

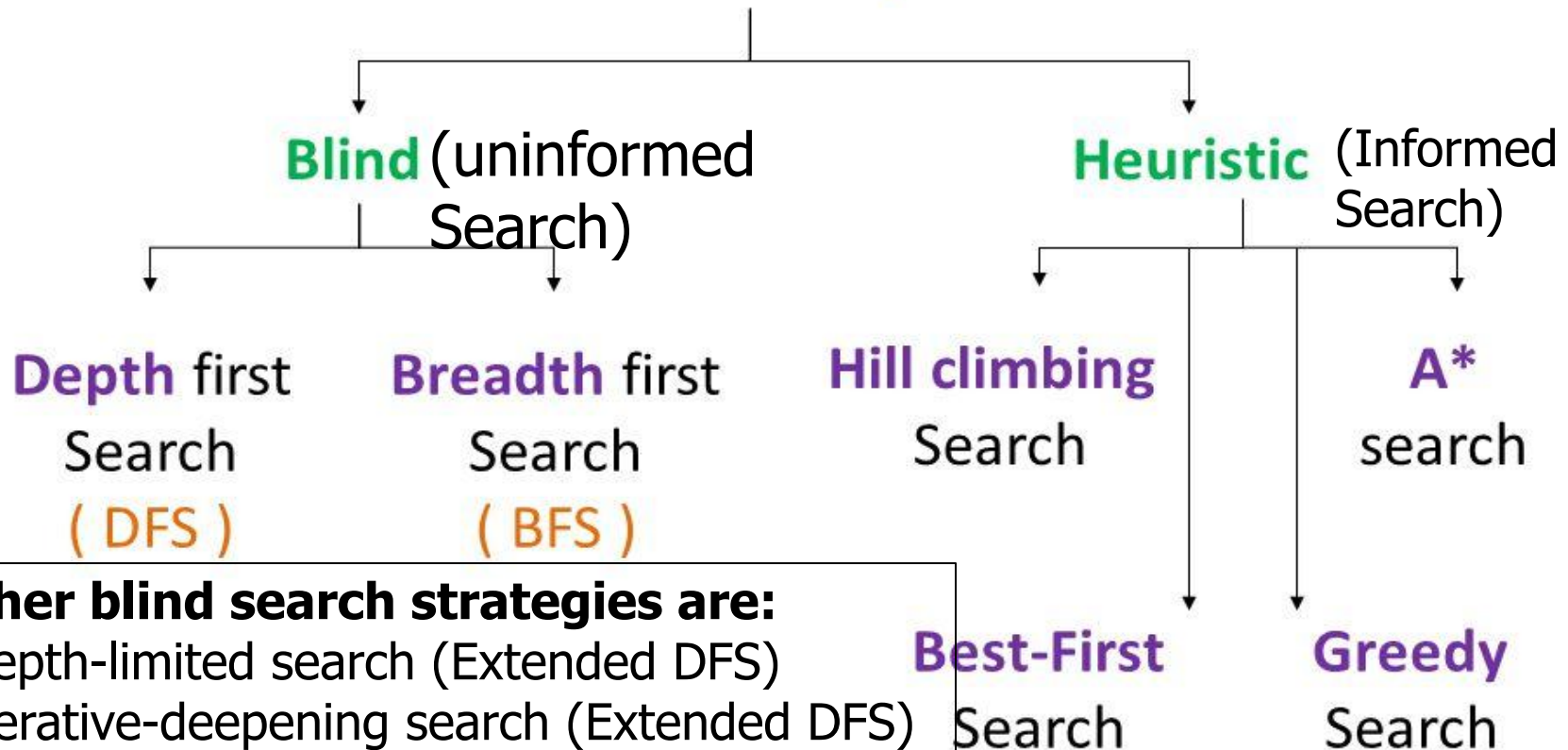


Artificial Intelligence

Informed Search

SEARCH TECHNIQUES

Search techniques



Other blind search strategies are:

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



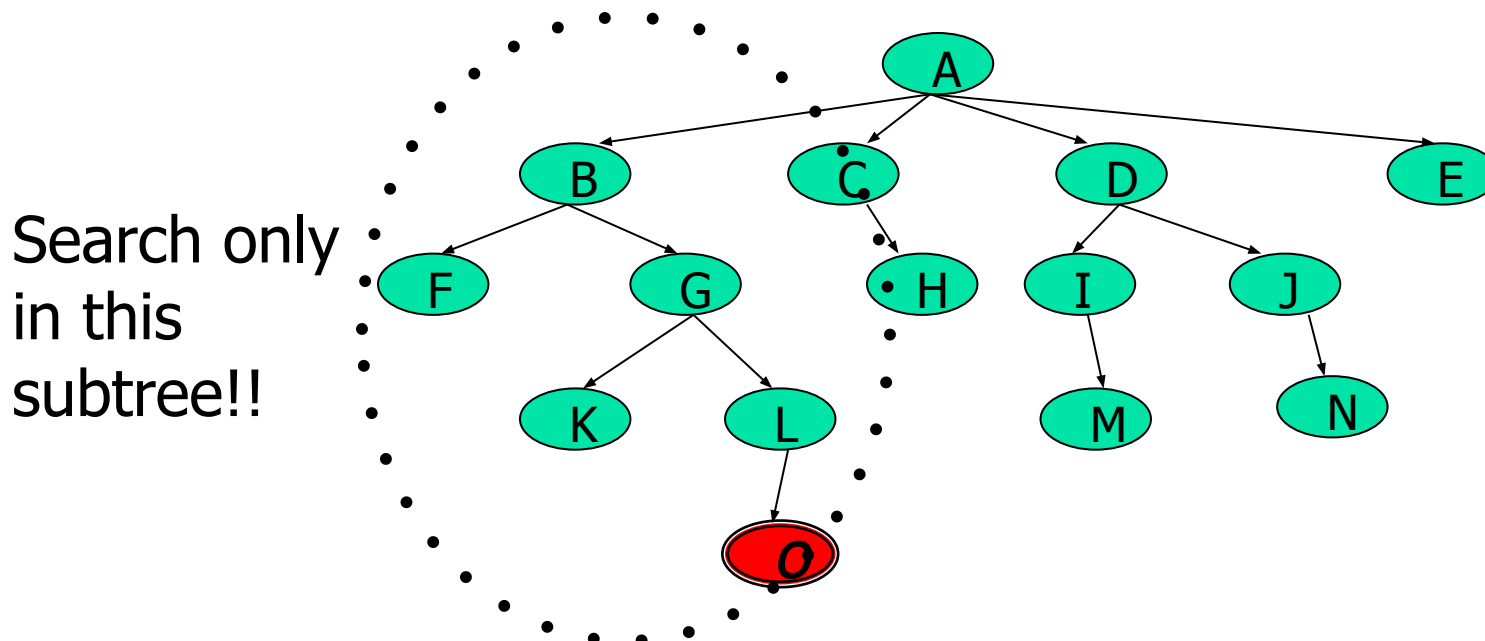
Uninformed Vs Informed Search

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.



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

- $1 \leq b^* \ll b$

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.

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

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 efficient 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*)

