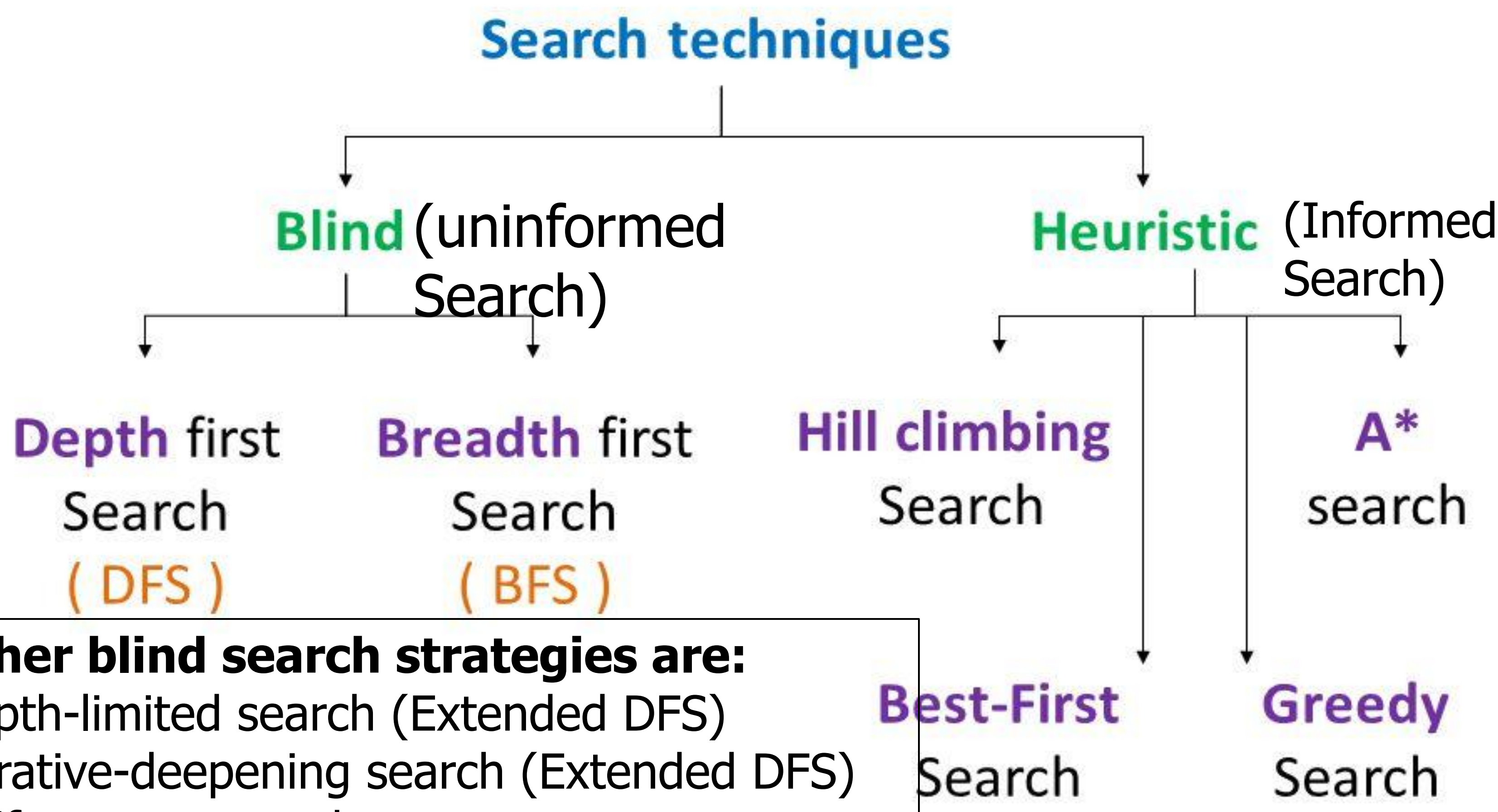
Artificial Intelligence
Informed Search

SEARCH TECHNIQUES



Other blind search strategies are:

- Depth-limited search (Extended DFS)
- Iterative-deepening search (Extended DFS)
- Uniform cost search
- Bi-directional search

Uninformed Vs Informed Search

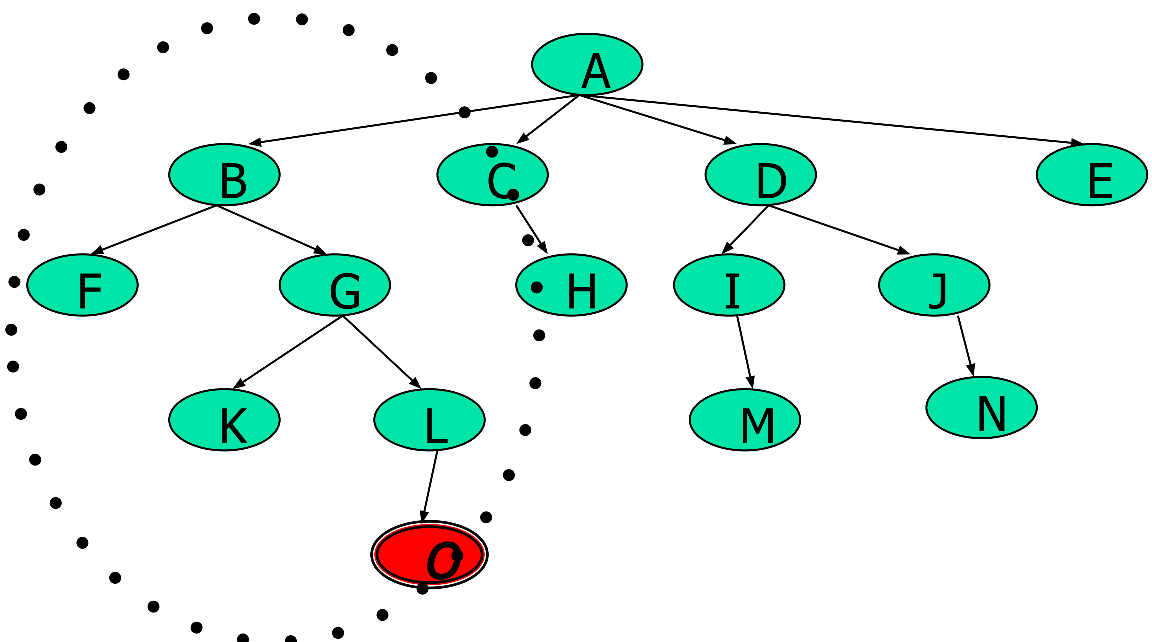
Uninformed search: Use only the information available in the problem definition. Example: breadth-first, depth-first, depth limited, iterative deepening, uniform cost and bidirectional search

Informed search: Use domain knowledge or heuristic to choose the best move. Example. Greedy best-first, A*, IDA*, and beam search

Using problem specific knowledge to aid searching

- With knowledge, one can search the state space as if he was given "hints" when exploring a maze.
  - Heuristic information in search = Hints
- Leads to dramatic speed up in efficiency.

Search only in this subtree!!



More formally, why heuristic functions work?

- In any search problem where there are at most b choices at each node and a depth of d at the goal node, a naive search algorithm would have to, in the worst case, search around $O(b^d)$ nodes before finding a solution (Exponential Time Complexity).

- Heuristics improve the efficiency of search algorithms by reducing the effective branching factor from b to (ideally) a low constant b* such that
  - 1 =< b* << b

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Heuristic Functions

- A heuristic function is a function *h(n)* that gives an estimation on the "cost" of getting from node *n* to the goal state – so that the node with the least cost among all possible choices can be selected for expansion first. An evaluation function f(n) can use h(n) to define the goodness of a state

- Three approaches to defining *f*:
  - *f* measures the value of the current state (its "goodness")
  - *f* measures the estimated cost of getting to the goal from the current state:
    - *f(n)* = *h(n)* where *h(n)* = an estimate of the cost to get from *n* to a goal
  - *f* measures the estimated cost of getting to the goal state from the *current state* and the cost of the existing path to it. Often, in this case, we decompose *f*:
    - *f(n)* = *g(n)* + *h(n)* where *g(n)* = the cost to get to *n* (from initial state)

Approach 1: *f* Measures the Value of the Current State

- Usually the case when solving optimization problems
  - Finding a state such that the value of the metric *f* is optimized

- Often, in these cases, *f* could be a weighted sum of a set of component values:

  - N-Queens
    - Example: the number of queens under attack …

  - Data mining
    - Example: the "predictive-ness" (a.k.a. accuracy) of a rule discovered

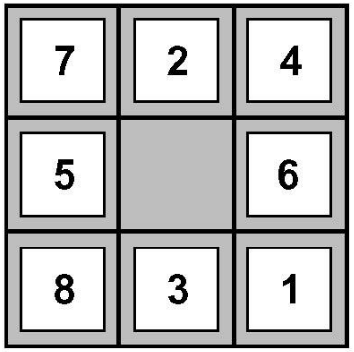
Approach 2: *f* Measures the Cost to the Goal

A state *X* would be better than a state *Y* if the estimated cost of getting from *X* to the goal is lower than that of *Y* – because *X* would be closer to the goal than *Y*

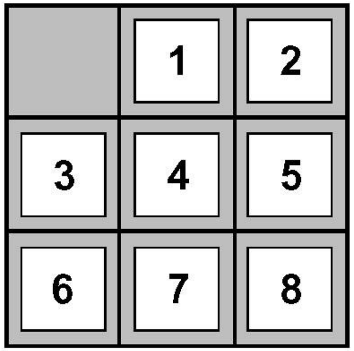
- 8–Puzzle

h_1: The number of misplaced tiles (squares with number).

h_2: The sum of the distances of the tiles from their goal positions.



Start State



Goal State

Approach 3: *f* measures the total cost of the solution path (With Admissible Heuristic Functions)

- A heuristic function is admissible if *h(n)* **never** overestimates the cost to reach the goal.
  - Admissible heuristics are “optimistic”: “the cost is not that much …”
- However, *g(n)* is the exact cost to reach node *n* from the initial state.
- Therefore, *f(n)* never over-estimate the true cost to reach the goal state through node *n*.
- Theorem: A search is optimal if *h(n)* is admissible.
  - I.e. The search using *h(n)* returns an optimal solution.
- Given $h_2(n) > h_1(n)$ for all *n,* it’s always more <u>efficient</u> to use $h_2(n)$.
  - h_2 is more realistic than h_1 *(more informed)*, though both are optimistic.

Traditional informed search strategies

- Greedy Best first search
  - "Always chooses the successor node with the best *f* value" where $f(n) = h(n)$
  - We choose the one that is nearest to the final state among all possible choices

- A* search
  - Best first search using an evaluation function *f* that takes into account the "admissible" heuristic function *h* and the current cost *g*
  - Always returns the optimal solution path

Informed Search Strategies

Best First Search

An implementation of Best First Search

function BEST-FIRST-SEARCH (*problem*, *eval-fn*)
returns a solution sequence, or failure

queuing-fn = a function that sorts nodes by *eval-fn*

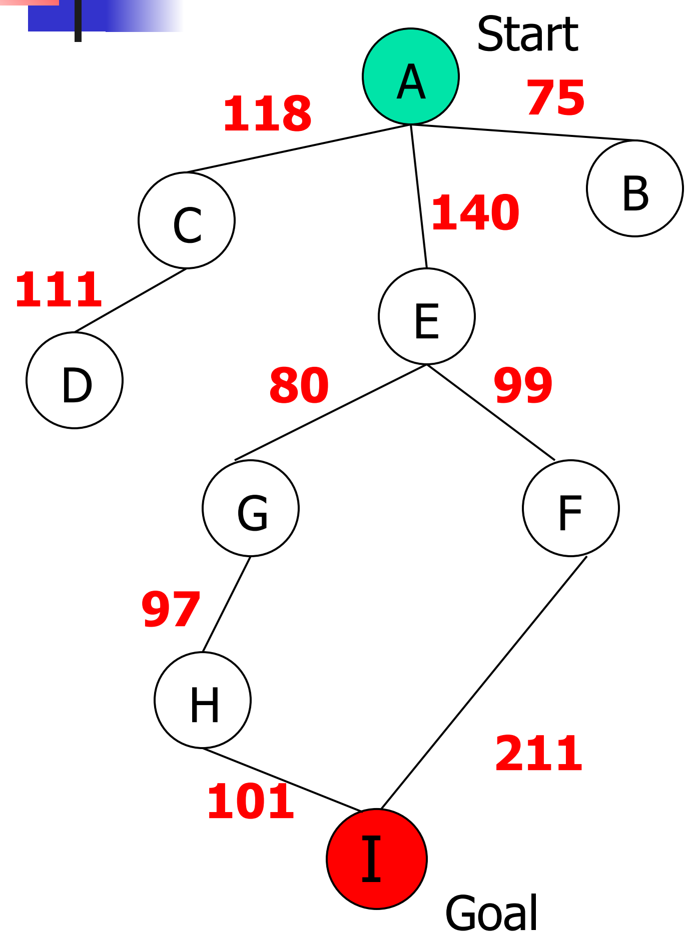
return GENERIC-SEARCH (*problem*,*queuing-fn*)

Informed Search Strategies

Greedy Search

eval-fn: f(n) = h(n)

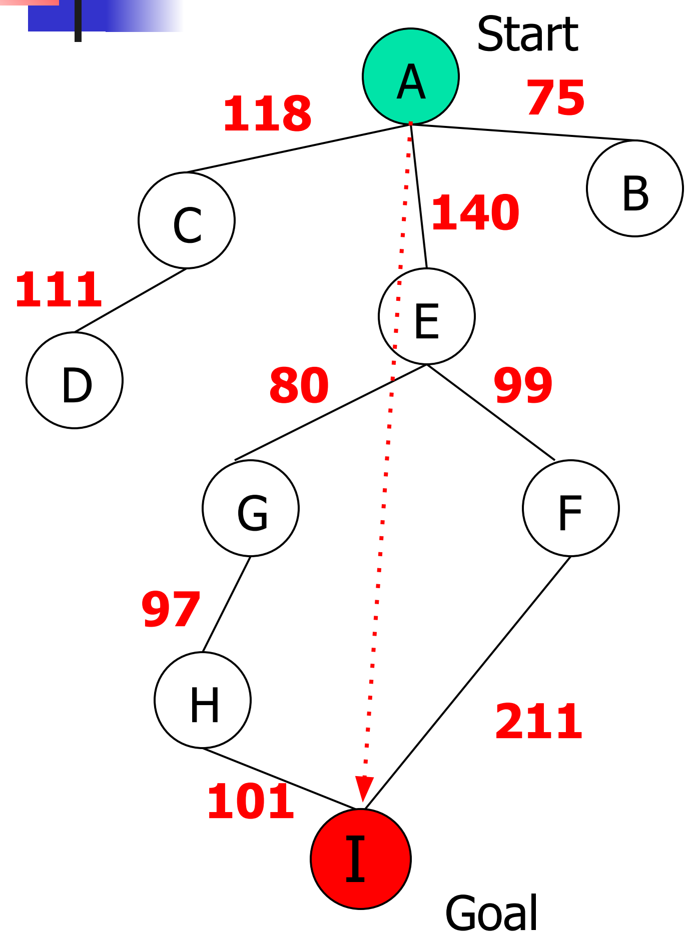
Greedy Search



State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

$f(n) = h\ (n)$ = **straight-line distance heuristic**

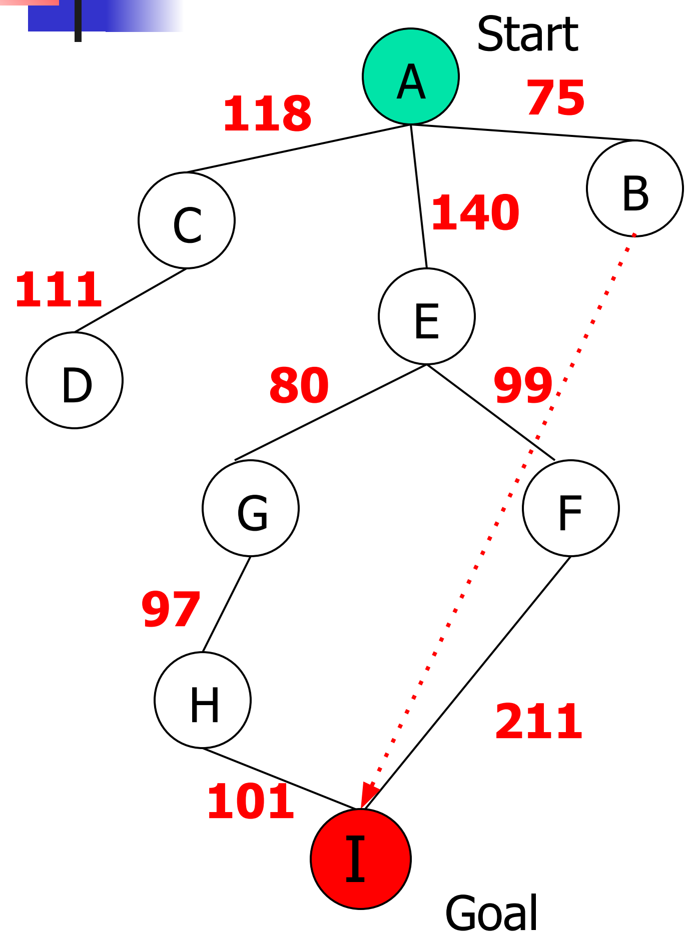
Greedy Search



State	Heuristic: h(n)
A	**366**
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

