Module 04: Graph Algorithms

Analysis and Design of Algorithms

Ammar Sherif

Nile University

3rd April, 2022

What are Graphs?

Graphs

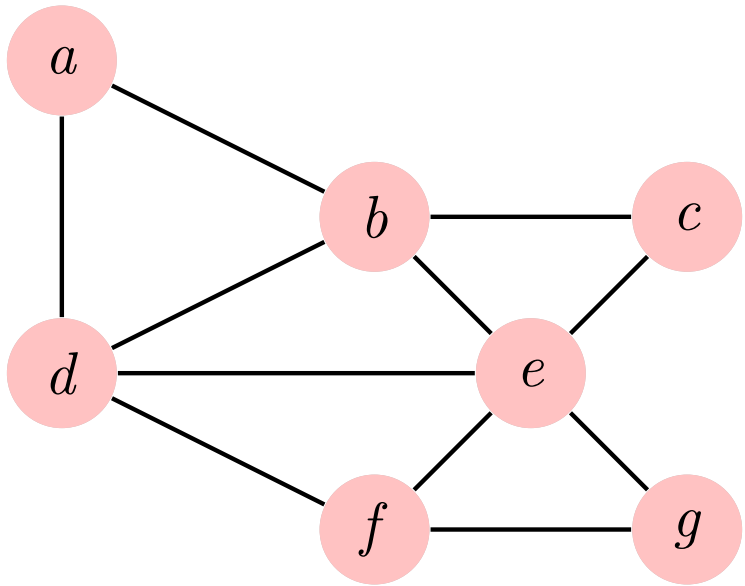
From a mathematical perspective, consist of a nodes/vertices and edges.

What are Graphs?

Graphs

From a mathematical perspective, consist of a nodes/vertices and edges.

- Nodes/Vertices

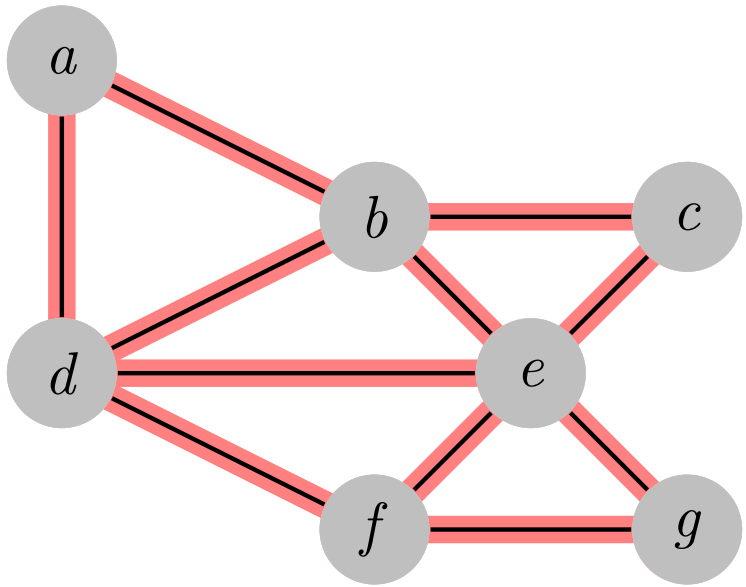


What are Graphs?

Graphs

From a mathematical perspective, consist of a nodes/vertices and edges.

- Nodes/Vertices
- Edges

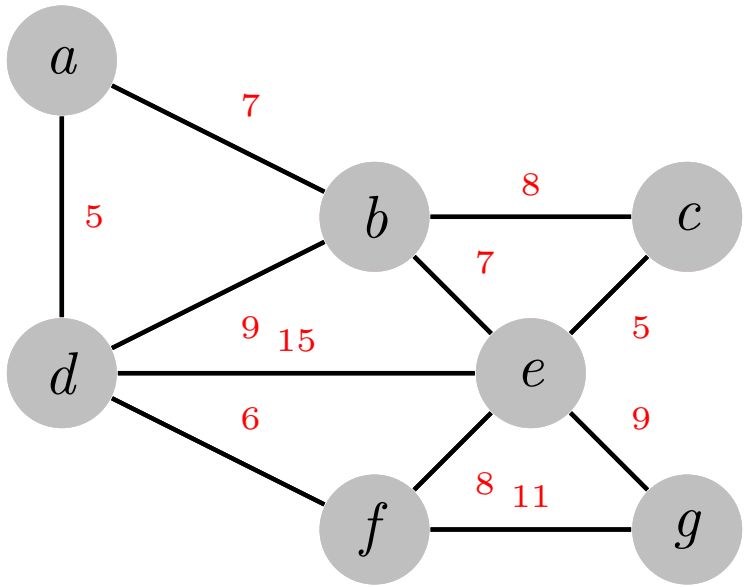


What are Graphs?

Graphs

From a mathematical perspective, consist of a nodes/vertices and edges.

- Nodes/Vertices
- Edges
- Weighted Graph

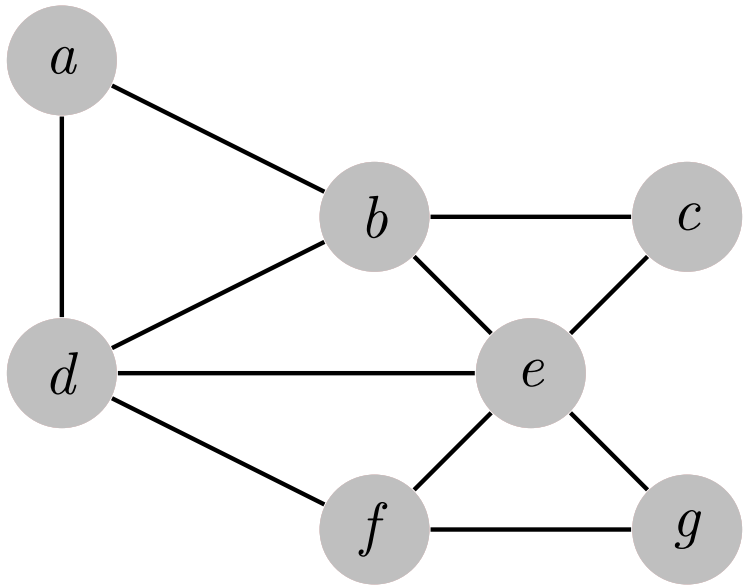


What are Graphs?

Graphs

From a mathematical perspective, consist of a nodes/vertices and edges.

- Nodes/Vertices
- Edges
- Weighted Graph
- Unweighted Graph

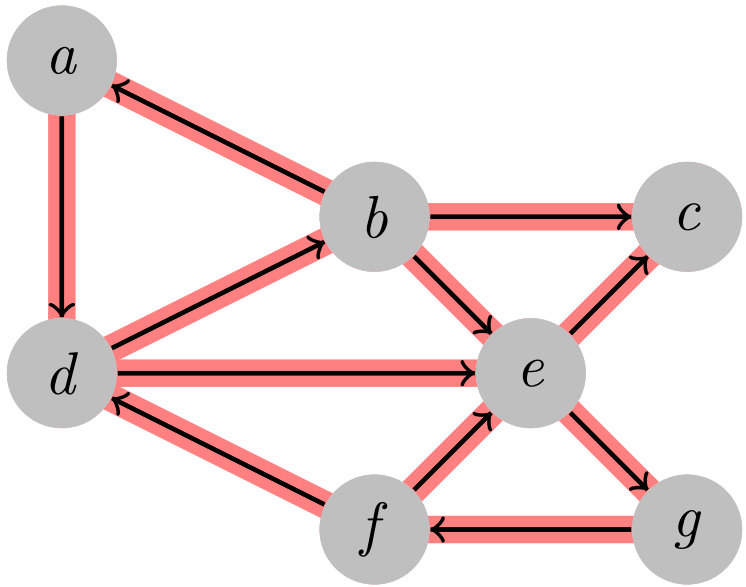


What are Graphs?

Graphs

From a mathematical perspective, consist of a nodes/vertices and edges.

- Nodes/Vertices
- Edges
- Weighted Graph
- Unweighted Graph
- Directed Graph

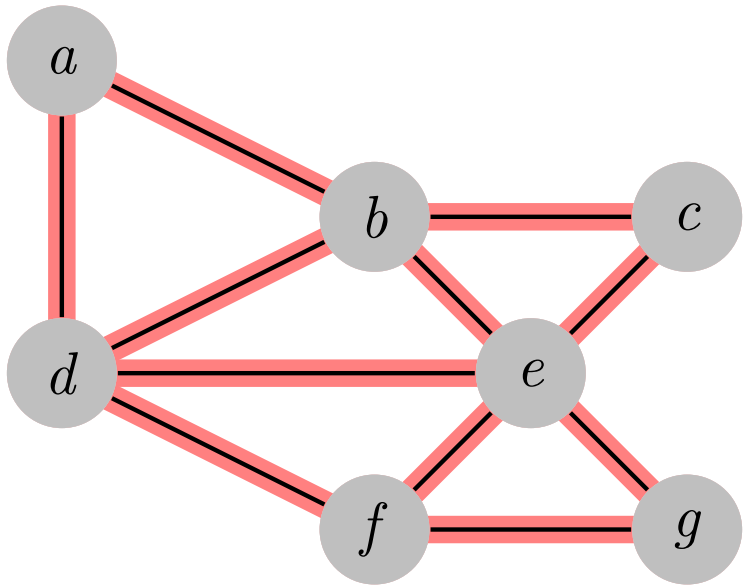


What are Graphs?

