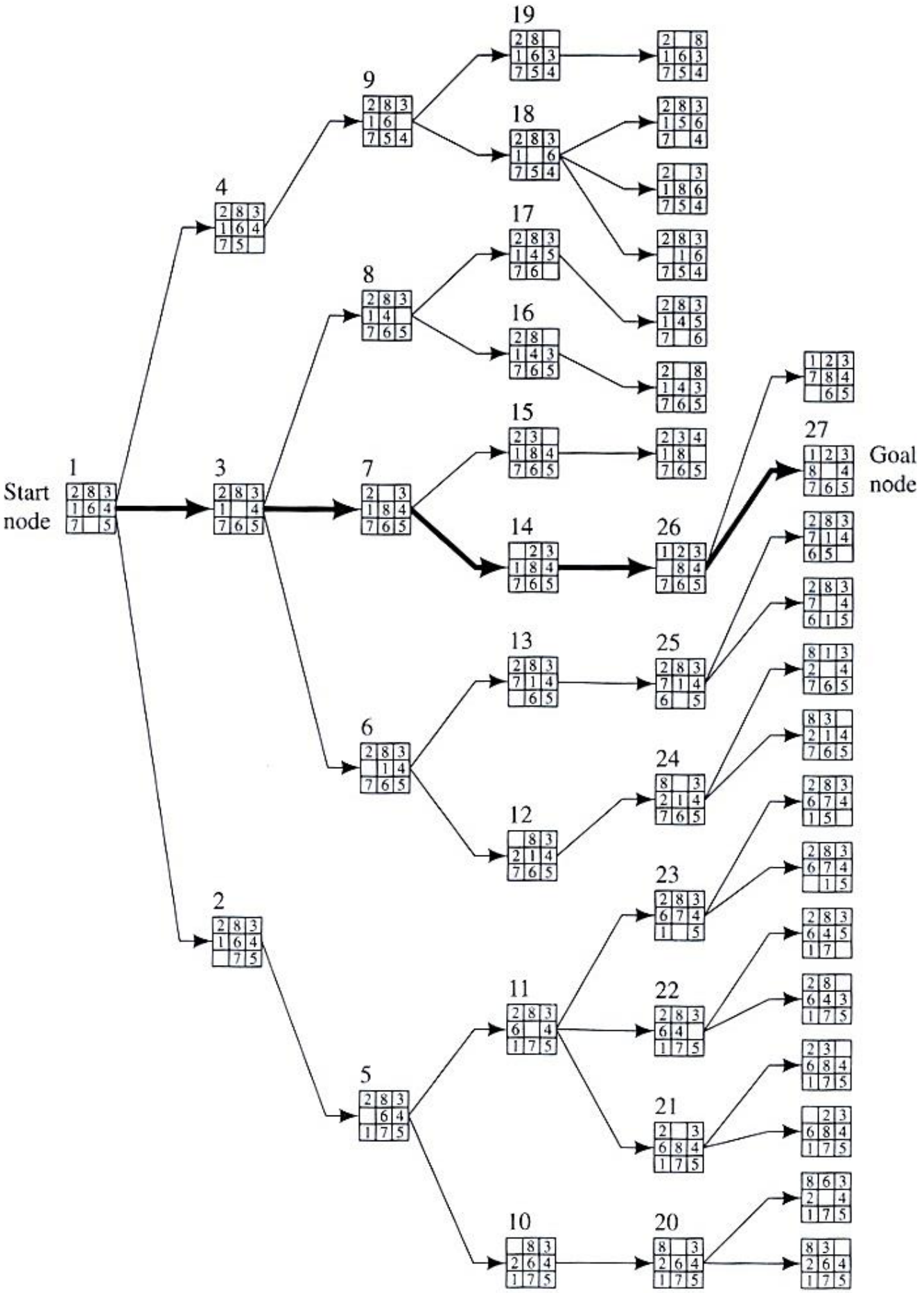
Search – The process of looking for a sequence of actions (i.e. solution) that reaches the goal state.

