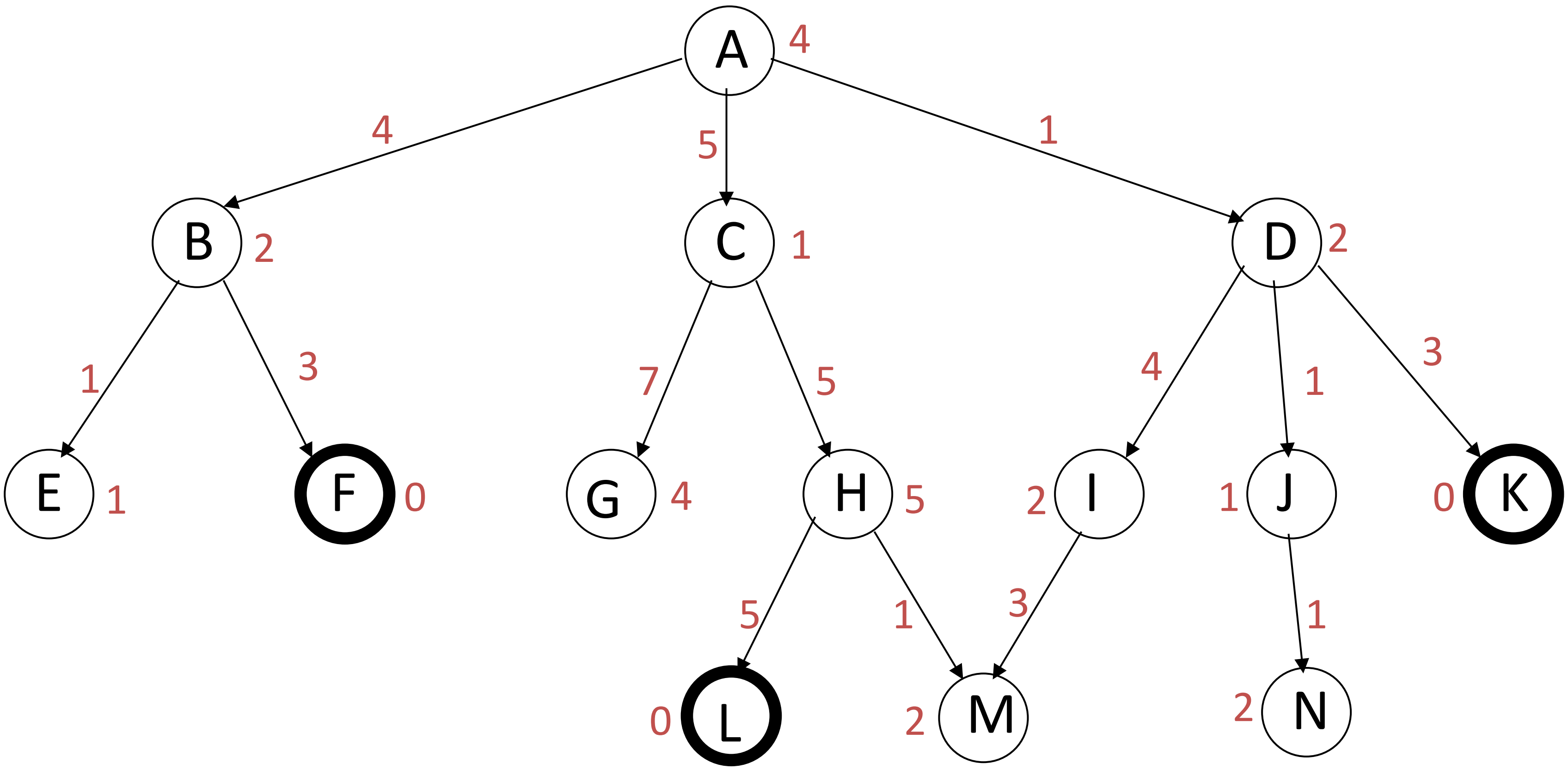
Consider the search graph below. The **h** value of a node is given adjacent to that node. The actual cost of traversing an arc is given adjacent to that arc. Node **A** is the start/initial state. Nodes **F, K, and L** are goals. Leaf states/nodes have no successors.



Give the order in which nodes are visited (i.e., checked for goalness) by each of the following search strategies. In the case of two or more nodes with the same evaluation score on the frontier, break the tie by visiting the nodes from left-to-right as the nodes appear in the graph above.

