CS 561 Artificial Intelligence
Lecture

SOLVING PROBLEMS BY SEARCHING
INFORMED SEARCH

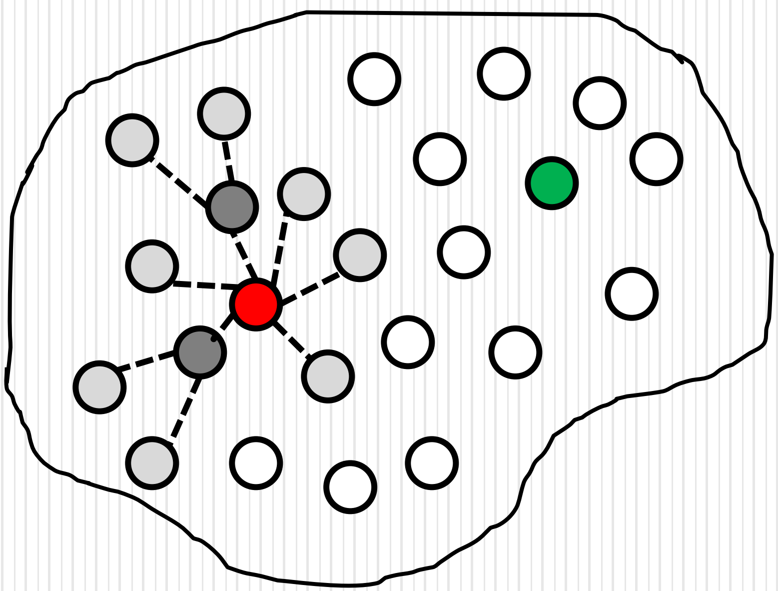
Rashmi Dutta Baruah
Department of Computer Science & Engineering
IIT Guwahati

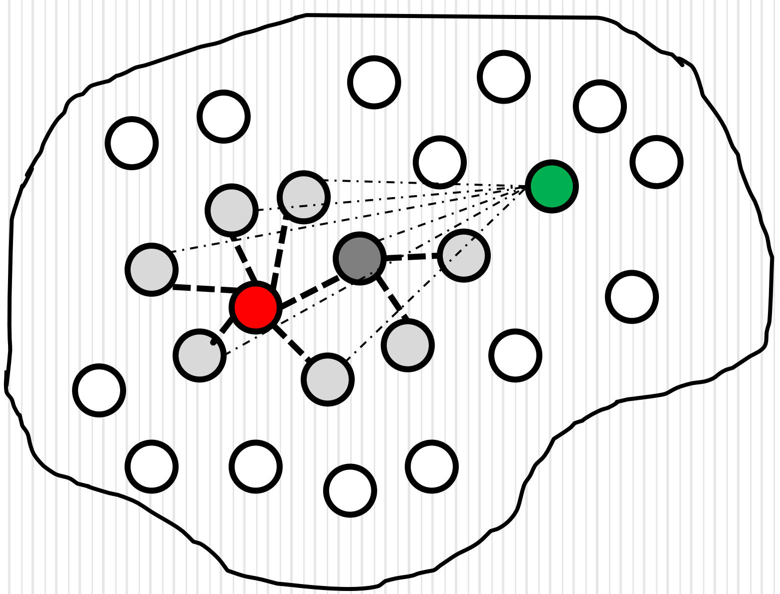
OUTLINE

- Informed (Heuristic) Search
  - Greedy Best-first Search
  - A* Search
  - Iterative Deepening A* (IDA*)
  - Recursive Best First Search (RBFS)
- More about heuristics
- Applications

INFORMED SEARCH STRATEGIES

Uses problem-specific knowledge beyond the definition of the problem itself





BEST-FIRST SEARCH

- Idea: use an evaluation function f(n) for each node
  - f(n) provides an estimate for the total cost.

  → Expand the node n with smallest f(n).

  → Choice of f determines the search strategy
- Implementation:
  - Order the nodes in frontier by f(n) (lowest f(n) first)
- Special cases:
  - greedy best-first search
  - A* search

Evaluation function is an estimate of node quality.

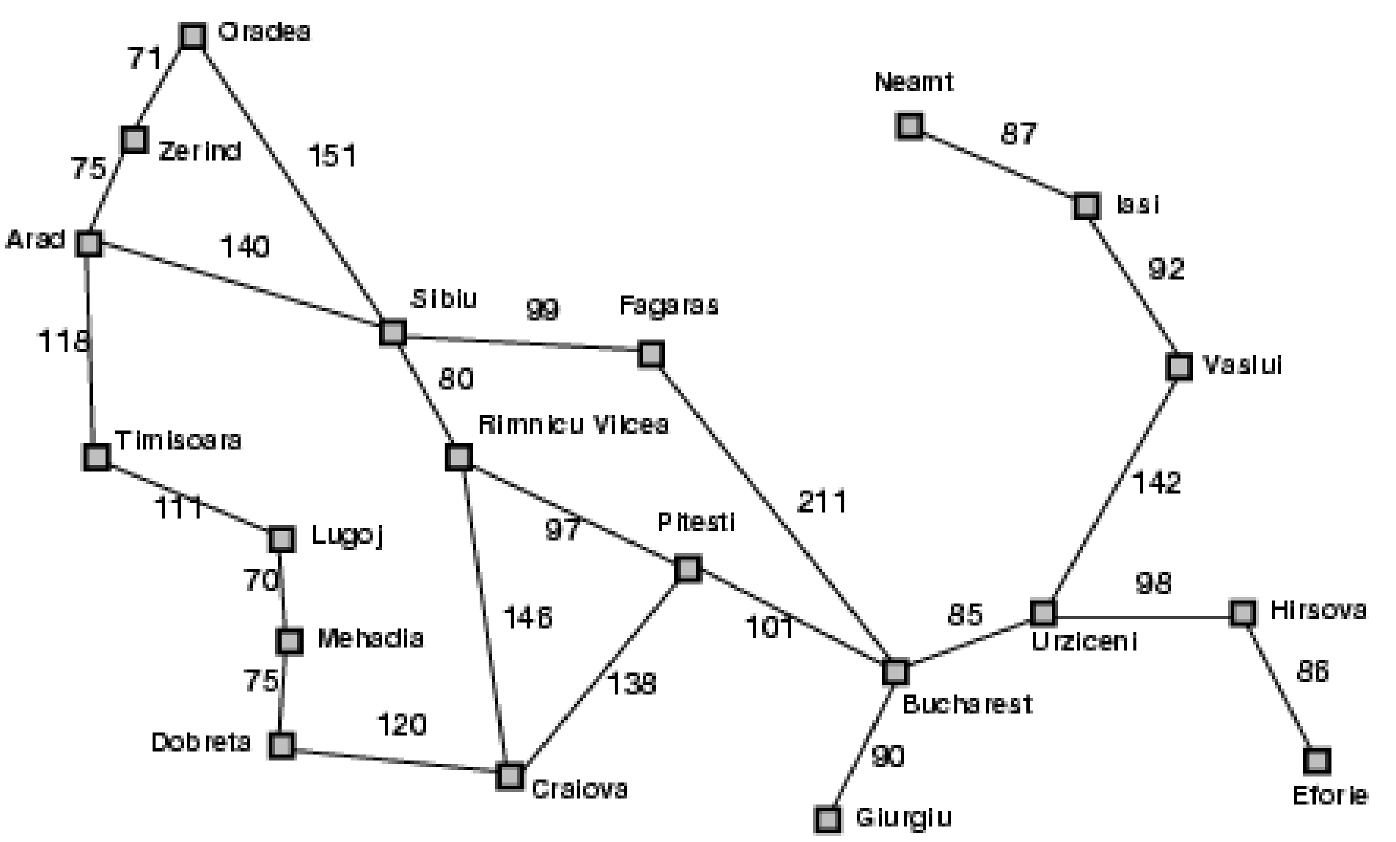
HEURISTIC FUNCTION

- f(n) consists of a heuristic function, denoted by h(n)
- Heuristic: "using rules of thumb to find answers"
- Heuristic function h(n)
  - Non-negative and problem-specific function
  - Estimate of (optimal) cost from n to goal
  - h(n) = 0 if n is a goal node

GREEDY BEST-FIRST SEARCH

- Special case of best-first search
  - Uses h(n) = heuristic function as its evaluation function
    - Example: straight line distance from n to Bucharest.
- Expand the node that appears closest to goal

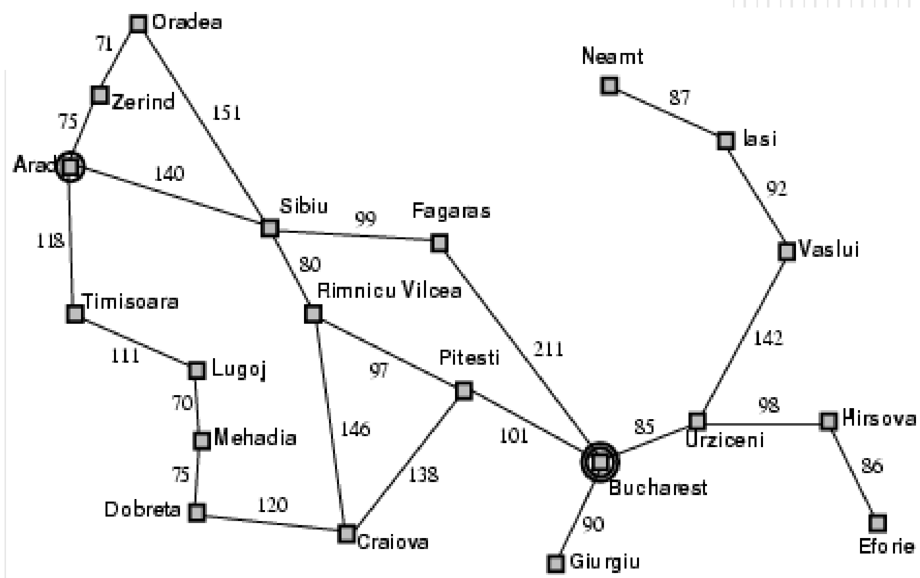
ROMANIA EXAMPLE



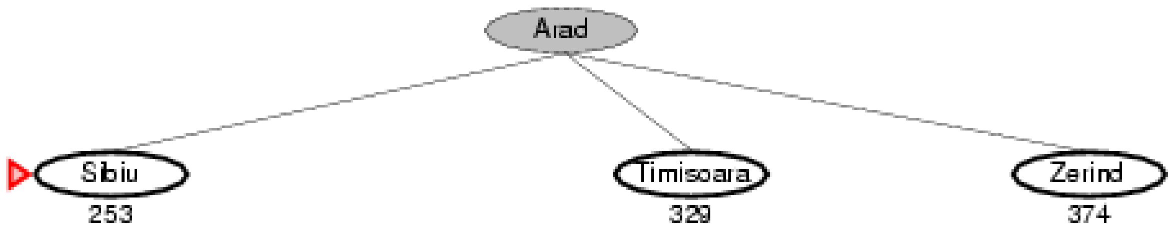
Straight−line distance to Bucharest

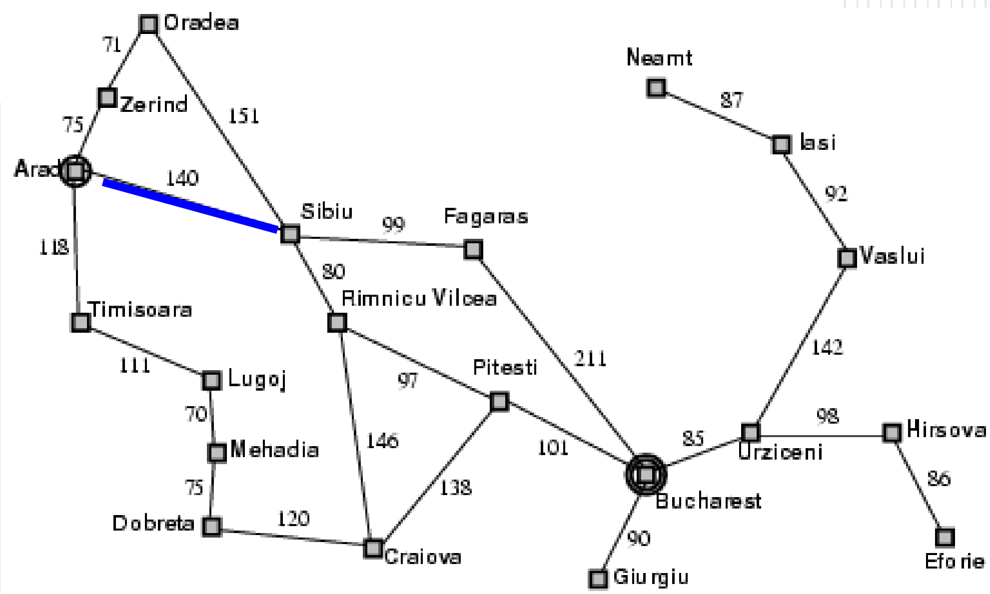
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