Graphs

From a mathematical perspective, consist of a nodes/vertices and edges.

- Nodes/Vertices
- Edges
- Weighted Graph
- Unweighted Graph
- Directed Graph
- Undirected Graph



Why Graphs & Graph Algorithms?

- Shortest Path and Route planning

Why Graphs & Graph Algorithms?

- Shortest Path and Route planning
- Robotics

Why Graphs & Graph Algorithms?

- Shortest Path and Route planning
- Robotics
  - Warehouses

Why Graphs & Graph Algorithms?

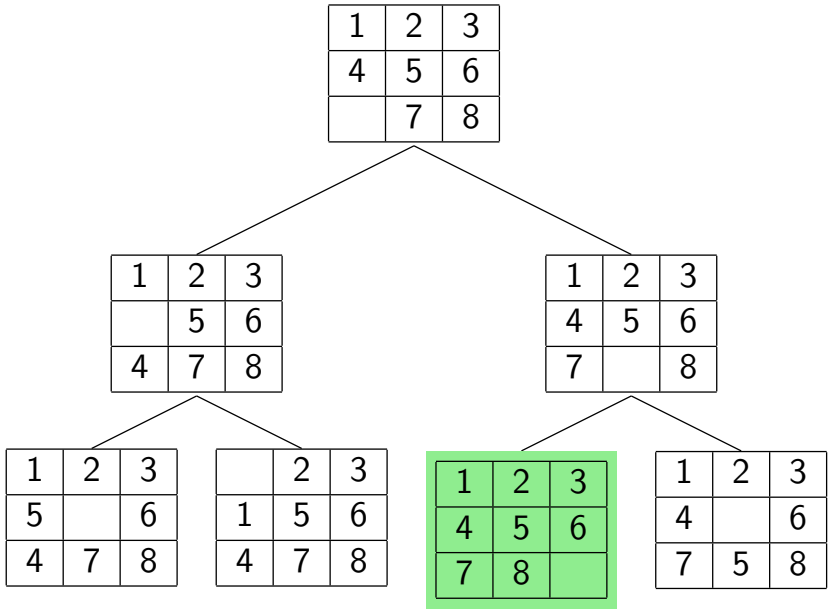
- Shortest Path and Route planning
- Robotics
  - Warehouses
  - Space Robots

Why Graphs & Graph Algorithms?

- Shortest Path and Route planning
- Robotics
  - Warehouses
  - Space Robots
  - Rescue Robots

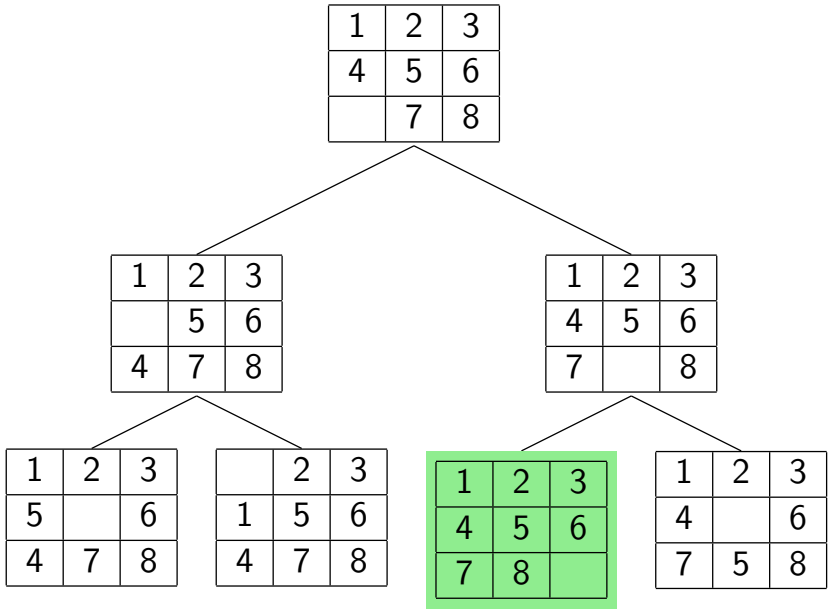
Why Graphs & Graph Algorithms?

- Shortest Path and Route planning
- Robotics
  - Warehouses
  - Space Robots
  - Rescue Robots
- Games



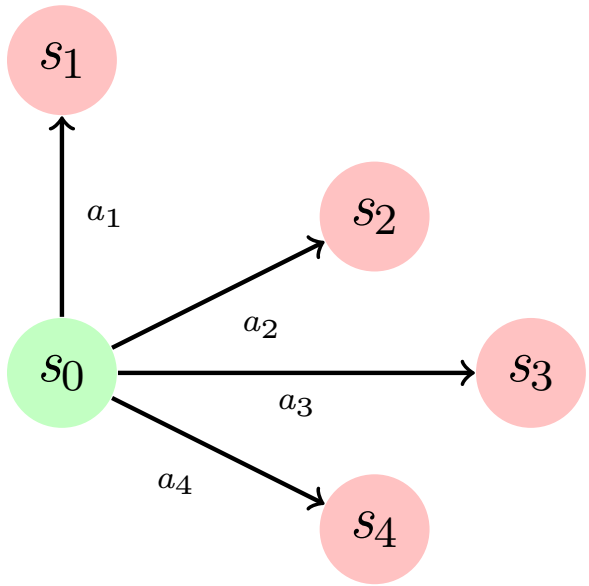
Why Graphs & Graph Algorithms?

- Shortest Path and Route planning
- Robotics
  - Warehouses
  - Space Robots
  - Rescue Robots
- Games
- Optimization Problems



Why Graphs & Graph Algorithms?

- Shortest Path and Route planning
- Robotics
  - Warehouses
  - Space Robots
  - Rescue Robots
- Games
- Optimization Problems
- Any decision-based problem



Graph Representation

Algorithms are implemented via programming, so how to represent graphs?

What our representation should provide?

Graph Representation

Algorithms are implemented via programming, so how to represent graphs?

What our representation should provide?

- Know the neighbors

Graph Representation

Algorithms are implemented via programming, so how to represent graphs?

What our representation should provide?

- Know the neighbors
- Weights of links

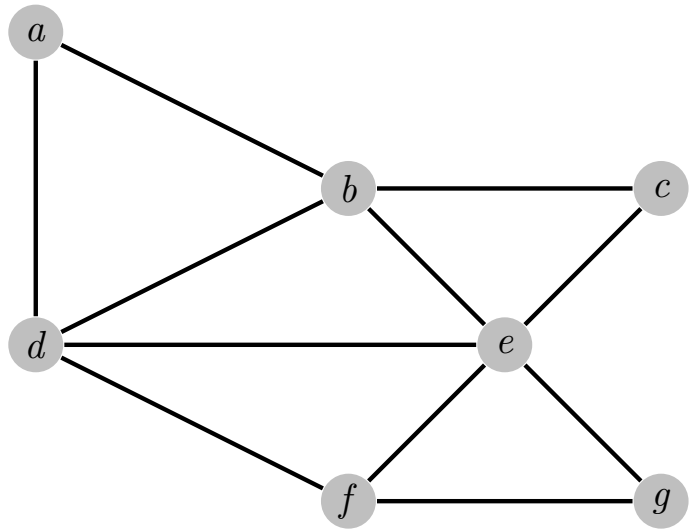
Graph Representation

Algorithms are implemented via programming, so how to represent graphs?

What our representation should provide?

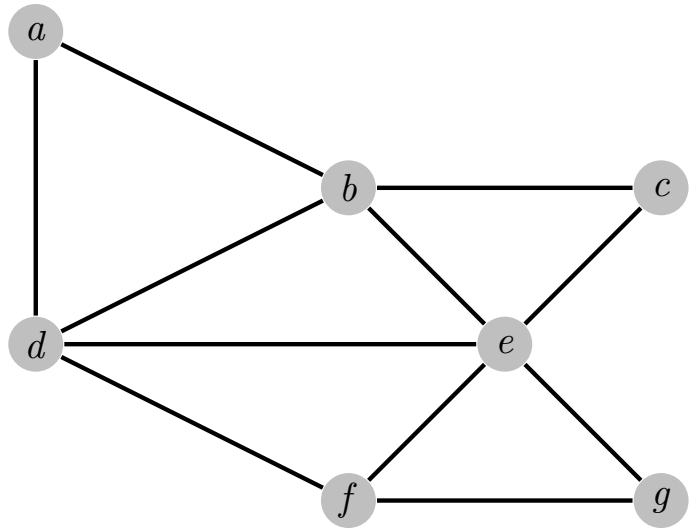
- Know the neighbors
- Weights of links
- list of nodes/edges

Graph Representation



Graph Representation

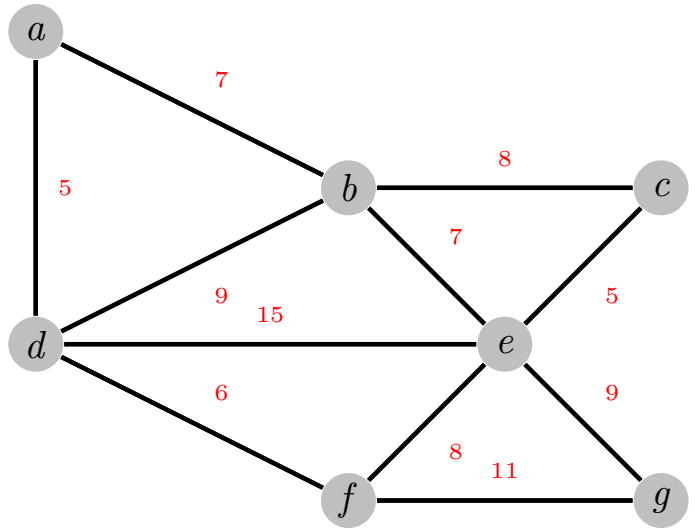
- Adjacency Matrix



	a	b	c	d	e	f	g
a	–	1		1			
b	1	–	1	1	1		
c		1	–		1		
d	1	1		–	1	1	
e		1	1	1	–	1	1
f				1	1	–	1
g					1	1	–

