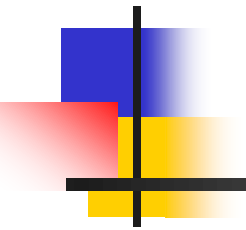
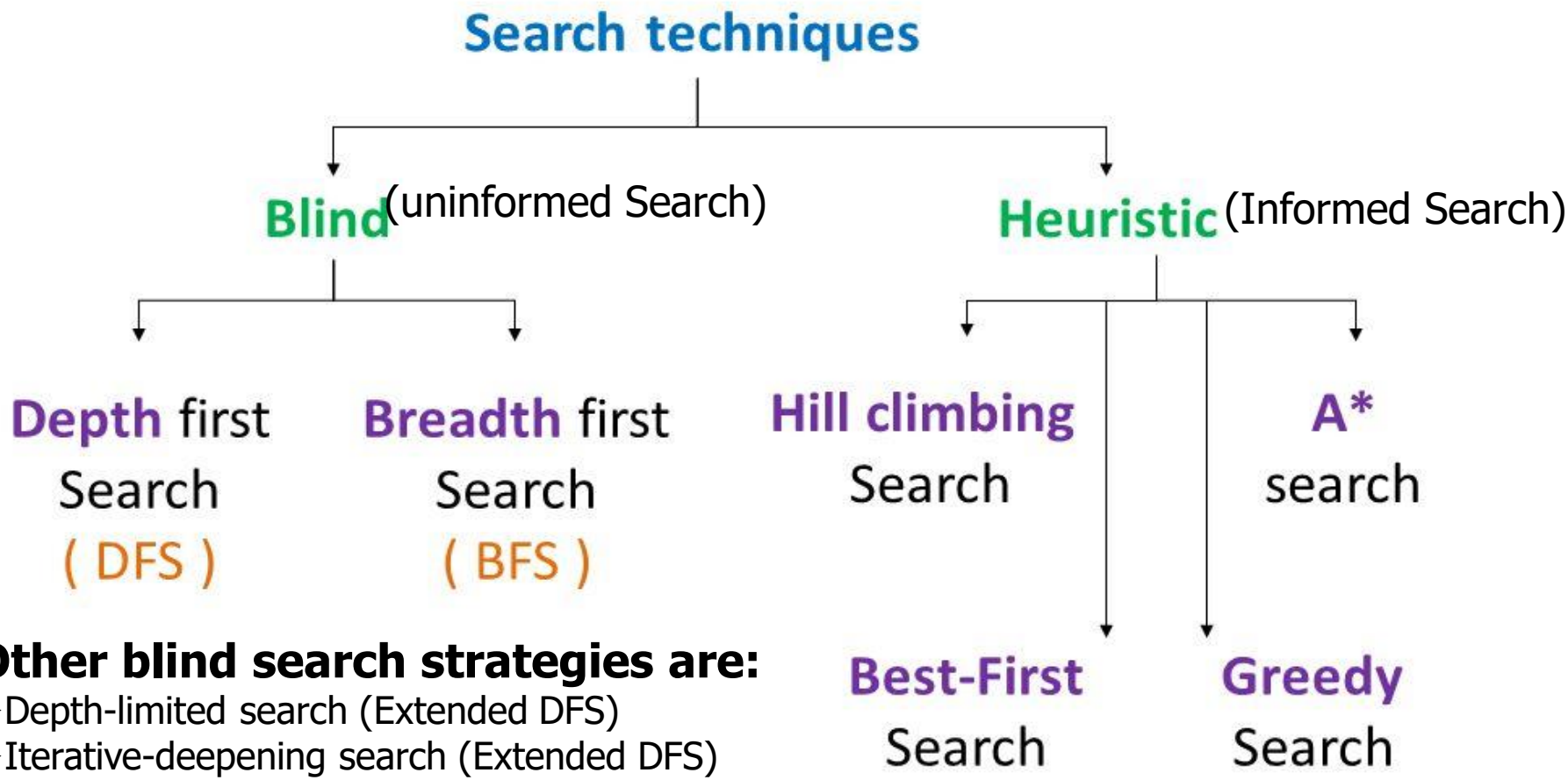


Artificial Intelligence
Informed Search

SEARCH TECHNIQUES



Other blind search strategies are:

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

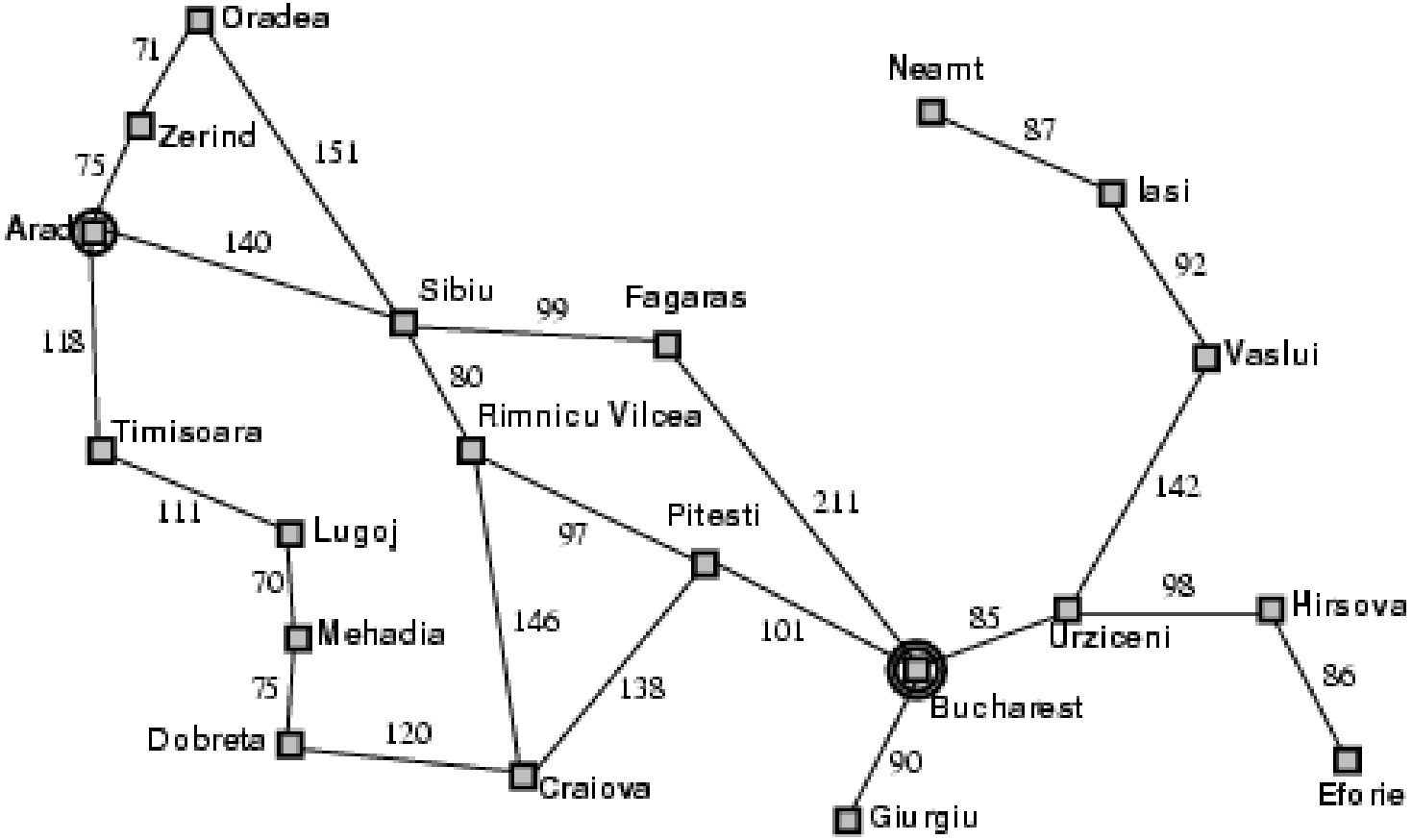
Uninformed Vs Informed Search

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

Popular AI Search Problems

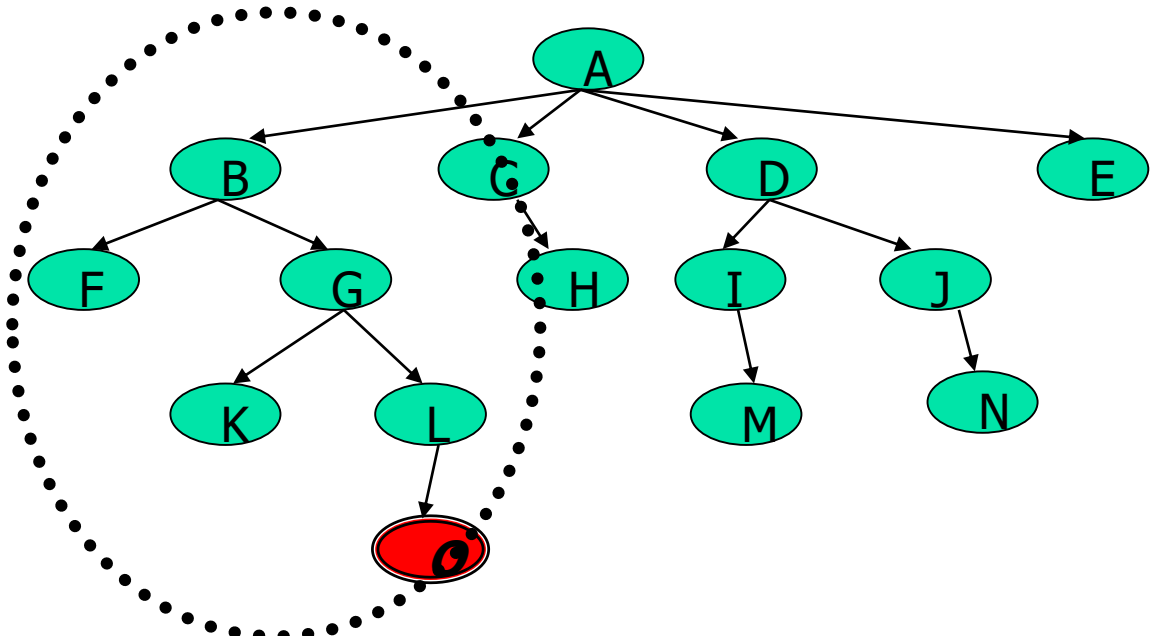
Classic AI search problems, Map searching (navigation)



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 *f(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.

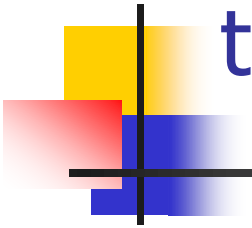
- Three approaches to defining *f*:

  - *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)

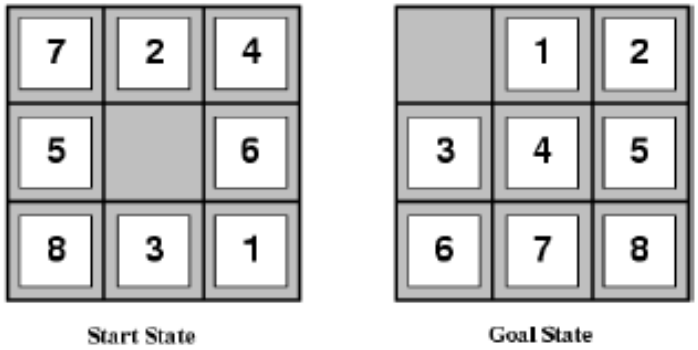
Approach 1: *f* Measures the Value of the Current State



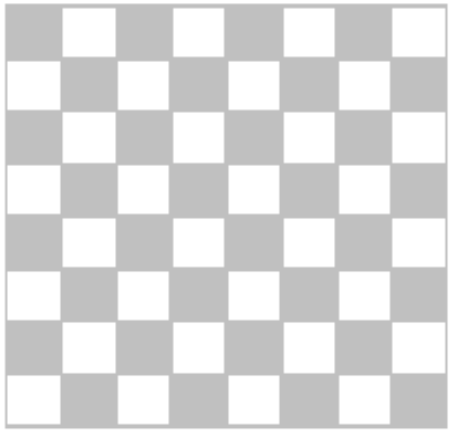
- Usually the case when solving optimization problems
  - Finding a state such that the value of the metric *f* is optimized
- Often, in these cases, *f* could be a weighted sum of a set of component values:
  - N-Queens
    - Example: the number of queens under attack …
  - Data mining
    - Example: the "predictive-ness" (a.k.a. accuracy) of a rule discovered

Heuristic Functions

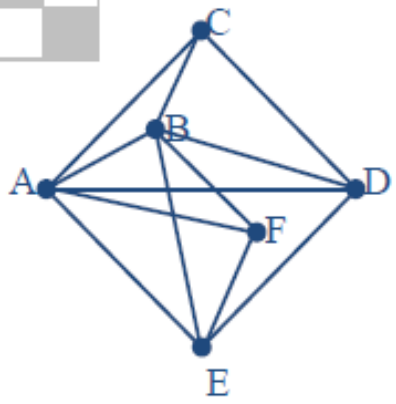
- 8-puzzle
  - Number of misplaced tiles
  - Manhattan distance
  - Gaschnig's



- 8-queen
  - Number of future feasible slots
  - Min number of feasible slots in a row
  - Min number of conflicts (in complete assignments states)



- Travelling salesperson
  - Minimum spanning tree
  - Minimum assignment problem



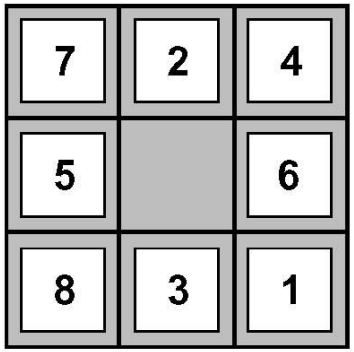
Approach 2: *f* Measures the Cost to the Goal

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*

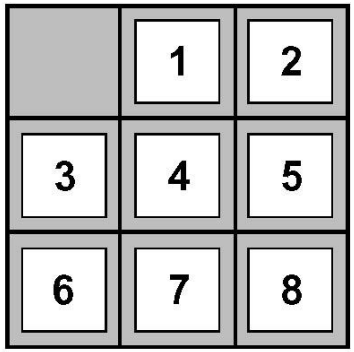
- 8–Puzzle

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

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



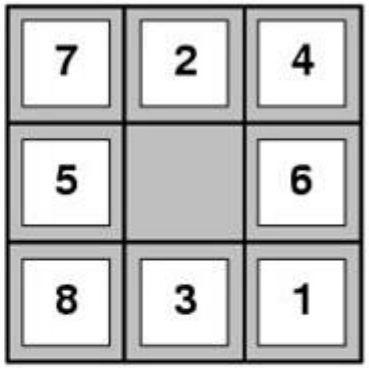
Start State



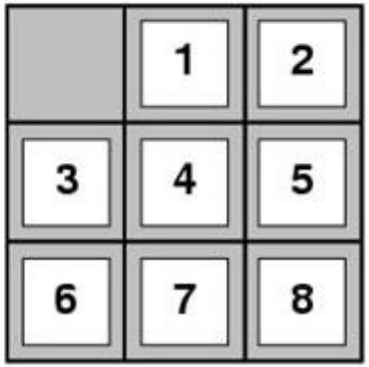
Goal State

Heuristic functions

- sample heuristics for 8-puzzle:
  - $h_1(n)$ = number of misplaced tiles
  - $h_2(n)$ = total Manhattan distance
- $h_1(S) = 8$
- $h_2(S) = 3+1+2+2+2+3+3+2 = 18$
- dominance:
  - $h_2(n) \geq h_1(n)$ for all n (both admissible)
  - h_2 is better for search (closer to perfect)
  - less nodes need to be expanded



Start State



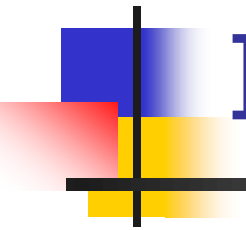
Goal State

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

- A heuristic function $f(n) = g(n) + h(n)$ 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 "admissible" heuristic function** *f* that takes into account the current cost *g(n) and h(n)*
  - Always returns the optimal solution path



Informed Search Strategies

Best First Search

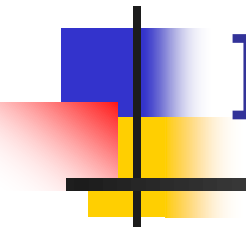
An implementation of Best First Search

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*)