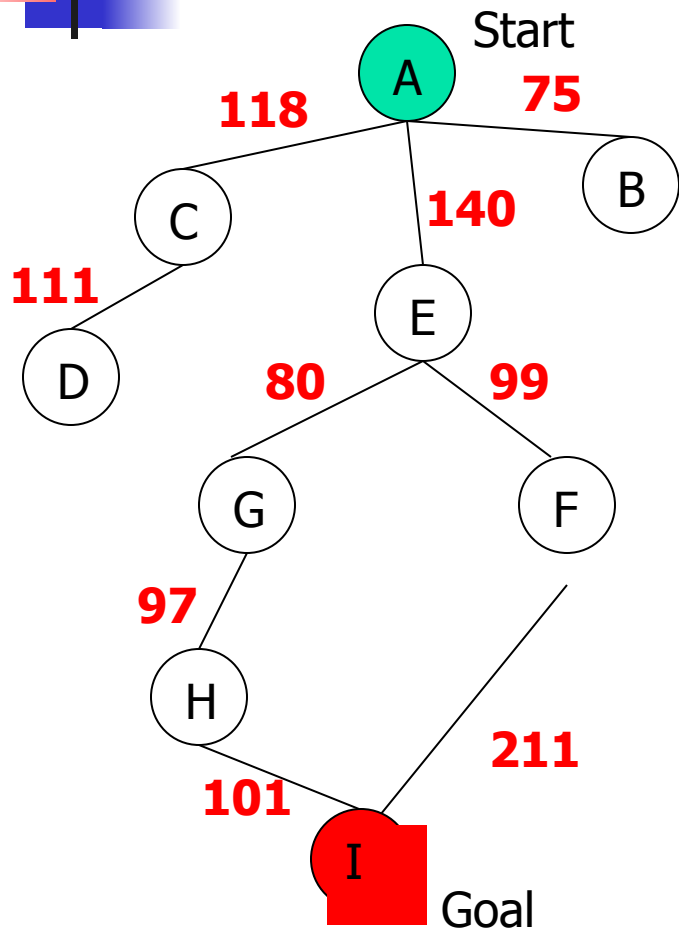


Informed Search Strategies

Greedy Search

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

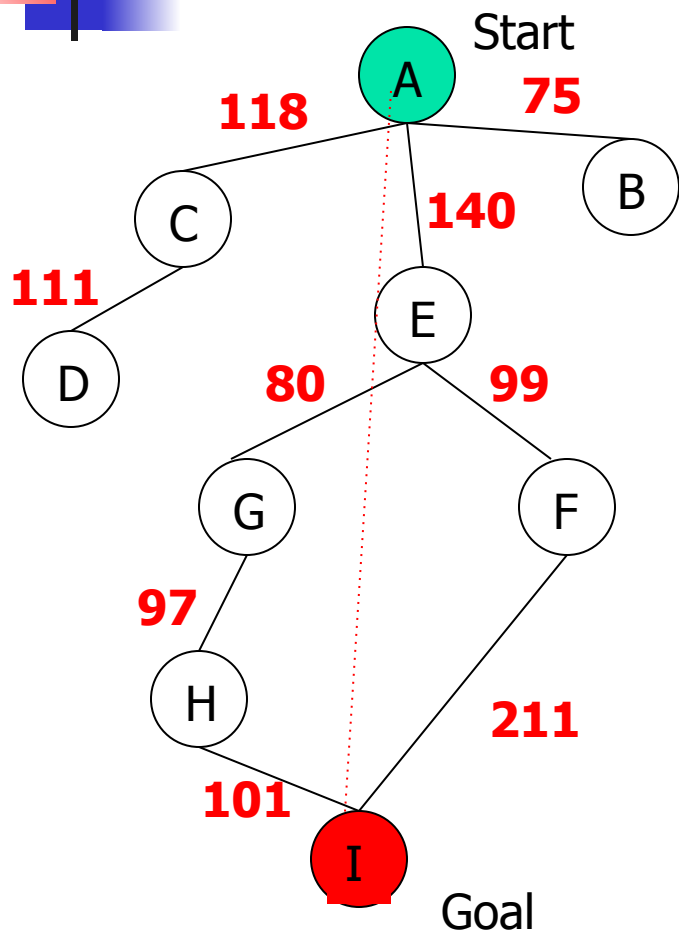
Greedy Search



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

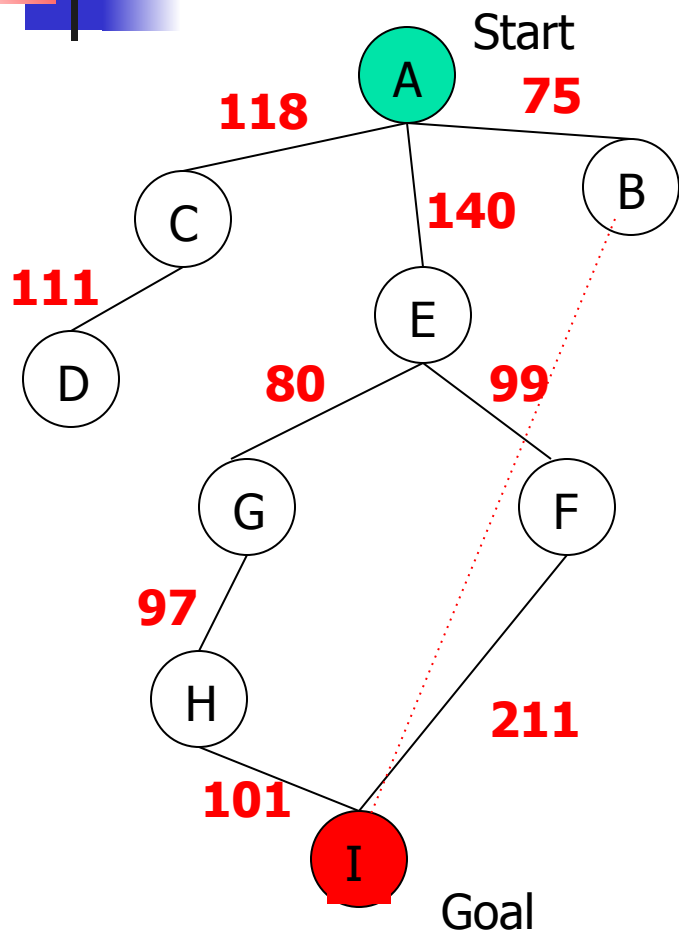
Greedy Search



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

$f(n) = h(n) = \text{straight-line distance heuristic}$

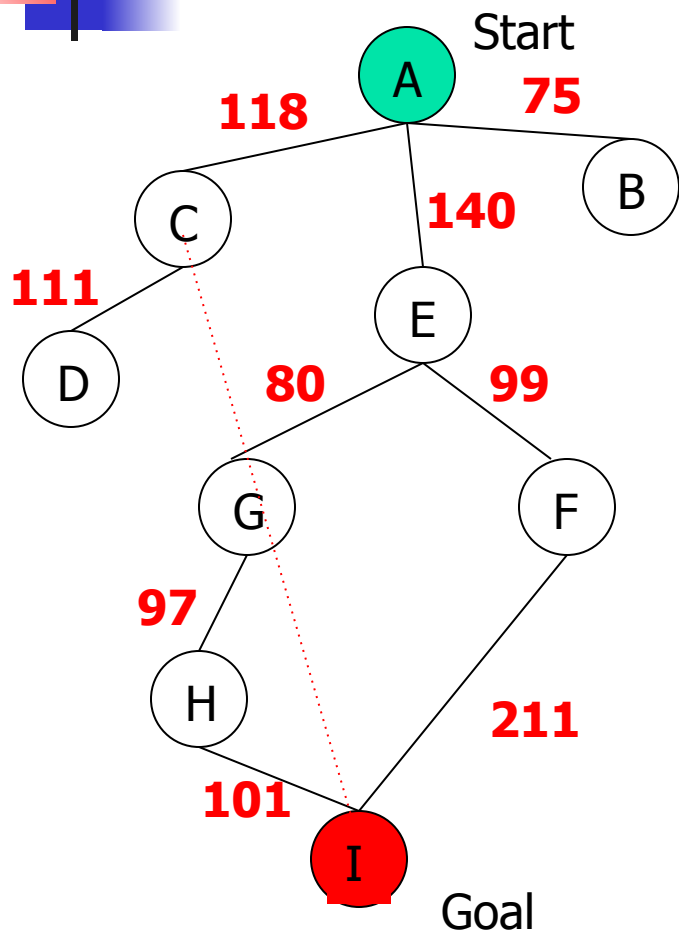
Greedy Search



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

$f(n) = h(n) = \text{straight-line distance heuristic}$

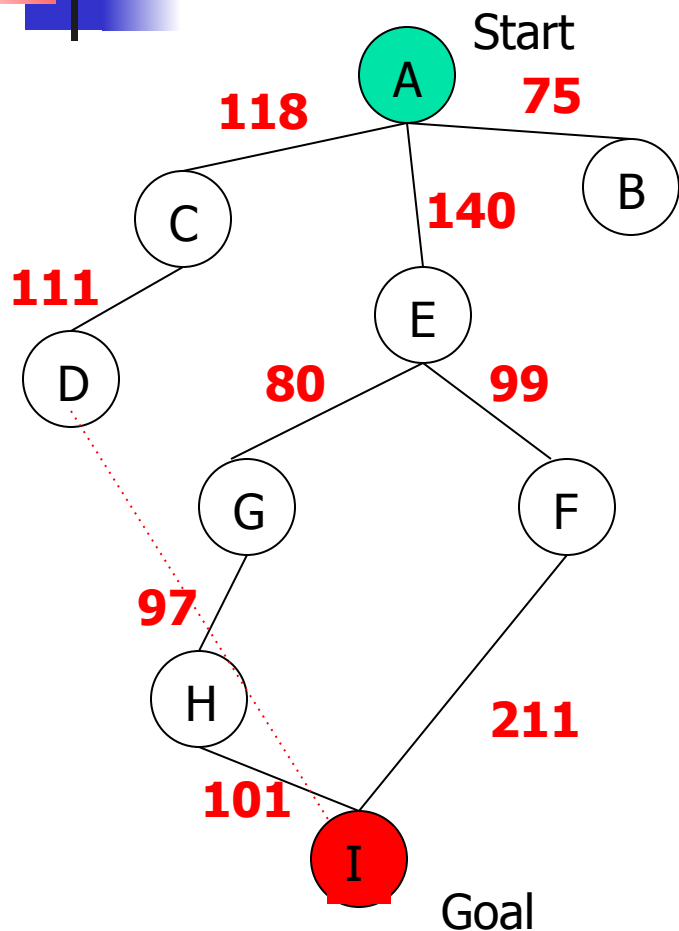
Greedy Search



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

$f(n) = h(n) = \text{straight-line distance heuristic}$

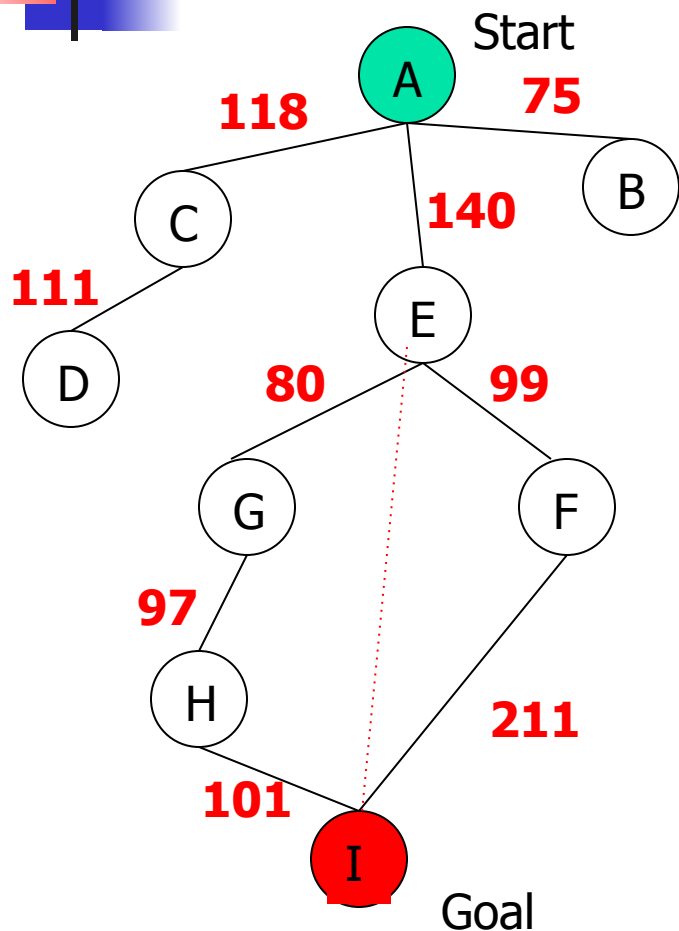
Greedy Search



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

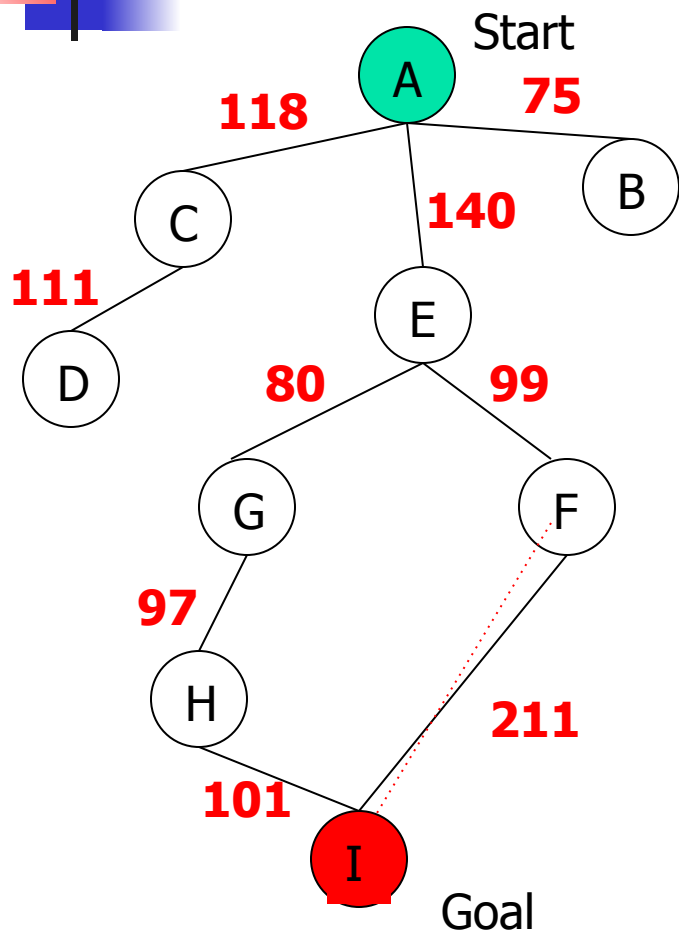
Greedy Search



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

$f(n) = h(n) = \text{straight-line distance heuristic}$

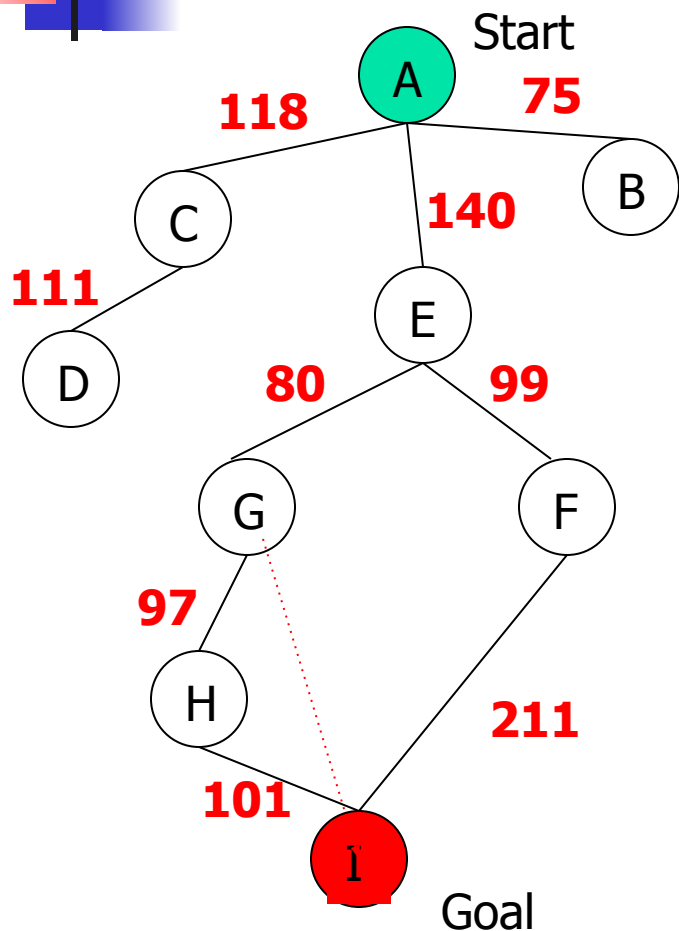
Greedy Search



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

$f(n) = h(n) = \text{straight-line distance heuristic}$

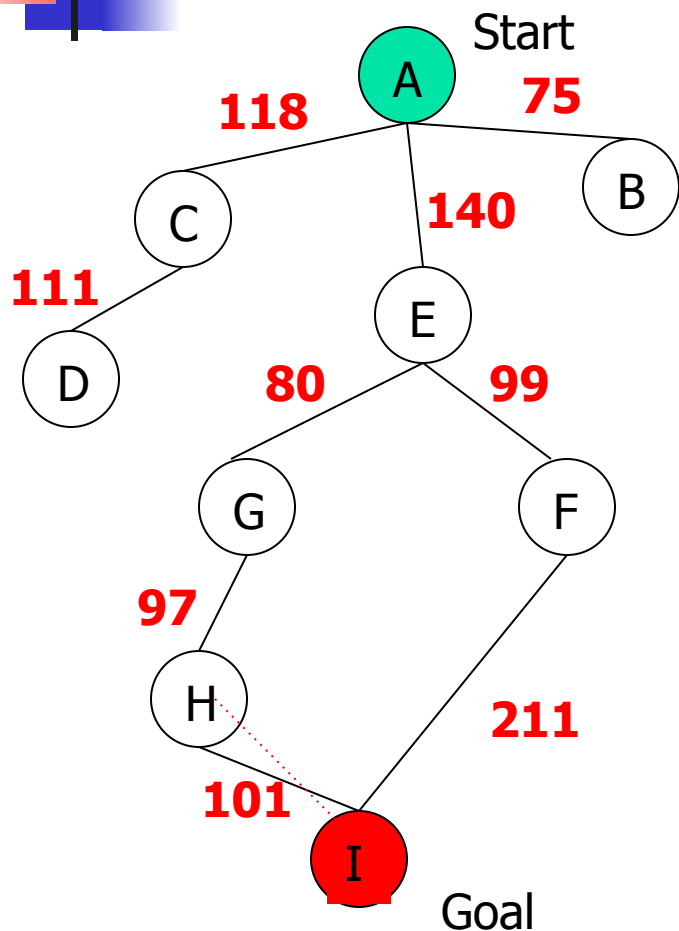
Greedy Search



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

$f(n) = h(n) = \text{straight-line distance heuristic}$

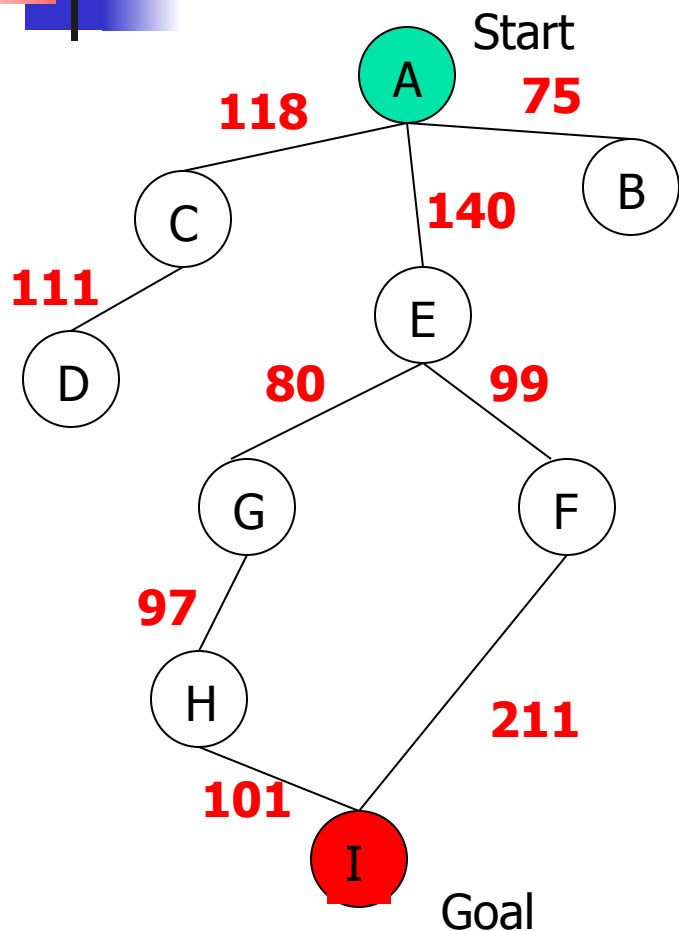
