ENCS3340 - Artificial Intelligence

Informed Search

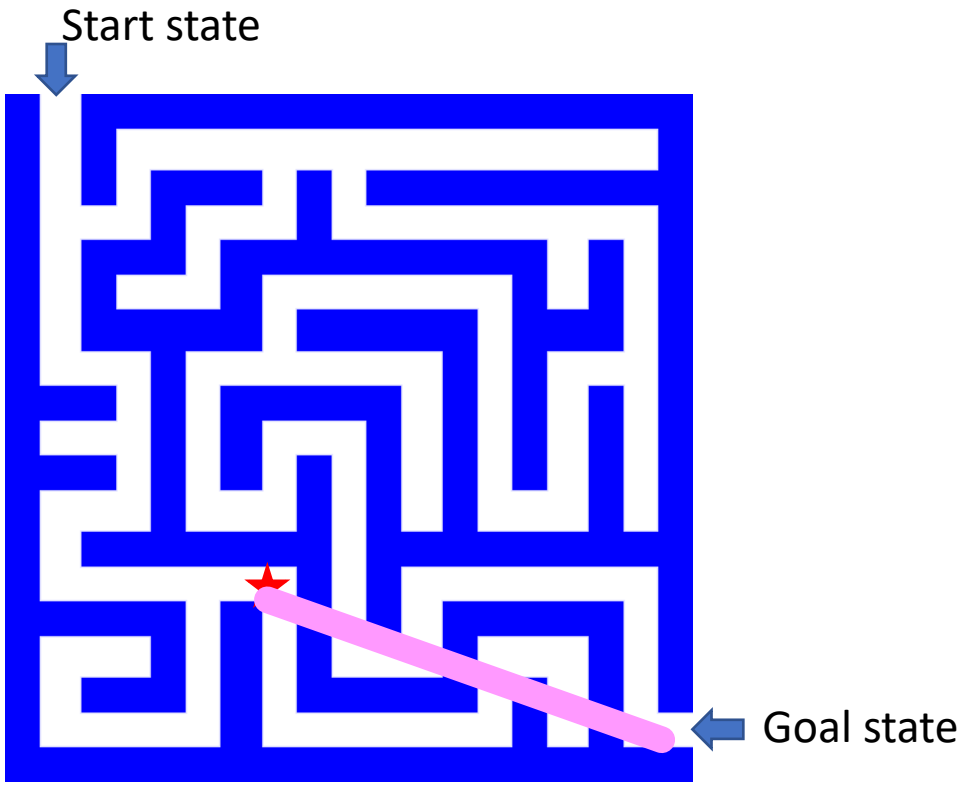
Informed Search

- relies on additional knowledge about the problem or domain
  - frequently expressed through heuristics (“rules of thumb”)
  - Idea: give the algorithm “hints” about the desirability of different states
  - Use an evaluation function to rank nodes and select the most promising one for expansion
- used to distinguish more promising paths towards a goal
  - may be mislead, depending on the quality of the heuristic
- in general, performs much better than uninformed search
  - but frequently still exponential in time and space for realistic problems
- Traditional Informed Search Strategies
  - Greedy best-first search
  - A* search

Heuristic Function

- Heuristic function h(n) estimates the cost of reaching goal from node n

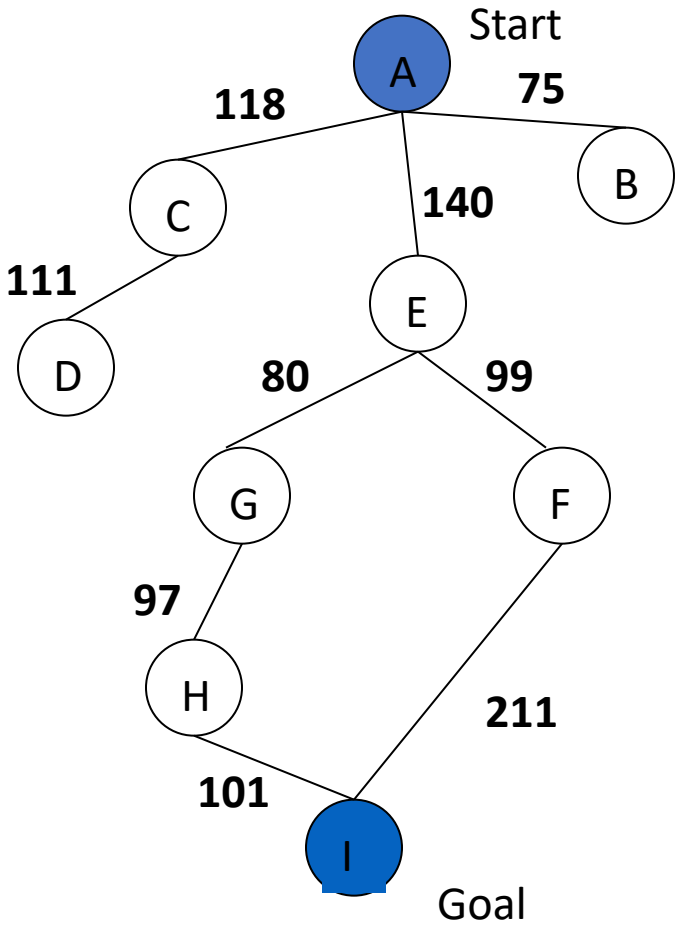
- Example:
  - the aerial (straight line distance) between n and goal in a maze problem



Greedy Best-First Search

- Expand the node that has the lowest value of the heuristic function h(n)

- Example:

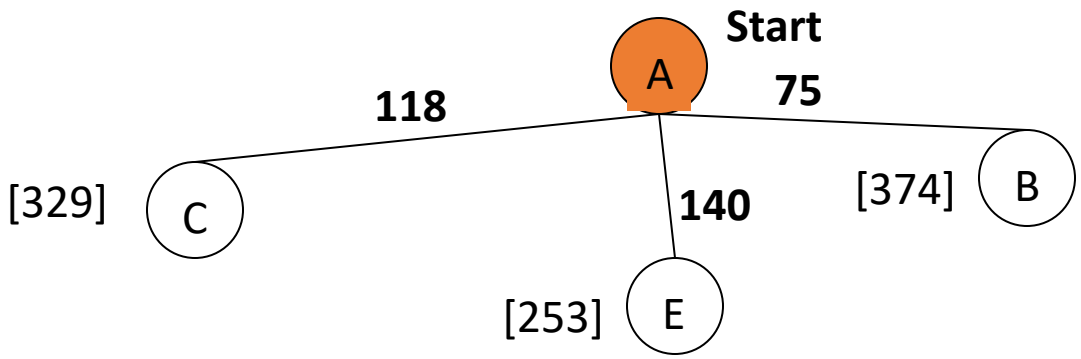


State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

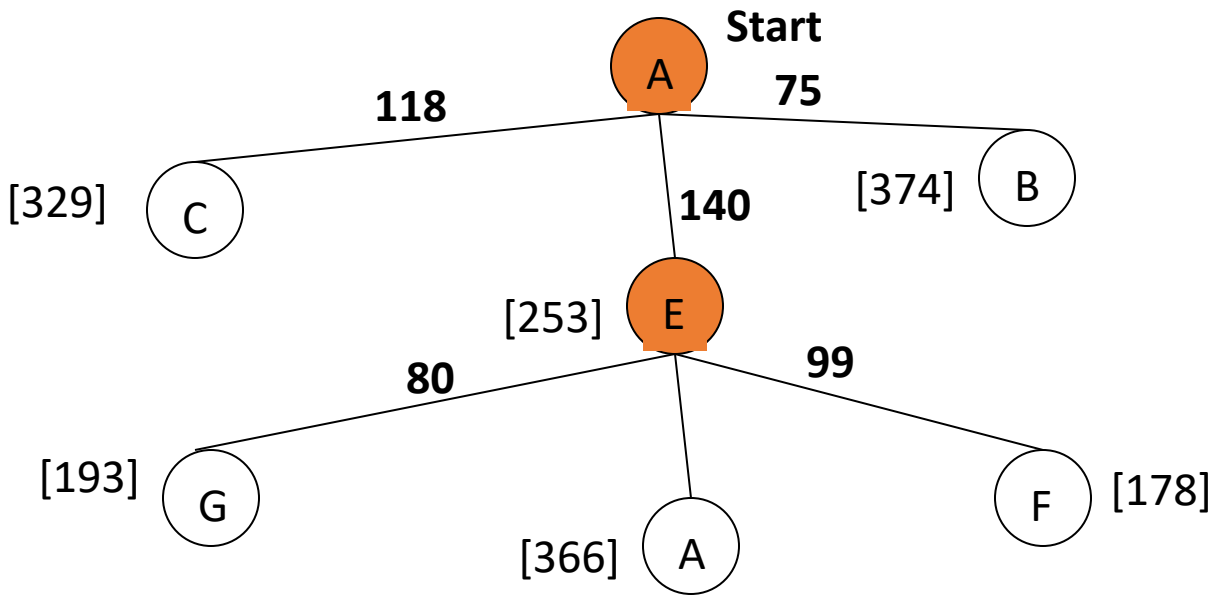
***h* (*n*) = straight-line distance heuristic**