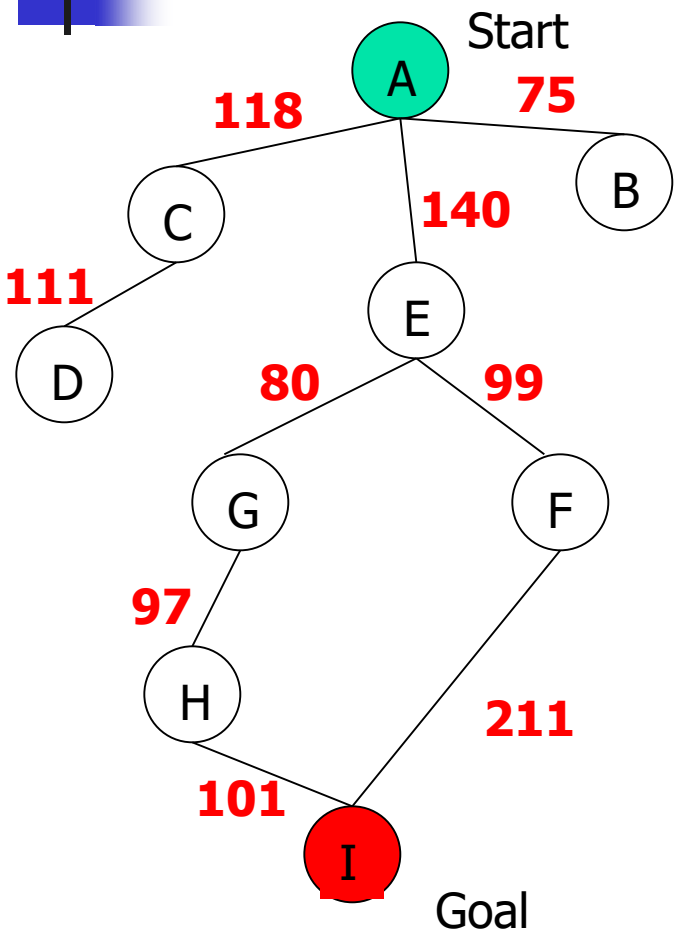


Informed Search Strategies

Greedy Search

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

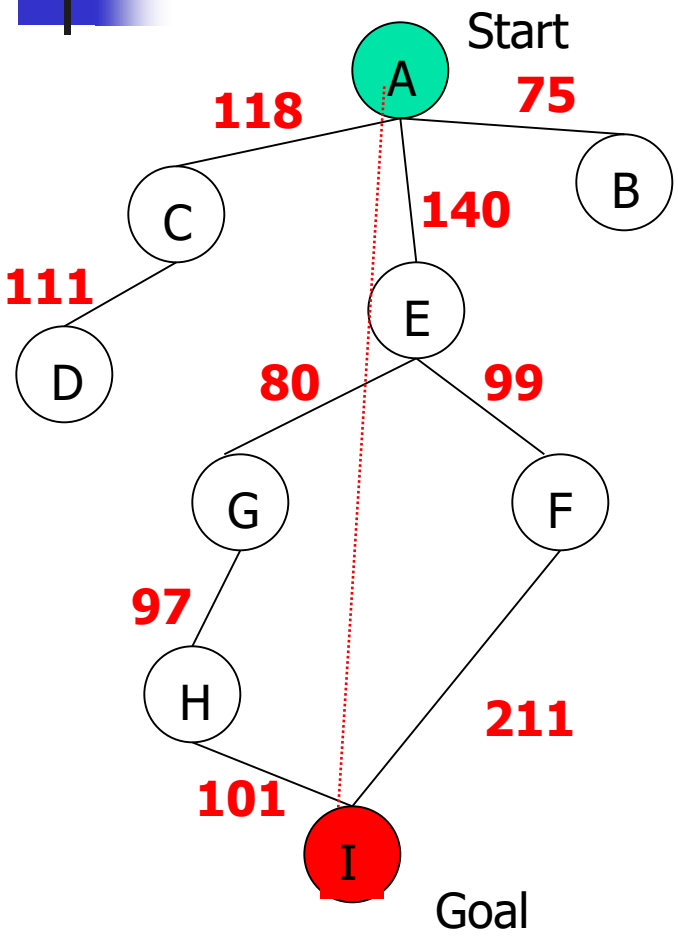
Greedy Search



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**

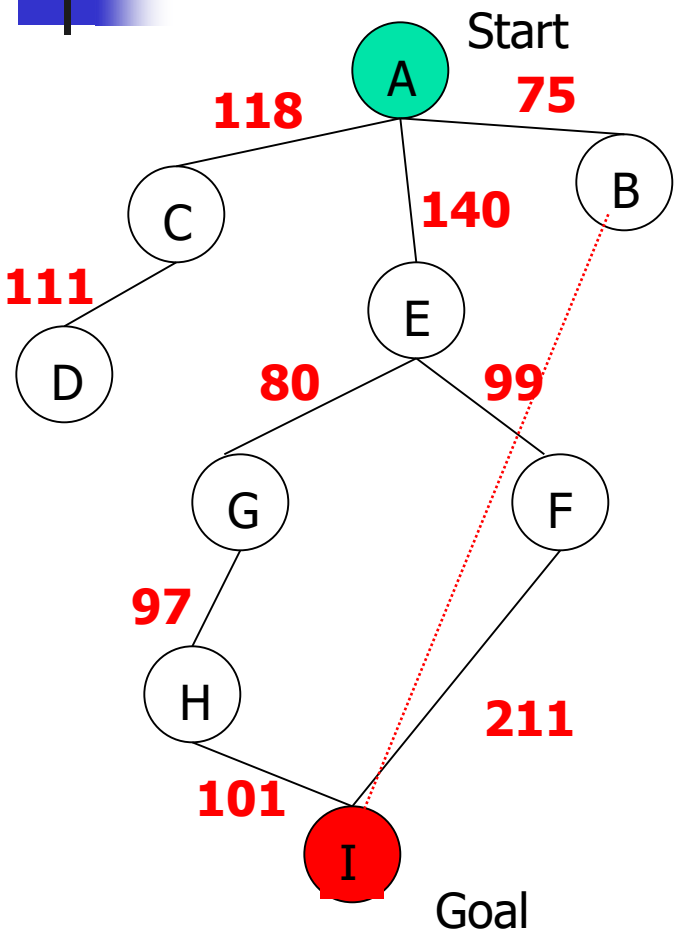
Greedy Search



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**

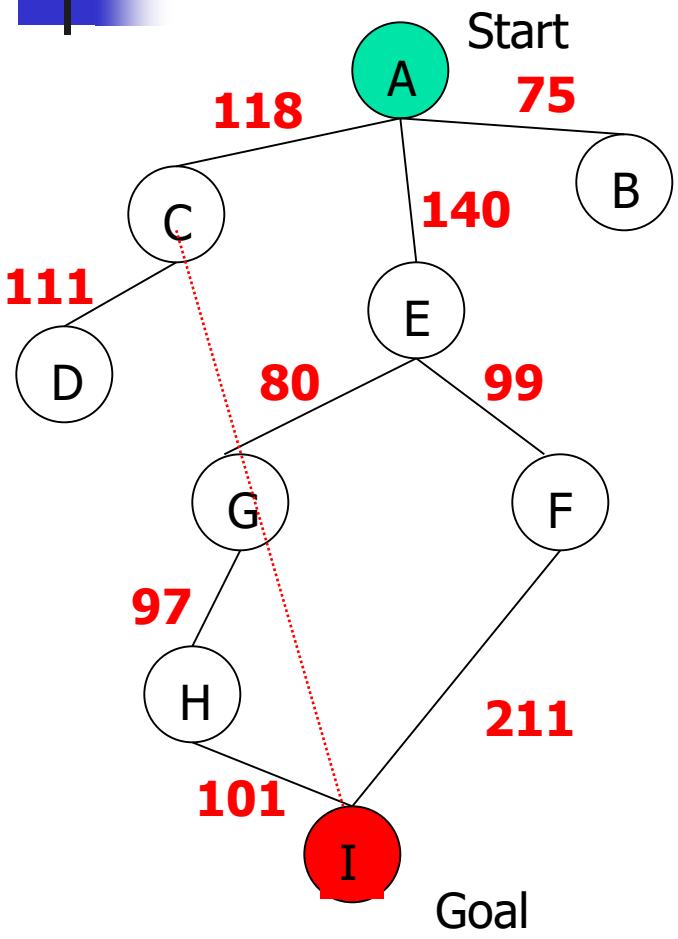
Greedy Search



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**

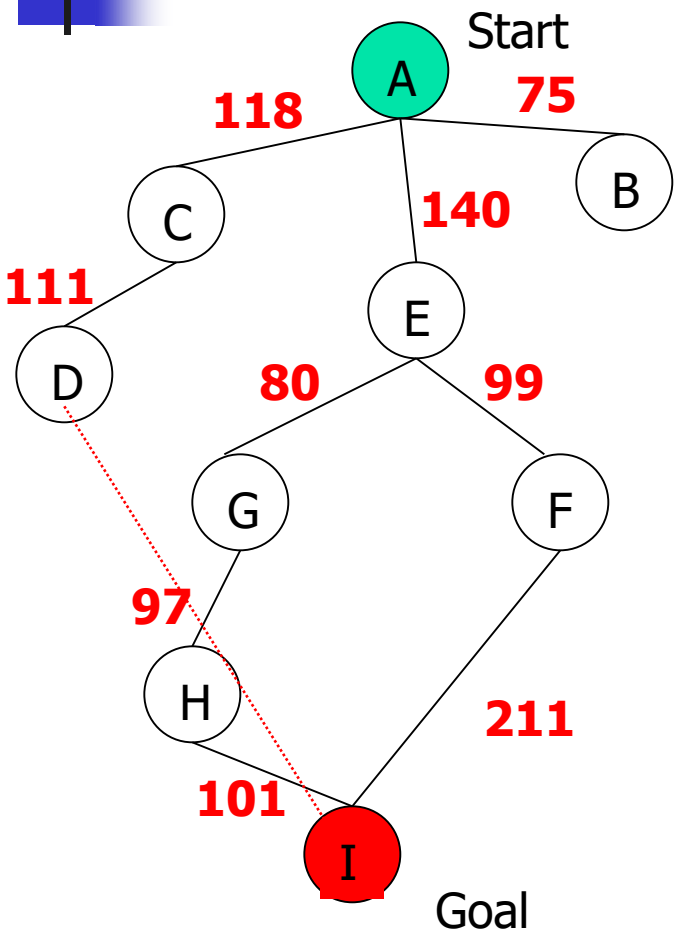
Greedy Search



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**

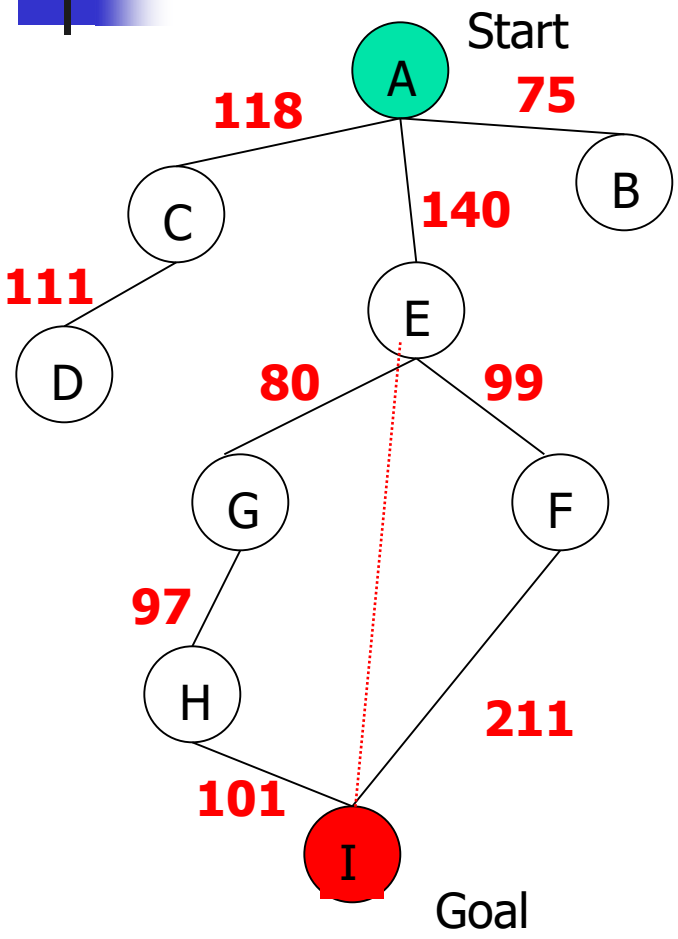
Greedy Search



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**

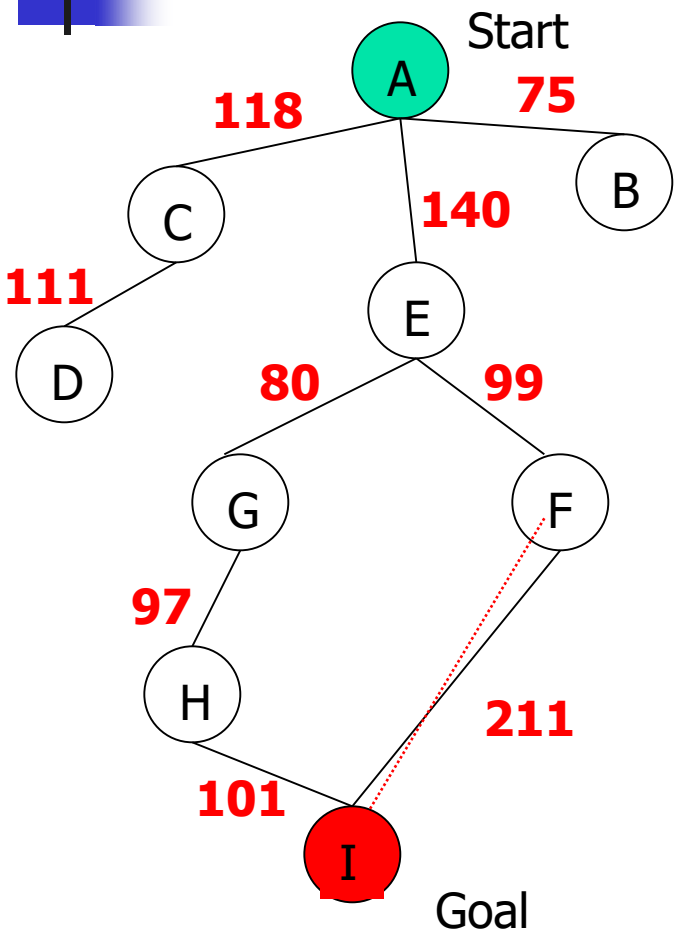
Greedy Search



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**

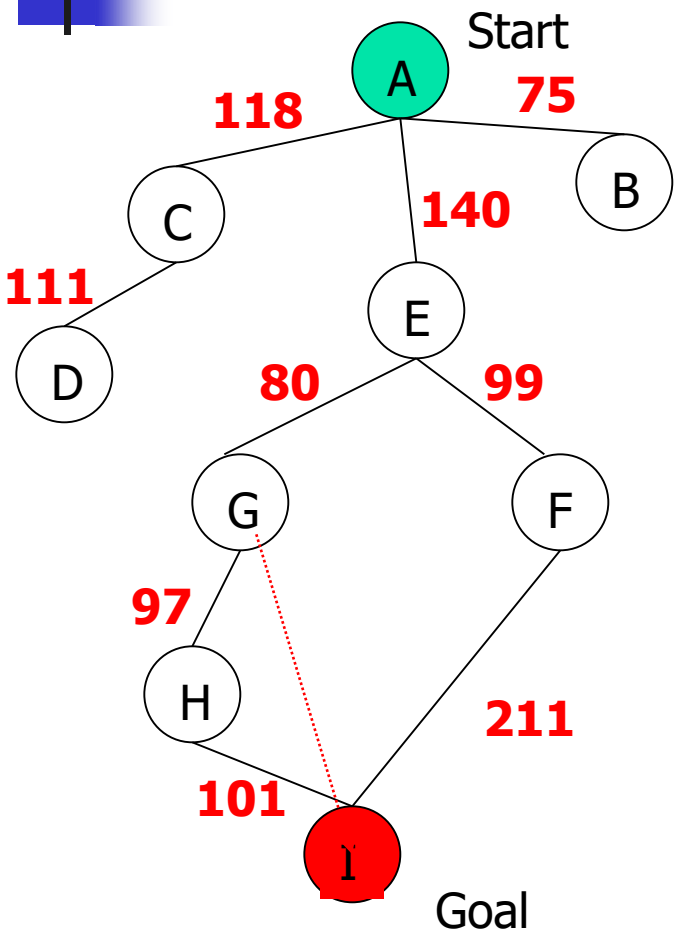
Greedy Search



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**

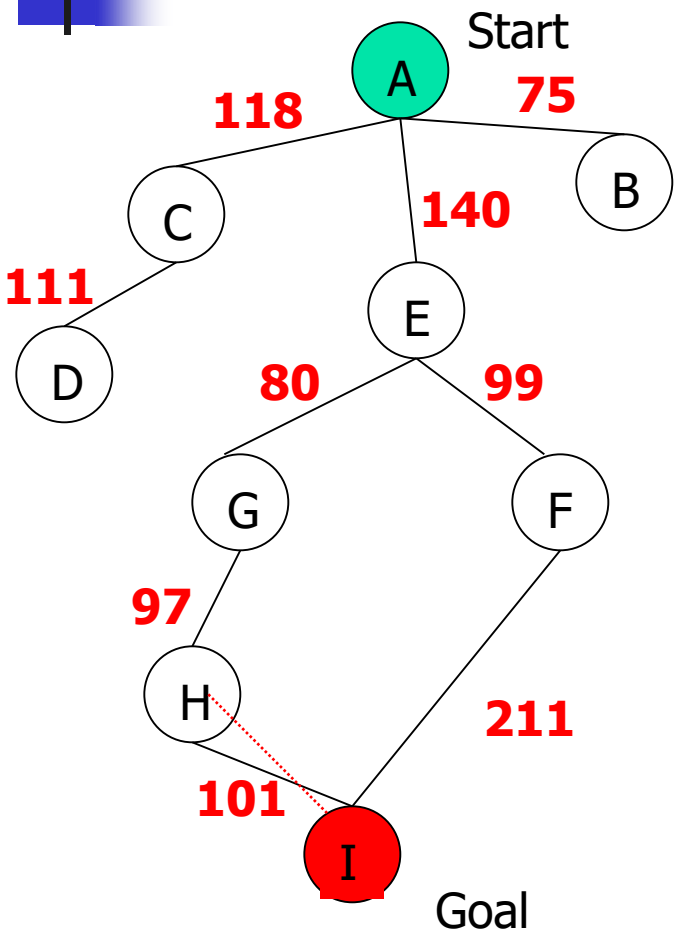
Greedy Search



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**

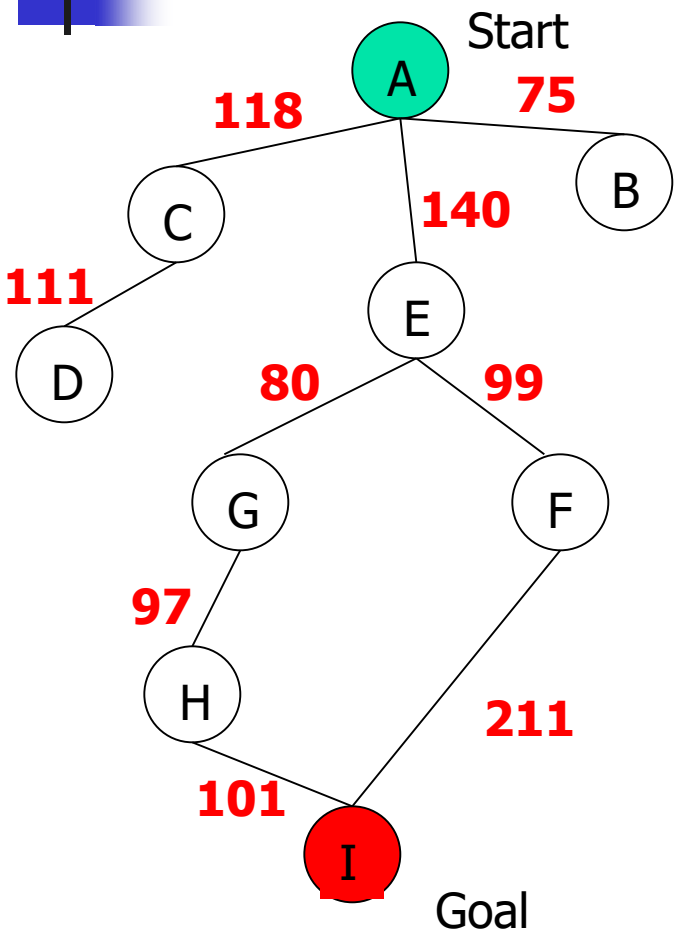
Greedy Search



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

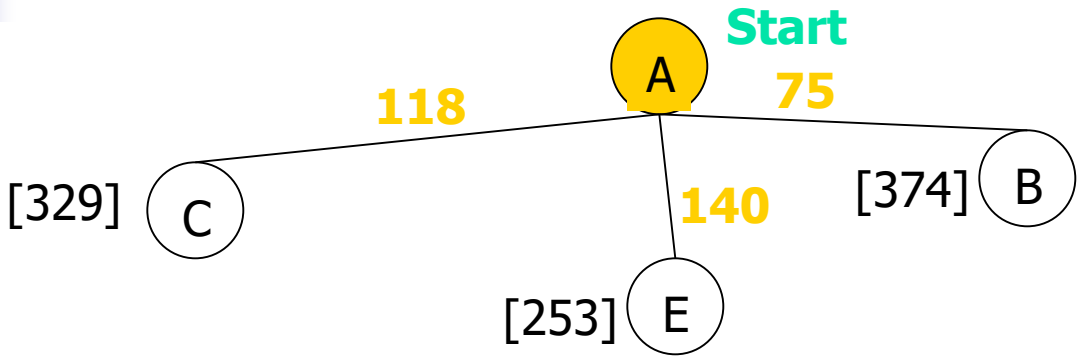


State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	**0**

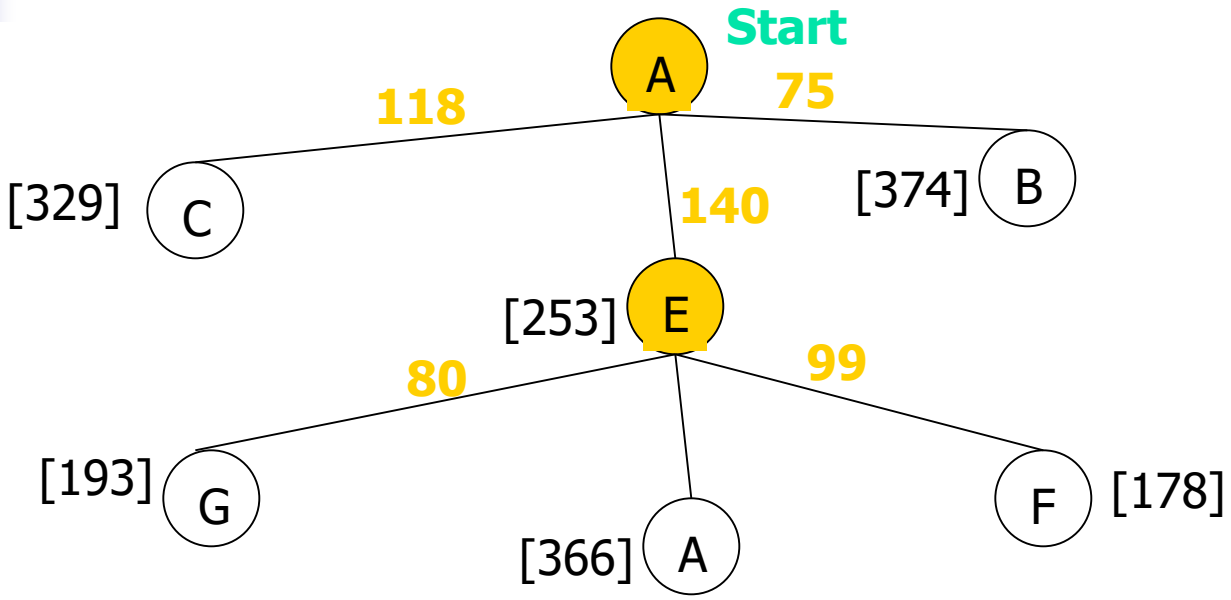
***f(n) = h (n)* = straight-line distance heuristic**