Greedy Search



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

$f(n) = h(n) = \text{straight-line distance heuristic}$

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



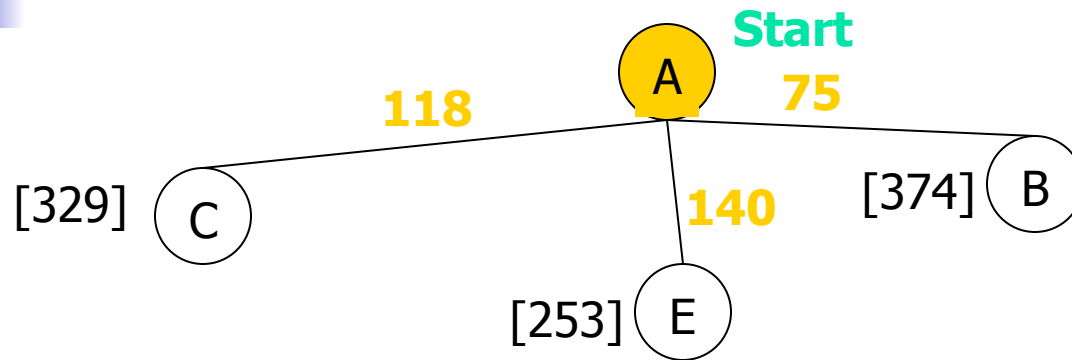
Greedy Search: Tree Search

A

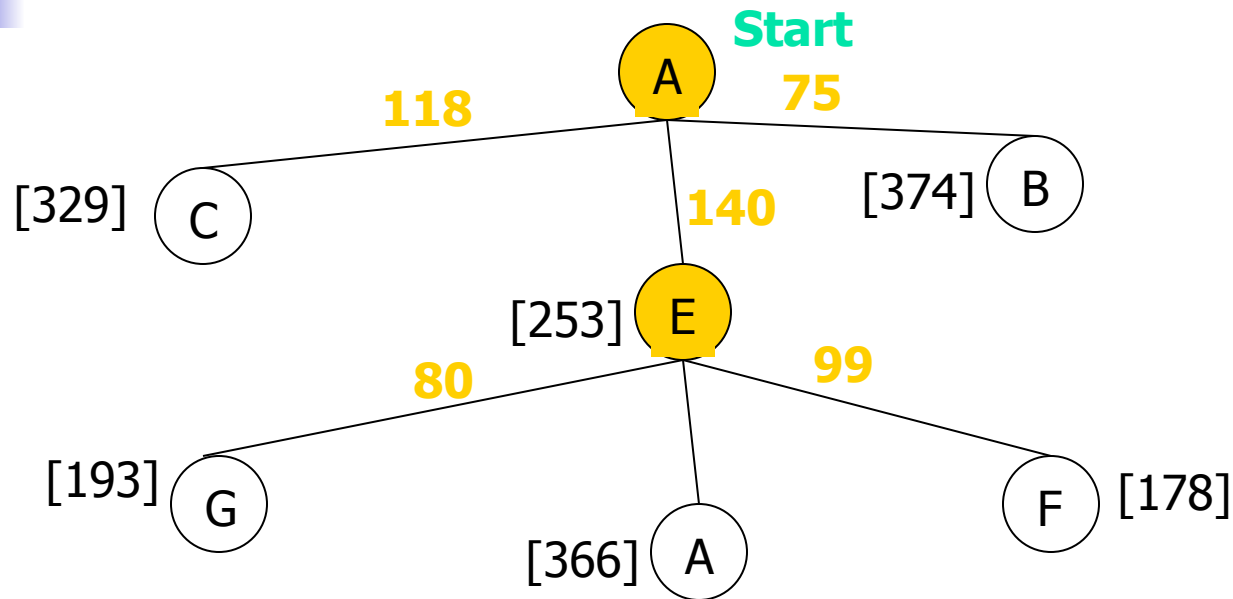
Start



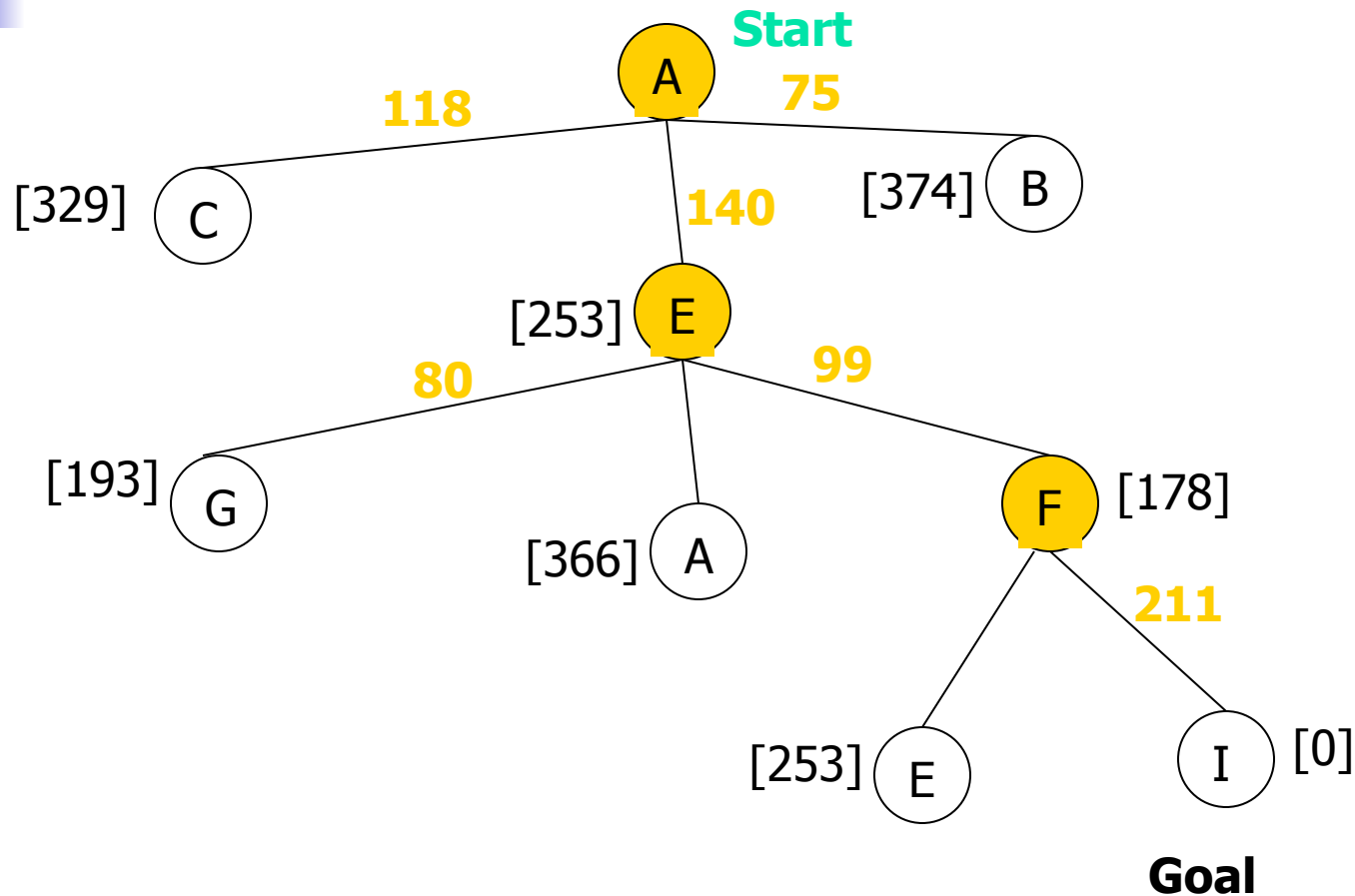
Greedy Search: Tree Search



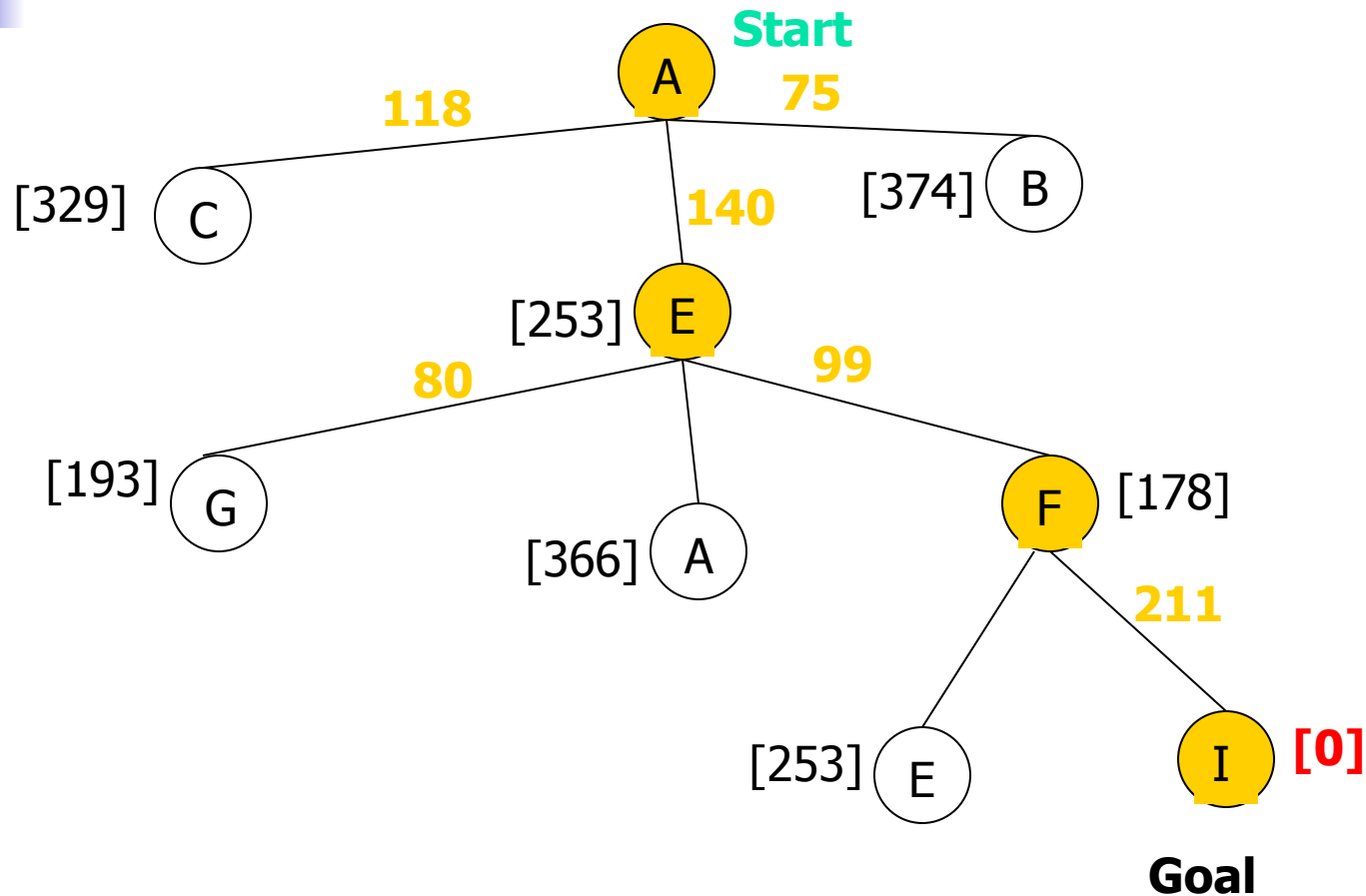
Greedy Search: Tree Search



Greedy Search: Tree Search



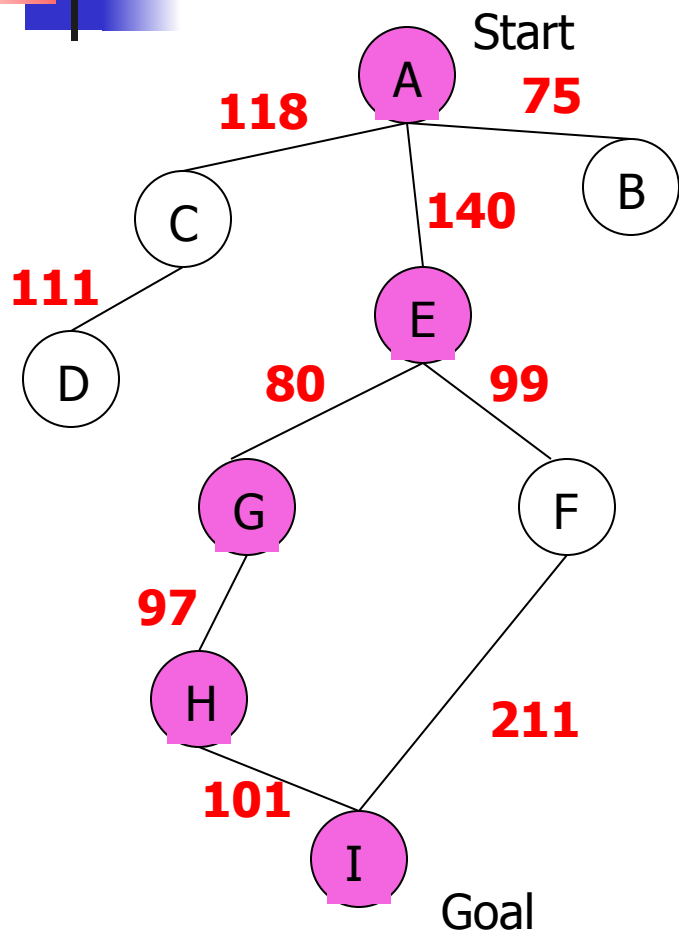
Greedy Search: Tree Search



Path cost(A-E-F-I) = 253 + 178 + 0 = **431**

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

Greedy Search: Optimal ?

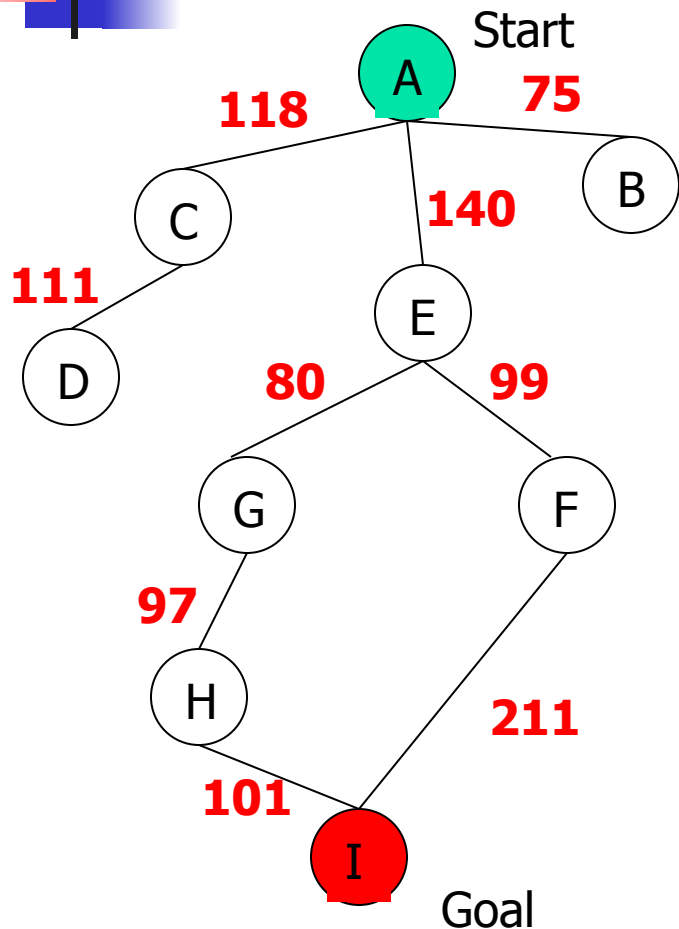


State	Heuristic: $h(n)$
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

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

$\text{dist}(A-E-G-H-I) = 140 + 80 + 97 + 101 = 418$

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

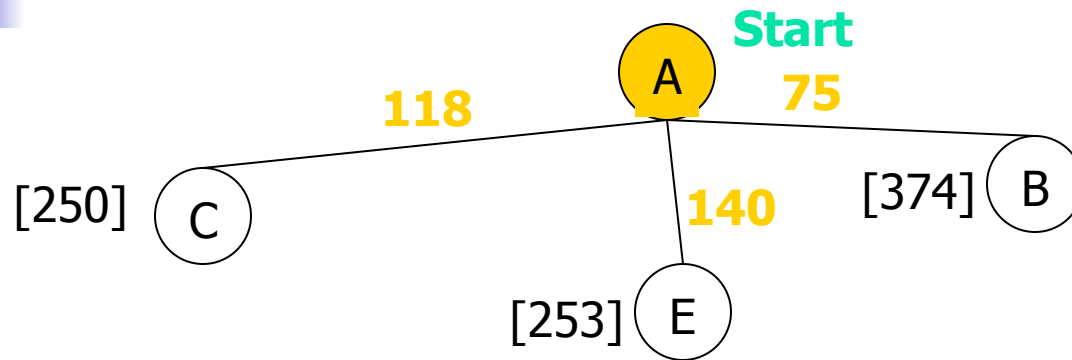


Greedy Search: Tree Search

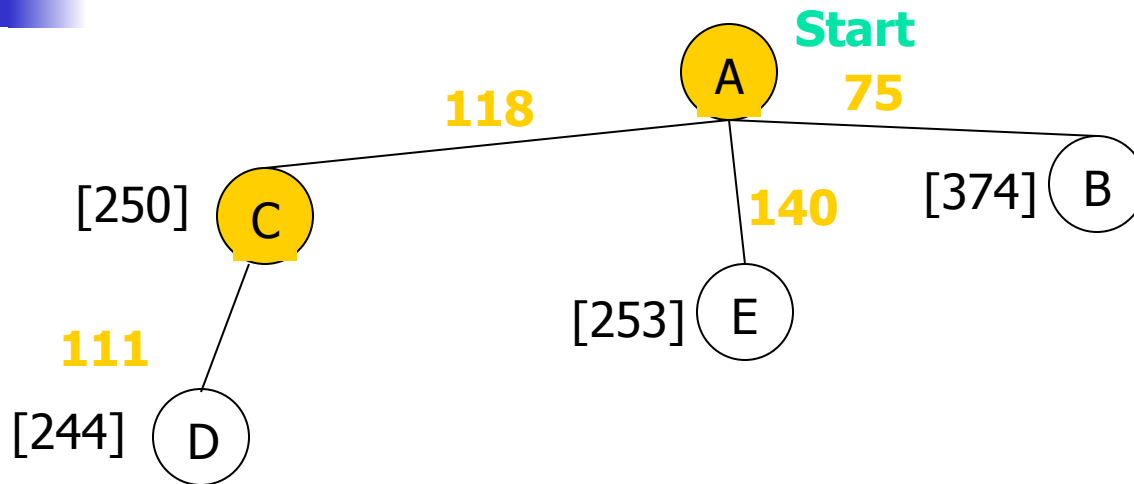
A

Start

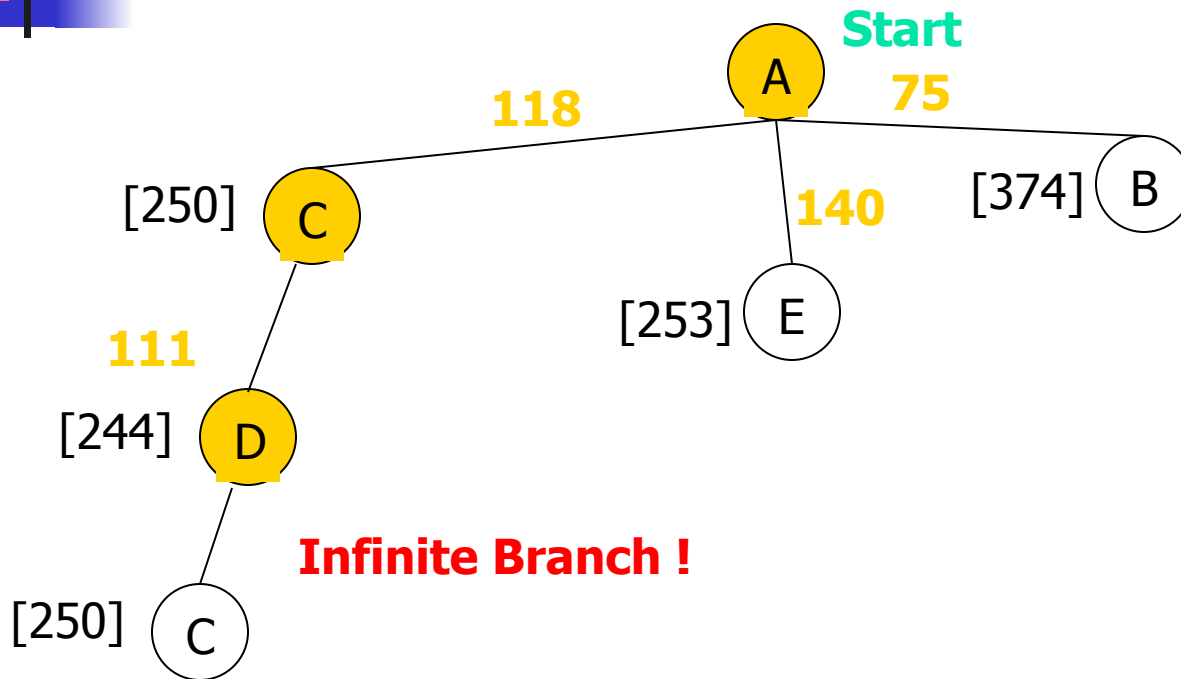
Greedy Search: Tree Search



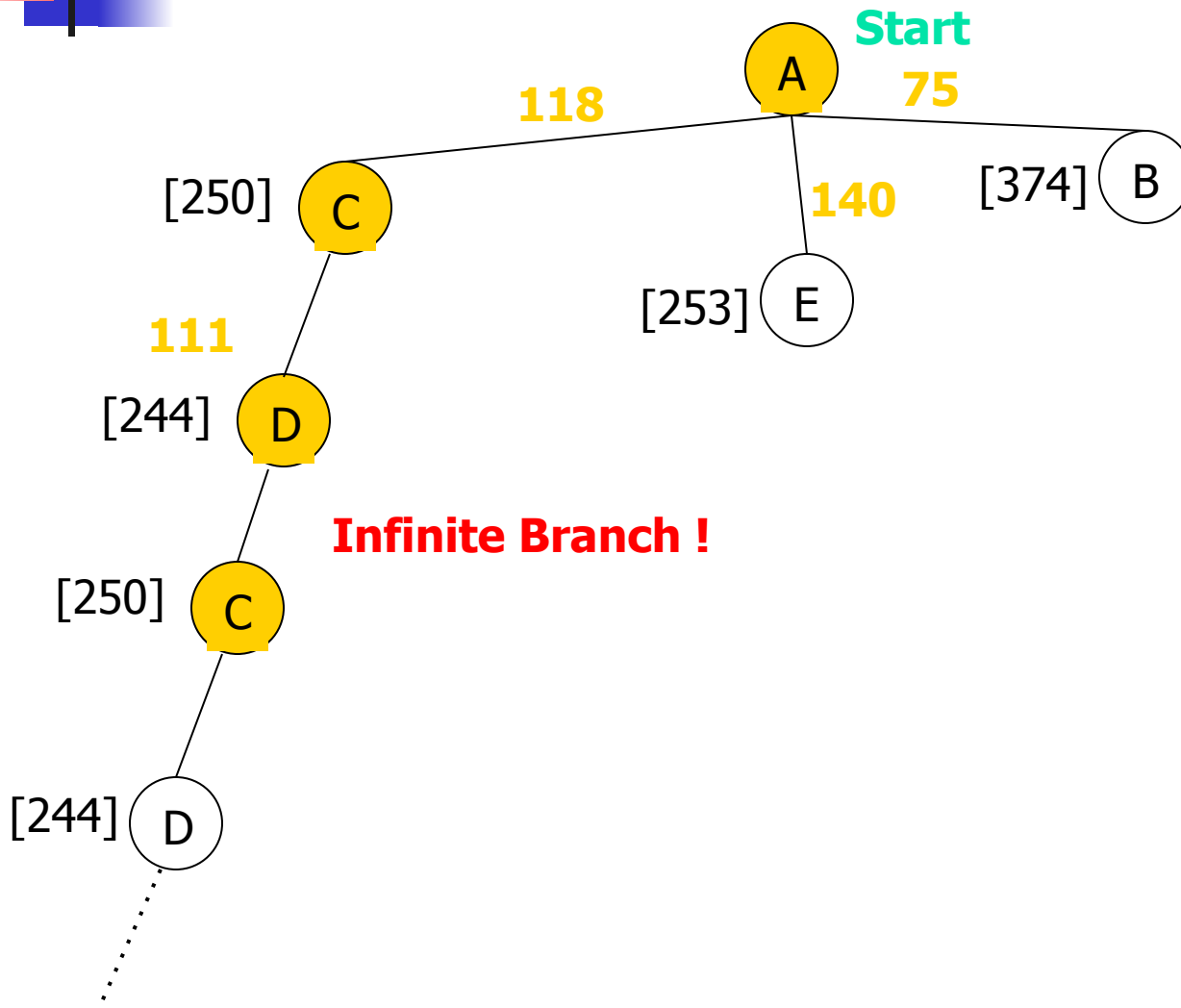
Greedy Search: Tree Search



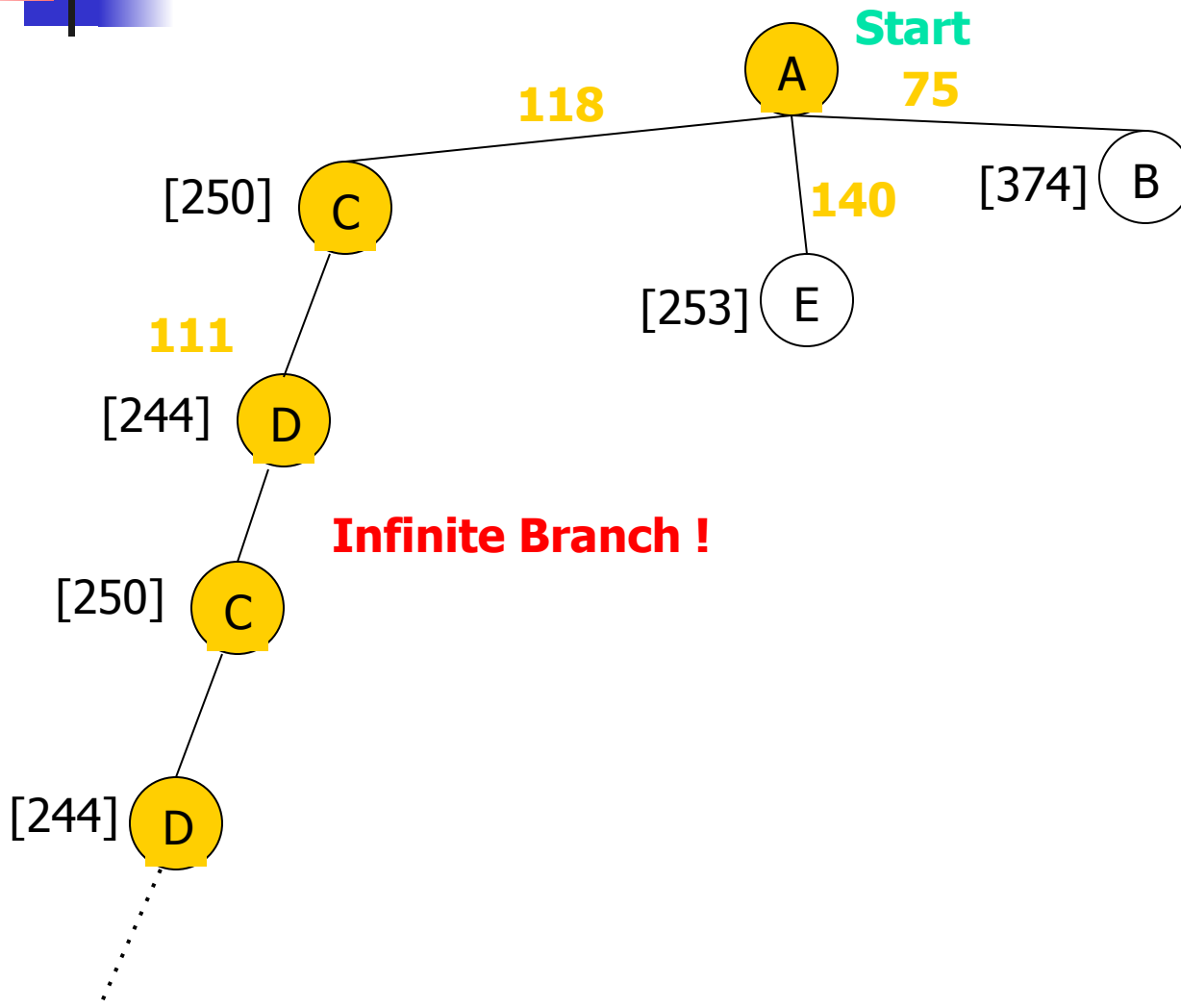
Greedy Search: Tree Search



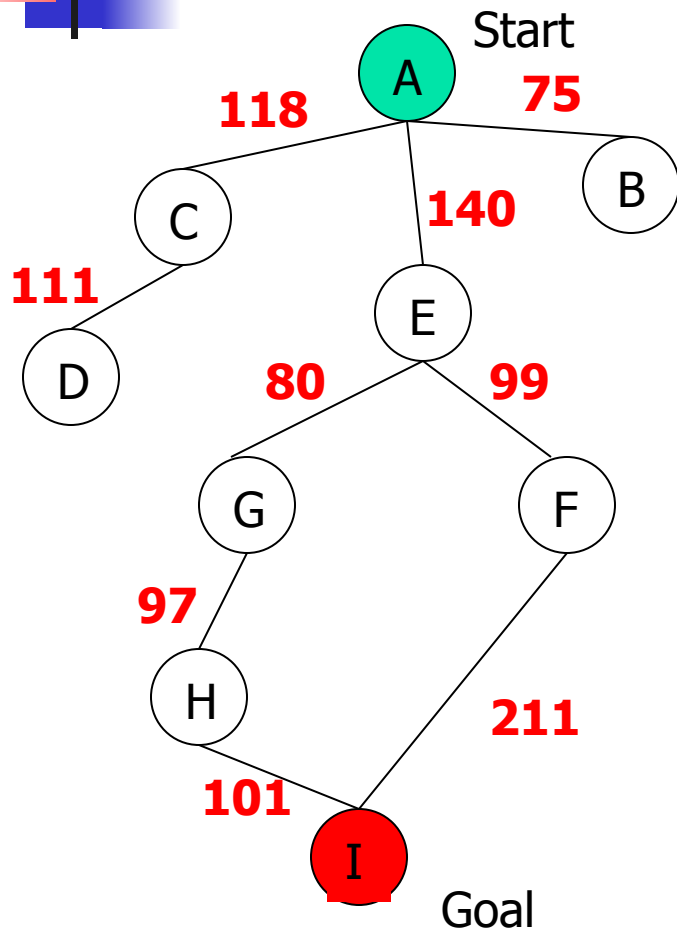
Greedy Search: Tree Search



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)