Lowest-Cost-First Search: ______________________________

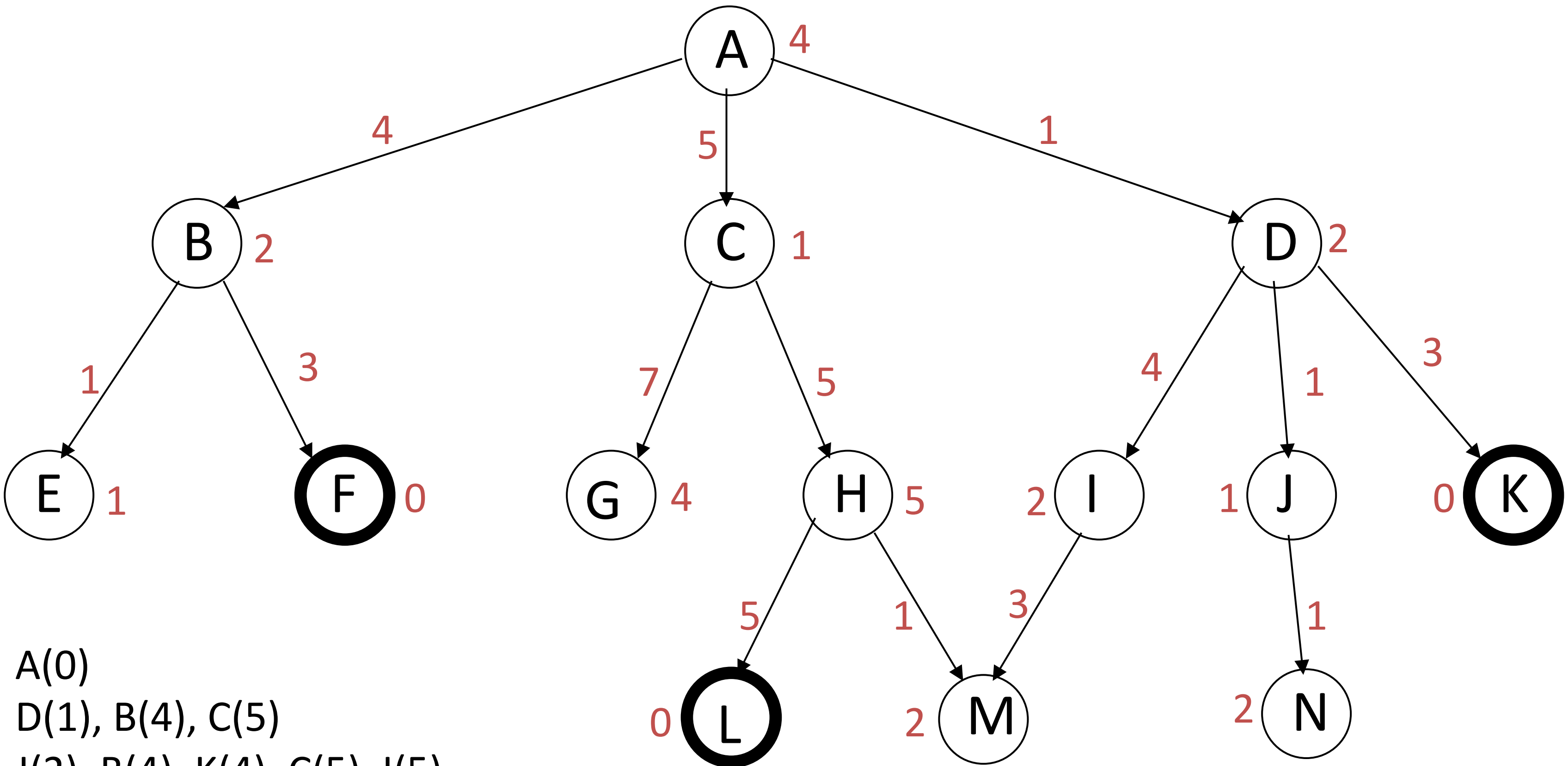
Heuristic Depth-First Search: ______________________________