Greedy Search: Tree Search

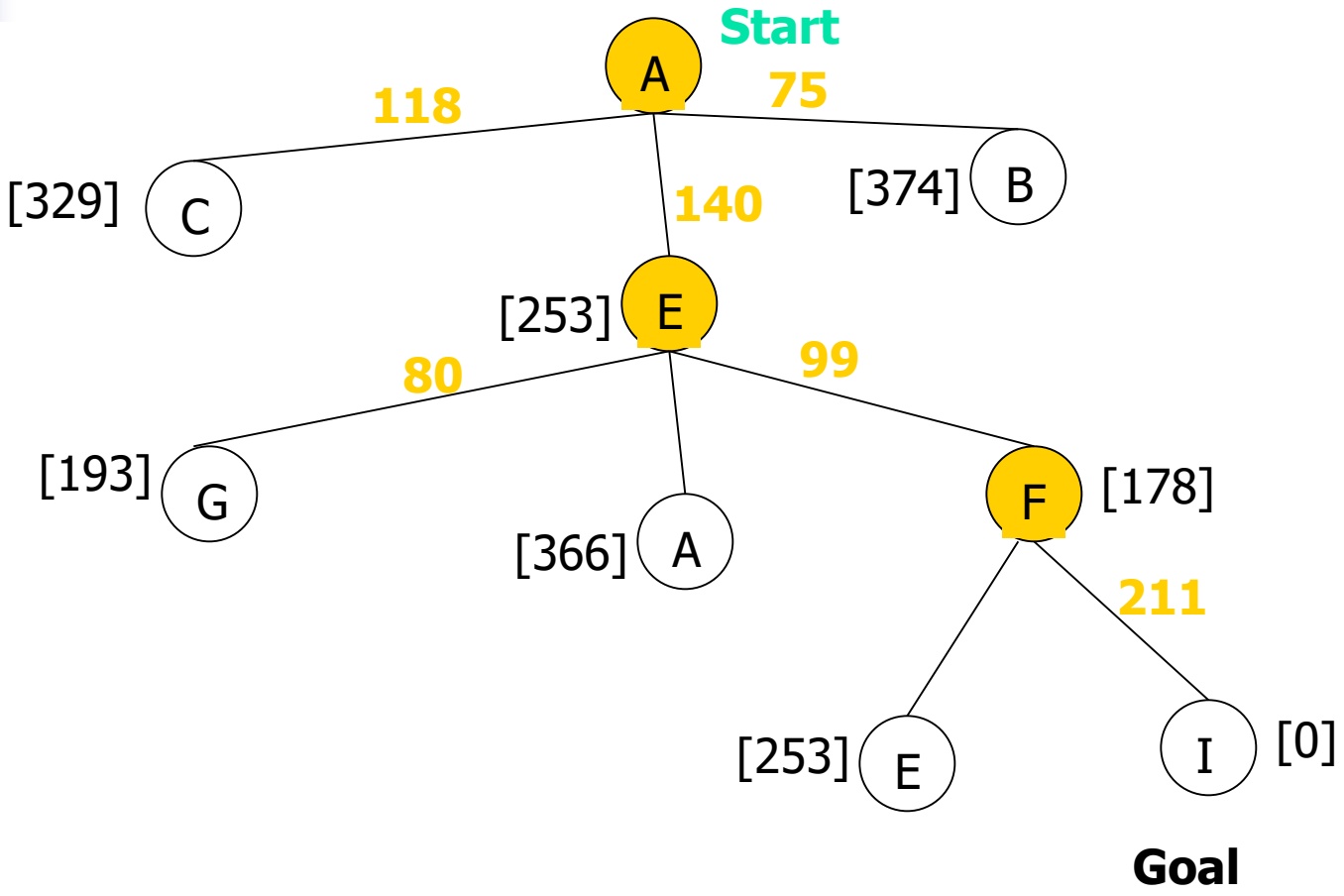
Greedy Search: Tree Search



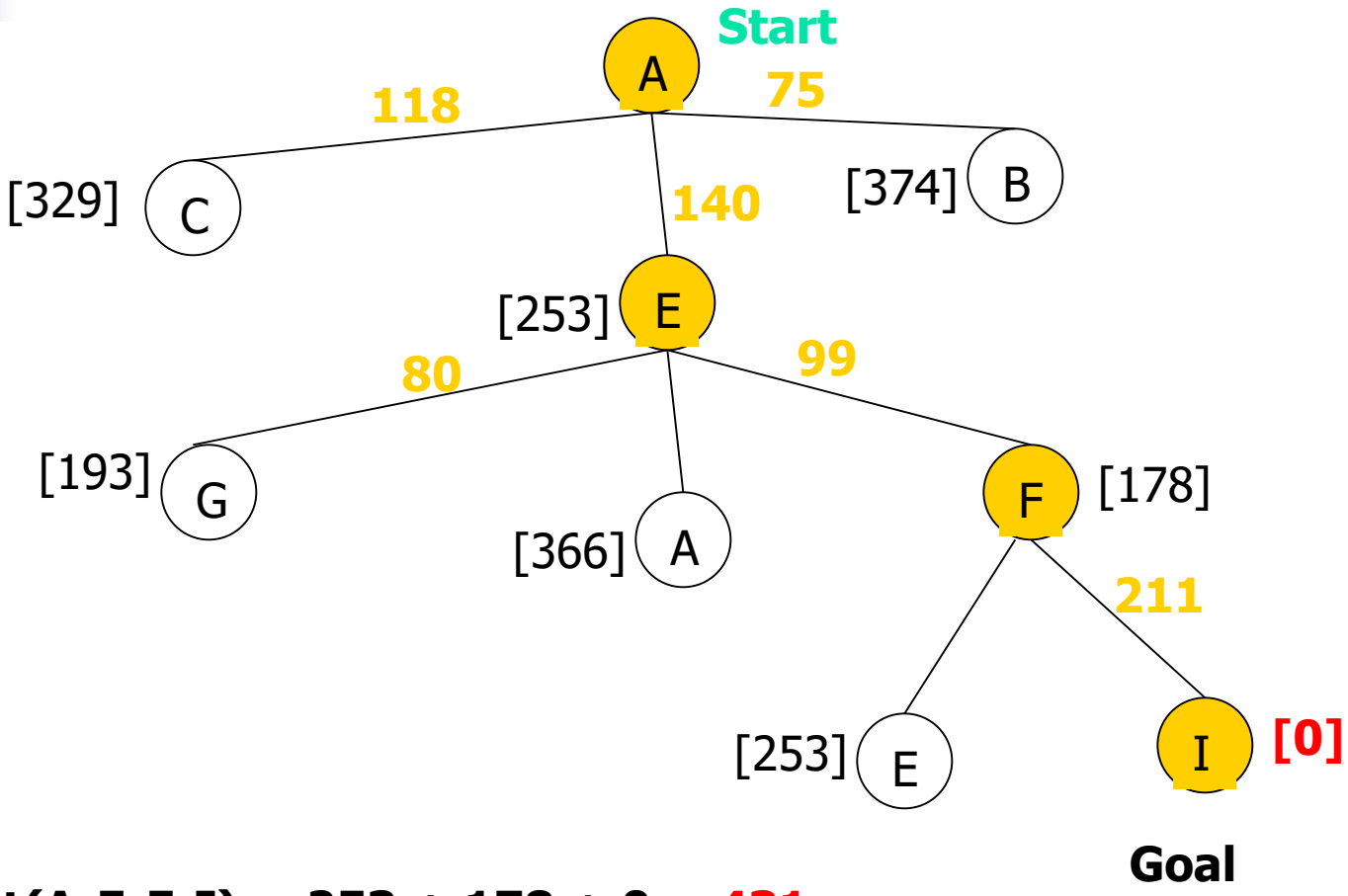
Greedy Search: Tree Search



Greedy Search: Tree Search



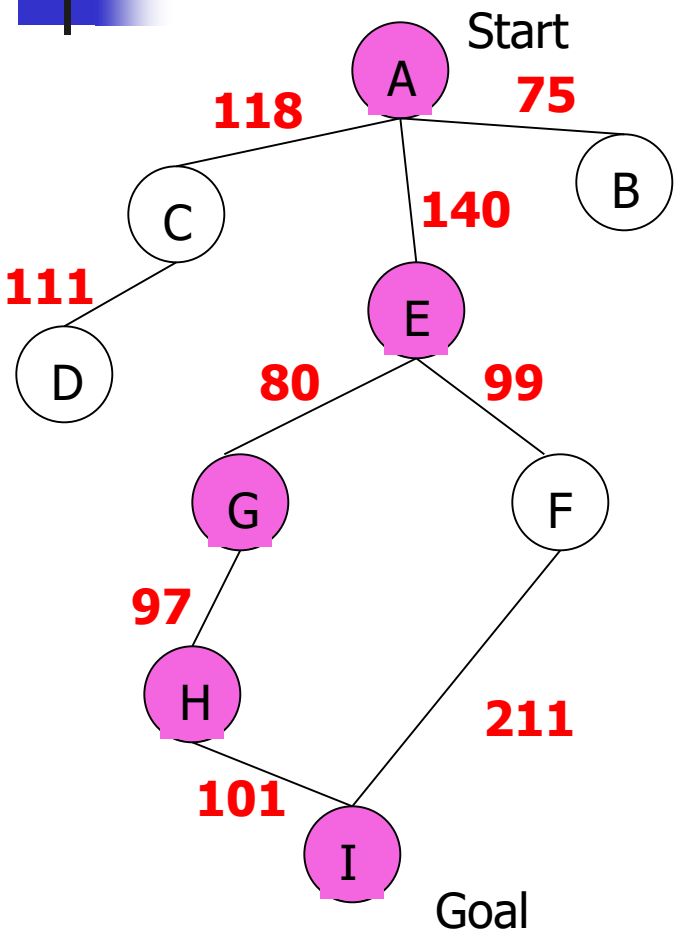
Greedy Search: Tree Search



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

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

Greedy Search: Optimal ?

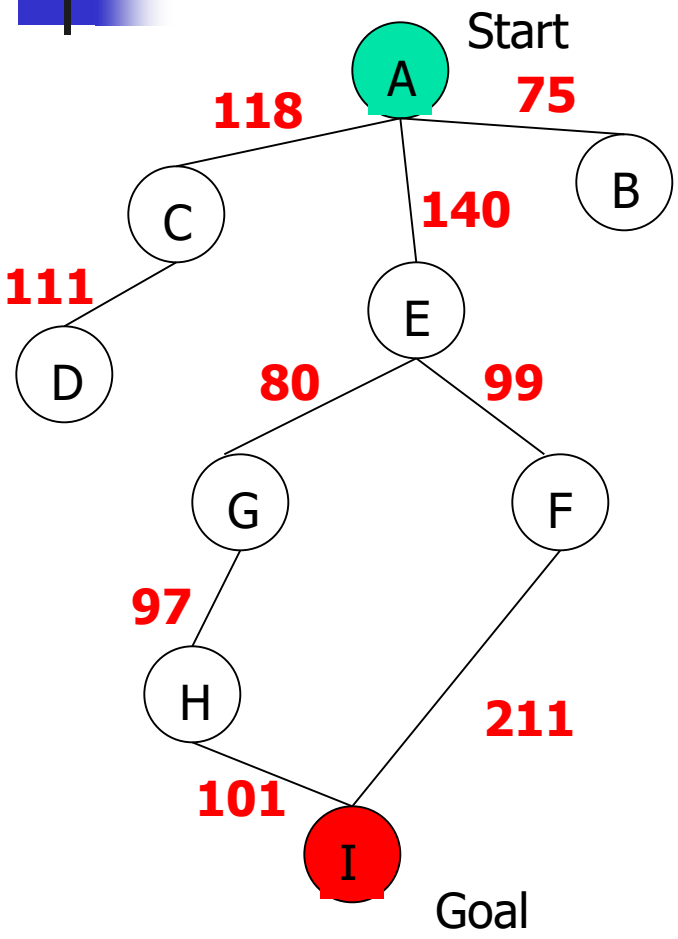


State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

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

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

Greedy Search: Complete ?

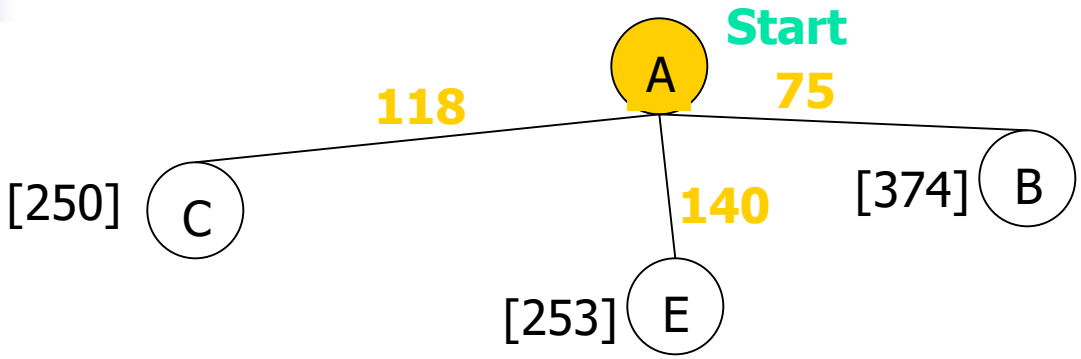


State	Heuristic: h(n)
A	366
B	374
** C	**250**
D	244
E	253
F	178
G	193
H	98
I	0

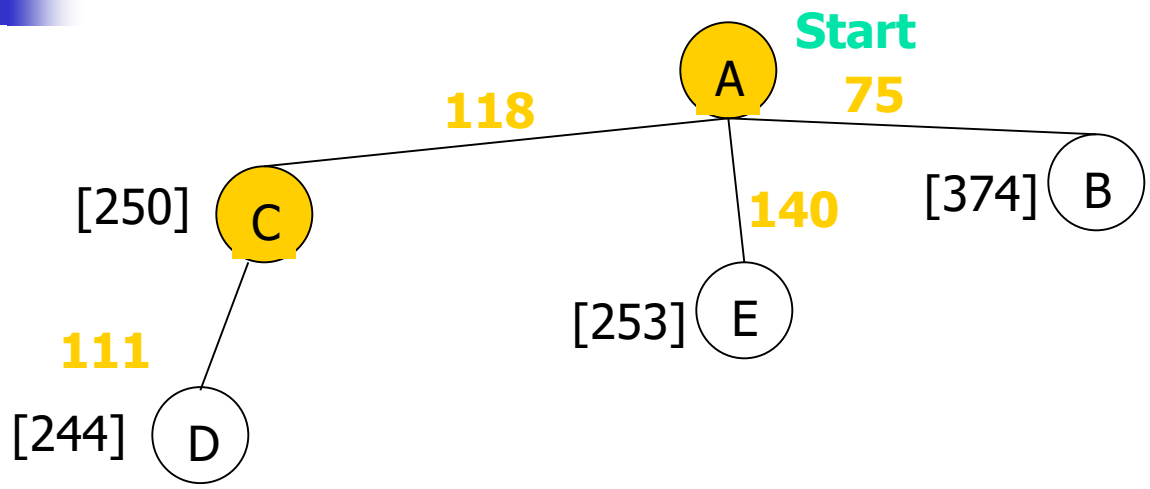
***f(n) = h (n)* = straight-line distance heuristic**