f(n) = h (n) = straight-line distance heuristic

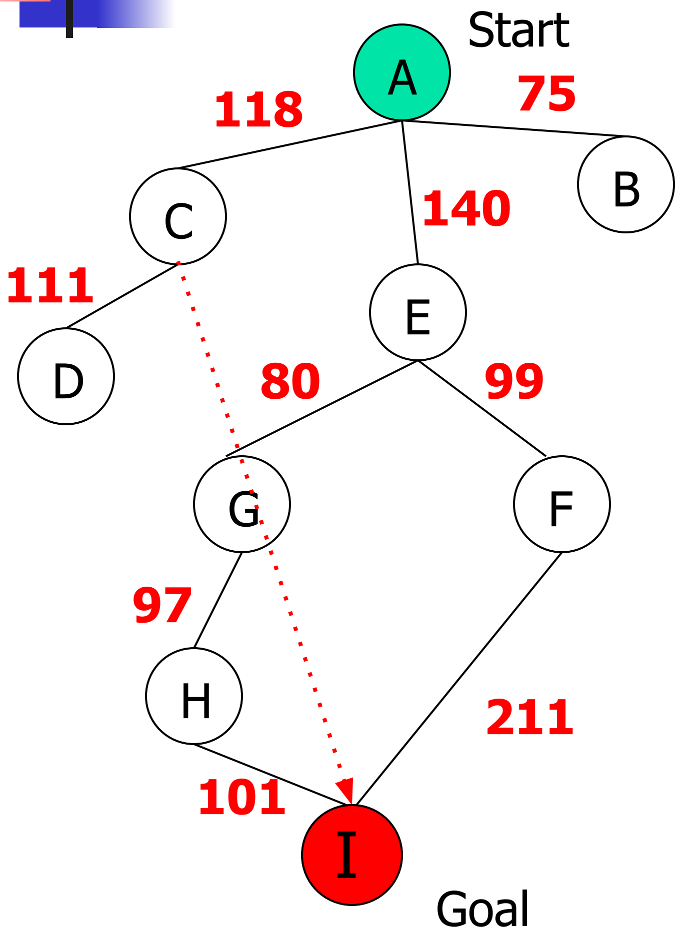
Greedy Search



State	Heuristic: h(n)
A	366
B	**374**
C	329
D	244
E	253
F	178
G	193
H	98
I	0

***f(n) = h (n)* = straight-line distance heuristic**

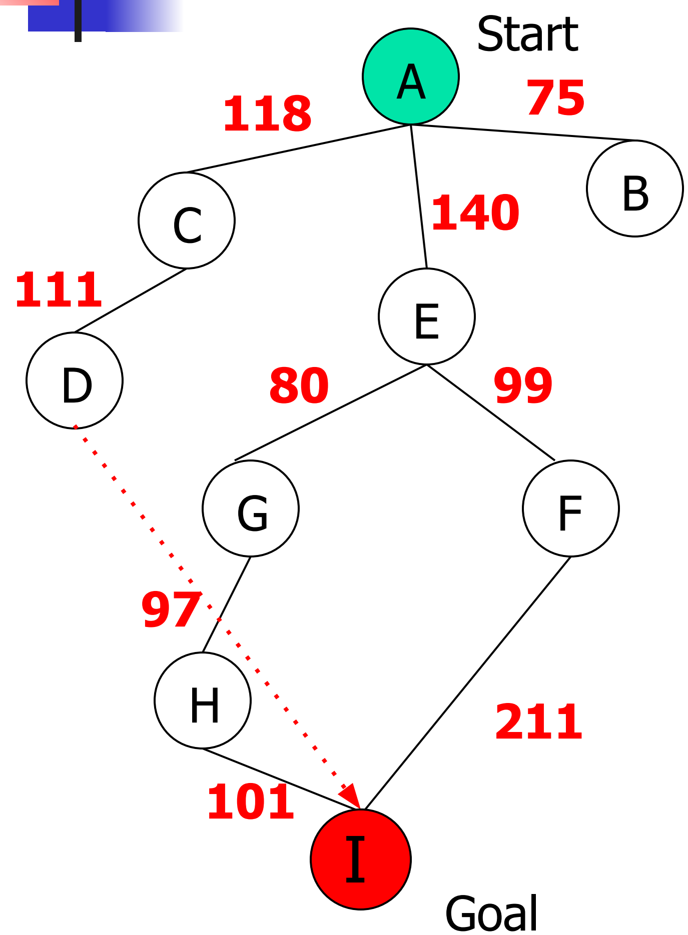
Greedy Search



State	Heuristic: h(n)
A	366
B	374
C	**329**
D	244
E	253
F	178
G	193
H	98
I	0

***f(n) = h (n)* = straight-line distance heuristic**

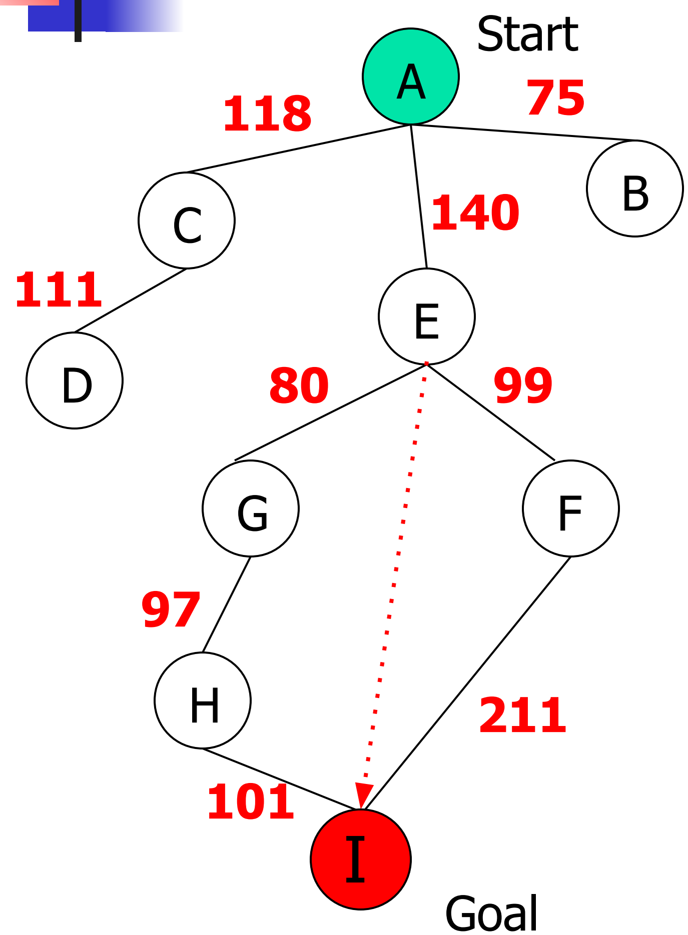
Greedy Search



State	Heuristic: h(n)
A	366
B	374
C	329
D	**244**
E	253
F	178
G	193
H	98
I	0

***f(n) = h (n)* = straight-line distance heuristic**

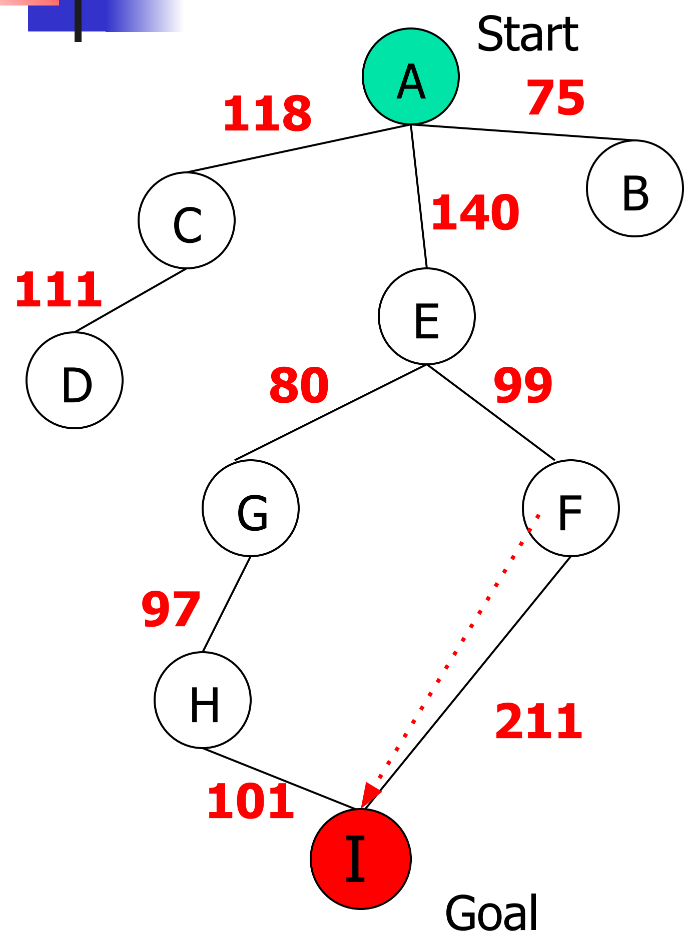
Greedy Search



State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	**253**
F	178
G	193
H	98
I	0

$f(n) = h\ (n)$ = **straight-line distance heuristic**

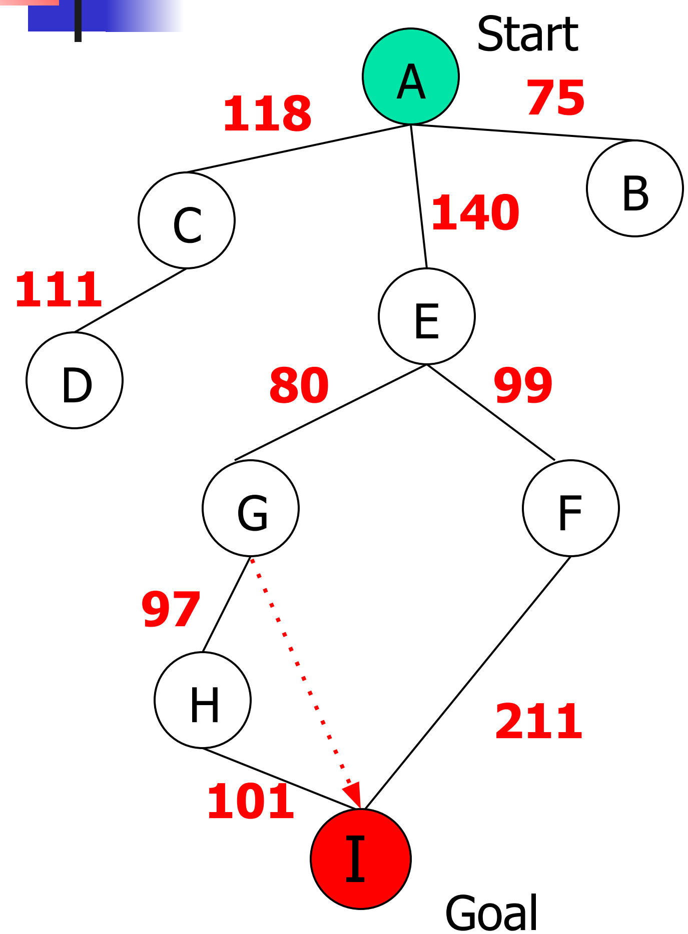
Greedy Search



State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	**178**
G	193
H	98
I	0

***f(n) = h (n)* = straight-line distance heuristic**

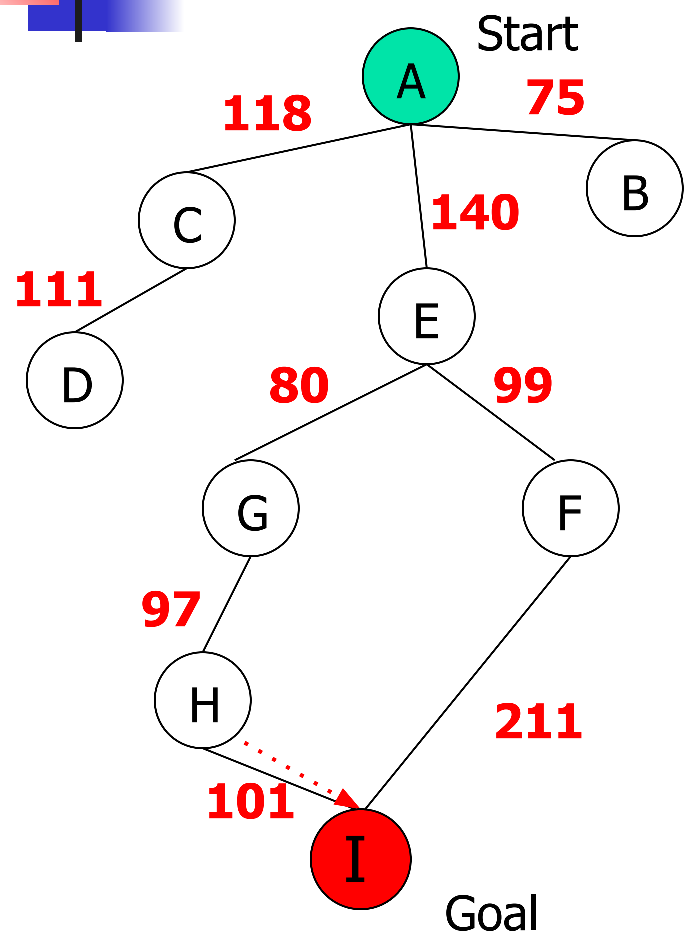
Greedy Search



State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	**193**
H	98
I	0

$f(n) = h\ (n)$ = **straight-line distance heuristic**

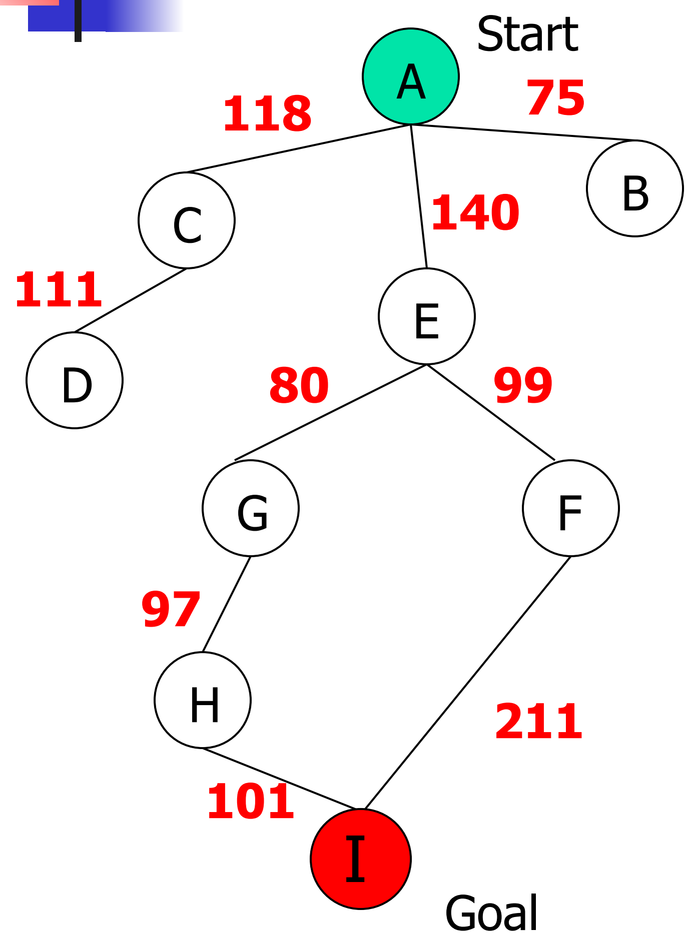
Greedy Search



State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	**98**
I	0

***f(n) = h (n)* = straight-line distance heuristic**

Greedy Search

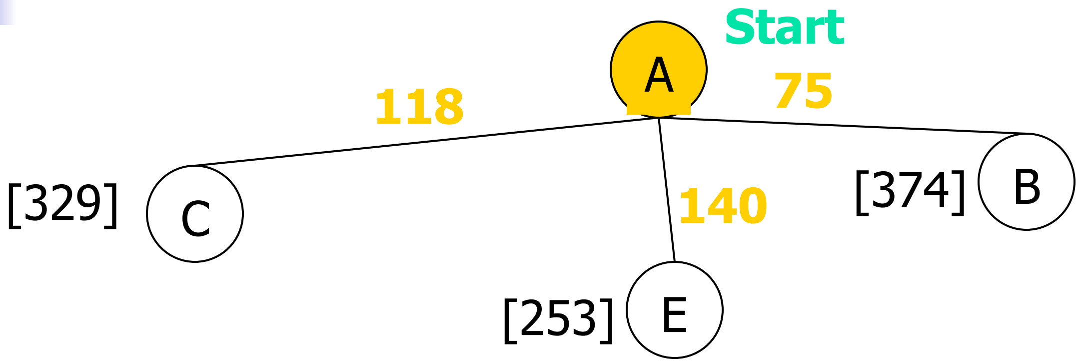


State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	**0**

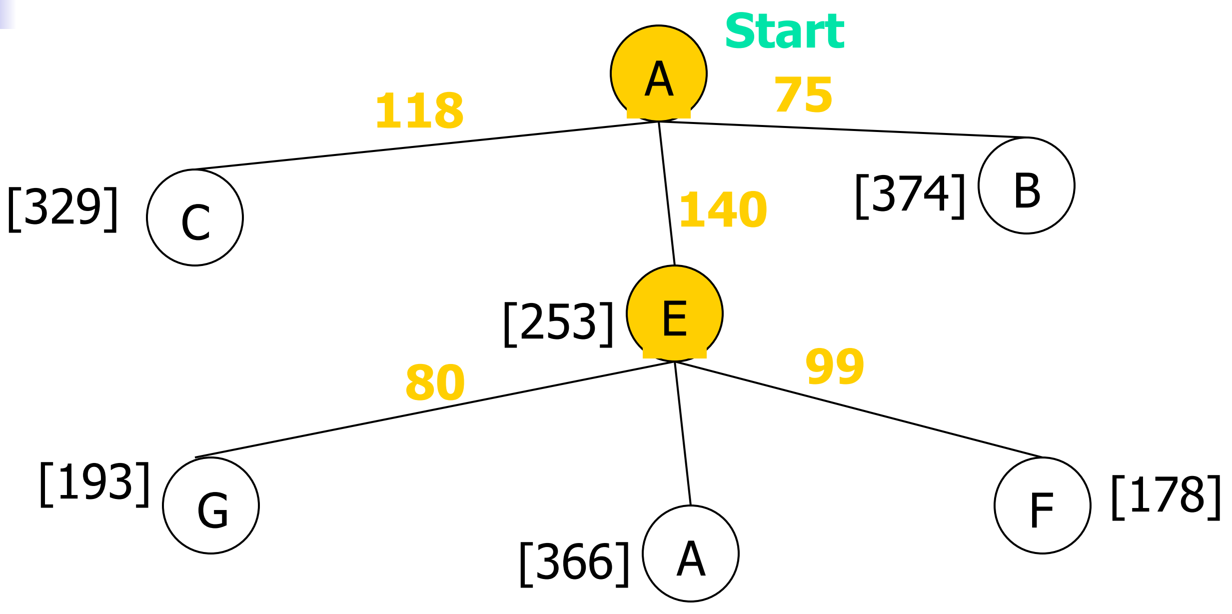
***f(n) = h (n)* = straight-line distance heuristic**