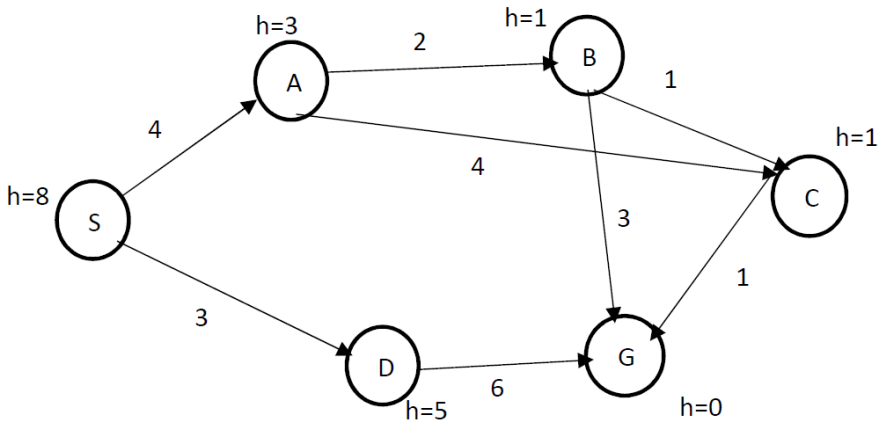


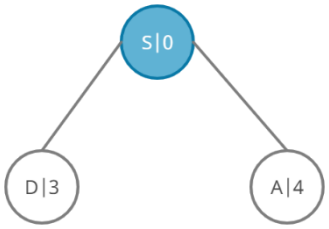
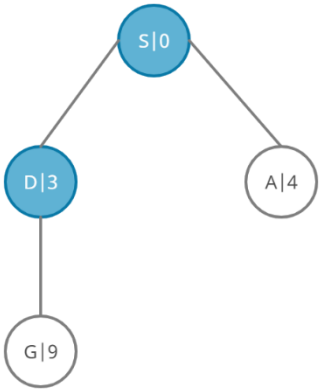
Search Strategy:

1) **Uninformed/Blind Search:** It has *no additional information* about the states beyond the problem definition.
   - Breadth-first search (BFS)
   - Uniform-cost search (UCS)
   - Depth-first search (DFS)
   - Depth-limited search (DLS)
   - Iterative deepening depth-first search (ID DFS)
   - Bidirectional search etc.

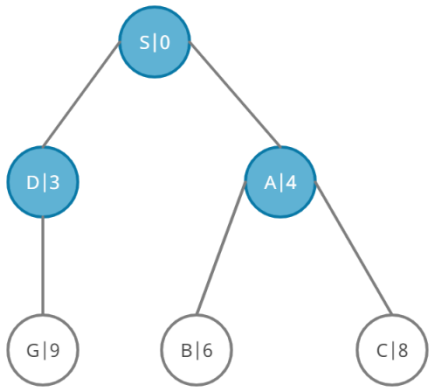
Uniform-cost search (UCS) – evaluation function, **f(n) = g(n) = cost to reach node n from root**

- Expands the node with the lowest path cost, **g(n).**
- Goal test is applied to a node when it is selected for expansion.
- Extra checkpoint is added in case a better path is found to a node currently on the queue (frontier).

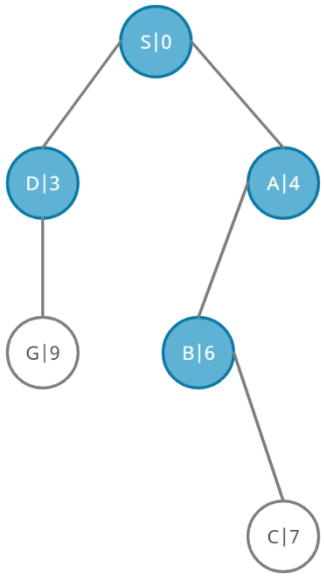


Step 1: (Initialization) E = { } Q = { (S, 0, NIL) }	
Step 2: popped_node = (S, 0, NIL) != Goal, G E = { (S, 0, NIL) } Q = { (D, 3, S), (A, 4, S) }	
Step 3: popped_node = (D, 3, S) != Goal, G E = { (S, 0, NIL), (D, 3, S) } Q = { (A, 4, S), (G, 9, D) }	