Graph Representation

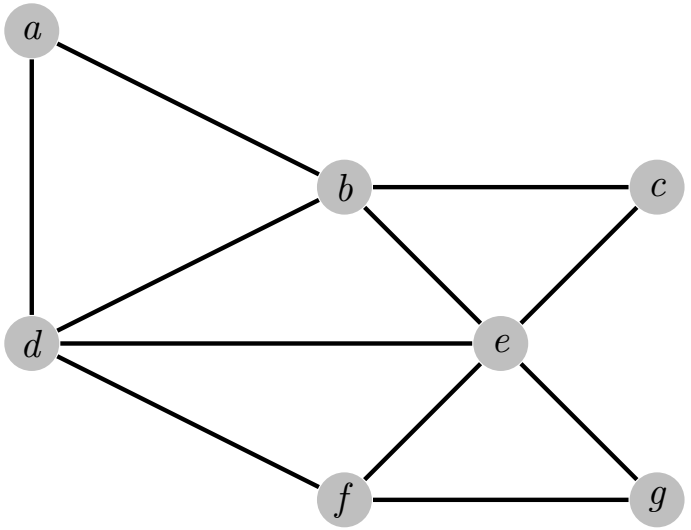
- Adjacency Matrix

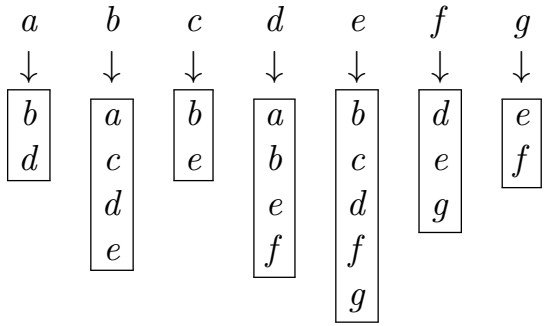


	a	b	c	d	e	f	g
a	–	7		5			
b	7	–	8	9	7		
c		8	–		5		
d	5	9		–	15	8	
e		7	5	15	–	8	9
f				8	8	–	11
g					9	11	–

Graph Representation

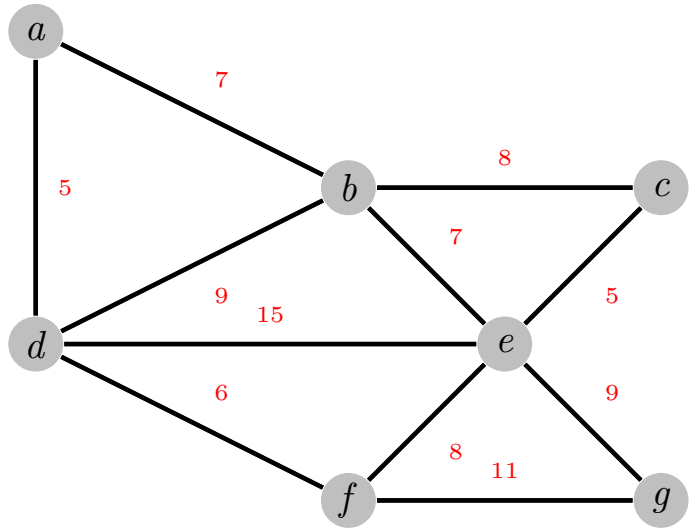
- Adjacency Matrix
- Adjacency List





Graph Representation

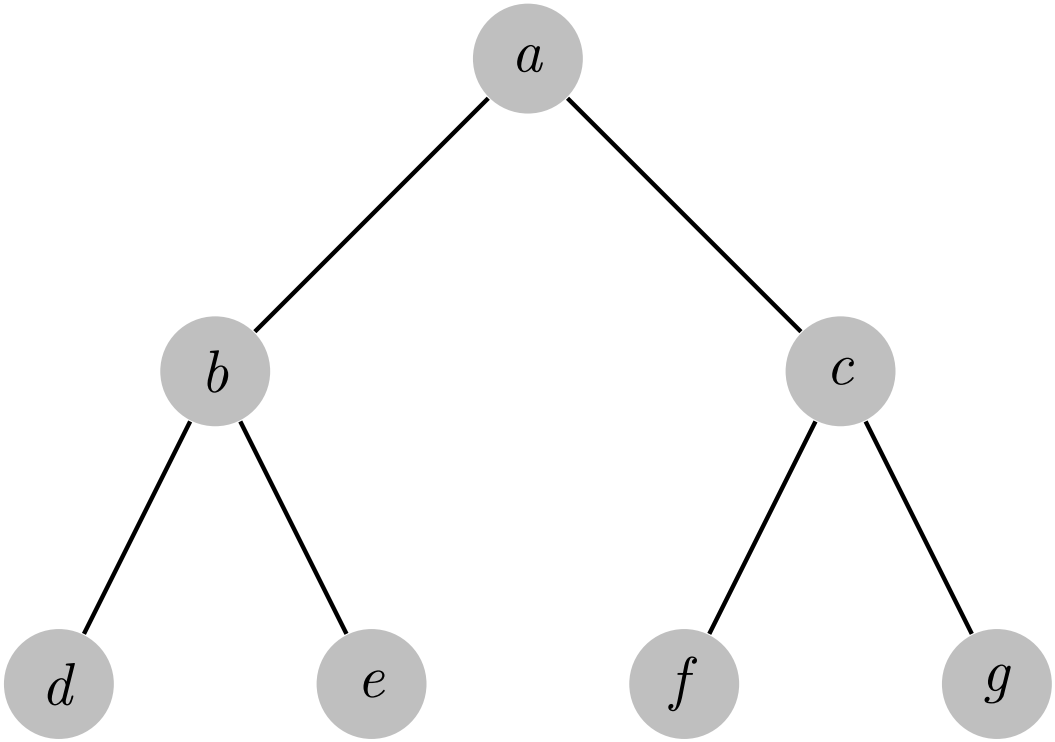
- Adjacency Matrix
- Adjacency List



a	b	c	d	e	f	g
$b:7$	$a:7$	$b:8$	$a:5$	$b:7$	$d:8$	$e:9$
$d:5$	$c:8$	$e:5$	$b:9$	$c:5$	$e:8$	$f:11$
	$d:9$		$e:15$	$d:15$	$g:11$	
	$e:7$		$f:8$	$f:8$		
				$g:9$		

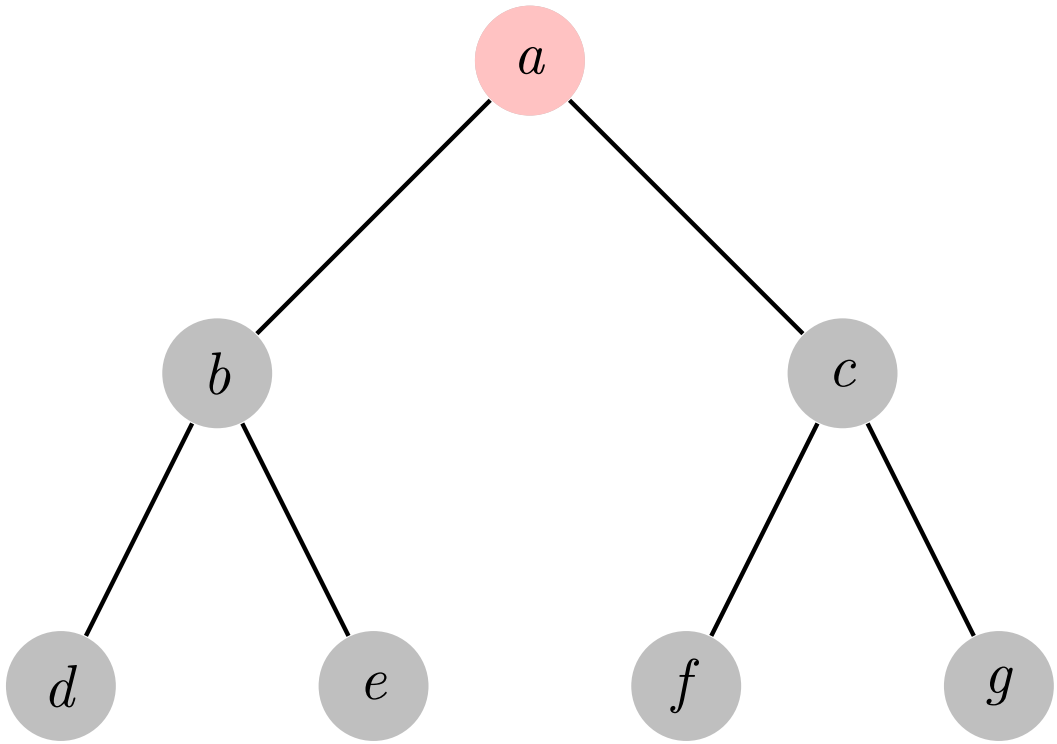
Depth First Search (DFS)