Greedy Search: Tree Search

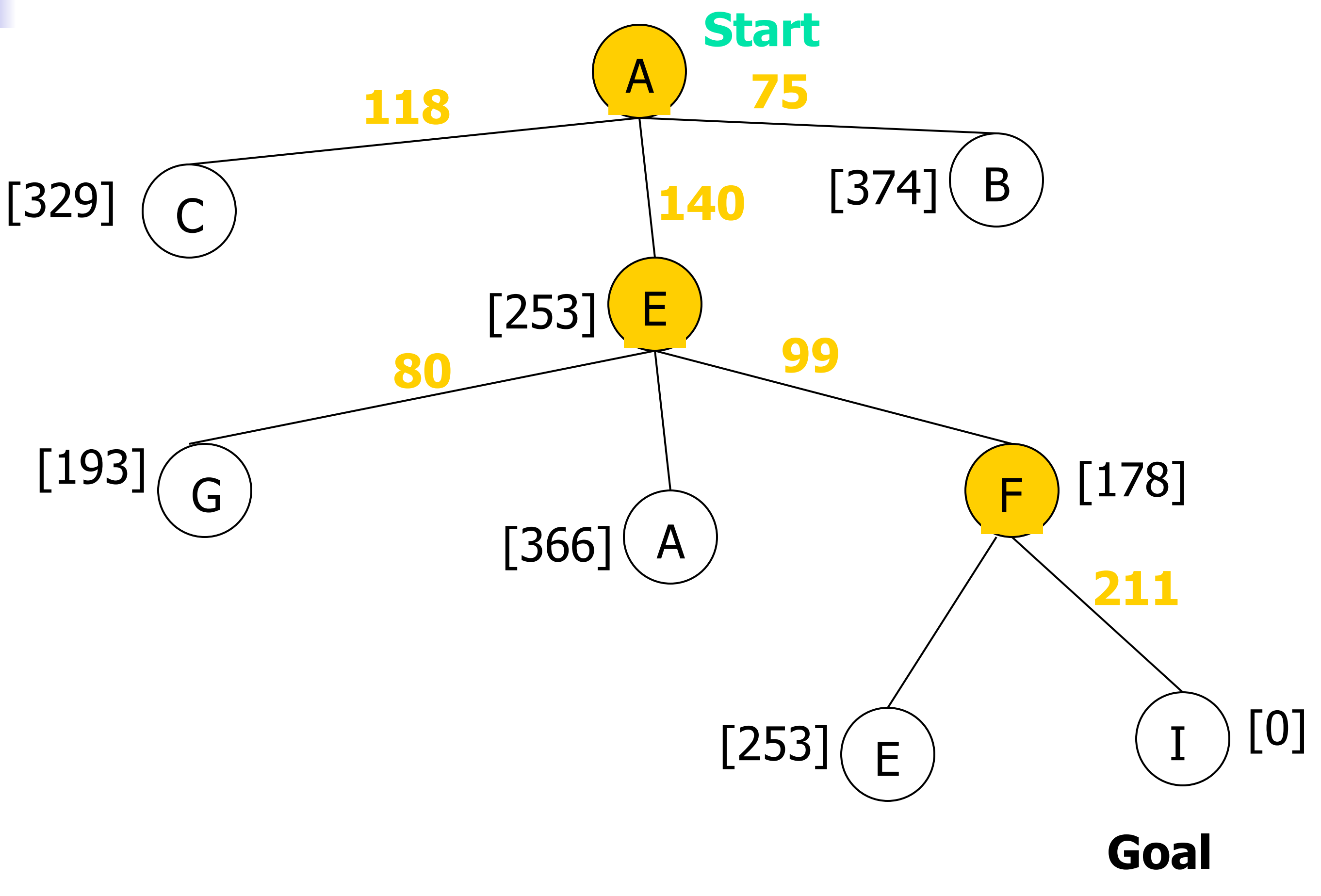
Greedy Search: Tree Search



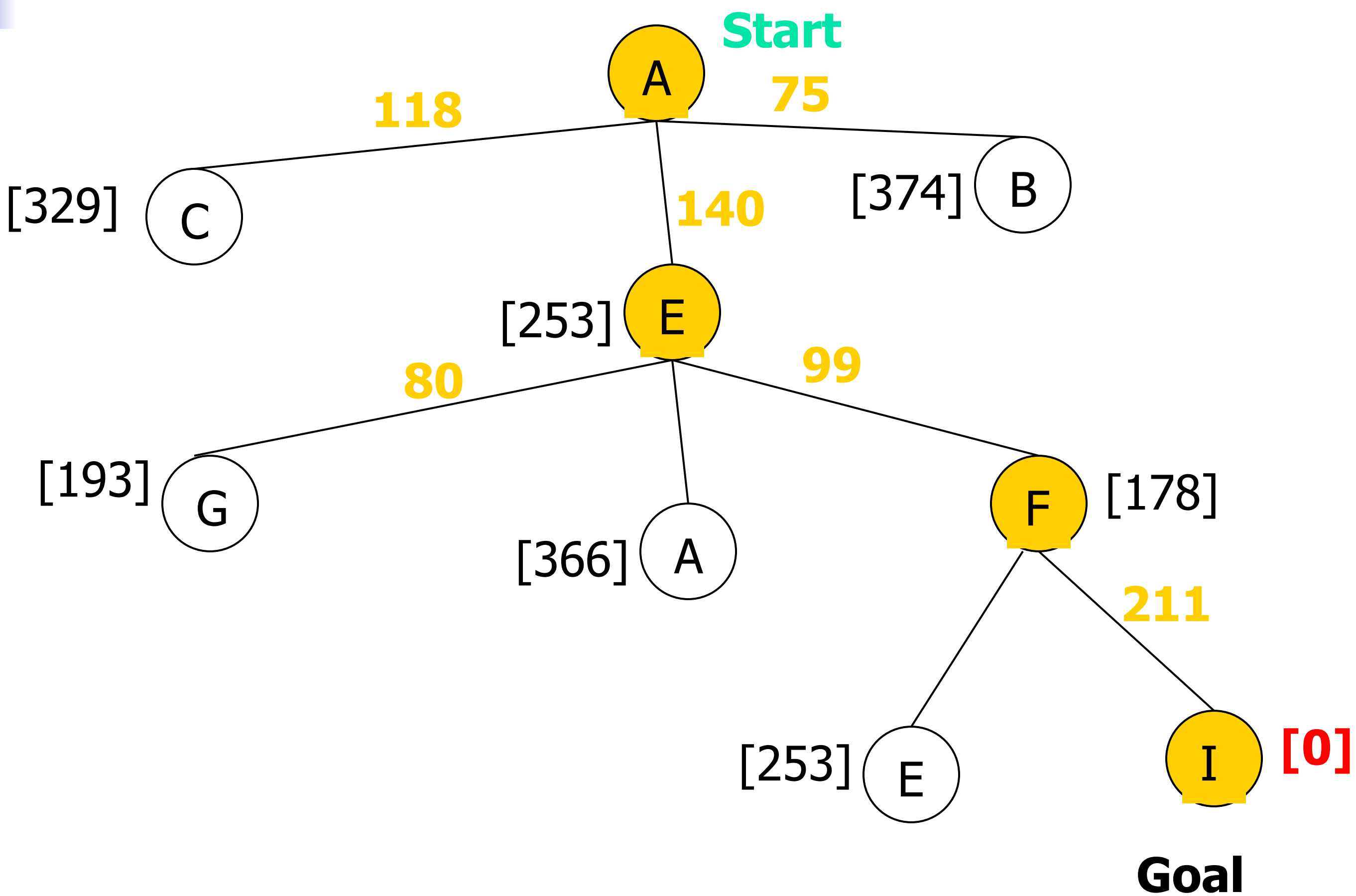
Greedy Search: Tree Search



Greedy Search: Tree Search



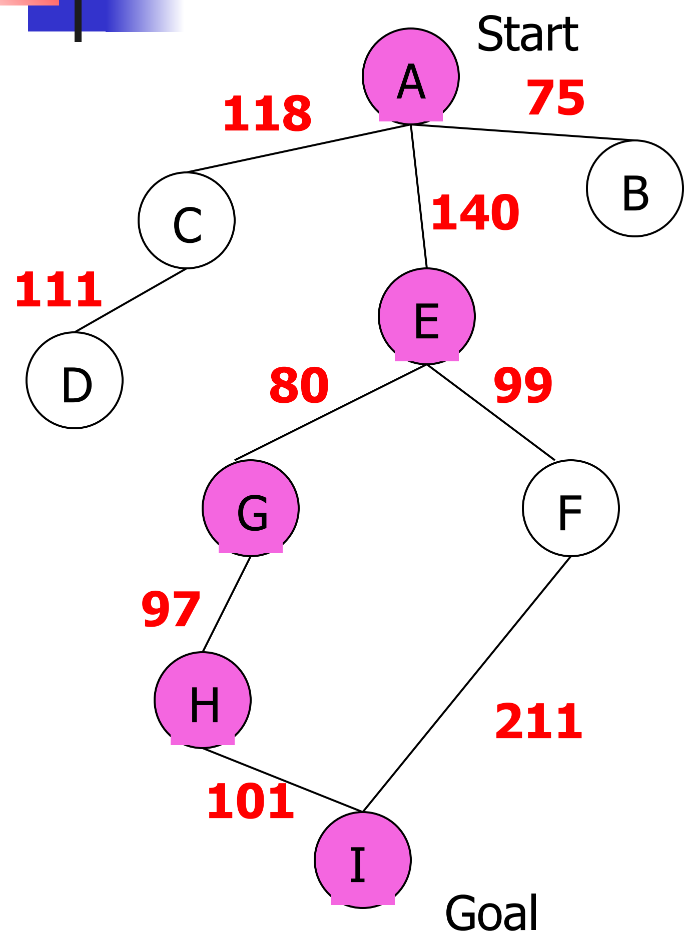
Greedy Search: Tree Search



Heuristic(A-E-F-I) = 253 + 178 + 0 = 431

dist(A-E-F-I) = 140 + 99 + 211 = 450

Greedy Search: Optimal ?

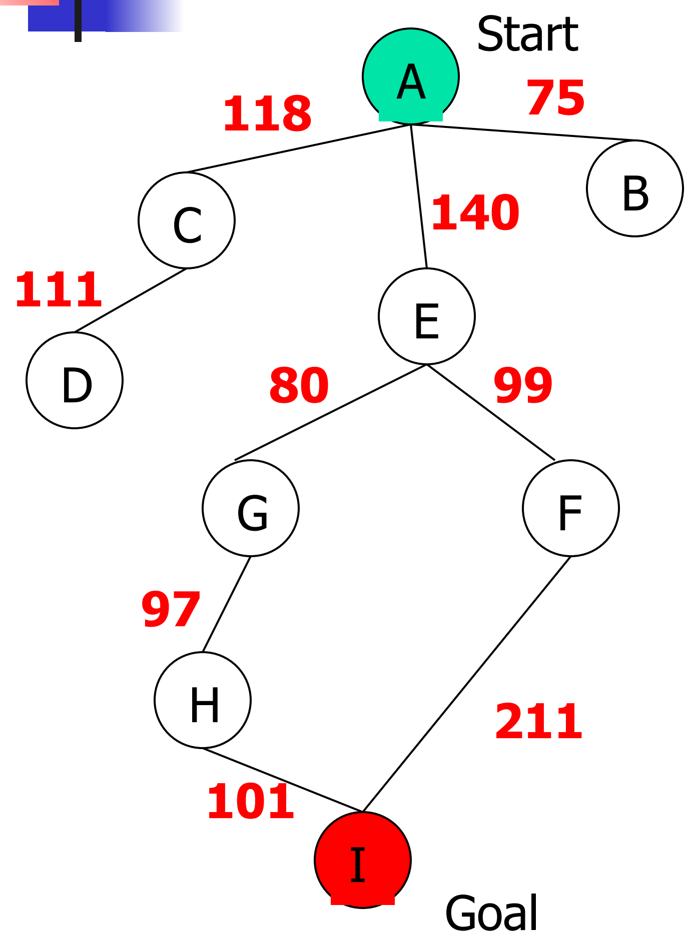


State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	**253**
F	178
G	**193**
H	**98**
I	0

***f(n) = h (n)* = straight-line distance heuristic**

dist(A-E-G-H-I) =140+80+97+101=418

Greedy Search: Complete ?

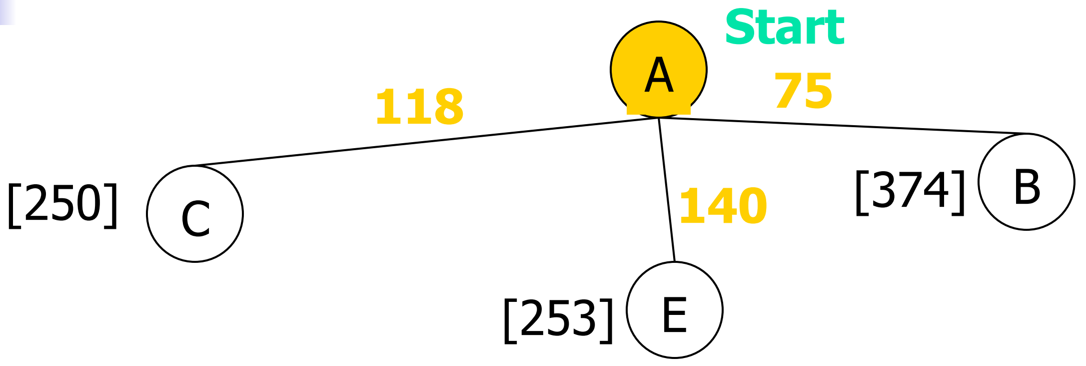


State	Heuristic: h(n)
A	366
B	374
** C	**250**
D	244
E	253
F	178
G	193
H	98
I	0

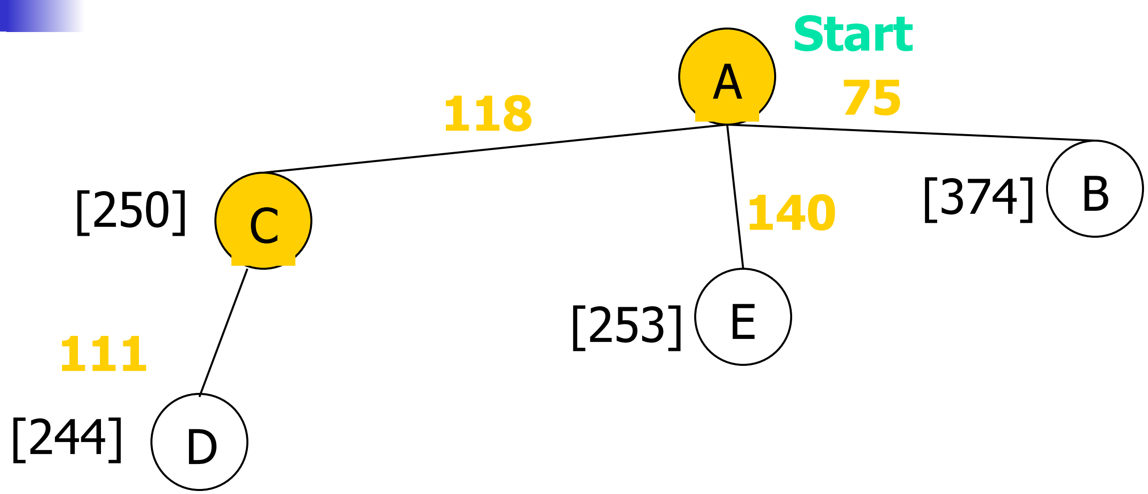
***f(n) = h (n)* = straight-line distance heuristic**