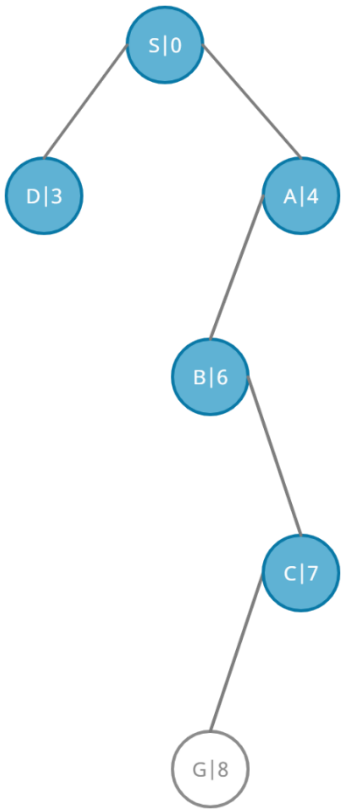
Step 4:
popped_node = (A, 4, S) != Goal, G
E = { (S, 0, NIL), (D, 3, S), (A, 4, S) }
Q = { (B, 6, A), (C, 8, A), (G, 9, D) }



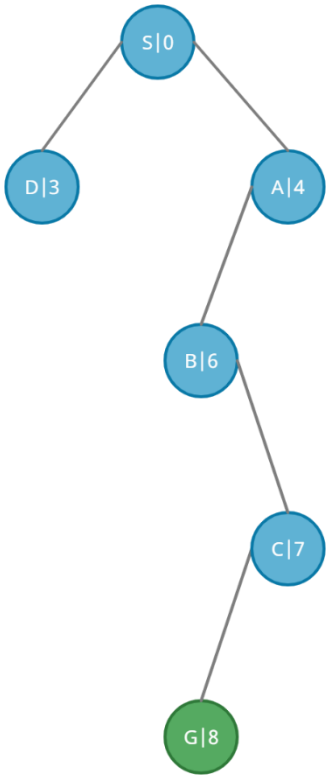
Step 5:
popped_node = (B, 6, A) != Goal, G
E = { (S, 0, NIL), (D, 3, S), (A, 4, S), (B, 6, A) }
Q = { (C, 7, B), (G, 9, D) }



Step 6:
popped_node = (C, 7, B) != Goal, G
E = { (S, 0, NIL), (D, 3, S), (A, 4, S), (B, 6, A), (C, 7, B) }
Q = { (G, 8, C) }



Step 7:
popped_node = (G, 8, C) = Goal, G (STOP)

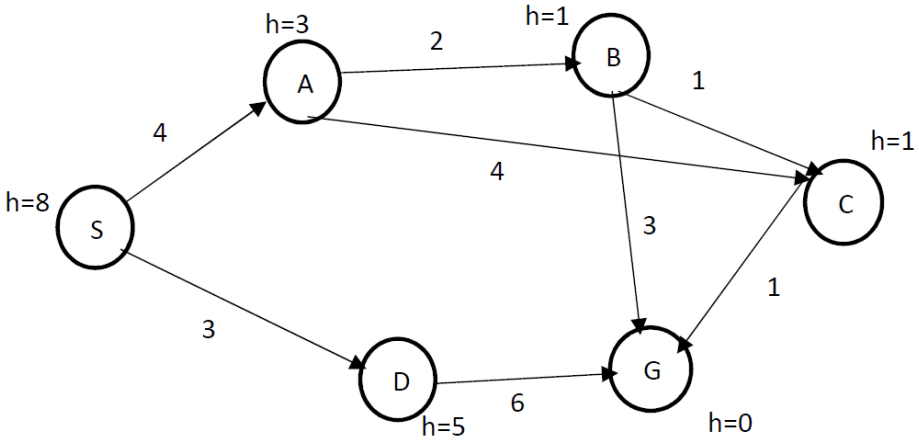


Optimal Path: (G, 8, C) ←(C, 7, B) ←(B, 6, A) ←(A, 4, S) ←(S, 0, NIL)