Greedy Search: Tree Search

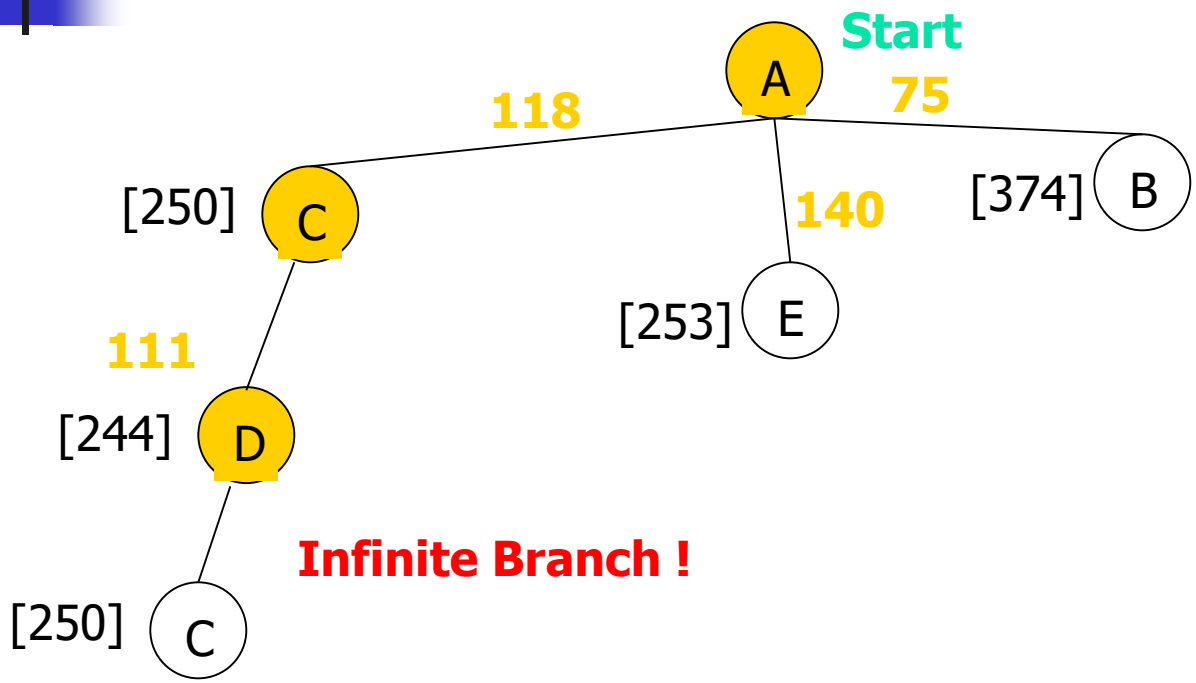
Greedy Search: Tree Search



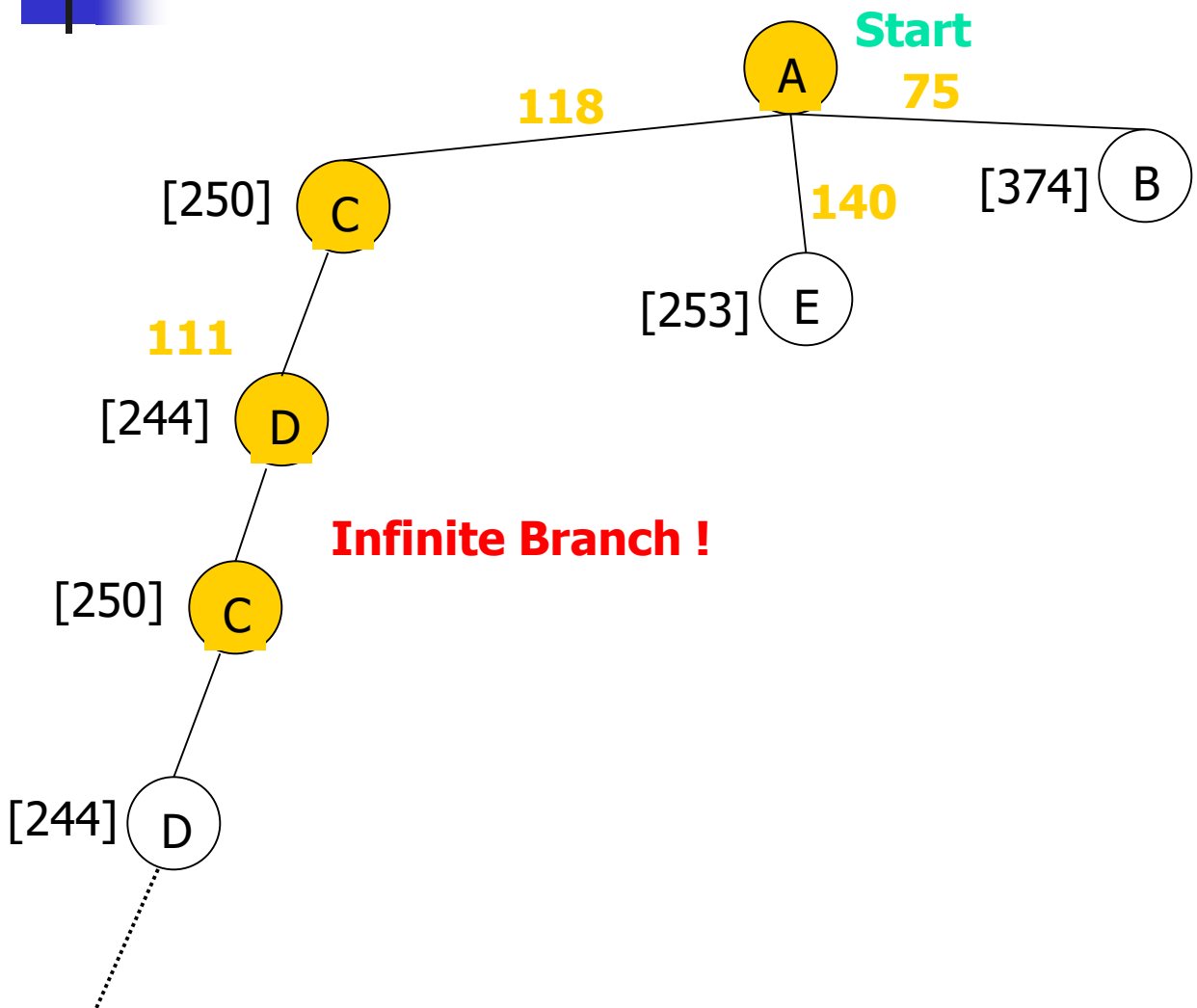
Greedy Search: Tree Search



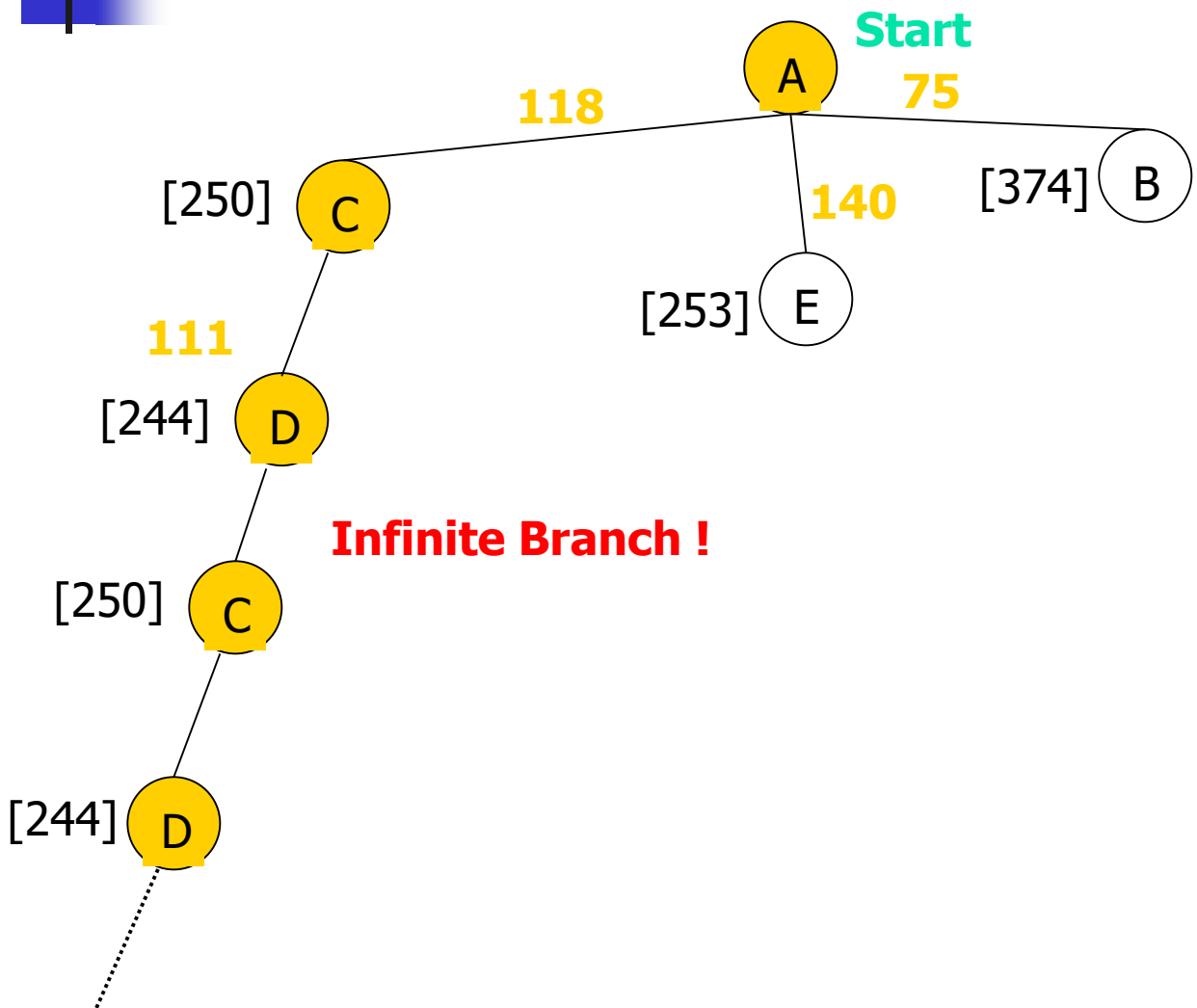
Greedy Search: Tree Search



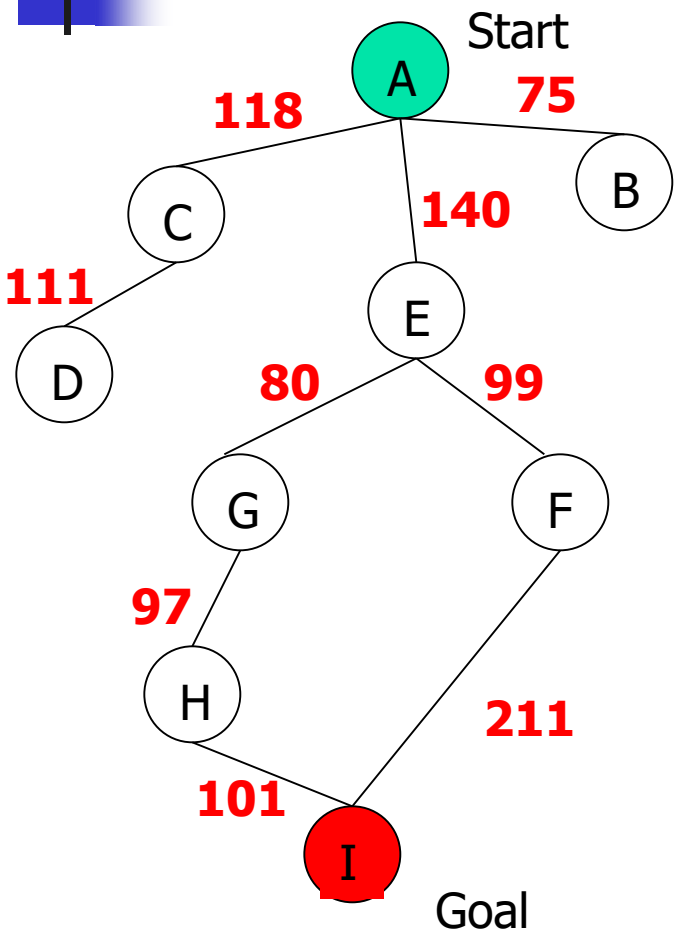
Greedy Search: Tree Search



Greedy Search: Tree Search

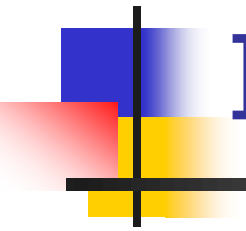


Greedy Search: Time and Space Complexity ?



- 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 $O(b^m)$

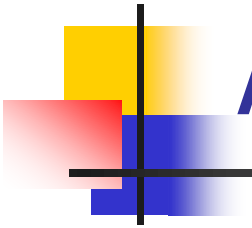
Where b is the branching factor and m the maximum path length



Informed Search Strategies

A* Search

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

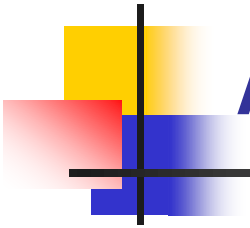


A* (A Star)

- 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.

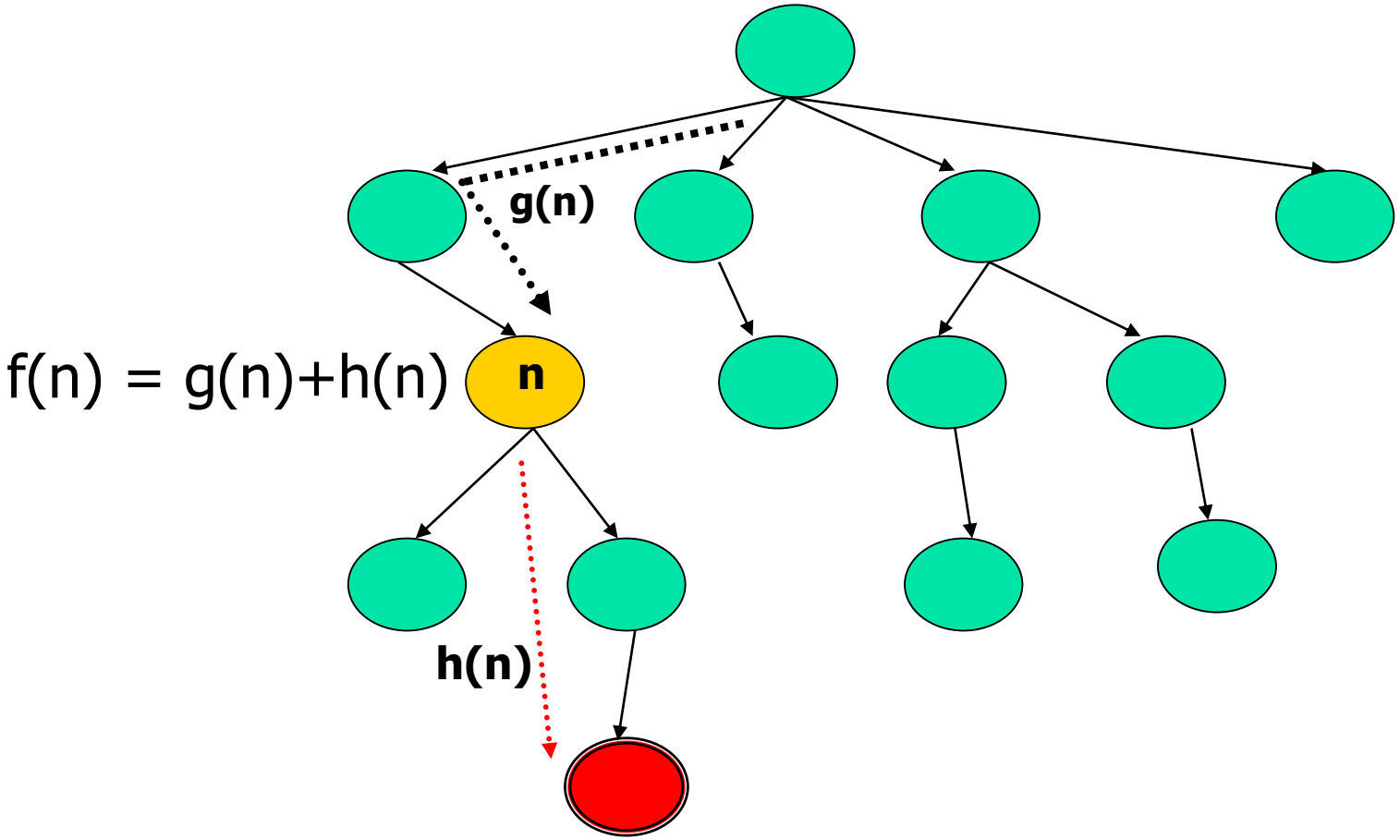
- **New Strategy**: Combine Greedy Search and UCS to get an efficient algorithm which is complete and optimal.



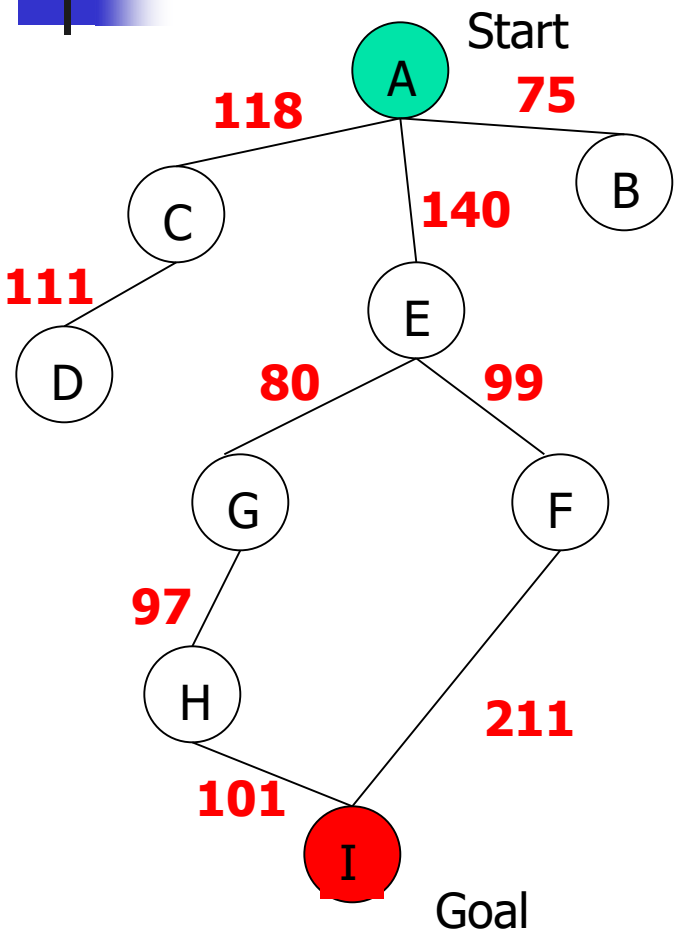
A* (A Star)

- A* uses a heuristic 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



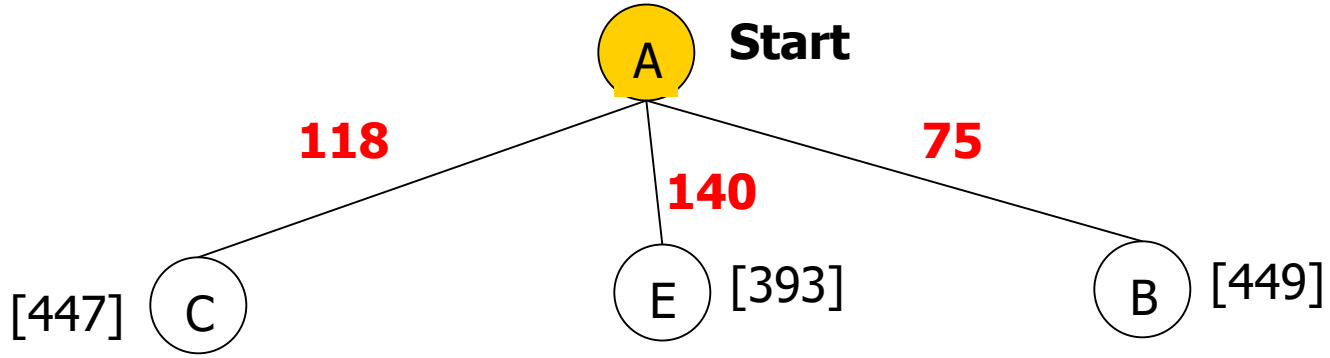
State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

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

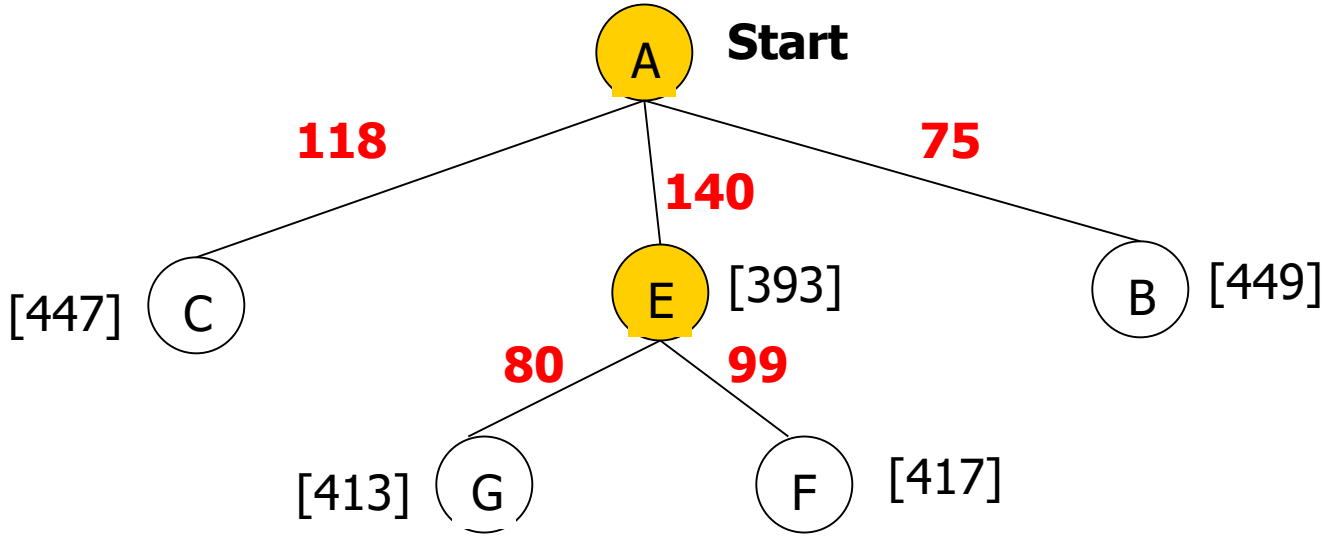
g(n): is the exact cost to reach node *n* from the initial state.