Greedy Search: Tree Search

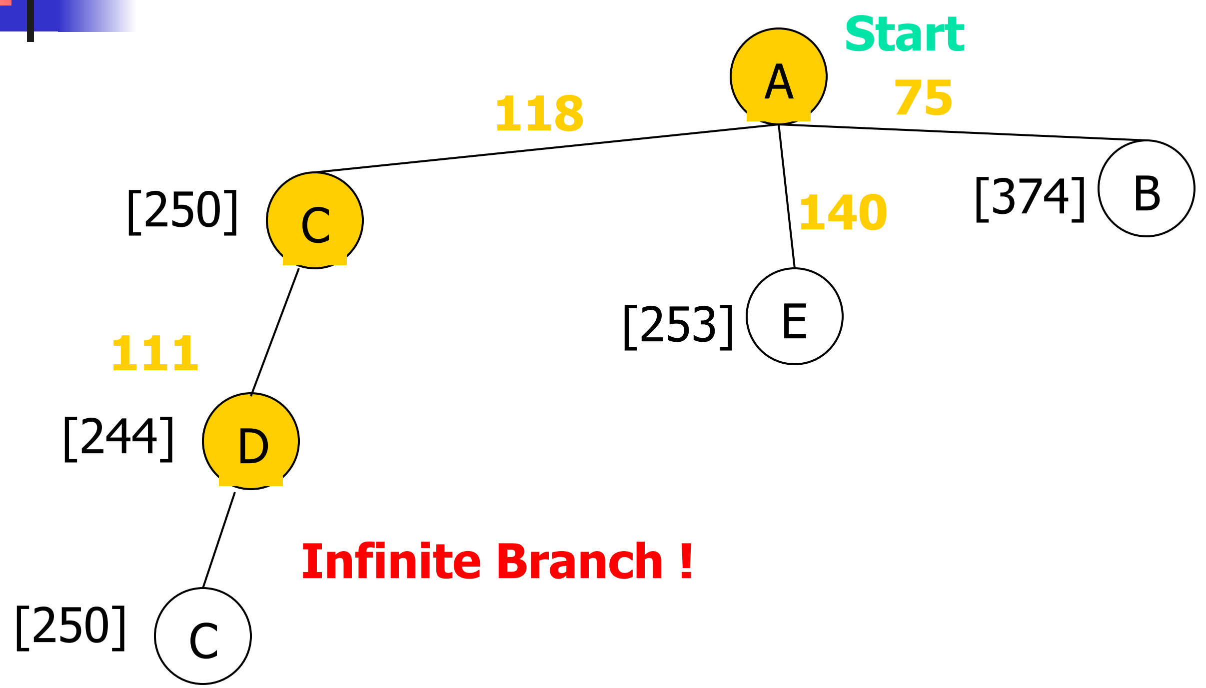
Greedy Search: Tree Search



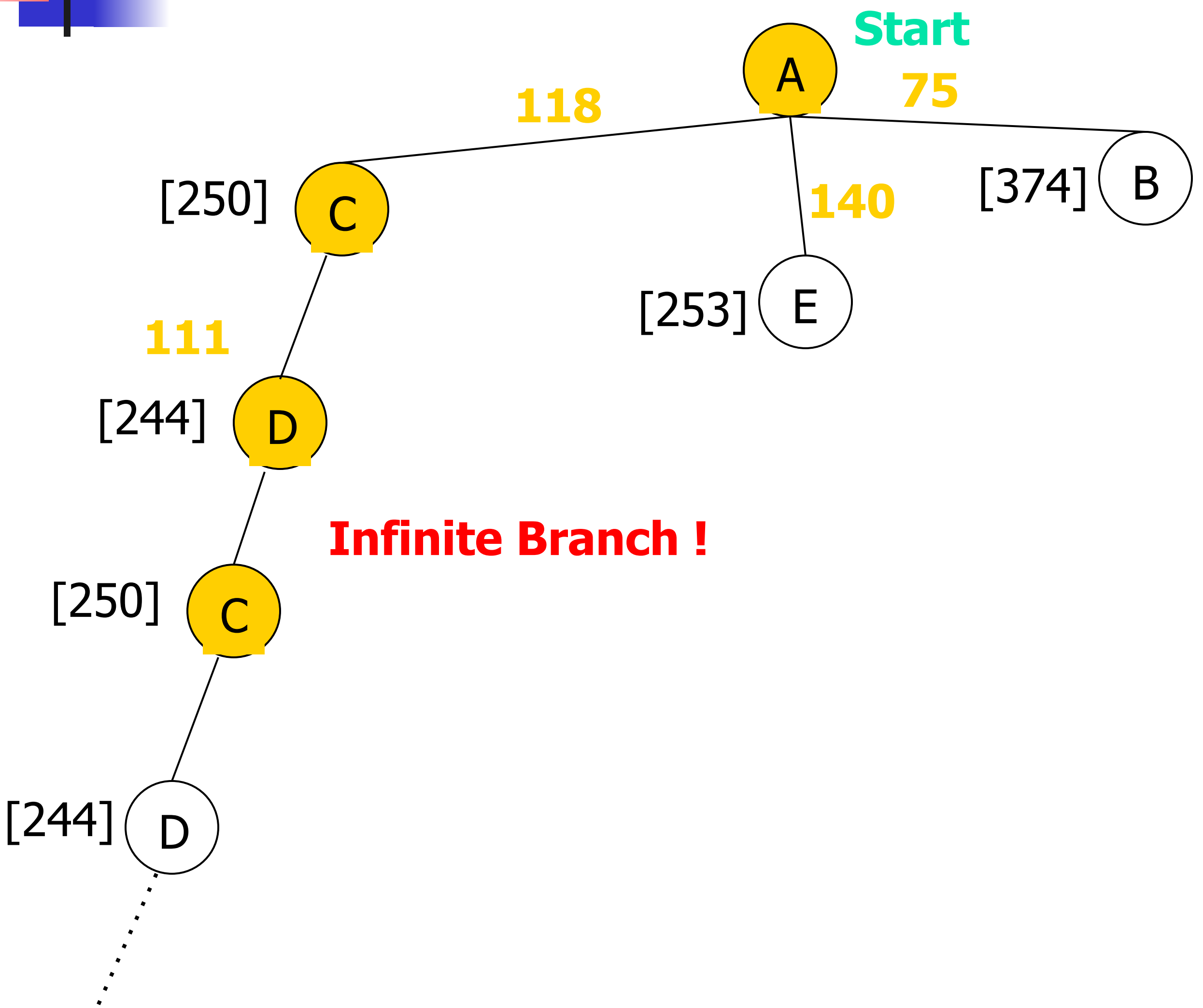
Greedy Search: Tree Search



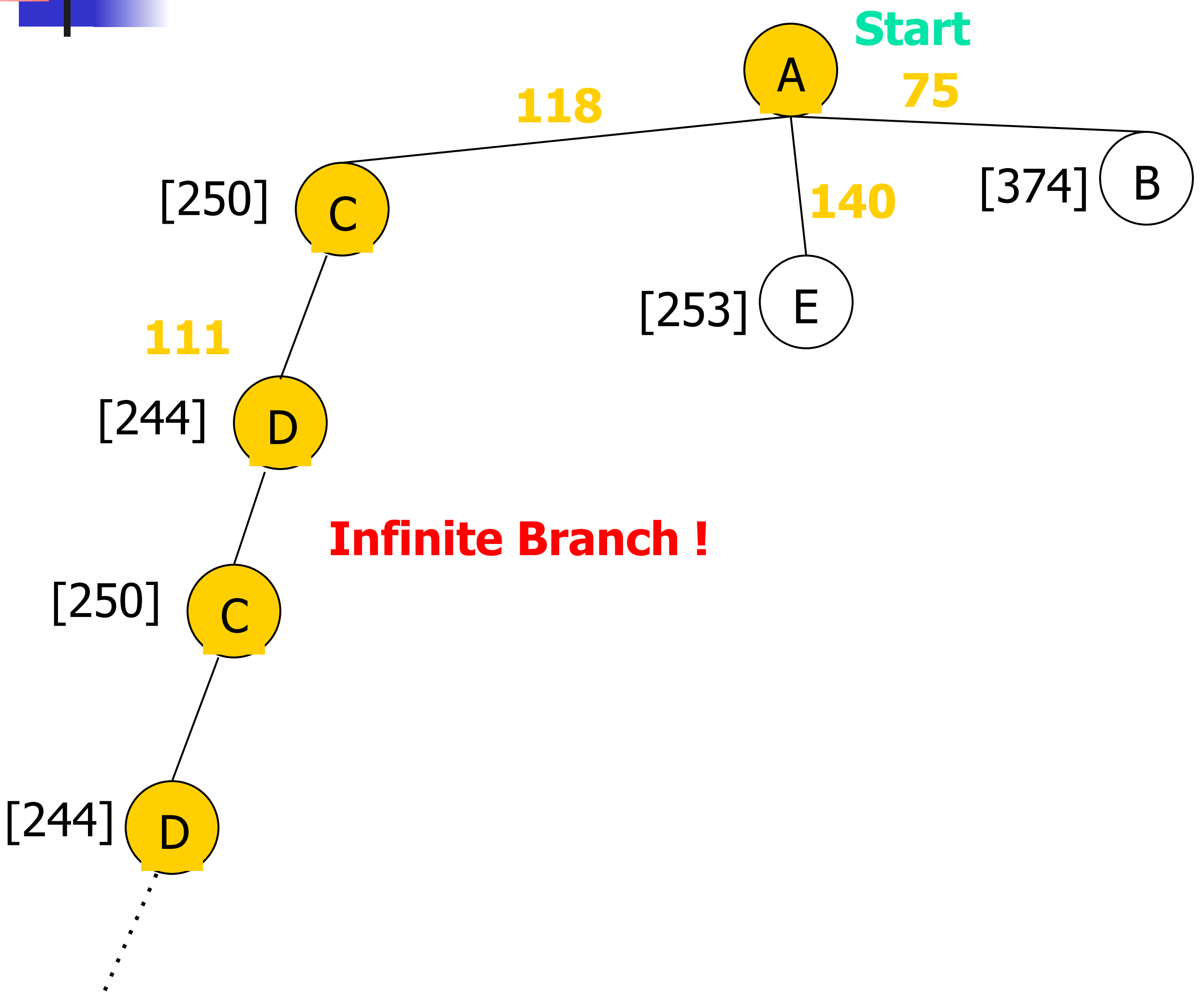
Greedy Search: Tree Search



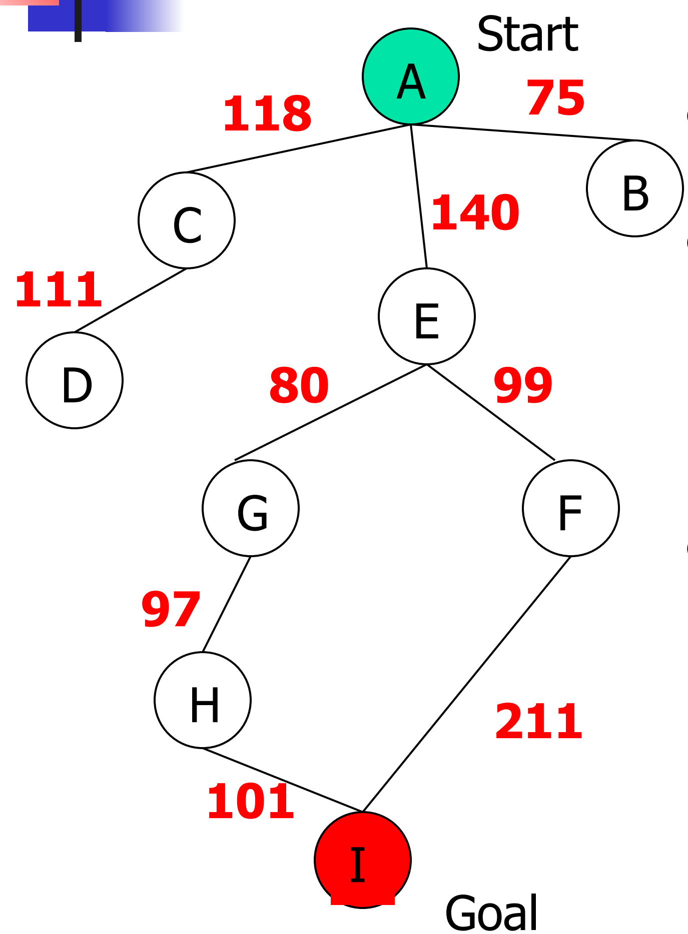
Greedy Search: Tree Search



Greedy Search: Tree Search



Greedy Search: Time and Space Complexity ?



- Greedy search is not optimal.
- Greedy search is incomplete without systematic checking of repeated states.
- In the worst case, the Time and Space Complexity of Greedy Search are both $O(b^m)$

Where b is the branching factor and m the maximum path length

Informed Search Strategies

A* Search

eval-fn: $f(n)=g(n)+h(n)$

A* (A Star)

- Greedy Search minimizes a heuristic h(n) which is an estimated cost from a node n to the goal state. Greedy Search is efficient but it is not optimal nor complete.
- Uniform Cost Search minimizes the cost g(n) from the initial state to n. UCS is optimal and complete but not efficient.
- **New Strategy**: Combine Greedy Search and UCS to get an efficient algorithm which is complete and optimal.

A* (A Star)

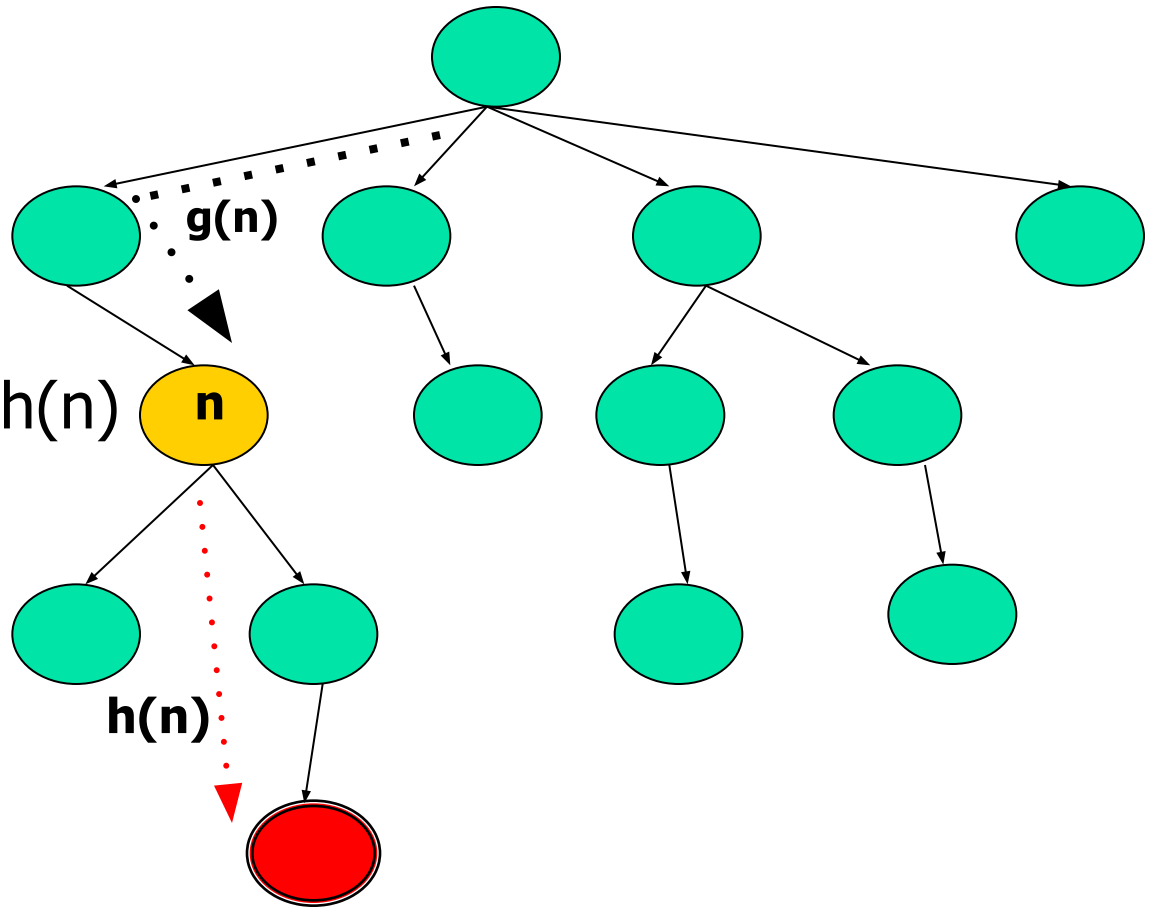
- A* uses an evaluation function which combines g(n) and h(n): $f(n) = g(n) + h(n)$

- **g(n)** is the exact cost to reach node *n* from the initial state.

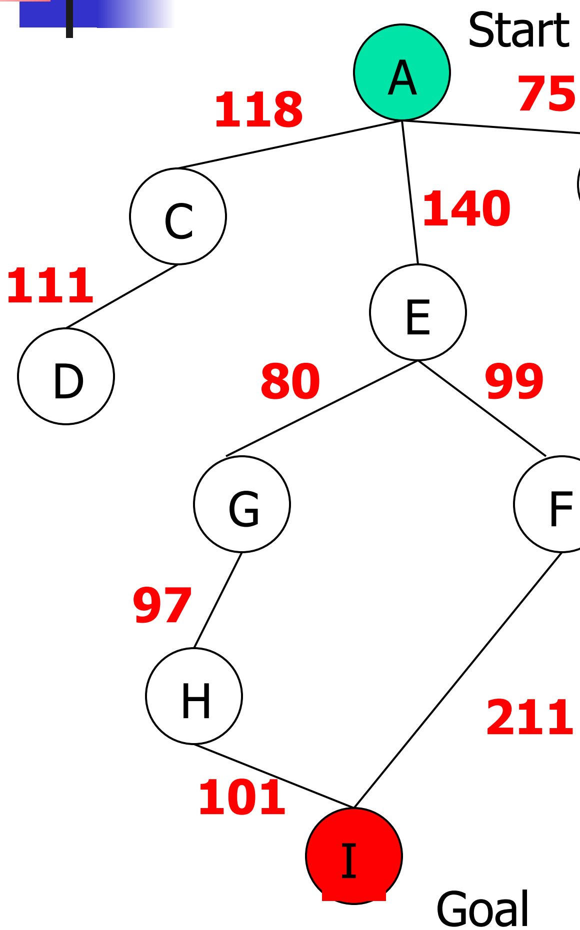
- **h(n)** is an estimation of the remaining cost to reach the goal.