Greedy Search: Tree Search

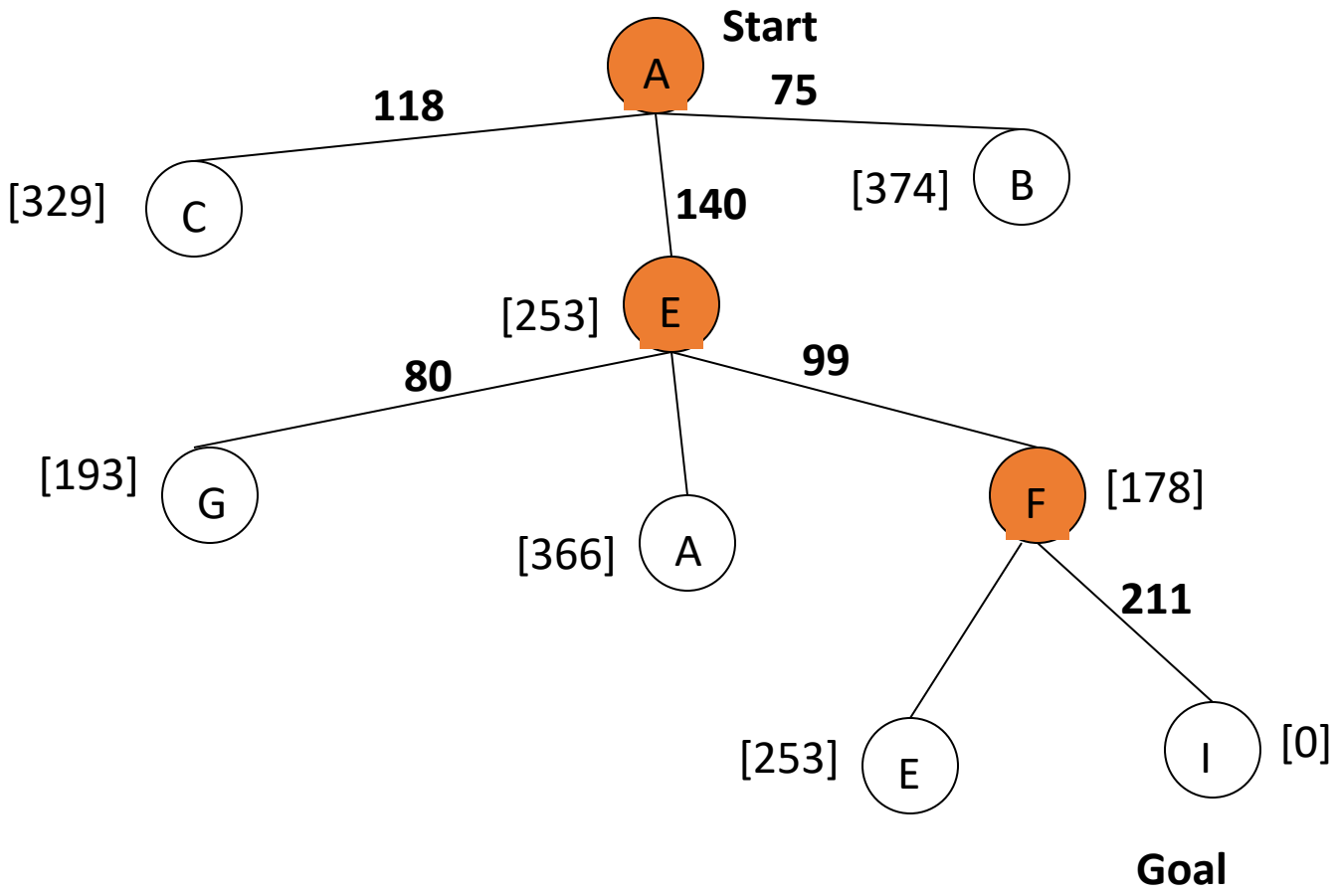
Greedy Search: Tree Search



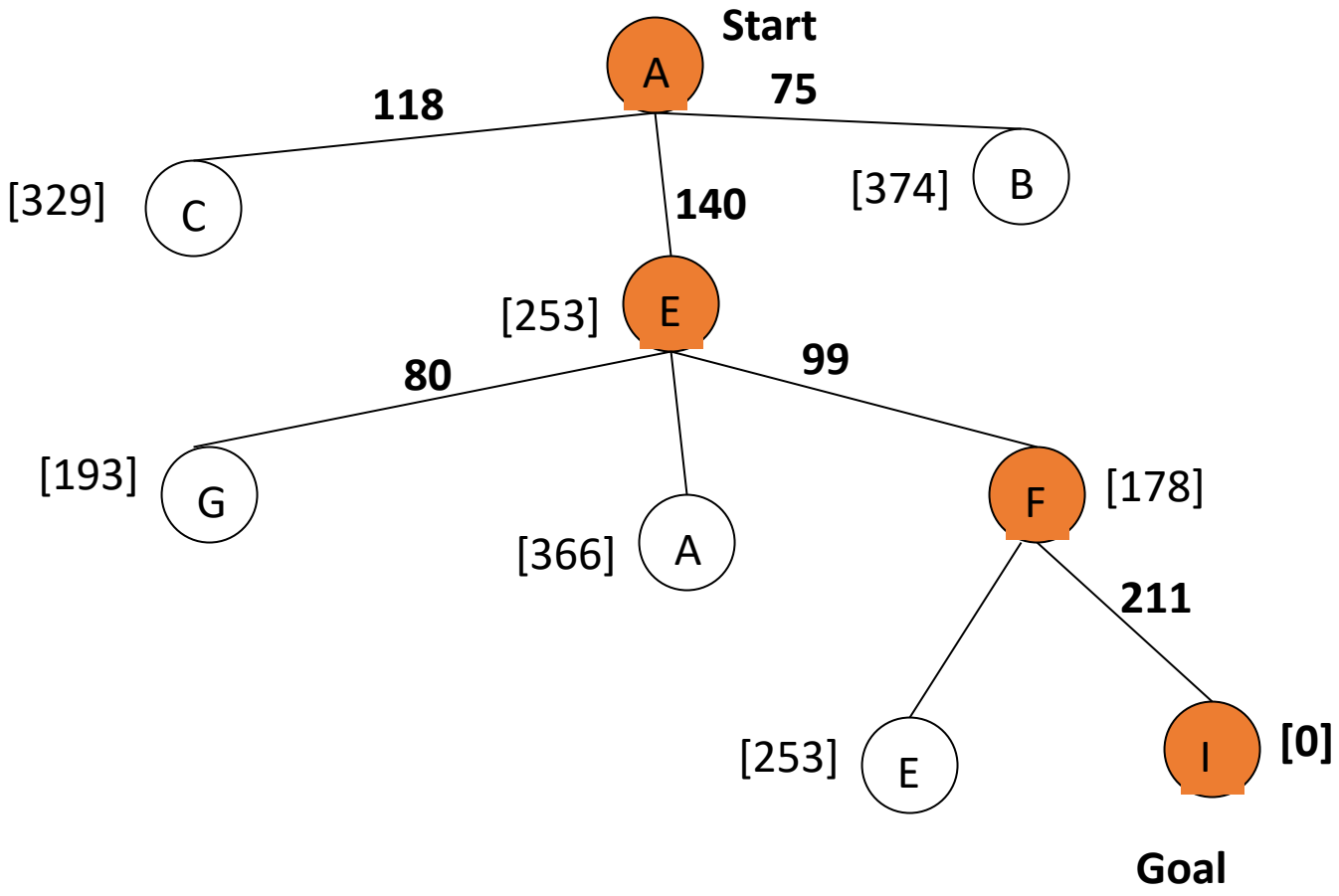
Greedy Search: Tree Search



Greedy Search: Tree Search

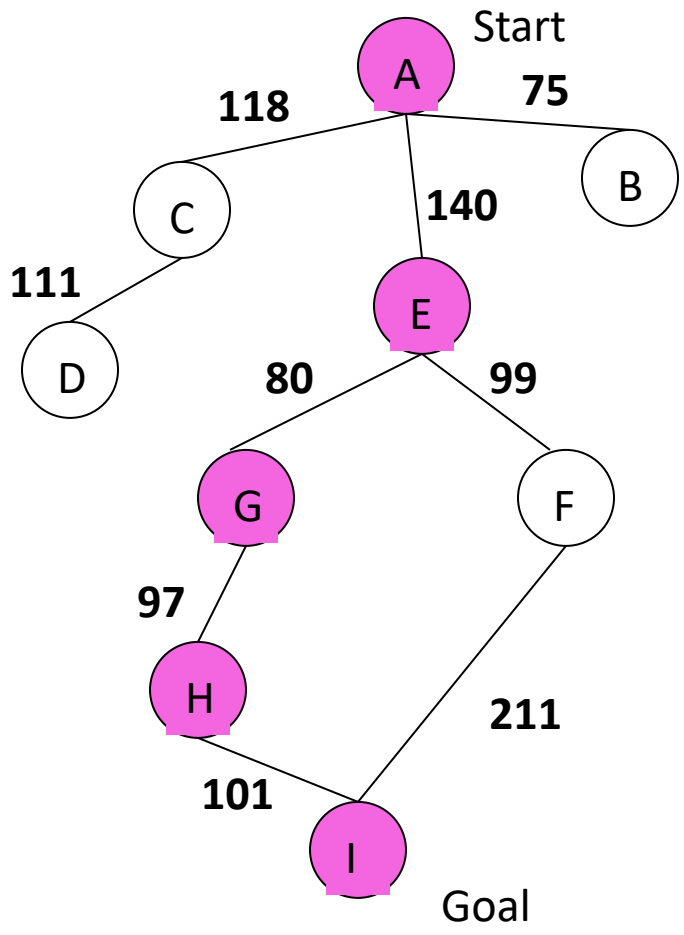


Greedy Search: Tree Search



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

Greedy Search: Optimal ?

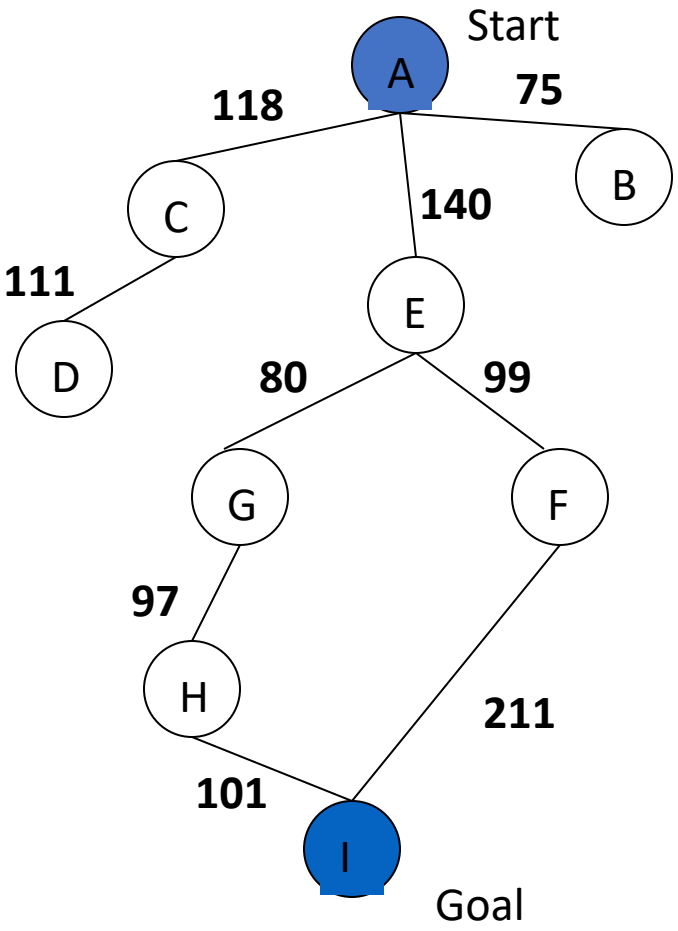


State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	**253**
F	178
G	**193**
H	**98**
I	0

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

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

Greedy Search: Complete ?

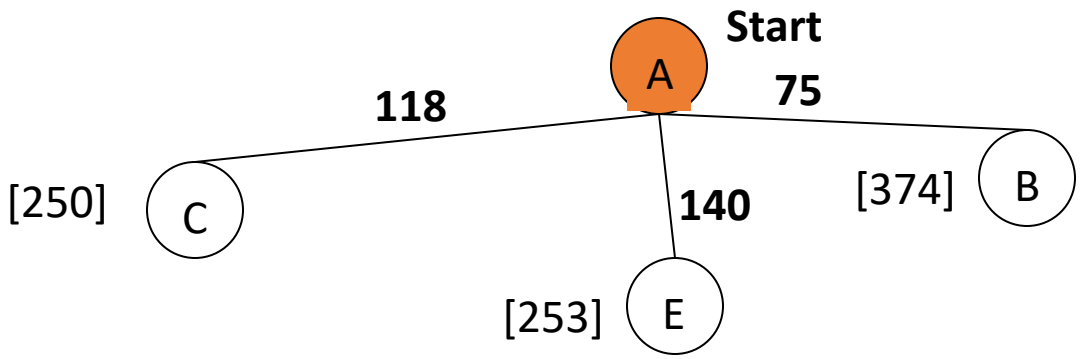


State	Heuristic: h(n)
A	366
B	374
** C	**250**
D	244
E	253
F	178
G	193
H	98
I	0

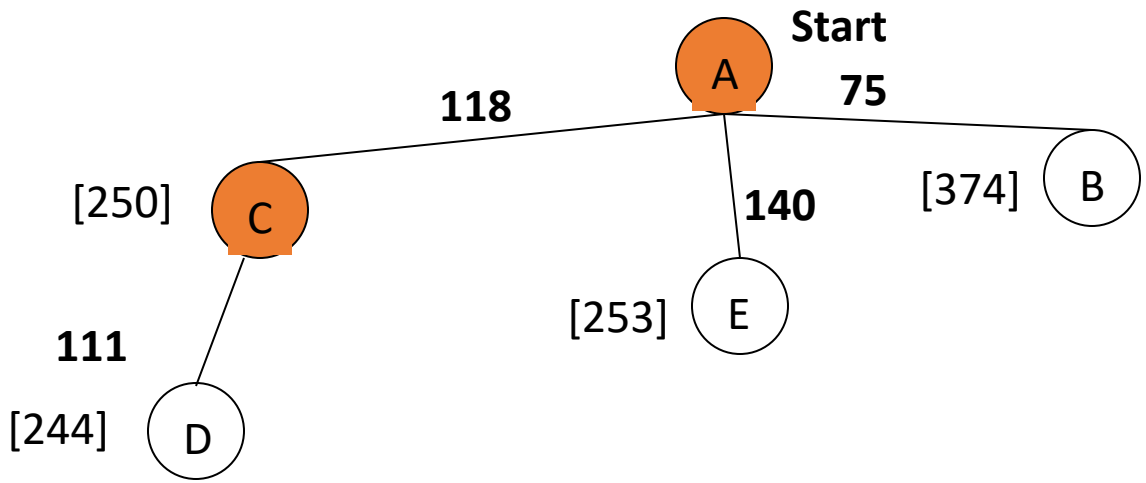
***h* (*n*) = straight-line distance heuristic**