A* Search: Tree Search

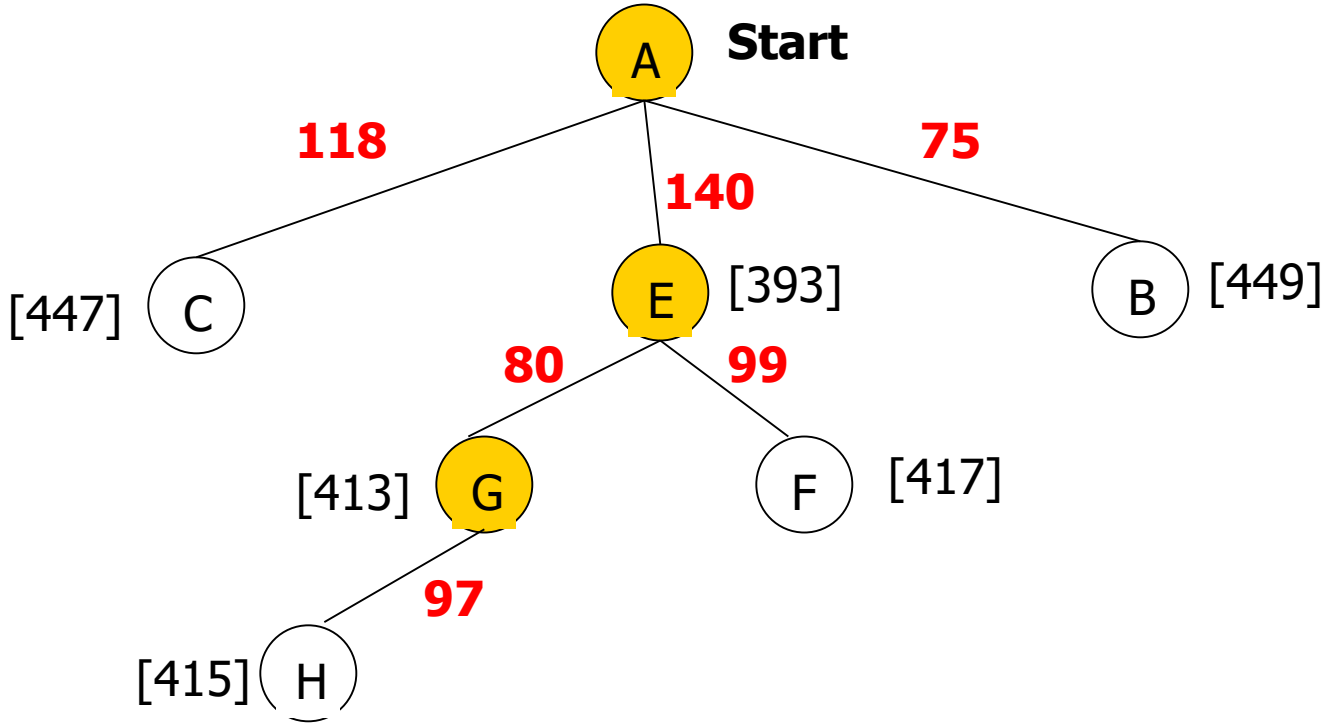
A* Search: Tree Search



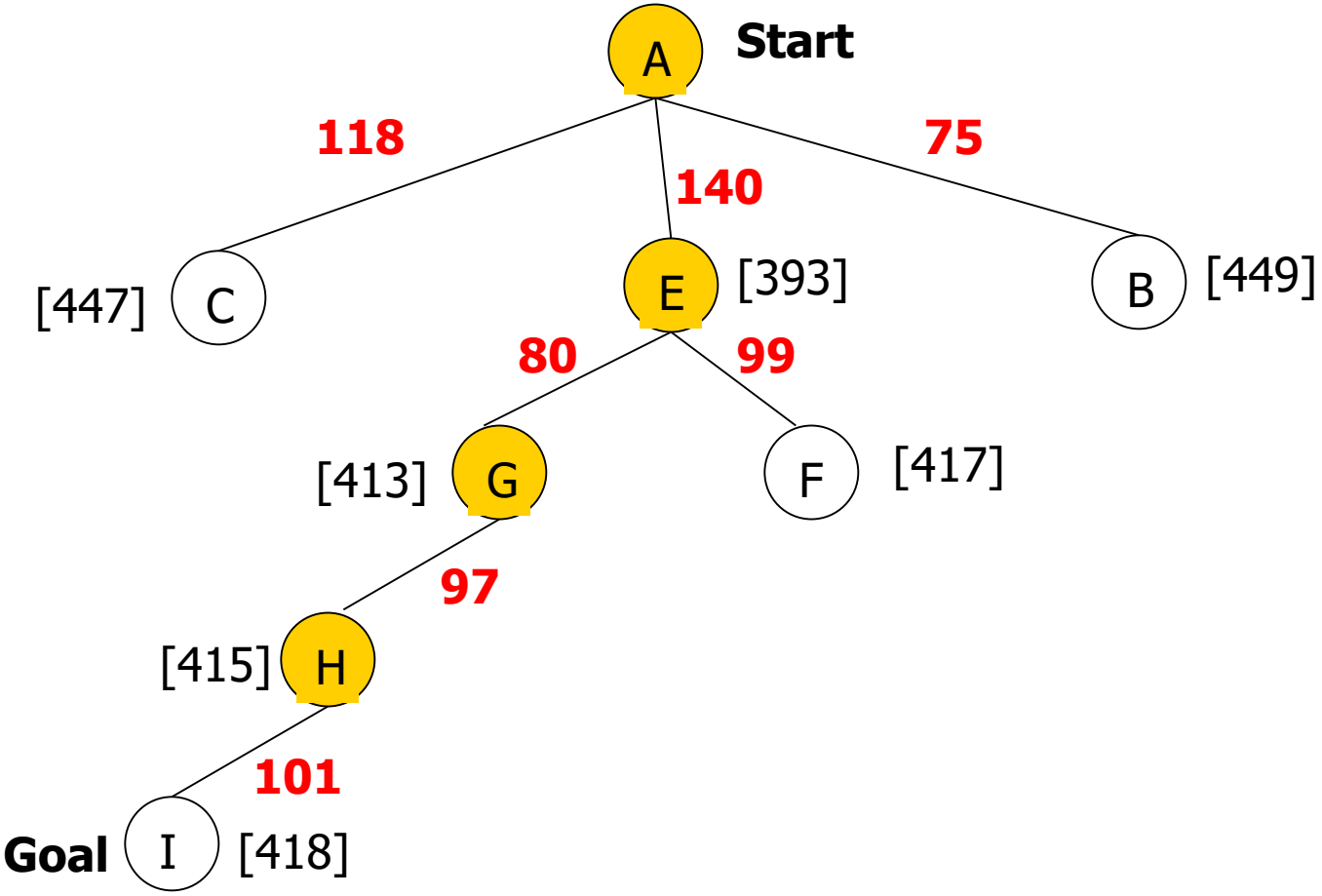
A* Search: Tree Search



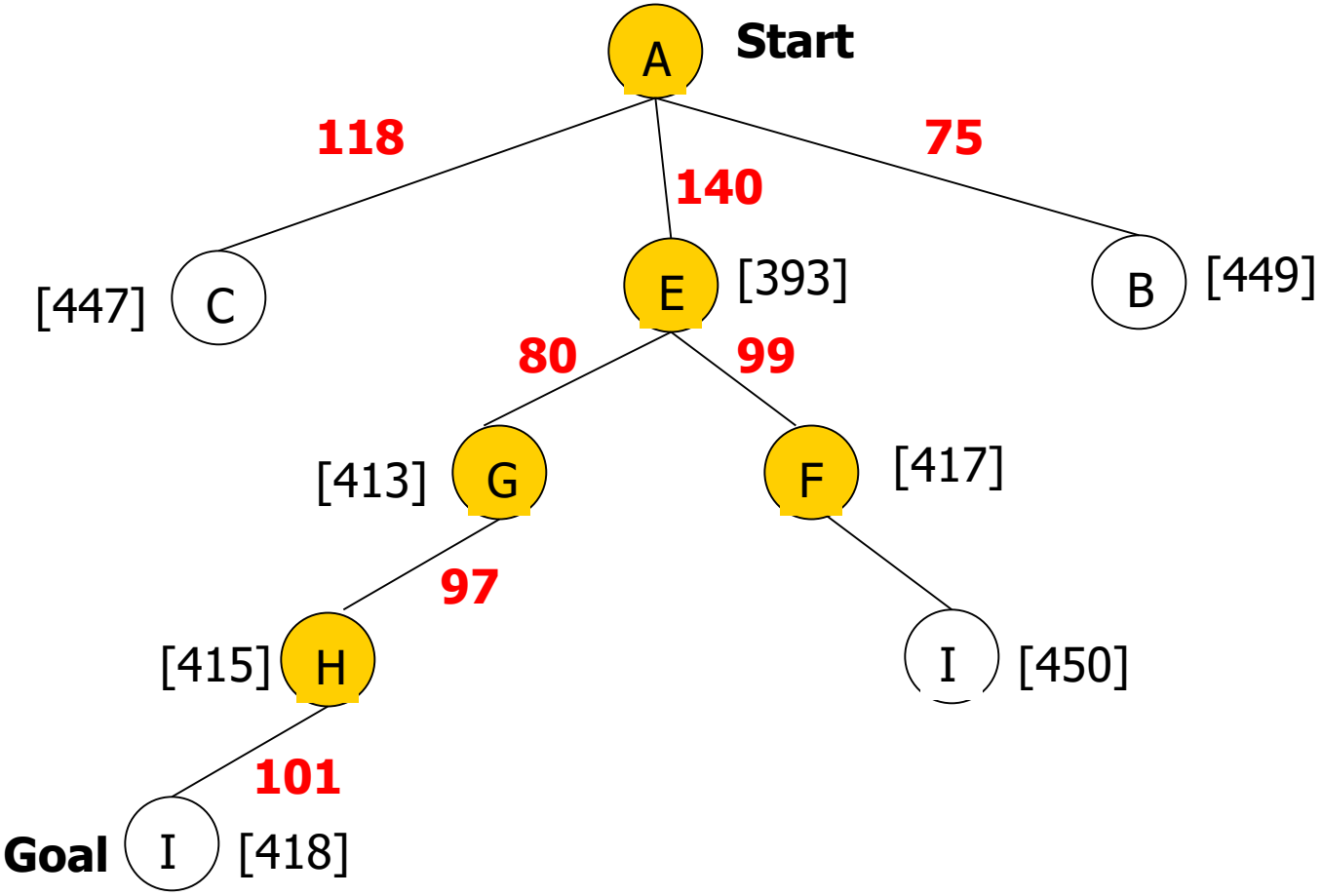
A* Search: Tree Search



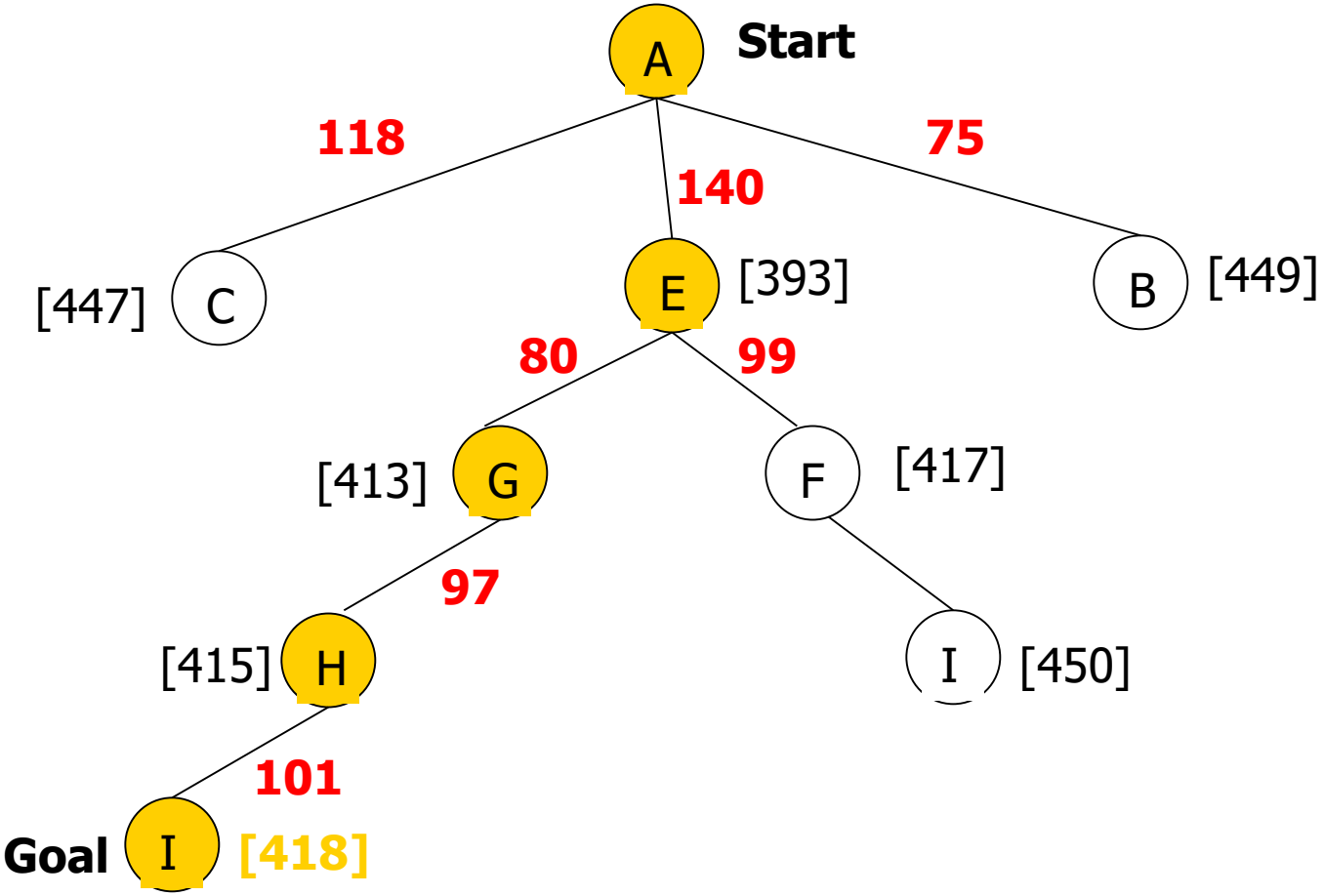
A* Search: Tree Search



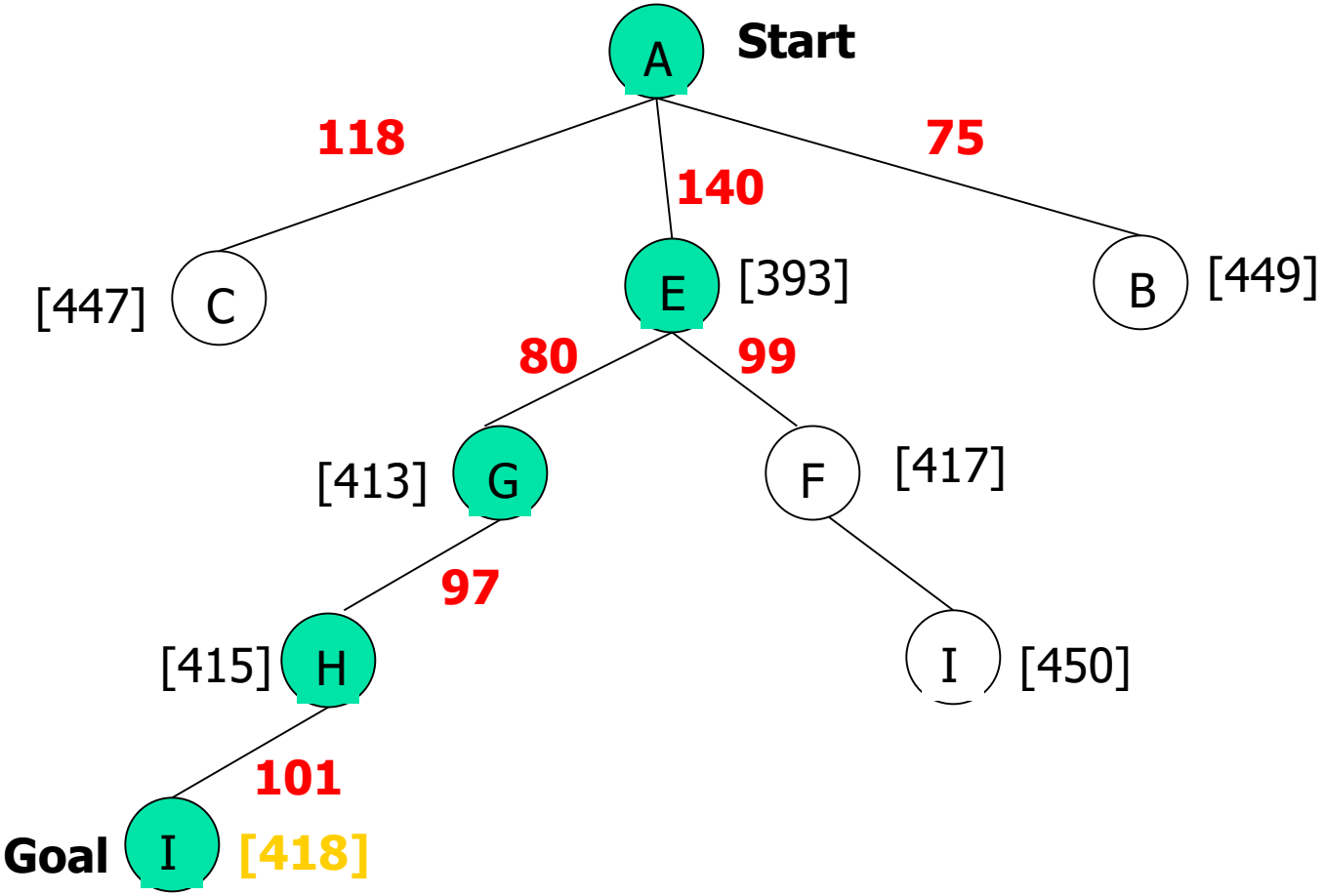
A* Search: Tree Search

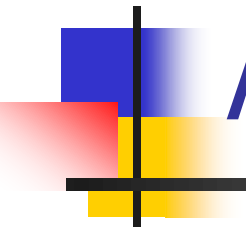


A* Search: Tree Search



A* Search: Tree Search

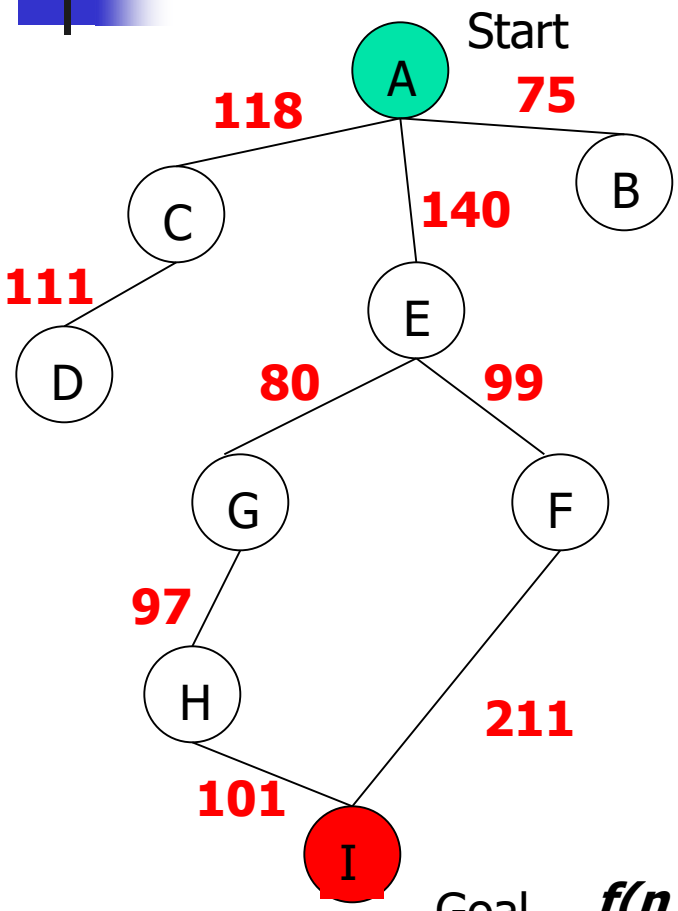




A* with f() not Admissible

h() overestimates the cost to reach the goal state

A* Search: *h* not admissible !



