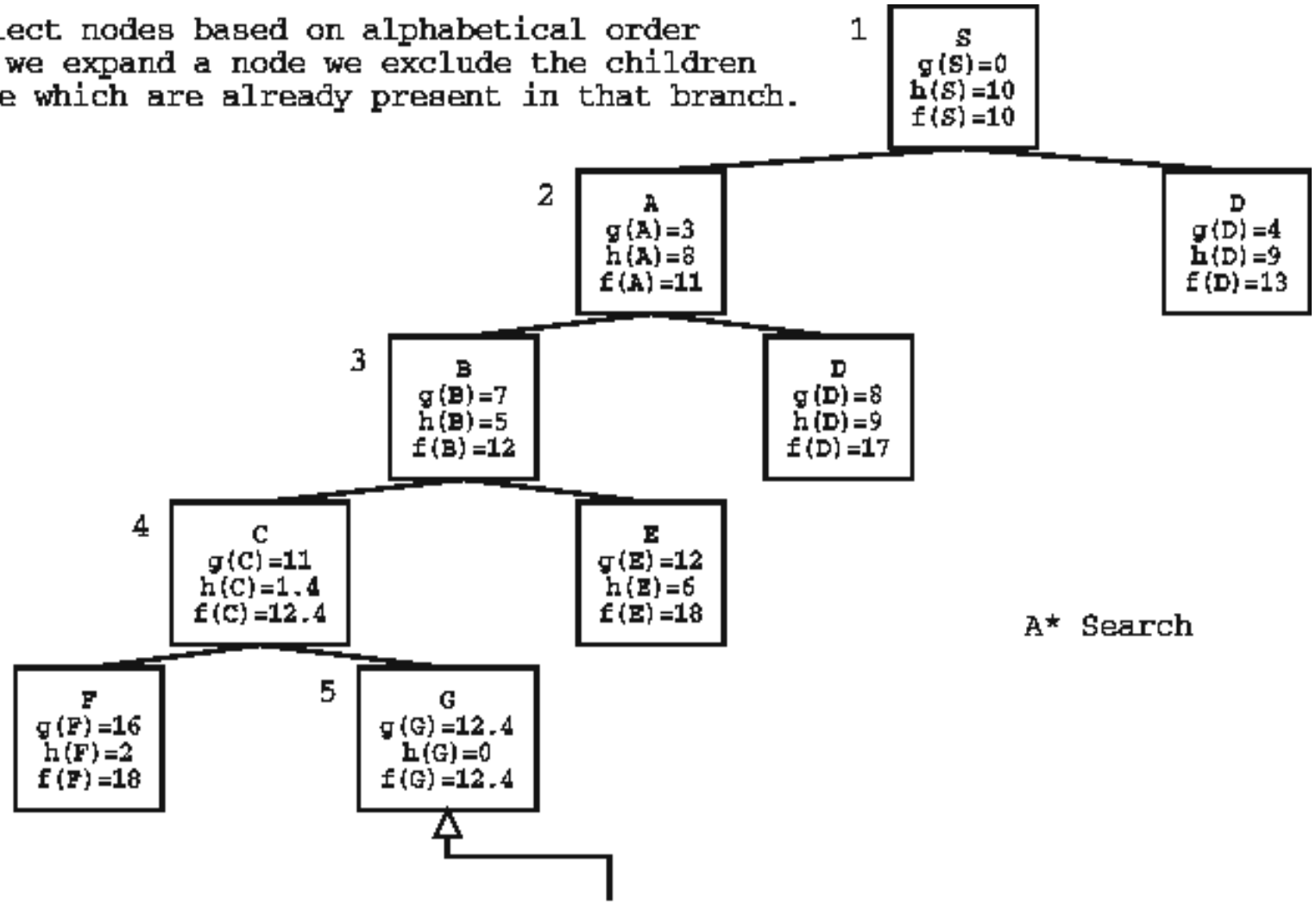
We select nodes based on alphabetical order
and when we expand a node we exclude the children
of that node which are already present in that branch.



Greedy Search
OR
Winston's Best Fit

We try to expand the node, but
noticing that it is the goal we stop.

We select nodes based on alphabetical order and when we expand a node we exclude the children of that node which are already present in that branch.

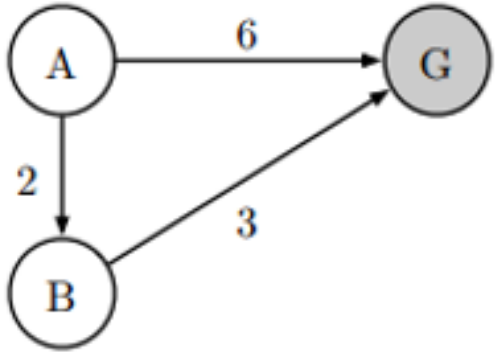


A* Search

We try to expand the node, but noticing that it is the goal we stop.

Class Exercise 2

Assume there is two heuristic function for the problem shown below.



	H1	H2
A	4	5
B	1	4
G	0	0

For each heuristic function, tell whether it is admissible and whether it is monotonic with respect to the search problem given above.

Class Exercise 3

- Assume you have the following Maze environment, and you want to help the slug to reach here food as soon as possible. Which among the following algorithm is the best to use: **Breadth-First Search, A-star,** or **Greedy search.** Explain your result by showing the sequence of tested node (In-order)?
- The slug can only move **up, down, left, or right**. Any of these actions can only be performed on condition that the resulting state remains within the maze and that the resulting state is not a black cell. Also, actions that bring you back to a previous state are not allowed. Note: Assume that ties are broken alphabetically (alphabetical order of the labels in the cells).