Greedy Search: Tree Search

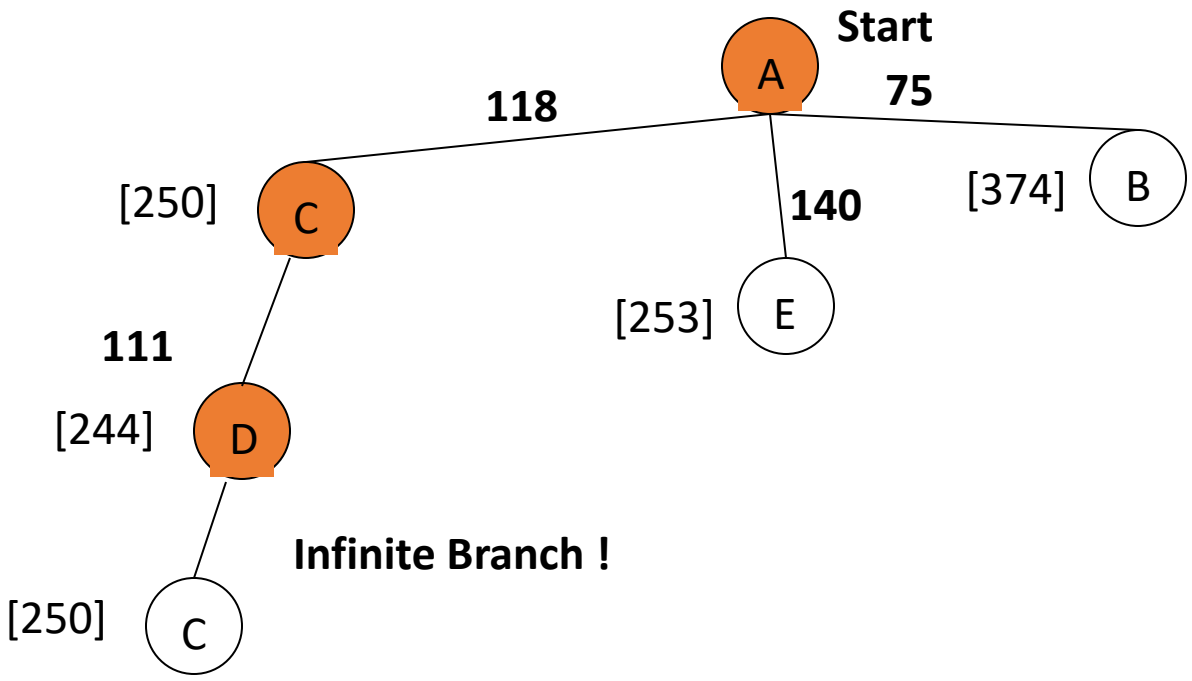
Greedy Search: Tree Search



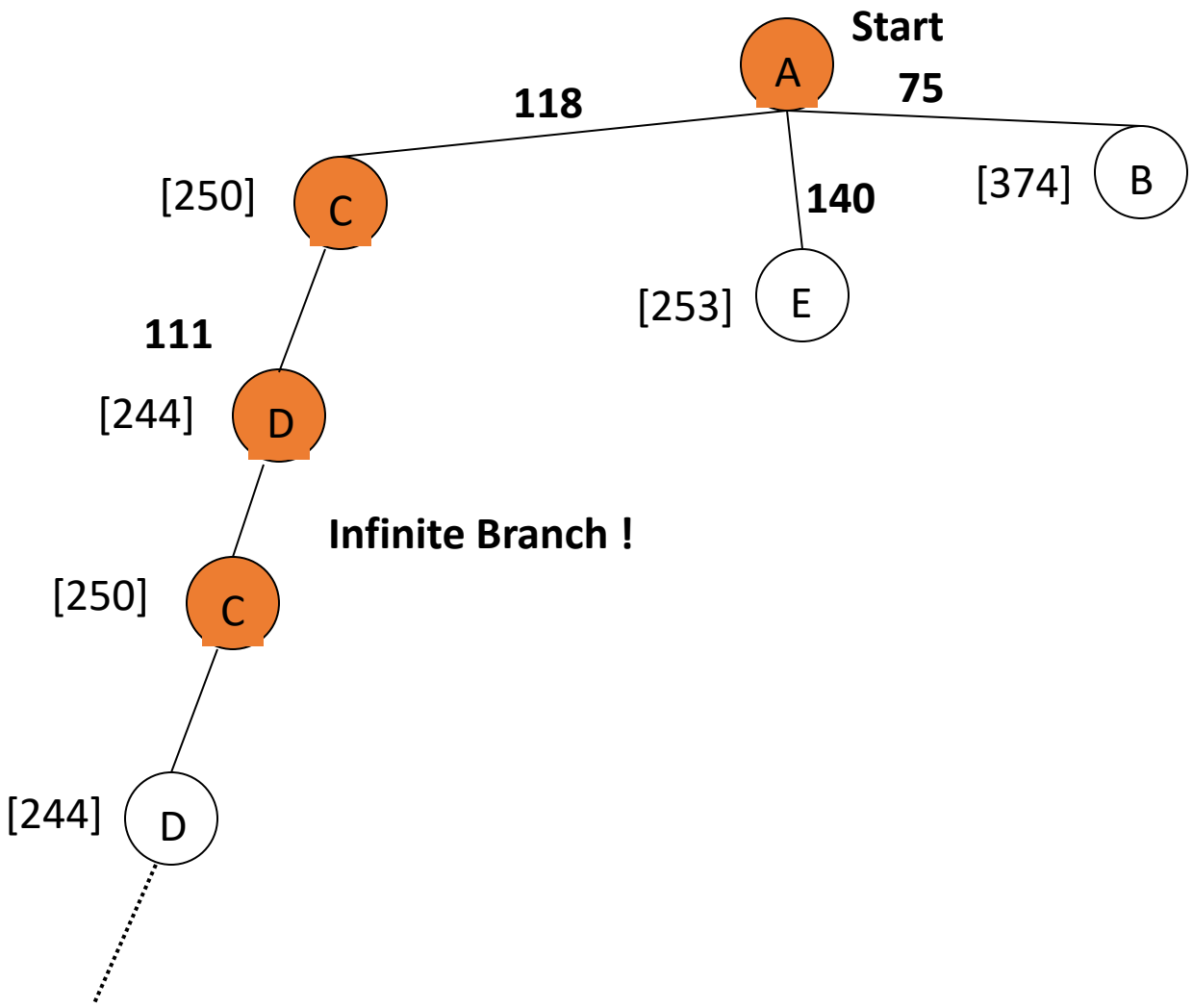
Greedy Search: Tree Search



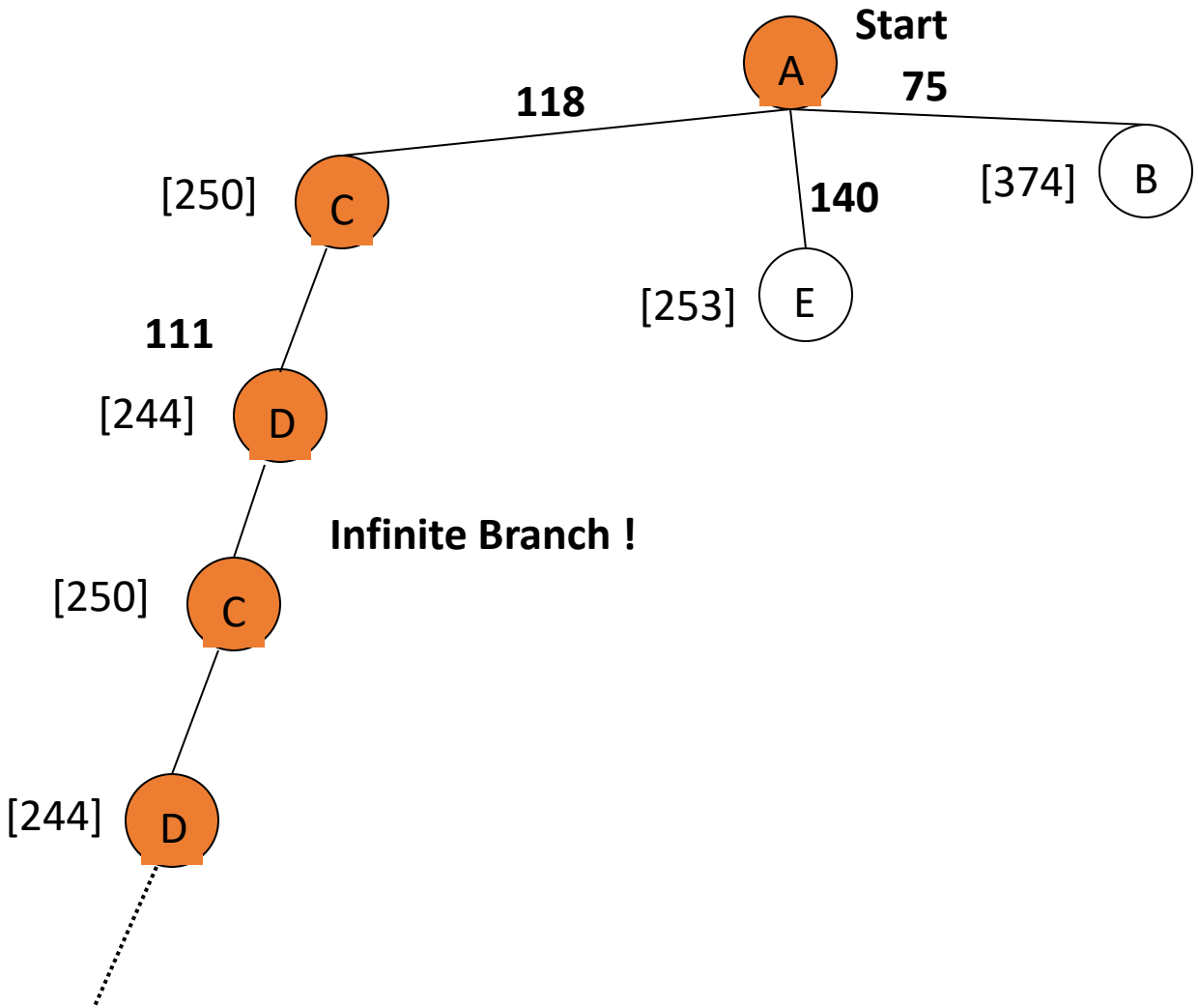
Greedy Search: Tree Search



Greedy Search: Tree Search



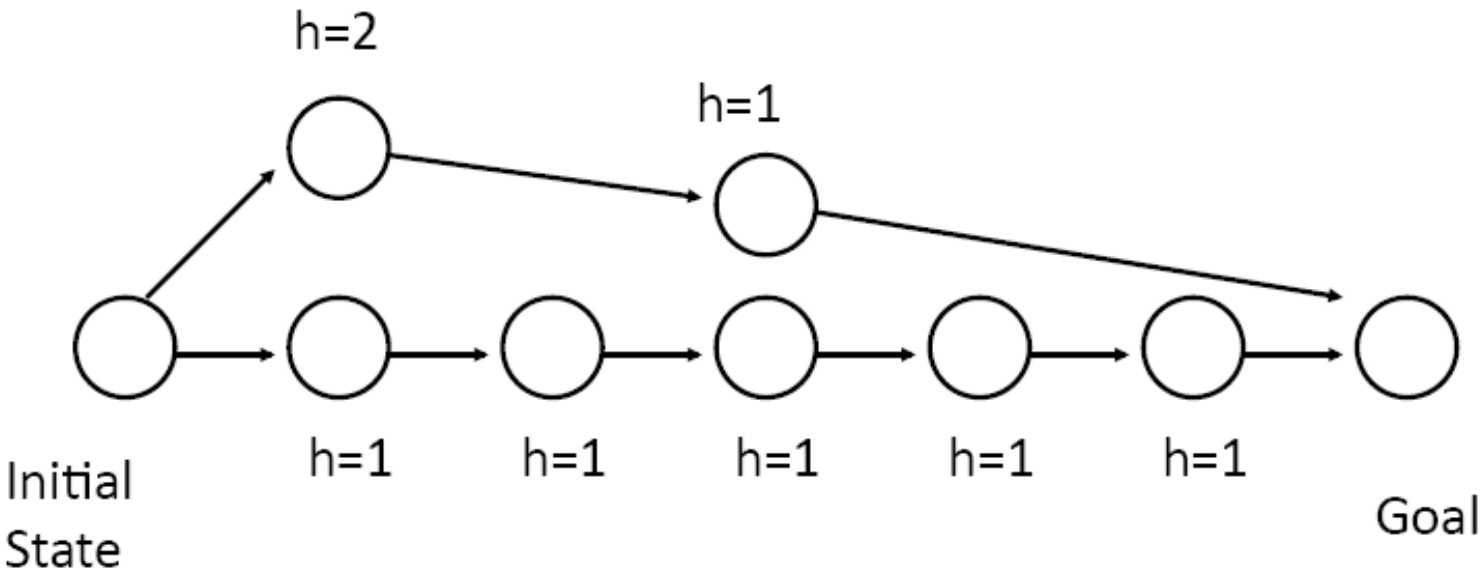
Greedy Search: Tree Search



Properties of Greedy Best-First Search

- Complete?
  - No – can get stuck in loops
- Optimal?
  - No
- Time?
  - Worst case: $O(b^m)$
  - Best case: O(bd) – If h(n) is 100% accurate
- Space?
  - Worst case: $O(b^m)$

How can we fix the greedy problem?



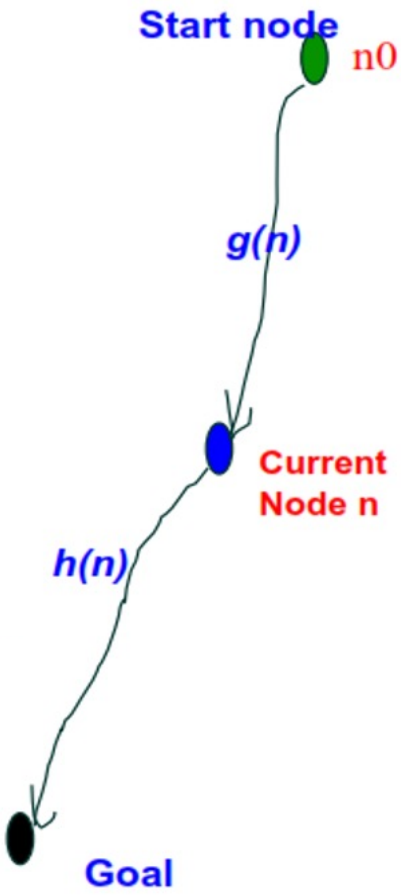
A* search

- Idea: avoid expanding paths that are already expensive
- Combines greedy and uniform-cost search to find the (estimated) cheapest path through the current node
- The evaluation function f(n) is the estimated total cost of the path through node n to the goal:

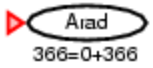
$$f(n) = g(n) + h(n)$$

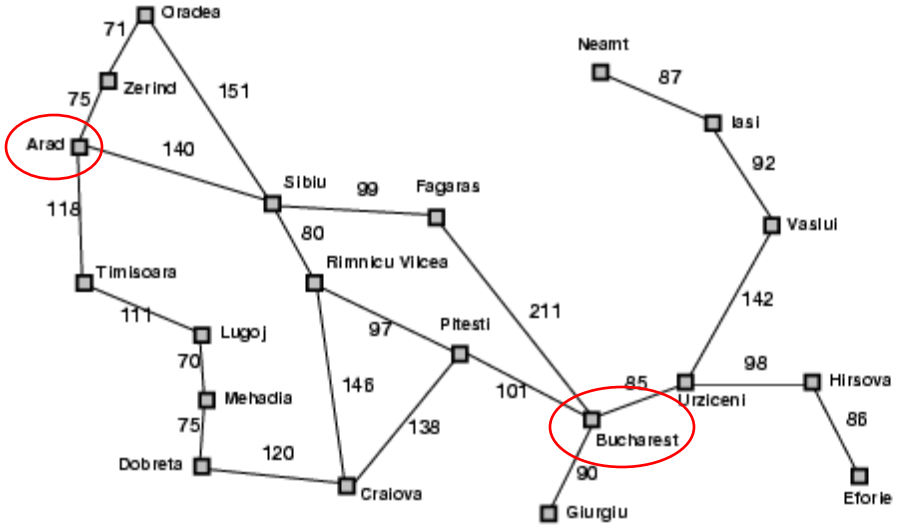
g(n): cost so far to reach n (path cost)

h(n): estimated cost from n to goal (heuristic)



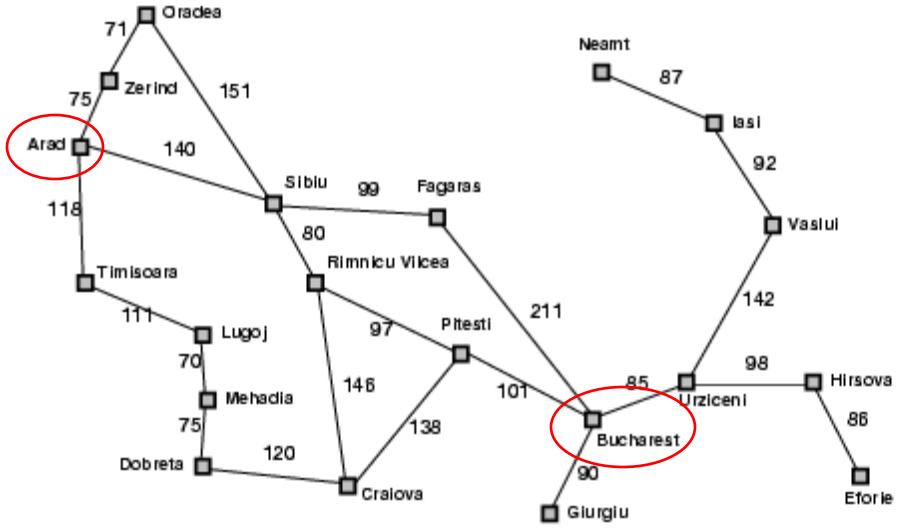
A* Search Example





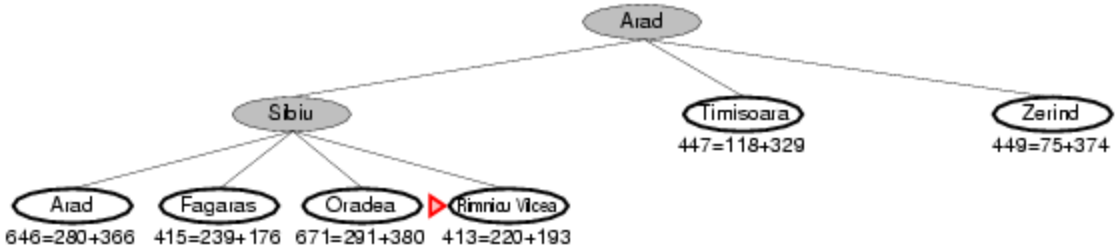
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