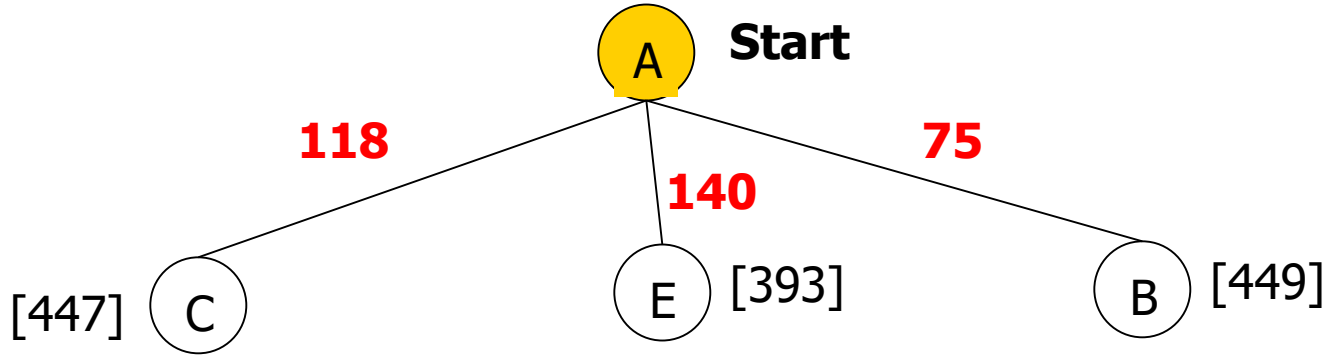
State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	**138**
I	0

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

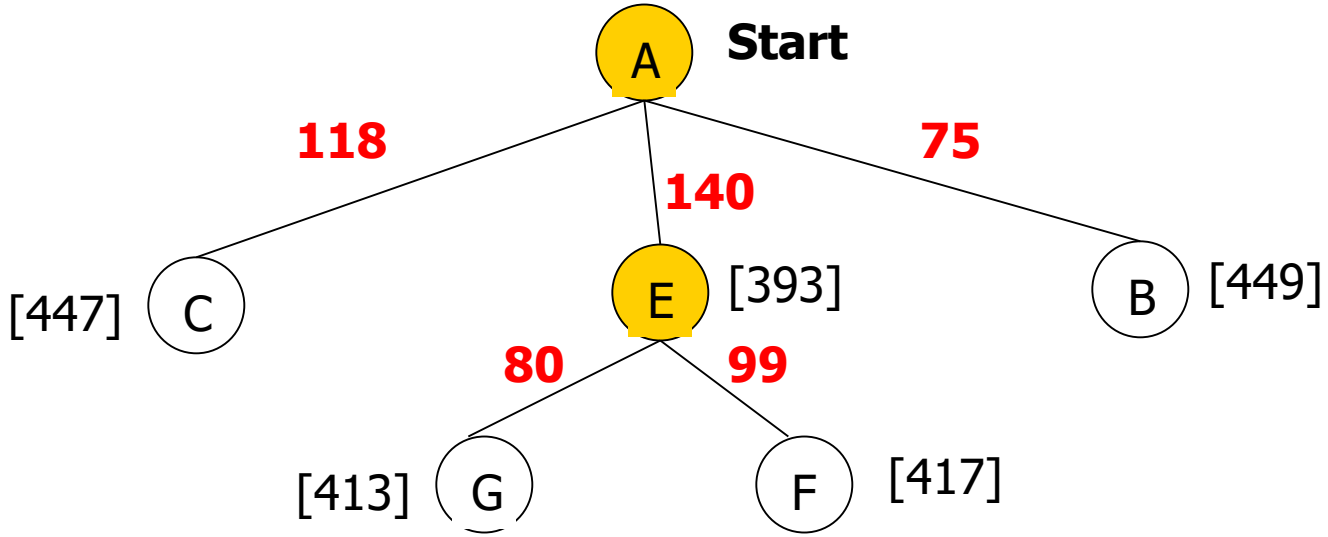
g(n): is the exact cost to reach node *n* from the initial state.