GREEDY BEST-FIRST SEARCH EXAMPLE

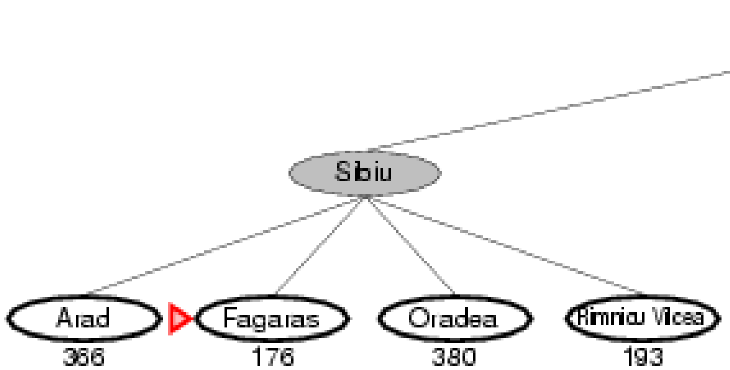


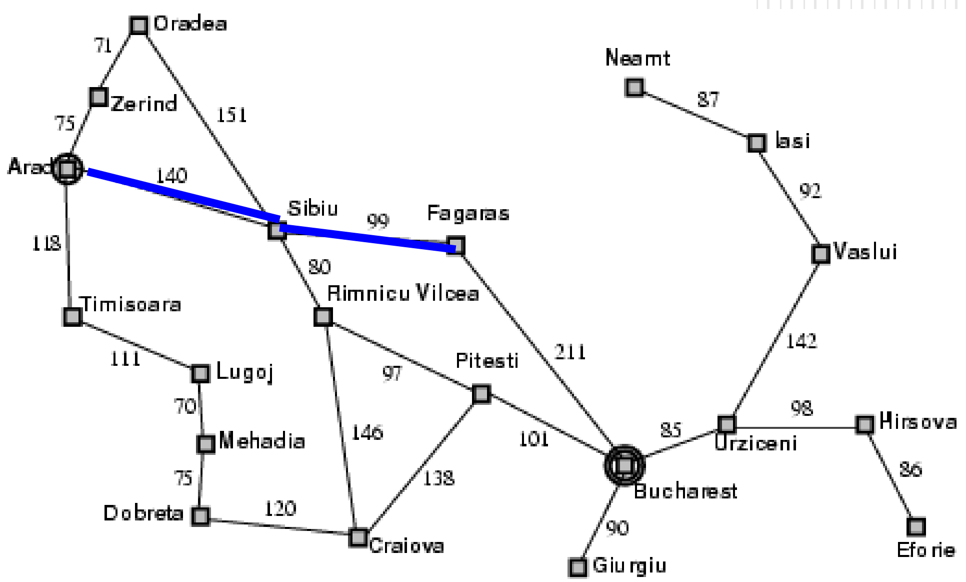
GREEDY BEST-FIRST SEARCH EXAMPLE



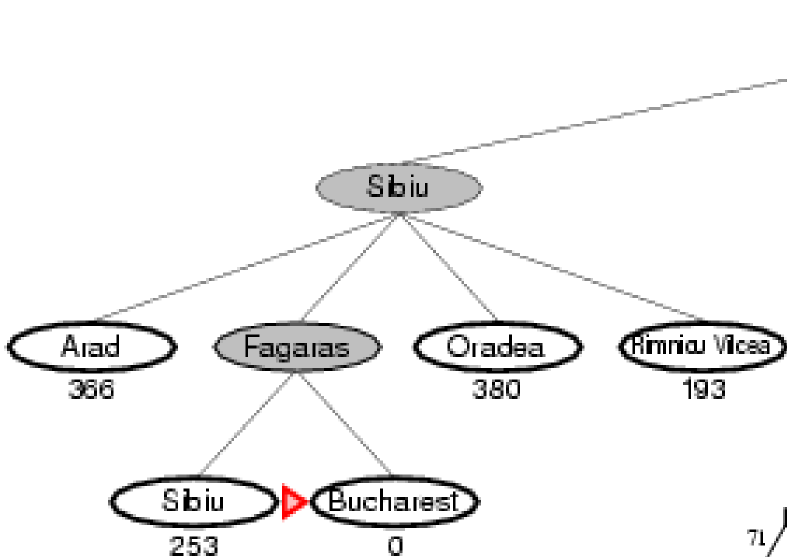


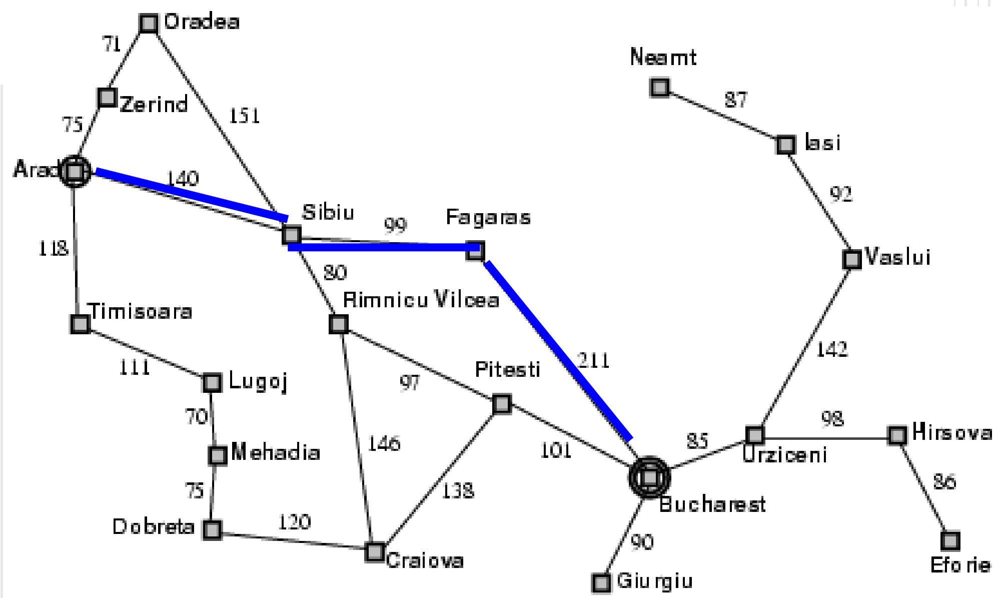
GREEDY BEST-FIRST SEARCH EXAMPLE



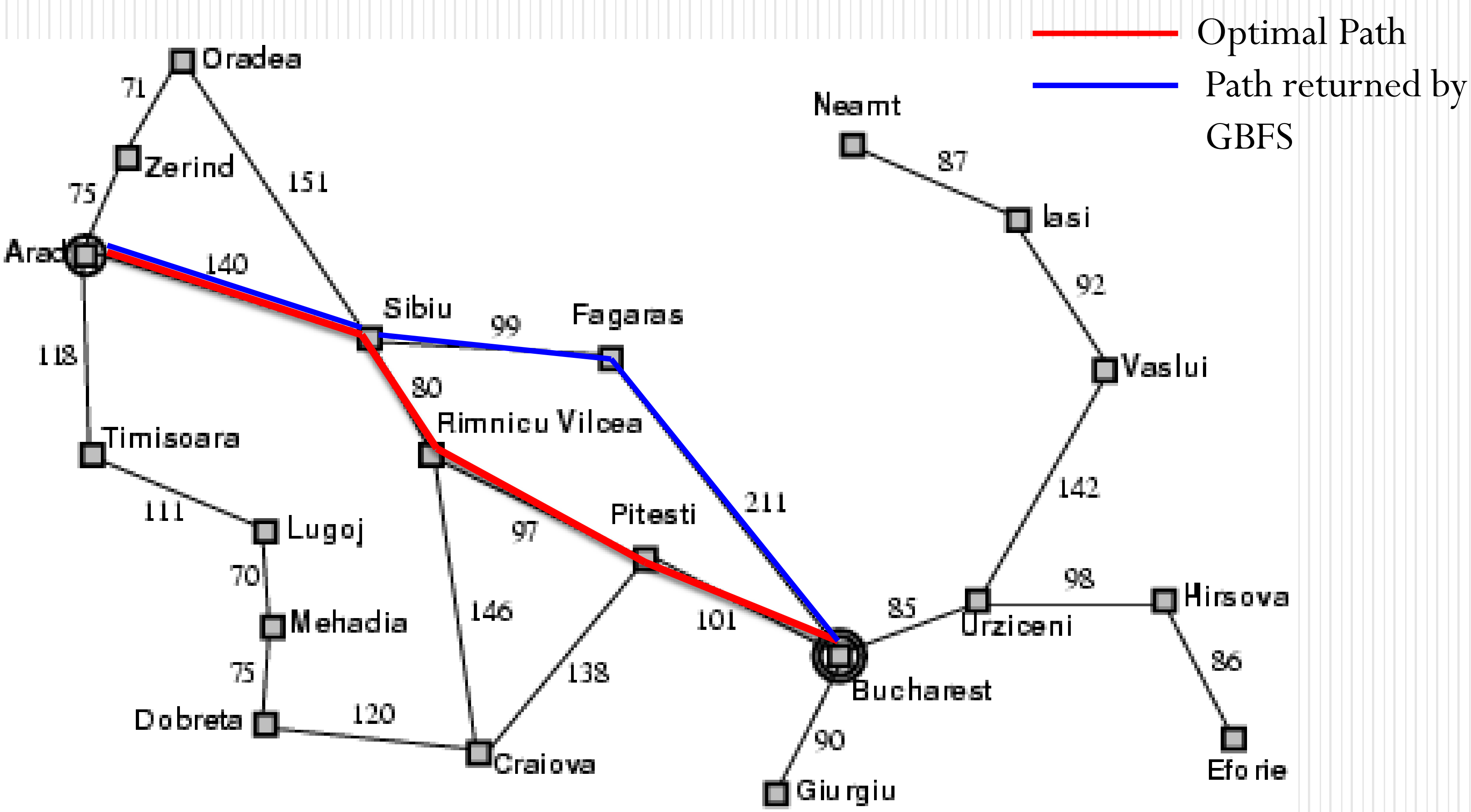


GREEDY BEST-FIRST SEARCH EXAMPLE



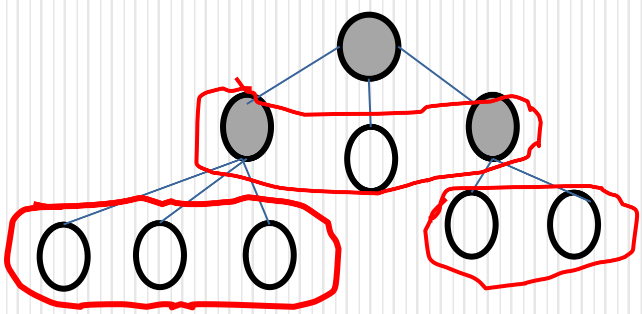


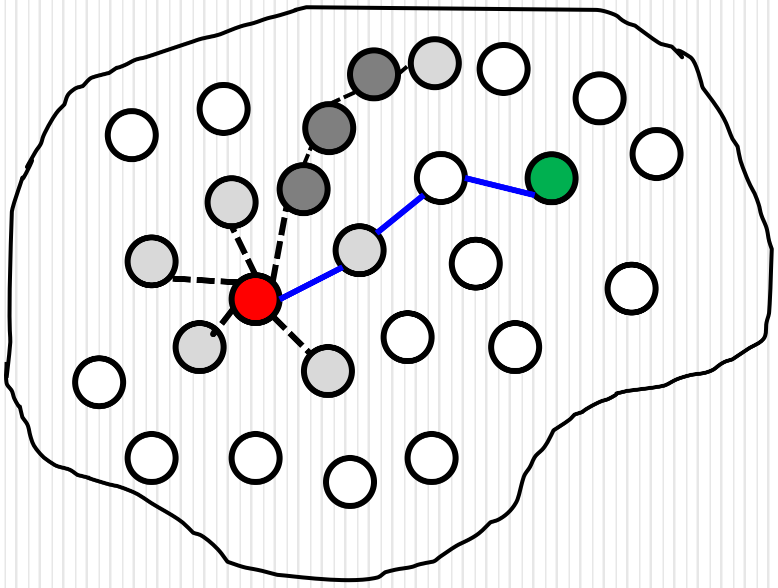
Optimal Path



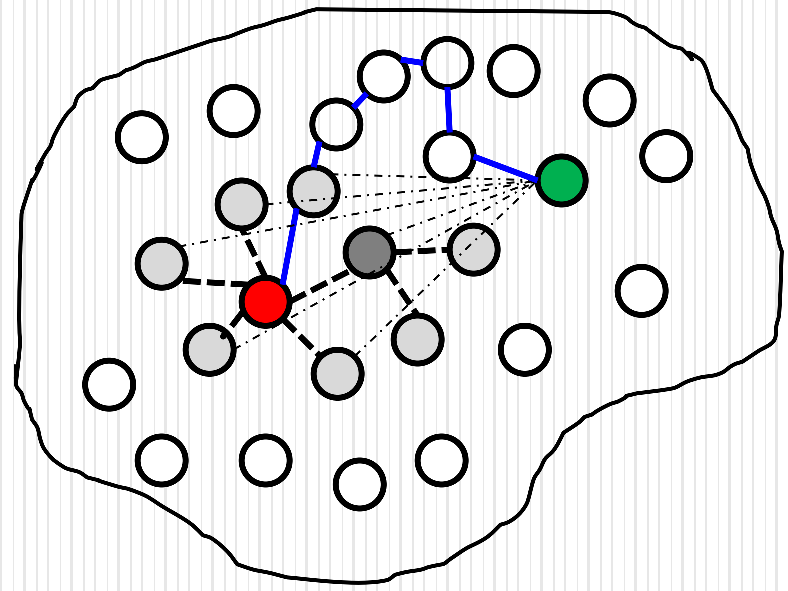
PROPERTIES OF GREEDY BEST-FIRST SEARCH

- Complete?
  - Not unless it keeps track of all states visited
- Optimal?
  - No – we just saw an example of a shorter path
- Time?
  - $O(b^m)$, can generate all nodes at depth m before finding solution
  - m = maximum depth of search space
- Space?
  - $O(b^m)$ – keeps all nodes in memory





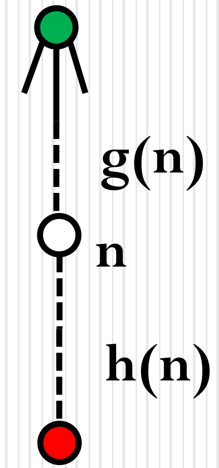
Uniform cost search $f(n) = g(n)$



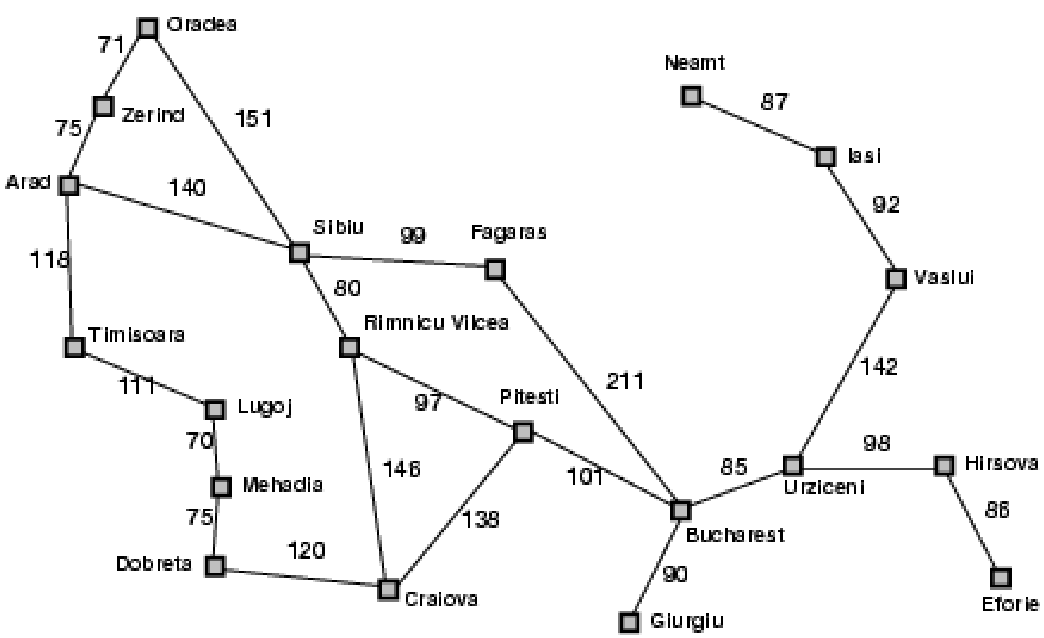
Greedy Best First Search $f(n) = h(n)$

A* SEARCH

- Expand node based on estimate of total path cost through node
- 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
- Greedy Best First search has $f(n)=h(n)$
- Uniform Cost search has $f(n)=g(n)$