Greedy Best-First Search: ______________________________

A*: ______________________________

Is the heuristic admissible? _______

Lowest-Cost-First Search



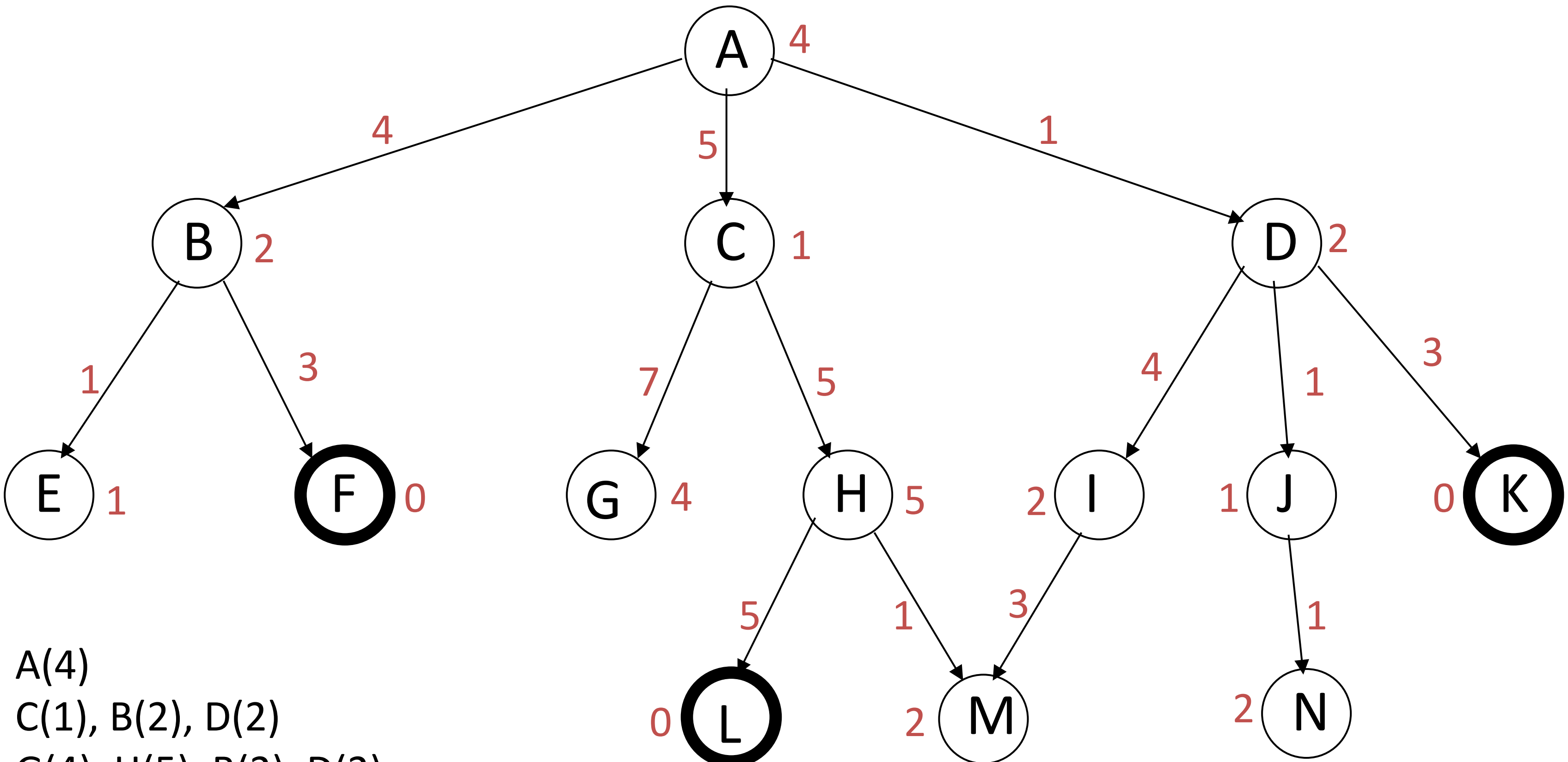
A(0)
D(1), B(4), C(5)
J(2), B(4), K(4), C(5), I(5)
N(3), B(4), K(4), C(5), I(5)
B(4), K(4), C(5), I(5)
K(4), E(5), C(5), I(5), F(7)

A, D, J, N, B, K

Frontier is a priority queue organized by Cost. Use Cost of the path to a node. For example Cost(E) is shorthand for Cost(A→B→E) = 4 + 1

Cost(M) is ambiguous in this case, but would typically mean the cost of the least-cost-path among those paths found so far – this probably isn't relevant to this problem (unless your answer went off on a tangent)