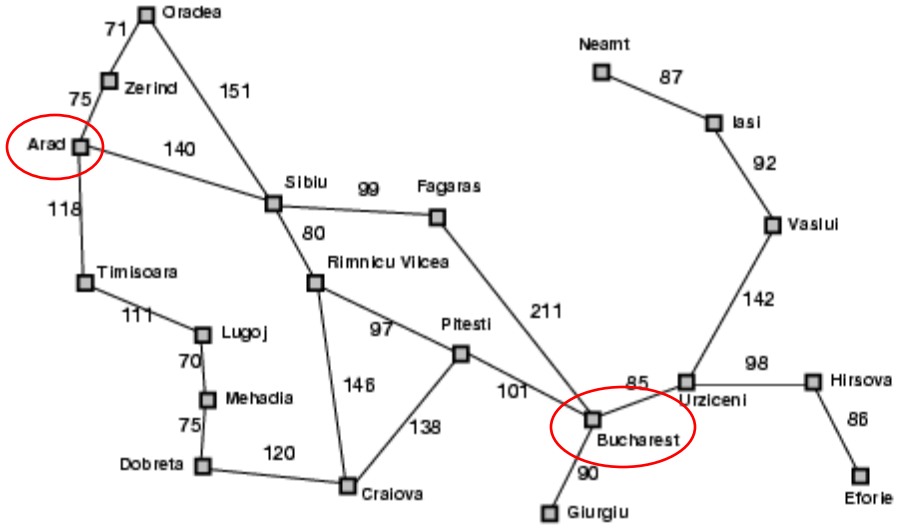
A* Search Example



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

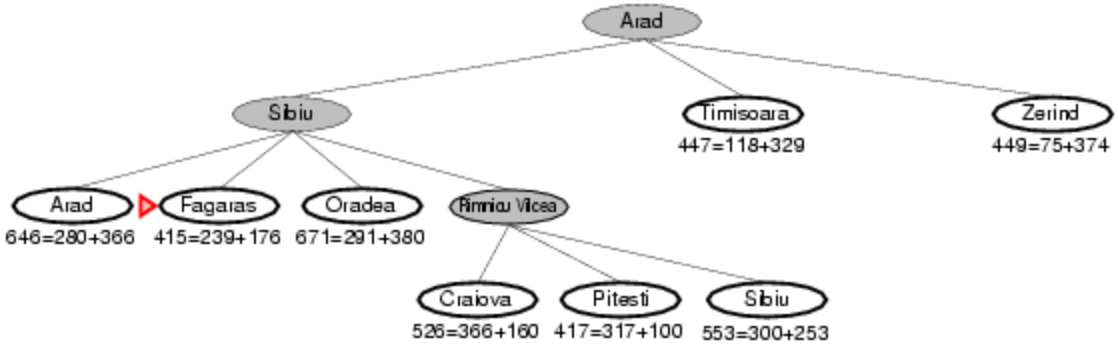
A* Search Example

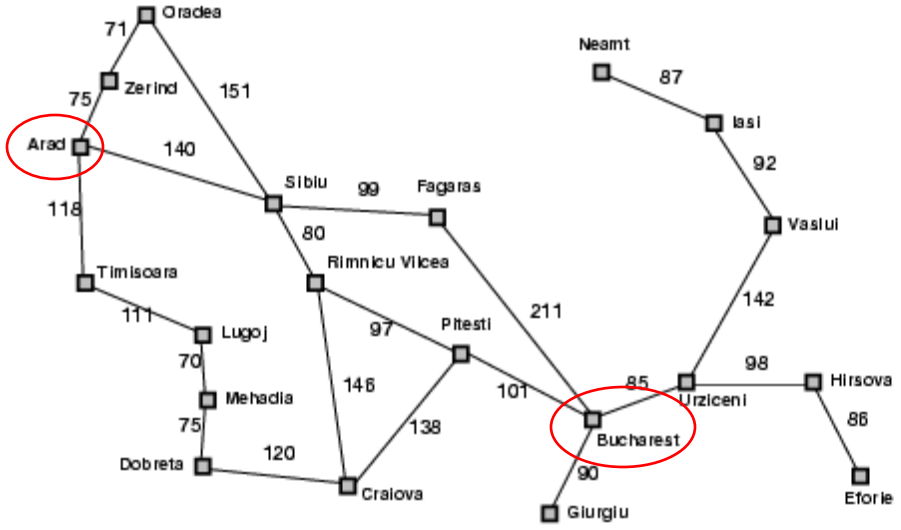




Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

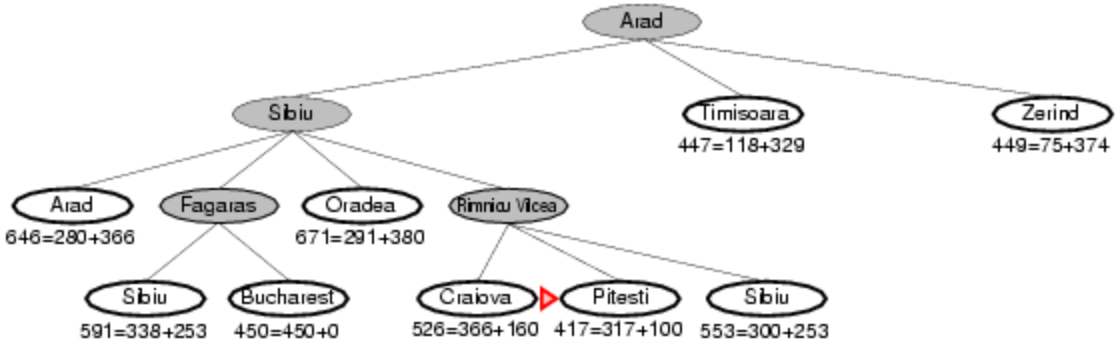
A* Search Example

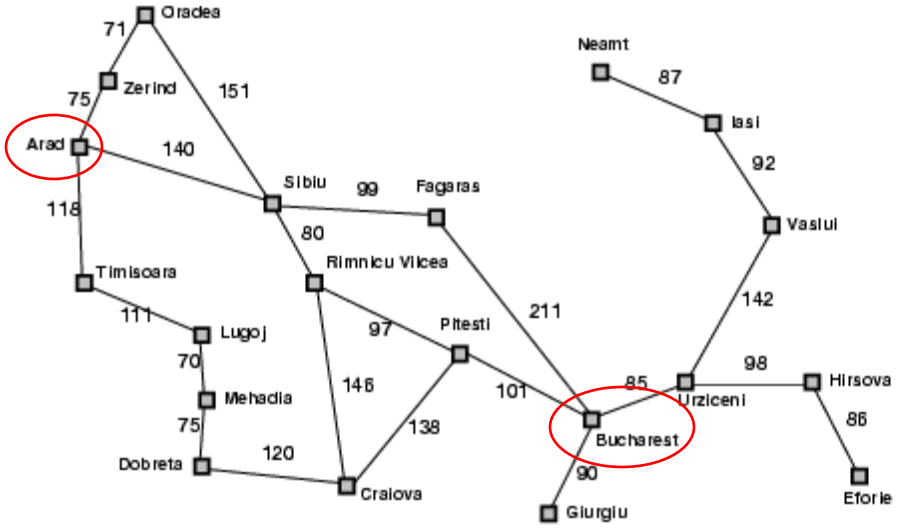




Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

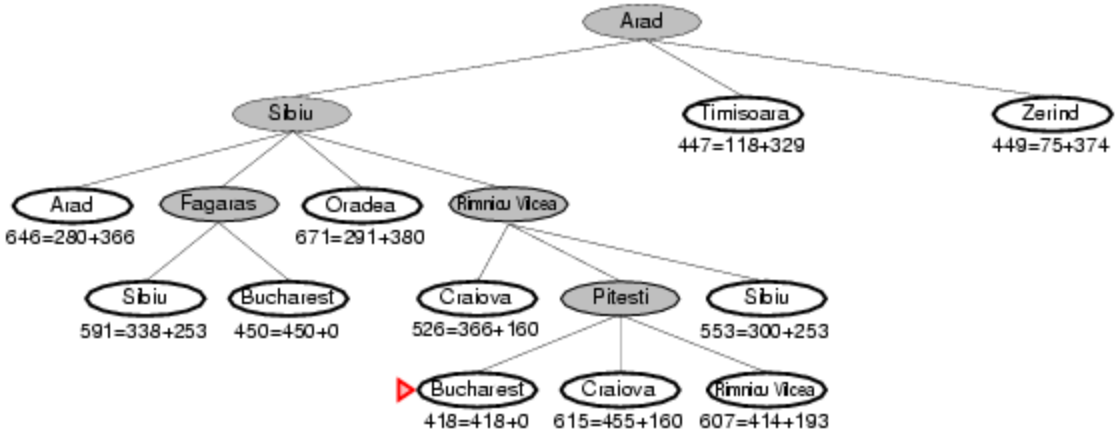
A* Search Example

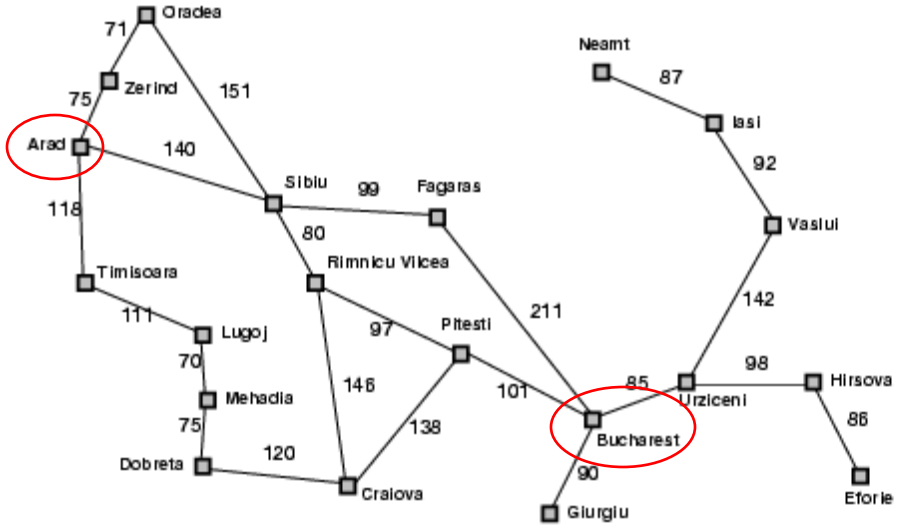




Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

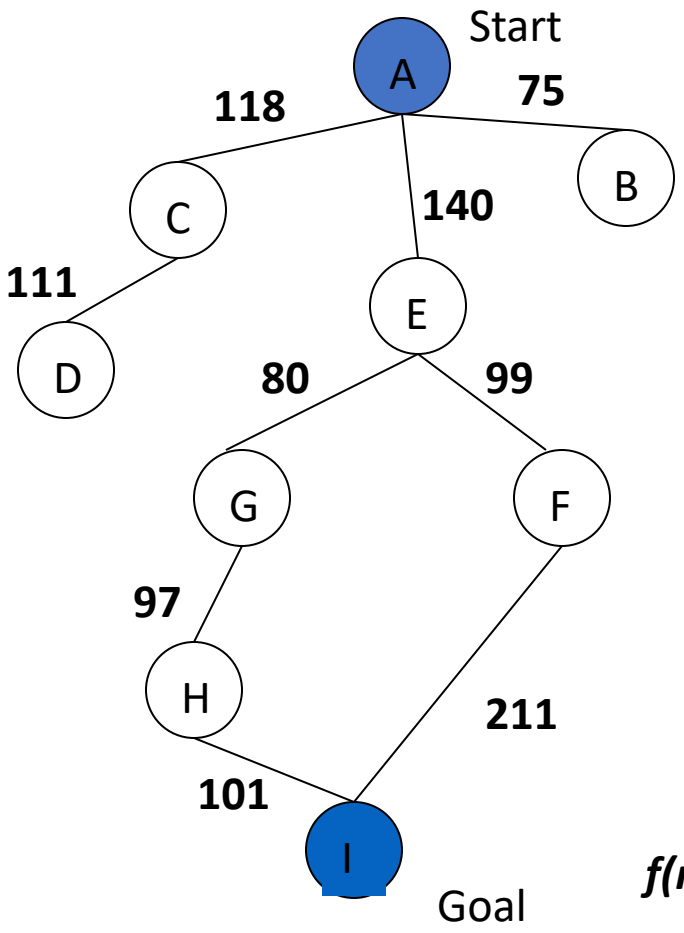
A* Search Example





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: Another example

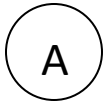


State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

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

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

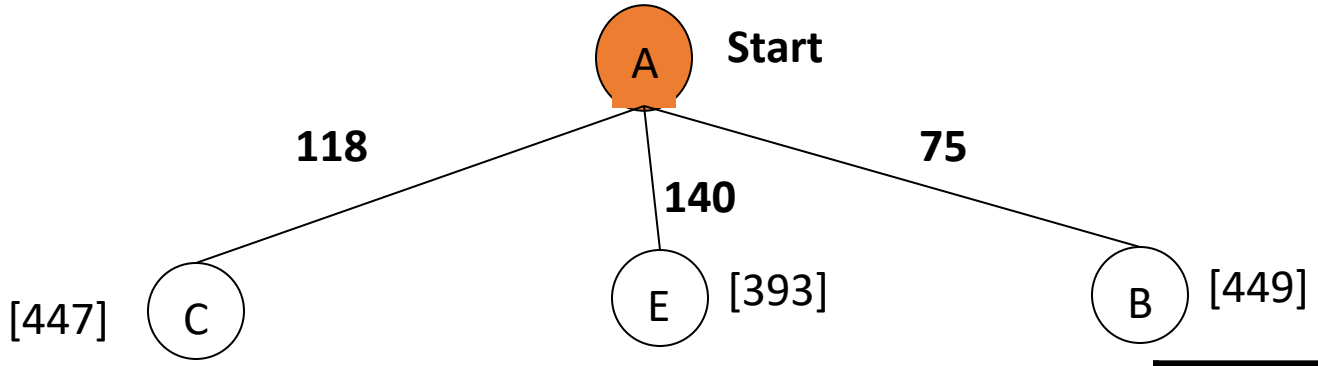
A* Search: Tree Search



Start

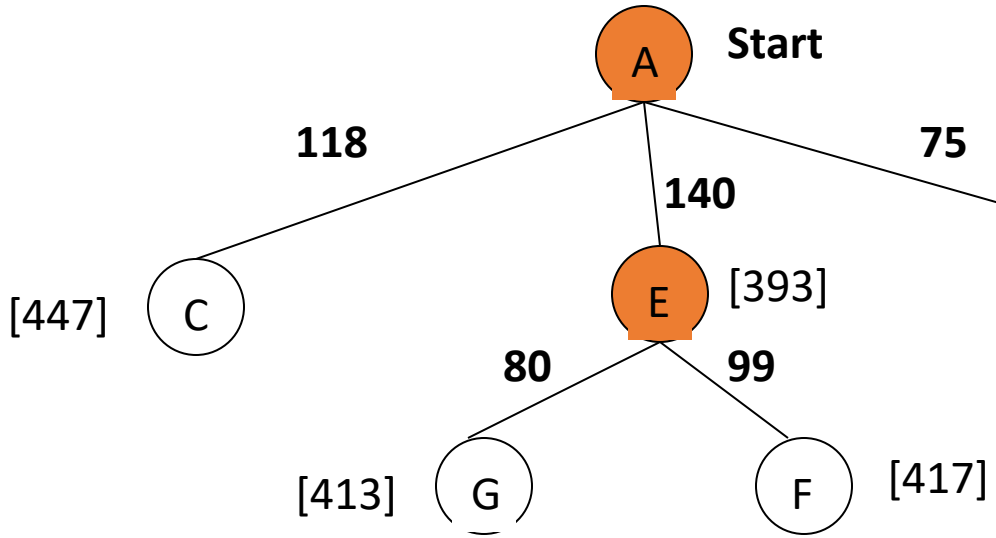
State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

A* Search: Tree Search



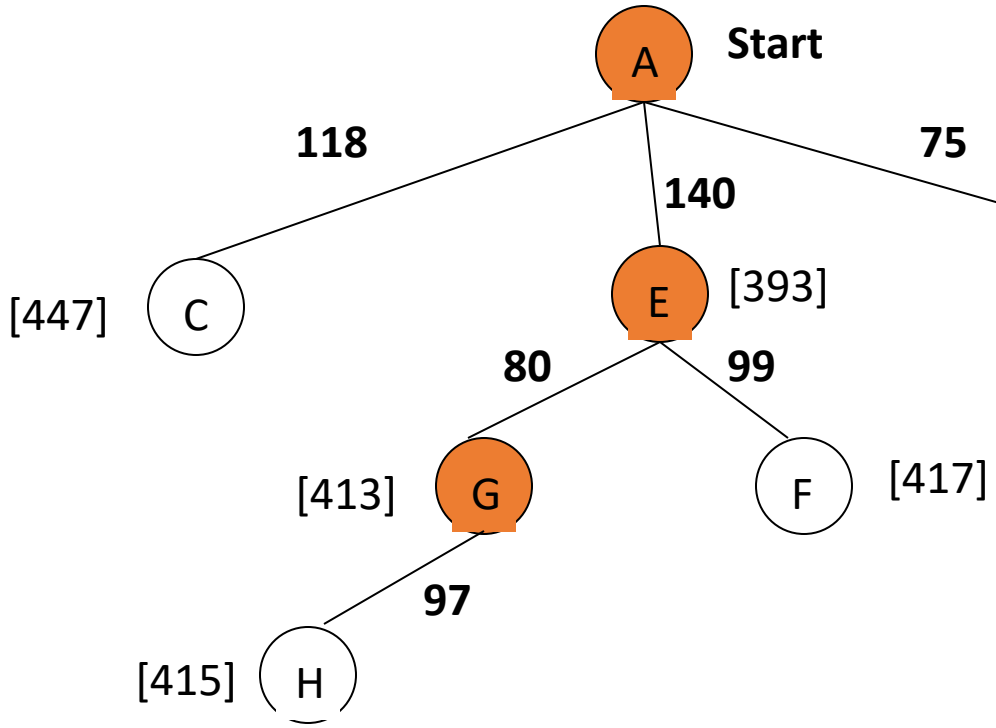
State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

A* Search: Tree Search



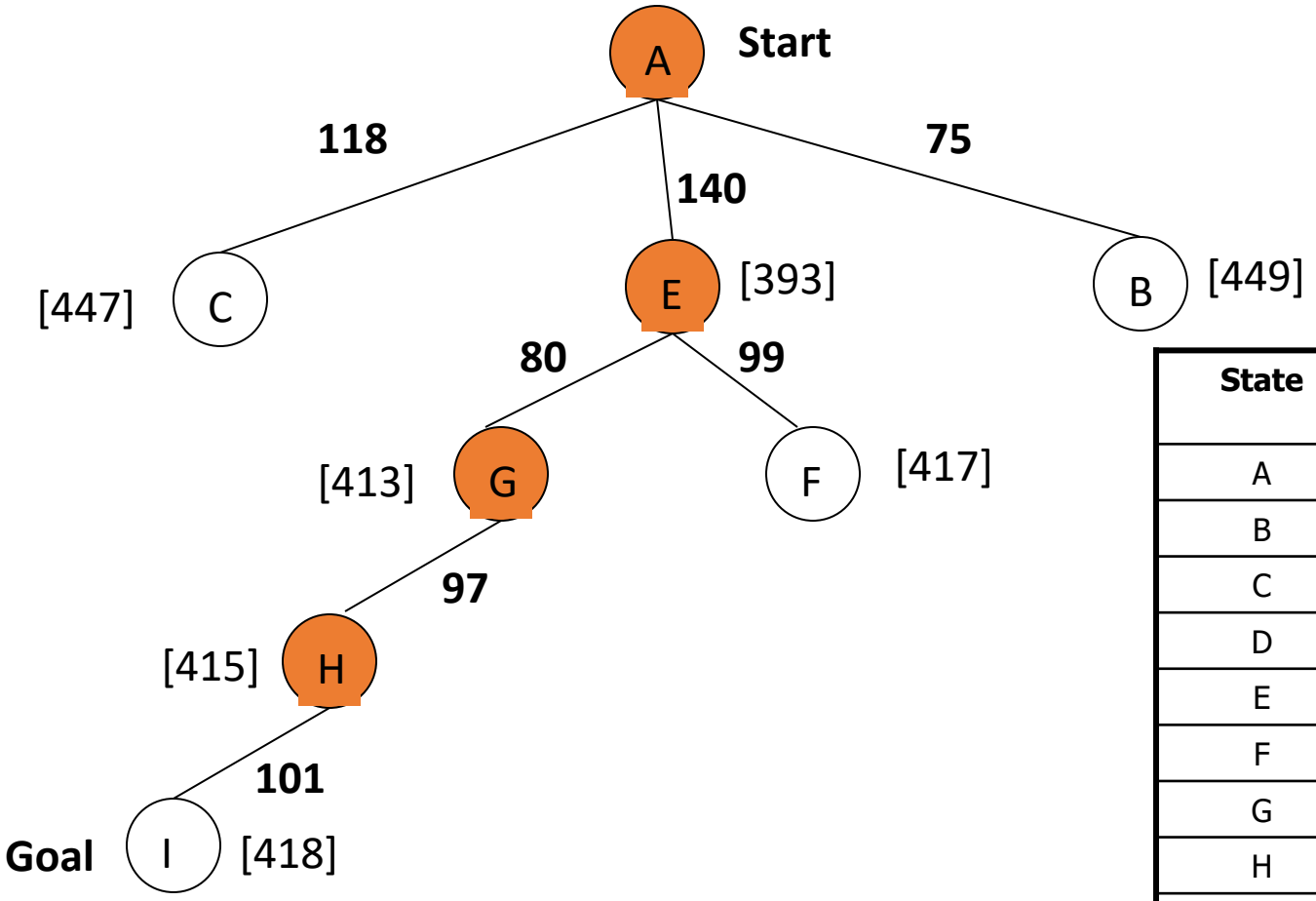
State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