A* (A Star)

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



A* Search



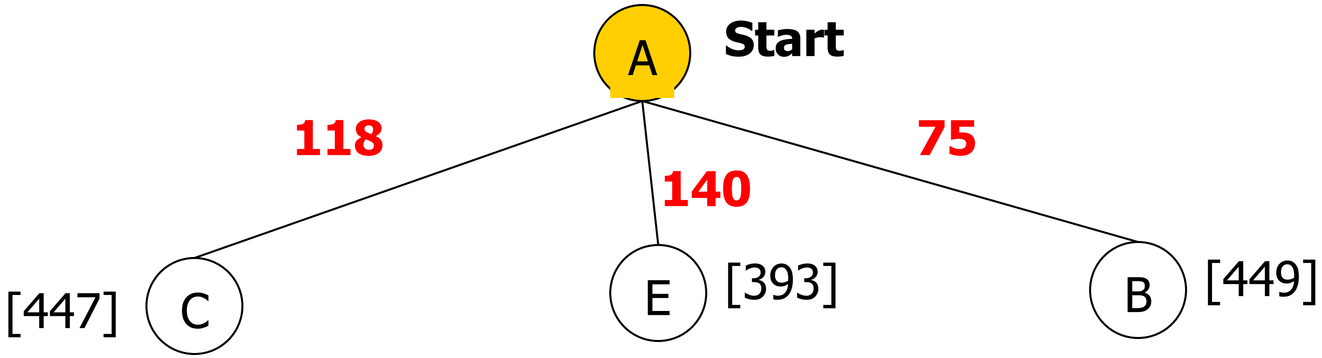
State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

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

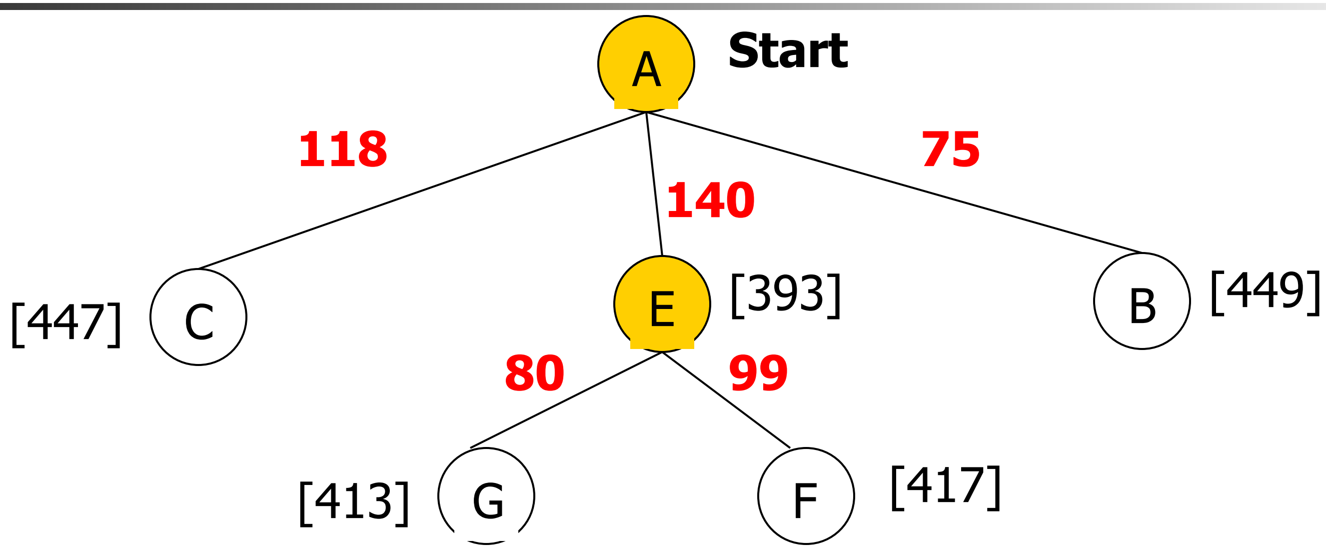
g(n): is the exact cost to reach node *n* from the initial state.