Heuristic Depth-First Search

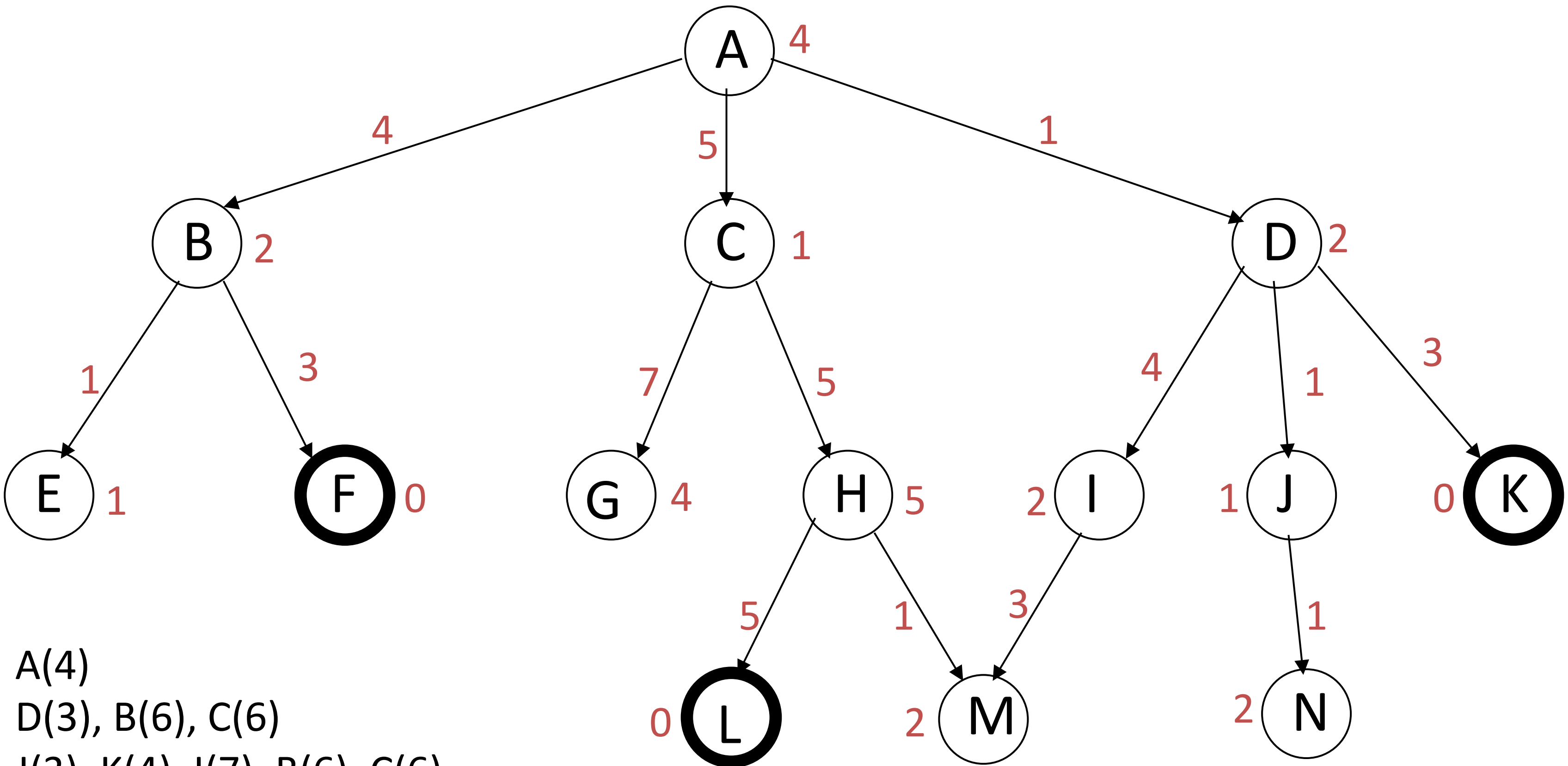


A(4)
C(1), B(2), D(2)
G(4), H(5), B(2), D(2)
H(5), B(2), D(2)
L(0), M(2), B(2), D(2)

A, C, G, H, L

Frontier is a stack, but each set of new neighbors is pushed on according to its h value. H(A) is 4; h(D) is 2; h(H) is 5, etc.

Lets make up a search strategy: Total-Cost Depth-First Search

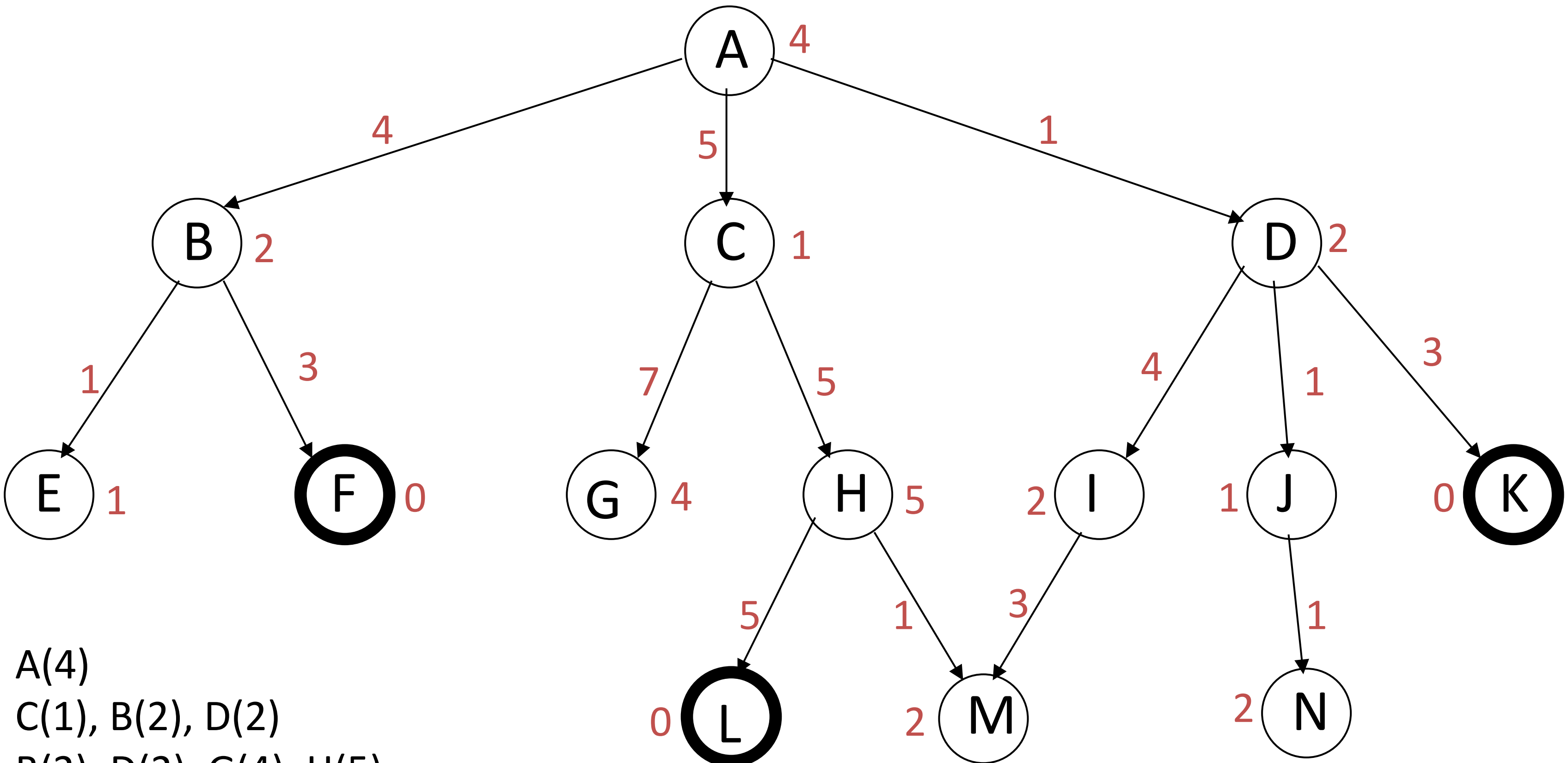


A(4)
D(3), B(6), C(6)
J(3), K(4), I(7), B(6), C(6)
N(5), K(4), I(7), B(6), C(6)
K(4), I(7), B(6), C(6)