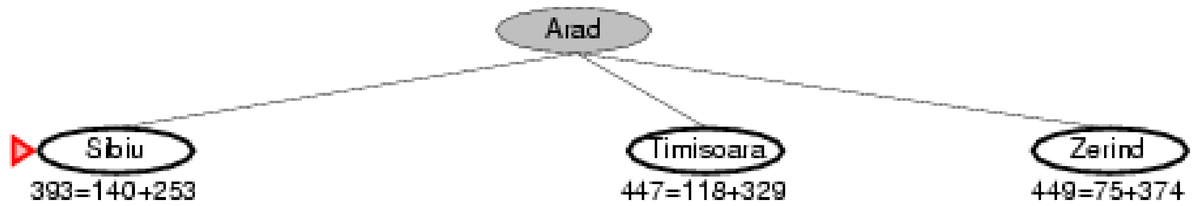
A* SEARCH EXAMPLE

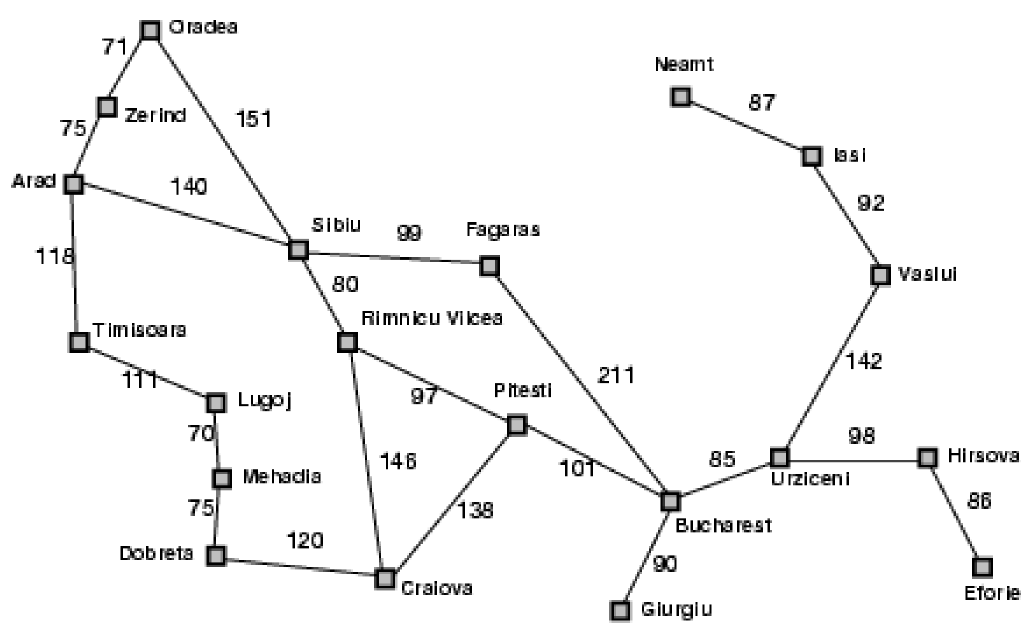


Straight–line distance to Bucharest

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

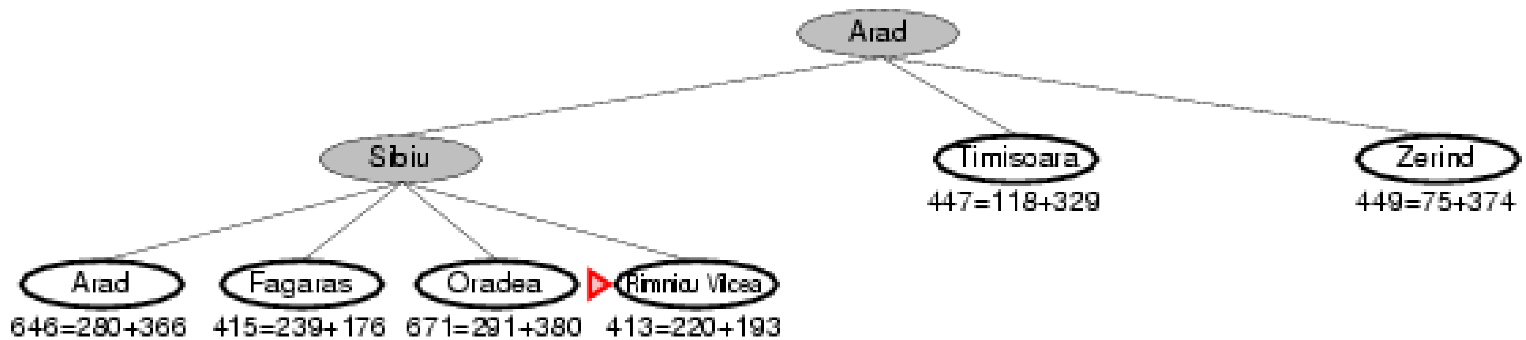


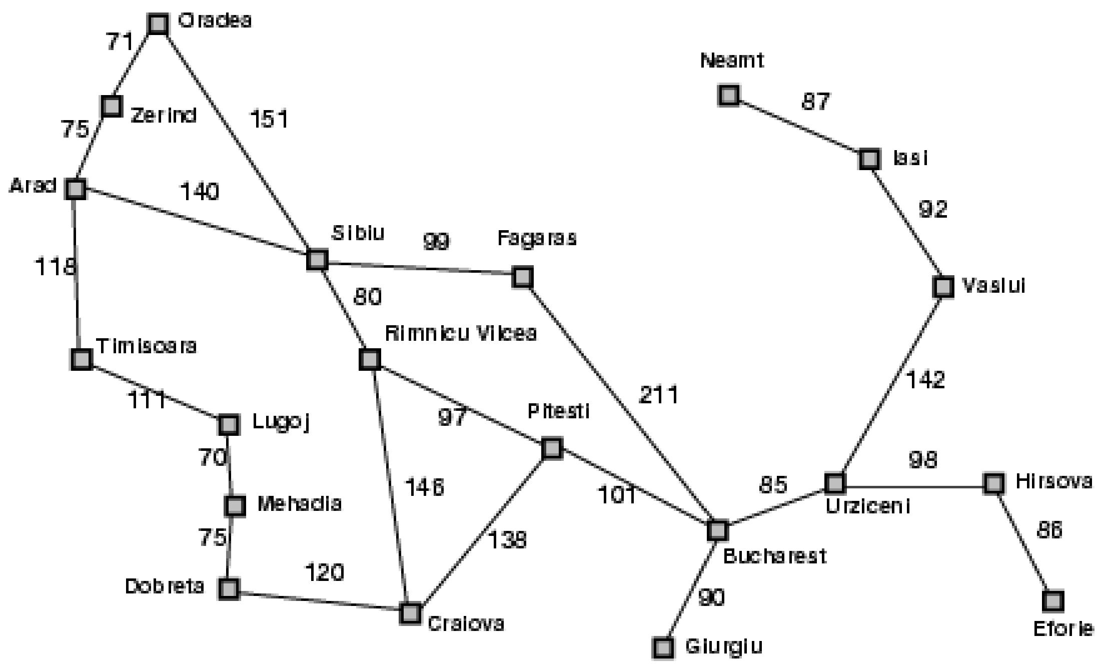


Straight–line distance to Bucharest

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

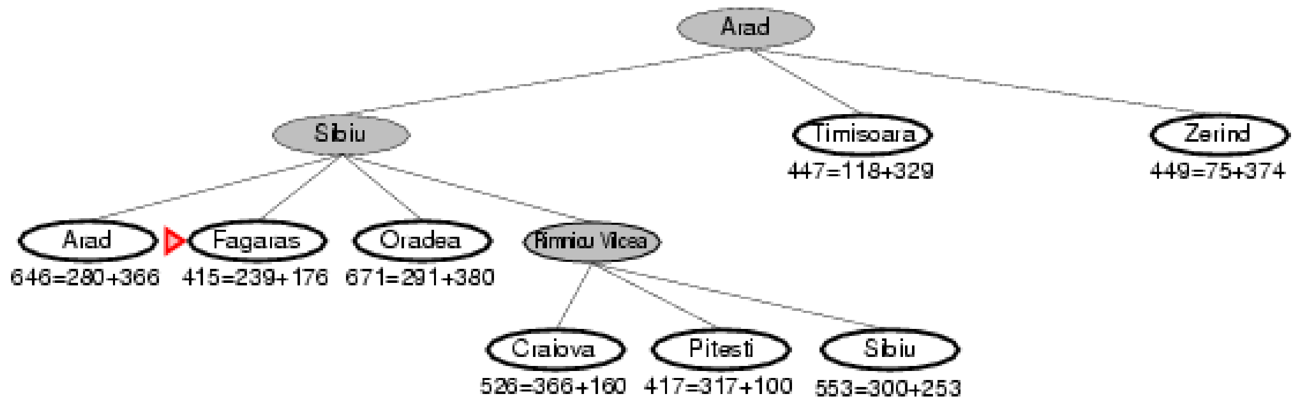


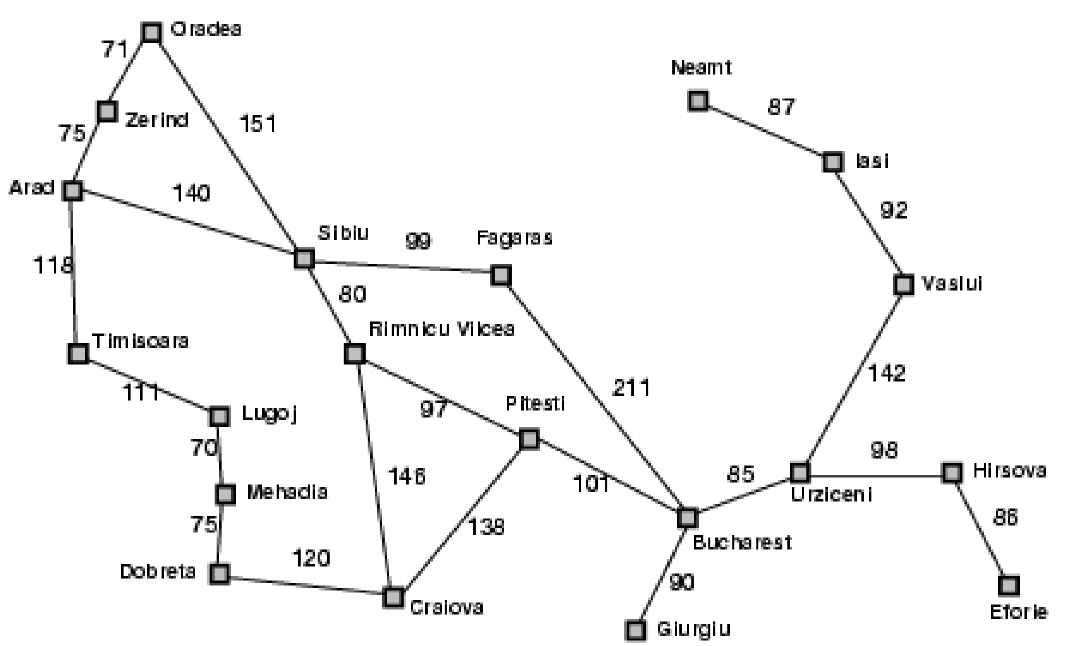


Straight–line distance to Bucharest

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

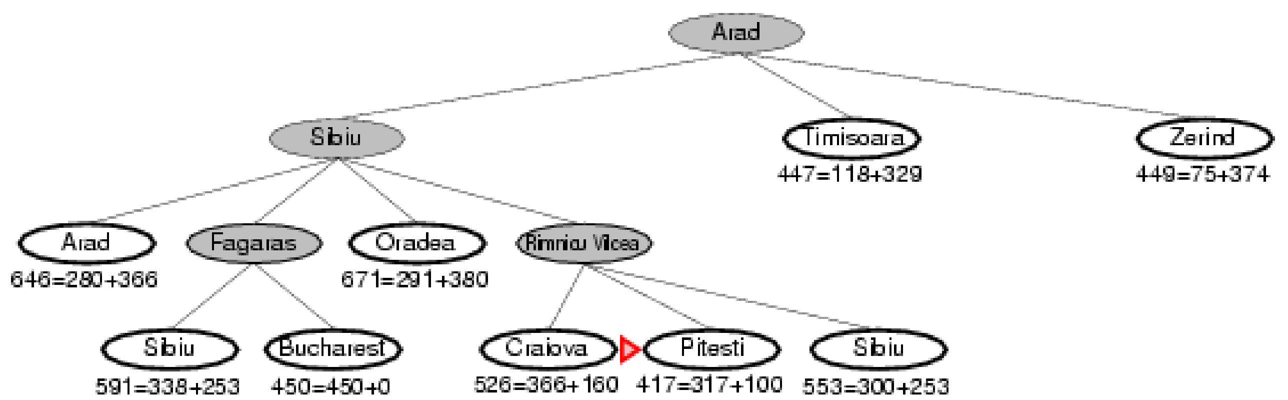
A* SEARCH EXAMPLE

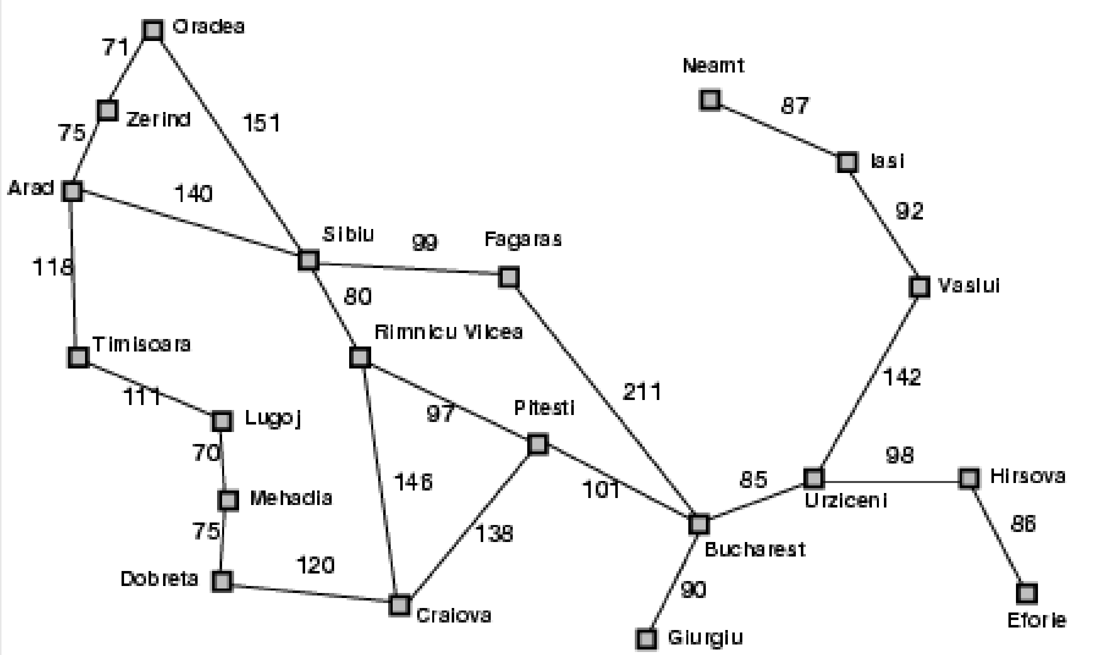




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

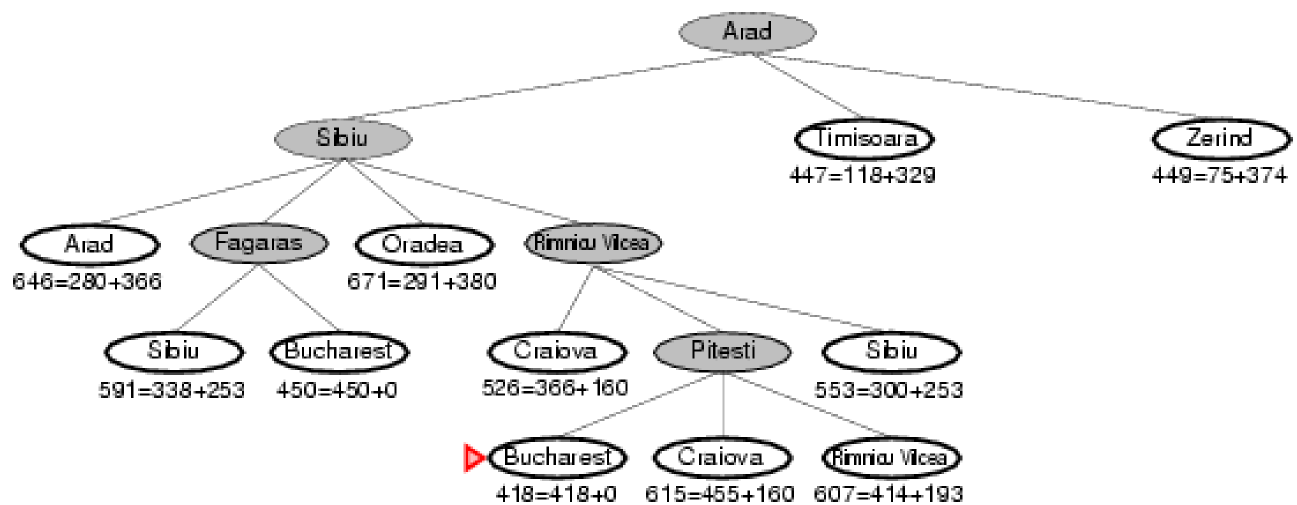
A* SEARCH EXAMPLE

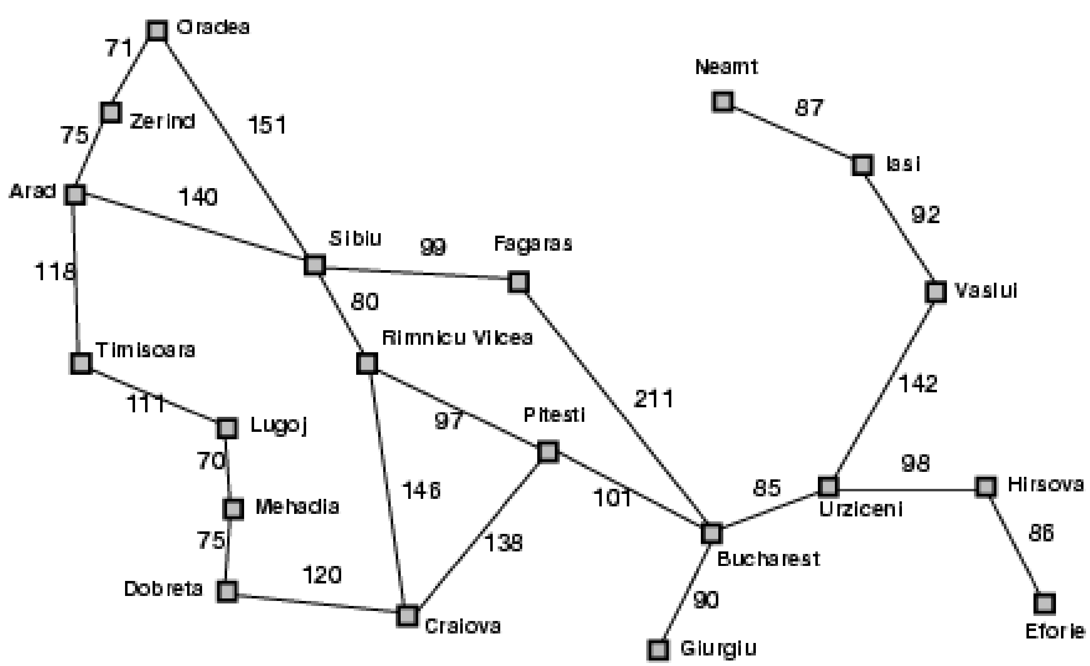




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

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

- A* search is both complete (finds a solution if one exists) and optimal (finds the optimal path to a goal) if:
  - the branching factor is finite
  - Step cost are $> \epsilon > 0$
  - h(n) satisfies the conditions of
    - Admissibility
    - Consistency

ADMISSIBLE HEURISTICS

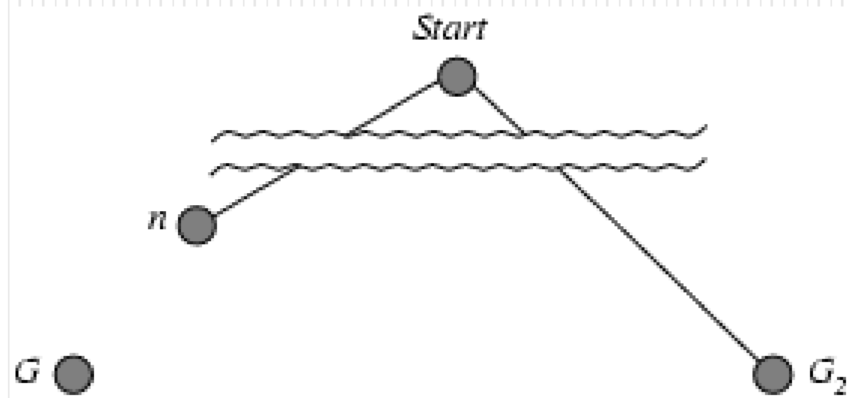
- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)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

  If $h(n)$ is admissible, A^* using `TREE-SEARCH` is optimal

OPTIMALITY OF A* (PROOF)

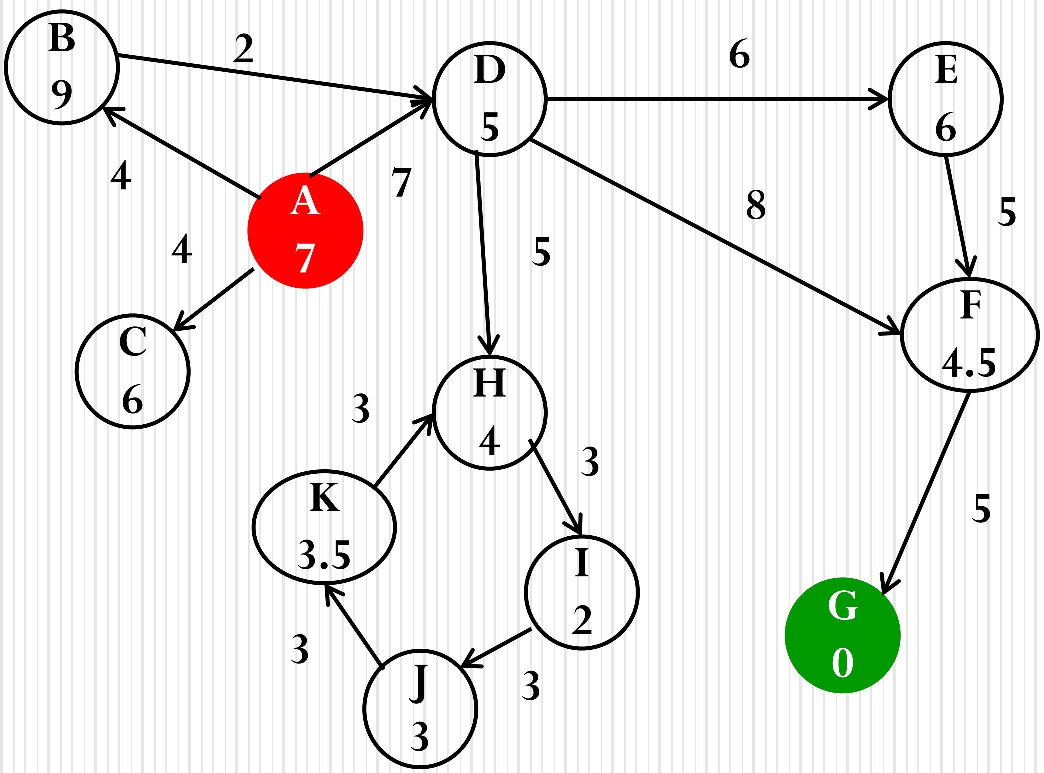
- Suppose some suboptimal goal G_2 has been generated and is in the frontier. Let n be an unexpanded node in the frontier such that n is on a shortest path to an optimal goal G.

We want to prove:
f(n) < f(G2)
(then A* will prefer n over G2)



- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $f(G) = g(G)$ since $h(G) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G_2) > f(G)$ from above
- $h(n) \leq h^*(n)$ since h is admissible (*under*-estimate)
- $g(n) + h(n) \leq g(n) + h^*(n)$ from above
- $f(n) \leq f(G)$ since $g(n)+h(n)=f(n)$ & $g(n)+h^*(n)=f(G)$
- $f(n) < f(G2)$ since $f(G) < f(G_2)$