A* Search: Tree Search

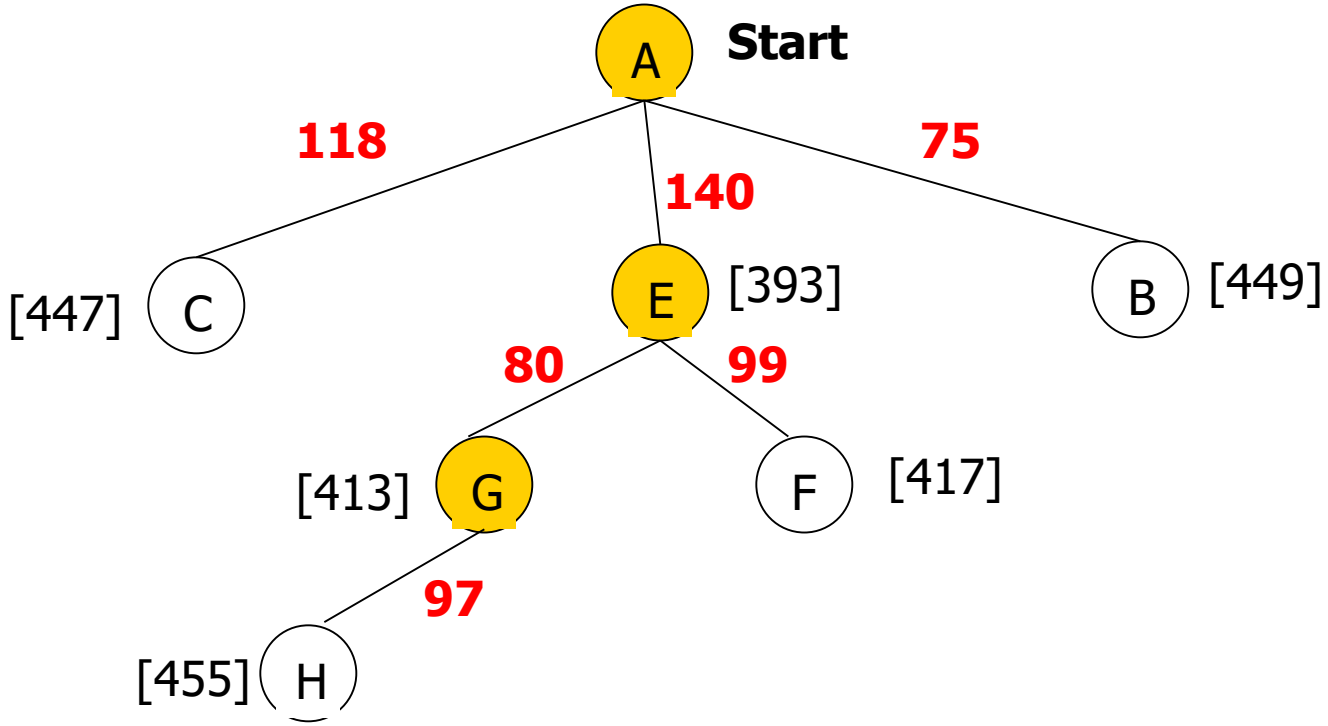
A* Search: Tree Search



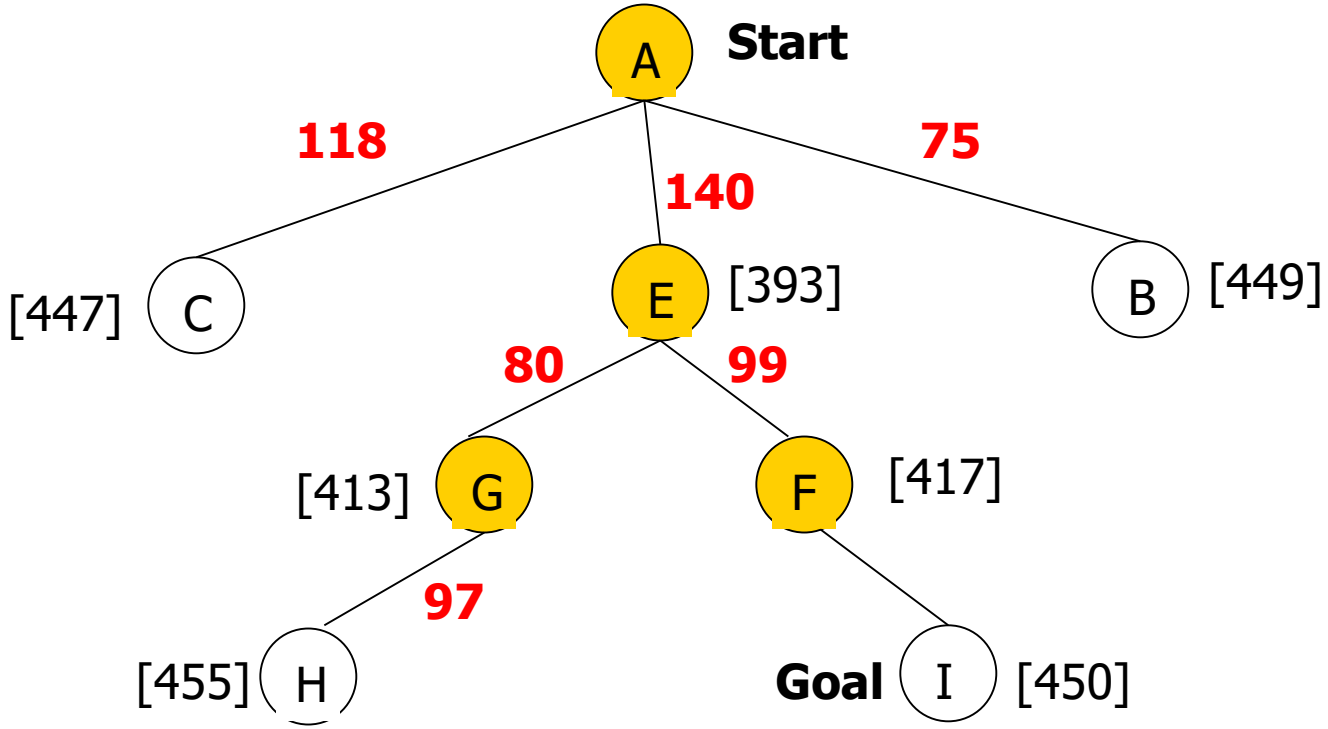
A* Search: Tree Search



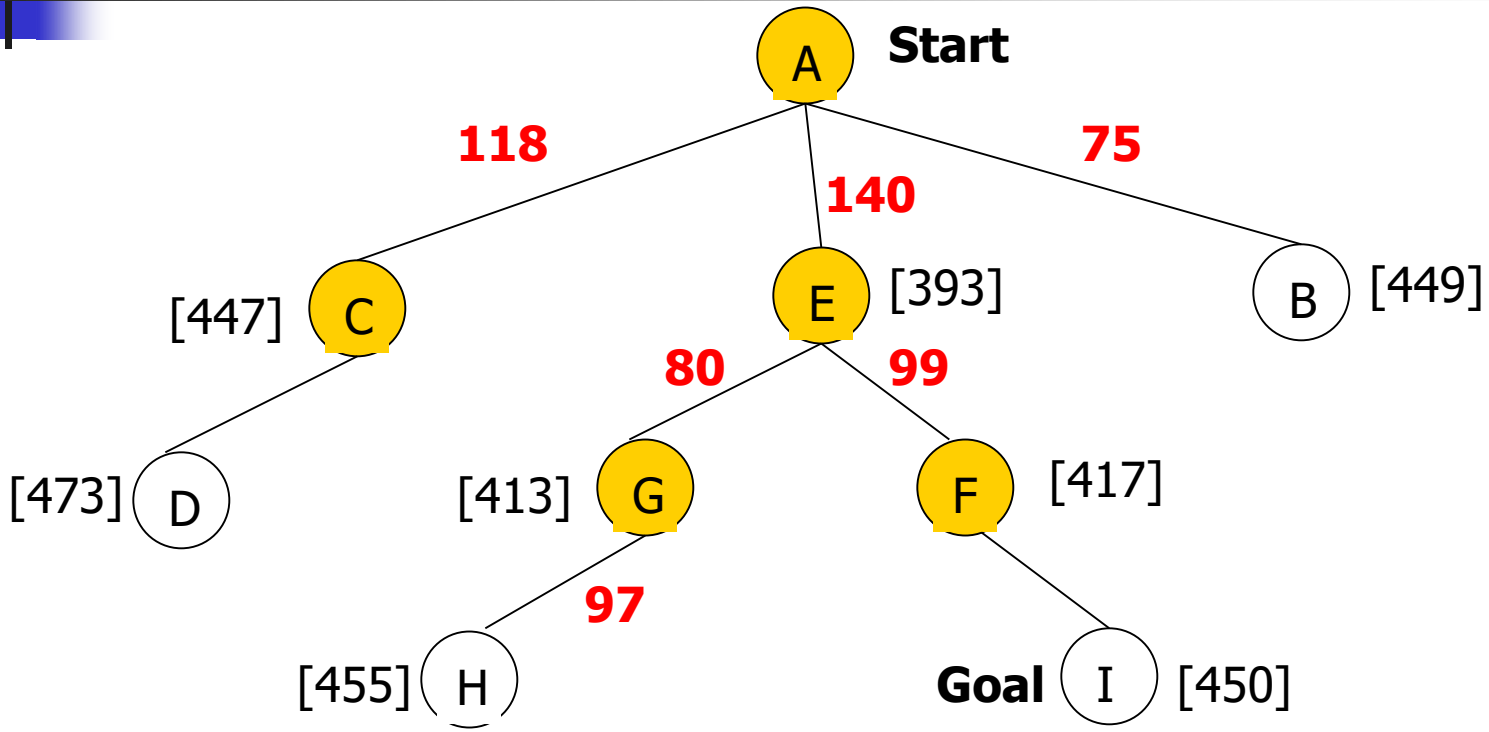
A* Search: Tree Search



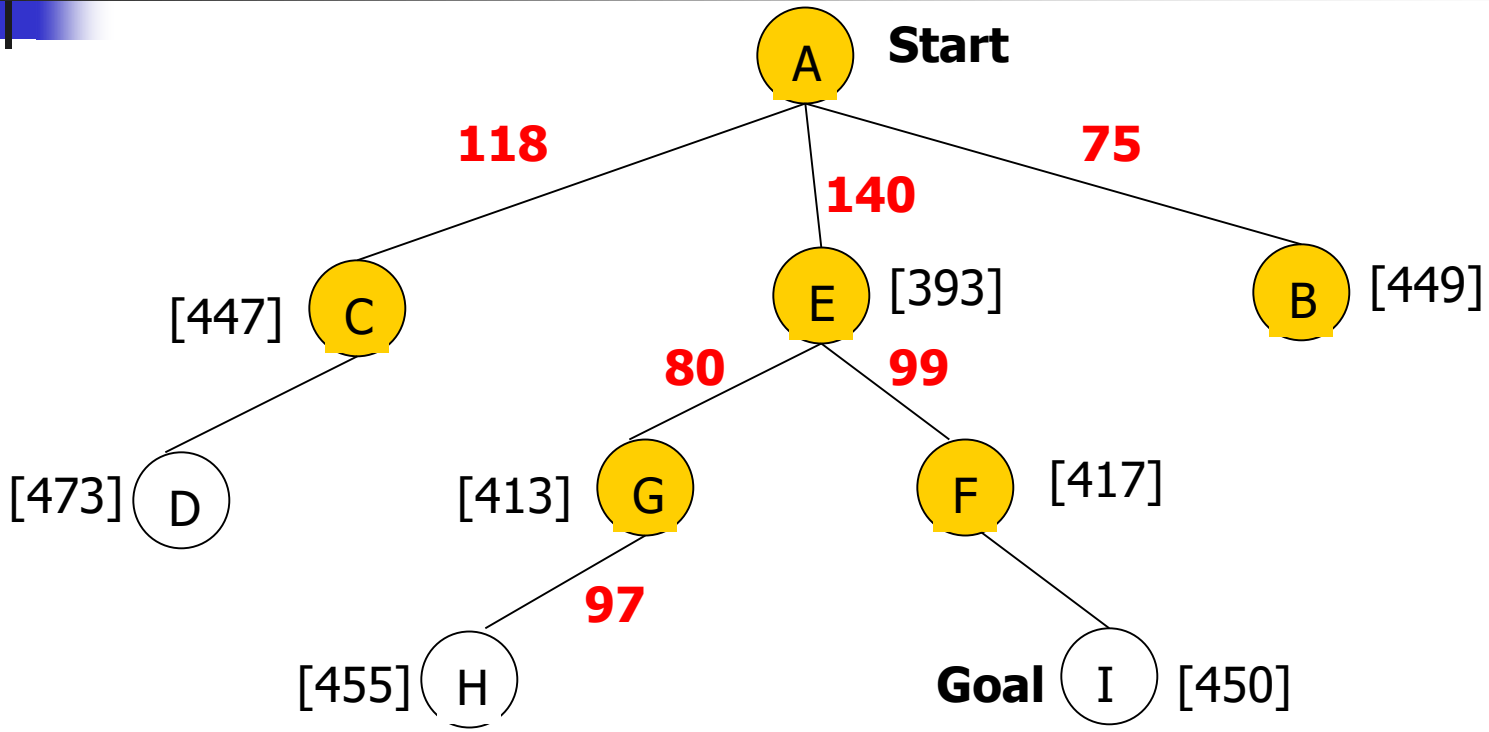
A* Search: Tree Search



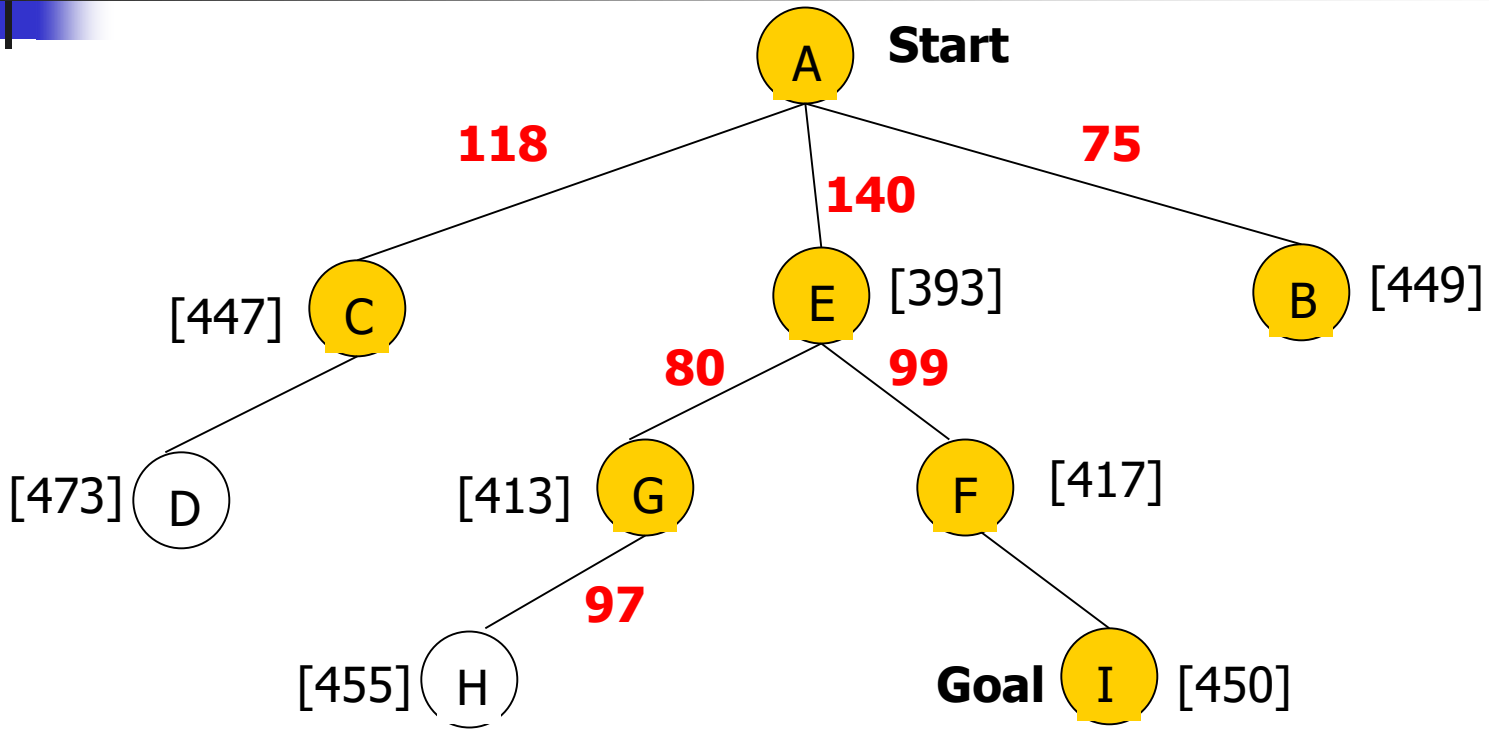
A* Search: Tree Search



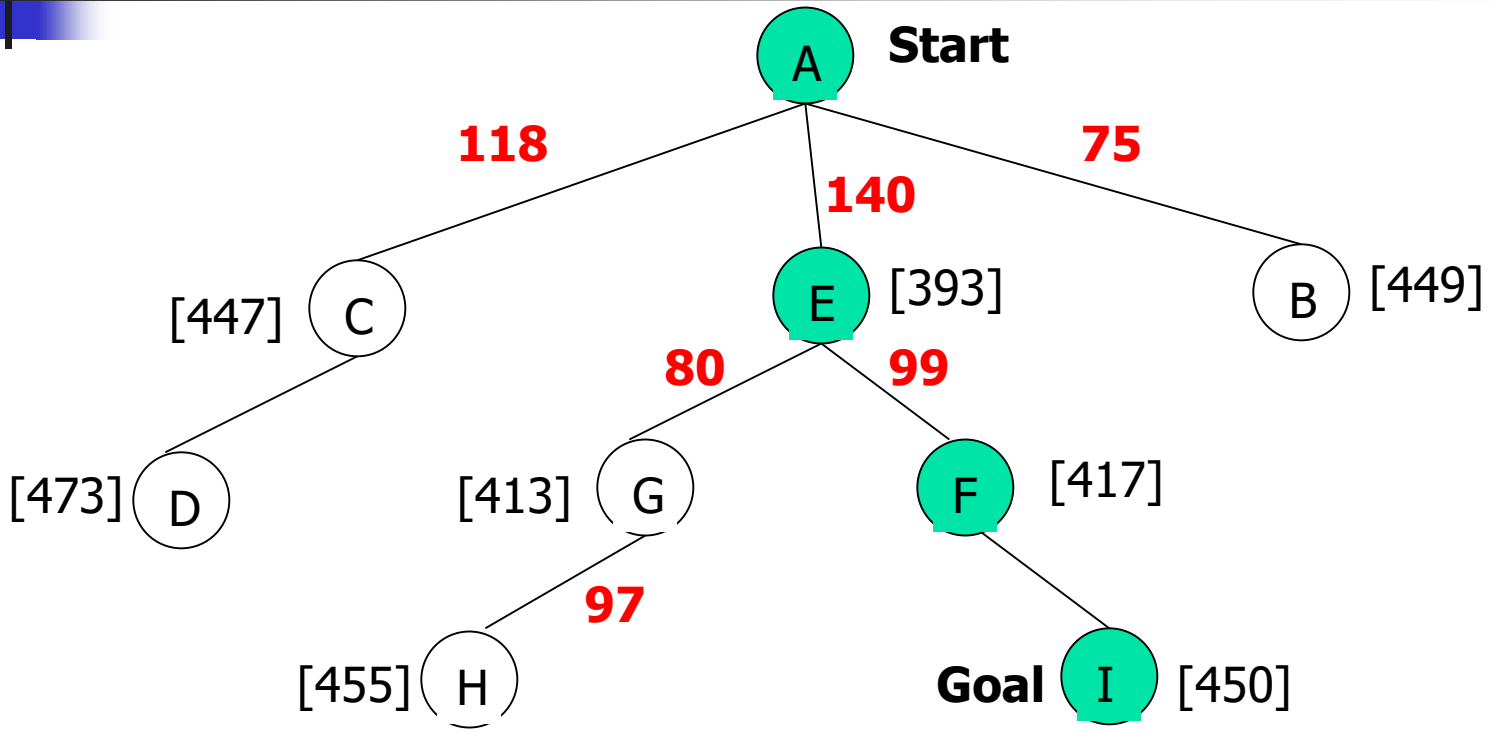
A* Search: Tree Search



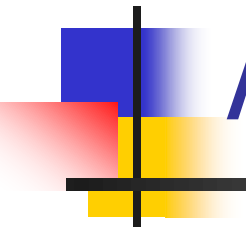
A* Search: Tree Search



A* Search: Tree Search



A* not optimal !!!



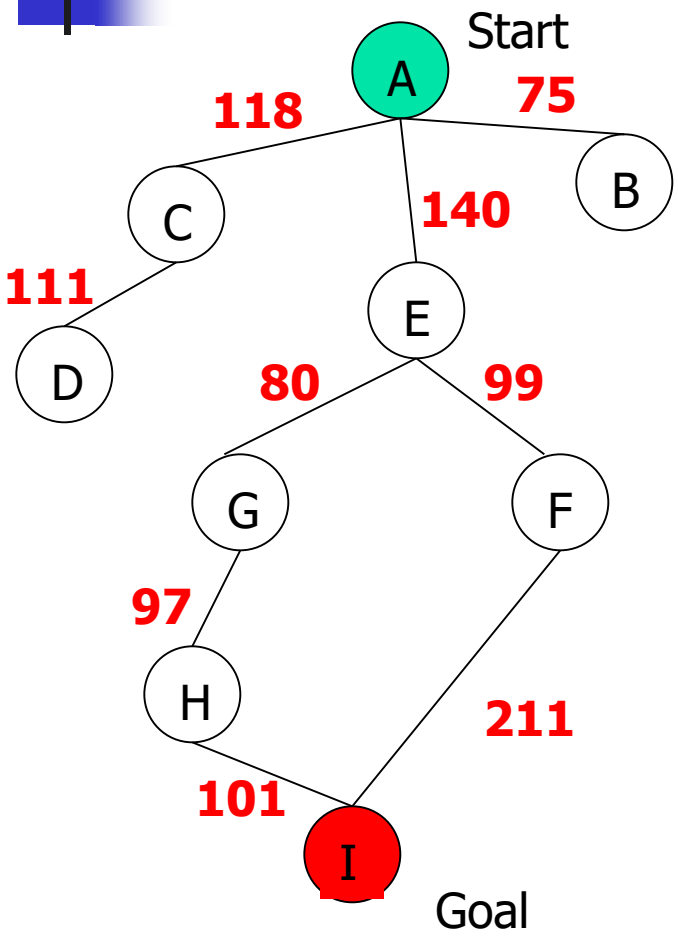
A* Algorithm

A* with systematic checking for repeated states …

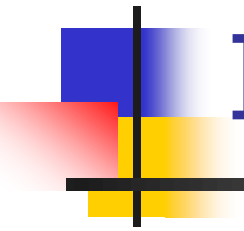
A* Algorithm

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) + 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

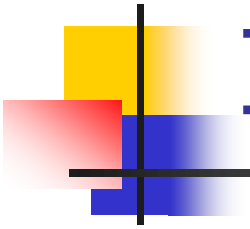


- A* is complete except if there is an infinity of nodes with f < f(G).
- A* is optimal if heuristic *h* is admissible.
- Time complexity depends on the quality of heuristic but is still exponential.
- For space complexity, A* keeps all nodes in memory. A* has worst case $O(b^d)$ space complexity, but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



Informed Search Strategies

Iterative Deepening A*



Iterative Deepening A*:IDA*

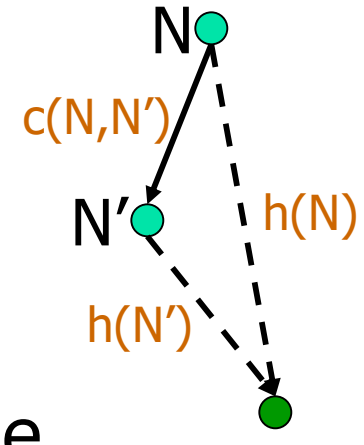
- Use f(N) = g(N) + h(N) with admissible and consistent h
- Each iteration is depth-first with cutoff on the value of f of expanded nodes

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)



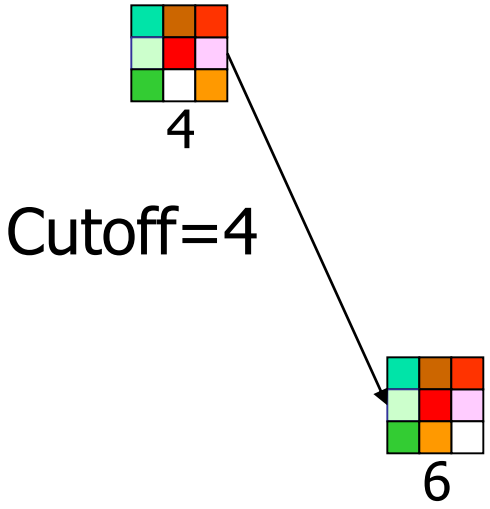
- A consistent heuristic is admissible.

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.