A. D, J, N, K

Not part of the exercise and not described in textbook, but just like Heuristic Depth first search except that new neighbors are pushed on the stack by their Cost()+h() values

Greedy Best-First Search

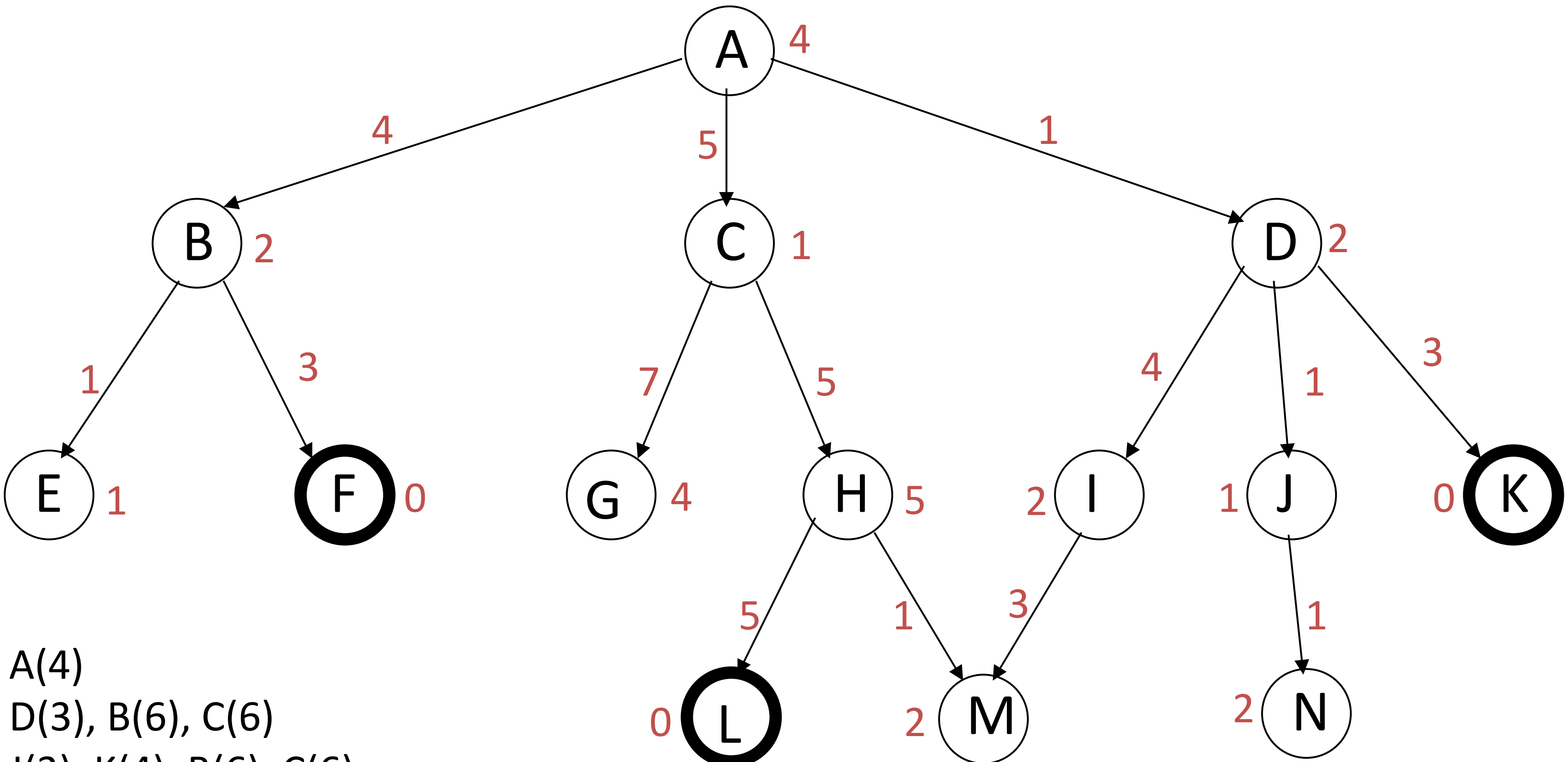


A(4)
C(1), B(2), D(2)
B(2), D(2), G(4), H(5)
F(0), E(1), D(2), G(4), H(5)

A, C, B, F

Frontier is a priority queue organized by h values.