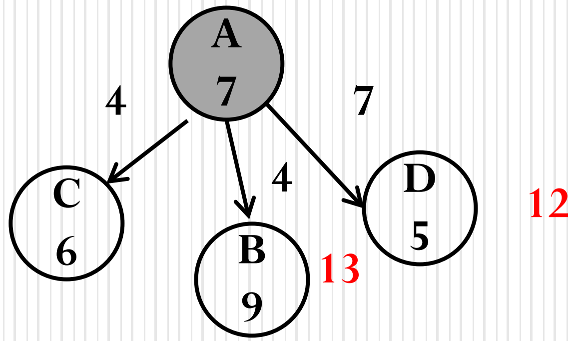
A * Example



A * Example

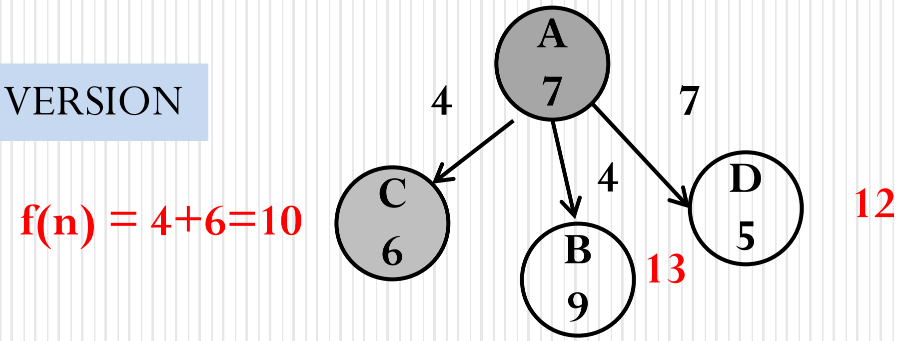
TREE-SEARCH VERSION

f(n) = 4+6=10



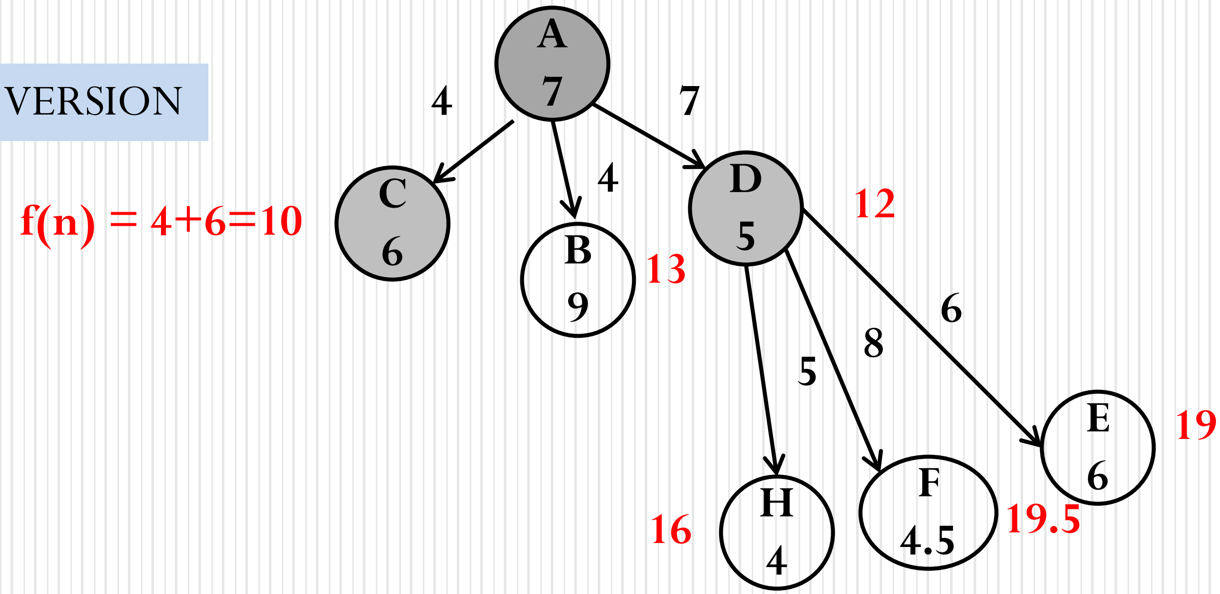
A * Example

TREE-SEARCH VERSION



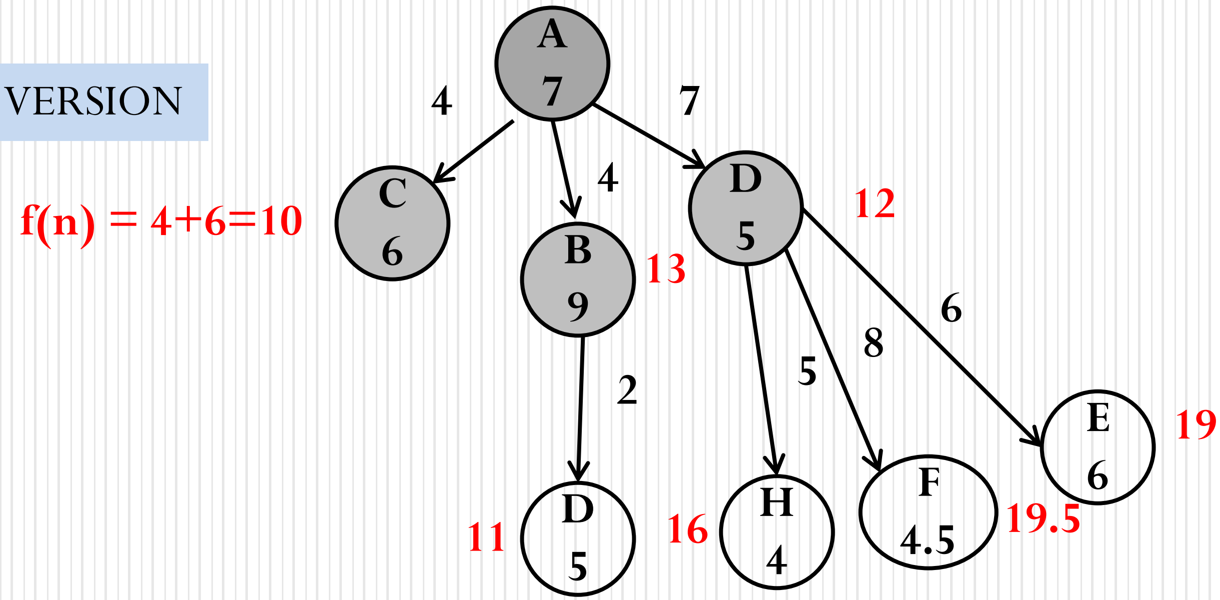
A * Example

TREE-SEARCH VERSION



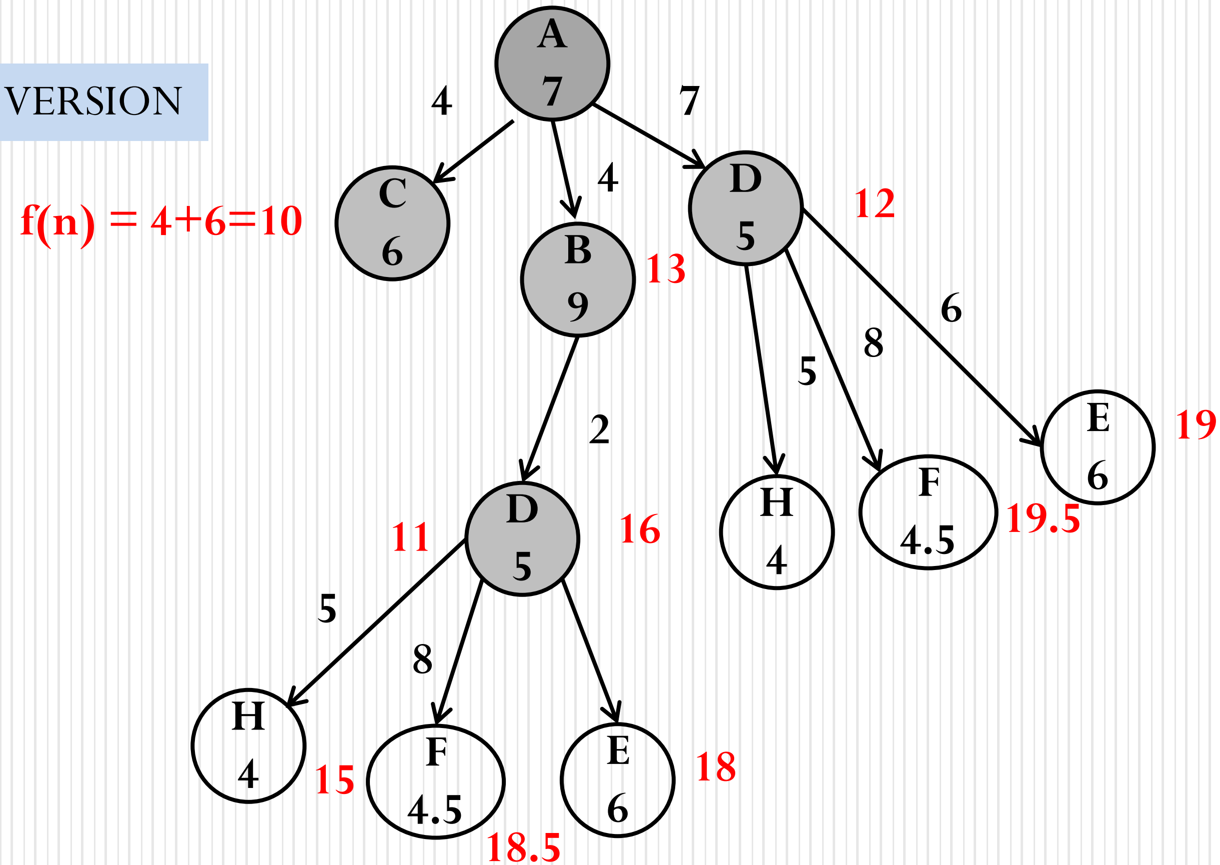
A * Example

TREE-SEARCH VERSION



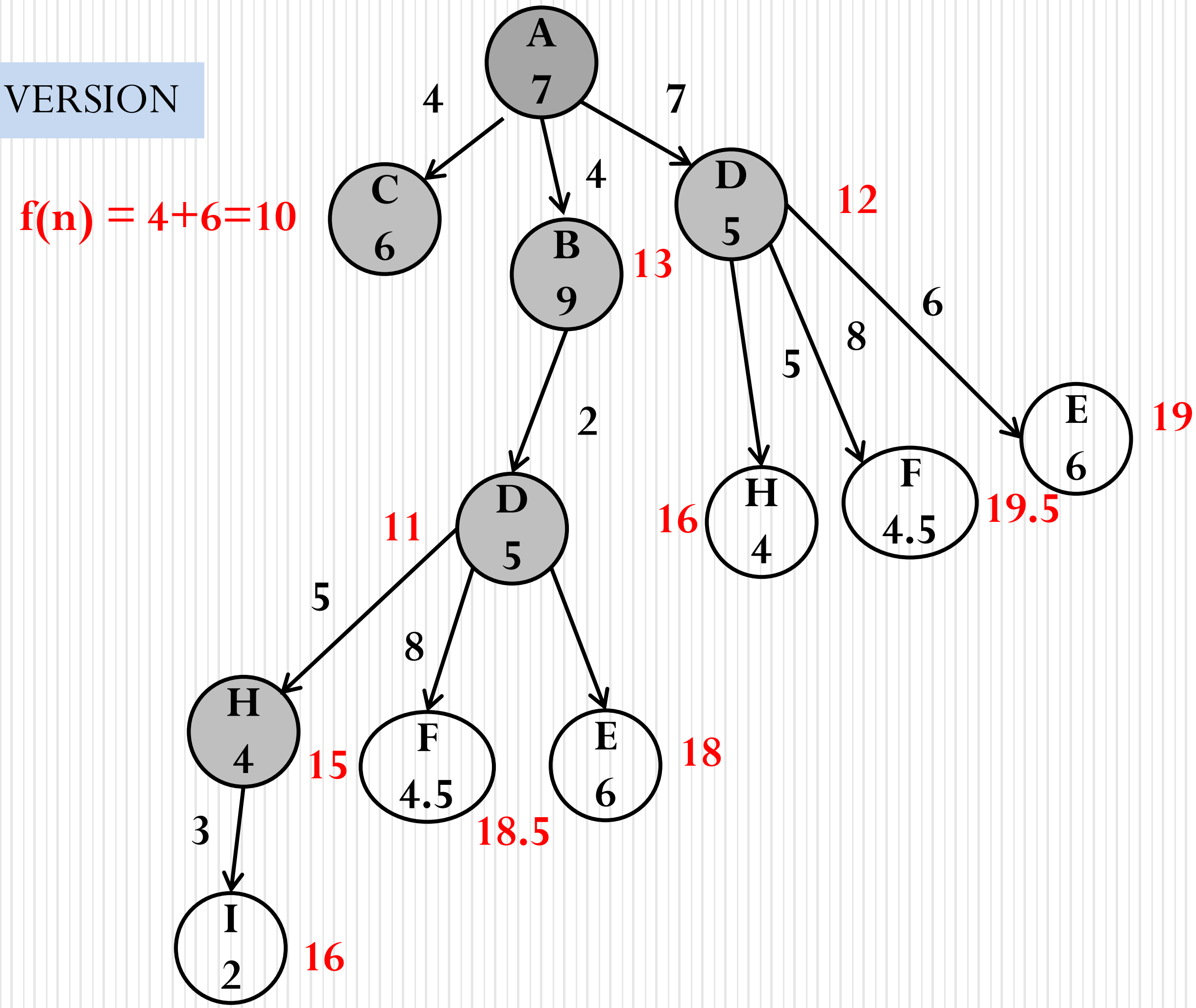
A * Example

TREE-SEARCH VERSION



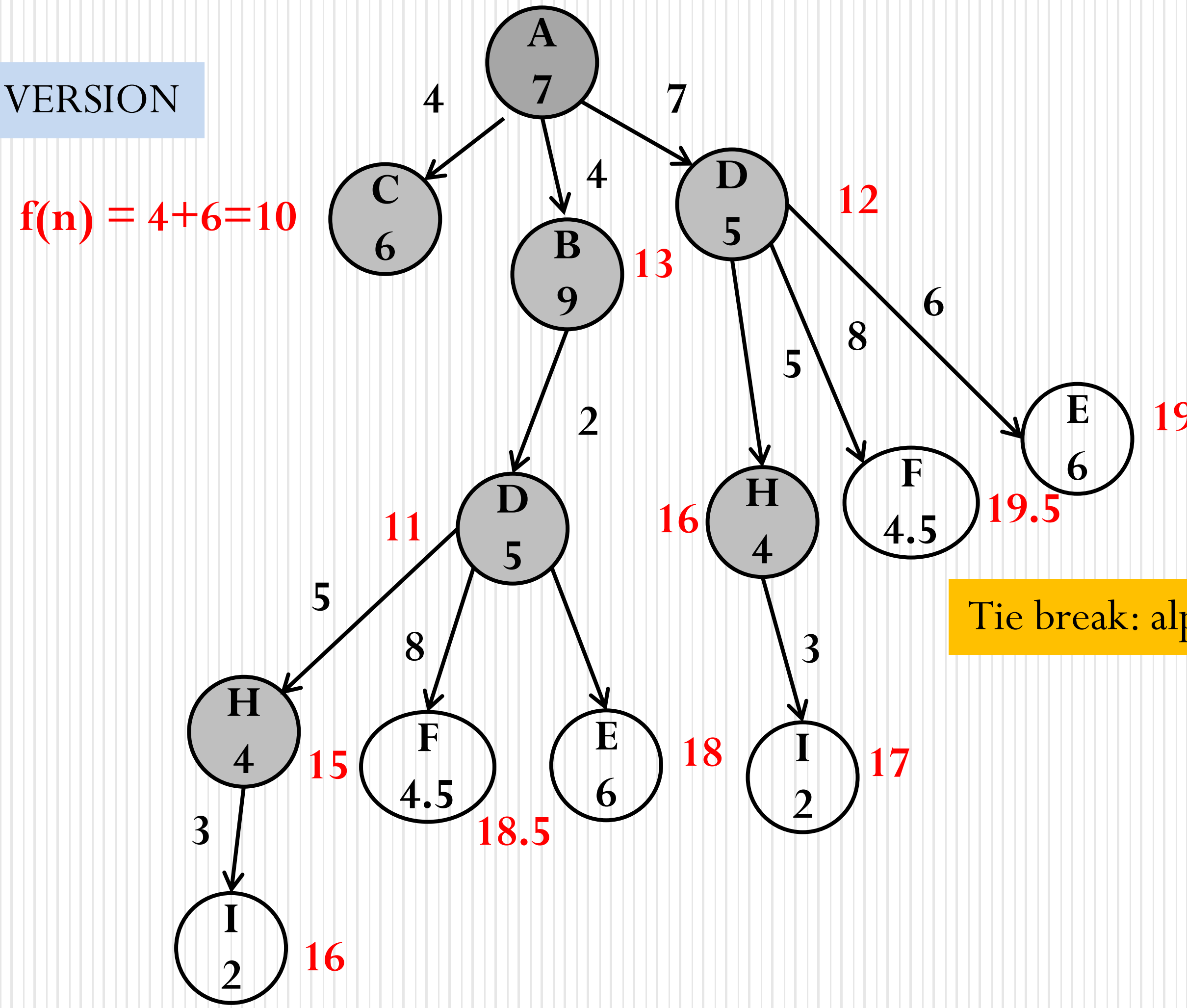
A * Example

TREE-SEARCH VERSION



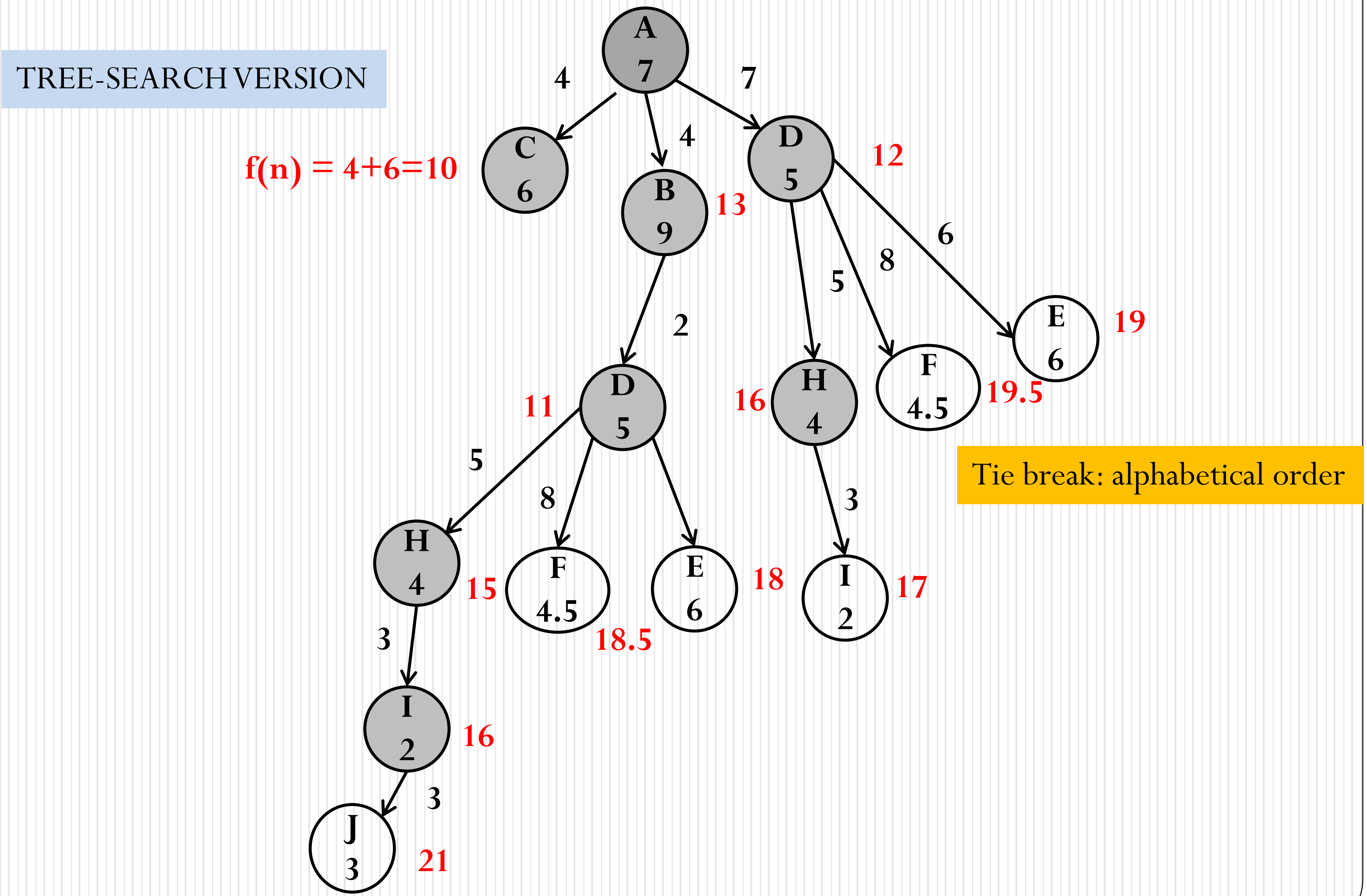
A * Example

TREE-SEARCH VERSION



Tie break: alphabetical order

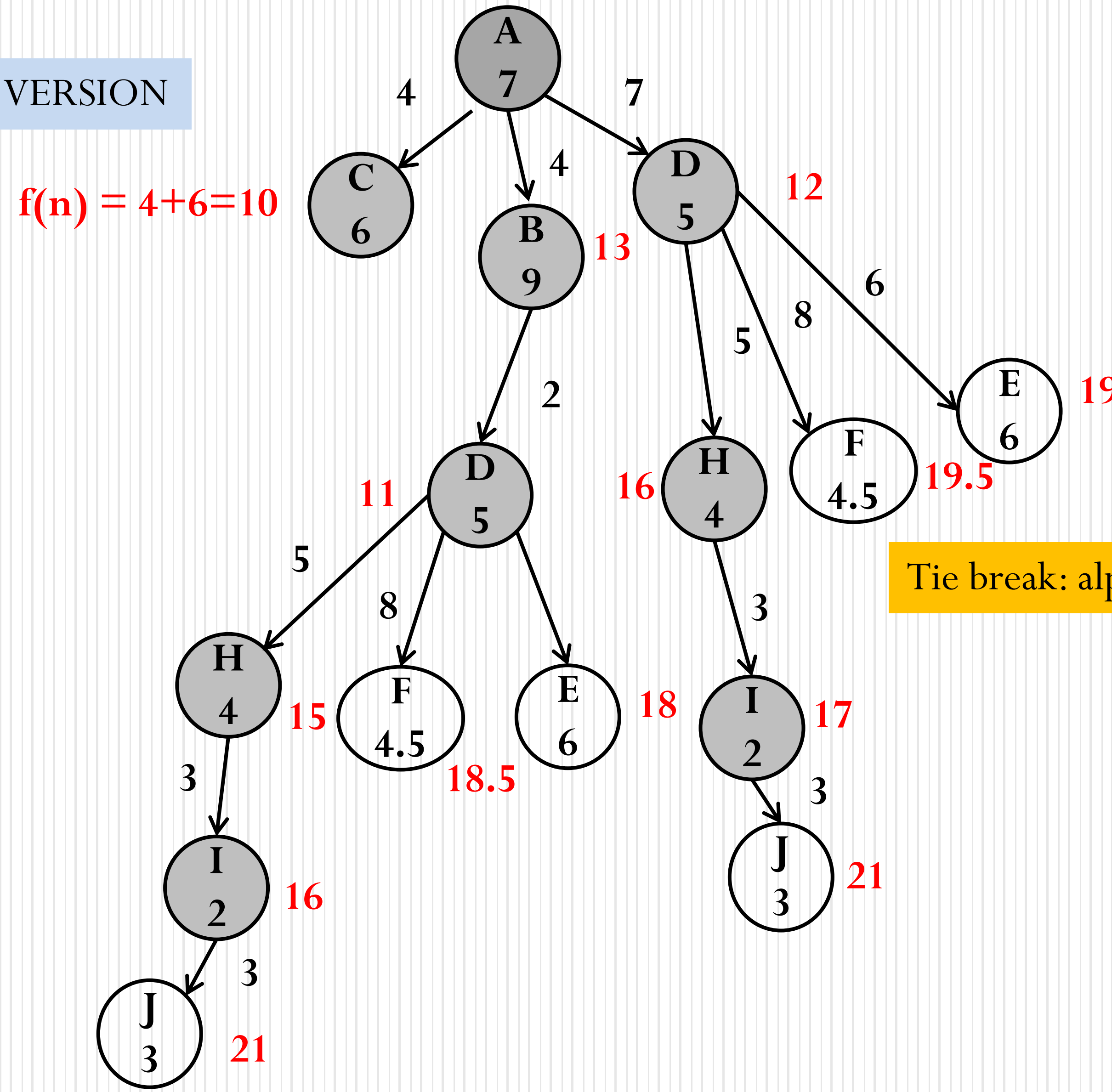
A * Example



TREE-SEARCH VERSION
f(n) = 4+6=10
A 7
C 6
B 9
D 5
4
4
7
13
12
2
6
8
5
E 6
19
F 4.5
19.5
H 4
16
D 5
11
5
8
H 4
15
F 4.5
18.5
E 6
18
3
I 2
17
3
I 2
16
3
J 3
21
Tie break: alphabetical order