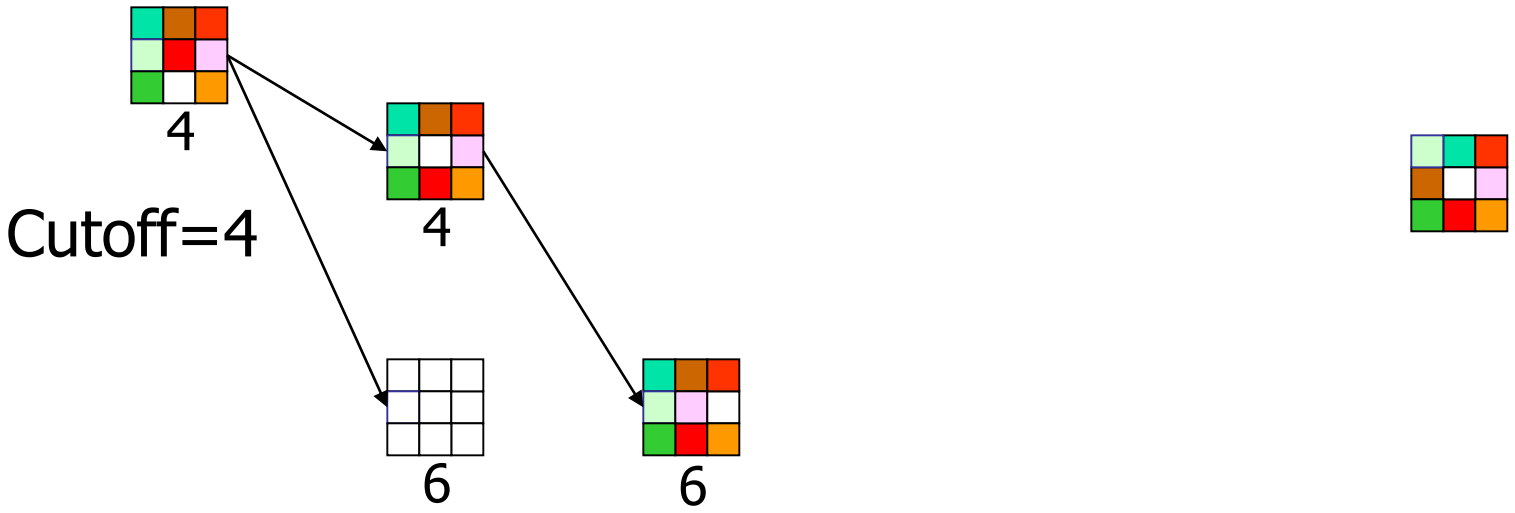
8-Puzzle

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



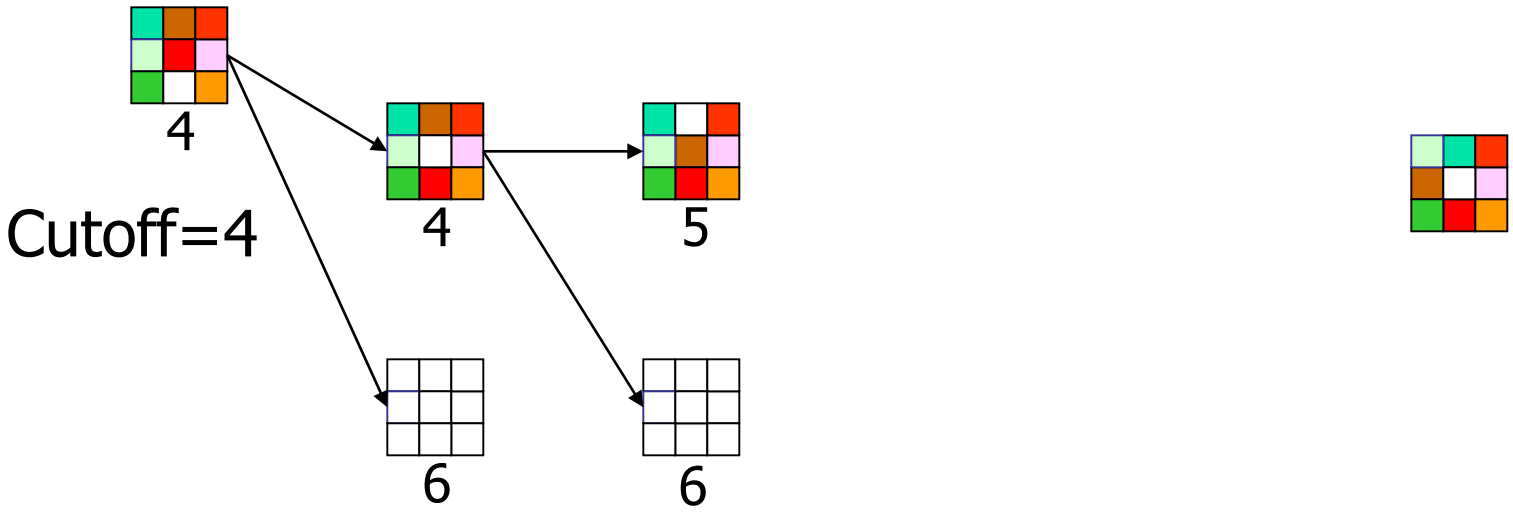
8-Puzzle

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



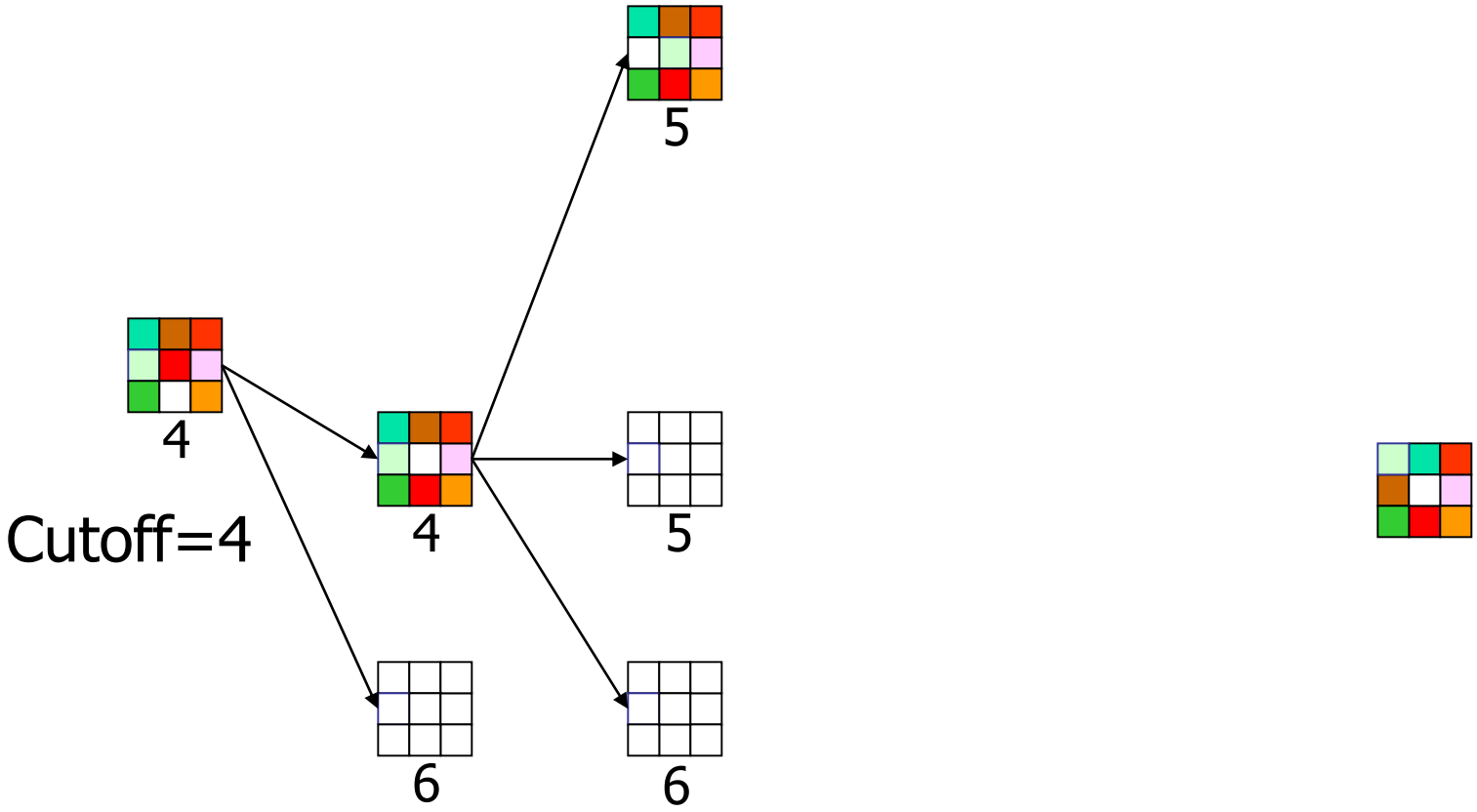
8-Puzzle

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



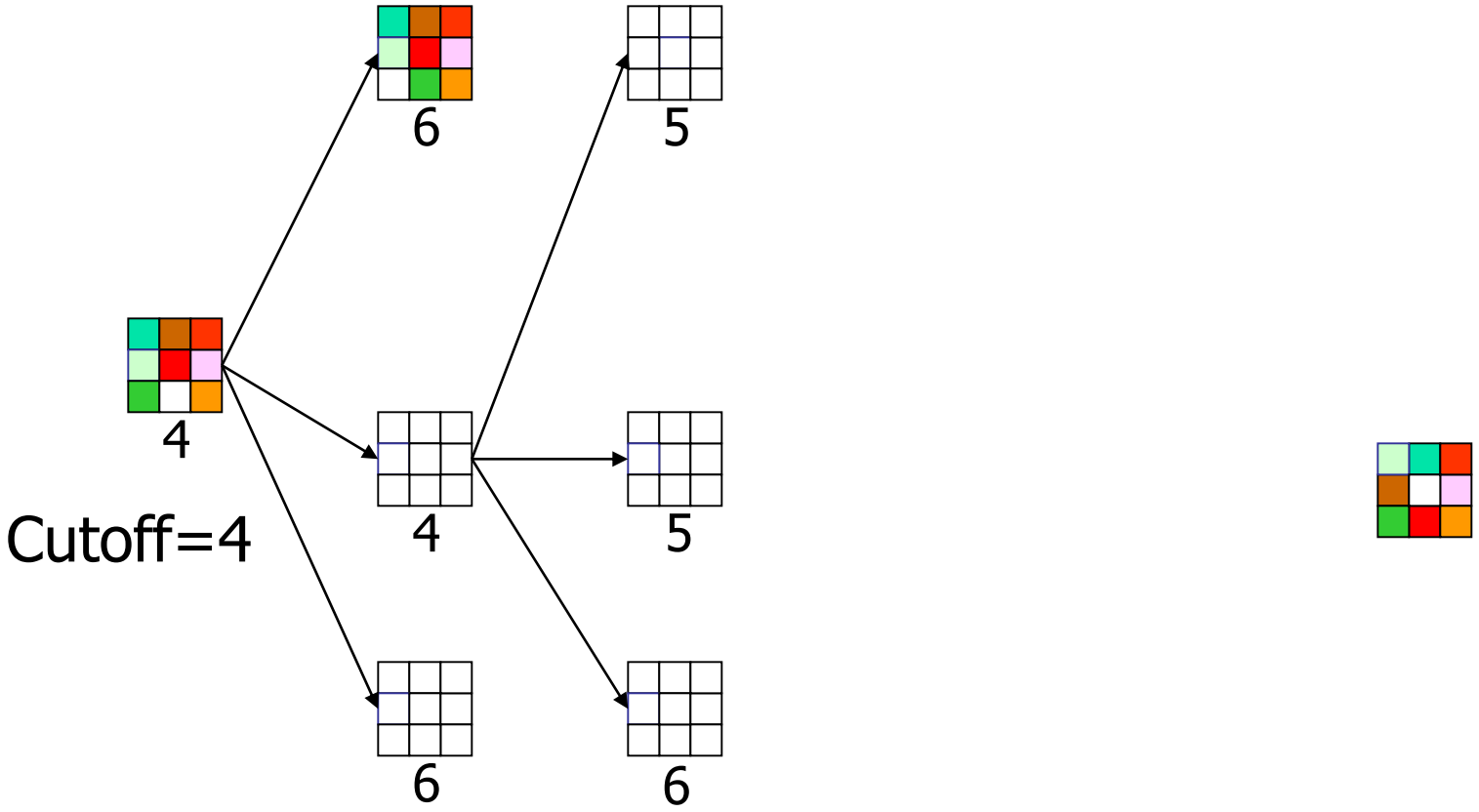
8-Puzzle

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



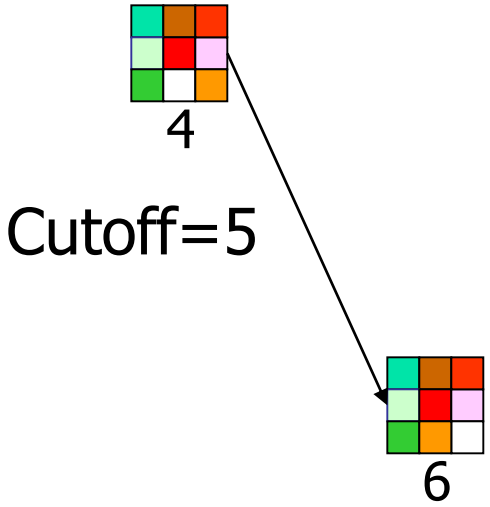
8-Puzzle

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



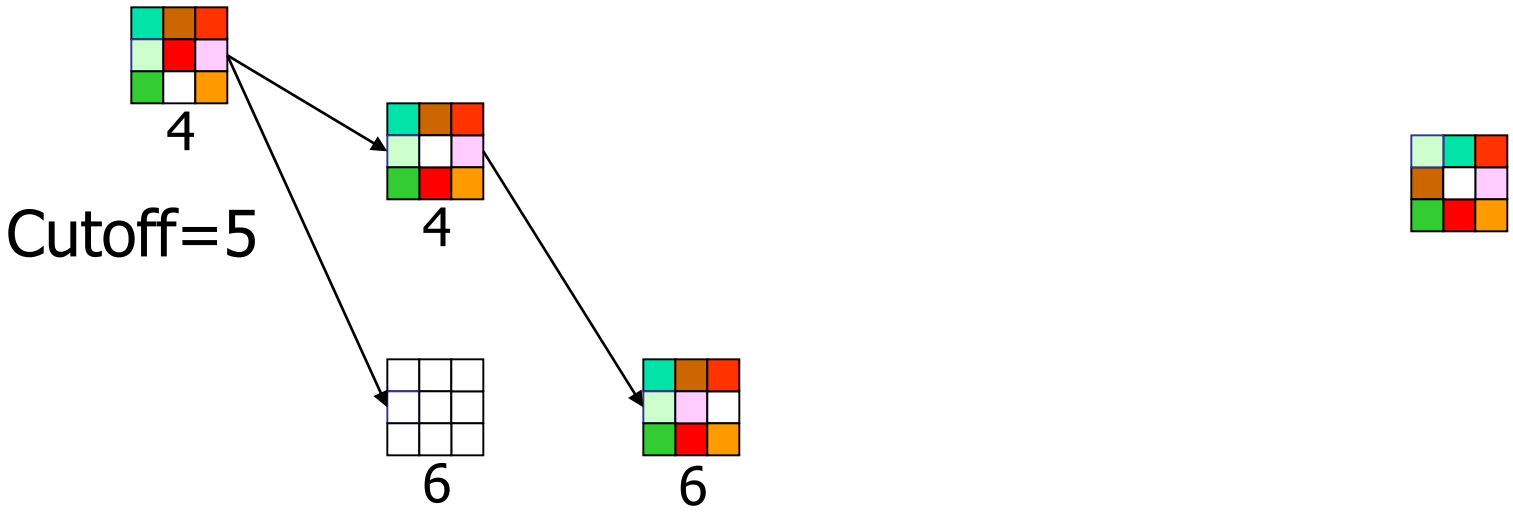
8-Puzzle

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



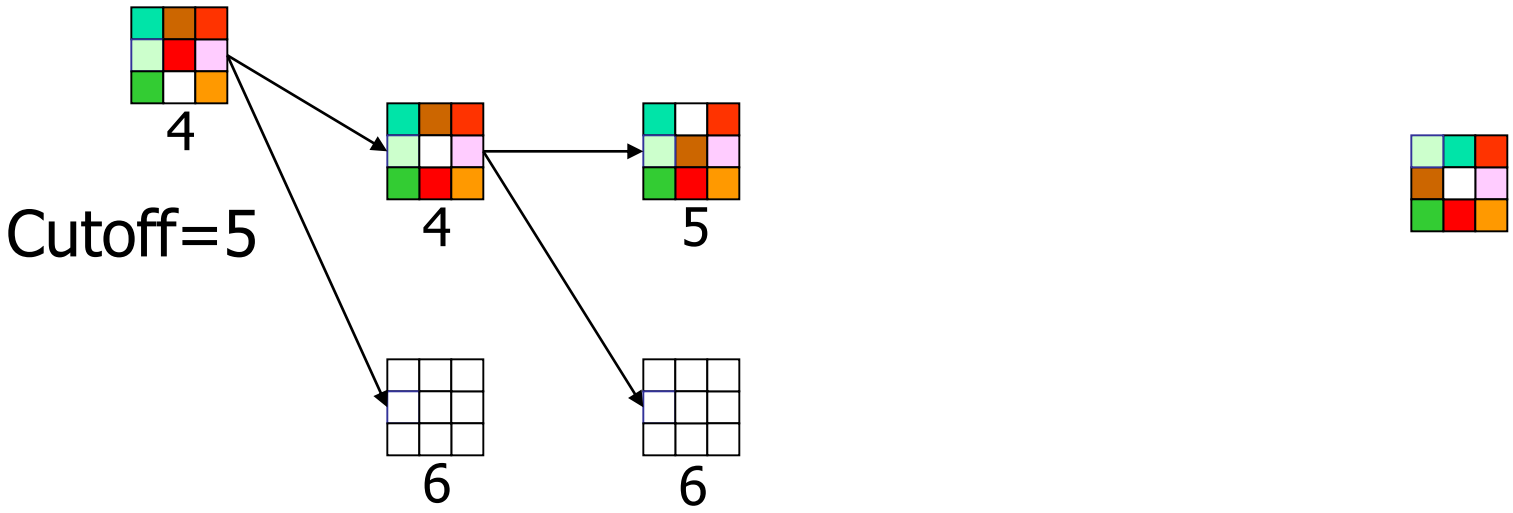
8-Puzzle

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



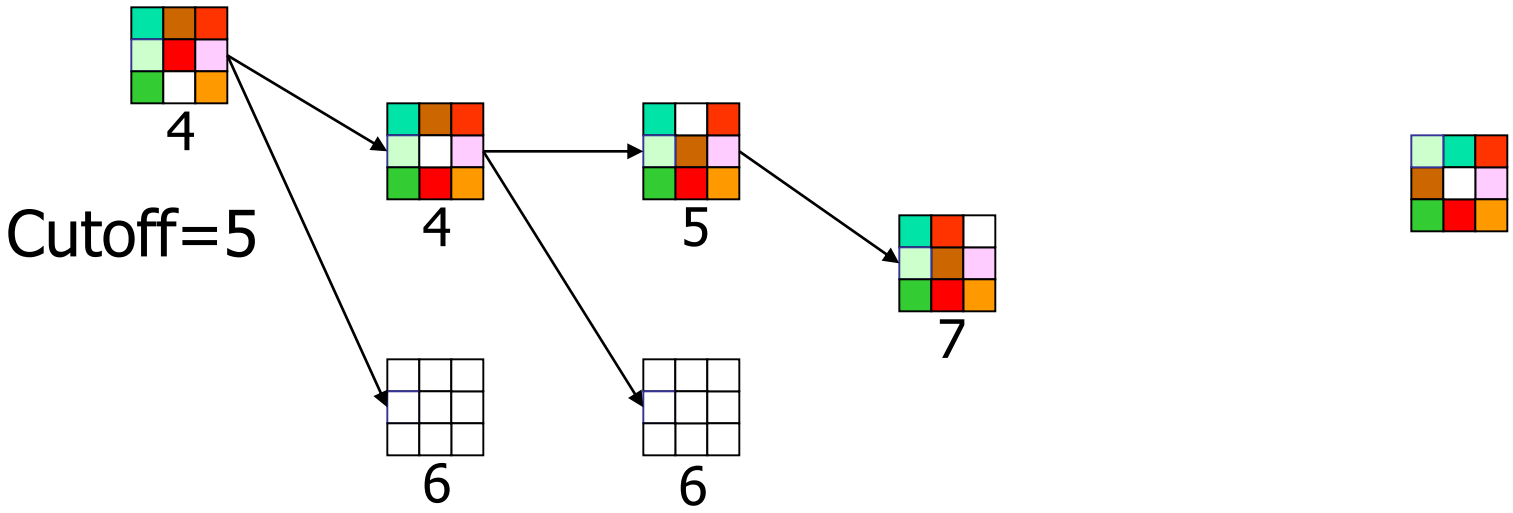
8-Puzzle

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



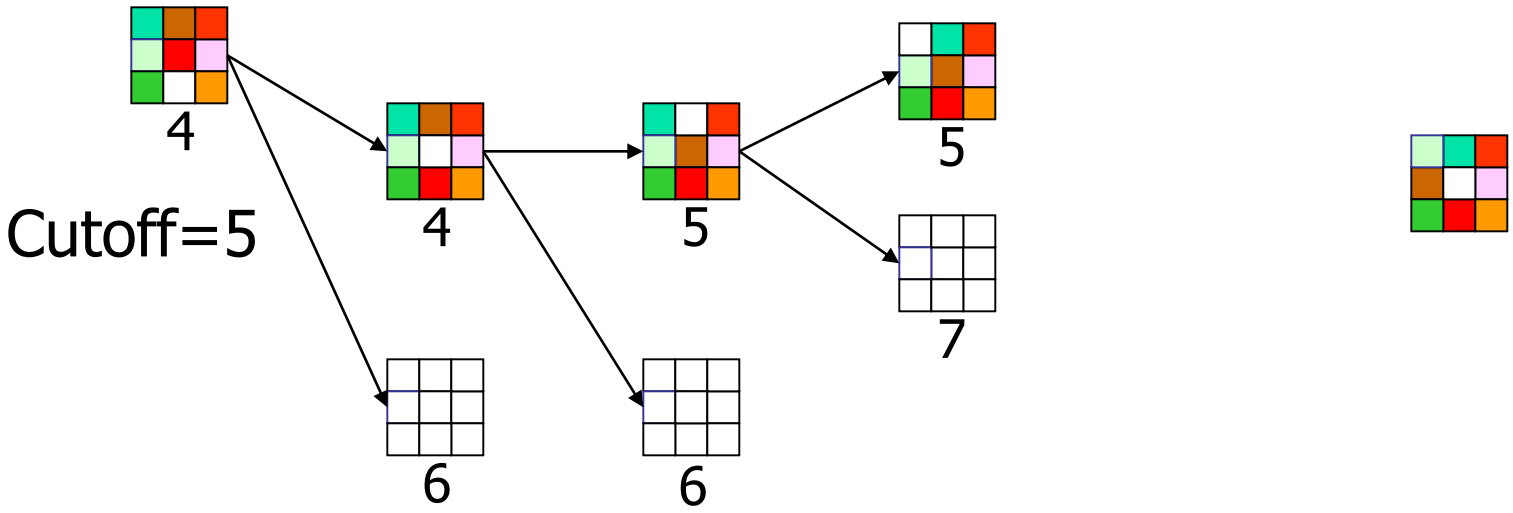
8-Puzzle

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



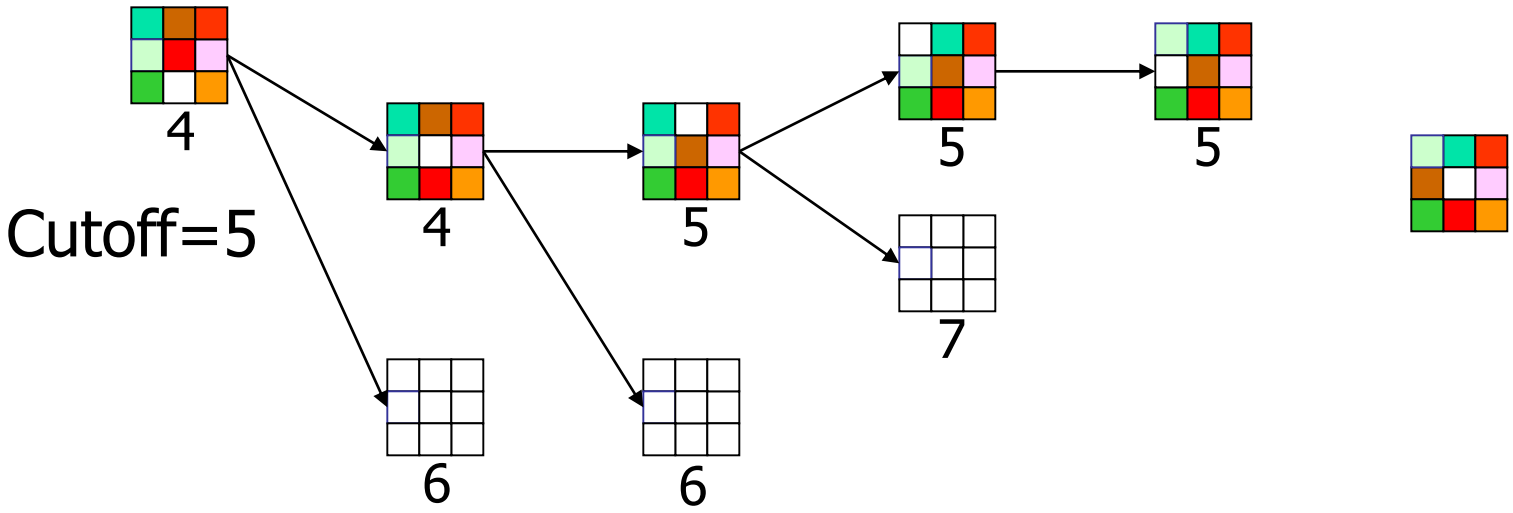
8-Puzzle

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



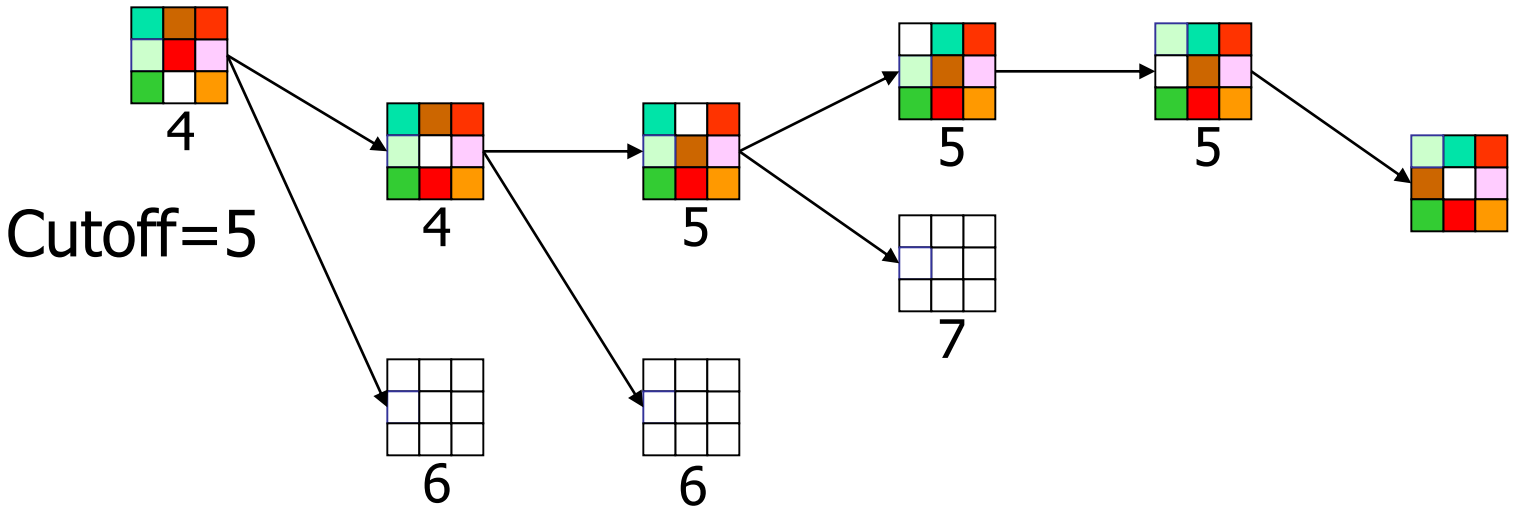
8-Puzzle

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

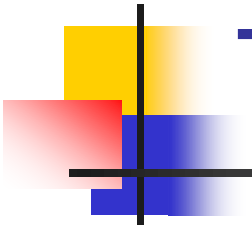


8-Puzzle

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

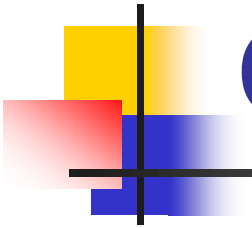


When to Use Search Techniques



- 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

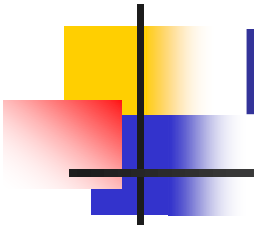
- The search space is large, and
  - There is no other available techniques, and
  - There exist "good" heuristics



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