A* Search: Tree Search

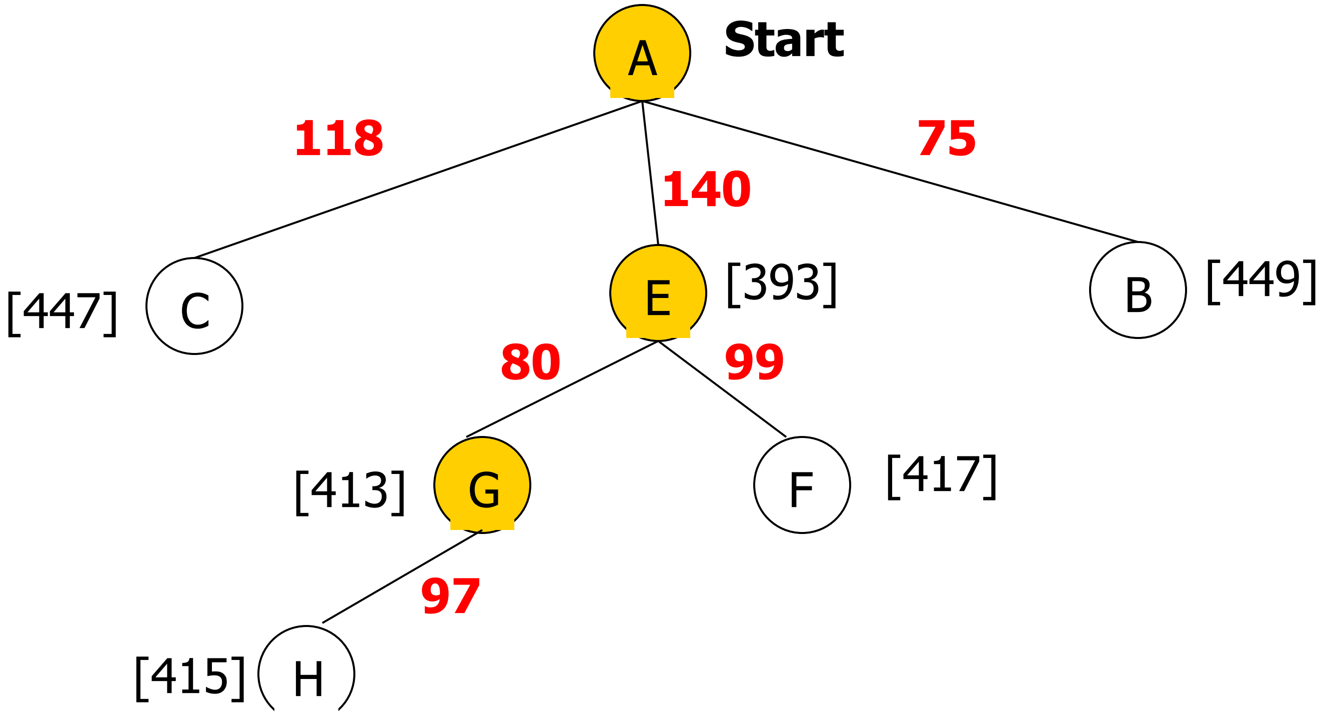
A* Search: Tree Search



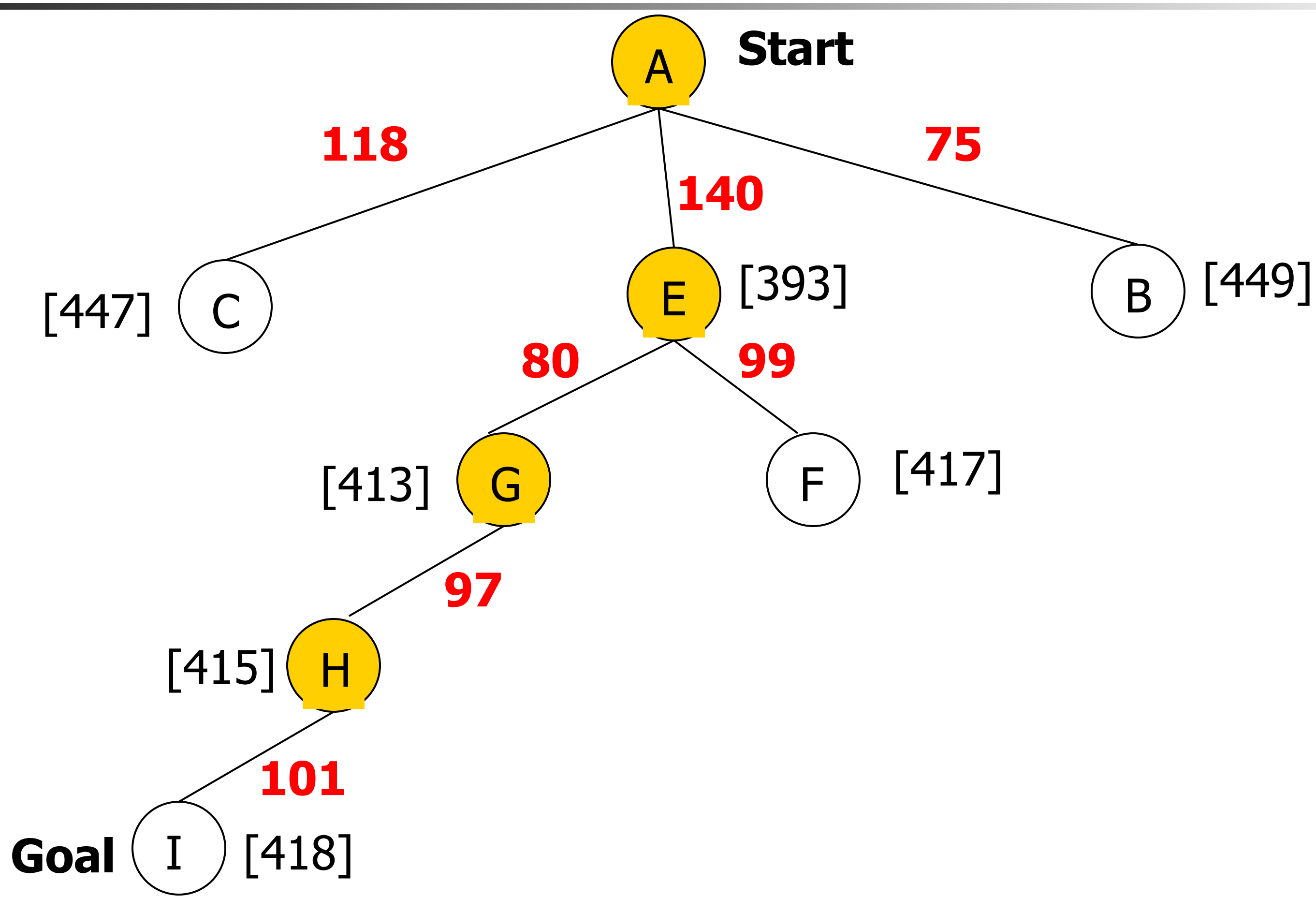
A* Search: Tree Search



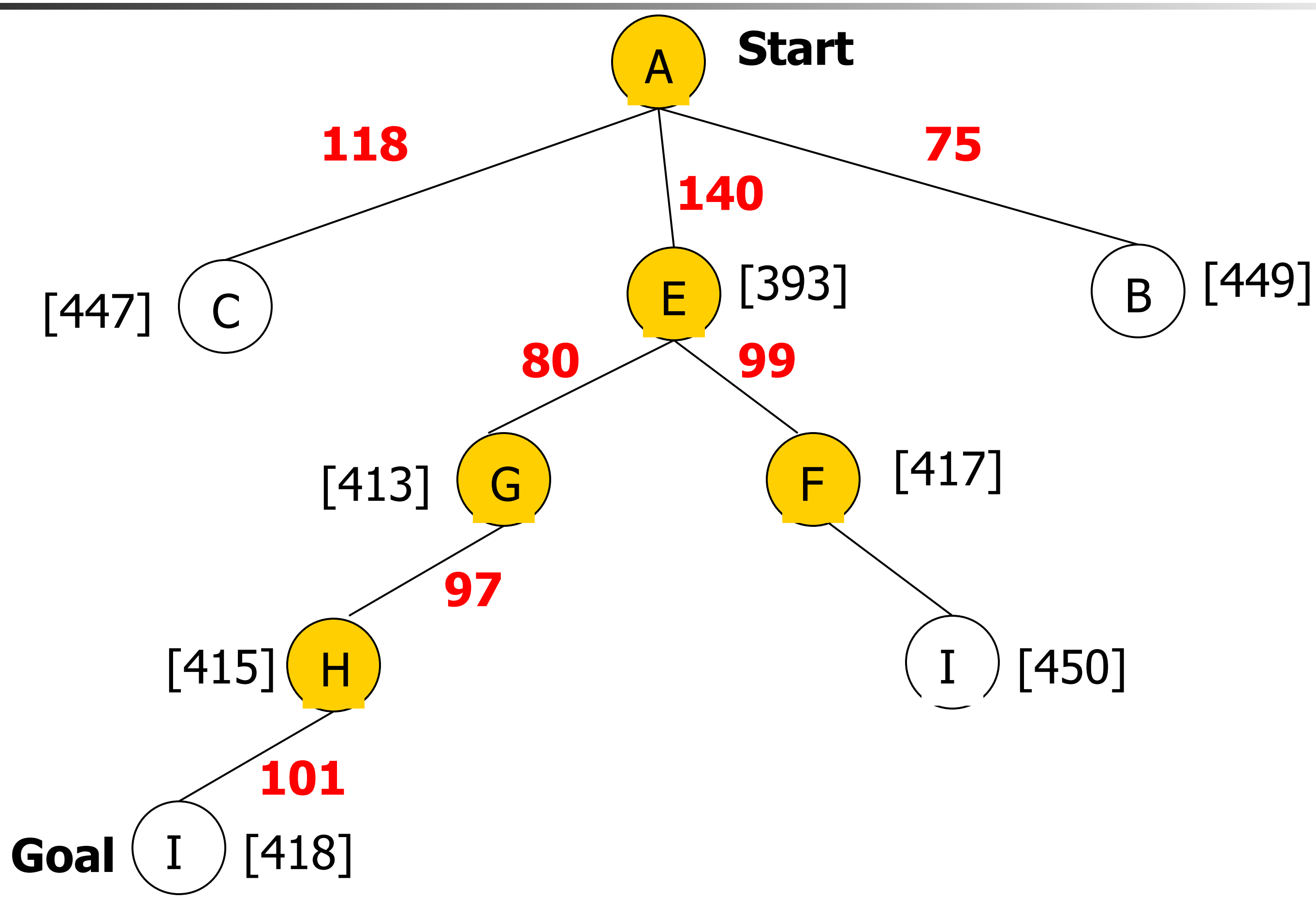
A* Search: Tree Search



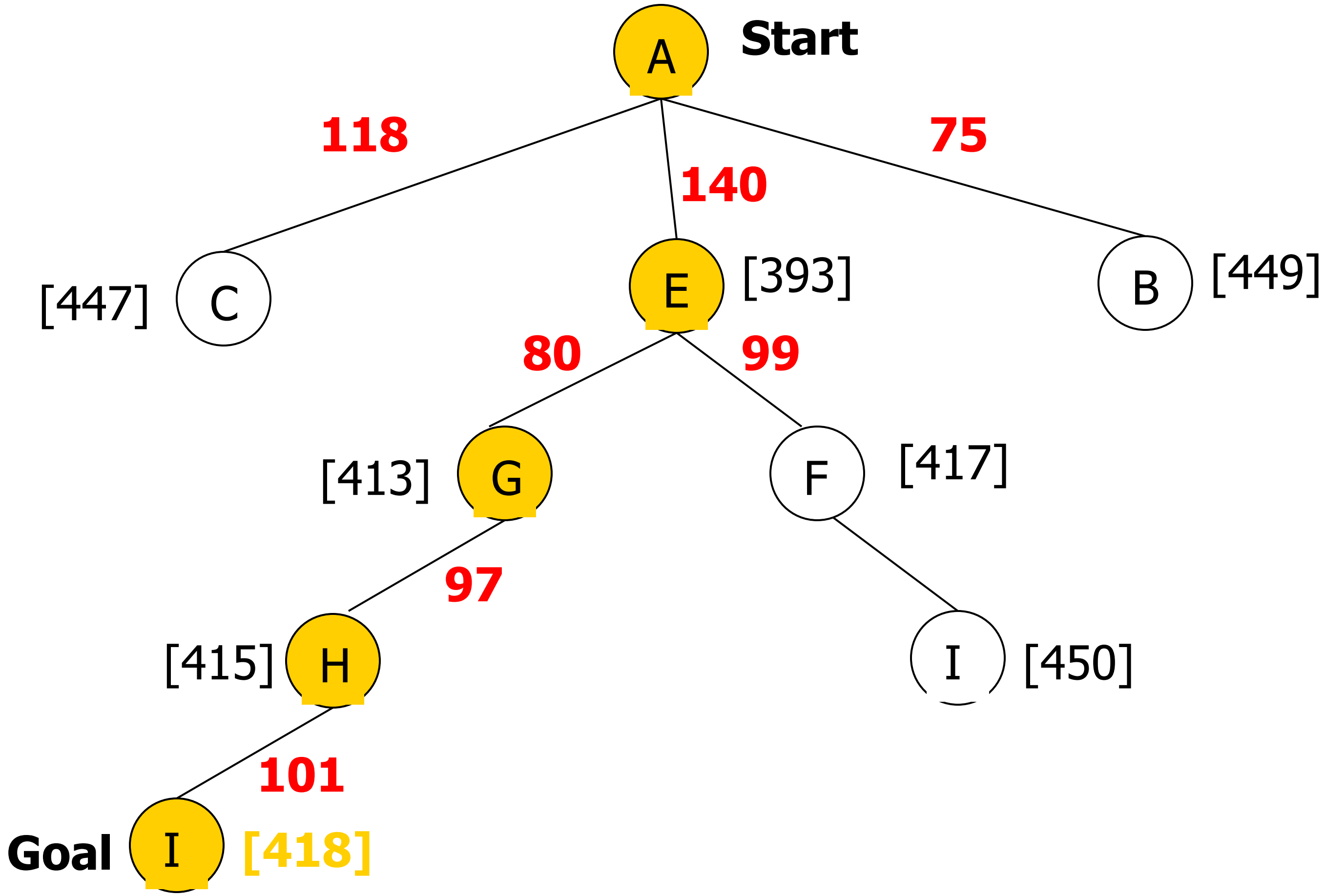
A* Search: Tree Search



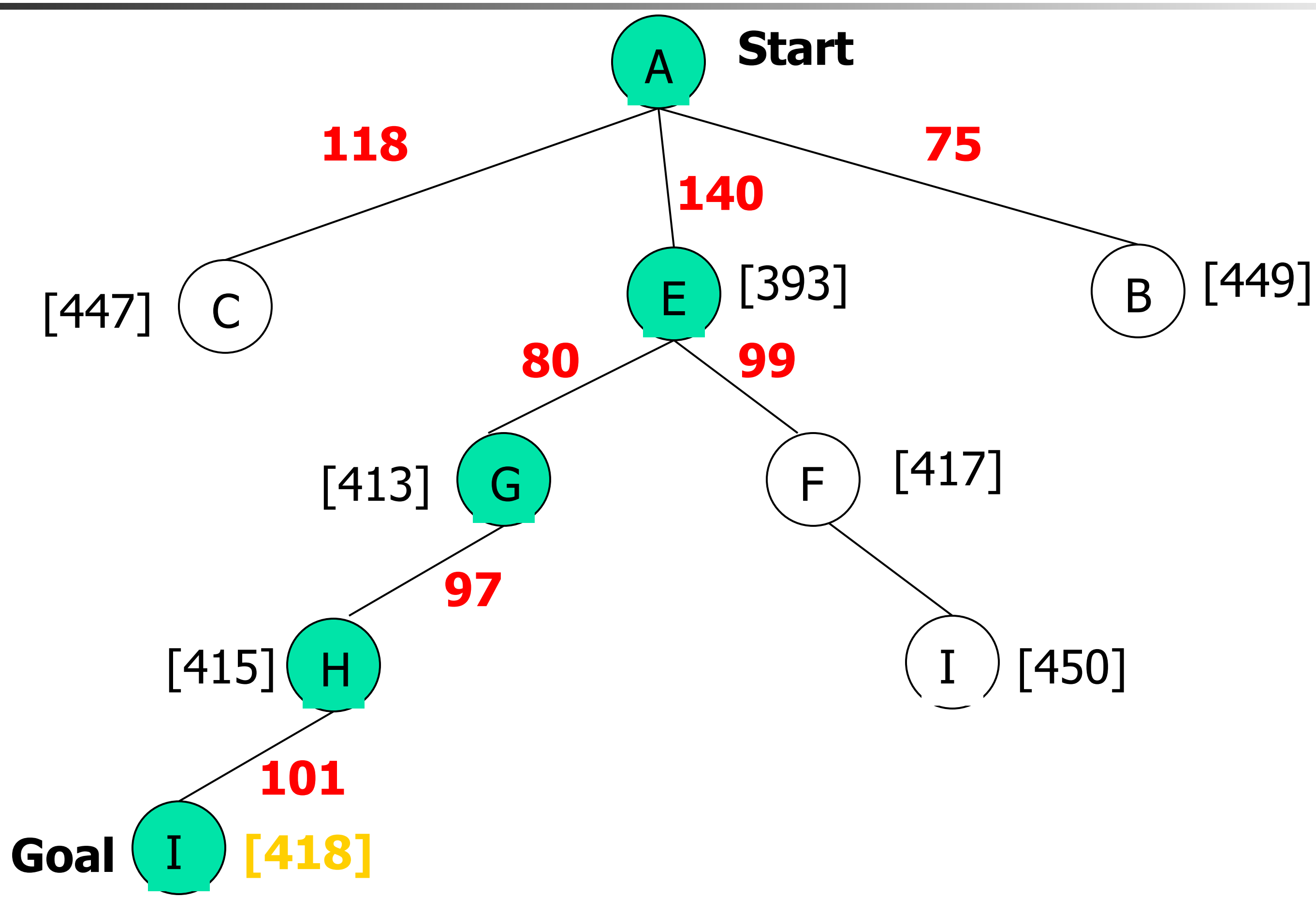
A* Search: Tree Search



A* Search: Tree Search



A* Search: Tree Search



Optimality of A* search

- Tree search of A* is optimal if the heuristic is **admissible**
- Graph search of A* is optimal if the heuristic is **consistent**

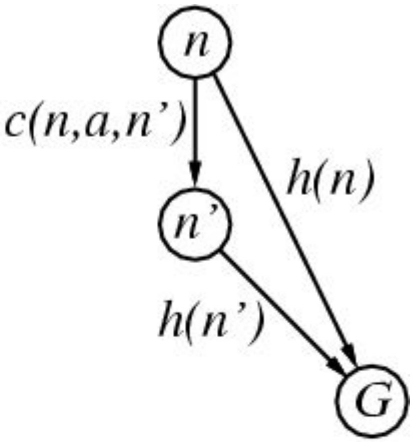
Admissible heuristics

- 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
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If $h(n)$ is admissible, A^* using `TREE-SEARCH` is optimal

Consistent heuristics

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

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



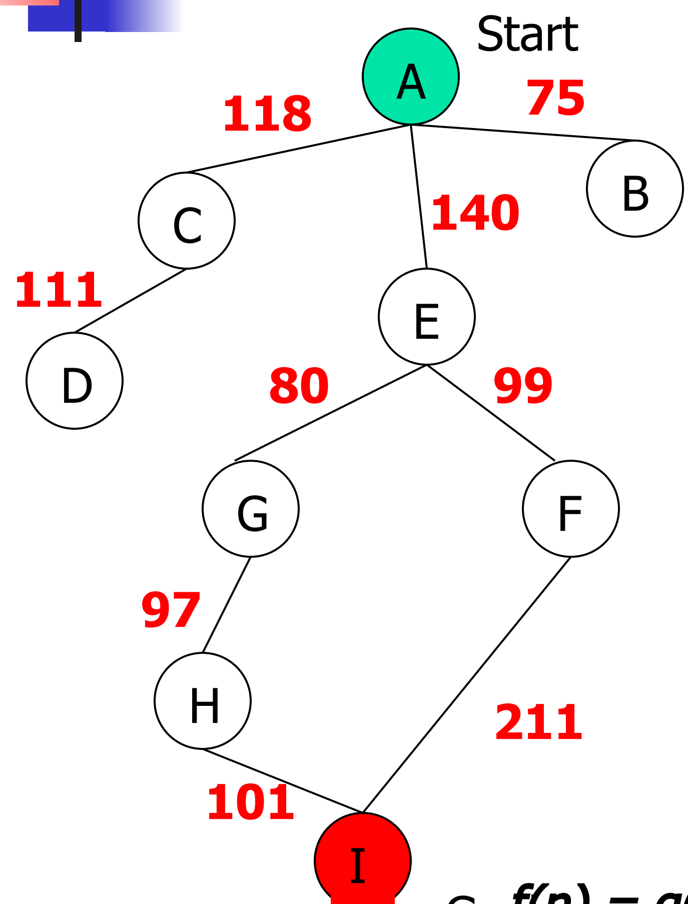
- If *h* is consistent, we have
$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

- i.e., *f(n)* is non-decreasing along any path.
- Theorem: If *h(n)* is consistent, A* using `GRAPH-SEARCH` is optimal

What if heuristic is **NOT** Admissible?

h() overestimates the cost to reach the goal state

A* Search: what if *h* is not admissible !



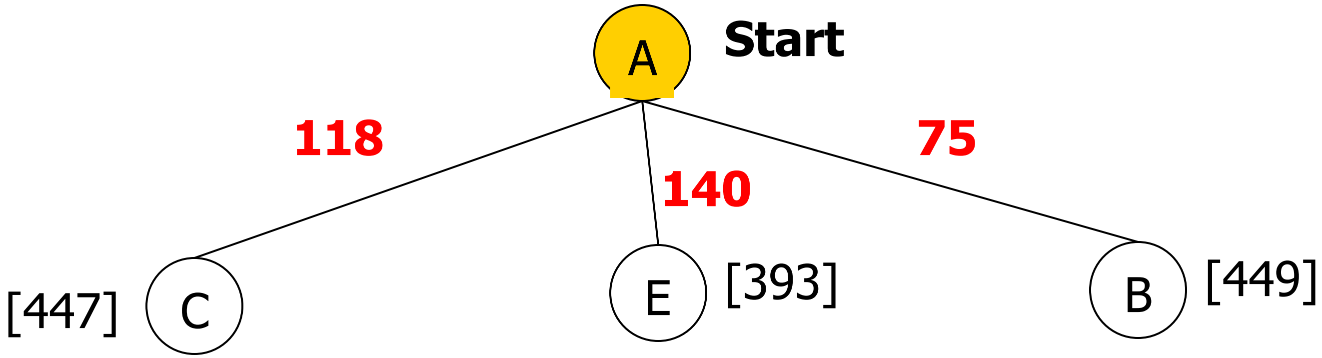
State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	**138**
I	0

f(n) = g(n) + h (n) **– (H-I) Overestimated**

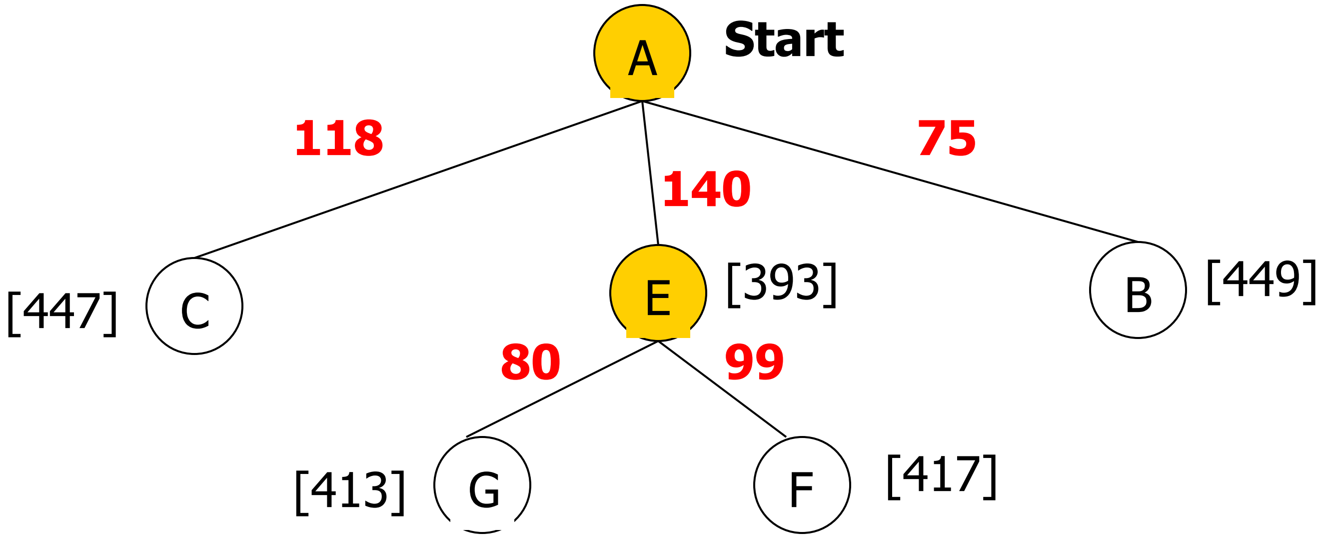
g(n): is the exact cost to reach node *n* from the initial state.