2) **Informed/Heuristic Search:** It uses *problem-specific knowledge* beyond the definition of the problem itself.
   - Greedy best-first search
   - A* search

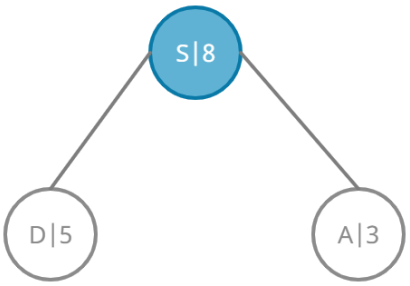
Greedy best-first search – evaluation function **f(n) = h(n) = estimated cost of the cheapest path from n to G**

- Tries to expand the node that is closest to the goal.
- Identical to UCS except it evaluates nodes by using the heuristic function, **h(n).** Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm.
- Final solution can be optimal or not.

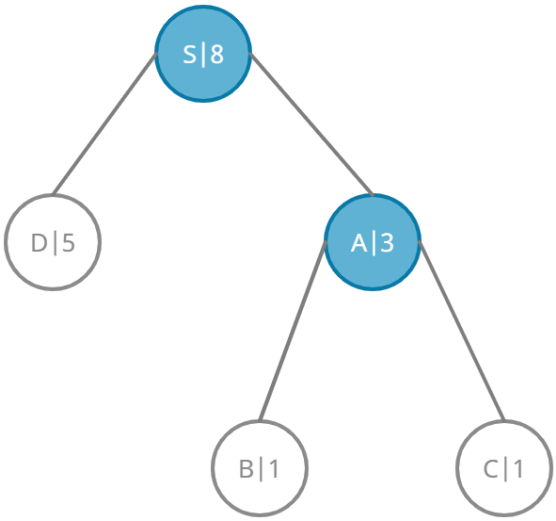


Step 1: (Initialization)
E = {}
Q = { (S, 8, NIL) }

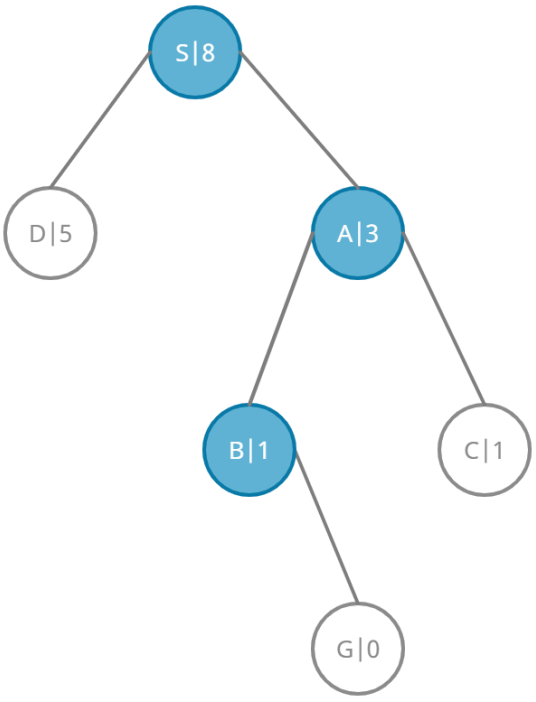
Step 2:
popped_node = (S, 8, NIL) != Goal, G
E = { (S, 8, NIL) }
Q = { (A, 3, S), (D, 5, S) }