A* Search

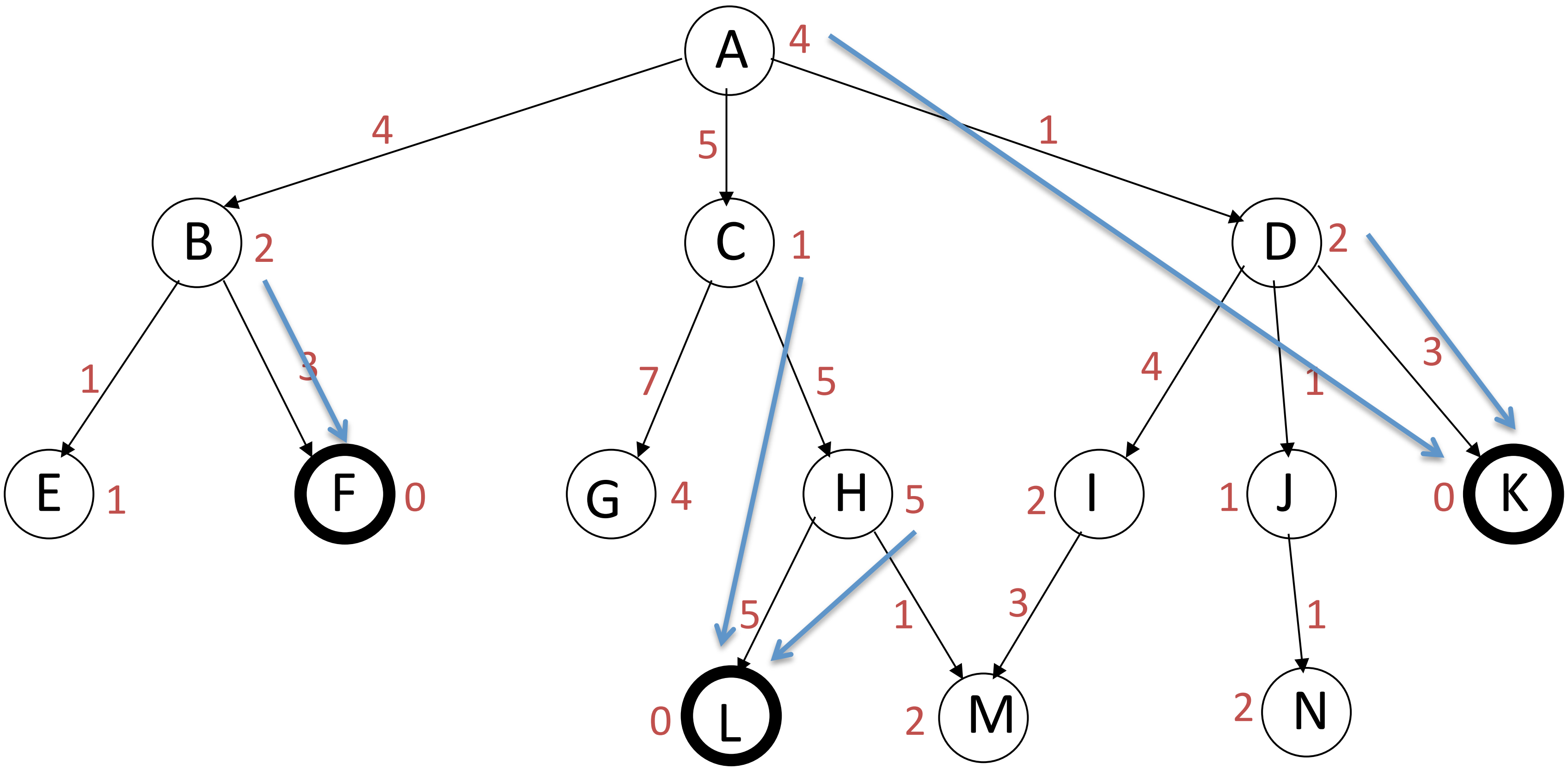


A(4)
D(3), B(6), C(6)
J(3), K(4), B(6), C(6)
K(4), N(5), B(6), C(6)

A, D, J, K

Frontier is a priority queue organized by f( ) = Cost( ) + h( ) values.

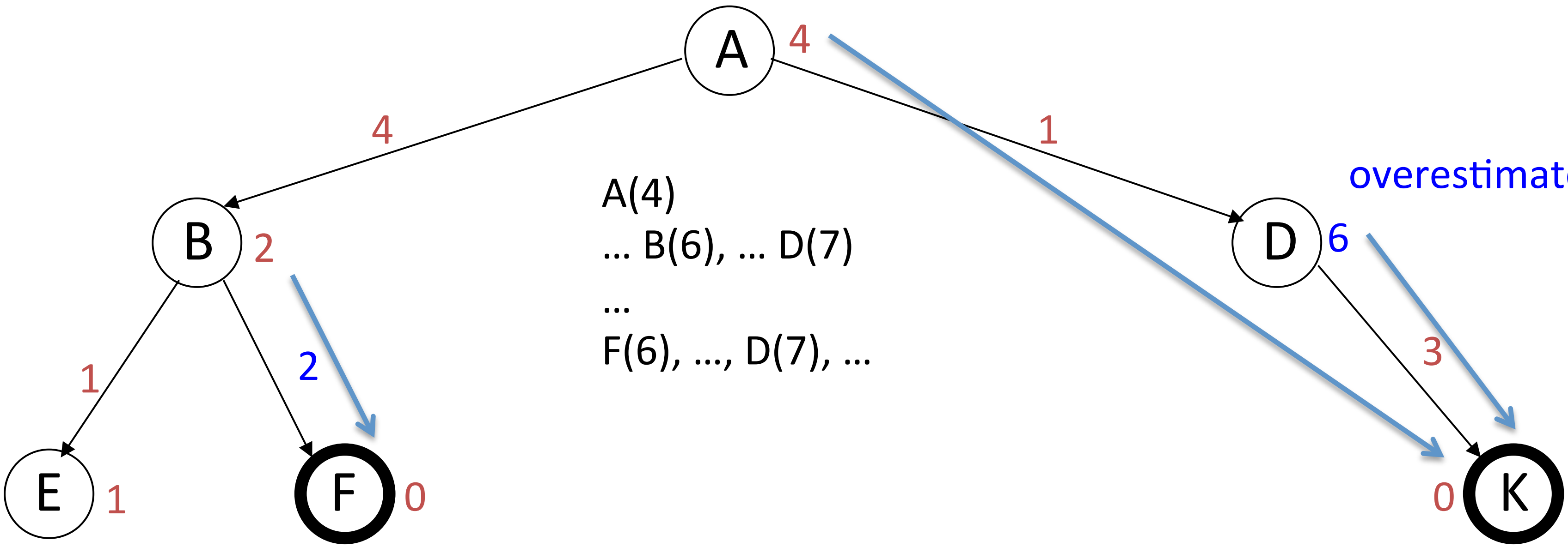
Consider the search graph below. The **h** value of a node is given adjacent to that node. The actual cost of traversing an arc is given adjacent to that arc. Node **A** is the start/initial state. Nodes **F, K, and L** are goals. Leaf states/nodes have no successors.