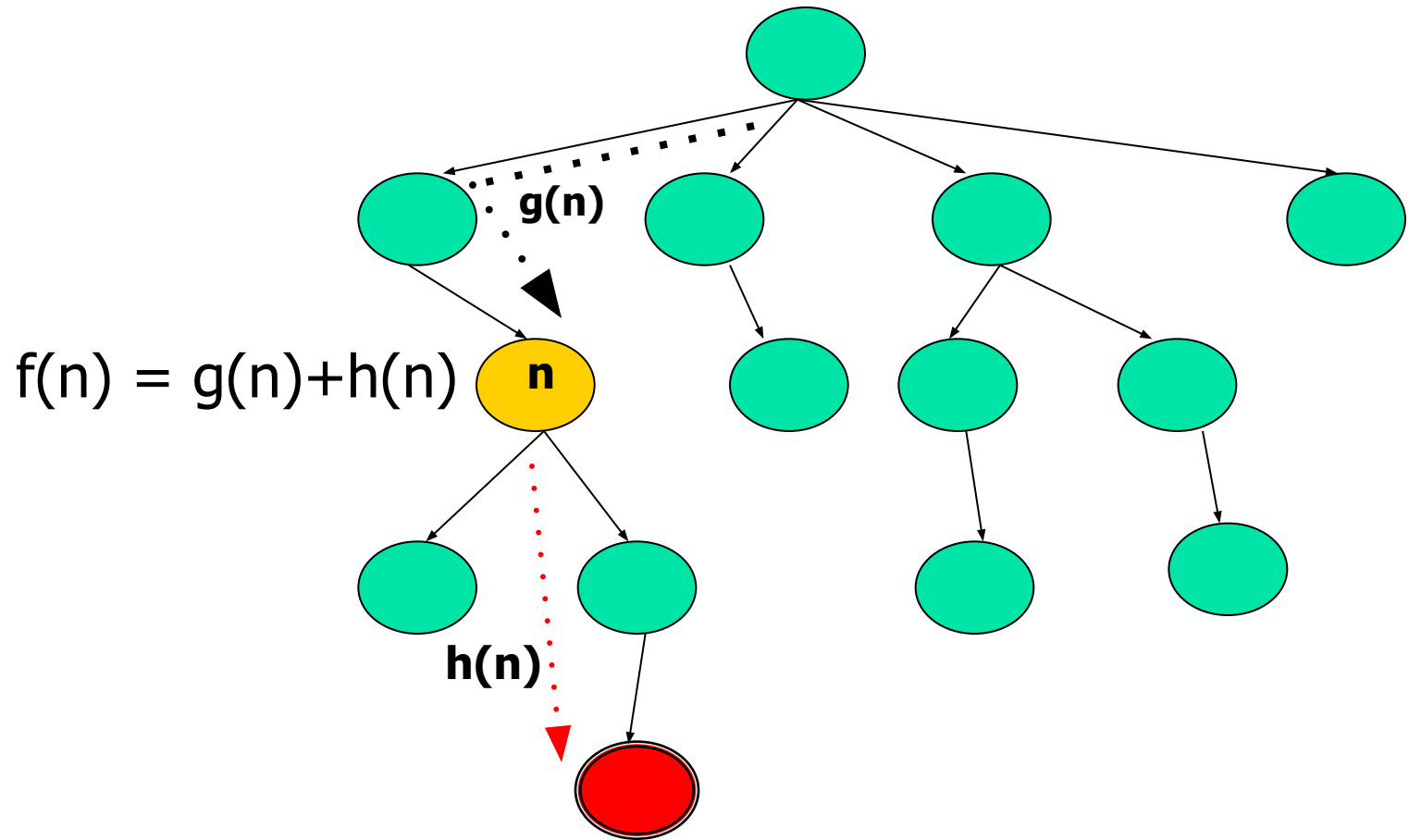
- Greedy Search minimizes a heuristic $h(n)$ which is an estimated cost from a node n to the goal state. Greedy Search is efficient but it is not optimal nor complete.
- Uniform Cost Search minimizes the cost $g(n)$ from the initial state to n . UCS is optimal and complete but not efficient.
- **New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



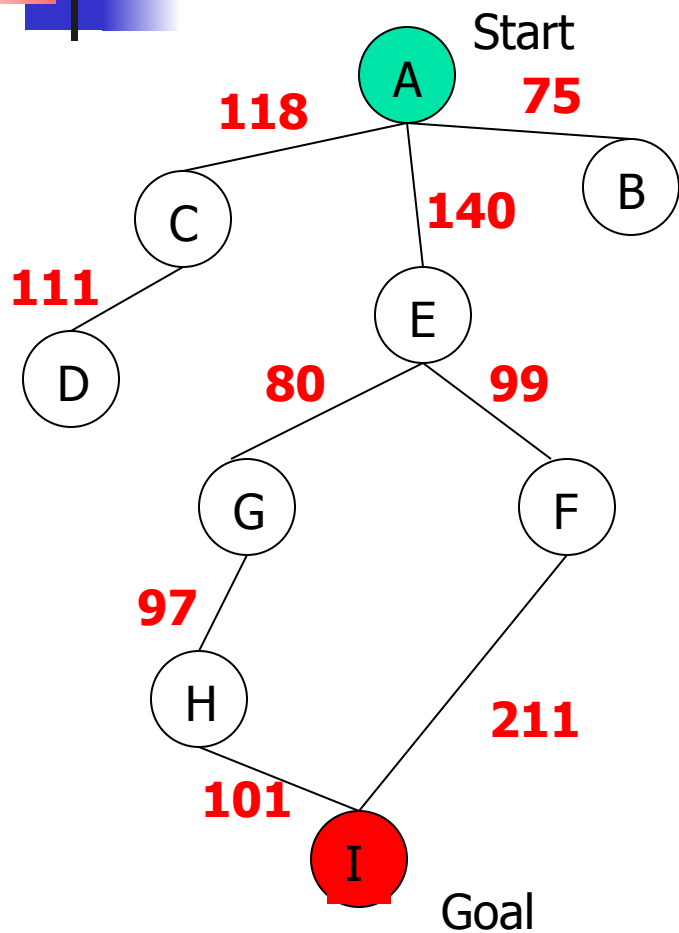
A* (A Star)

- 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)