A * Example

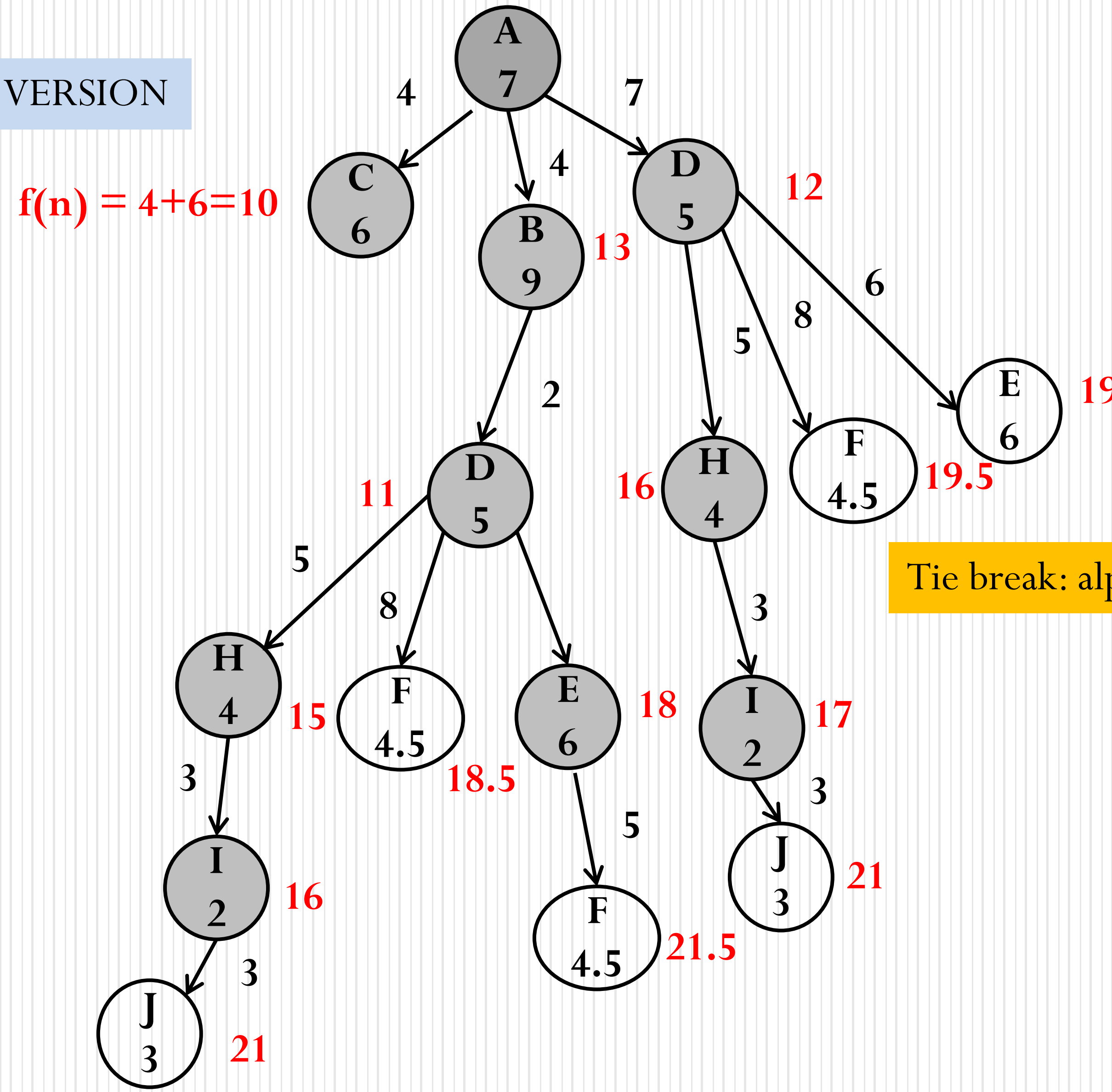
TREE-SEARCH VERSION



Tie break: alphabetical order

A * Example

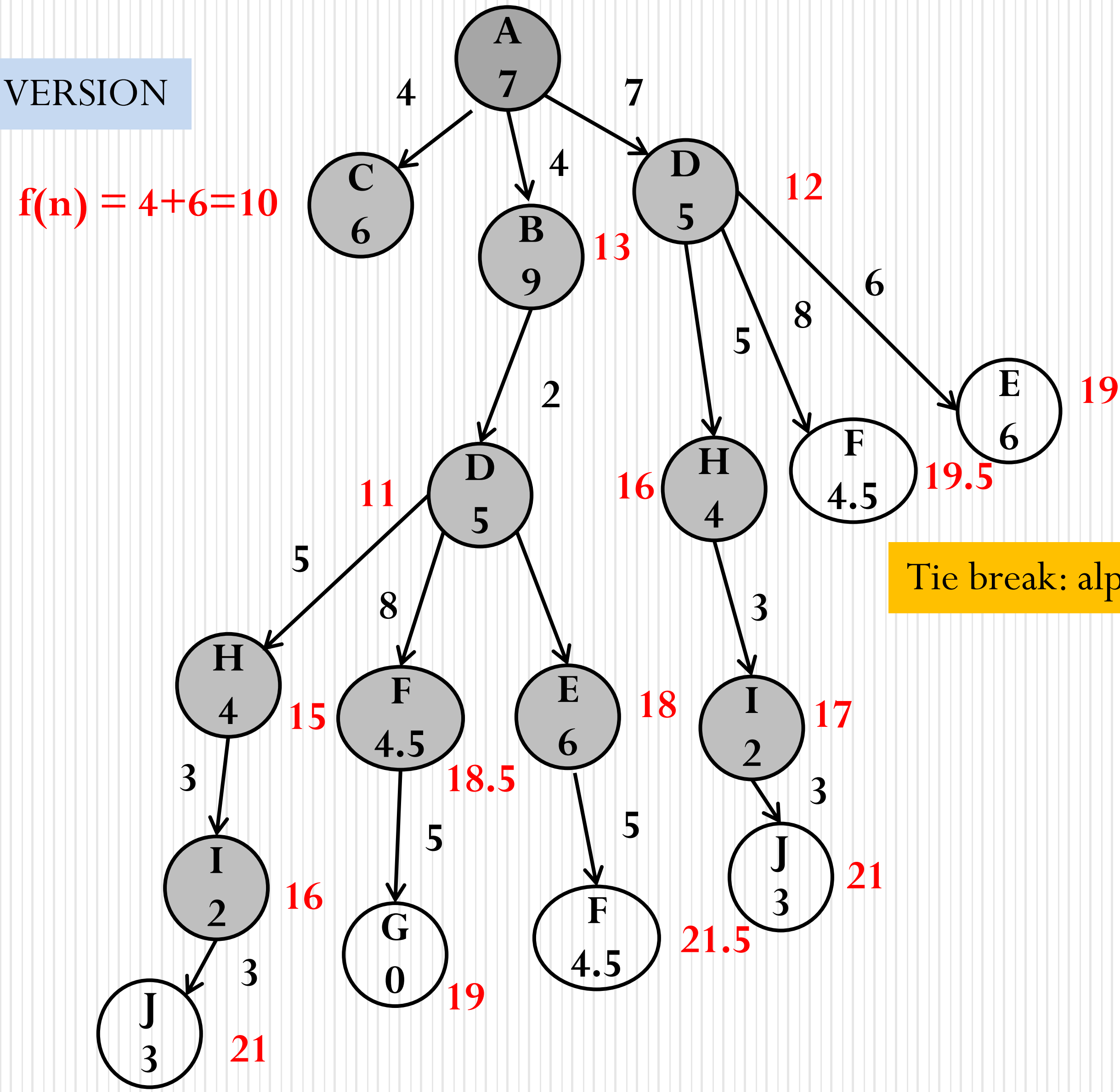
TREE-SEARCH VERSION



Tie break: alphabetical order

A * Example

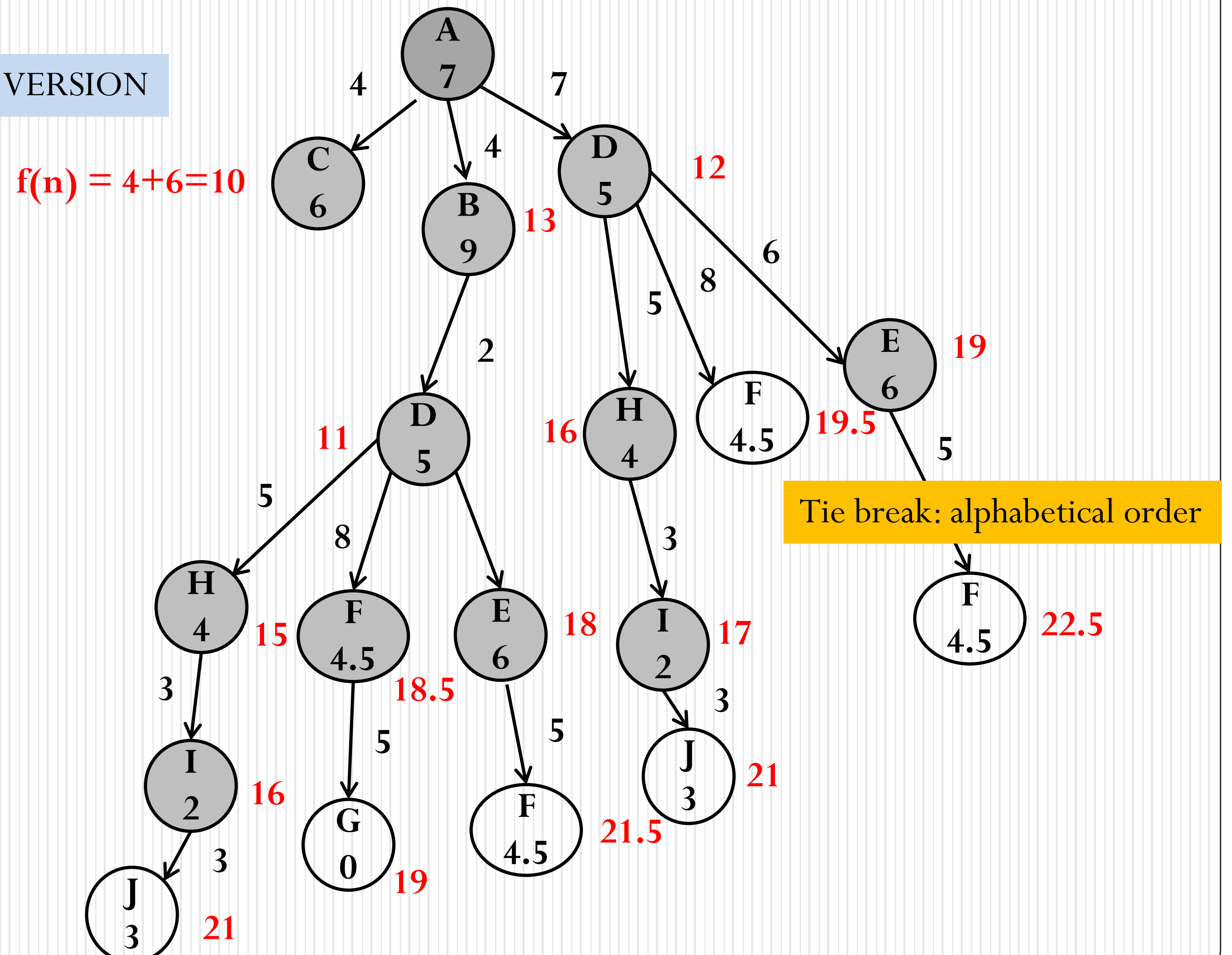
TREE-SEARCH VERSION



Tie break: alphabetical order

A * Example

TREE-SEARCH VERSION

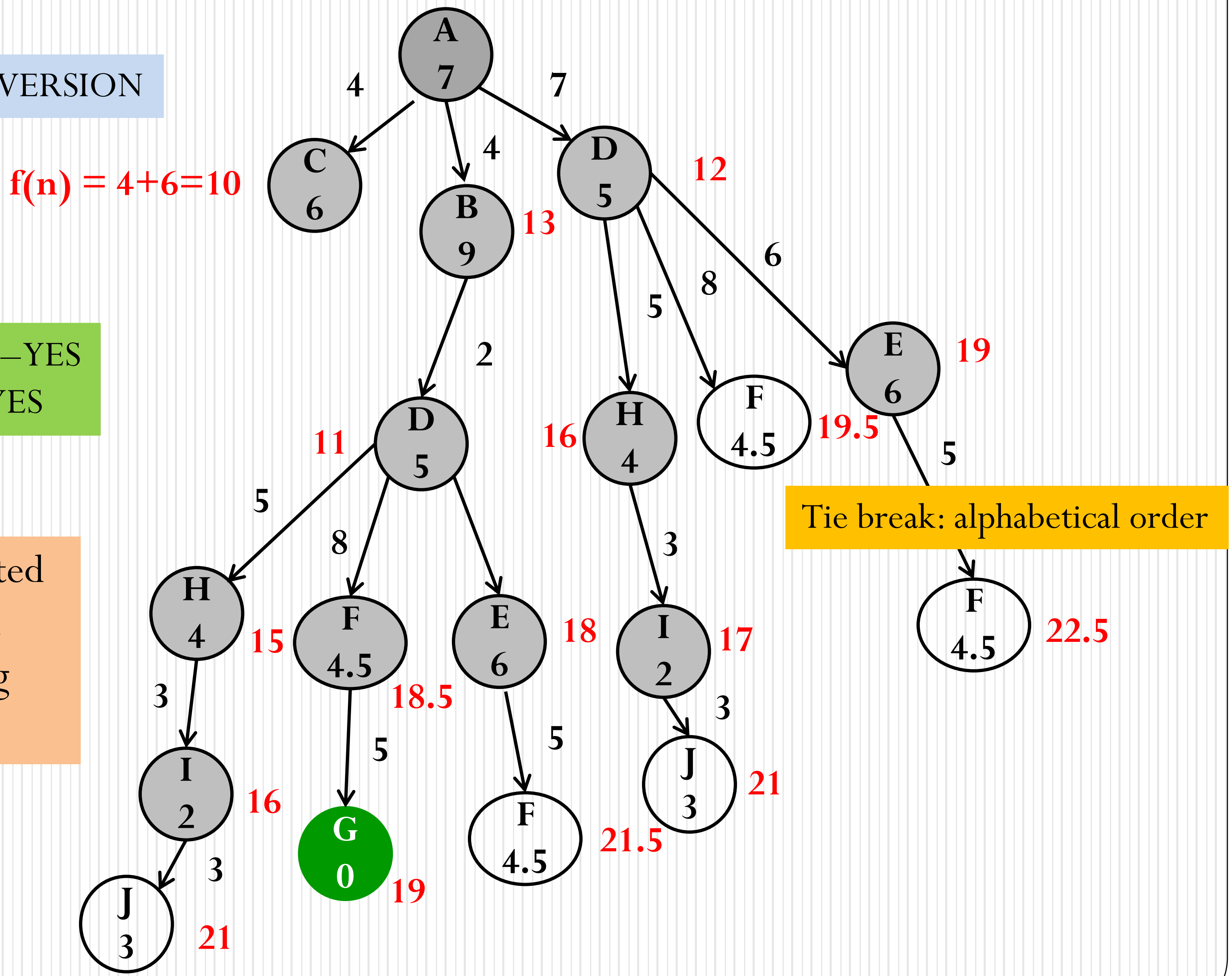


A * Example

TREE-SEARCH VERSION

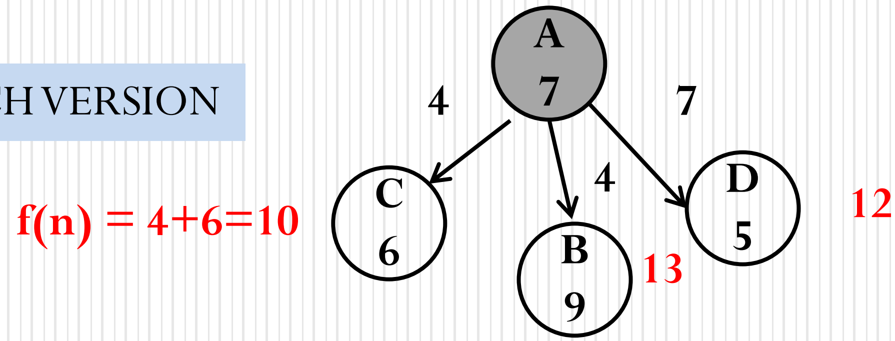
COMPLETE ? – YES
OPYIMAL ? - YES

Search is duplicated from D, identical sub-trees hanging below



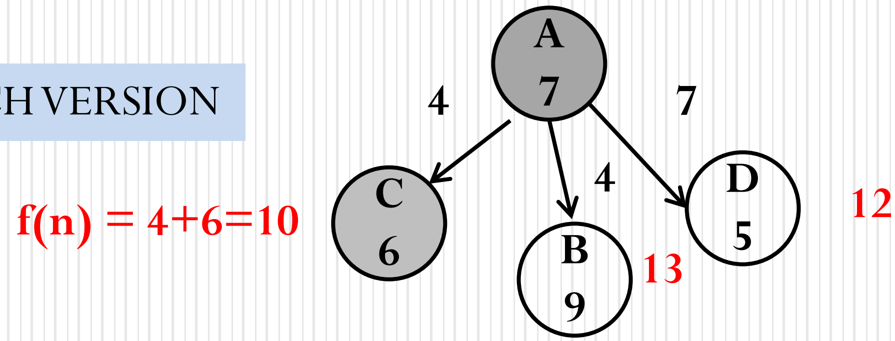
A * Example

GRAPH-SEARCH VERSION



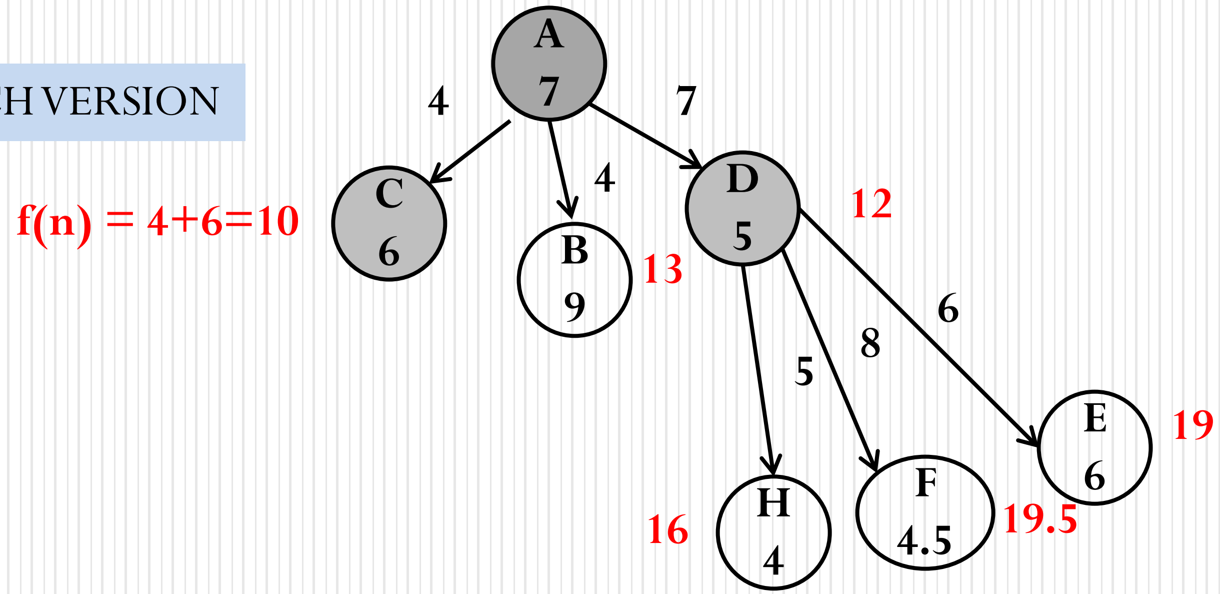
A * Example

GRAPH-SEARCH VERSION



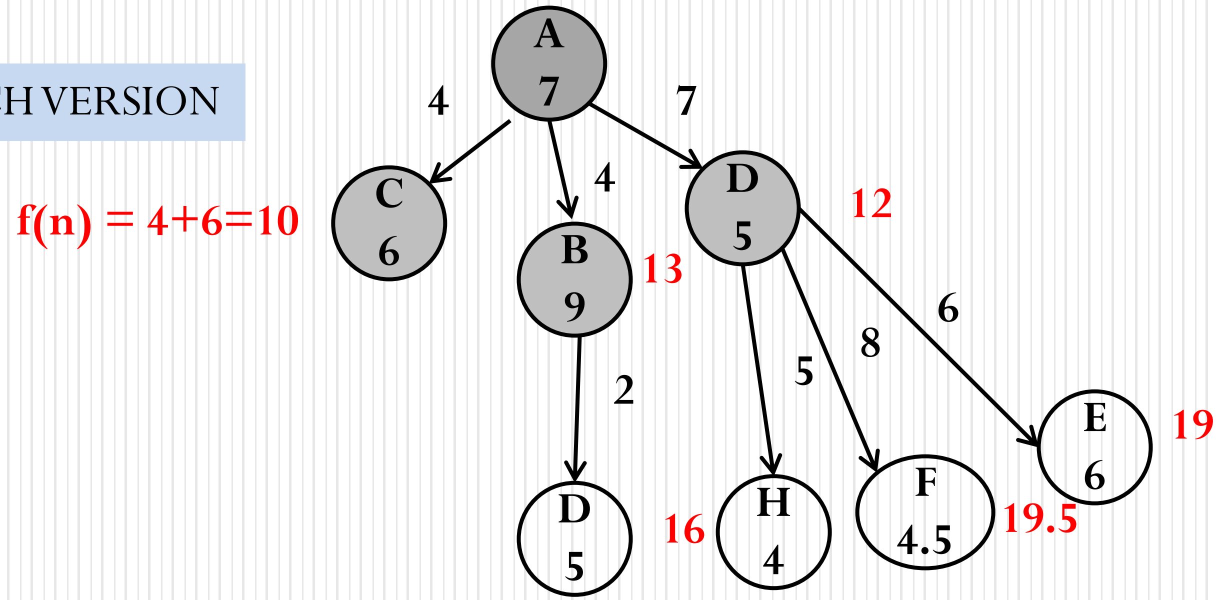
A * Example

GRAPH-SEARCH VERSION



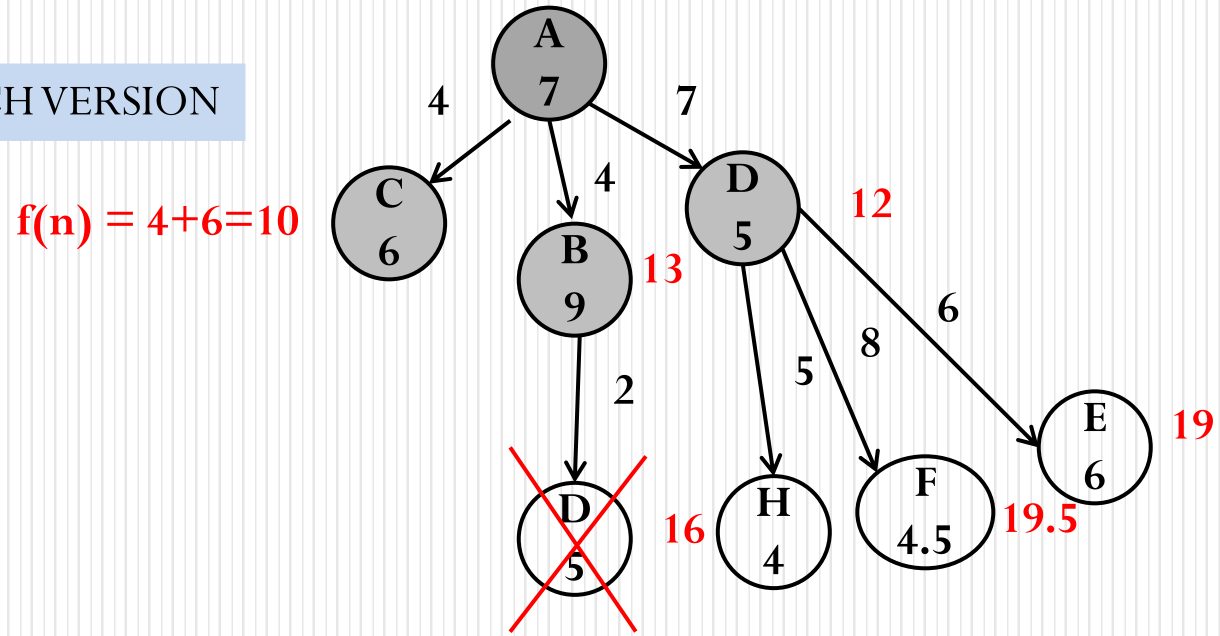
A * Example

GRAPH-SEARCH VERSION



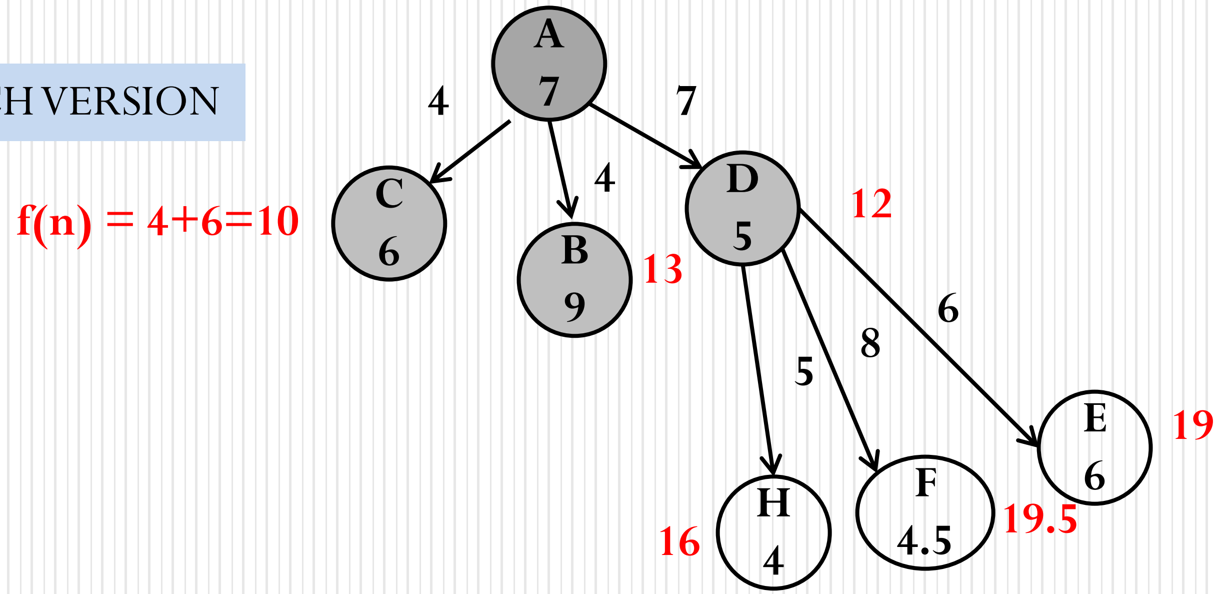
A * Example

GRAPH-SEARCH VERSION



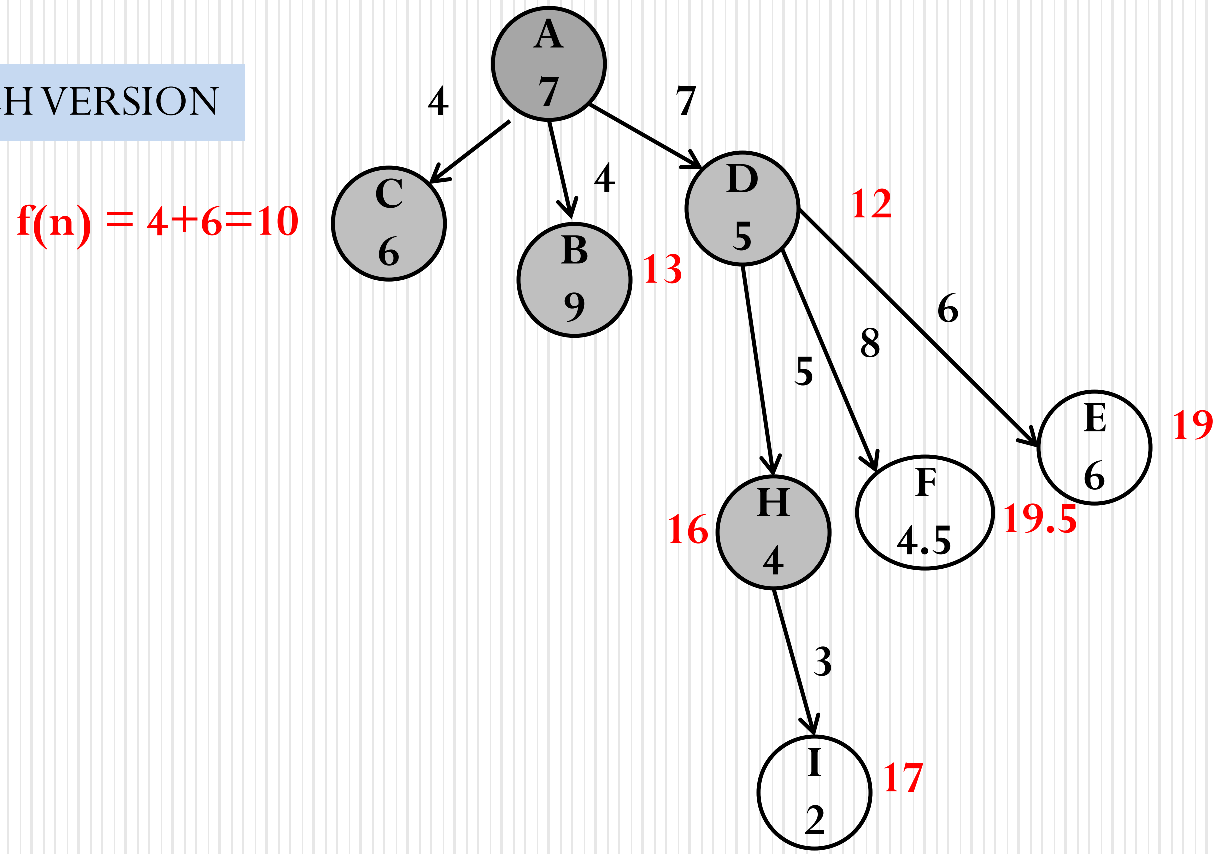
A * Example

GRAPH-SEARCH VERSION



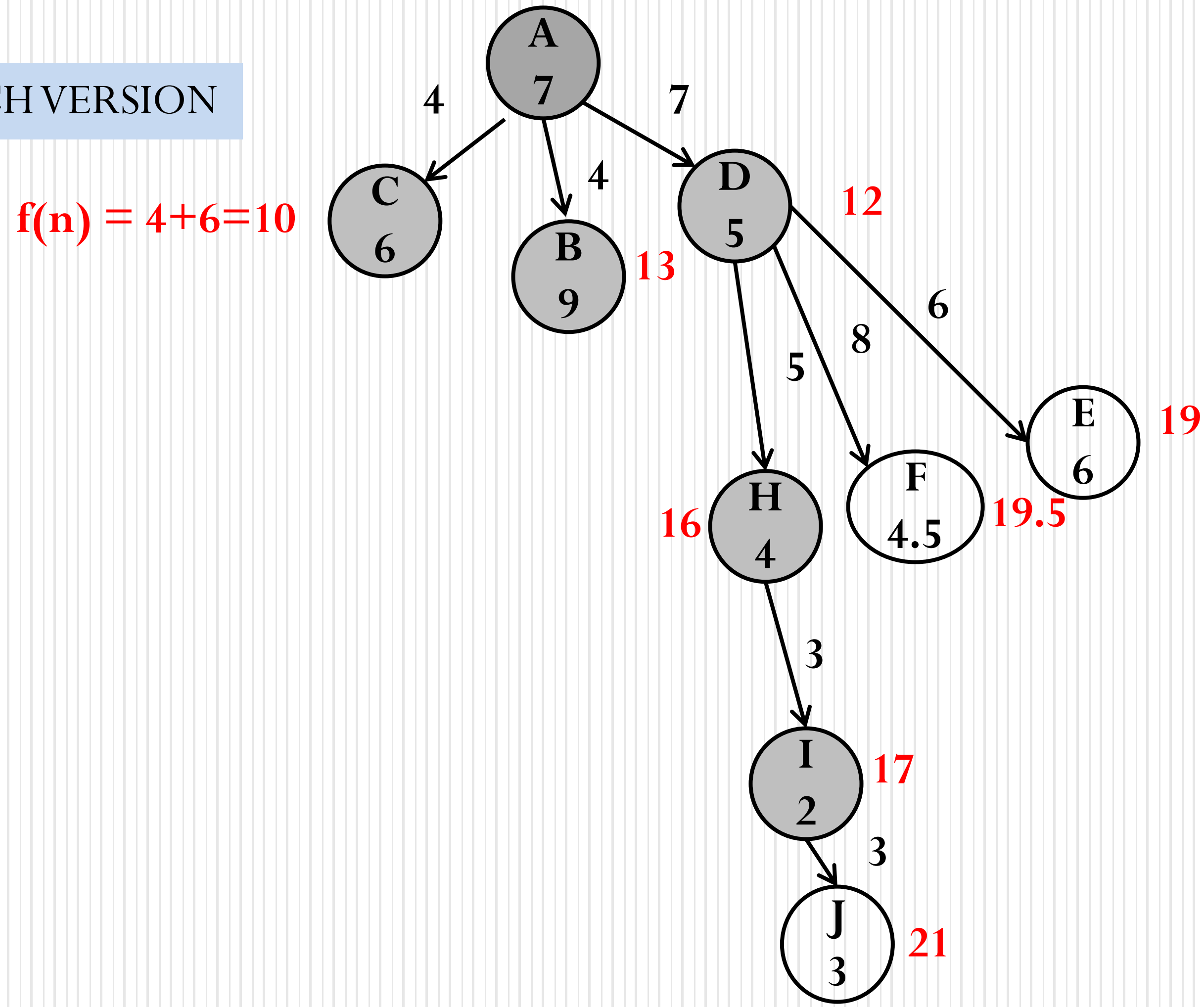
A * Example

GRAPH-SEARCH VERSION



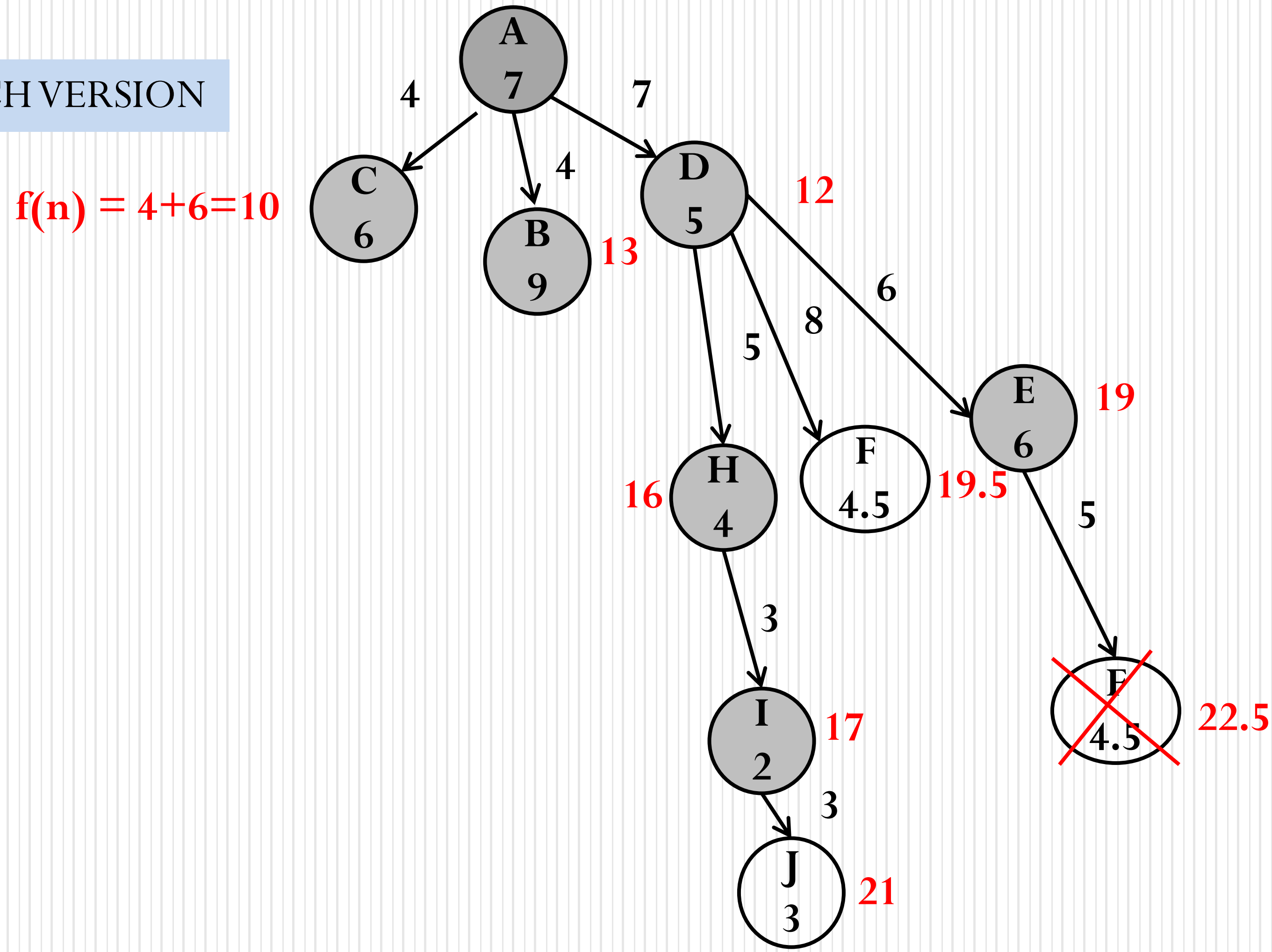
A * Example

GRAPH-SEARCH VERSION



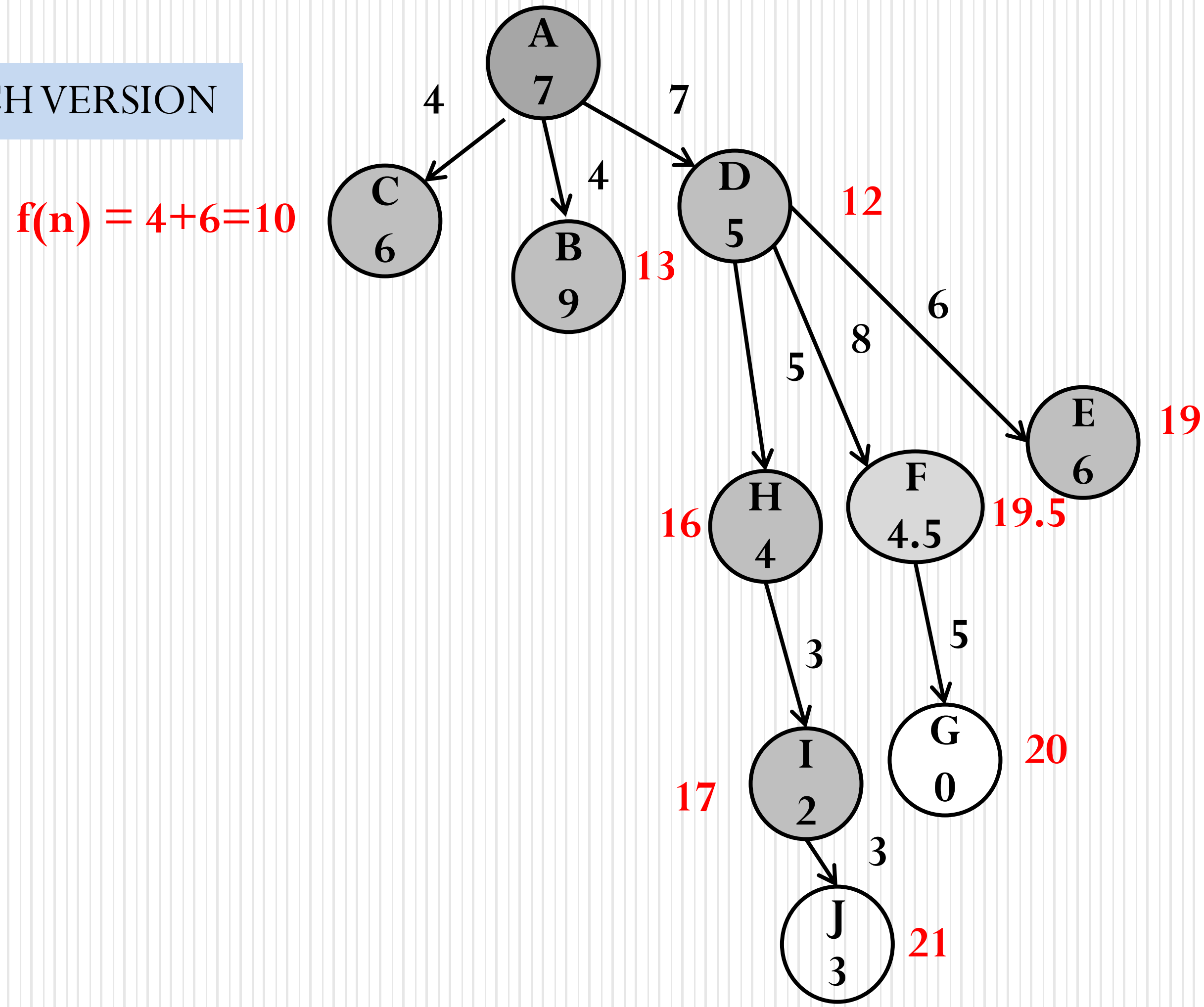
A * Example

GRAPH-SEARCH VERSION



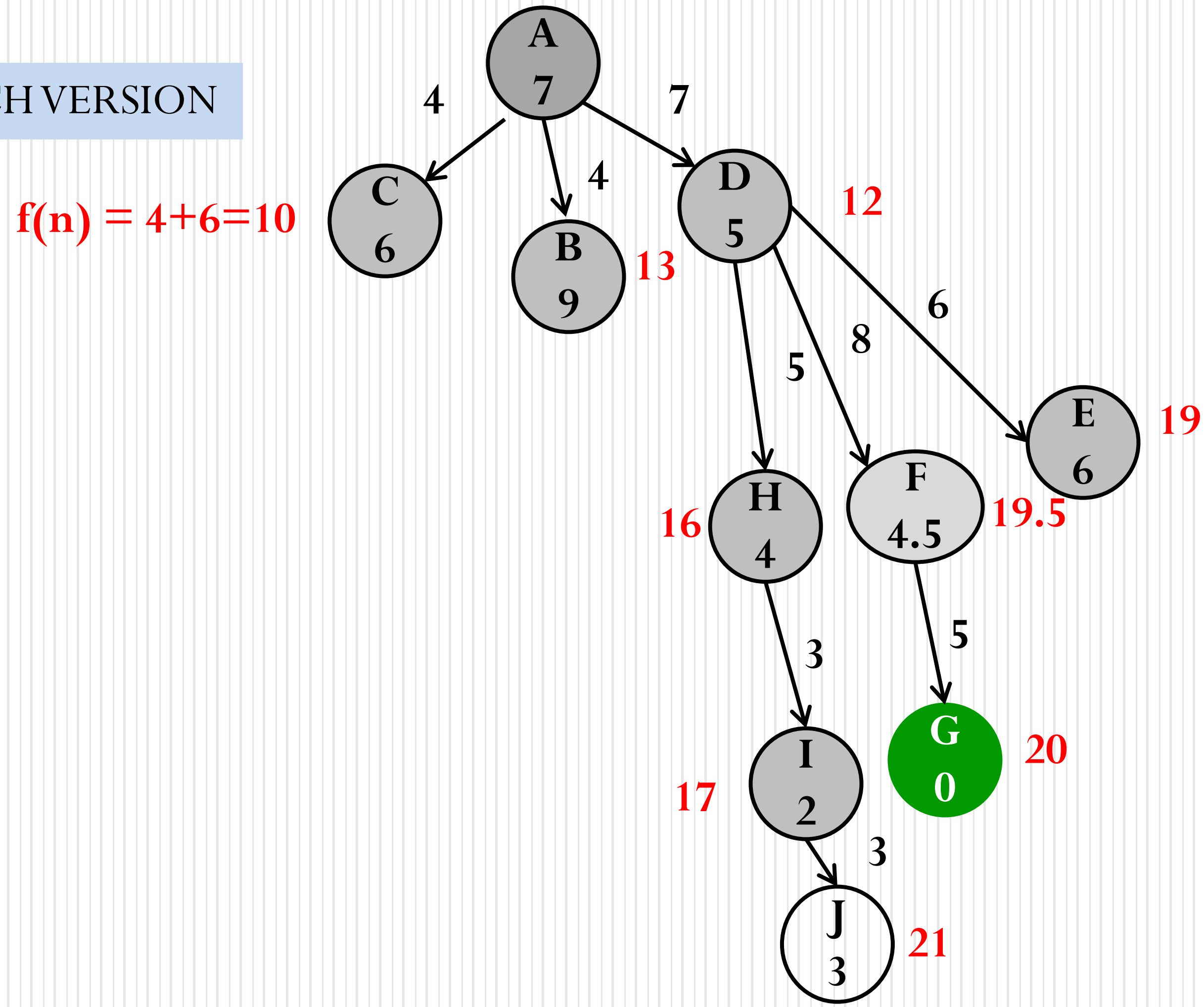
A * Example

GRAPH-SEARCH VERSION



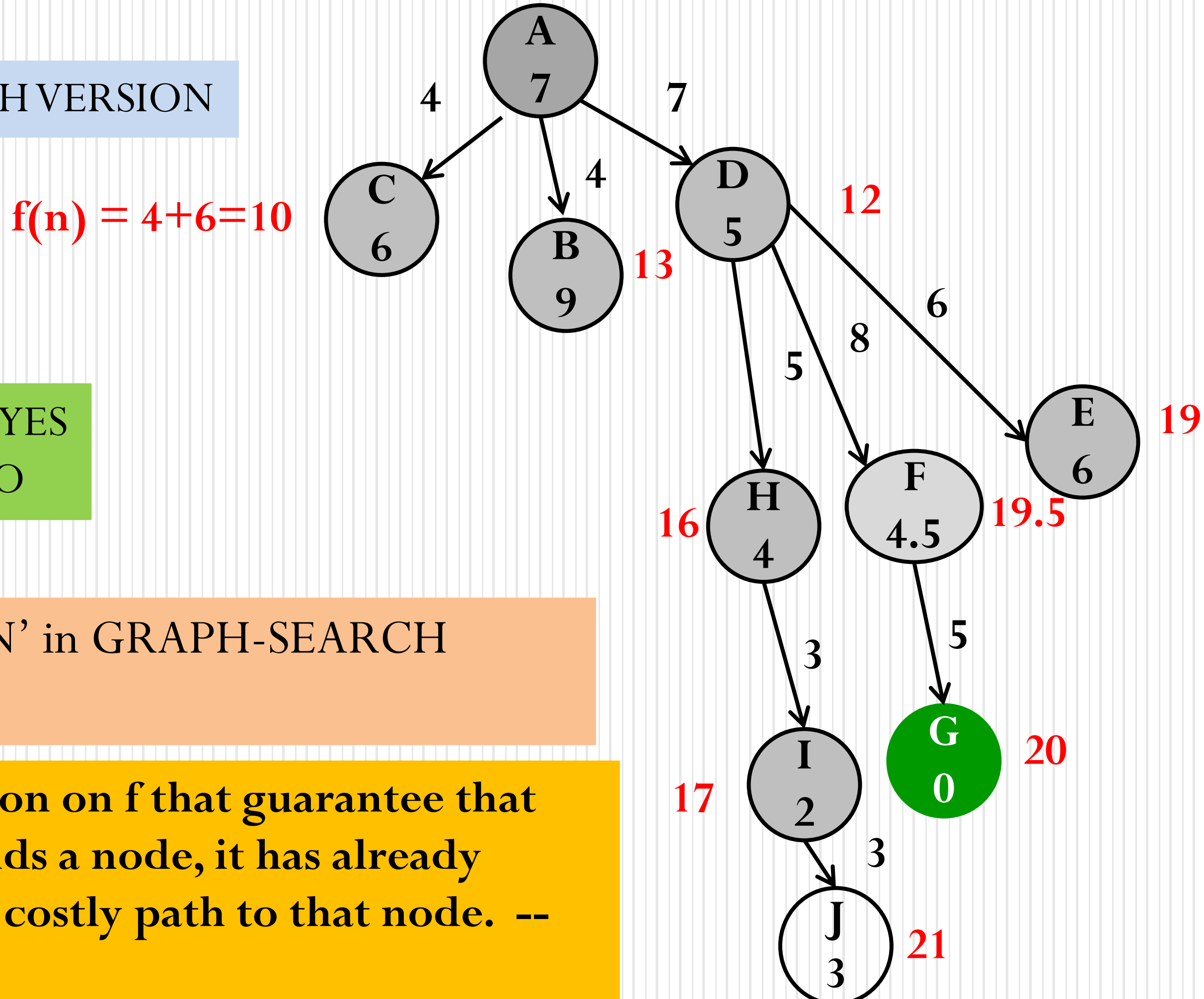
A * Example

GRAPH-SEARCH VERSION



A * Example

GRAPH-SEARCH VERSION



COMPLETE ? – YES
OPTIMAL ? - NO

Allow 'REOPEN' in GRAPH-SEARCH Version

Impose condition on f that guarantee that when A* expands a node, it has already found the least costly path to that node. -- CONSITENCY

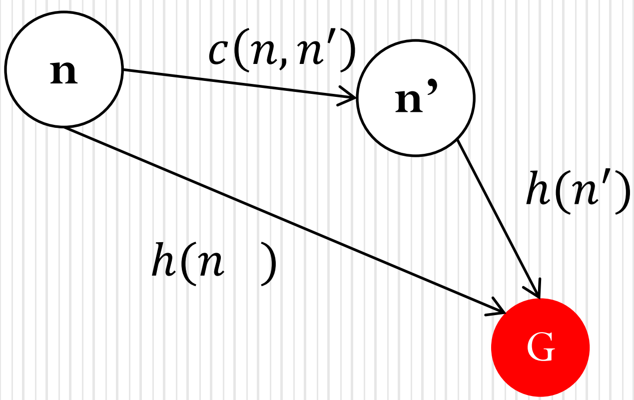
A* Search

- If n' is successor of n , h is **consistent** if $\forall\, n$, n'