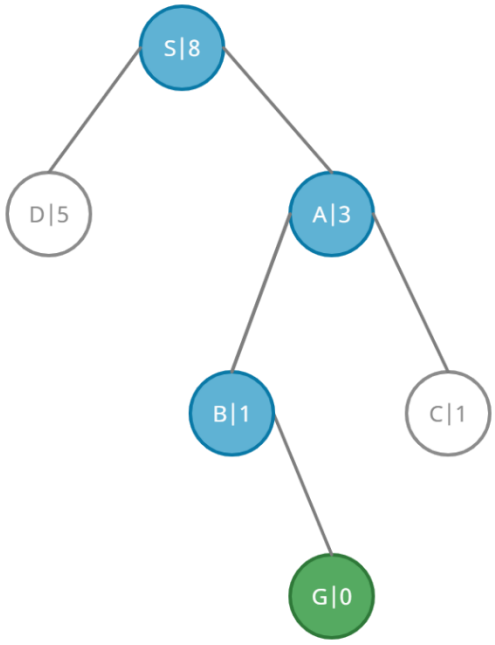
Step 3:
popped_node = (A, 3, S) != Goal, G
E = { (S, 8, NIL), (A, 3, S) }
Q = { (B, 1, A), (C, 1, A), (D, 5, S) }



Step 4:
popped_node = (B, 1, A) != Goal, G
E = { (S, 8, NIL), (A, 3, S), (B, 1, A) }
Q = { (G, 0, B), (C, 1, A), (D, 5, S) }



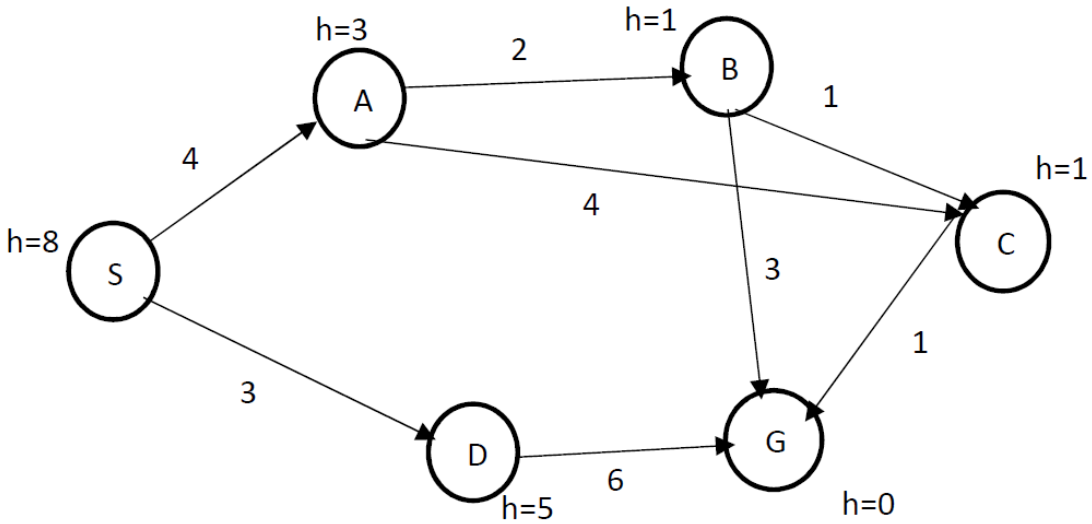
Step 5:
popped_node = (G, 0, B) = Goal, G (STOP)



Sub-optimal Path: (G, 0, B) ←(B, 1, A) ←(A, 3, S) ←(S, 8, NIL)

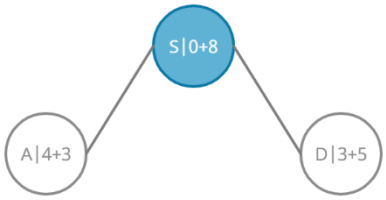
A* Search - evaluation function, **f(n) = g(n) + h(n)**
= actual cost to reach node n from the root
+ estimated cost of the cheapest path from node n to goal G

- Evaluates nodes by combining the actual cost to reach the node, **g(n)** and the estimated cost to reach the goal from the node, **h(n)**
- Identical to UCS except it uses **g(n) + h(n)** instead of **g(n)**.
- Optimality condition:
  - For tree search, **h(n)** needs to be admissible i.e. h(n) <= true minimum cost from n to goal, h*(n)
  - For graph search, **h(n)** needs to be consistent i.e. h(n) <= c(n, a, n') + h(n'), n' a successor or n.