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



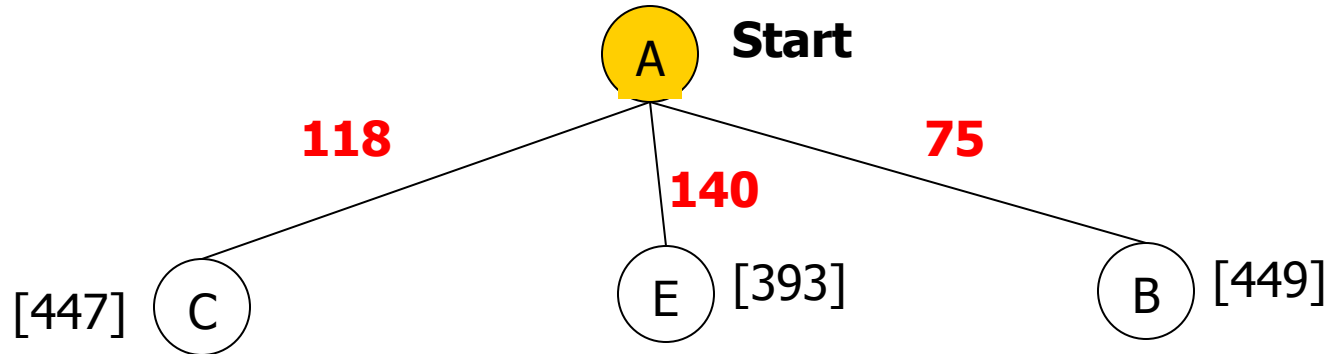
A* Search: Tree Search

A

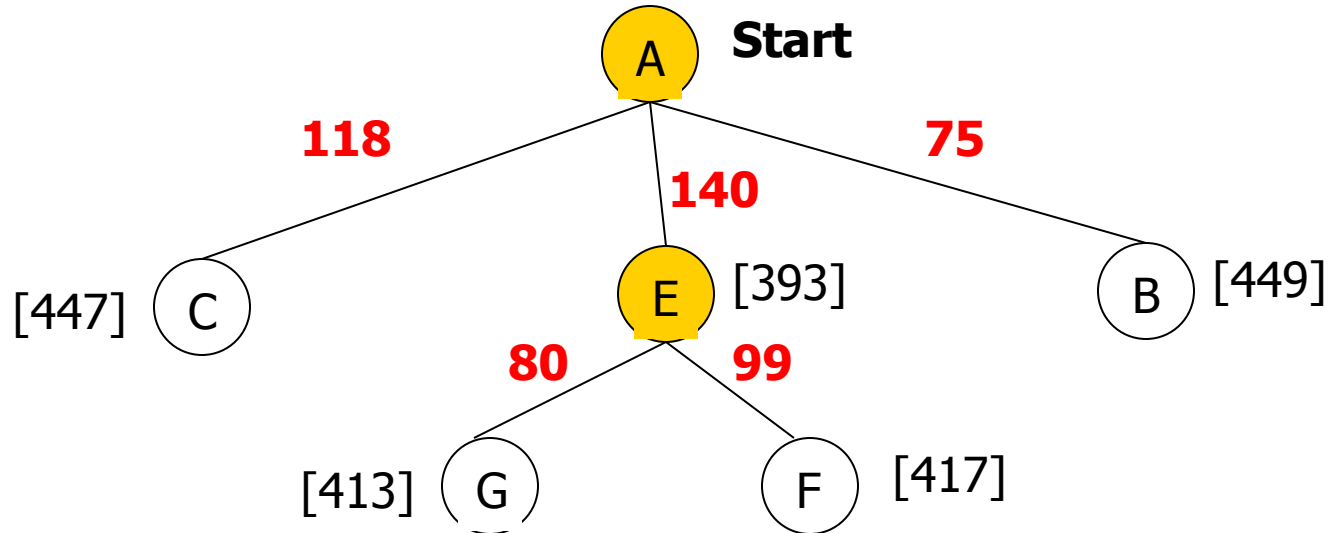
Start



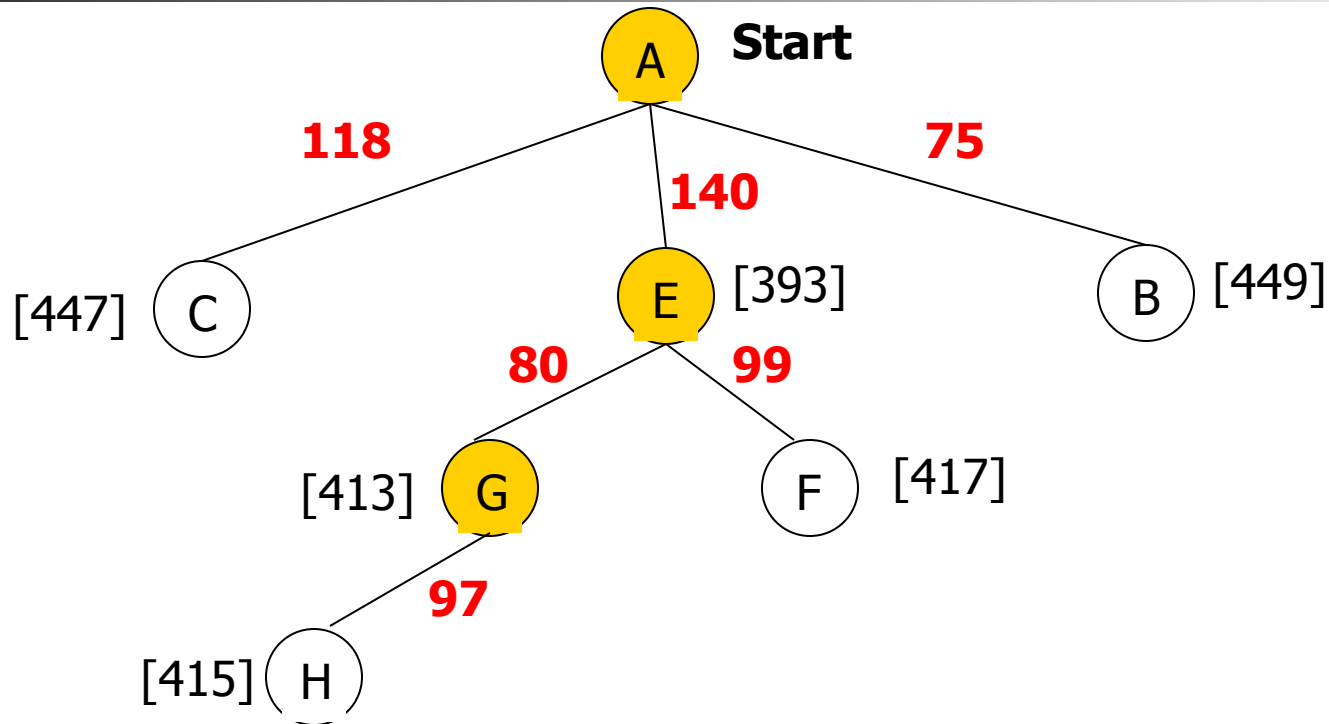
A* Search: Tree Search



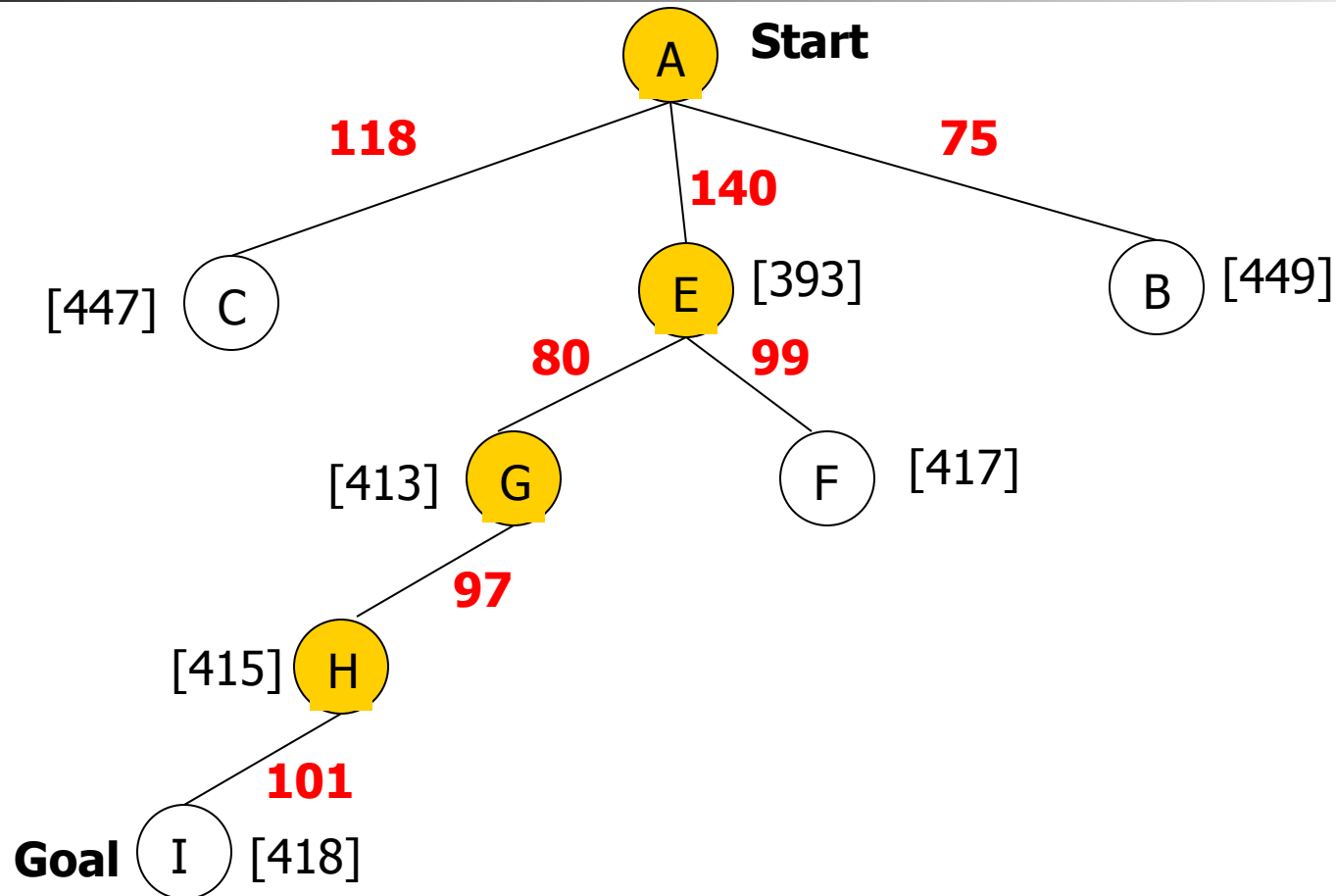
A* Search: Tree Search



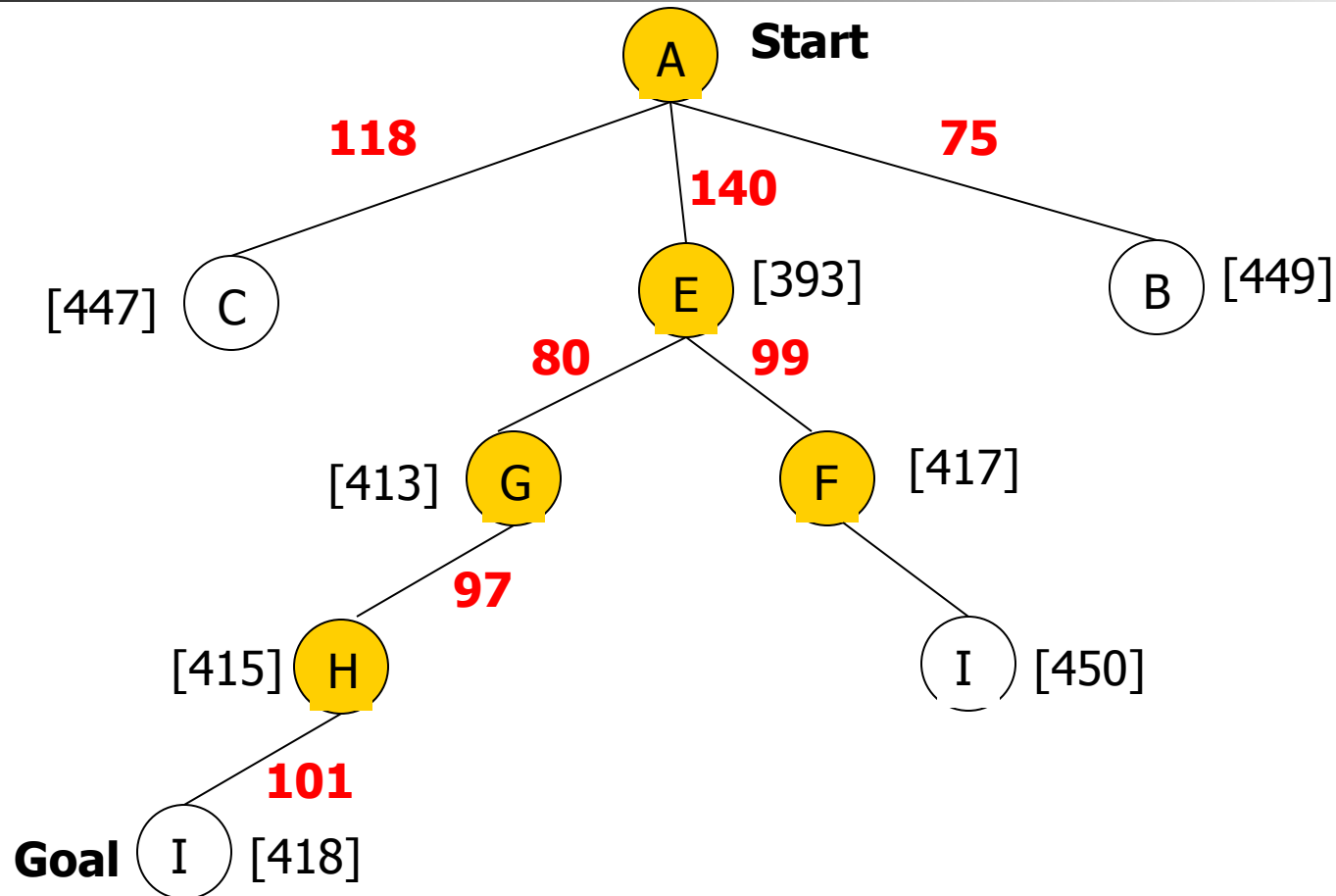
A* Search: Tree Search



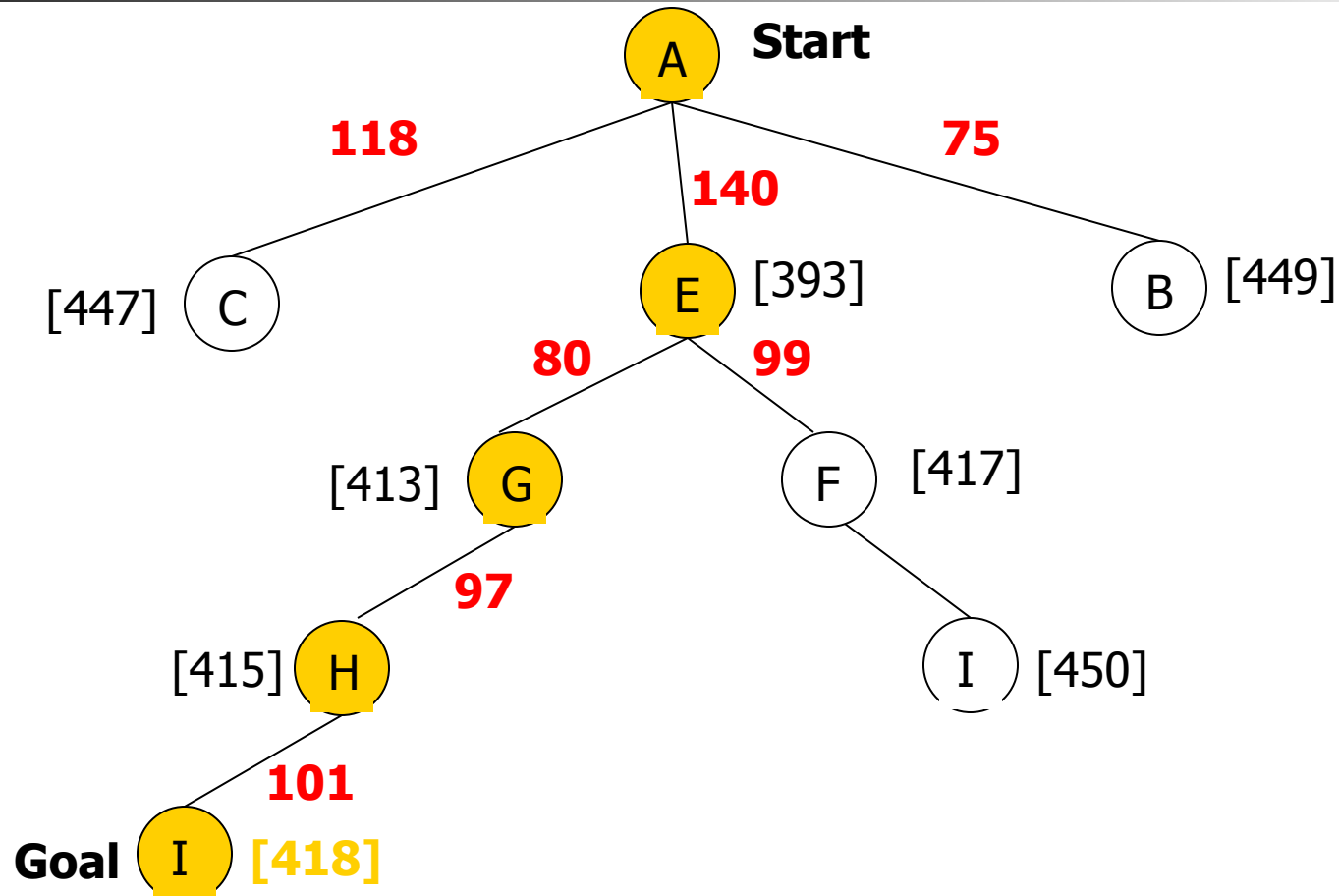
A* Search: Tree Search



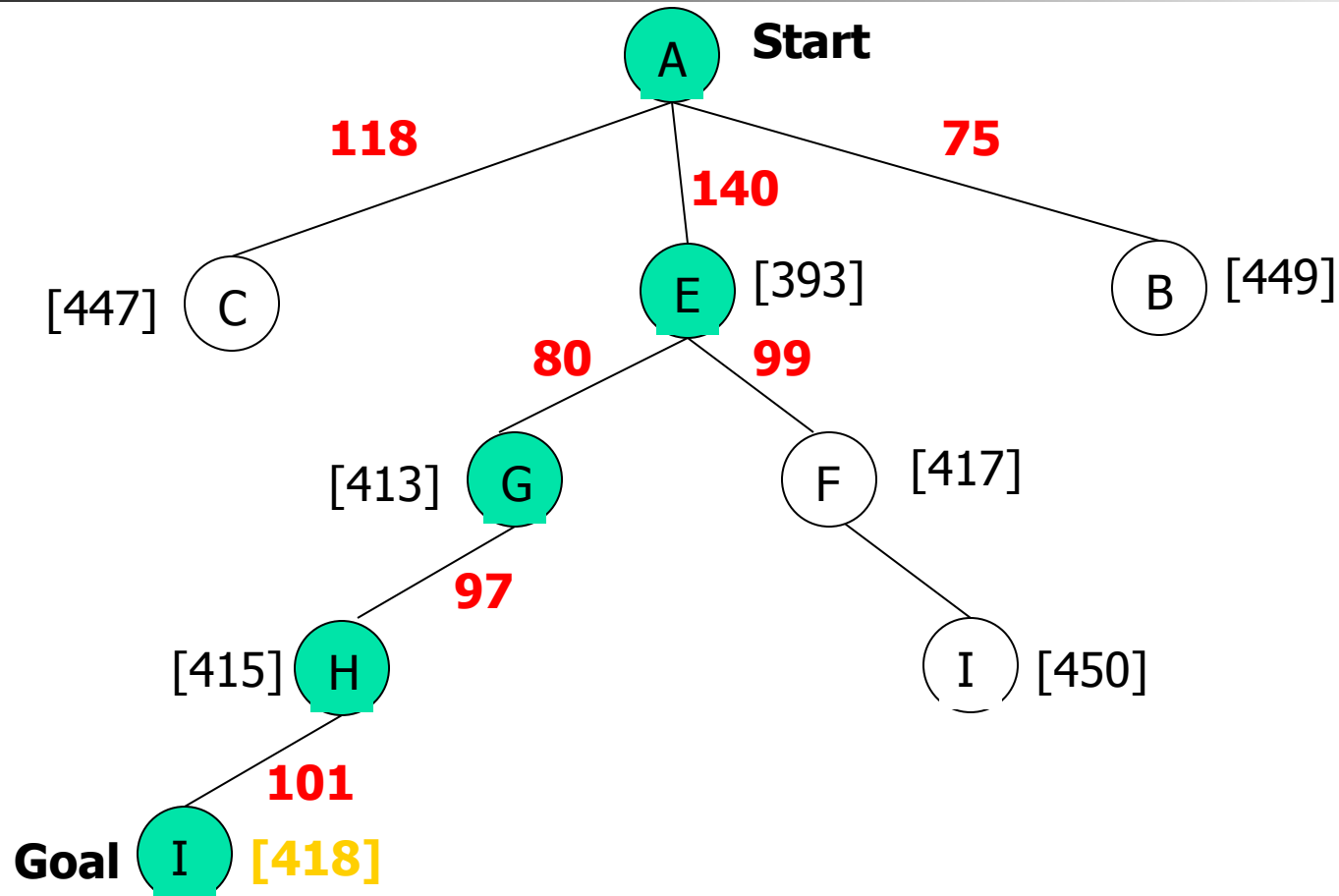
A* Search: Tree Search

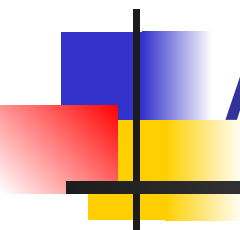


A* Search: Tree Search



A* Search: Tree Search

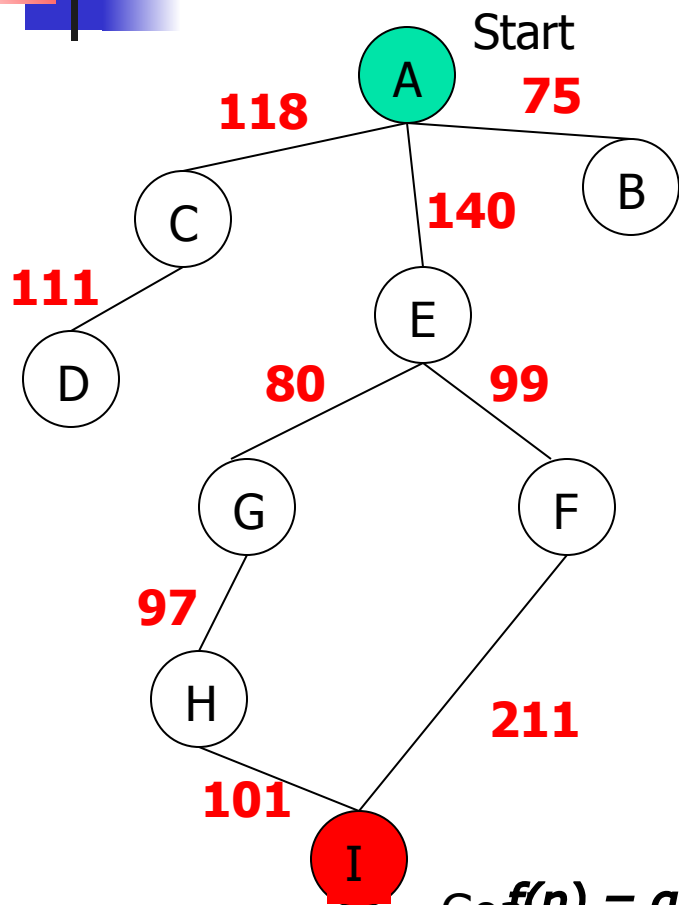




A^* with $f()$ not Admissible

$h()$ overestimates the cost to
reach the goal state

A* Search: h not admissible !



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

Goal $f(n) = g(n) + h(n) - \text{(H-I) Overestimated}$
 $g(n)$: is the exact cost to reach node n from the initial state. 52