A* Search: Tree Search



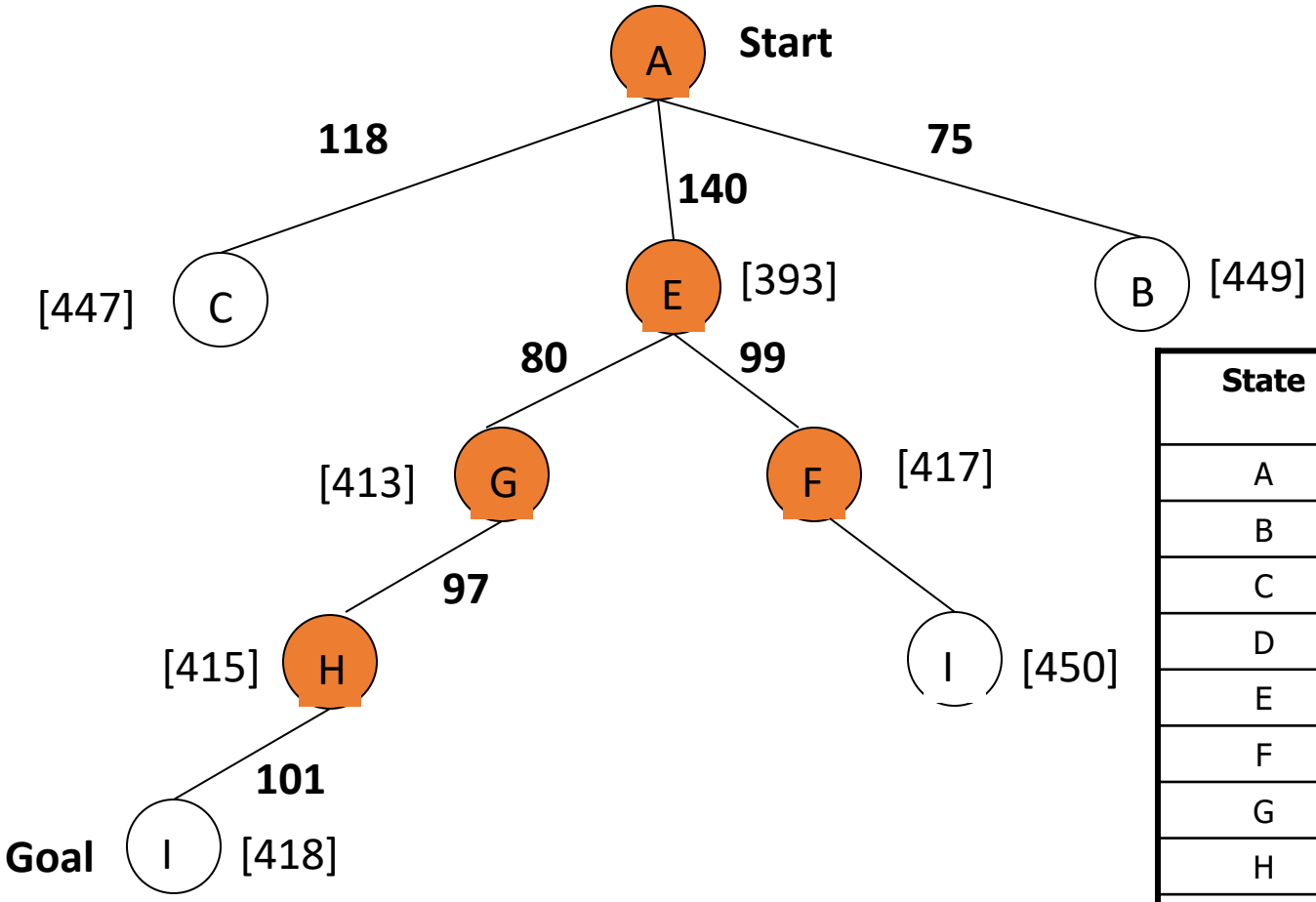
State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

A* Search: Tree Search



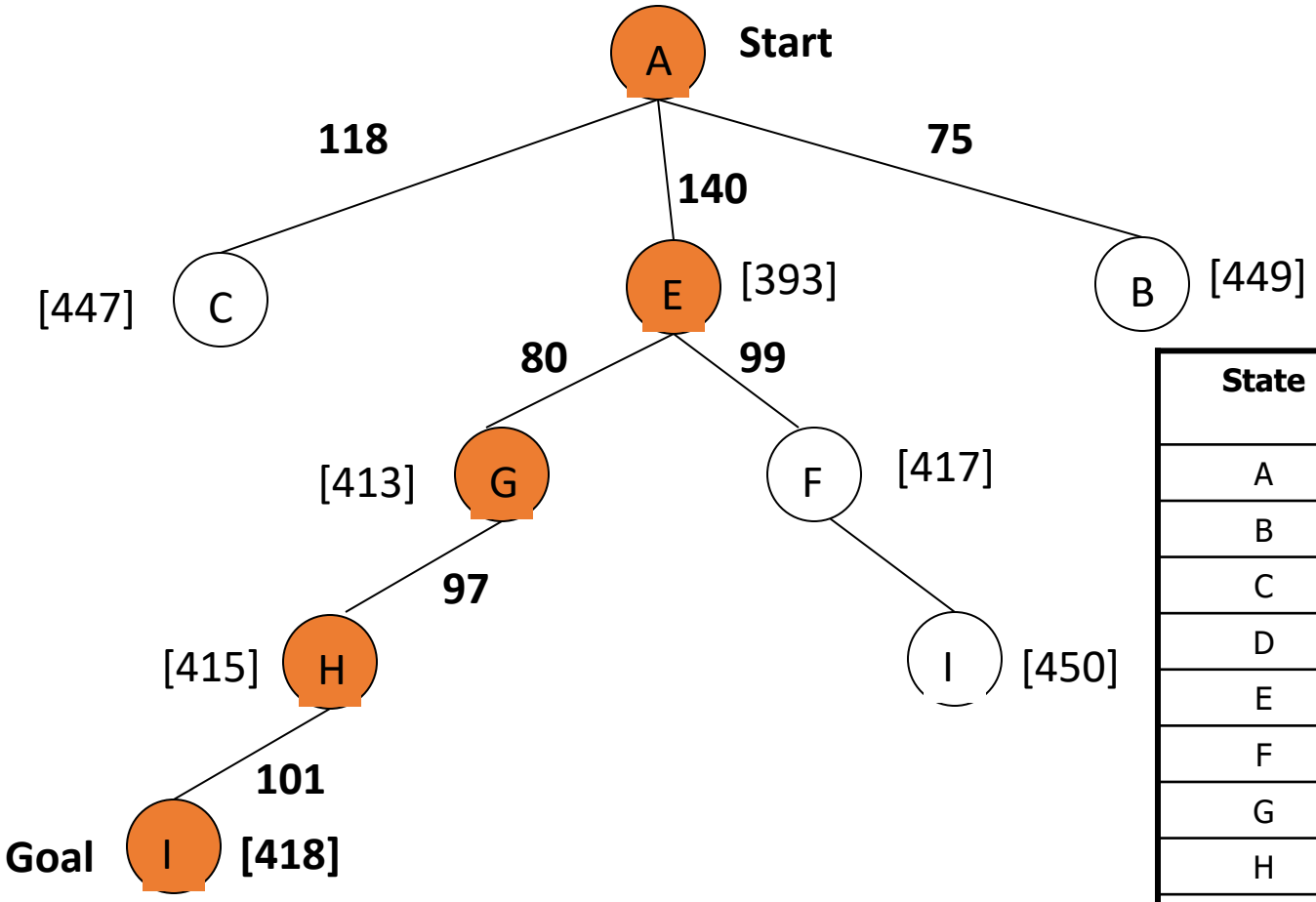
State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

A* Search: Tree Search



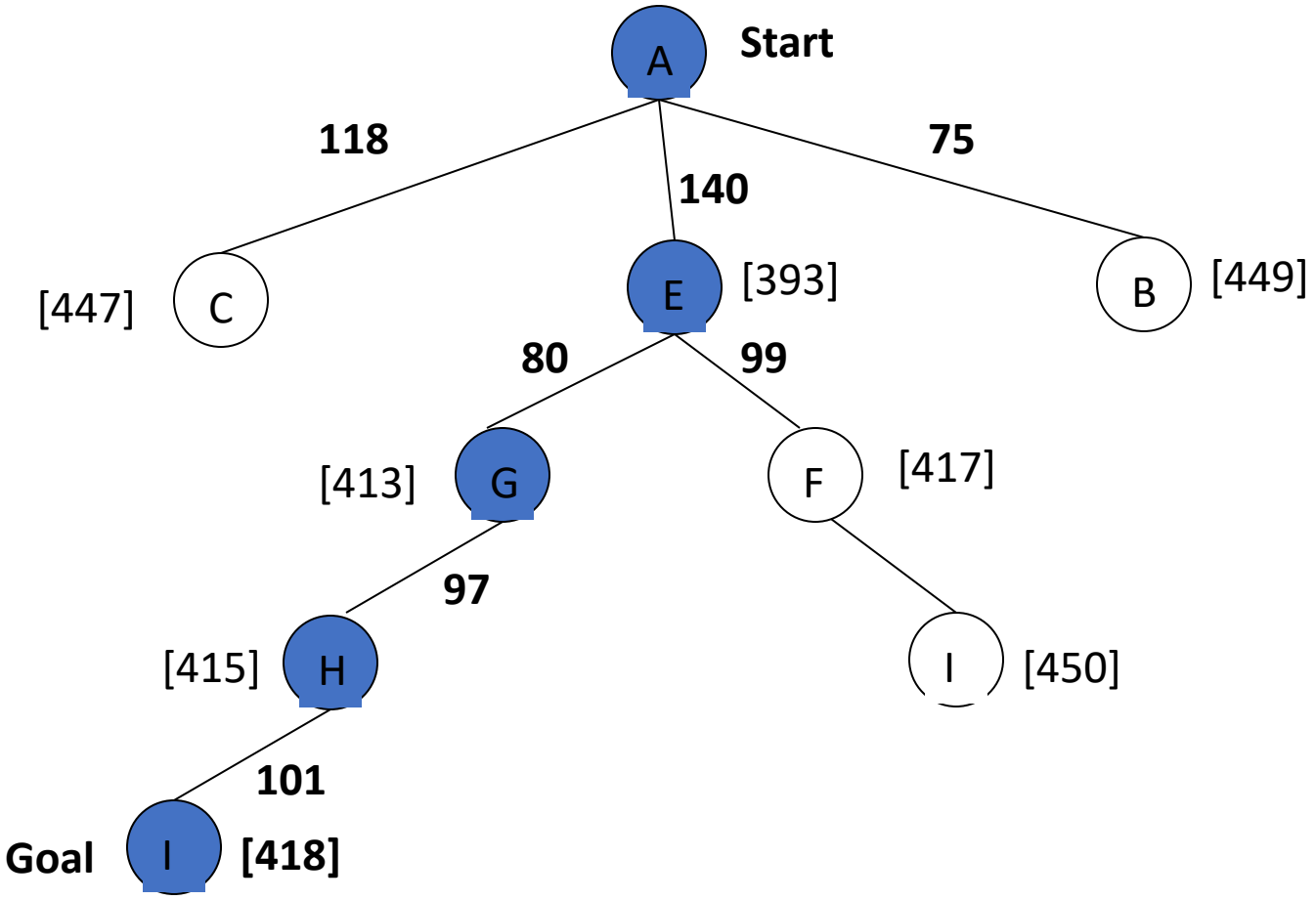
State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

A* Search: Tree Search



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

A* Search: Tree Search



Admissible Heuristics

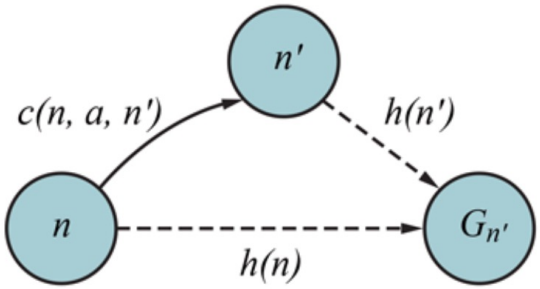
- A heuristic h(n) is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: straight line distance never overestimates the actual road distance
- Theorem: If h(n) is admissible, A* is optimal

Consistent Heuristics

- A heuristic h(n) is consistent if, for every node n and every successor n' of n generated by an action a we have:

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

- This is a form of the triangle inequality, If the heuristic h is consistent, then the single number h(n) will be less than the sum of the cost c(n, a, n') of the action from n to n' plus the heuristic estimate h(n')

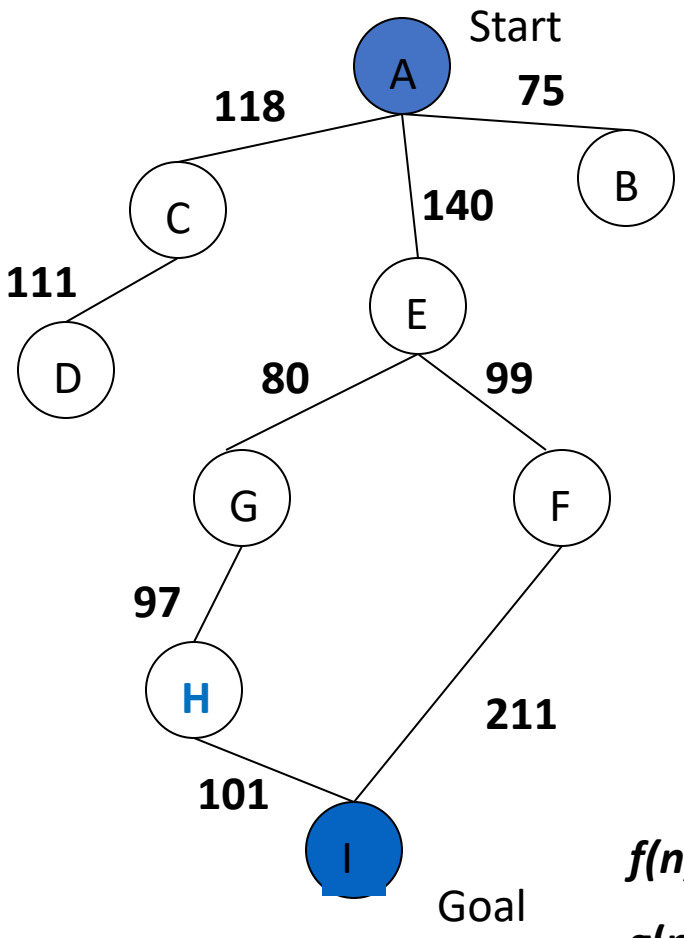


- Consistency is a stronger property than admissibility. Every consistent heuristic is admissible (but not vice versa).

- The straight line distance is a consistent heuristic for routing problems.