A* Search: Tree Search

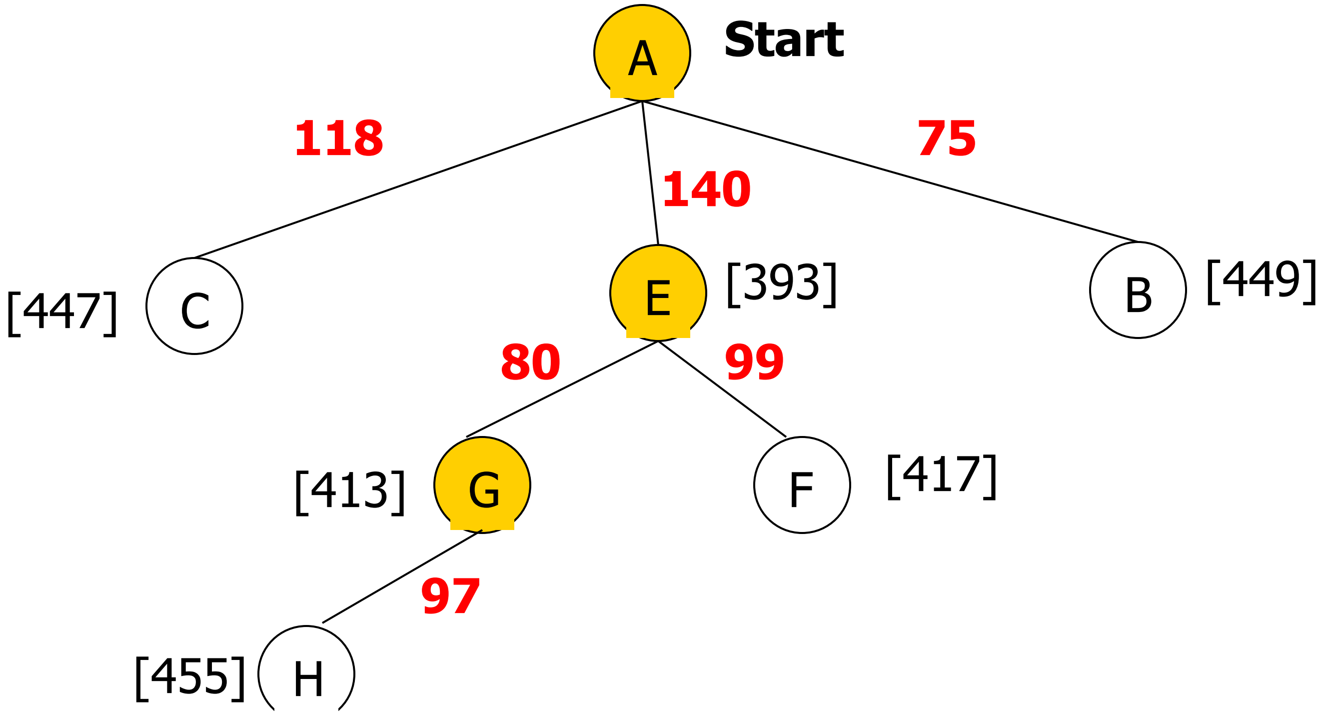
A* Search: Tree Search



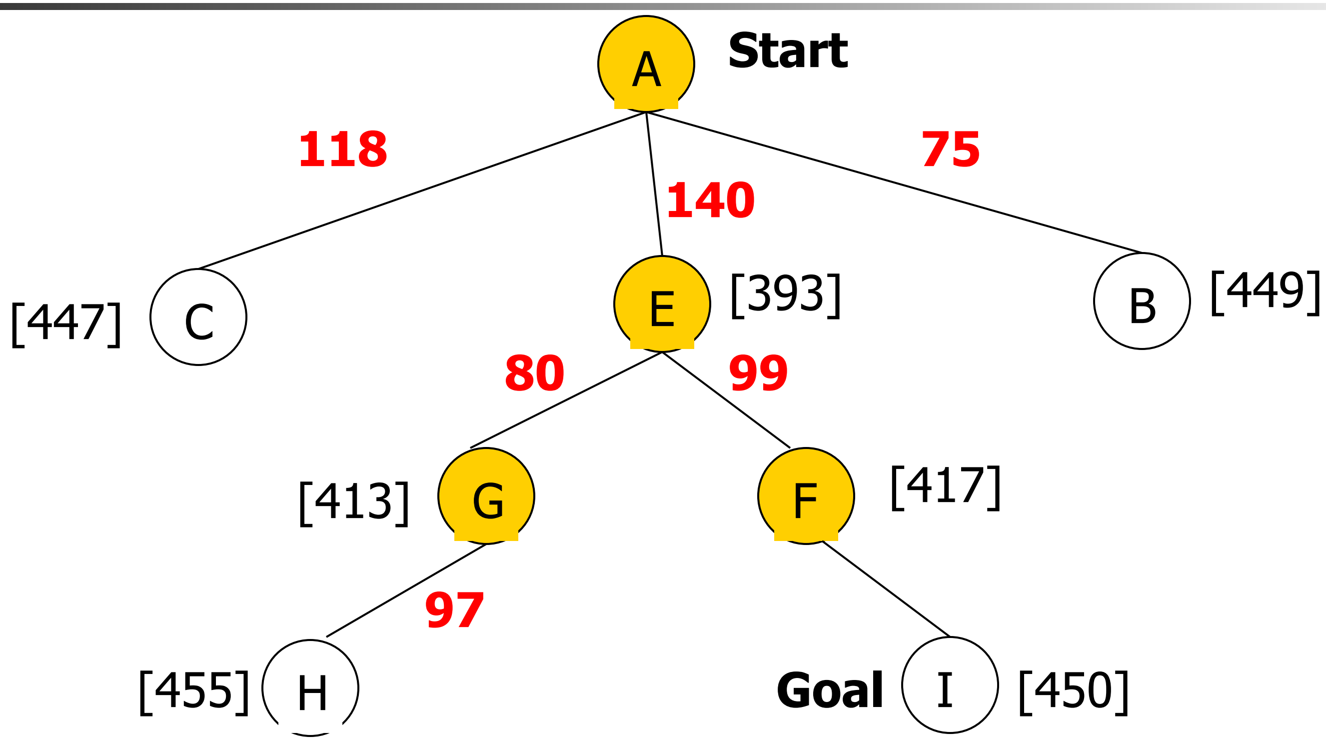
A* Search: Tree Search



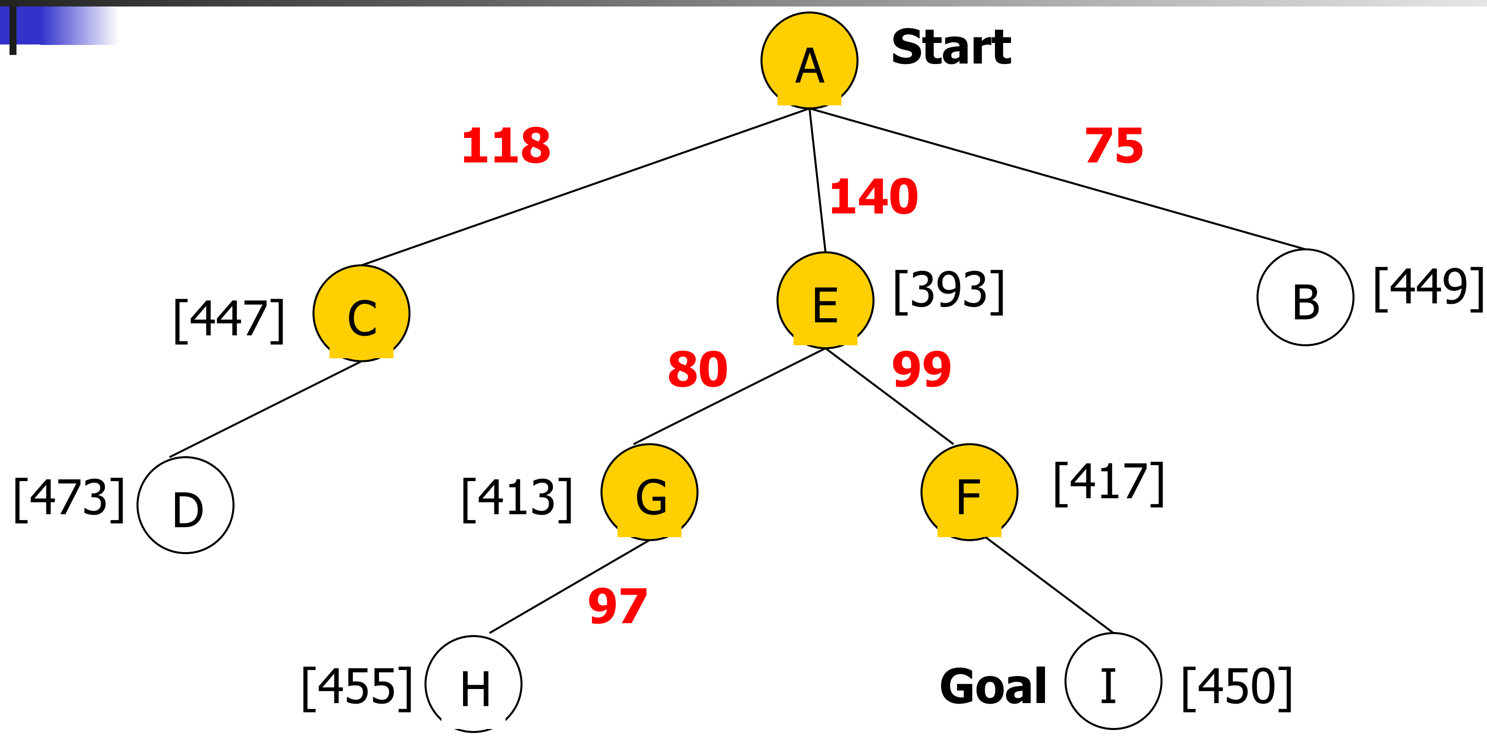
A* Search: Tree Search



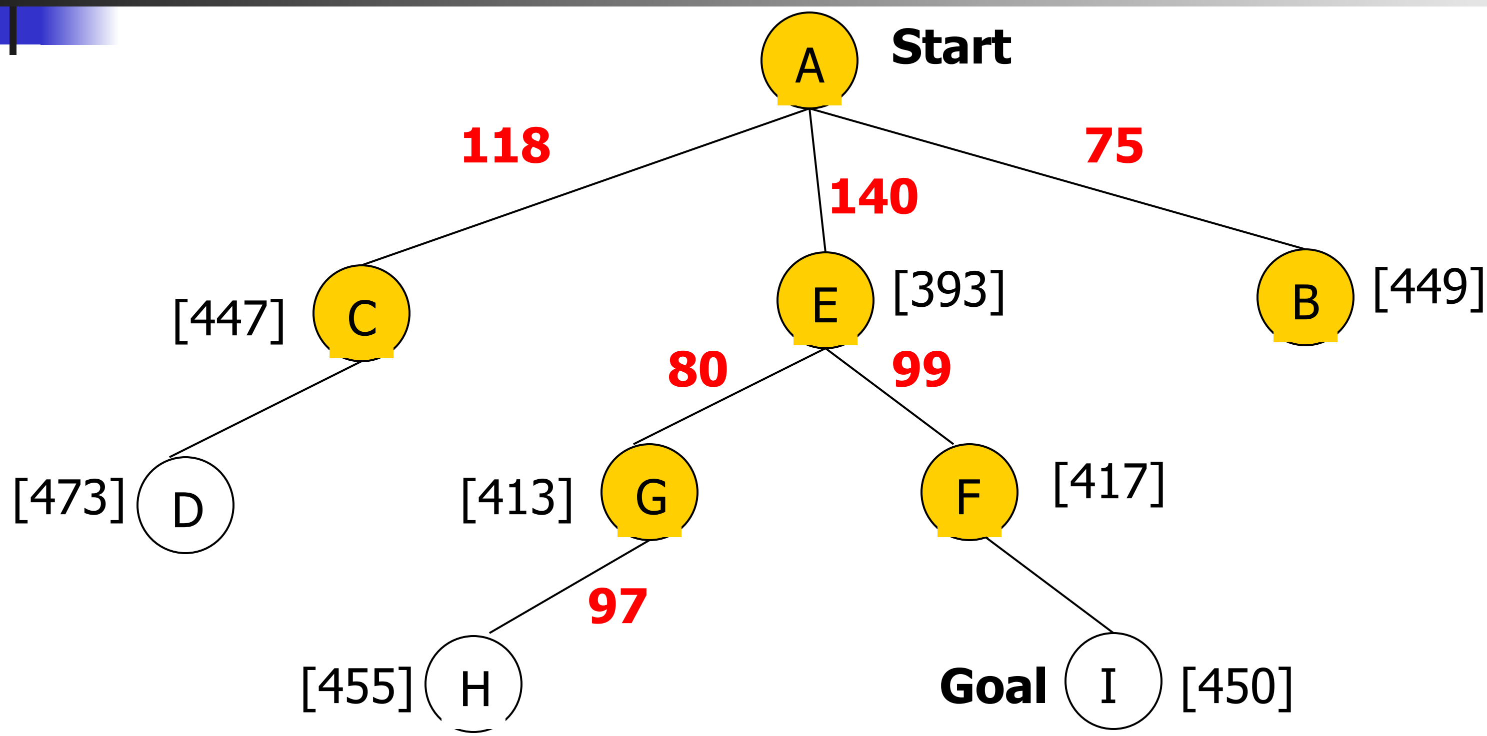
A* Search: Tree Search



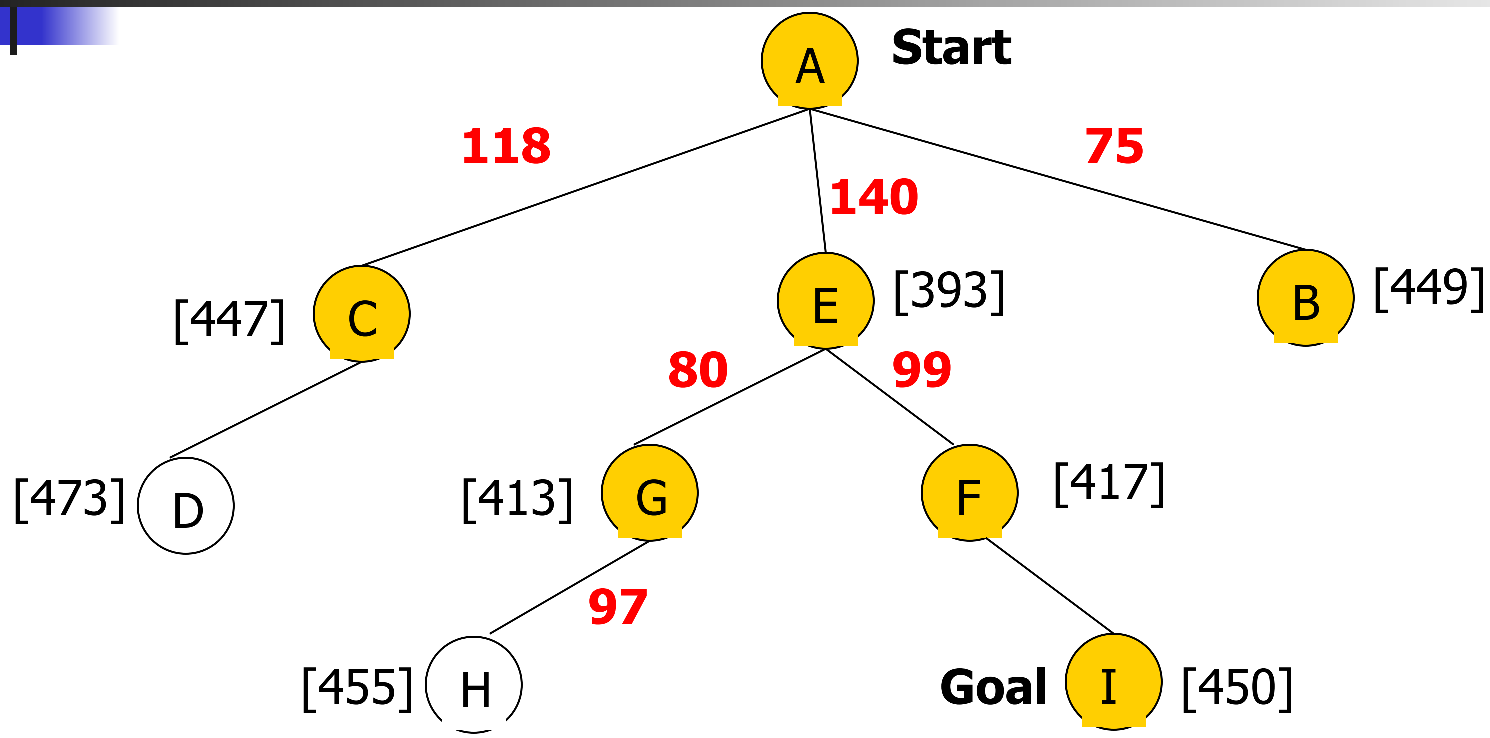
A* Search: Tree Search



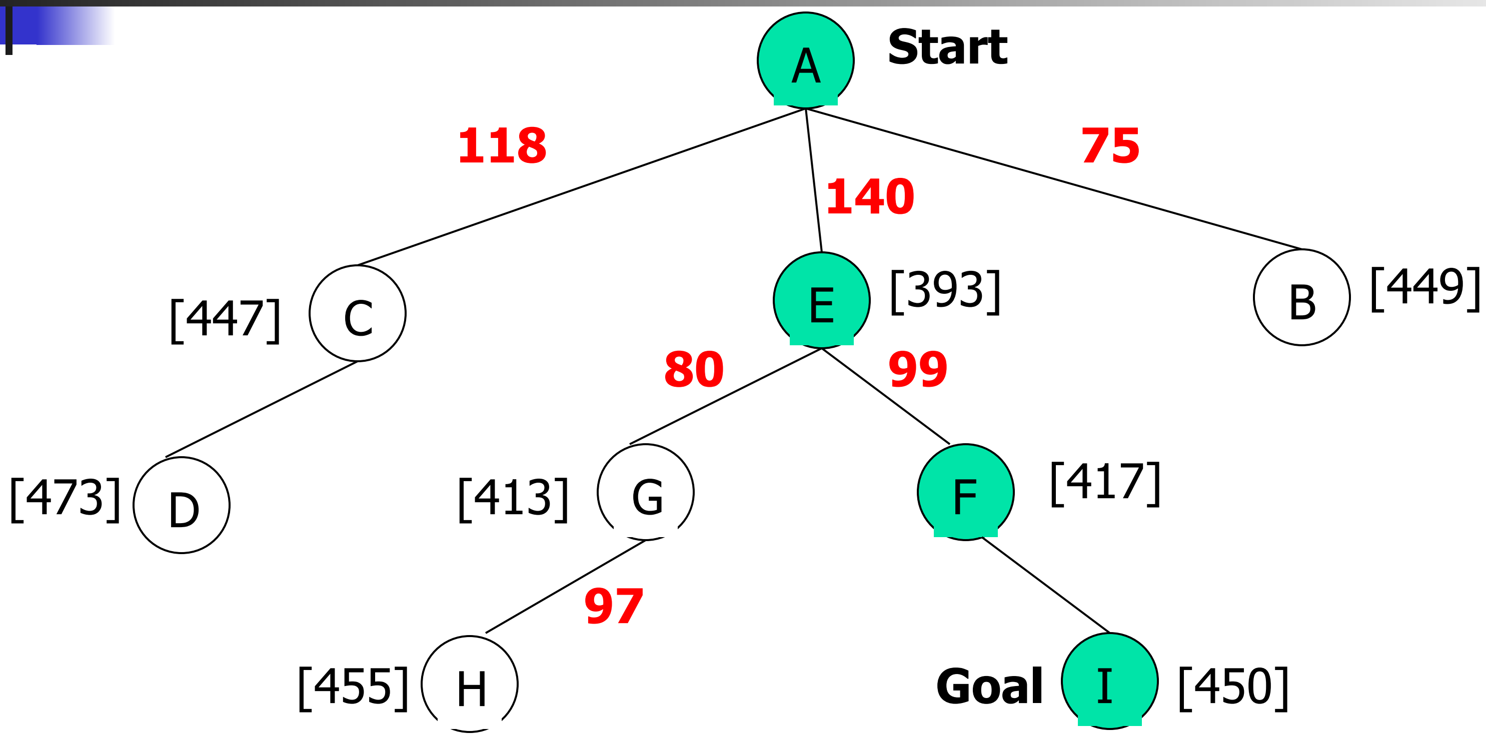
A* Search: Tree Search



A* Search: Tree Search



A* Search: Tree Search



Not optimal !!! (better path is A -> E -> G -> H -> I

A* Algorithm

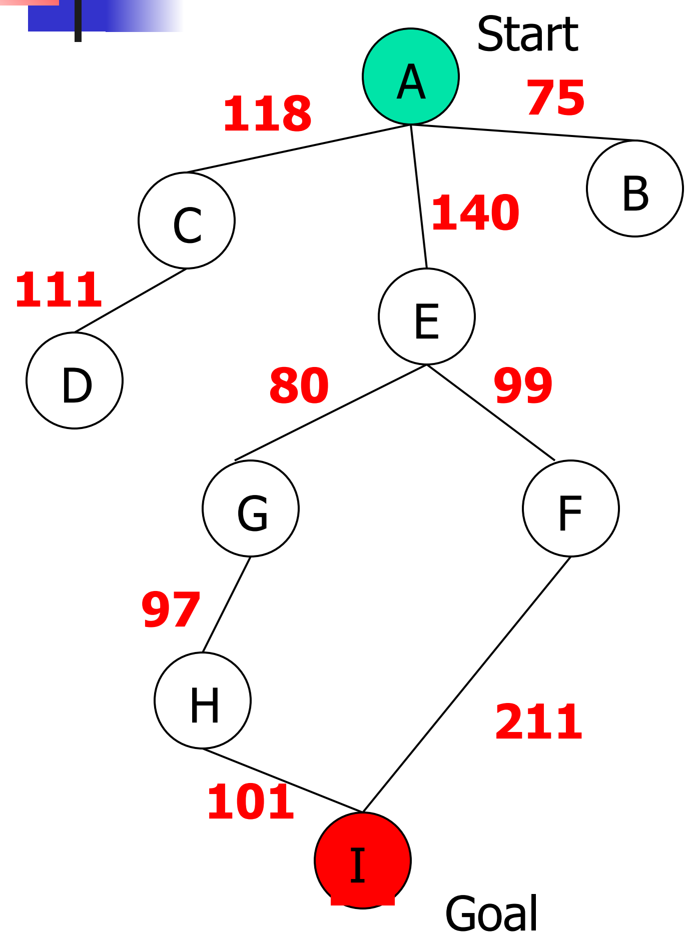
A* with systematic checking for repeated states …

A* Algorithm

1. Search queue Q is empty.
2. Place the start state s in Q with f value h(s).
3. If Q is empty, return failure.
4. Take node n from Q with lowest f value.
   (Keep Q sorted by f values and pick the first element).
5. If n is a goal node, stop and return solution.
6. Generate successors of node n.
7. For each successor n′ of n do:
   - a) Compute f(n′) = g(n) + cost(n,n′) + h(n′).
   - b) If n′ is new (never generated before), add n′ to Q.
   - c) If node n′ is already in Q with a higher f value, replace it with current f(n′) and place it in sorted order in Q.

   End for
8. Go back to step 3.

A* Search: Analysis



- A* is complete except if there is an infinity of nodes with f < f(G).
- A* is optimal if heuristic *h* is admissible.
- Time complexity depends on the quality of heuristic but is still exponential.
- For space complexity, A* keeps all nodes in memory. A* has worst case $O(b^d)$ space complexity, but an iterative deepening version is possible (IDA*).