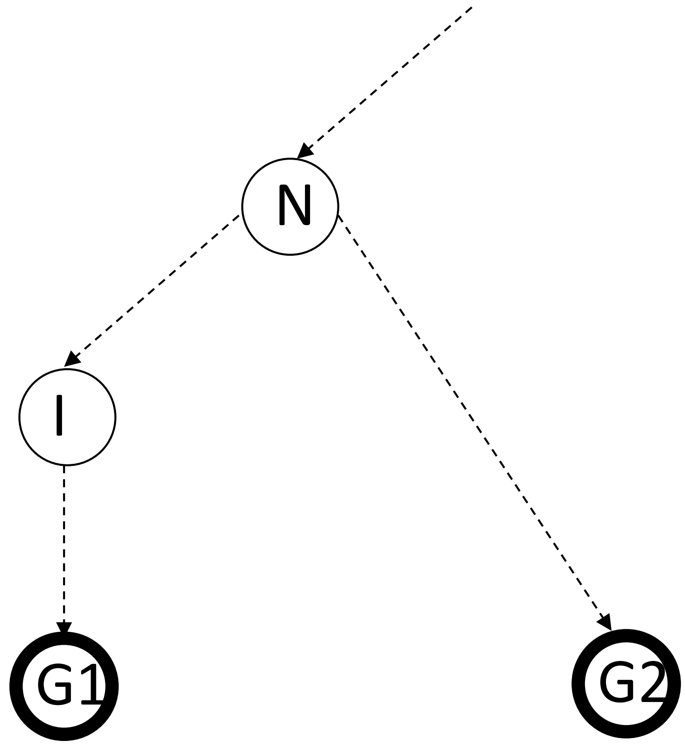
Is the heuristic admissible? Yes – the heuristic never overestimates the cost to the "nearest" goal from an intermediate state

Guarantee under admissibility of A*: a least cost goal will always be found

In contrast, suppose that h not admissible – e.g., it overestimates cost to goal at node D, then a more expensive goal F might be visited before the least cost goal is ever expanded and placed on the frontier priority queue



General case under admissibility. Remember h(goal) = 0.



f(I) = cost(I) + h(I)