Depth has the max priority



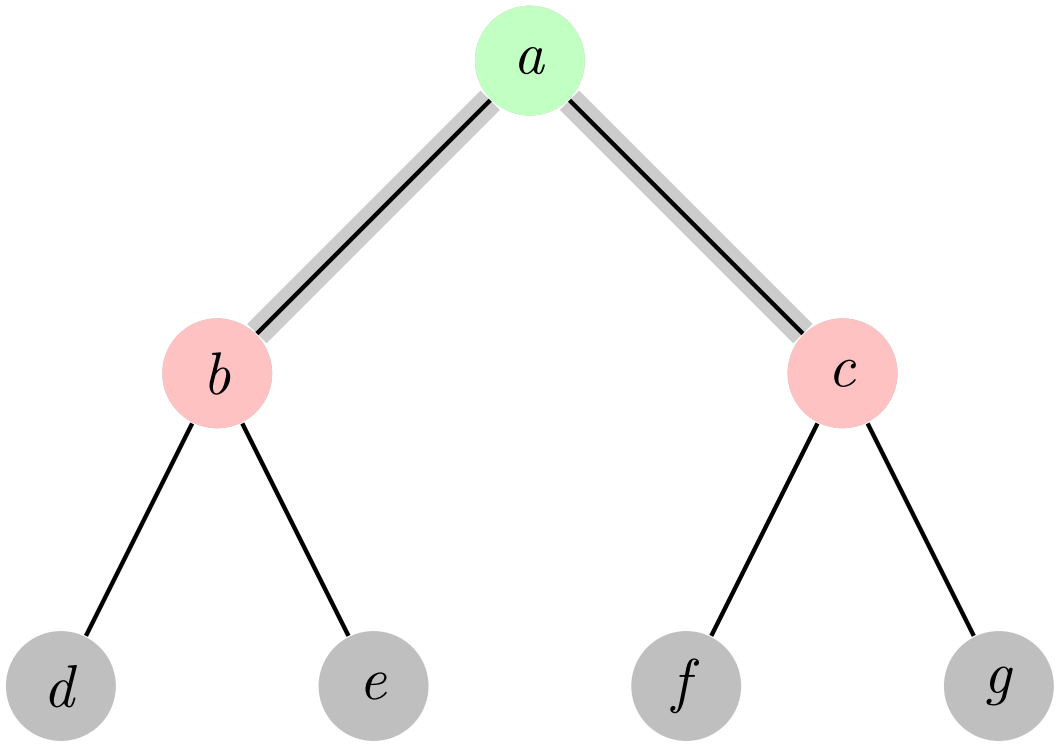
Depth First Search (DFS)

Depth has the max priority



Depth First Search (DFS)

Depth has the max priority

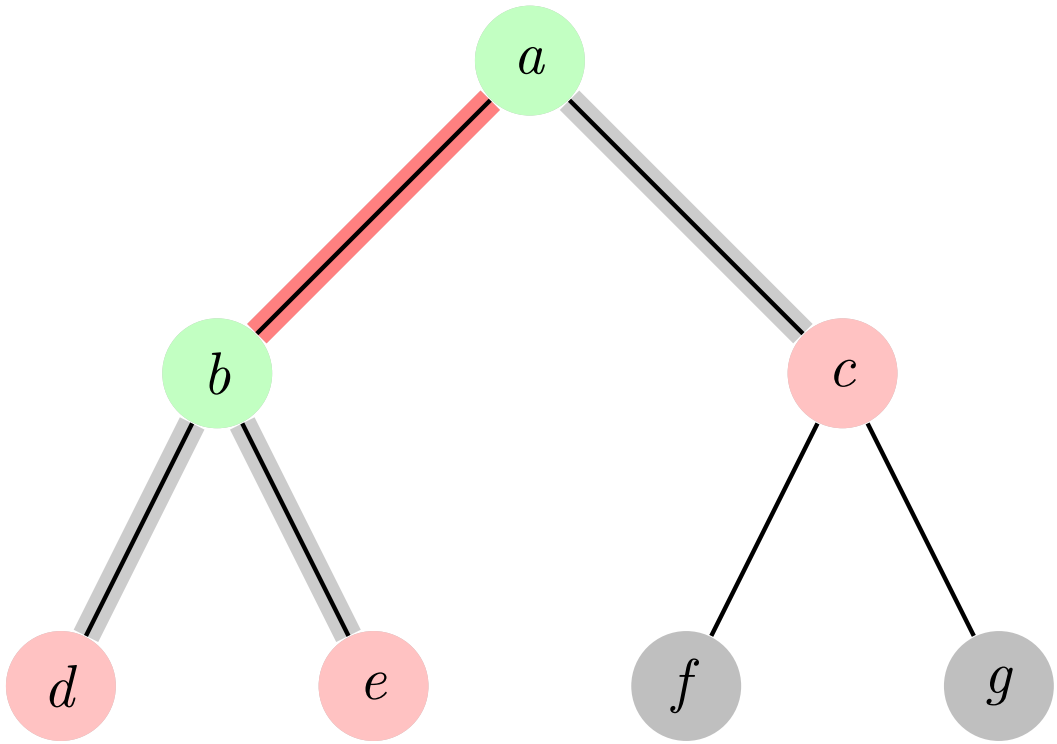


Depth First Search (DFS)

Depth has the max priority

Child $\longleftarrow$ Parent

$b \longleftarrow a$

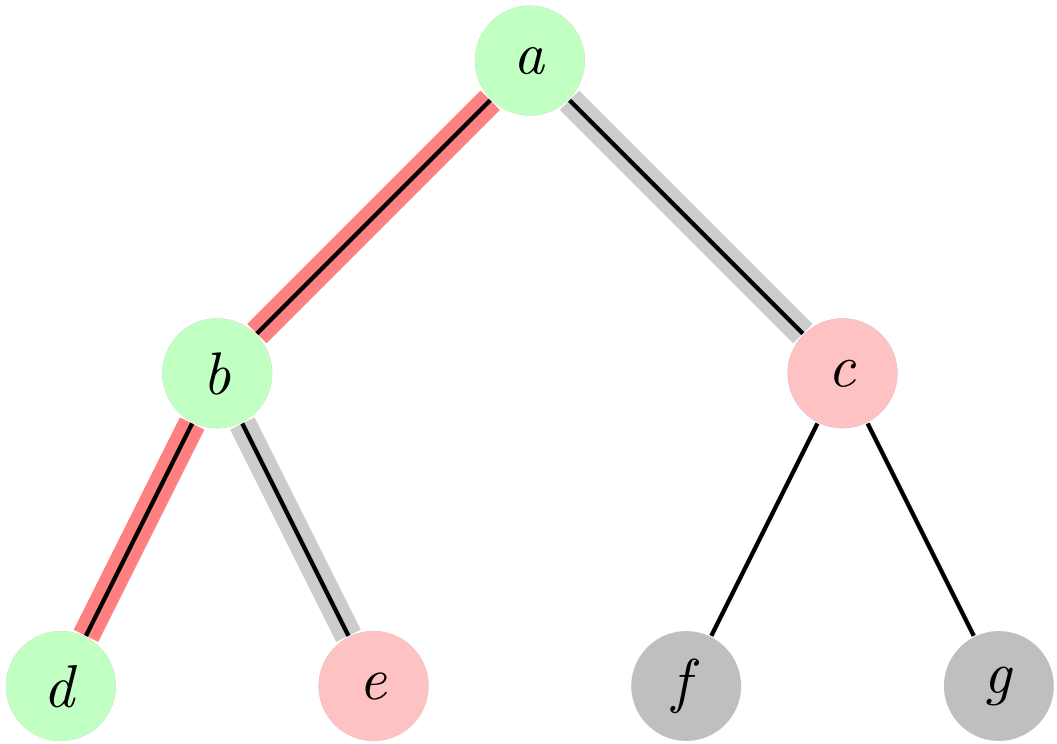


Depth First Search (DFS)

Depth has the max priority

Child $\longleftarrow$ Parent

$b \longleftarrow a$
$d \longleftarrow b$

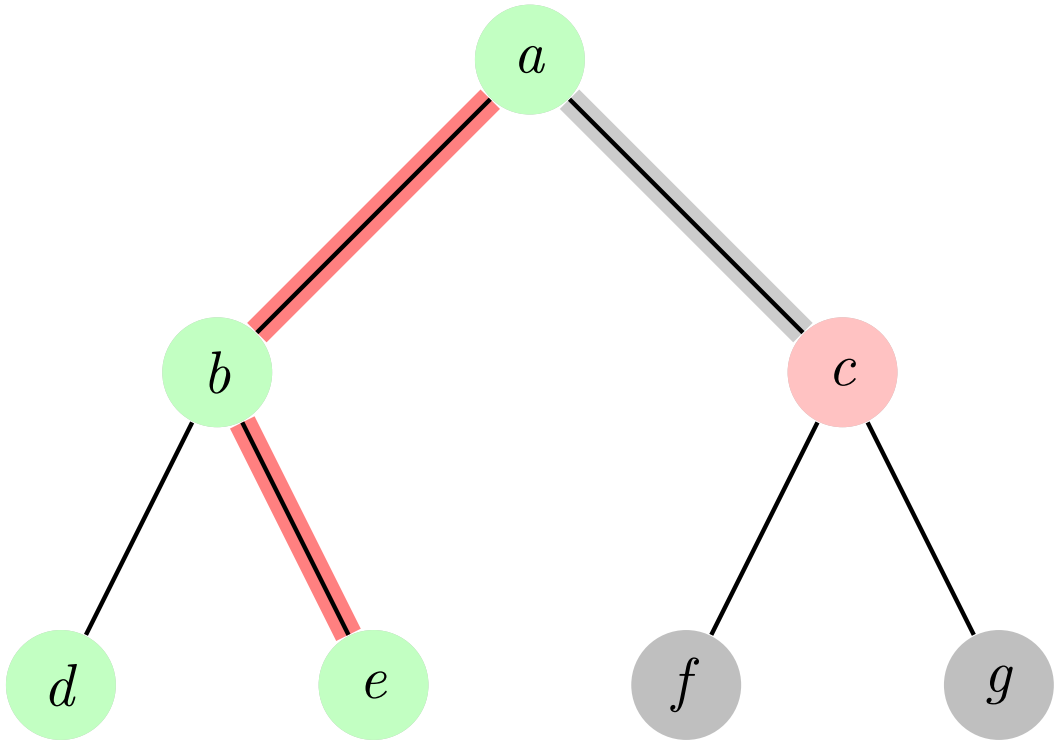


Depth First Search (DFS)