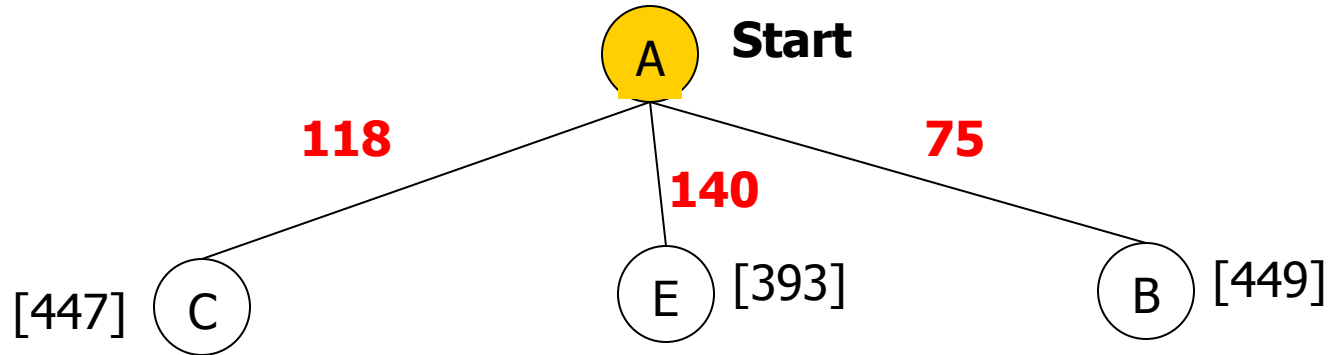
A* Search: Tree Search

A

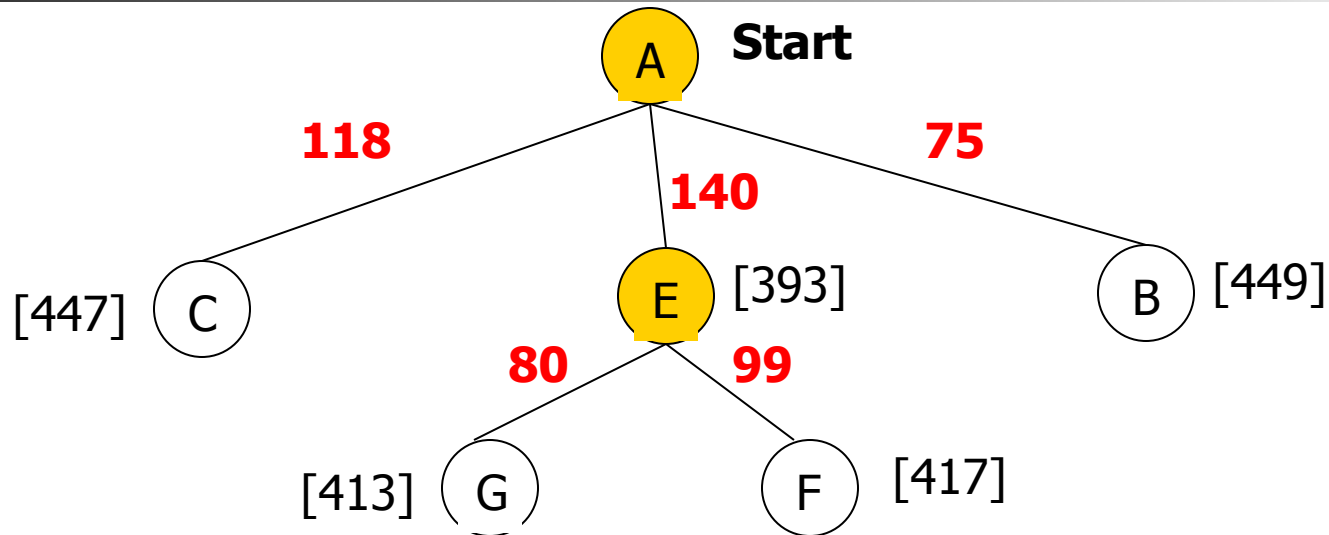
Start



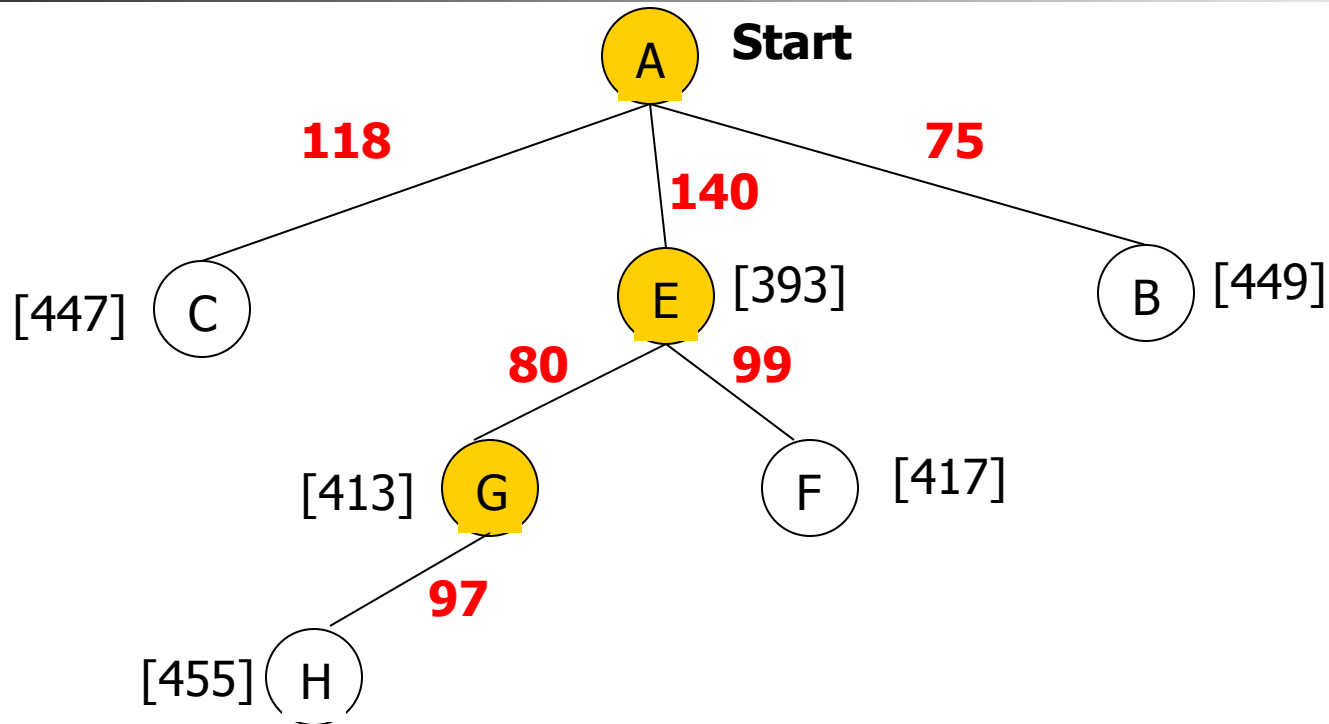
A* Search: Tree Search



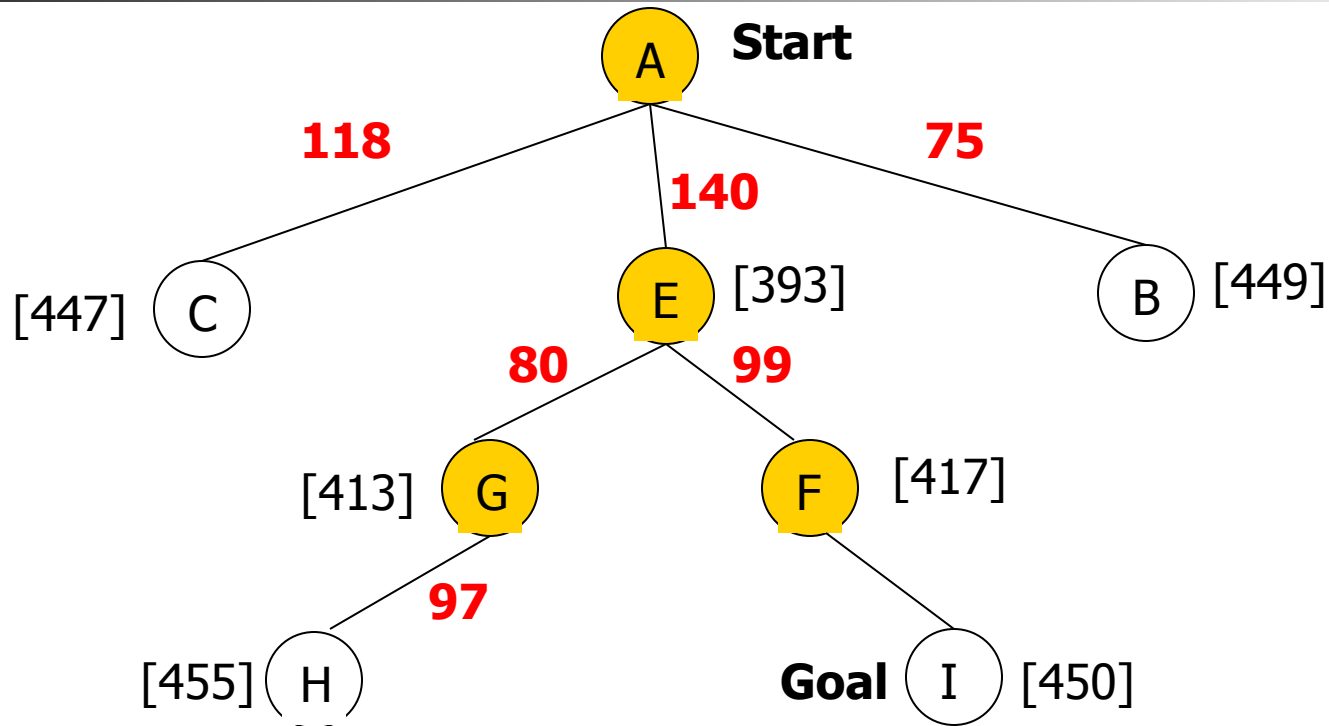
A* Search: Tree Search



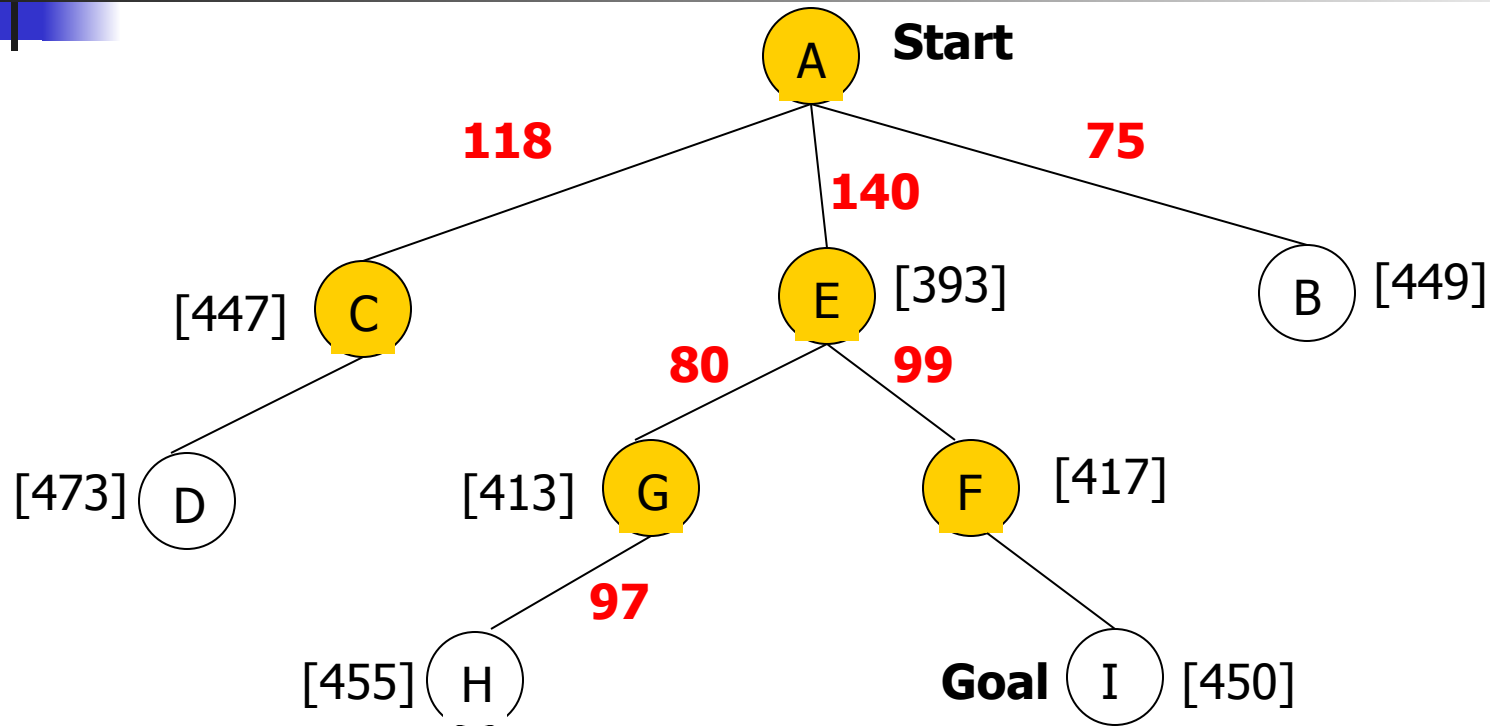
A* Search: Tree Search



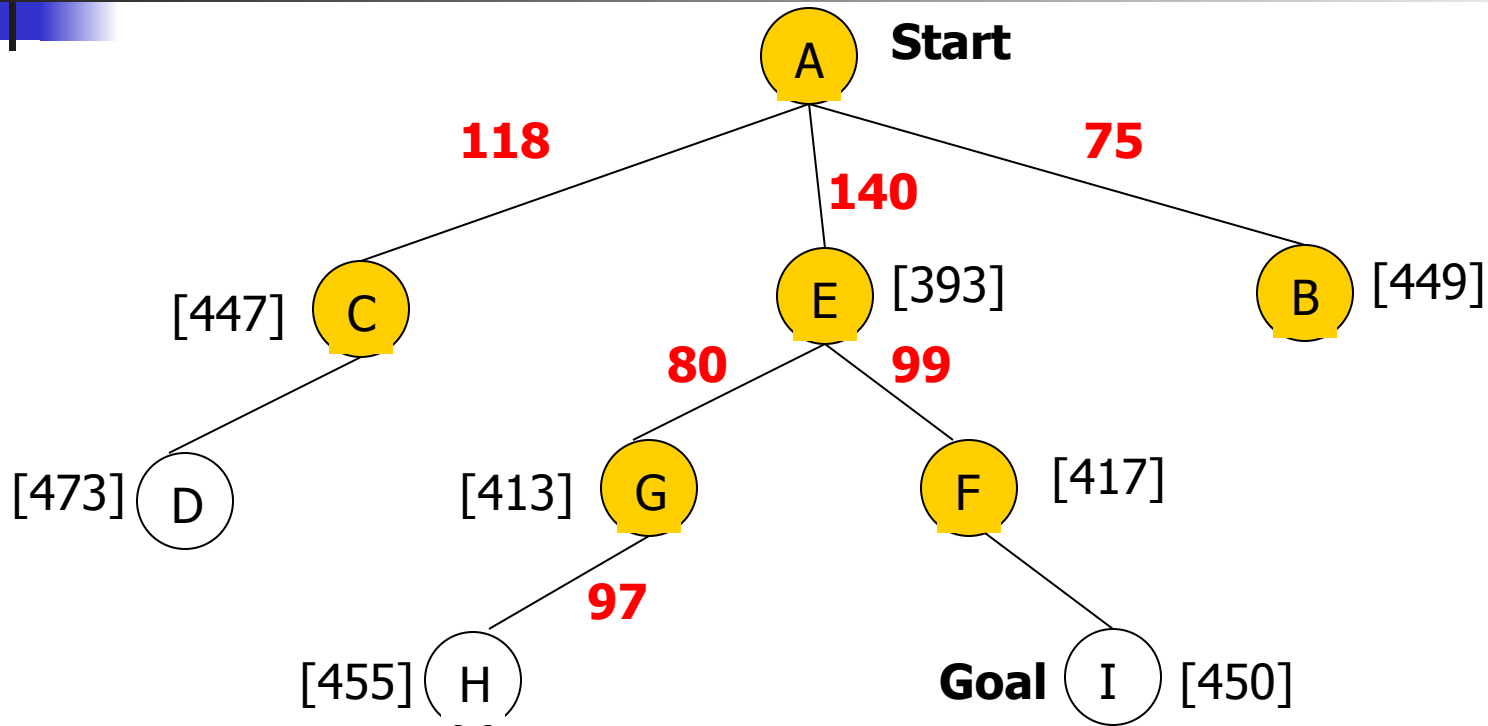
A* Search: Tree Search



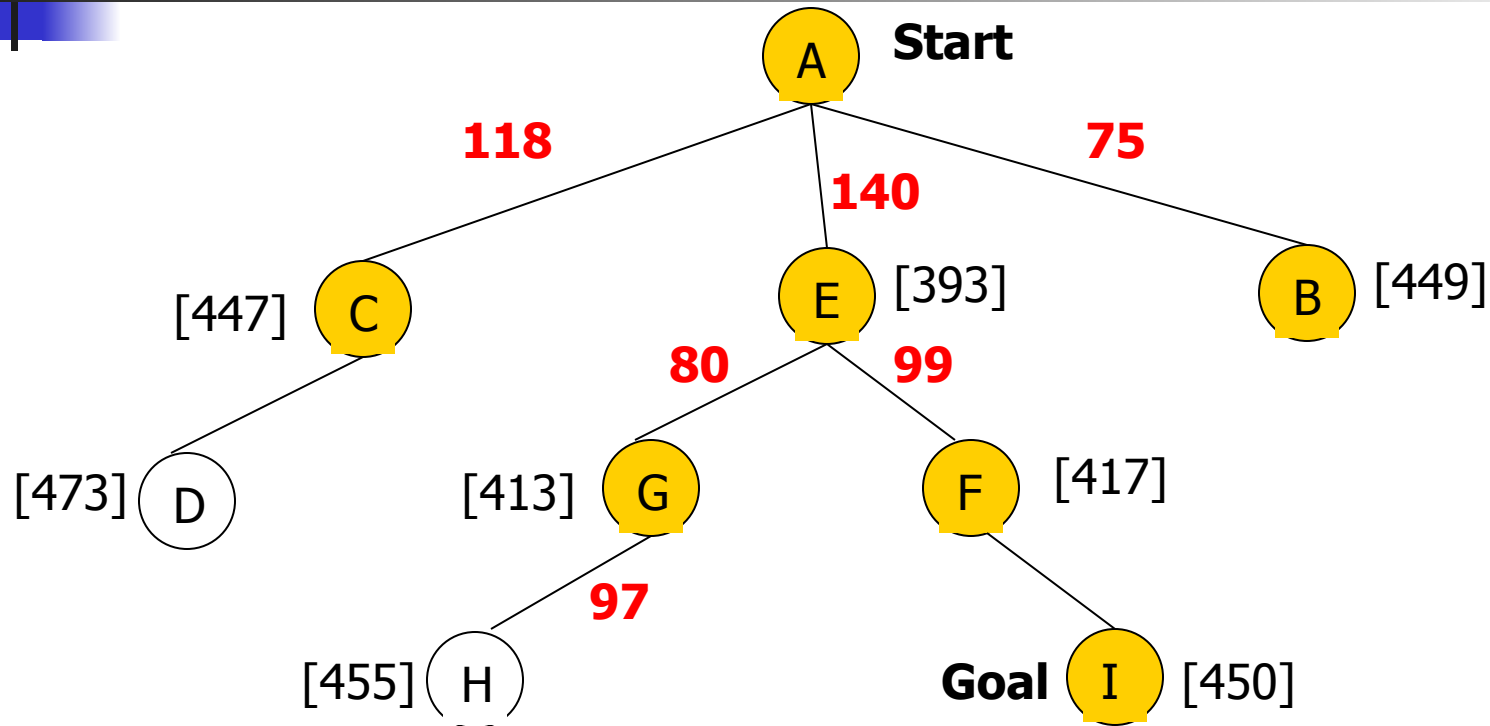
A* Search: Tree Search



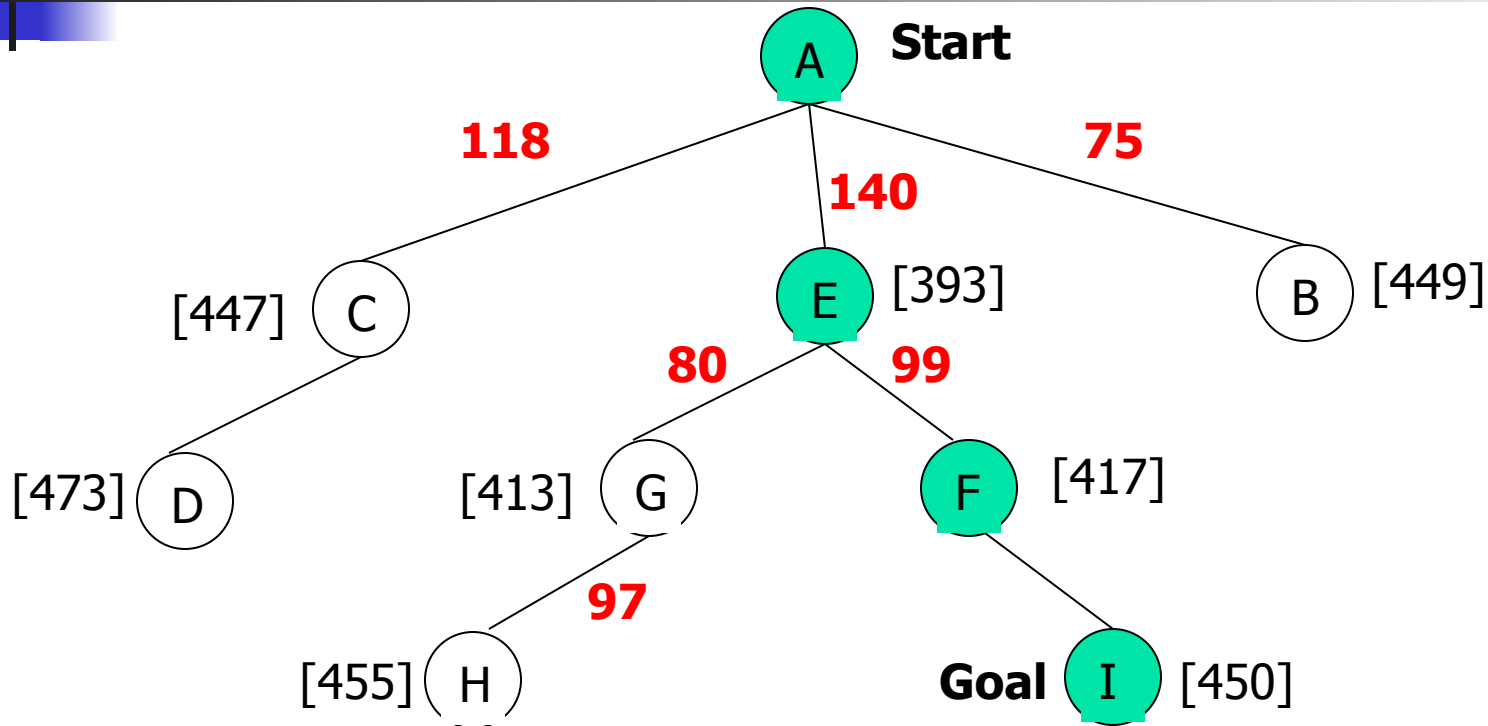
A* Search: Tree Search



A* Search: Tree Search



A* Search: Tree Search



A* not optimal !!!



A* Algorithm

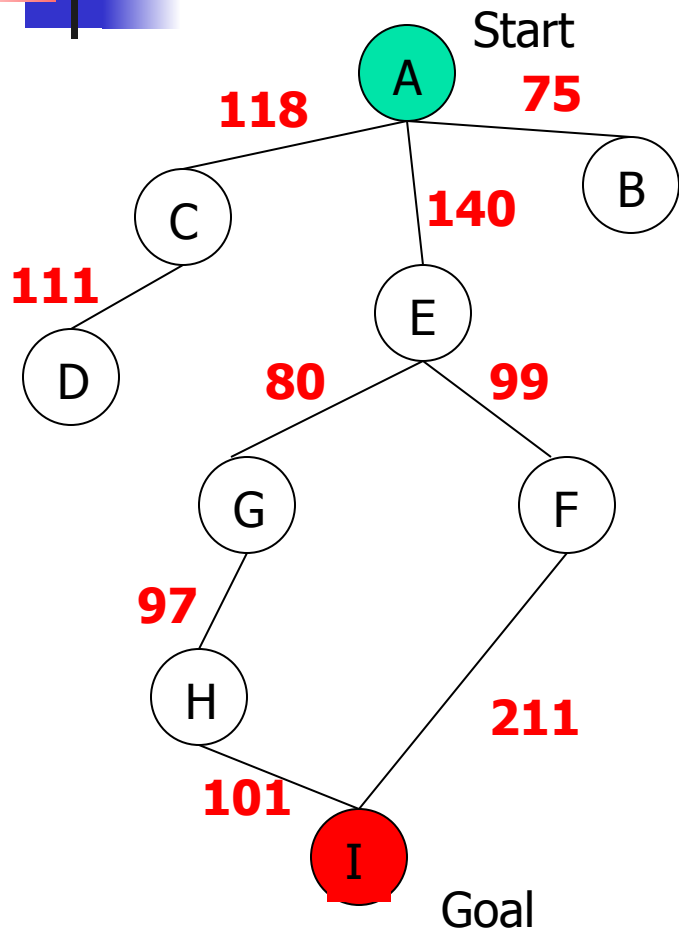
A* with systematic checking for repeated states ...