A* Search: h not admissible !

h() overestimates the cost to reach the goal state



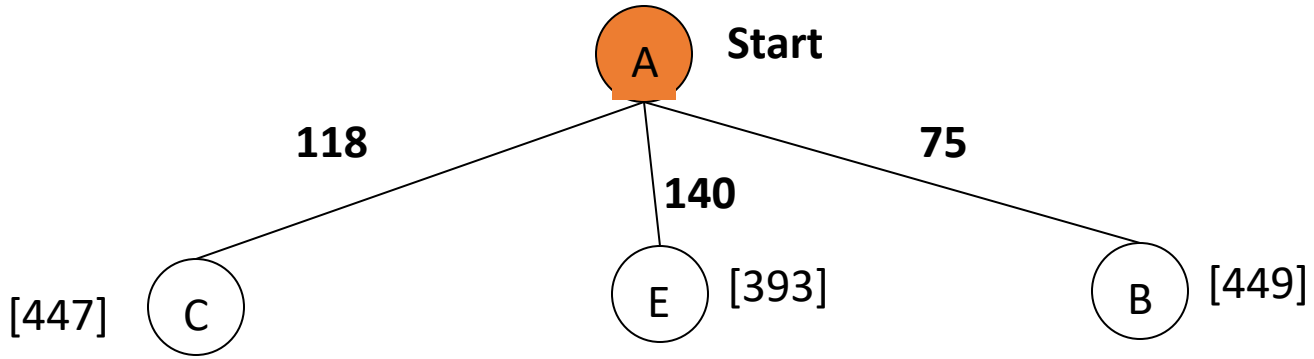
State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	**138>101**
I	0

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

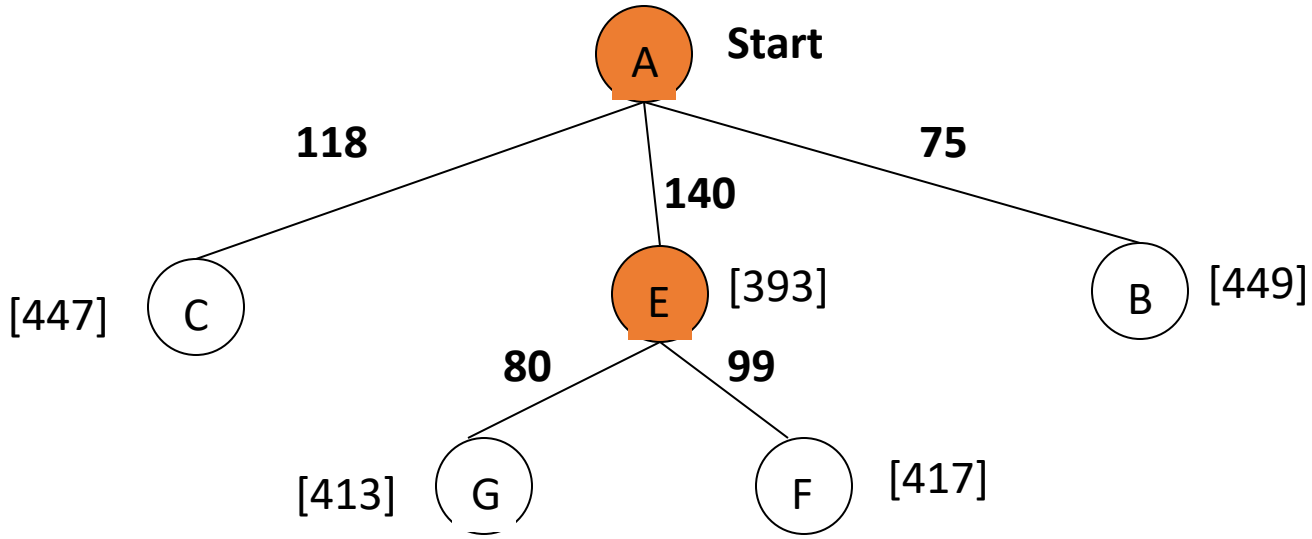
g(n): is the exact cost to reach node *n* from the initial state.