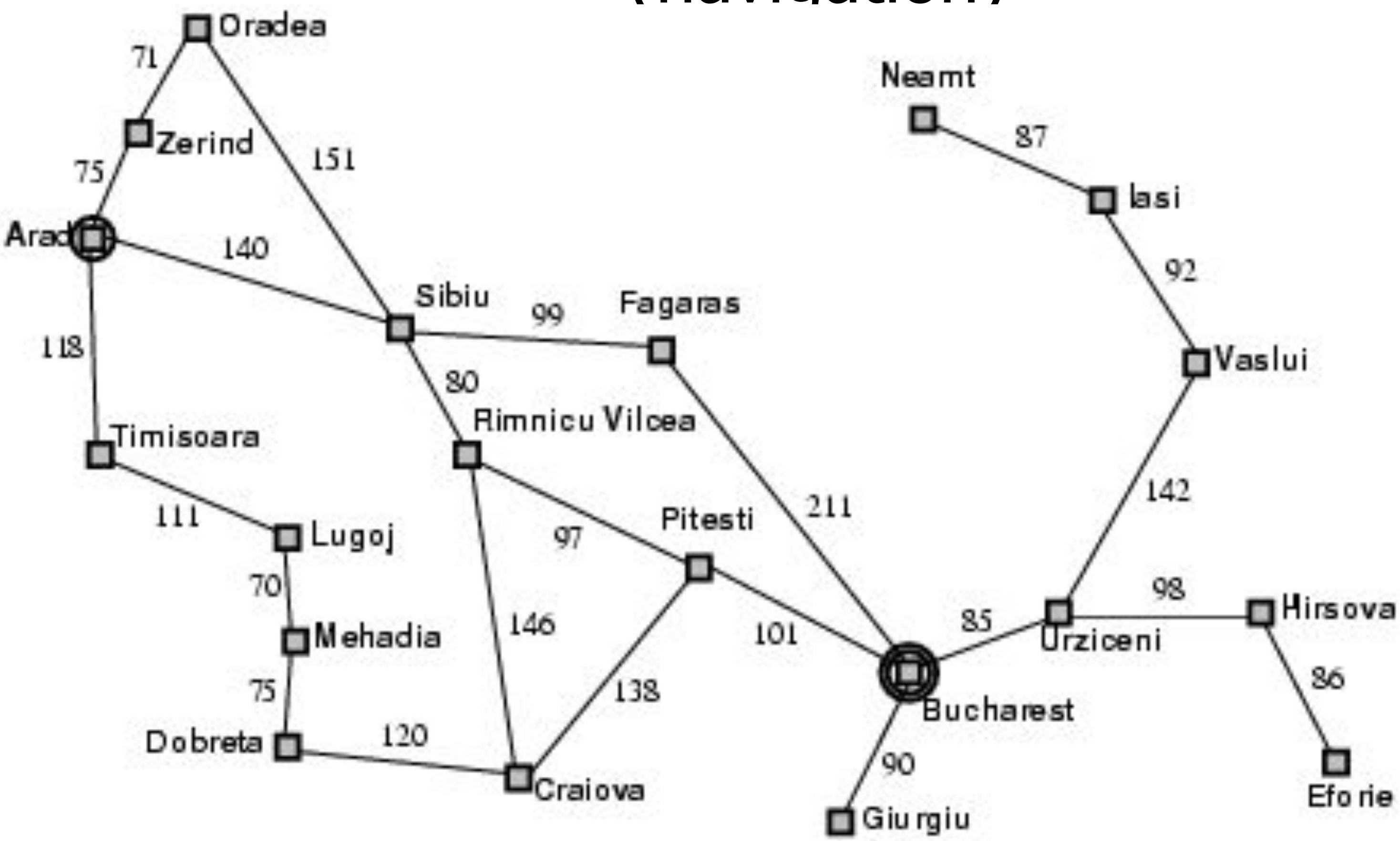
When to Use Search Techniques

- The search space is small, and
  - There are no other available techniques, or
  - It is not worth the effort to develop a more efficient technique

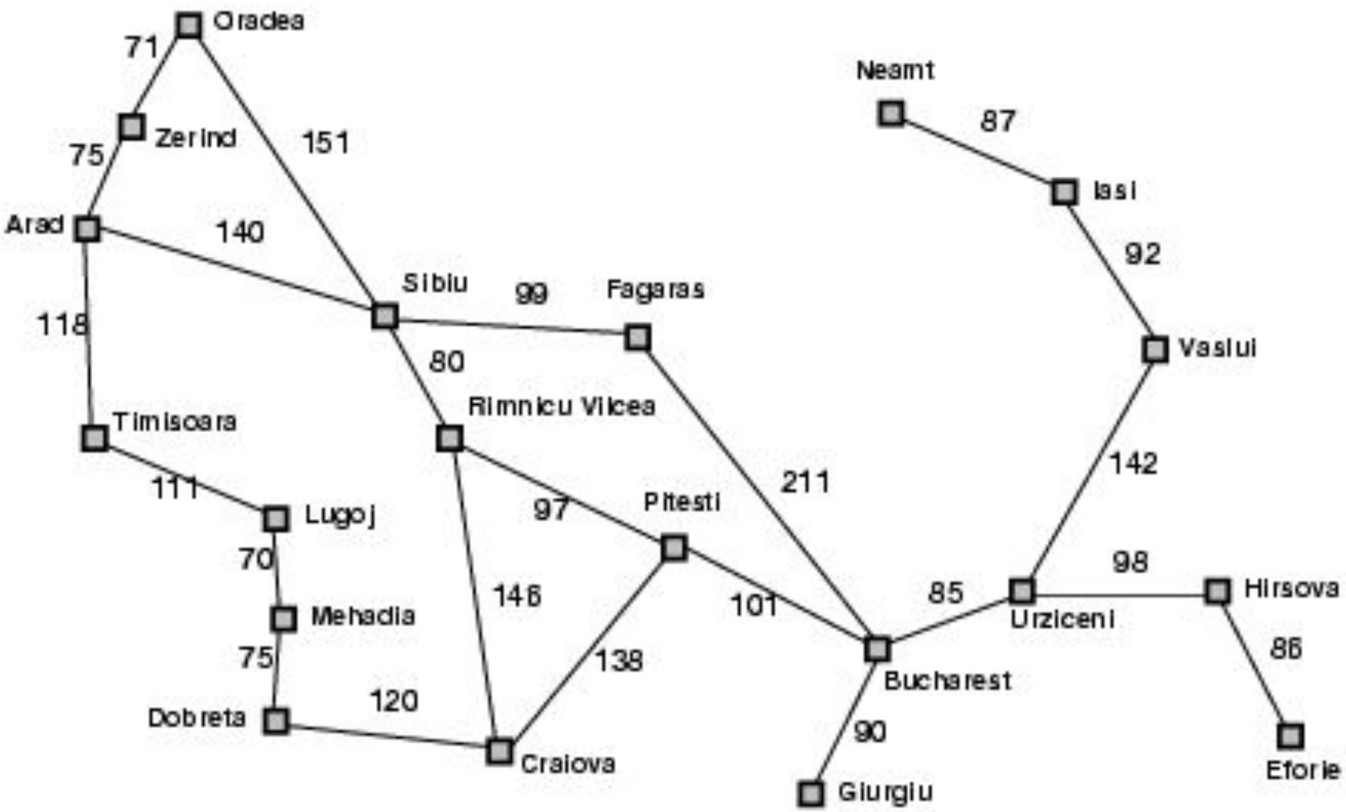
- The search space is large, and
  - There is no other available techniques, and
  - There exist “good” heuristics

Popular AI Search Problems

Classic AI search problems, Map searching (navigation)



Romania with step costs in km

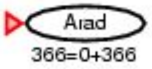


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

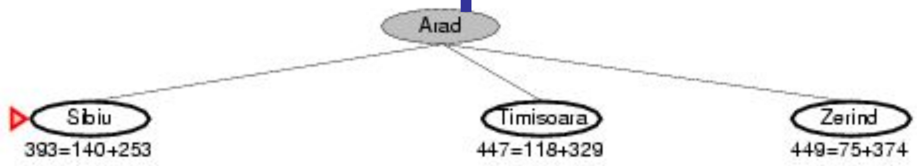
A^* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost so far to reach n
  - $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal

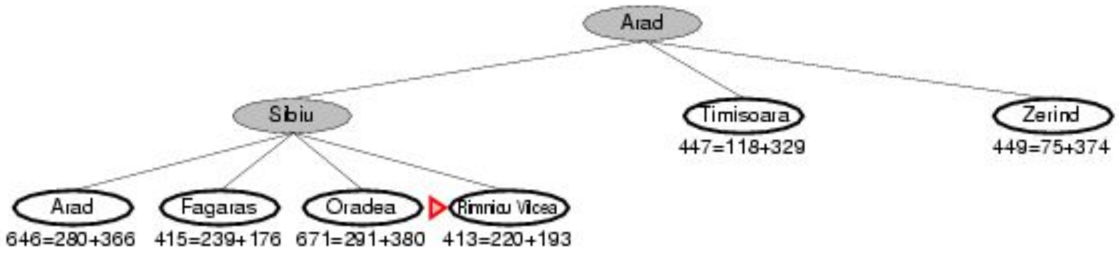
A* search example



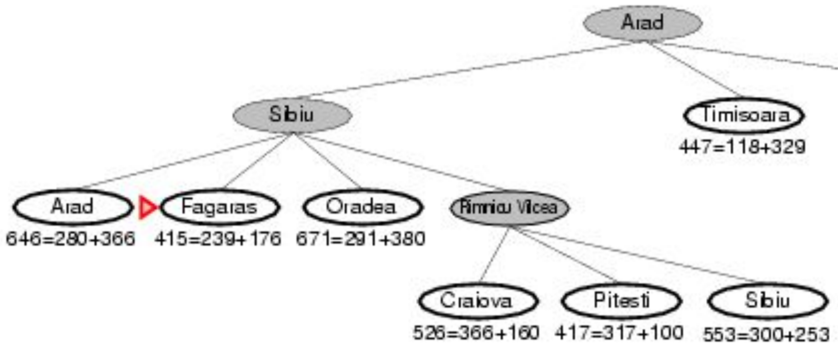
A^* search example



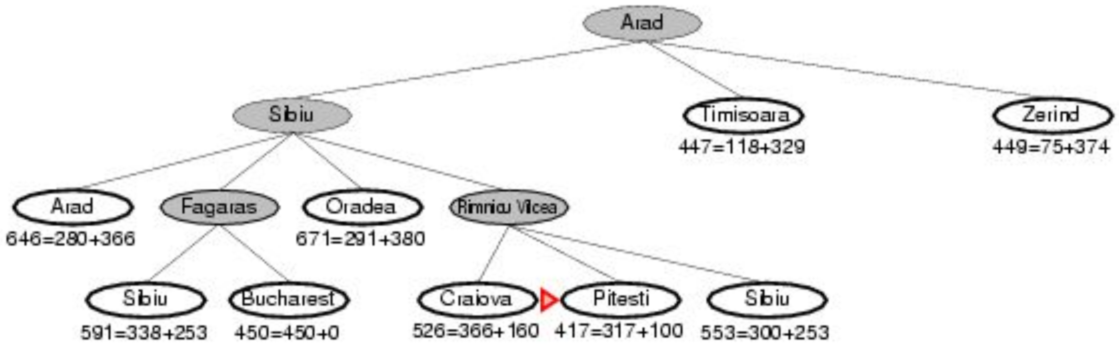
A* search example



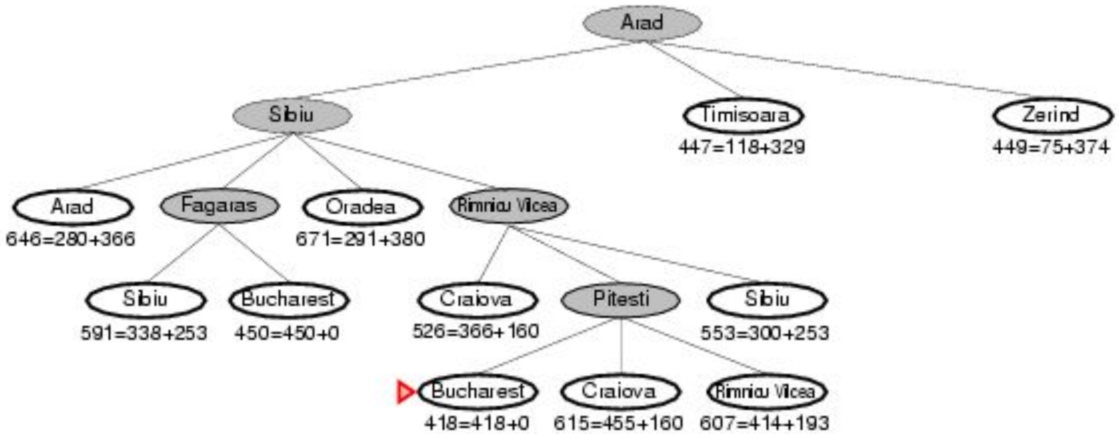
A* search example



A* search example

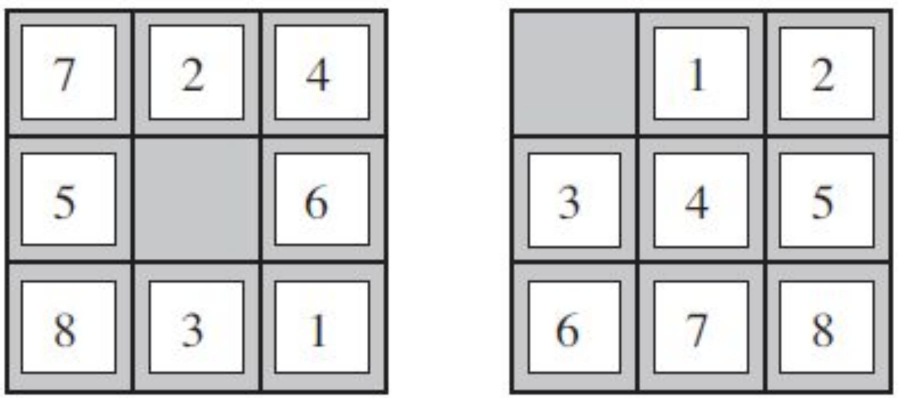


A* search example



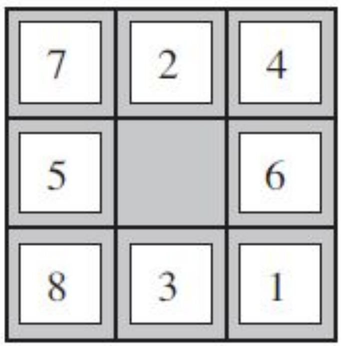
But what is a **good** heuristic?

8-puzzle problem

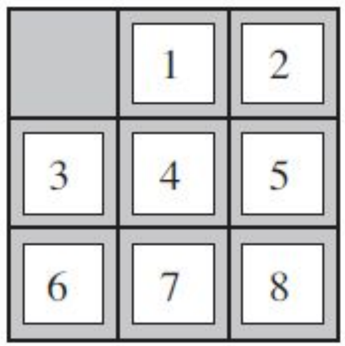


Start State　　Goal State

Heuristic 1: Misplaced Tiles



Start State

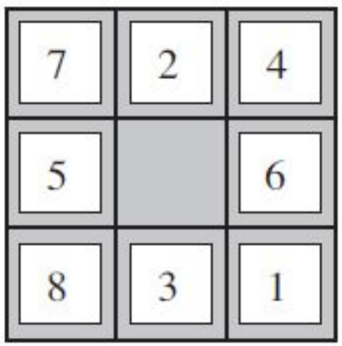


Goal State

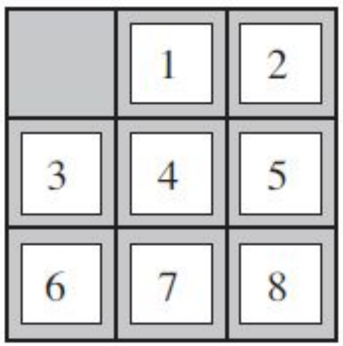
h(n) = number of tiles that are out of position in state/node n

- For the start state h(n) = 8 as all tiles are out of position in comparison to the goal
- Following a best first search procedure, expanding the start state, we will get 4 newer states.
- Calculate value of h(n) for each state and expand the state with the least value of h(n) i.e. the best state
- Continue same process until we reach goal state

Heuristic 2: Manhattan Distance



Start State

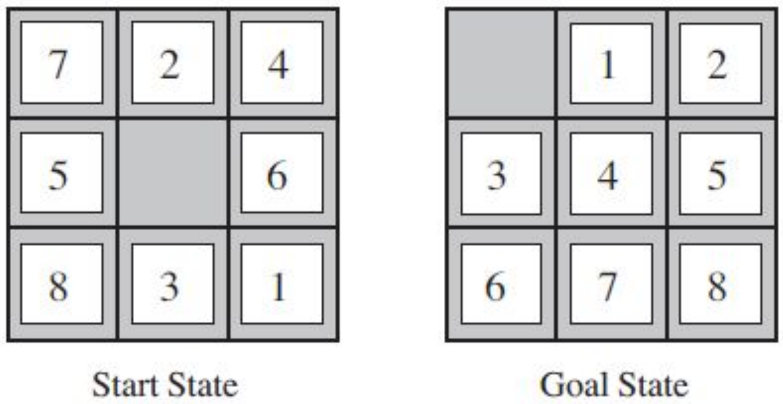


Goal State

h(n) = sum of the distances of the tiles from their current position to the goal positions for a state n

- For the start state h(n) = 3 + 1 + 2 + 2+ 2 + 3 + 3 + 2 = 18 [Remember that tiles can only move in right angles i.e. either vertically or horizontally]
- Following a best first search procedure, expanding the start state, we will get 4 newer states.
- Calculate value of h(n) for each state and expand the state with the least value of h(n) i.e. the best state
- Continue same process until we reach goal state

Heuristic Dominance



Total Misplaced tiles, h1 = 8
Total Manhattan distance, h2 = 18

- Now if both h1(n) and h2(n) are admissible and h2(n) ≥ h1(n) for all nodes n, then we say h2(n) dominates h1(n).
- h2(n) will be better for search and will search less nodes than h1(n)
- If there are several admissible heuristic, the one with highest value should be chosen
- The estimated cost, h (n) should be made as large as possible without exceeding the actual cost, h*(n)

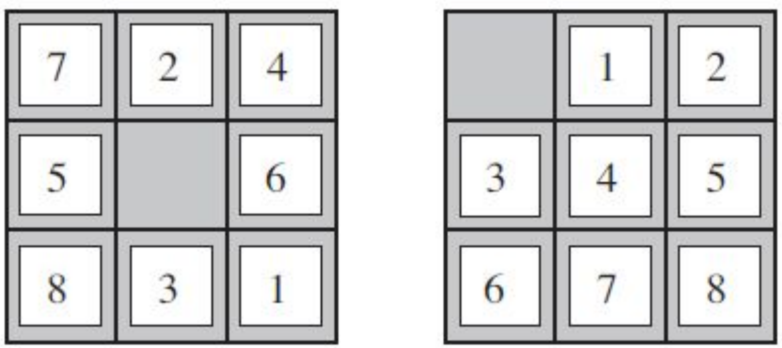
How to choose an **admissible** heuristic?

Generate an Admissible Heuristic by **Constraint Relaxation**

- Identify the preconditions or constraints
- Create relaxed problems by dropping the preconditions or constraints
- Solve the relaxed problems without searching
- An optimal solution of a relaxed problem will be an admissible heuristic for the original problem.

Let us understand this with the help of the 8 puzzle problem

Generate an Admissible Heuristic by **Constraint Relaxation**



Start State Goal State

Let us Keep precondition/ constraint (I), (II) and drop (III) :

- A newer relaxed version of the problem is thus created
- A tile will be placed initially in a specific cell and any tile can move either vertically or horizontally **without taking into account whether destination cell or any cell in between is empty**
- An optimum solution can be found by calculating the shortest path between the initial position and the goal positions for each tile
- This solution is exactly the same as Manhattan Distance heuristic.

Properties of A*

- Complete? Yes (unless there are infinitely many nodes with $f \leq f(G)$)
- Time? Depends on the quality of heuristic but still exponential.
- Space? Keeps all nodes in memory. A* has worst case $O(b^d)$ space complexity
- Optimal? Yes