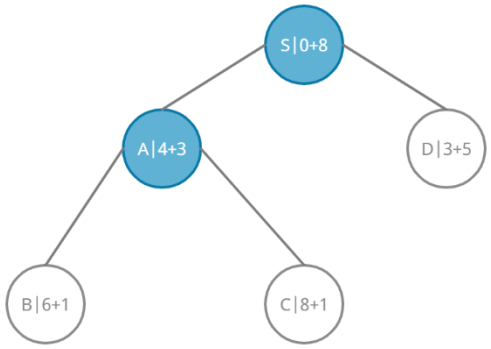


Step 1: (Initialization)
E = { }
Q = { (S, t=8, a=0, NIL) }

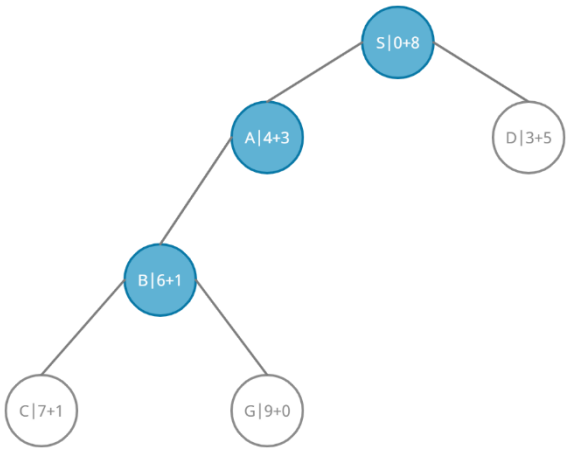
Step 2:
popped_value = (S, t=8, a=0, NIL)
E = { (S, t=8, a=0, NIL) }
Q = { (A, t=7, a=4, S), (D, t=8, a=3, S) }



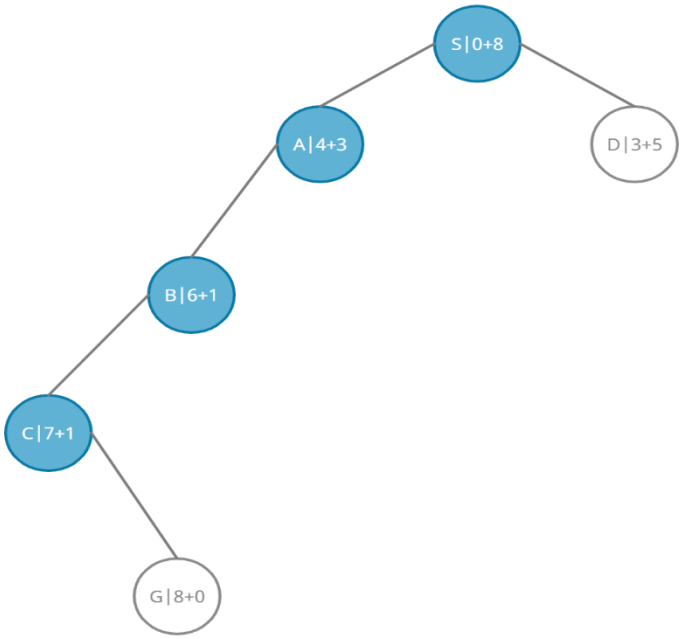
Step 3:
popped_value = (A, t=7, a=4, S)
E = { (S, t=8, a=0, NIL), (A, t=7, a=4, S) }
Q = { (B, t=7, a=6, A), (D, t=8, a=3, S), (C, t=9, a=8, A) }



Step 4:
popped_value = (B, t=7, a=6, A)
E = { (S, t=8, a=0, NIL), (A, t=7, a=4, S), (B, t=7, a=6, A) }
Q = { (C, t=8, a=7, B), (D, t=8, a=3, S), (G, t=9, a=9, B) }



Step 5:
popped_value = (C, t=8, a=7, B)
E = { (S, t=8, a=0, NIL), (A, t=7, a=4, S),
(B, t=7, a=6, A), (C, t=8, a=7, B) }
Q = { (D, t=8, a=3, S), (G, t=8, a=8, C)}