Depth has the max priority

Child $\longleftarrow$ Parent

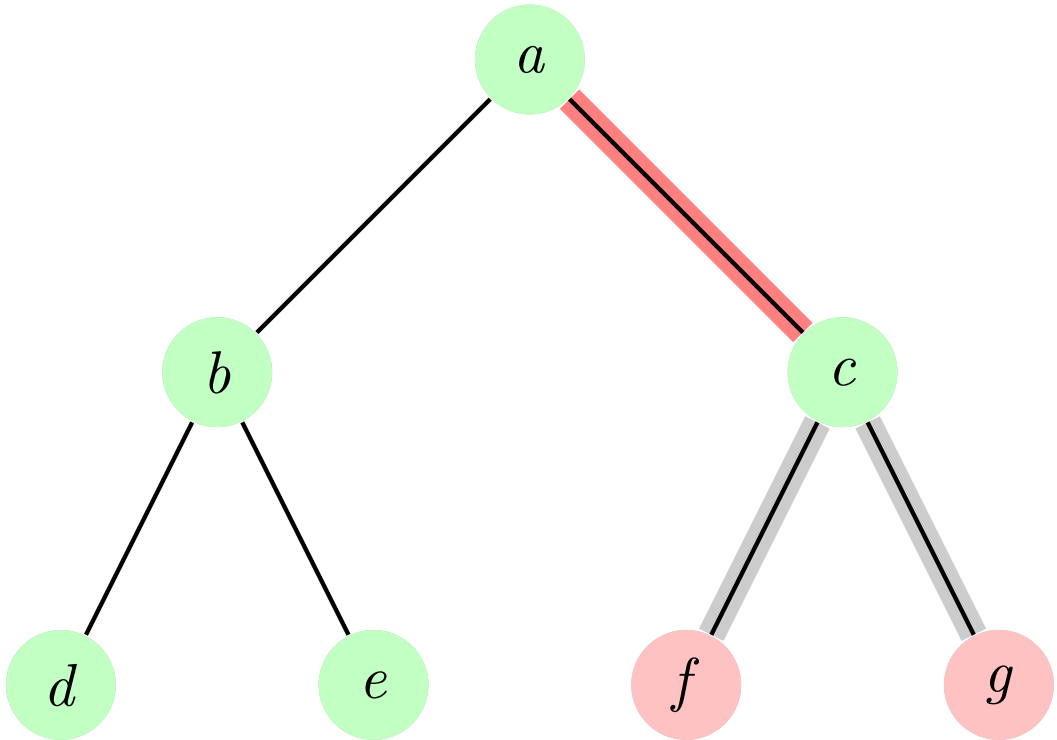
$b \longleftarrow a$
$d \longleftarrow b$
$e \longleftarrow b$



Depth First Search (DFS)

Depth has the max priority
Child $\longleftarrow$ Parent

$b \longleftarrow a$
$d \longleftarrow b$
$e \longleftarrow b$
$c \longleftarrow a$

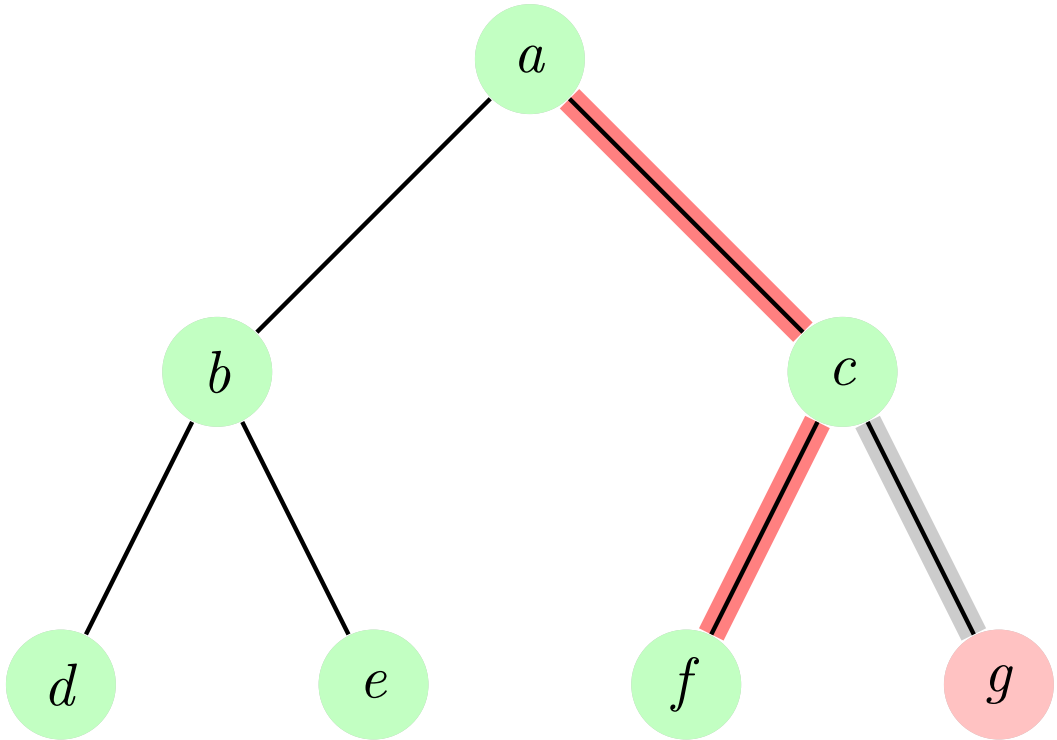


Depth First Search (DFS)

Depth has the max priority

Child $\longleftarrow$ Parent

$b \longleftarrow a$
$d \longleftarrow b$
$e \longleftarrow b$
$c \longleftarrow a$
$f \longleftarrow c$

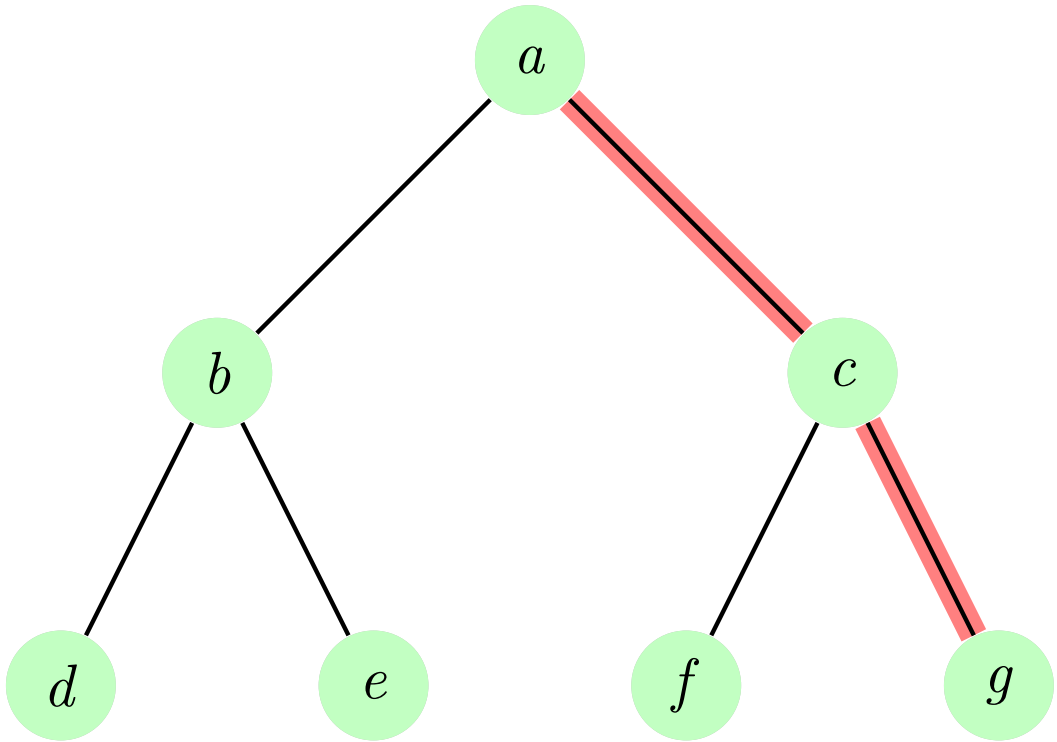


Depth First Search (DFS)

Therefore, the below table summarizes how did we get to any node through our traversal