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 $f(n') = g(n) + \text{cost}(n, n') + h(n')$.
 - 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*).



Informed Search Strategies

Iterative Deepening A^*



Iterative Deepening A*:IDA*

- Use $f(N) = g(N) + h(N)$ with admissible and consistent h
- Each iteration is depth-first with cutoff on the value of f of expanded nodes

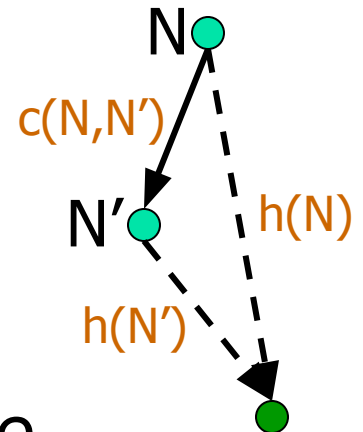
Consistent Heuristic

- The admissible heuristic h is **consistent** (or satisfies the **monotone restriction**) if for every node N and every successor N' of N :

$$h(N) \leq c(N, N') + h(N')$$

(triangular inequality)

- A consistent heuristic is admissible.



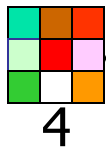


IDA* Algorithm

- In the first iteration, we determine a **“f-cost limit” – cut-off value** $f(n_0) = g(n_0) + h(n_0) = h(n_0)$, where n_0 is the start node.
- We expand nodes using the **depth-first algorithm** and backtrack whenever $f(n)$ for an expanded node n exceeds the cut-off value.
- If this search does not succeed, determine the **lowest f-value** among the nodes that were visited but not expanded.
- Use this f-value as the **new limit value – cut-off value** and do another depth-first search.
- Repeat this procedure until a goal node is found.

8-Puzzle

$f(N) = g(N) + h(N)$
with $h(N)$ = number of misplaced tiles

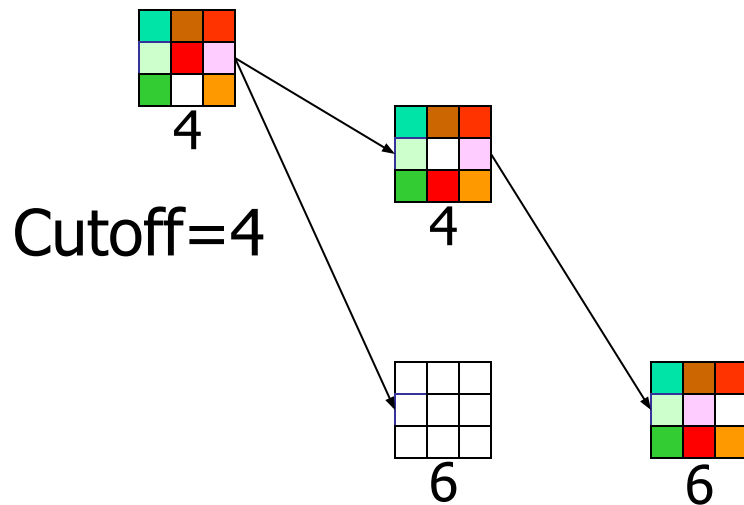


Cutoff=4



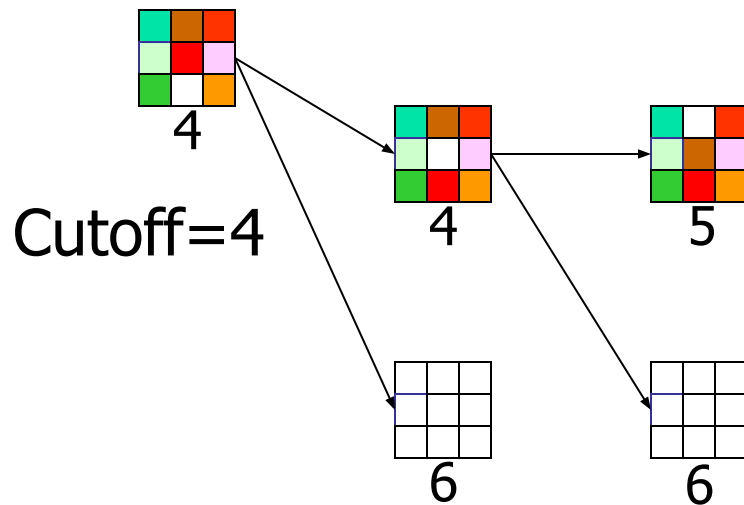
8-Puzzle

$f(N) = g(N) + h(N)$
with $h(N)$ = number of misplaced tiles



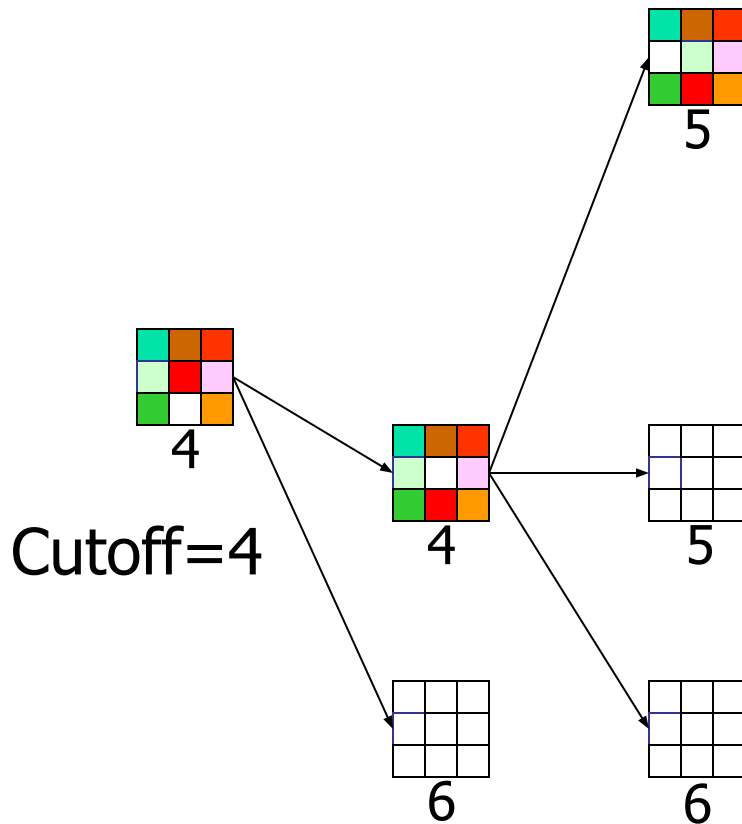
8-Puzzle

$f(N) = g(N) + h(N)$
with $h(N)$ = number of misplaced tiles



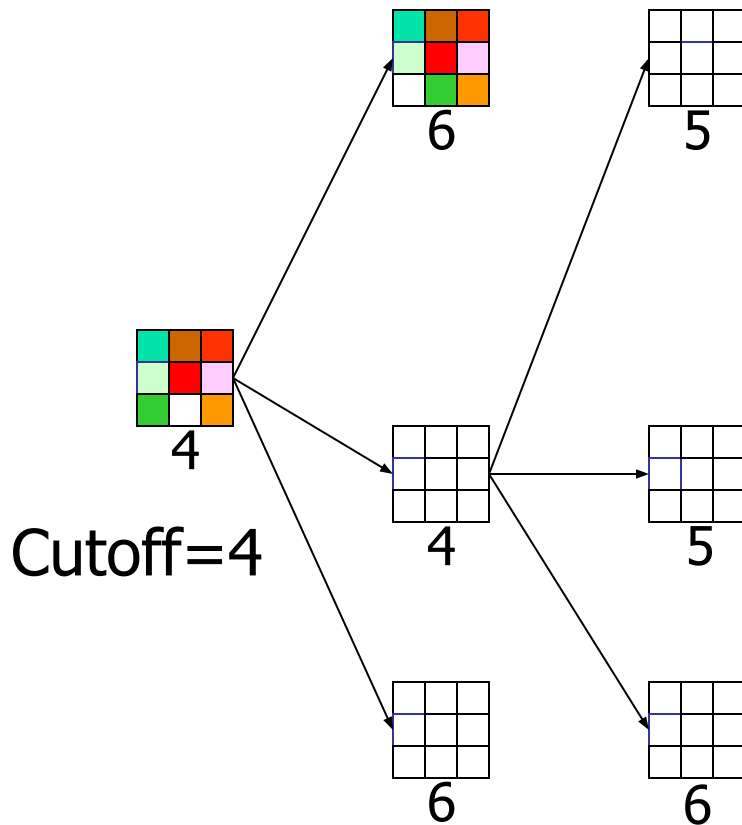
8-Puzzle

$f(N) = g(N) + h(N)$
with $h(N)$ = number of misplaced tiles



8-Puzzle

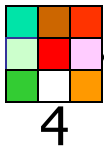
$f(N) = g(N) + h(N)$
with $h(N)$ = number of misplaced tiles





8-Puzzle

$f(N) = g(N) + h(N)$
with $h(N)$ = number of misplaced tiles

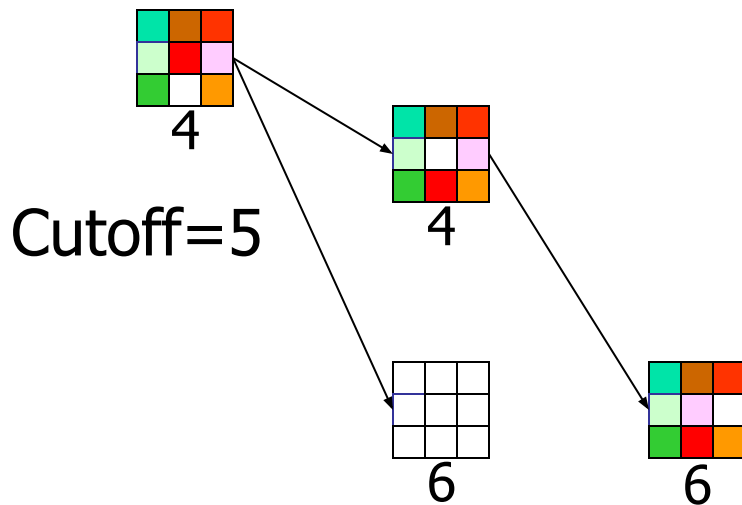


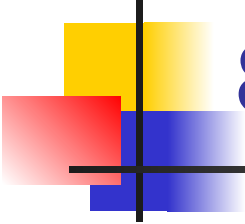
Cutoff=5



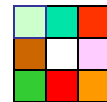
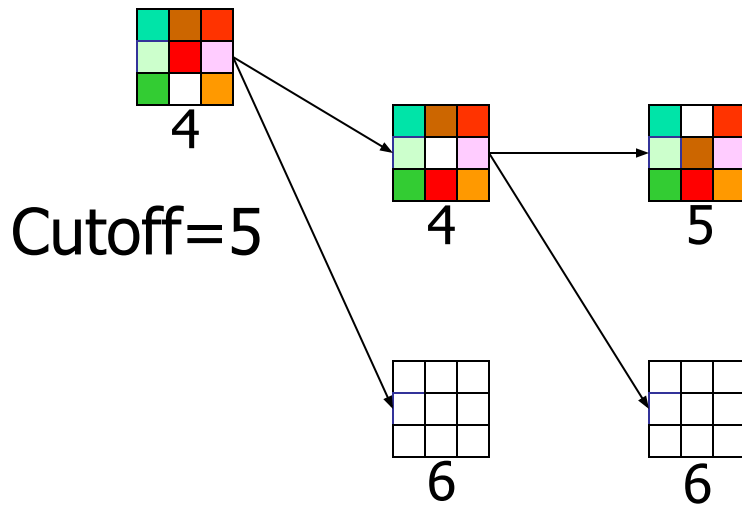
8-Puzzle

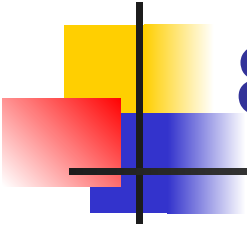
$f(N) = g(N) + h(N)$
with $h(N)$ = number of misplaced tiles



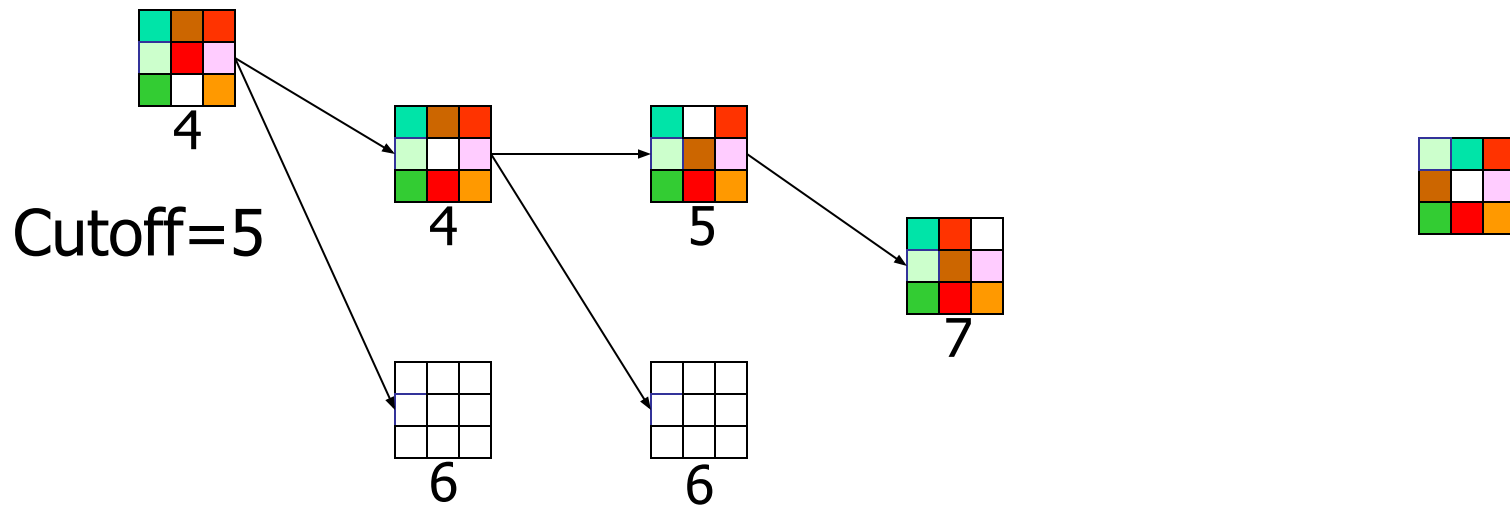


$f(N) = g(N) + h(N)$
with $h(N)$ = number of misplaced tiles



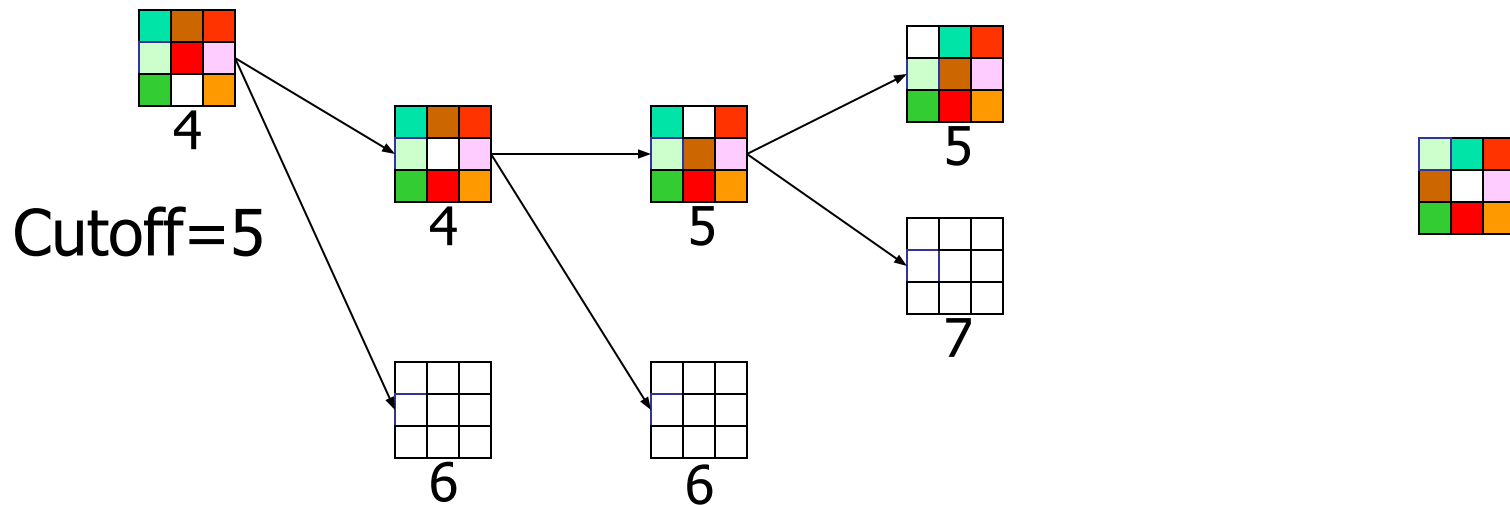


$f(N) = g(N) + h(N)$
with $h(N)$ = number of misplaced tiles



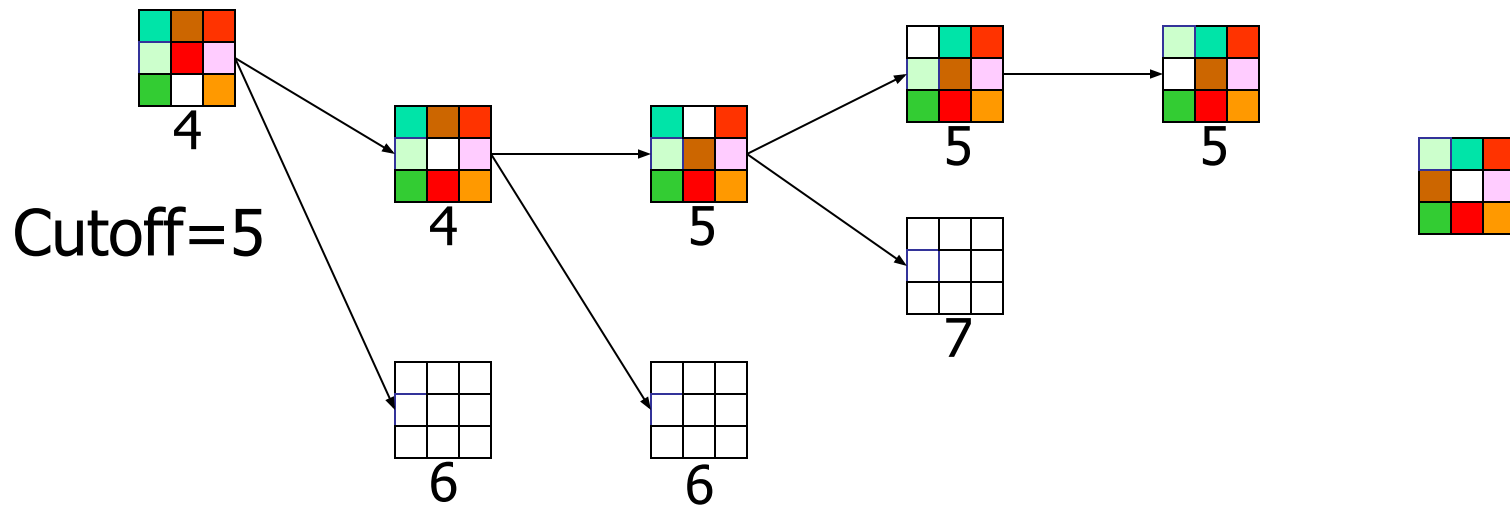
8-Puzzle

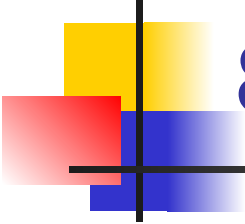
$f(N) = g(N) + h(N)$
with $h(N)$ = number of misplaced tiles