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

where $c(n, n')$ is the step cost from n to n'

i.e. the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n'

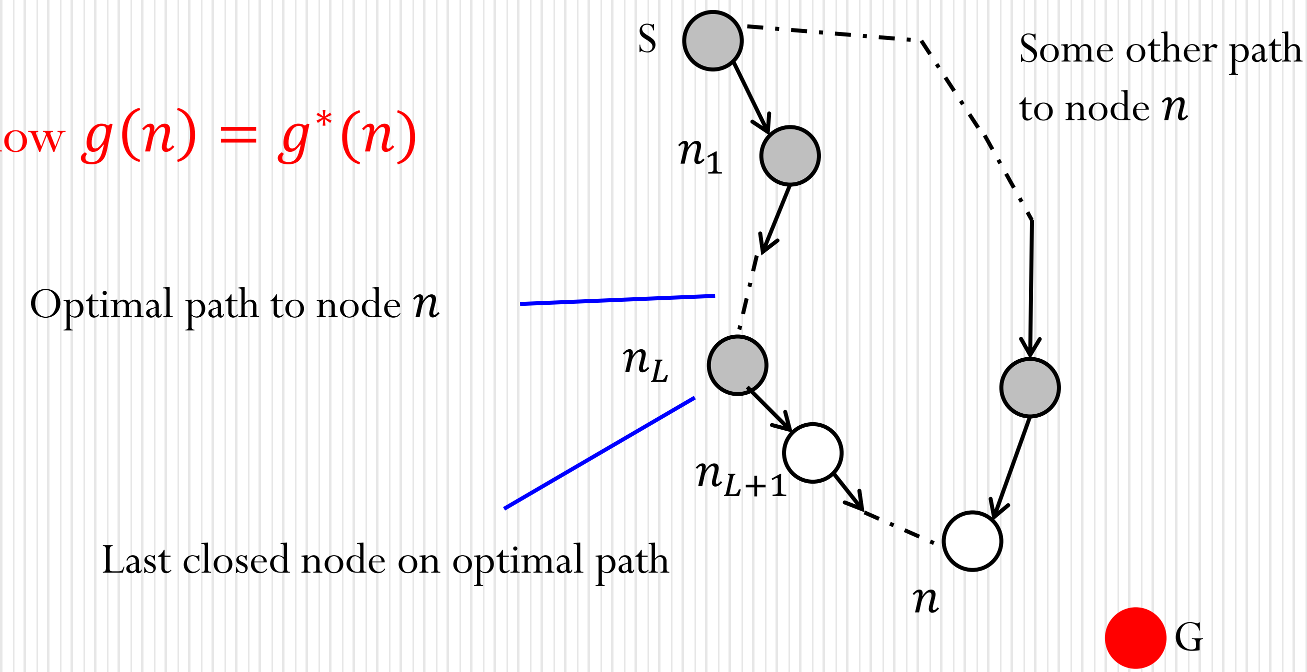


If $h(n)$ is consistent, then the values of f(n) along any path are non-decreasing

A* Search

- Theorem: if the consistency condition on h is satisfied, then when A* expands a node n, it has already found an optimal path to n.
- Proof:

  we will try to show $g(n) = g^*(n)$



A* Search

- Proof (contd.) : let A* is about to pick up node n with a value g(n). Let there be a (known) optimal path from S to n via n_L and n_{L+1}.
- Let A* expand node n with cost g(n).
- Let n_L be the last node on the optimal path from S to n that has been expanded.
- Let n_{L+1} be the successor of n_L that must be on OPEN.

- The following property holds
  - $h(n_L) \leq h(n_{L+1}) + C(n_L, n_{L+1})$
  - $h(n_L) + g(n_L) \leq h(n_{L+1}) + C(n_L, n_{L+1}) + g(n_L)$ (add $g(n_L)$)
  - $h(n_L) + g(n_L) \leq h(n_{L+1}) + g(n_{L+1})$
  - $h(n_L) + g^*(n_L) \leq h(n_{L+1}) + g^*(n_{L+1})$ (as both are in optimal path)

A* Search

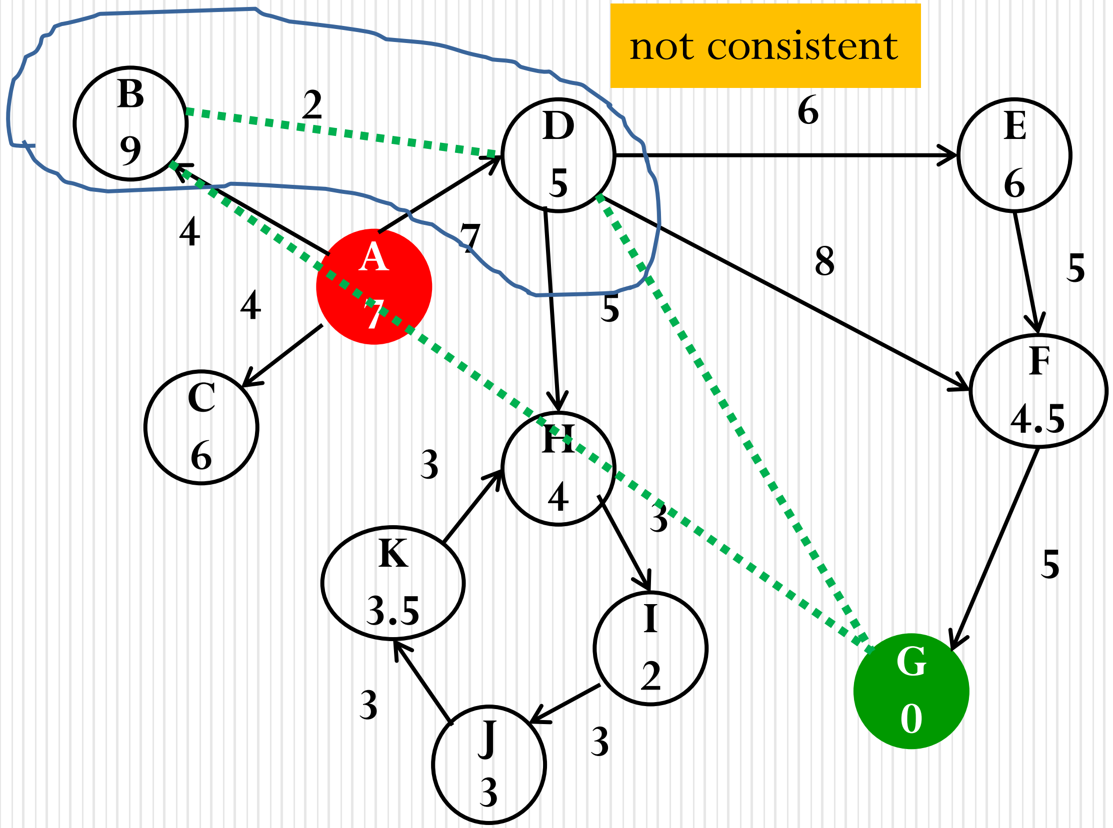
- By transitivity of $\leq$, the above property holds true for any two nodes on the optimal path, in particular it holds for n_{L+1} and node n.
  - $h(n_{L+1}) + g^*(n_{L+1}) \leq h(n)+g^*(n)$ **---------(1)**
  - $f(n_{L+1}) \leq h(n)+g^*(n)$ (because n_{L+1} is on the optimal path to n)
- But since A* is about to pick node n instead,
  - $f(n) \leq f(n_{L+1})$ or
  - $h(n) + g(n) \leq h(n_{L+1}) + g^*(n_{L+1})$ **---------(2)**

  Combining (1) and (2)
  - $h(n) + g(n) \leq h(n_{L+1}) + g^*(n_{L+1}) \leq h(n)+g^*(n)$
  - $h(n) + g(n) \leq h(n)+g^*(n)$
  - $g(n) \leq g^*(n)$

  g(n) cannot be less than g*(n) as g*(n) is the optimal cost.