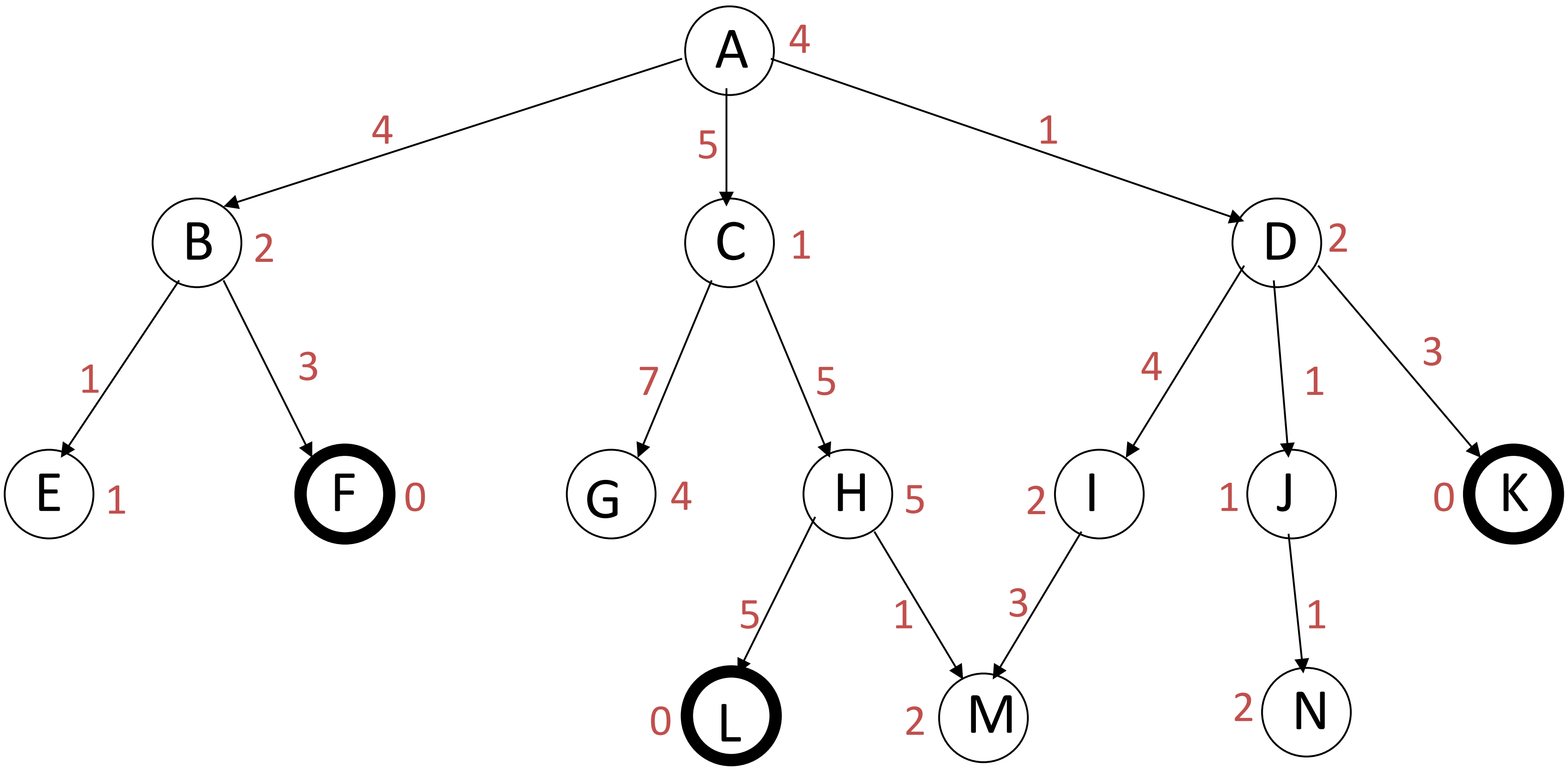
f(G1) = cost(G1) + h(G1) = cost(G1)

f(G2) = cost(G2) + h(G2) = cost(G2)

Suppose f(G1) < f(G2), then we want G1 to be found in preference to G2

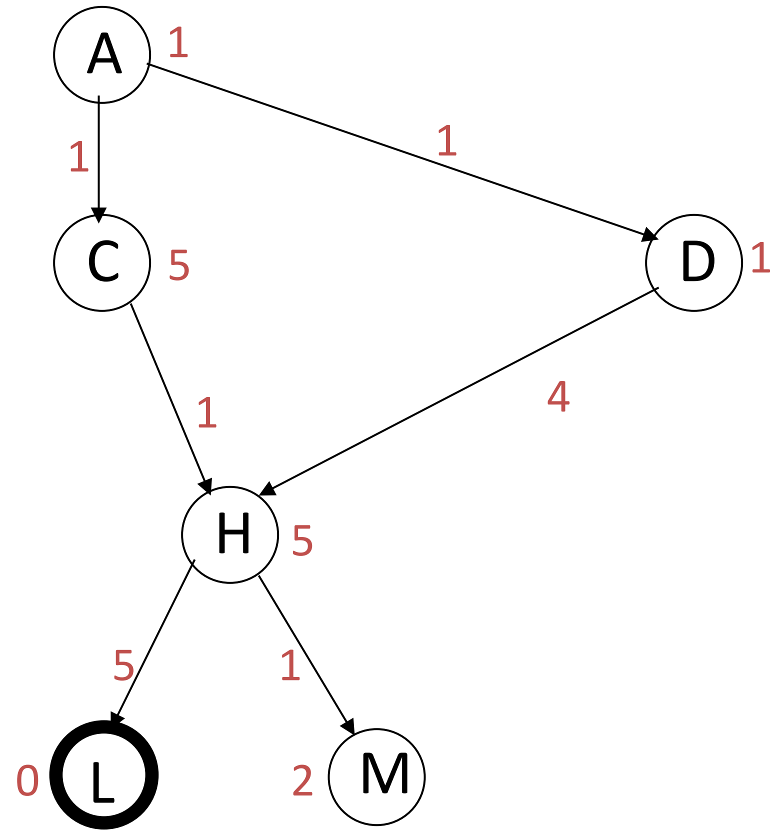
Remember, h is admissible,
so f(I) <= f(G1) <= f(G2) (because cost(I) + h(I) <= cost(I) + cost(I→G1)) = cost(I) + (cost(G1) – cost(I)), so 'I' will be visited before G2

Aside: the heuristic violates the monotone restriction – h is not monotonic non-decreasing in all cases. For example, h(C) < h(H)



The implications of this violation are

- that search is misled down wrongfully optimistic paths,
- a non-goal node can be first visited along a path that is more costly than exists to that node



The implications of this violation are

- that search is misled down wrongfully optimistic paths,
- a non-goal node can be first visited along a path that is more costly than exists to that node