8-Puzzle

$f(N) = g(N) + h(N)$
with $h(N)$ = number of misplaced tiles





with $h(N)$ = number of misplaced tiles



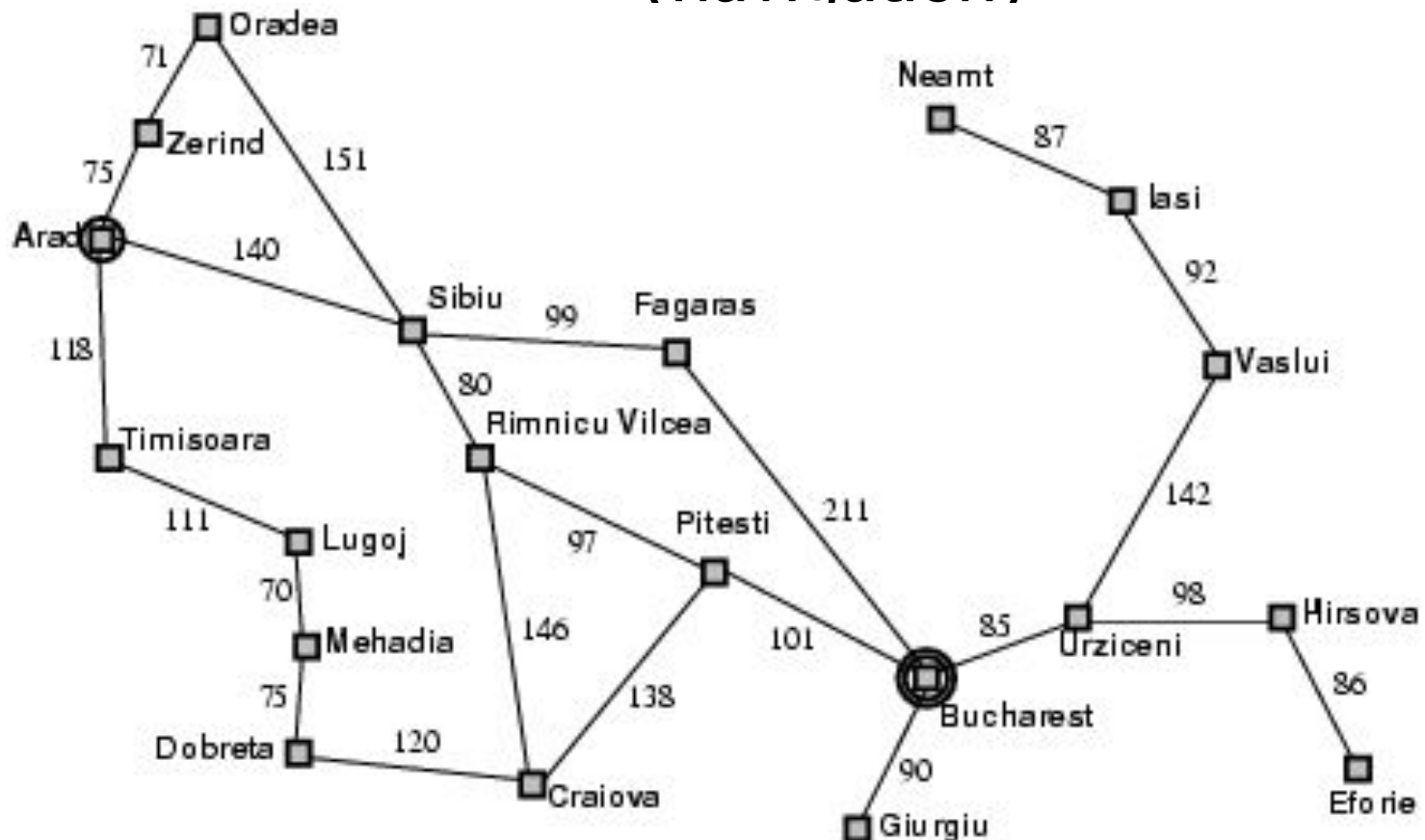
When to Use Search Techniques



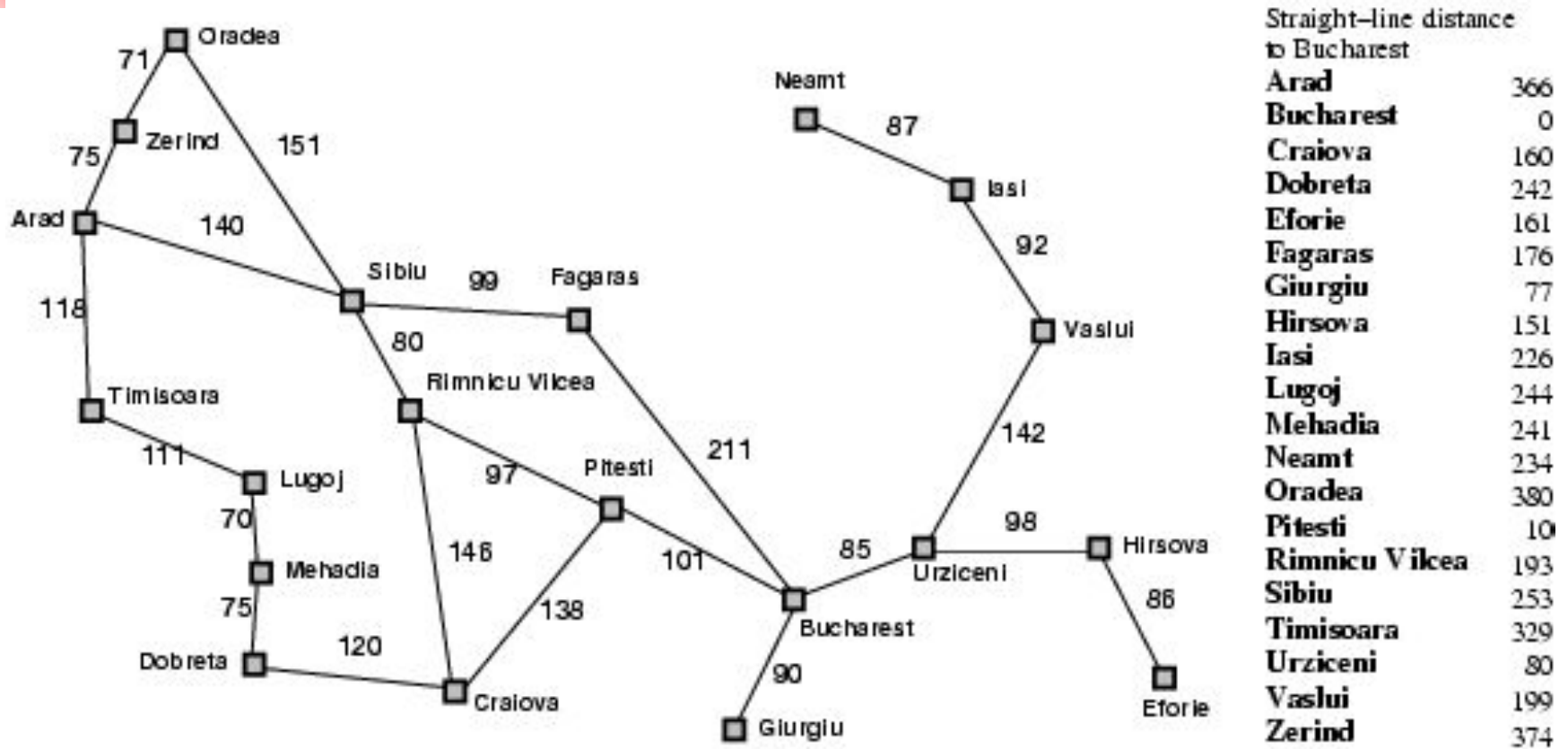
- The search space is small, and
 - There are no other available techniques, or
 - It is not worth the effort to develop a more efficient technique
- The search space is large, and
 - There is no other available techniques, and
 - There exist “good” heuristics

Popular AI Search Problems

Classic AI search problems, Map searching (navigation)



Romania with step costs in km



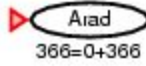


A* search

- 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



Arad
366=0+366

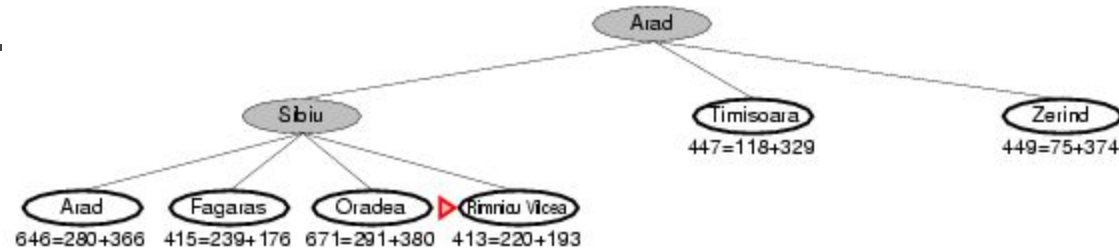


A* search example



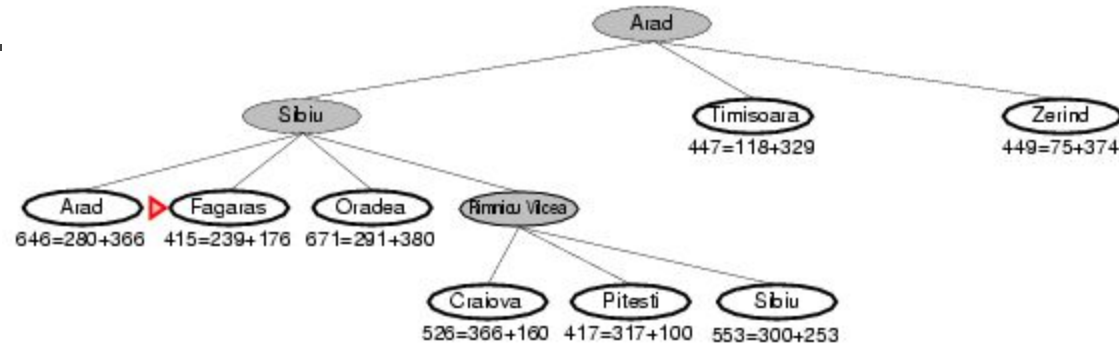


A* search example



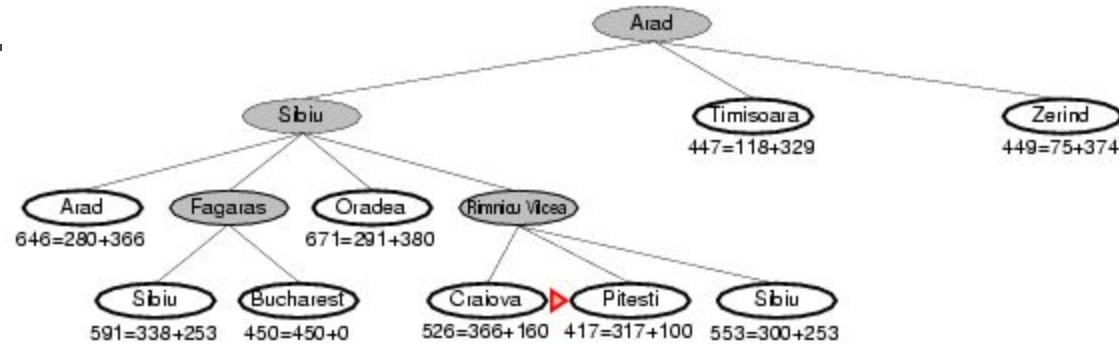


A* search example



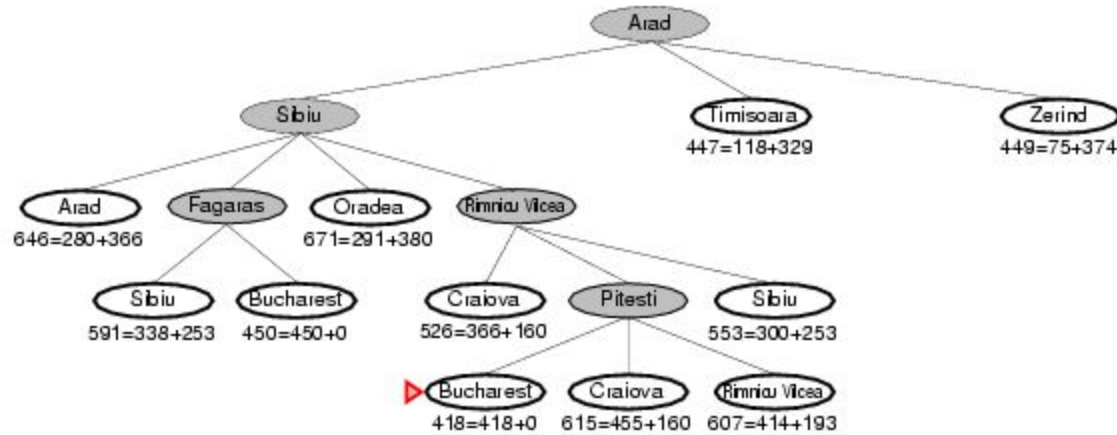


A* search example





A* search example



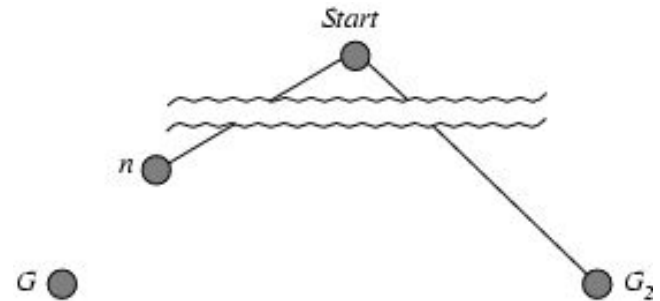


Admissible heuristics

- 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

Optimality of A^* (proof)

- 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 .

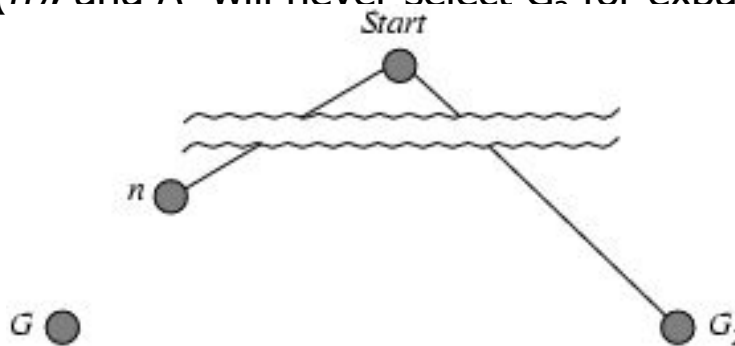


- $f(G_2) = g(G_2) + h(G_2)$
- $f(G_2) = g(G_2)$ [since $h(G_2) = 0$](1)
- Again, $f(G) = g(G) + h(G)$
- $f(G) = g(G)$ [since $h(G) = 0$](2)
- But, $g(G_2) > g(G)$ [since G_2 is suboptimal].....(3)
- Therefore, $f(G_2) > f(G)$ [from equation (1), (2) and (3)] (4)

Optimality of A^* (proof)

- 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 .
- Therefore, $f(G_2) > f(G)$ (4) [from equation (1), (2) and (3)]
- Again, $h(n) \leq h^*(n)$ (5) [since h is admissible; Here, $h^*(n)$ is the **true** cost to reach the goal state from n]
- $g(n) + h(n) \leq g(n) + h^*(n)$ (6) [Adding $g(n)$ in both sides of equation (5)]
- $f(n) \leq f(G)$ (7) [Because, $f(n) = g(n) + h(n)$; and $f(G) = g(n) + h^*(n)$; Here, $h^*(n)$ is the **true** cost to reach the goal state from n]
 $f(n) \leq f(G) < f(G_2)$ [From equation (4) and (7)]

Therefore, $f(G_2) > f(n)$. and A^* will never select G_2 for expansion before expanding n to reach at optimal goal G .

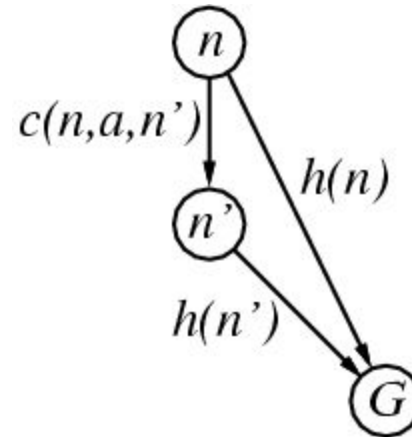


Consistent heuristics

- 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



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



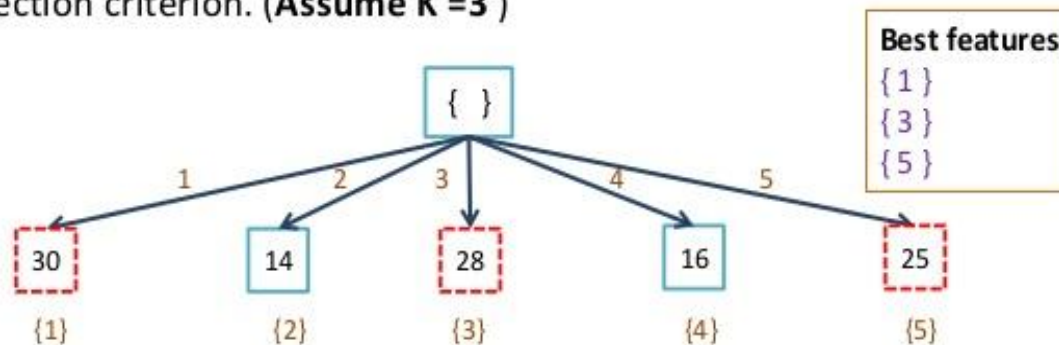
Local beam search

- Keep track of k states rather than just one
 - Start with k randomly generated states
 - At each iteration, all the successors of all k states are generated
 - If any one is a goal state, stop; else select the k best successors from the complete list and repeat.

Local Beam Search

- Begin with k random states
- Generate all successors of these states
- Keep the k best states
- Stochastic beam search: Probability of keeping a state is *a function* of its heuristic value

- Select the best K (beam width) features based on a pre-defined selection criterion. (Assume $K = 3$)





Conclusions

- Frustration with *uninformed* search led to the idea of using domain specific knowledge in a search so that one can intelligently explore only the relevant part of the search space that has a good chance of containing the goal state. These new techniques are called informed (heuristic) search strategies.
- Even though heuristics improve the performance of informed search algorithms, they are still time consuming especially for large size instances.



References

- University of Berkeley, USA
- <http://www.aima.cs.berkeley.edu>