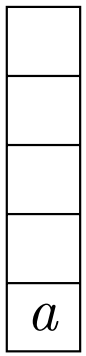
Child $\longleftarrow$ Parent

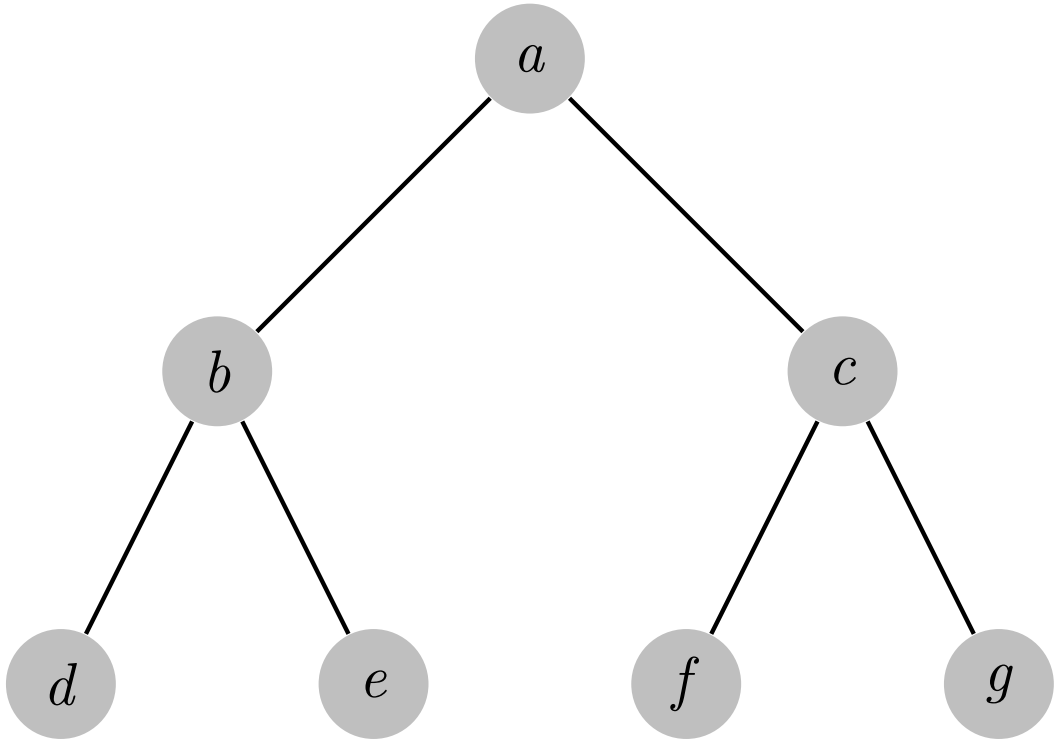
$b \longleftarrow a$
$d \longleftarrow b$
$e \longleftarrow b$
$c \longleftarrow a$
$f \longleftarrow c$
$g \longleftarrow c$



Depth First Search (DFS)

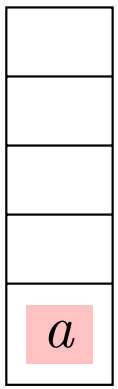
Let's do it again, to notice the **pattern** of nodes to be visited

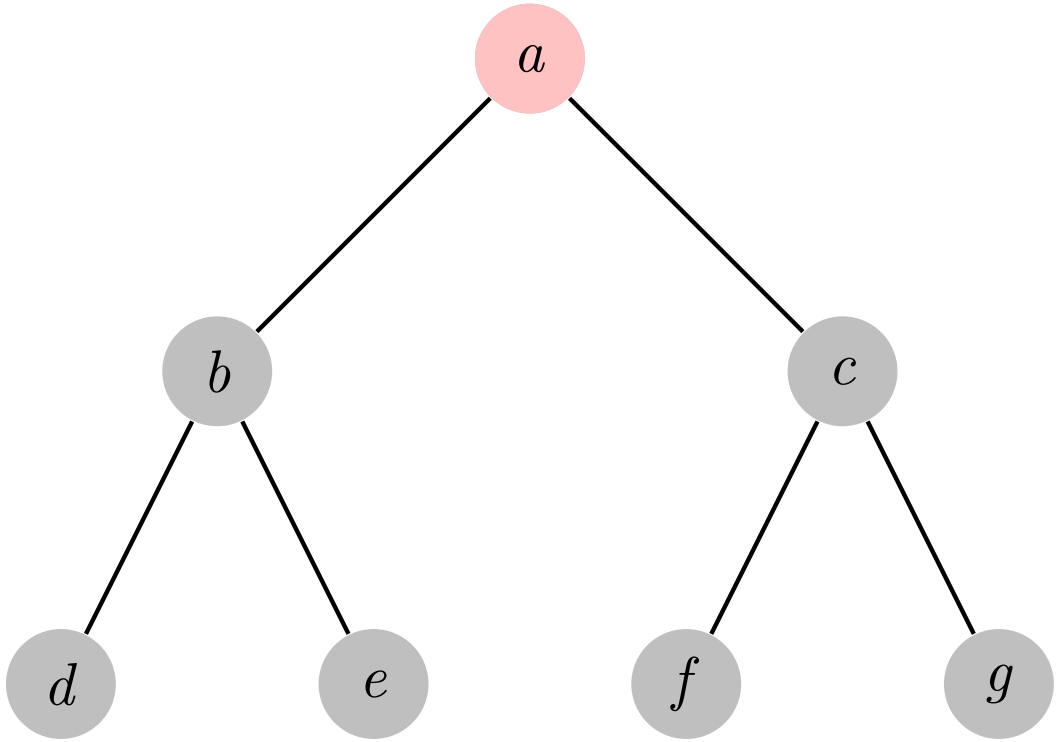




Depth First Search (DFS)

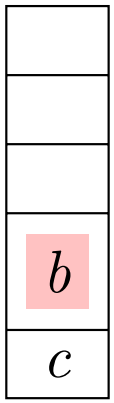
Let's do it again, to notice the **pattern** of nodes to be visited

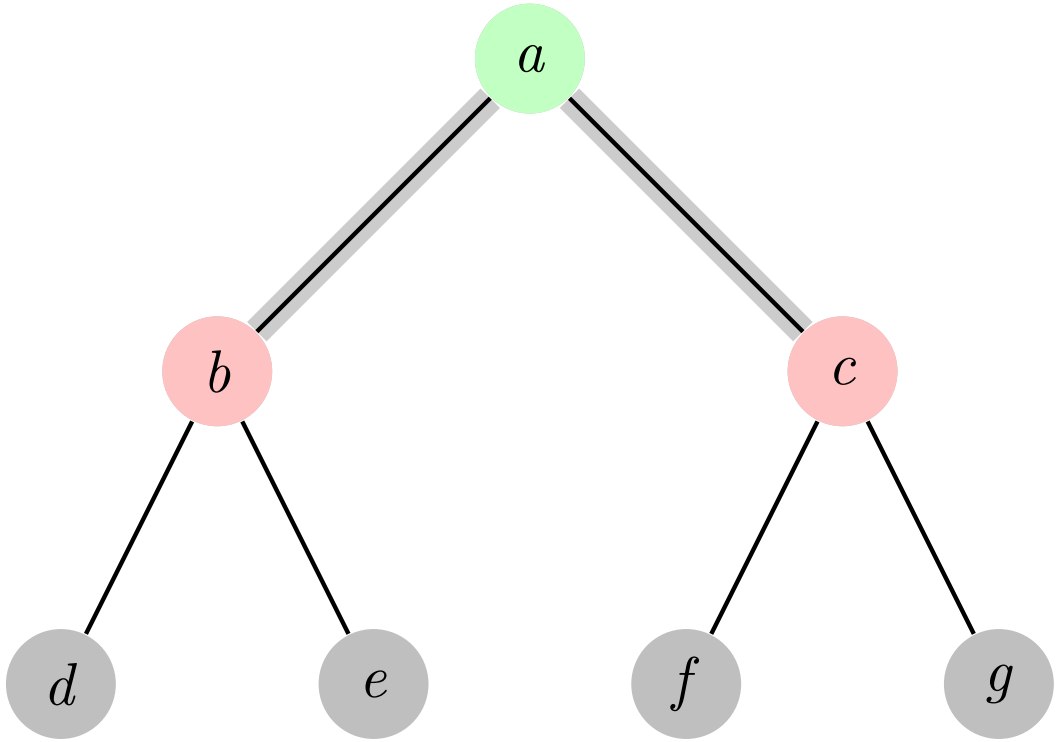




Depth First Search (DFS)

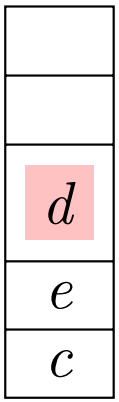
Let's do it again, to notice the **pattern** of nodes to be visited