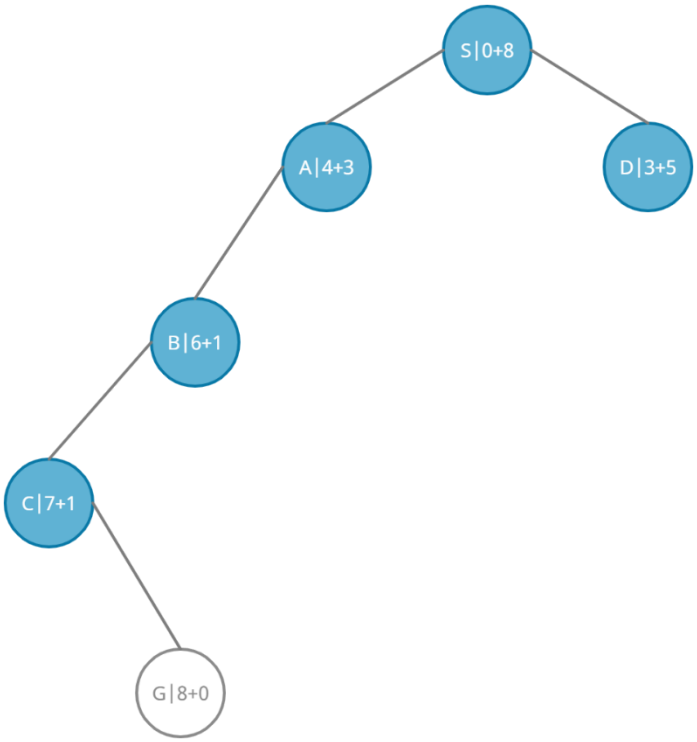


Step 6:

popped_value = (D, t=8, a=3, S)

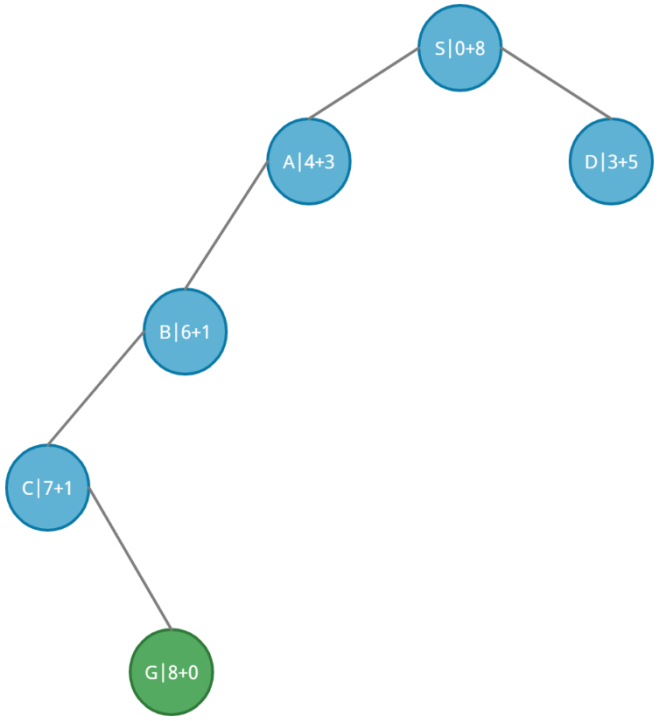
E = { (S, t=8, a=0, NIL), (A, t=7, a=4, S),
(B, t=7, a=6, A), (C, t=8, a=7, B),
(D, t=8, a=3, S) }

Q = { (G, t=8, a=8, C) } #no change



Step 7:

popped_value = (G, t=8, a=8, C) = Goal, G (STOP)



Optimal Path: (G, t=8, a=8, C) ← (C, t=8, a=7, B) ← (B, t=7, a=6, A) ← (A, t=7, a=4, S) ← (S, t=8, a=0, NIL)