A* Search: Tree Search

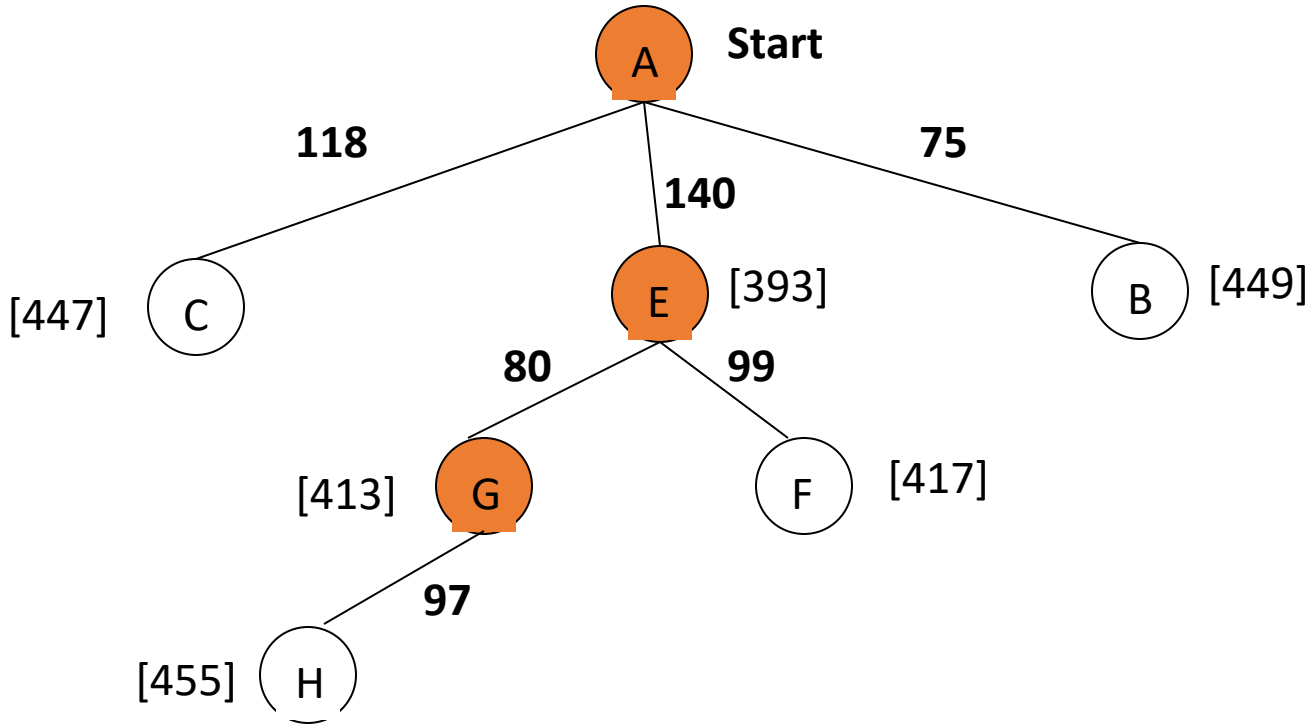
A* Search: Tree Search



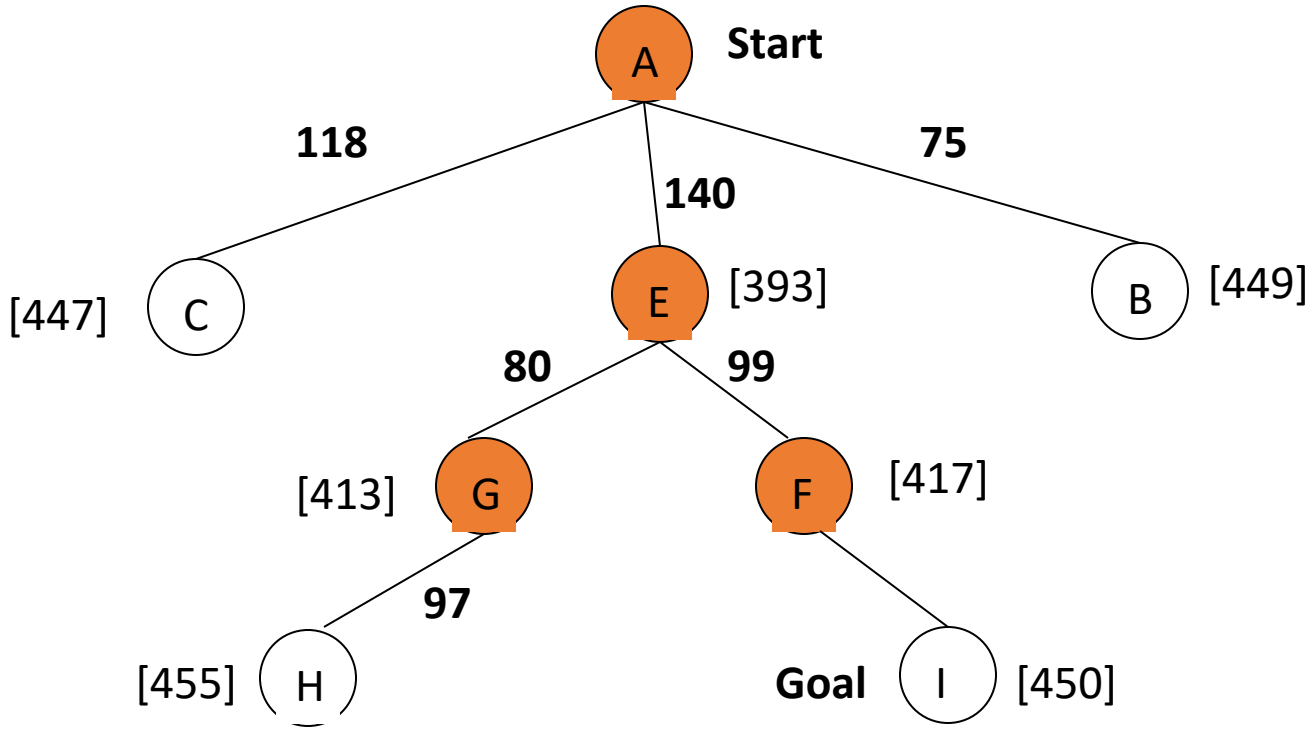
A* Search: Tree Search



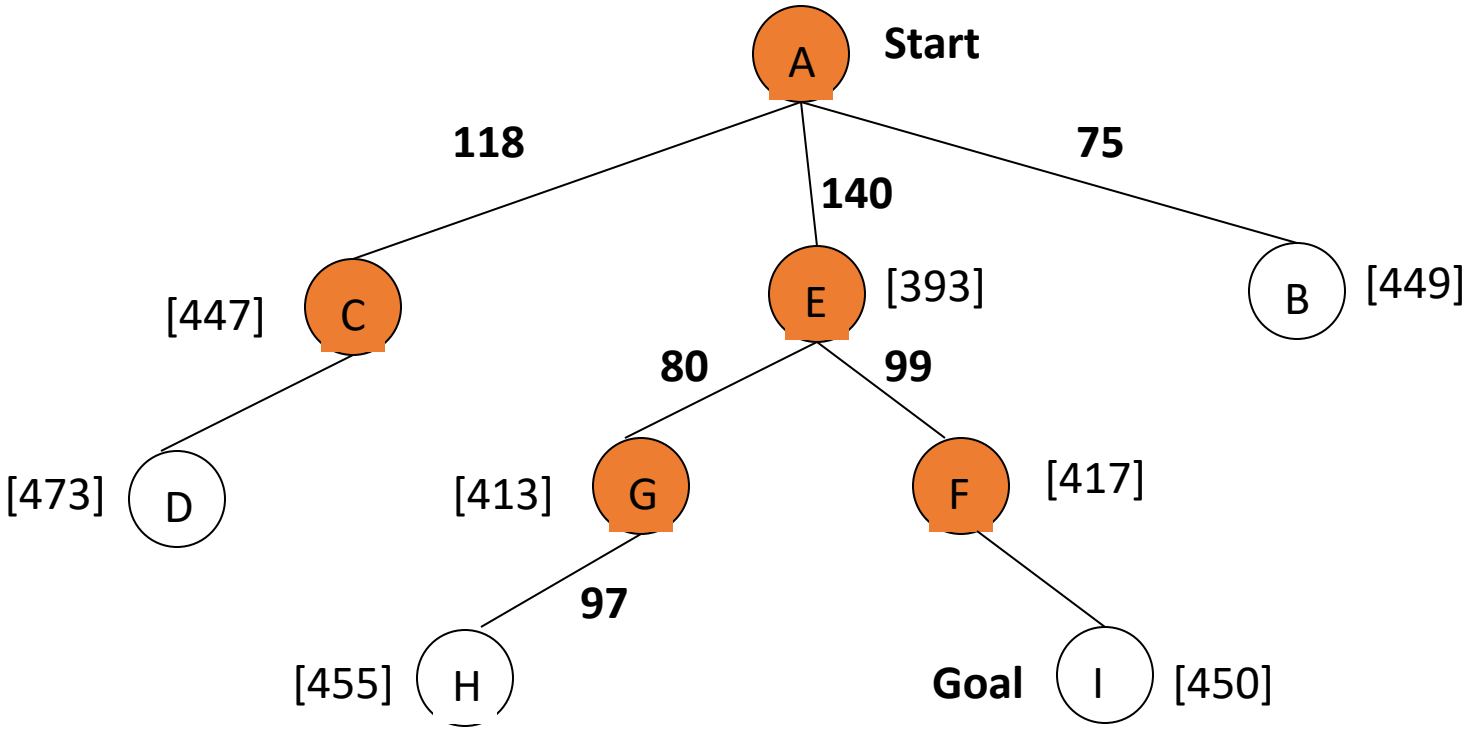
A* Search: Tree Search



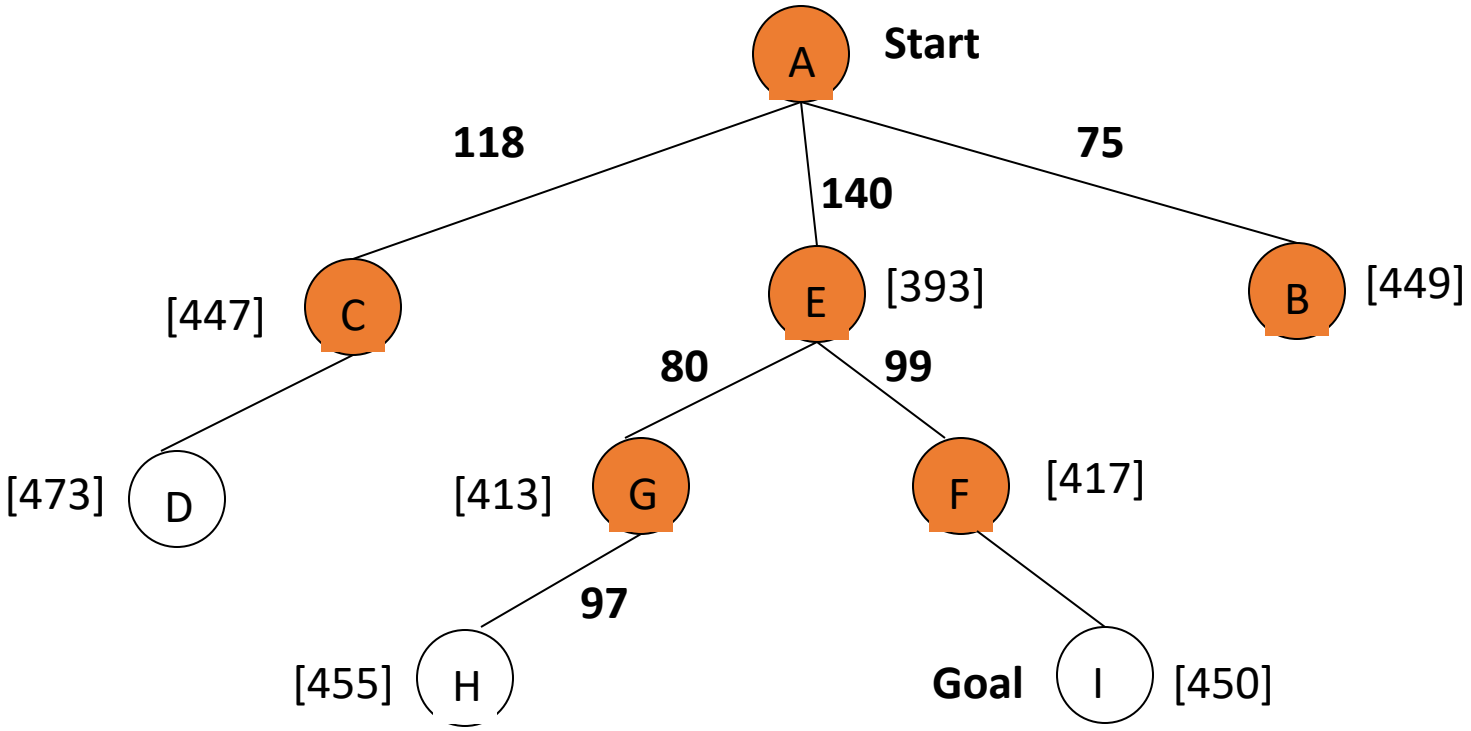
A* Search: Tree Search



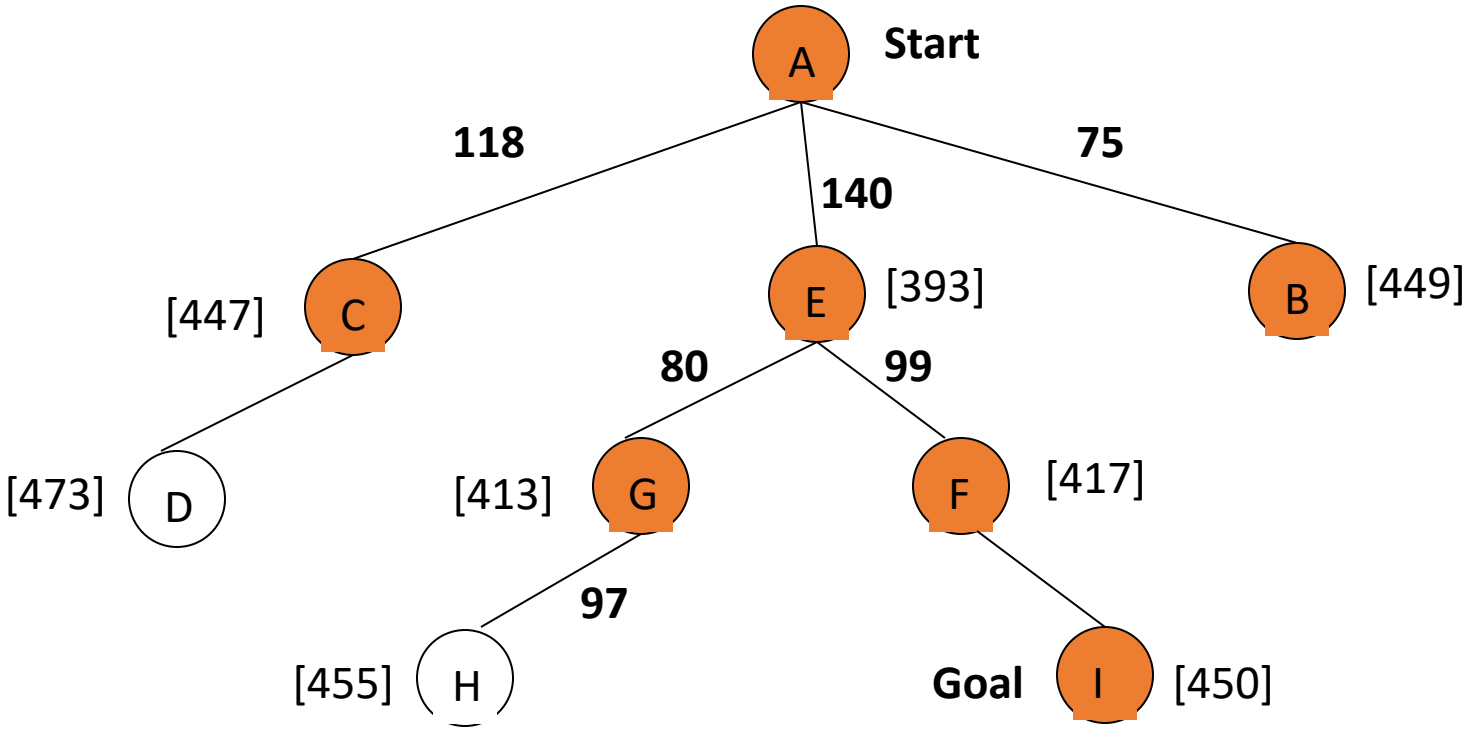
A* Search: Tree Search



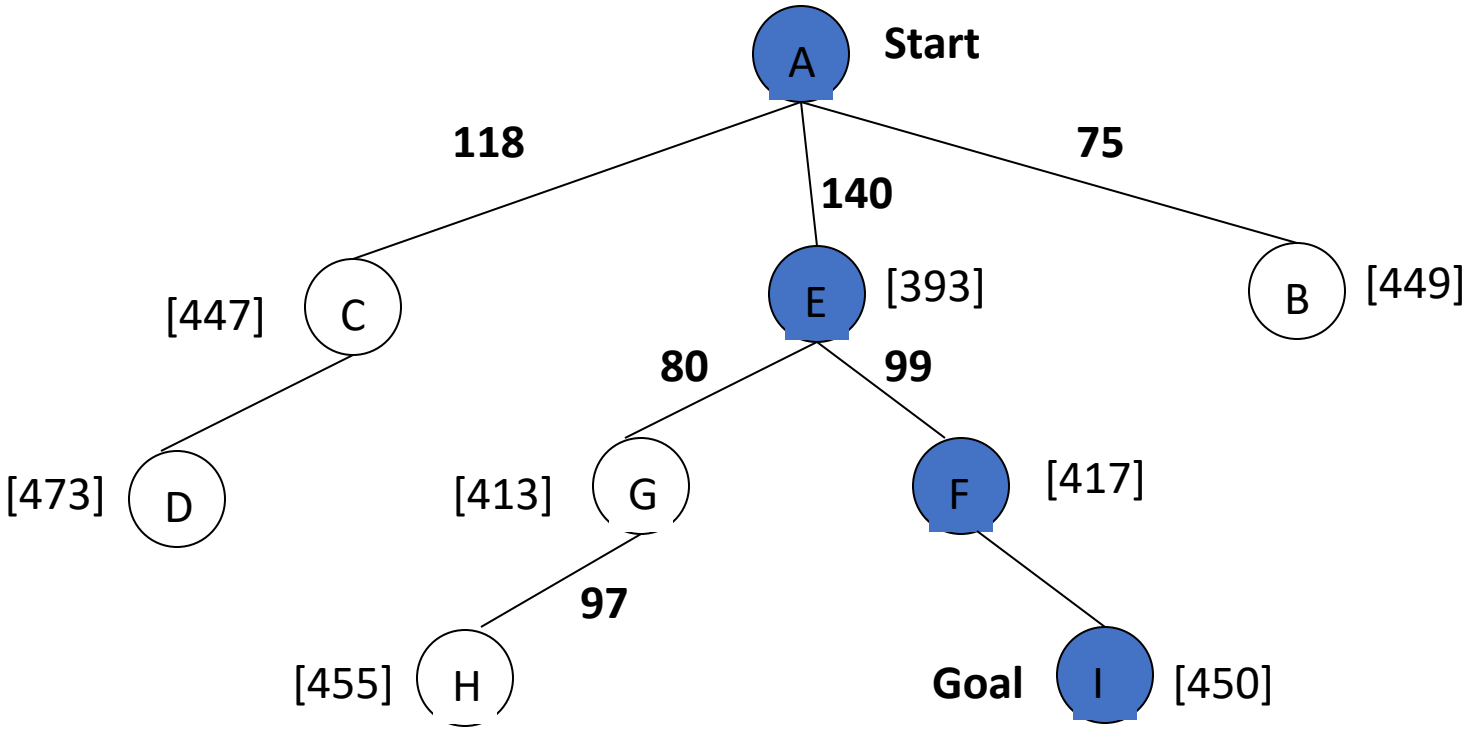
A* Search: Tree Search



A* Search: Tree Search



A* Search: Tree Search



A* not optimal !!!, **h not admissible**

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*).

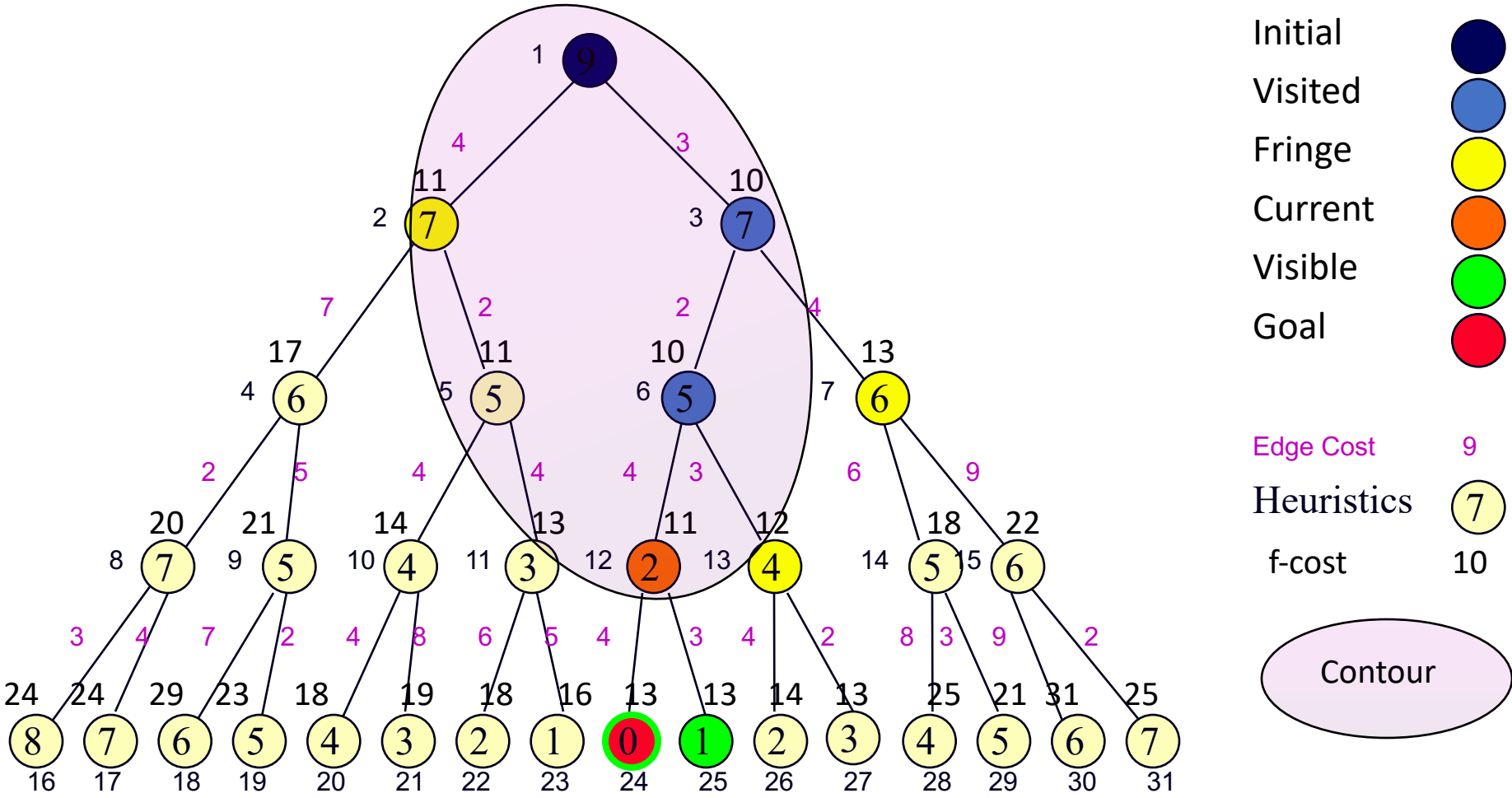
A* Properties

- with consistent heuristics, the first time we reach a state (add it to fringe) it will be on an optimal path. But with inconsistent heuristics, we may end up with multiple paths reaching the same state, and if the new path has a lower cost than the previous one, then we will end up with multiple nodes for that state in the frontier, costing us both time and space.

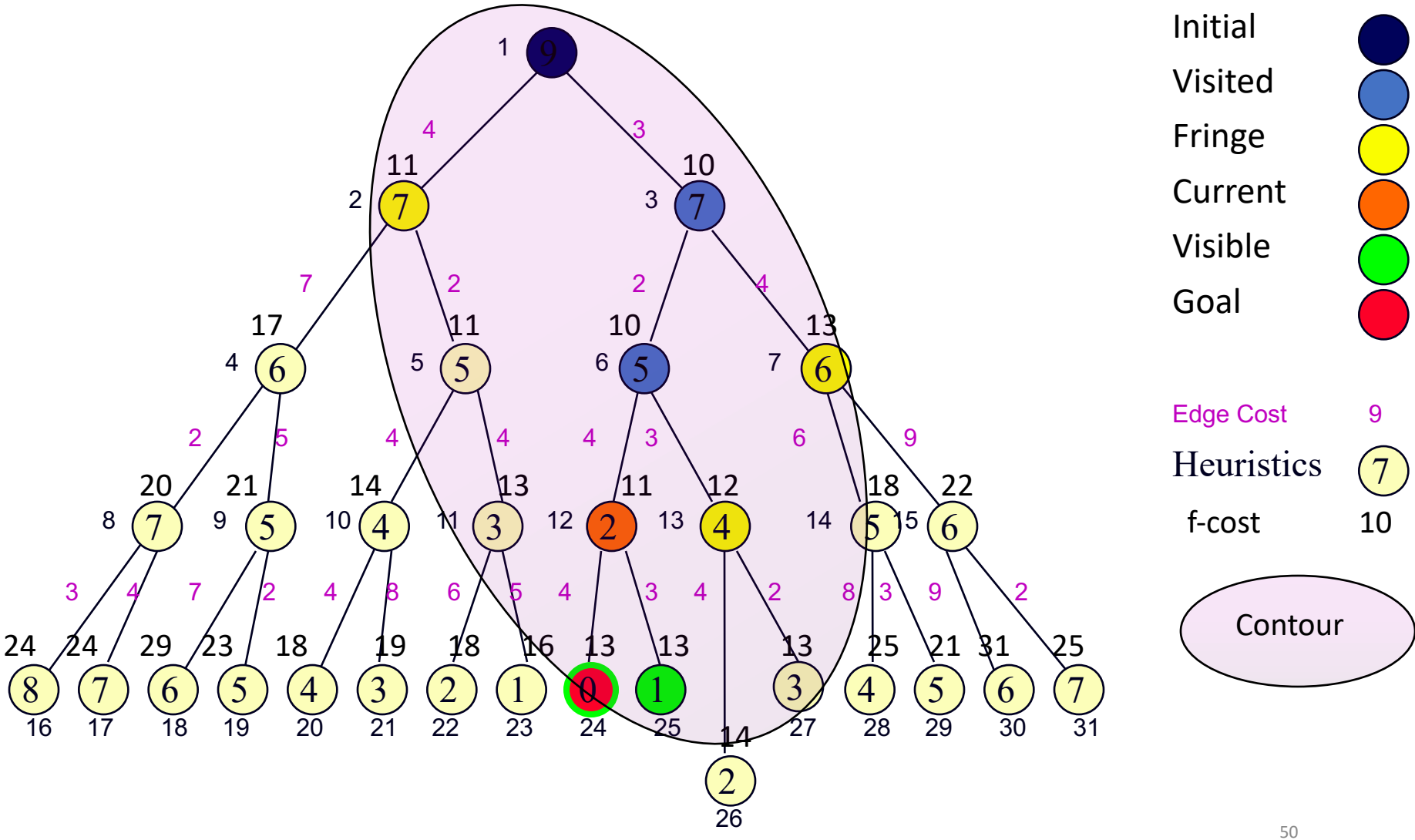
- the value of f never decreases along any path starting from the initial node
  - also known as monotonicity of the function
  - All consistent heuristics show monotonicity
  - those that don't can be modified through minor changes

- this property can be used to draw contours
  - regions where the f-cost is below a certain threshold
  - with uniform cost search (h = 0), the contours are circular
  - the better the heuristics h, the narrower the contour around the optimal path

A* Snapshot with Contour f=11



A* Snapshot with Contour f=13



Optimality of A*

- A* will find the optimal solution
  - the first solution found is the optimal one
- A* is optimally efficient
  - no other algorithm is guaranteed to expand fewer nodes than A*
- A* is not always “the best” algorithm
  - optimality refers to the expansion of nodes, other criteria might be more relevant
  - it generates and keeps all nodes in memory
    - improved in variations of A*

Complexity of A*

- the number of nodes within the goal contour search space is still exponential
  - with respect to the length of the solution
  - better than other algorithms, but still problematic

- frequently, space complexity is more severe than time complexity
  - A* keeps all generated nodes in memory

Properties of A*

- Complete?
  - Yes – unless there are infinitely many nodes with f(n) ≤ C*.
    (C* is the cost of the optimal solution)
- Optimal?
  - Yes
- Time?
  - Number of nodes for which f(n) ≤ C* (exponential)
- Space?
  - Exponential

Designing Heuristic Functions

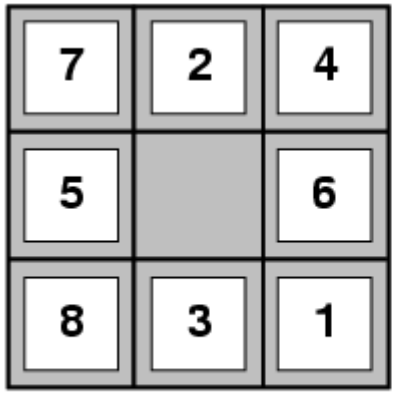
- Heuristics for the 8-puzzle

h1(n) = number of misplaced tiles

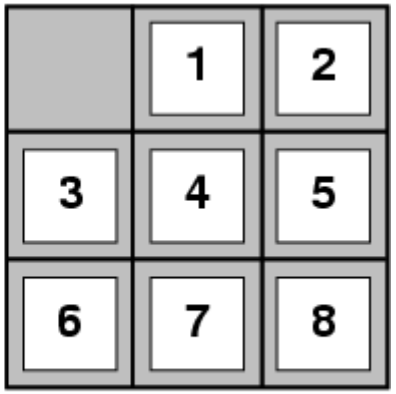
h2(n) = total Manhattan distance (number of squares from desired location of each tile)

h1(start) = 8

h2(start) = 3+1+2+2+2+3+3+2 = 18



Start State



Goal State

- Are h1 and h2 admissible?