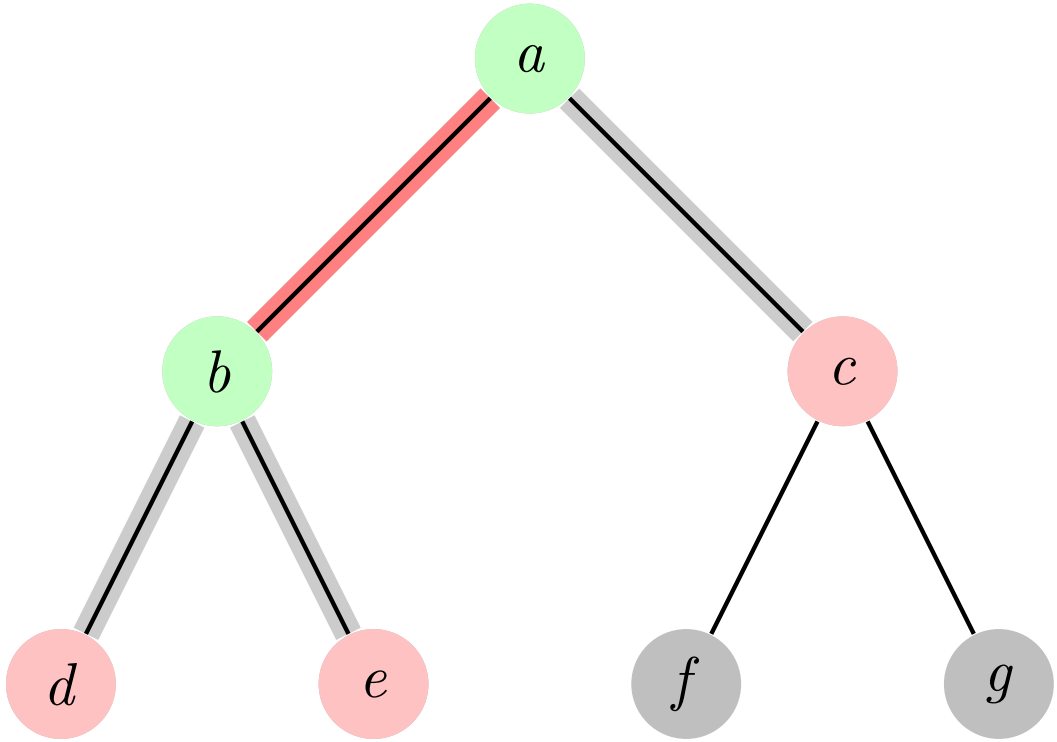




Depth First Search (DFS)

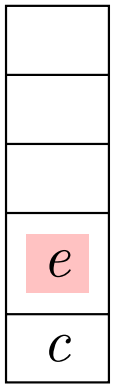
Let's do it again, to notice the **pattern** of nodes to be visited

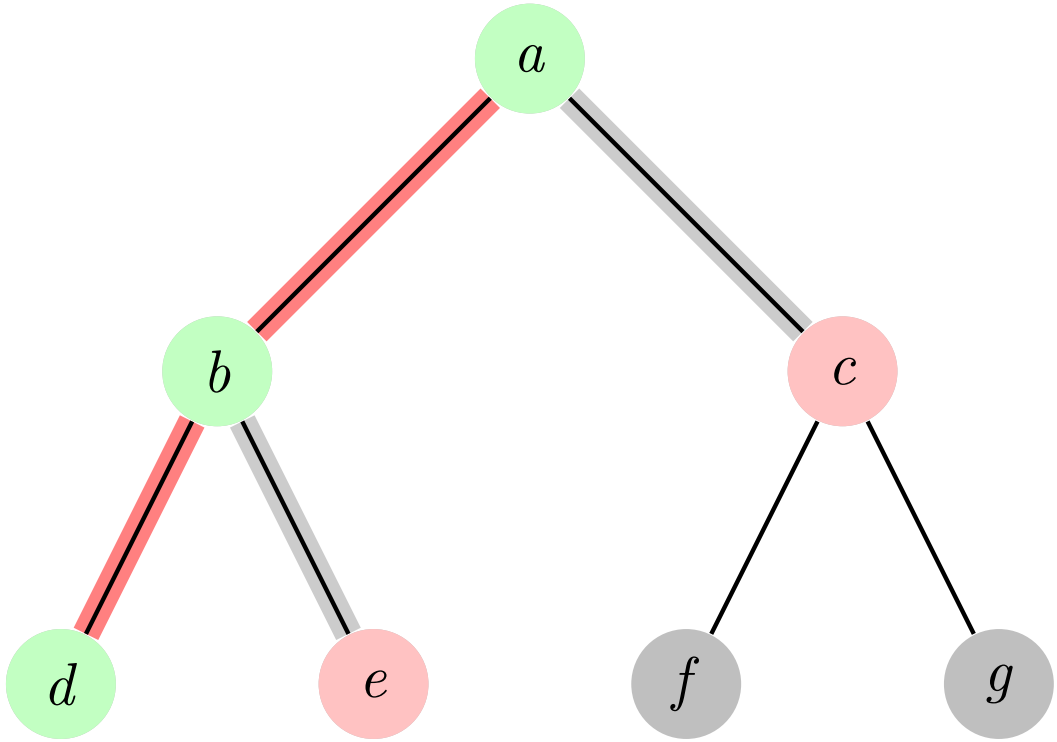




Depth First Search (DFS)

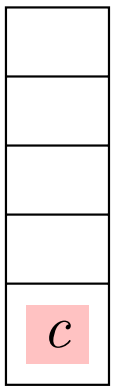
Let's do it again, to notice the **pattern** of nodes to be visited

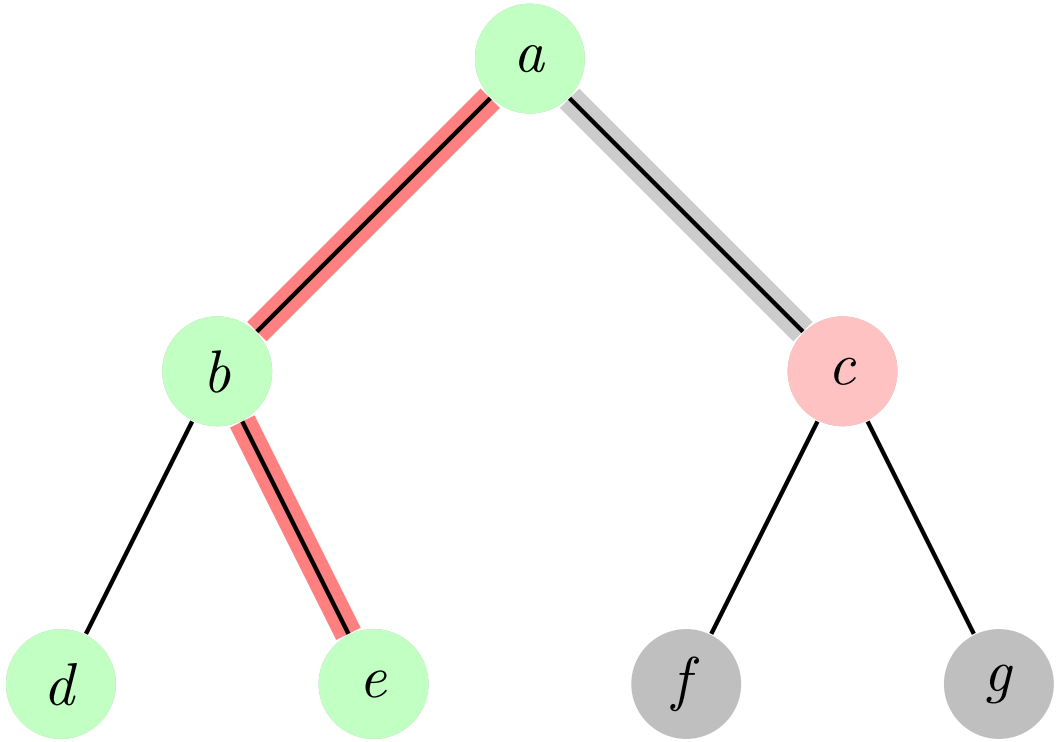




Depth First Search (DFS)

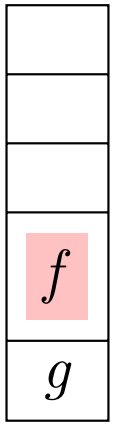
Did you get the pattern of nodes to be visited?

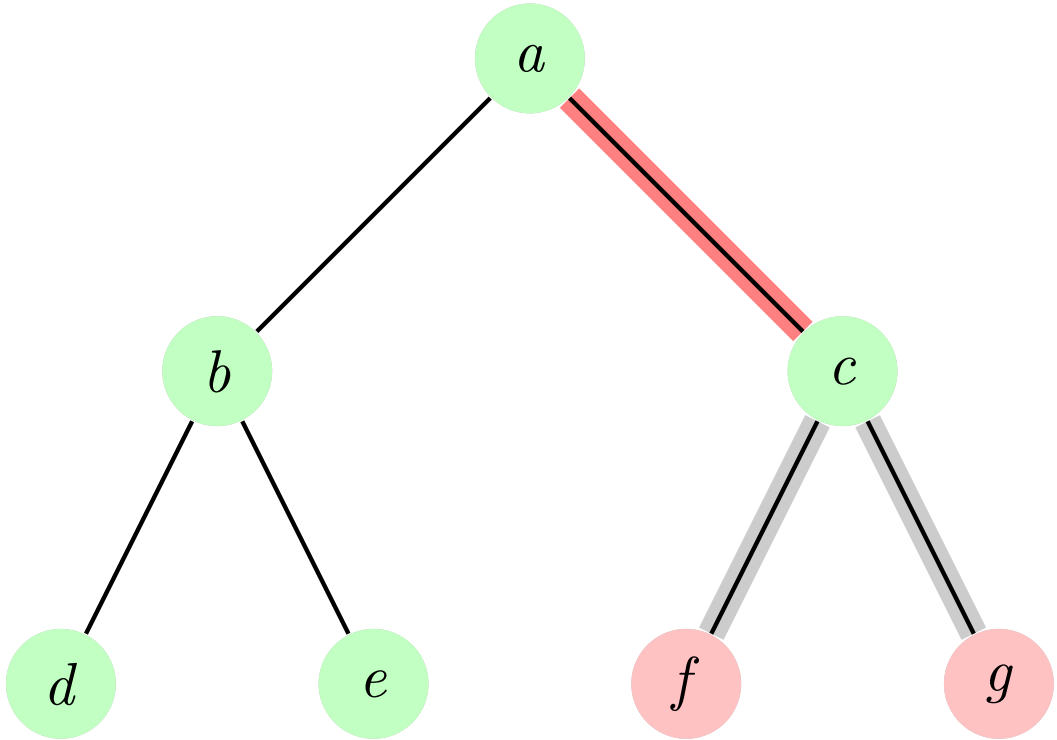




Depth First Search (DFS)

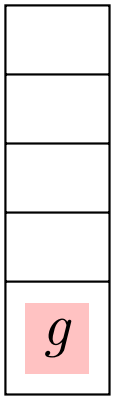
Did you get the pattern of nodes to be visited? **Last** inserted element is **first** to explore.