  $\mathbf{g(n) = g^*(n)}$

A * Example



not consistent

h(D) can be changed to 8 for consistency property

Iterative Deepening A* Search

- Heuristic search can perform as bad as Breadth first search (specially in terms of space) if a *good* heuristic is not chosen.
- Basic idea of IDA* is to achieve similar benefits as iterative deepening DFS for heuristic search.
  - While finding minimal cost path the memory grows only linearly with depth of goal.
- Perform a depth-first search at each iteration with cut-off value as f cost.

Iteration 1

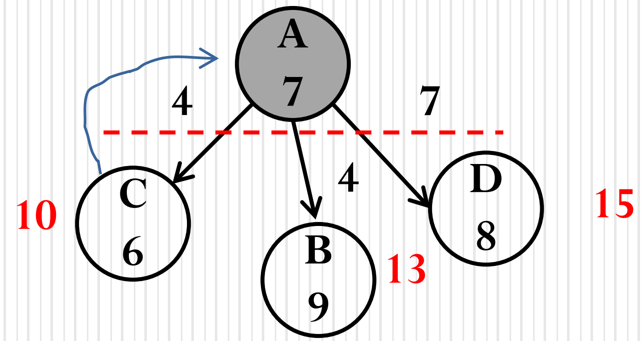
cost cut-off is $\mathrm{h}(n_0), n_0$ is the start node.

Successive Iterations

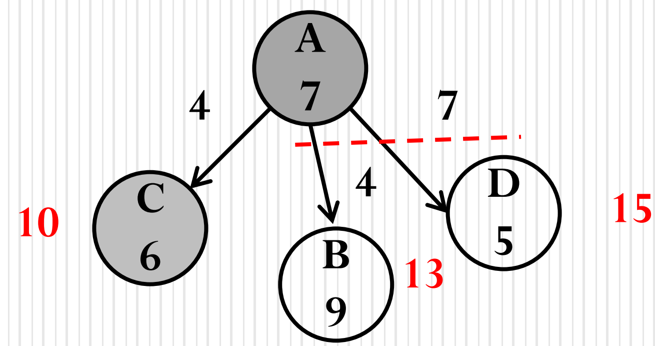
cut-off value is the smallest $\boldsymbol{f}$ cost of any node that exceeded the cut-off on previous iteration.

Iterative Deepening A*

- Expand nodes in depth-first fashion- backtracking whenever the f value of a successor of an expanded node exceeds the cut-off value



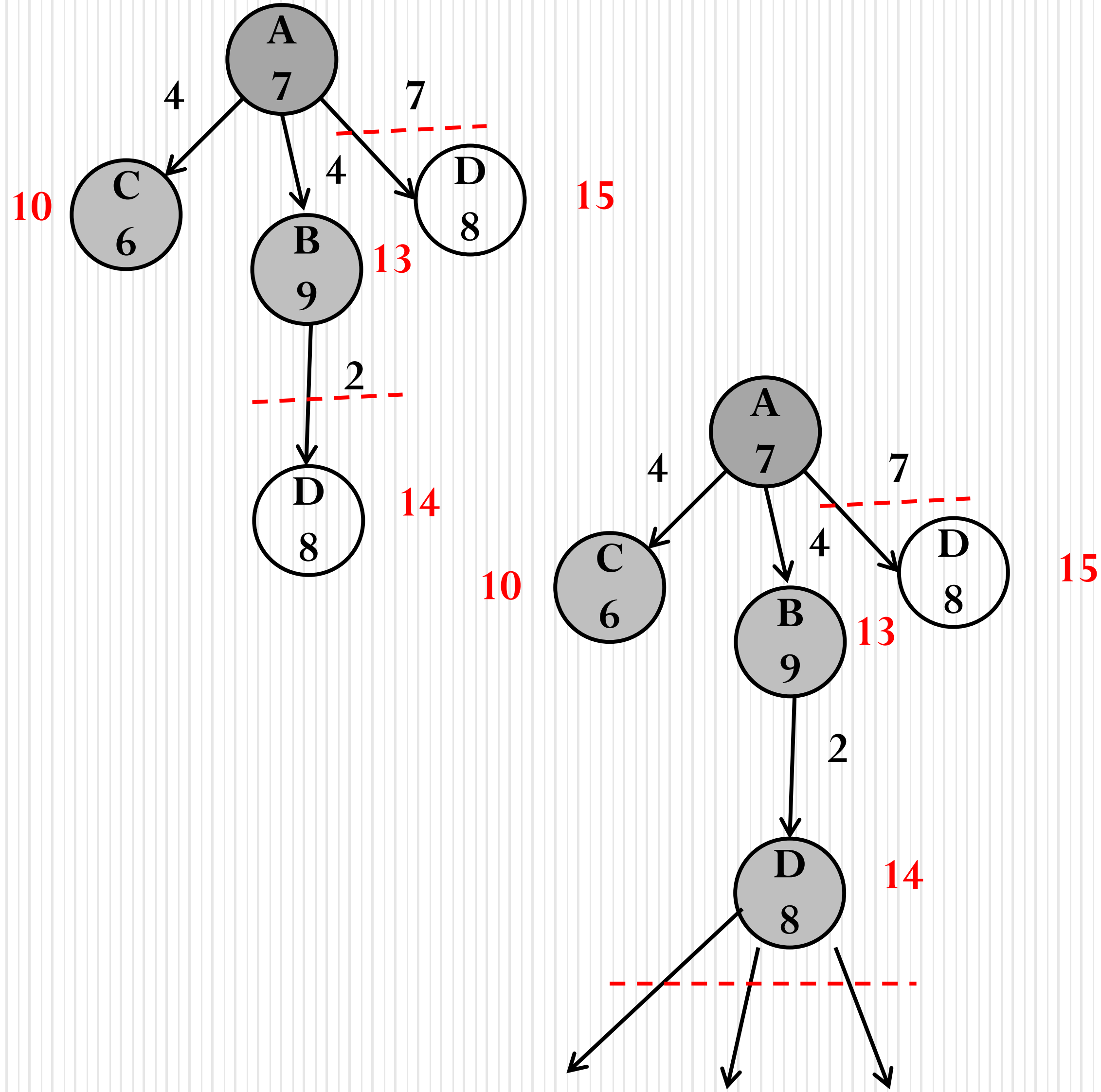
- **Iteration 1:**
  - cut-off = f(A) = h(A) = 7



- **Iteration 2:**
  - cut-off = f(C) = 10
  - Lowest of the f value of the nodes visited (but not expanded) in the previous DFS

Iterative Deepening A*

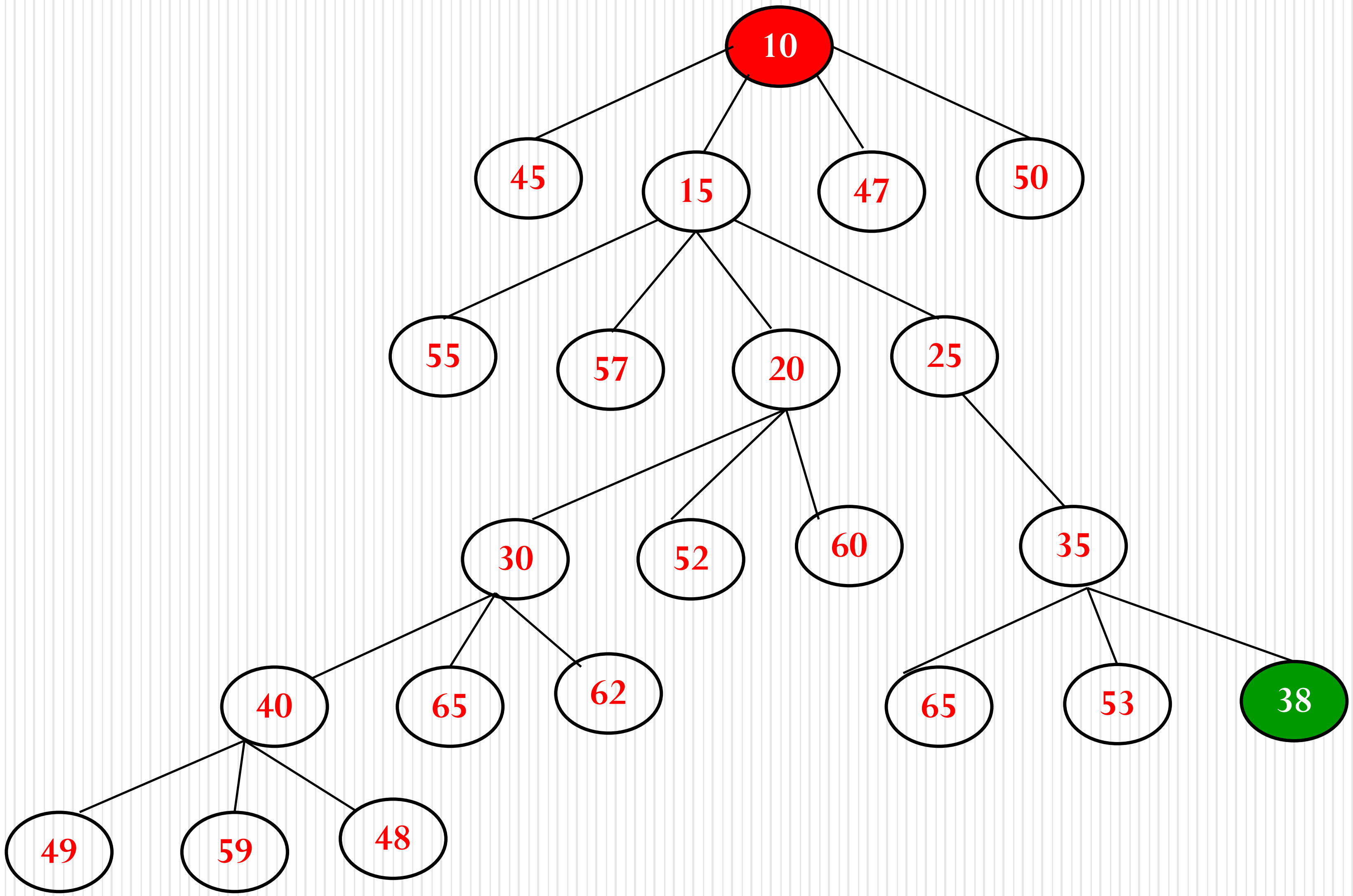
- **Iteration 3:**
  - cut-off = f(B) = 13
- **Iteration 4:**
  - cut-off = f(D) = 14



Recursive Best First Search

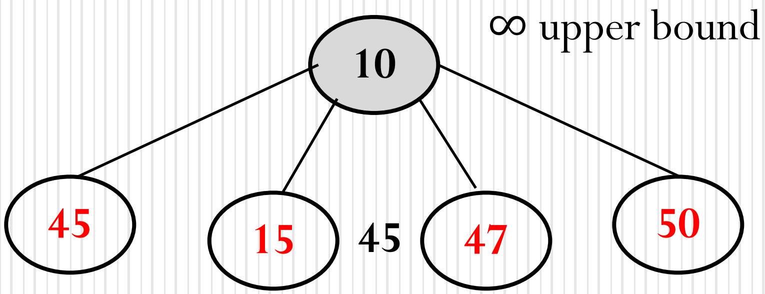
- Mimics best first search with linear space
- Similar to recursive depth-first but do not continue indefinitely down the current path
  - keeps track of the f value of the best alternative path available
  - if current f value exceeds this alternative f value then backtrack to alternative path
  - upon backtracking replace the f value of each node along the path with a ***backed-up*** value- the best f value of the children
- Like A* (Tree search) , RBFS is optimal if the heuristic is admissible.
- Space complexity is O(bd) and time complexity depends on the heuristic function and how often the best path changes as nodes are expanded.

RBFS Example



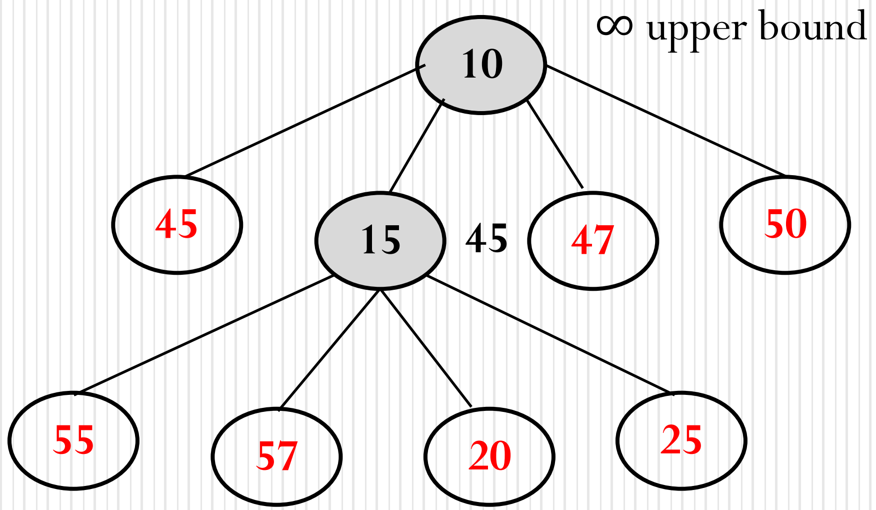
RBFS Example

Upperbound (n) =
min [upperbound(n's parent), current value of lowest cost brother]

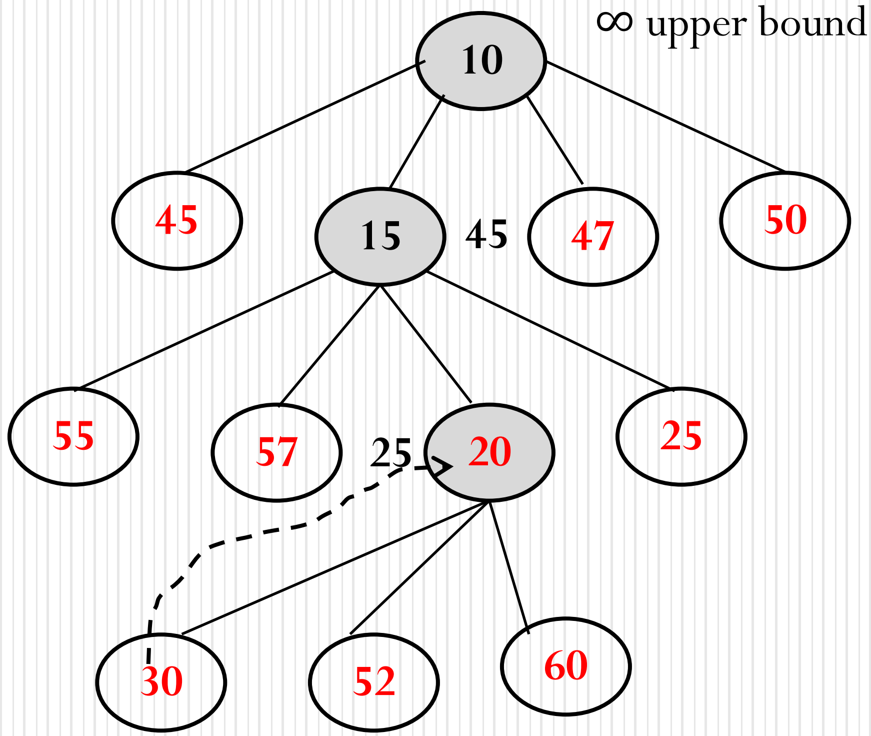


$f_{backed\text{-}up}(n) = f(n)$ /*if n is leaf*/
$f_{backed\text{-}up}(n) = f_{backed\text{-}up}(child)$,
child is successor of n /*if n is not a leaf node*/