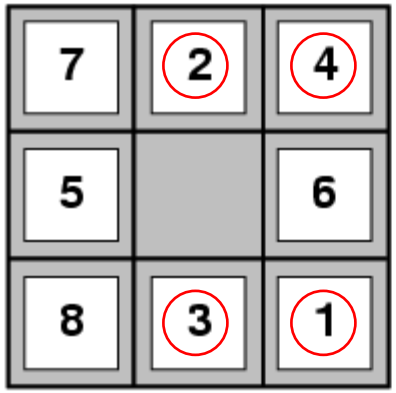
Heuristics from relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then h1(n) gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then h2(n) gives the shortest solution

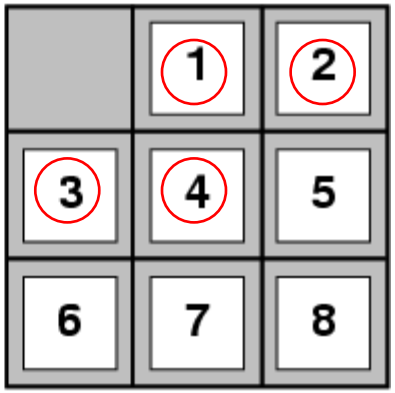
Heuristics from subproblems

- Let h3(n) be the cost of getting a subset of tiles (say, 1,2,3,4) into their correct positions

- Can precompute and save the exact solution cost for every possible subproblem instance – pattern database



Start State



Goal State

Dominance

- If h1 and h2 are both admissible heuristics and $h2(n) \geq h1(n)$ for all n, then h2 dominates h1

- Which one is better for search?

  - A* search expands every node with $f(n) < C^*$ or $h(n) < C^* - g(n)$

  - Therefore, A* search with h1 will expand more nodes

Dominance

- Typical search costs for the 8-puzzle (average number of nodes expanded for different solution depths): d:depth

- d=12 IDS = 3,644,035 nodes
  A*(h1) = 227 nodes
  A*(h2) = 73 nodes

- d=24 IDS ≈ 54,000,000,000 nodes
  A*(h1) = 39,135 nodes
  A*(h2) = 1,641 nodes

Combining heuristics

- Suppose we have a collection of admissible heuristics h1(n), h2(n), …, hm(n), but none of them dominates the others

- How can we combine them?

$$h(n) = \max\{h_1(n), h_2(n), \ldots, h_m(n)\} \quad ???$$

Memory-Bounded Search

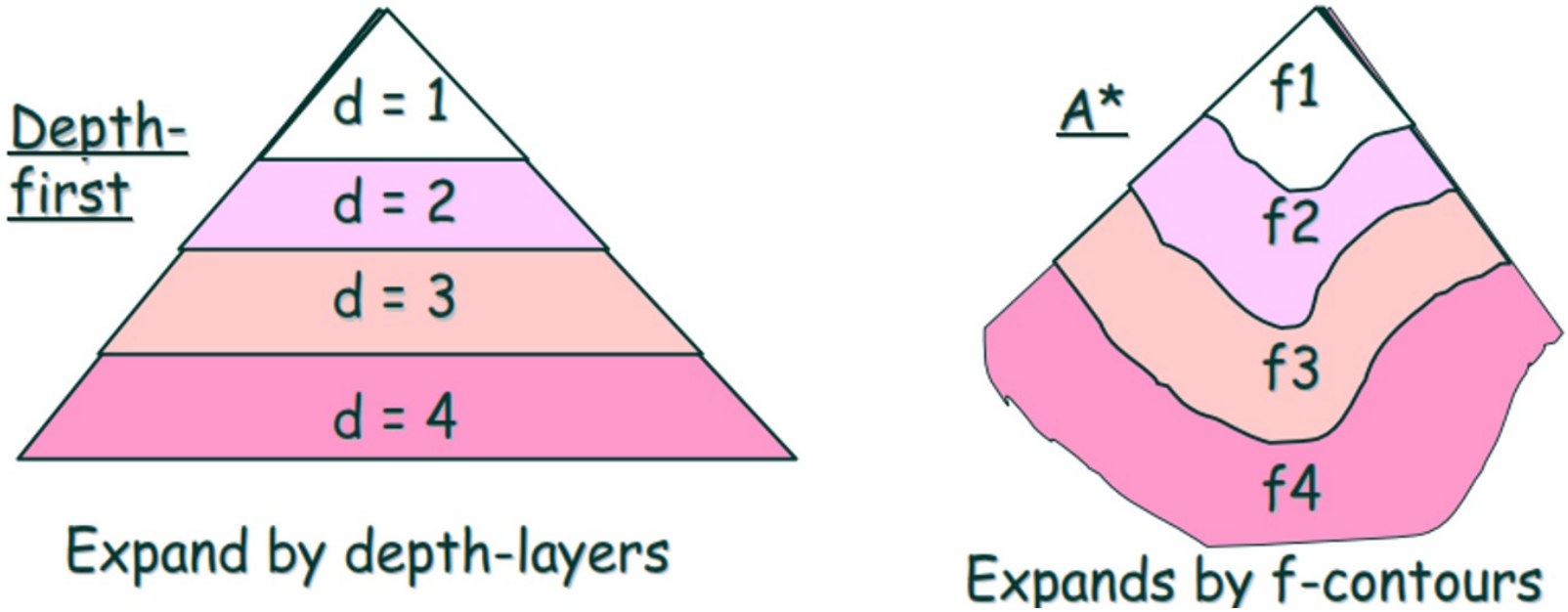
- search algorithms that try to conserve memory
- most are modifications of A*
  - iterative deepening A* (IDA*)
  - Recursive best-first search, simplified memory-bounded A* (SMA*)
    - Forget some subtrees but remember the best f-value in these subtrees and regenerate them later if necessary
- Problems: memory-bounded strategies can be complicated to implement, suffer from “thrashing”: keep needing things thought irrelevant!

Iterative Deepening A* (IDA*)

- IDA* is to A* what iterative-deepening search is to depth- first.
- IDA* gives us the benefits of A* without the requirement to keep all reached states in memory, at a cost of visiting some states multiple times.
- IDA* is commonly used for problems that do not fit in memory.
- In standard iterative deepening the cutoff is the depth, which is increased by one each iteration. In IDA* the cutoff is the f-cost (g+h); at each iteration, the cutoff value is the smallest f-cost of any node that exceeded the cutoff on the previous iteration.
- In other words, each iteration exhaustively searches an f-contour, finds a node just beyond that contour, and uses that node's f-cost as the next contour.

Iterative Deepening A* (IDA*)

- IDA* is similar to iterative-deepening search.



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.

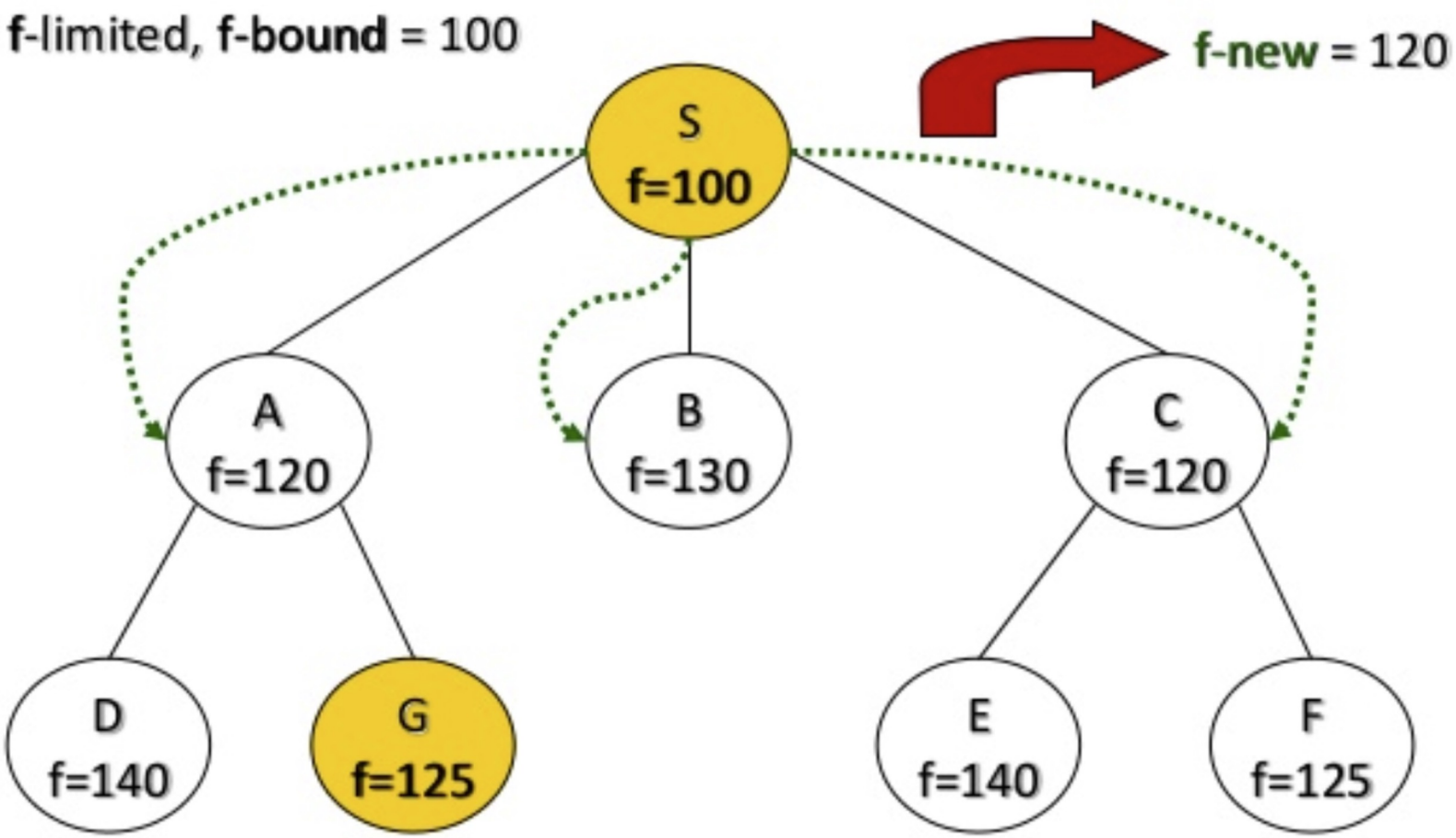
- We expand nodes using the depth-first algorithm and backtrack whenever f(n) for an expanded node n exceeds the cut-off value.

- If this search does not succeed, determine the lowest f-value among the nodes that were visited but not expanded.

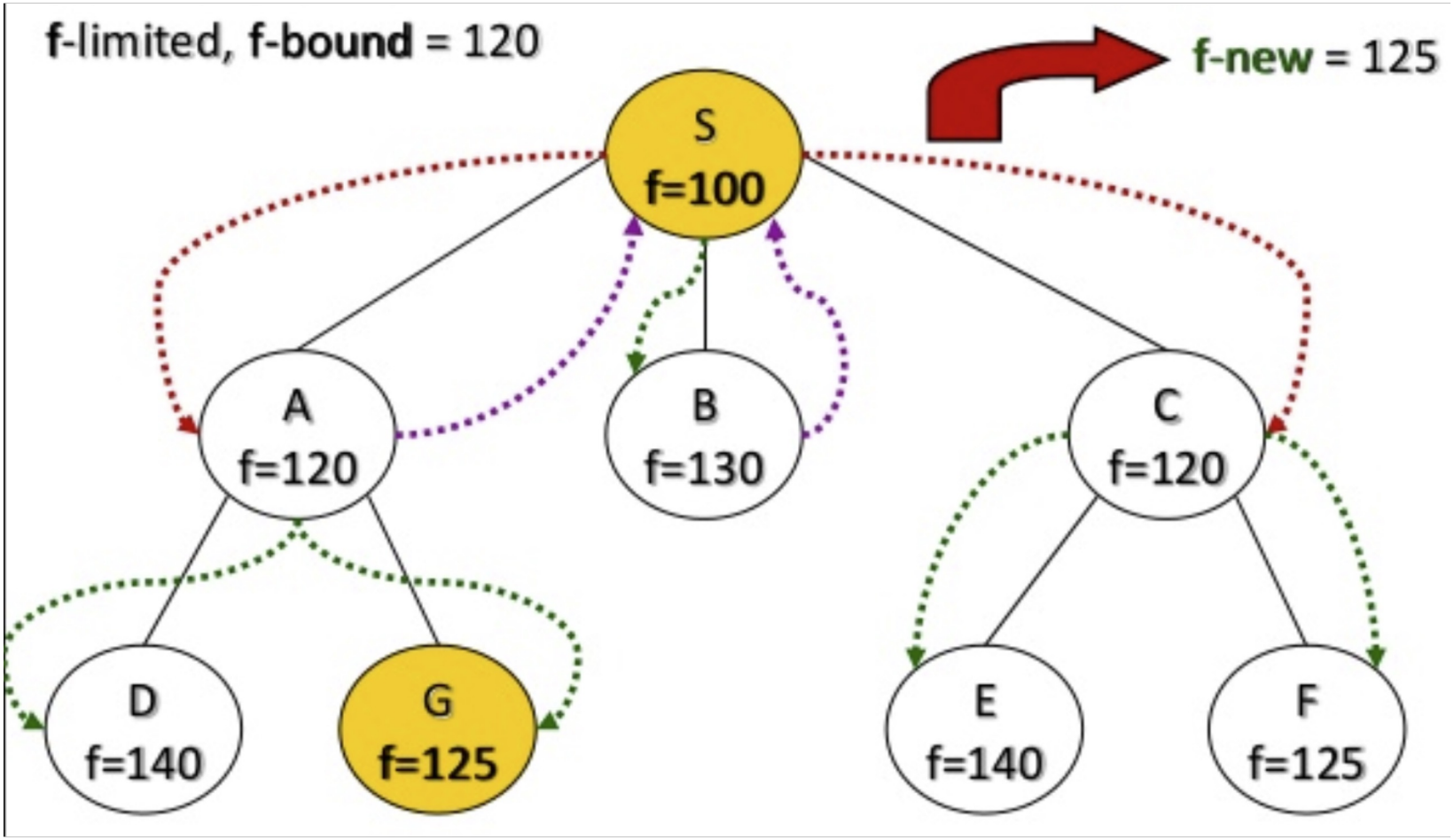
- Use this f-value as the new limit value – cut-off value and do another depth-first search.

- Repeat this procedure until a goal node is found.

IDA* Example



IDA* Example



IDA* Example