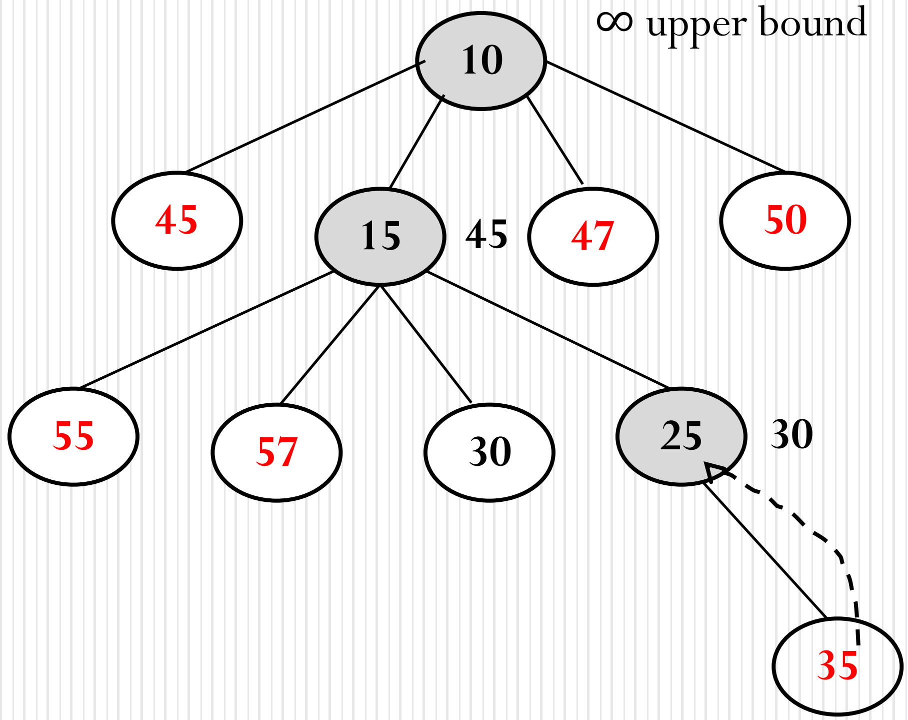
RBFS Example



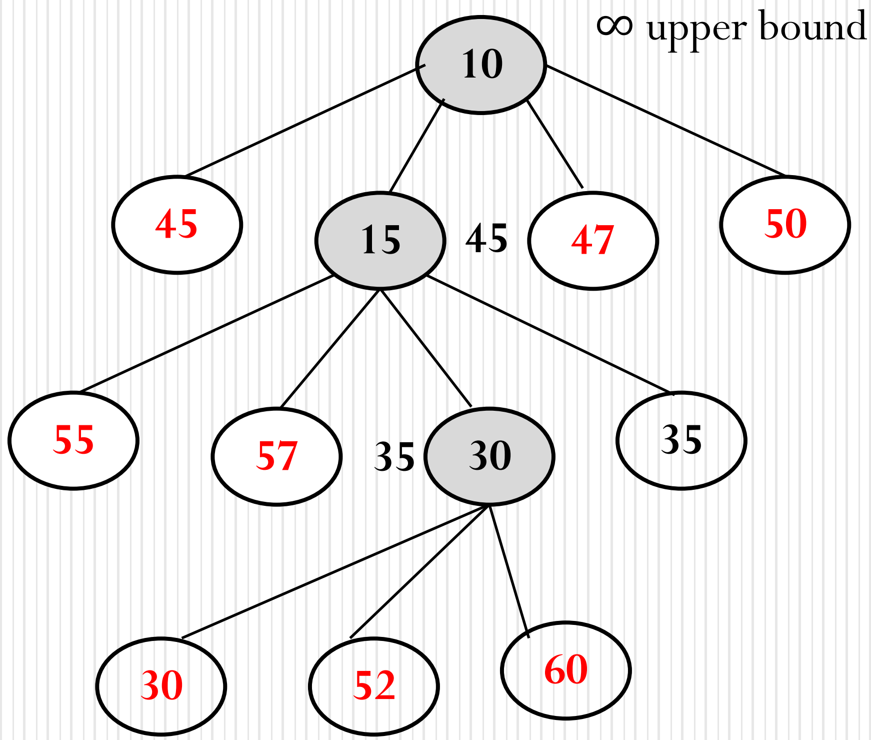
RBFS Example



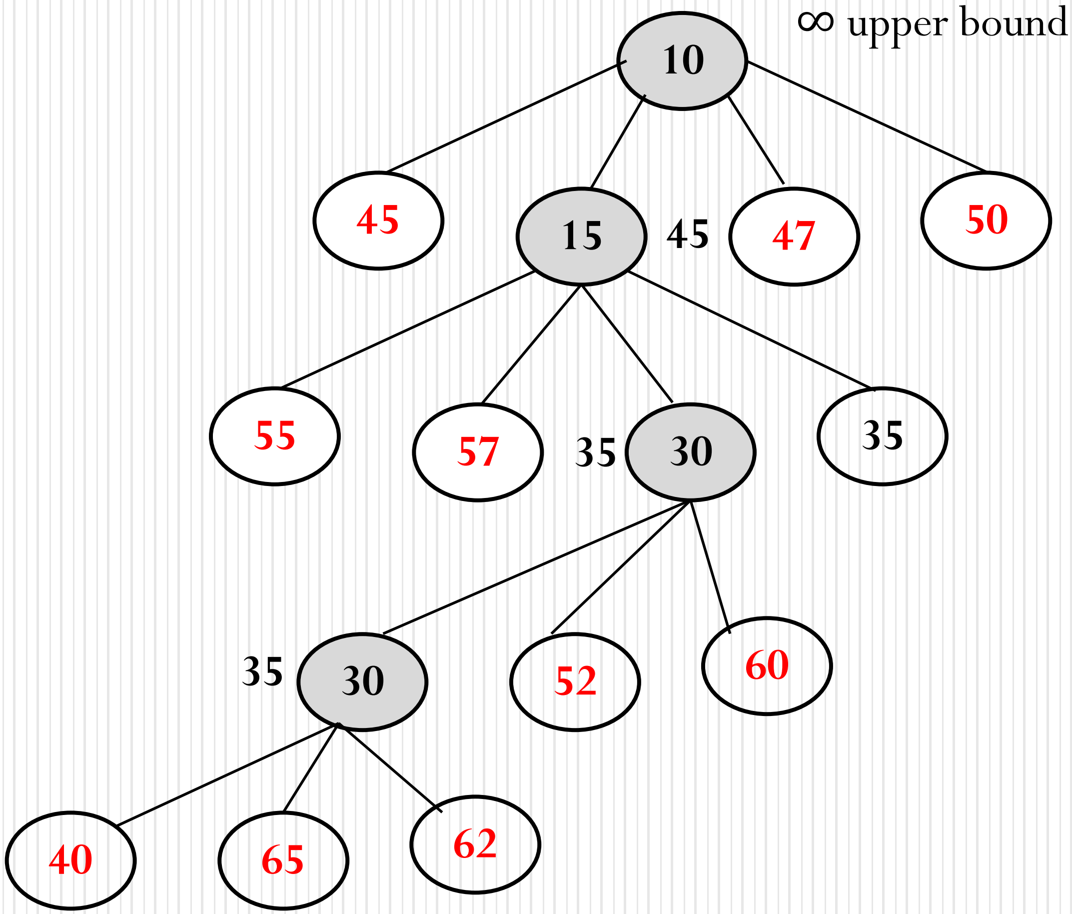
RBFS Example



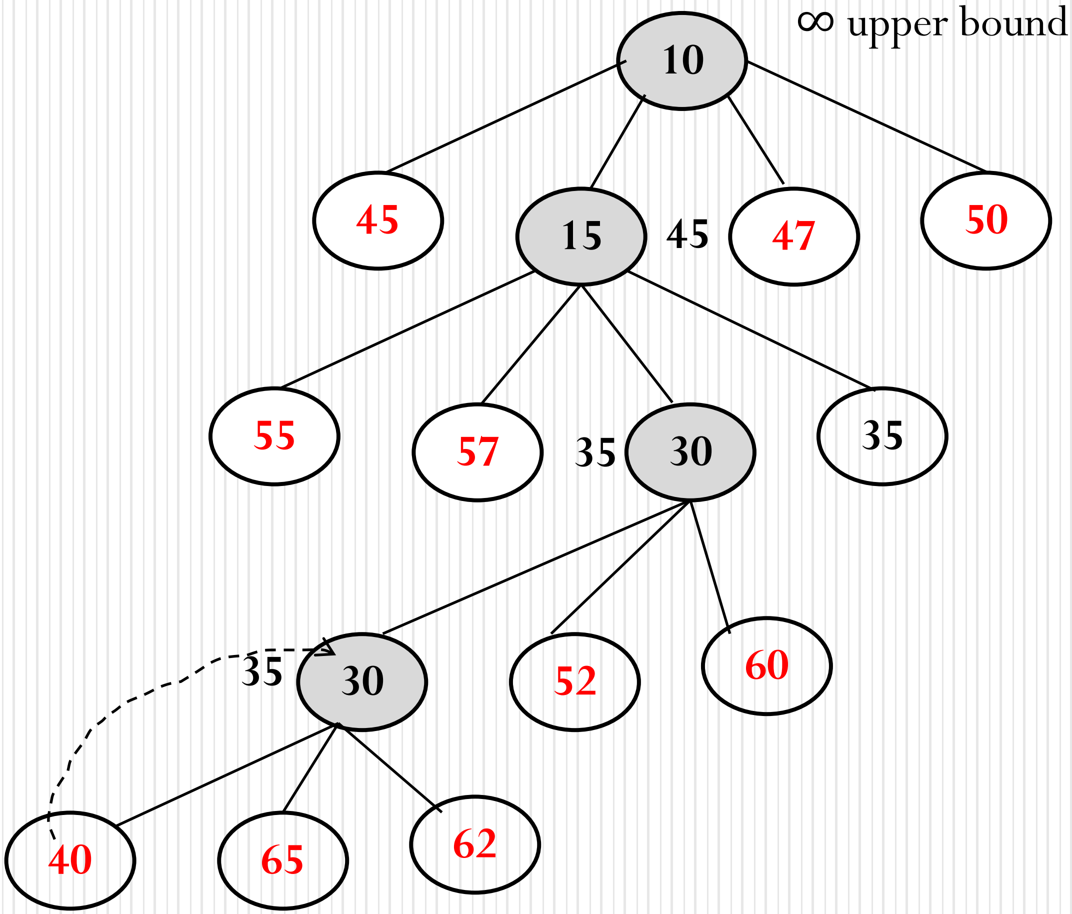
RBFS Example



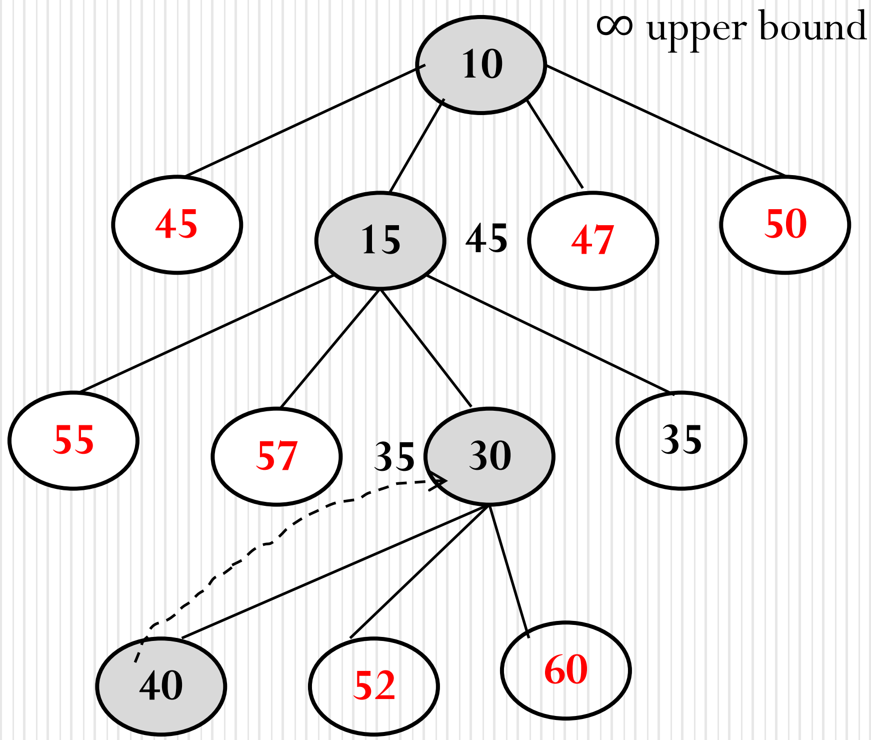
RBFS Example



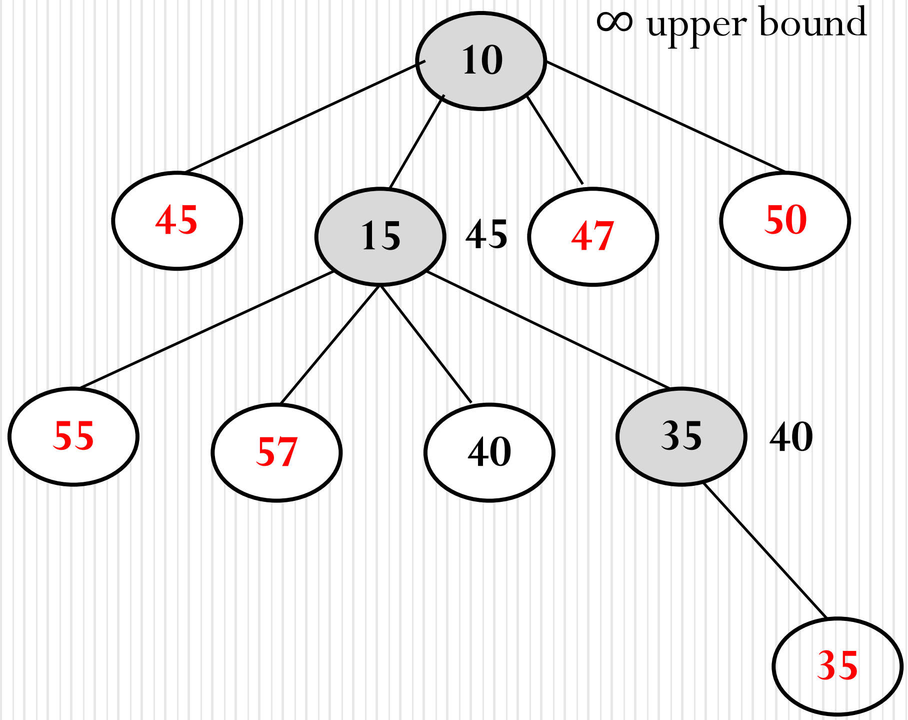
RBFS Example



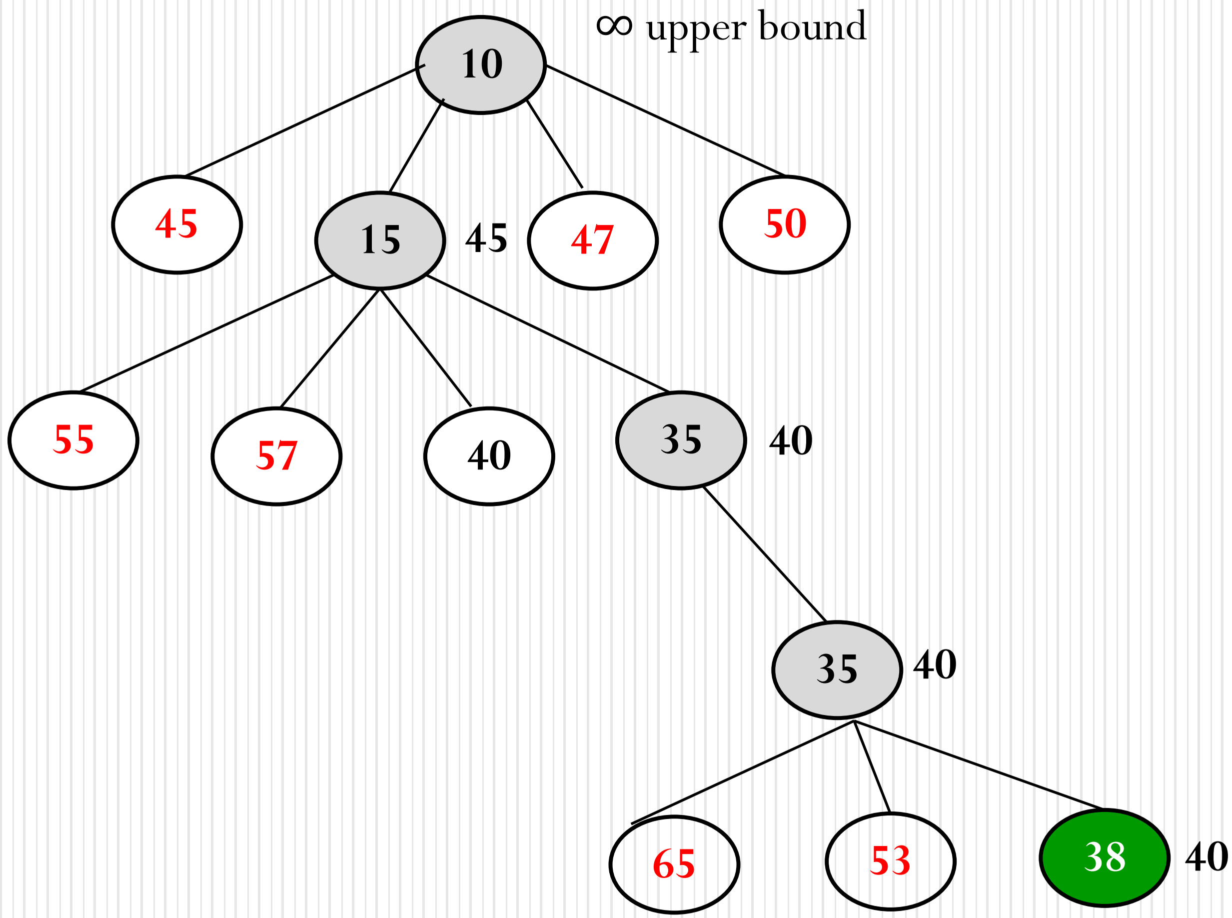
RBFS Example



RBFS Example



RBFS Example



More about Heuristics

- How to generate heuristics?
  - Relaxing problem definitions
  - Solution cost of a subproblem of a given problem
  - Learning from experience

- When more then one heuristic is available for the same problem, which one to choose?

Heuristics

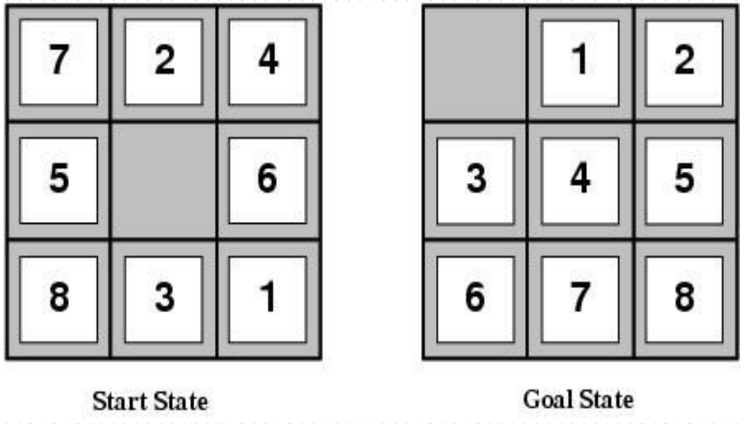
- Generating heuristics from Relaxed Problems
  - A problem with fewer restrictions on the actions is called a relaxed problem
  - The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.
- Example: 8-puzzle problem
  - A tile can move from square A to square B if A is horizontally or vertically adjacent to B and B is blank.
  - If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ = number of misplaced tiles.

Heuristics

- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ = sum of horizontal and vertical distances of the tiles from their goal positions (Manhattan distance)

$h_1(n) = 8$, admissible because any tile that is out of place must be moved at least once.

$h_2(n) = 3+1+2+2+2+2+3+3+3+2 = 18$, admissible because any move can slide tile one step closer to the goal.



Heuristics

- Generating Heuristics from subproblems
  - Admissible heuristics can also be derived from the solution cost of a subproblem of a given problem.
  - This cost is a lower bound on the cost of the real problem.

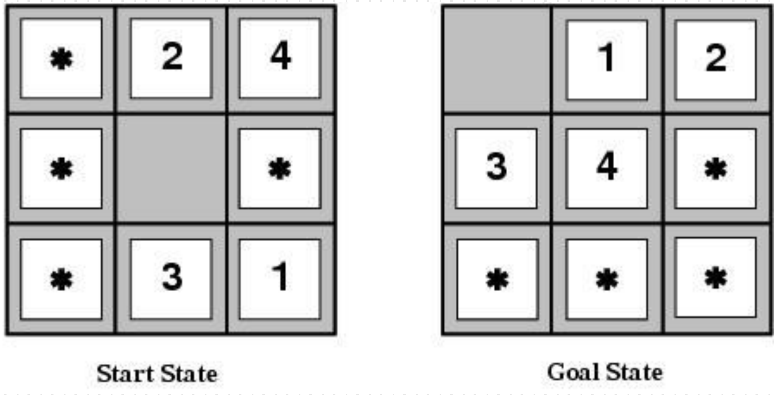


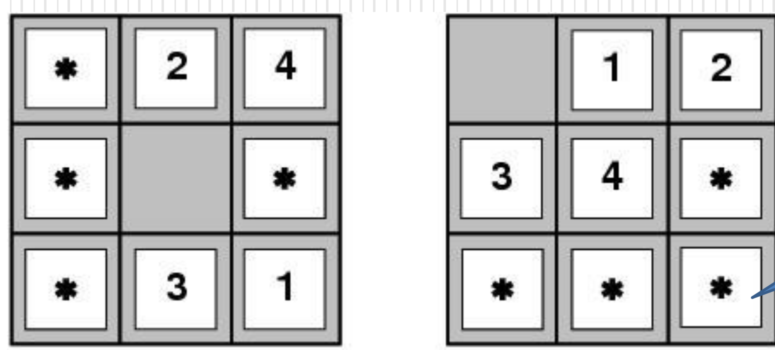
Fig. A subproblem of the 8-puzzle instance. The task is to get the tiles 1,2,3, and 4 into their correct positions without worrying about other tiles

Heuristics

- Pattern databases (PDB) store the exact solution costs for every possible subproblem instance (every possible configuration of four tiles and the blank).
  - The complete heuristic is constructed using the patterns in the DB

PDB consists of the set of all patterns which can be obtained by permutations of a target pattern

Pattern: partial specification of state i.e. tiles occupying certain states are unspecified



Start State   Goal State

Target Pattern: partial specification of goal state

Heuristics

- Learning Heuristics from Experience
  - Allow the agent to learn heuristic function h(n) from experience.
  - Experience – solving lots of 8-puzzles
  - Each optimal solution to an 8-puzzle problem = examples from which h(n) can be learned.
  - Example : (state from solution path, actual cost of solution path from that point)
  - Apply a learning algorithm to such examples to construct a function h(n) that predicts solution costs for other states that arise during search.

state1	**g(state1)**
state2	**g(state2)**

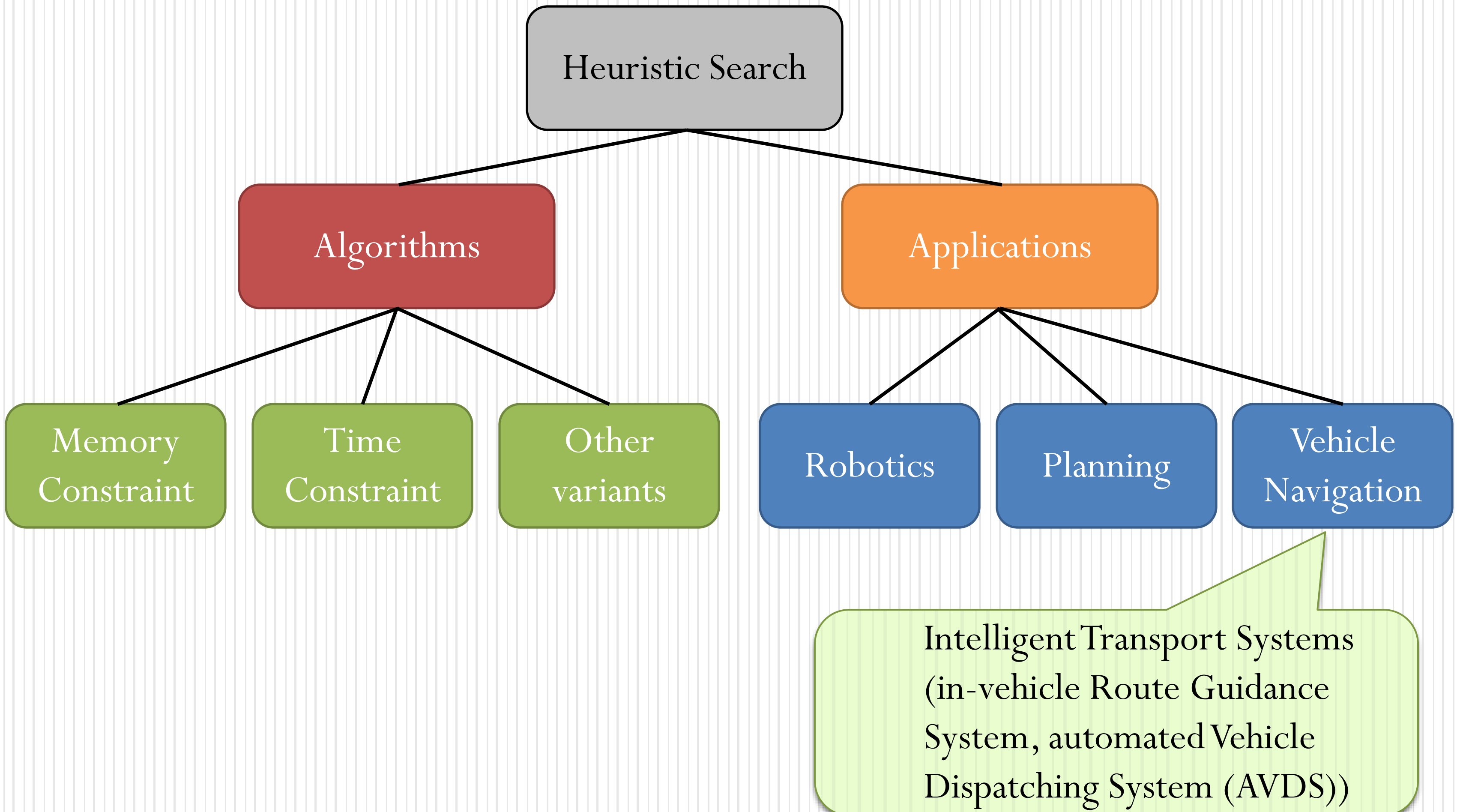
feature 11	**feature 12**	**g(state1)**
feature 21	**feature 22**	**g(State2)**

Heuristics

- Could try to learn a heuristic function based on "features" rather than state itself
  - For example, $x1(n)$ = number of misplaced tiles, $x2(n)$ = number of goal-adjacent-pairs that are currently adjacent
  - $h(n) = w_1\ x1(n) + w_2\ x2(n)$
    - Weights could be learned to identify which features are predictive of path cost
- **More than one heuristic for the same problem?**
  - If for any node n, $h_2(n) > h_1(n)$ then we say that $h_2(n)$ dominates $h_1(n)$
  - use h_2 provided it is consistent and that the computation time for the heuristic is not too long.
  - Given h_1, h_2, ..h_m, if none of them dominates the other then use

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

Heuristic search: Trends



Summary

- We discussed various informed search strategies that an agent can use to select actions before execution.
- Heuristics and heuristic (informed) search: Greedy Best-first search, A* search, IDA* , RBFS
- Performance of search strategies in terms of completeness, optimality, time complexity, and space complexity.
- How to generate admissible heuristics?