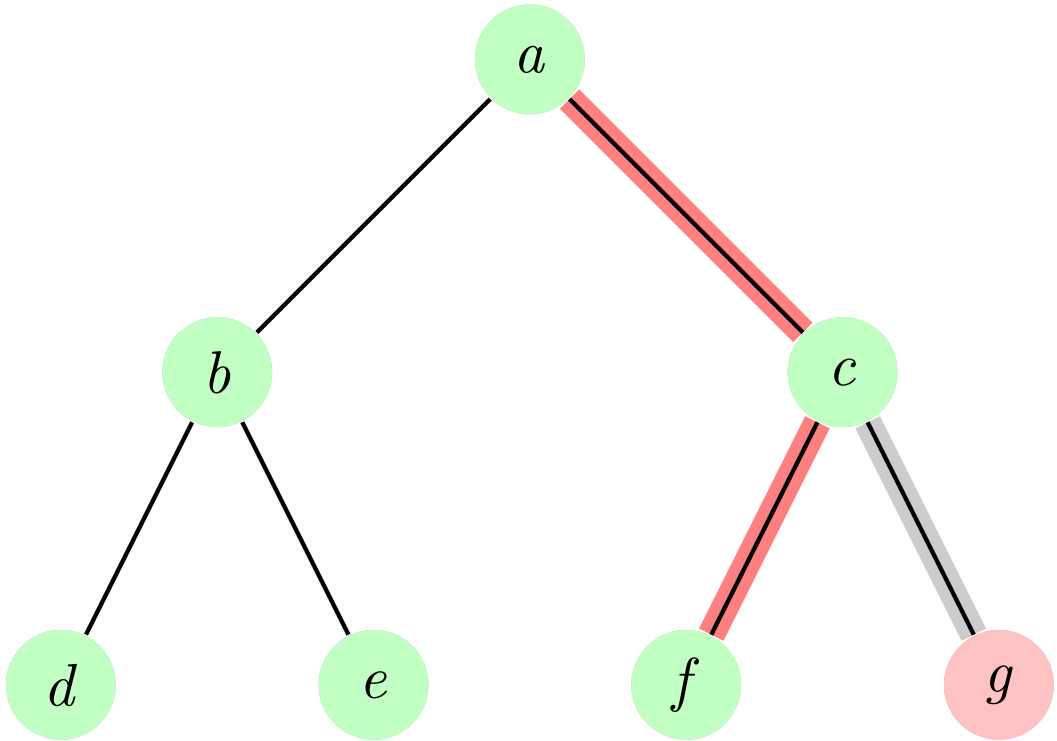




Depth First Search (DFS)

Did you notice what might be this structure? Hint: Last-in First-out

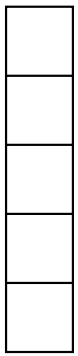


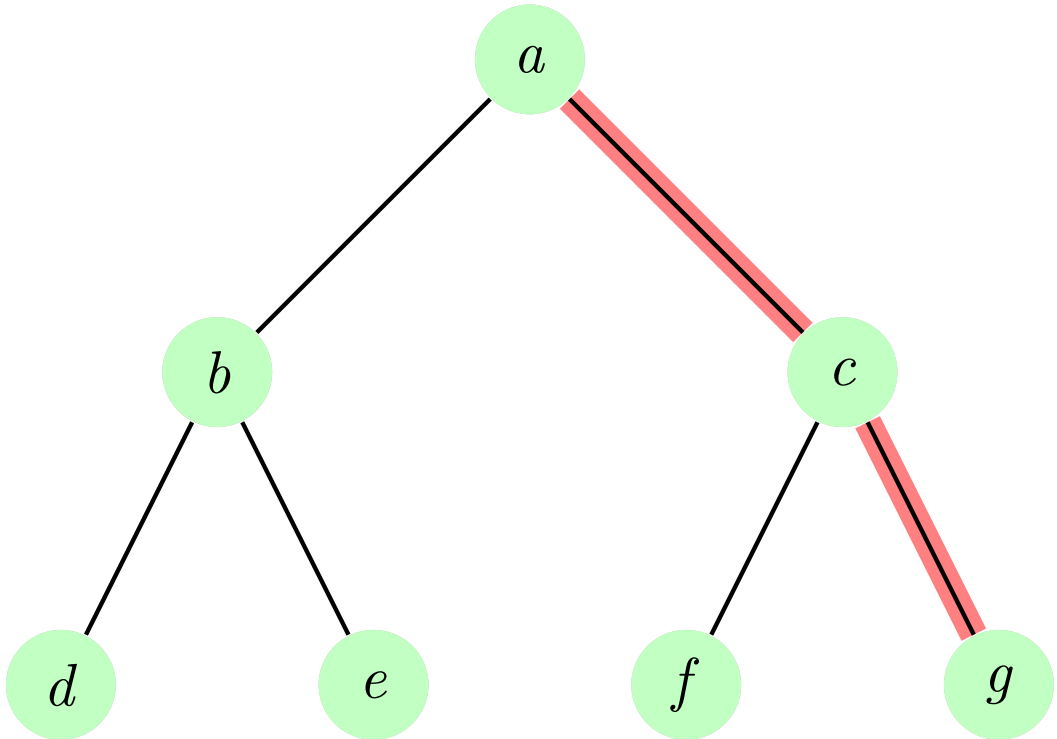


Depth First Search (DFS)

Did you notice what might be this structure? Hint: Last-in First-out

Yes, it is *Stack*





Depth First Search (DFS)

Algorithm 1: DEPTH-FIRST($root$)

```
def S to be Stack;
visited ⟵ {};
S.push(root);
while S ≠ ϕ do
    node ⟵ S.pop();
    if node ∉ visited then
        visited ⟵ visited ∪ {node};
        for n ∈ adjacent(node) do
            S.push(n);
        end
    end
end
```

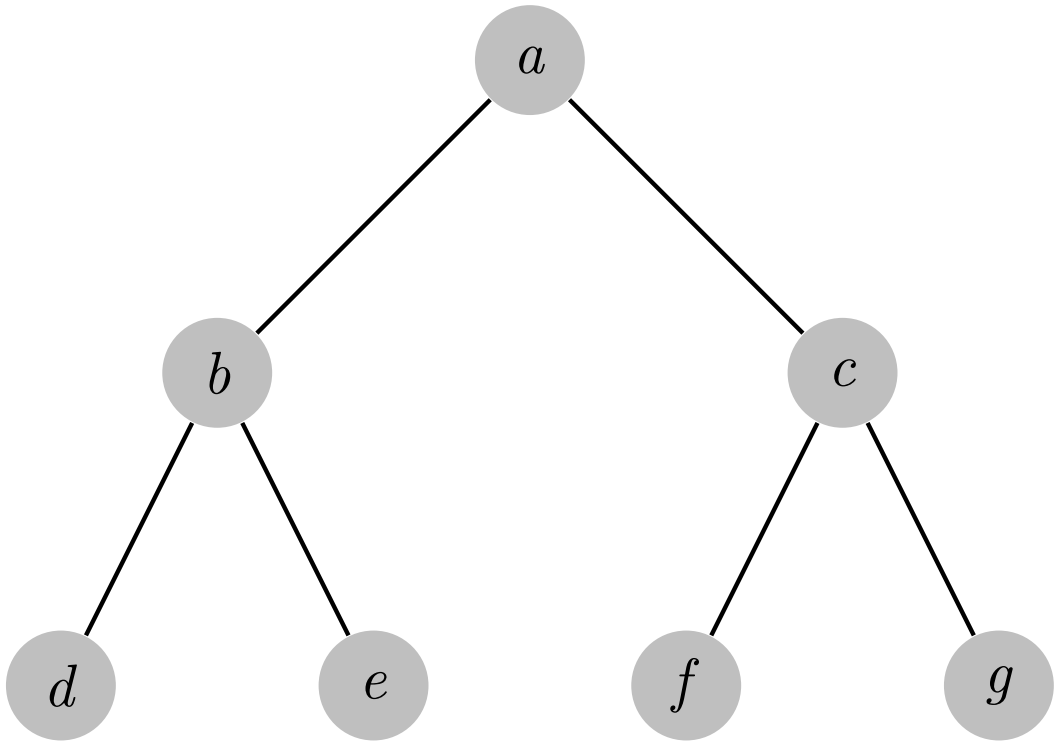
Path Construction from Traversing Algorithm

Cool; now, we have shown how the algorithm traverses, *visits the nodes*, and checked the pseudo-code, but and important question is

how to construct a path from a traversal algorithm like DFS?

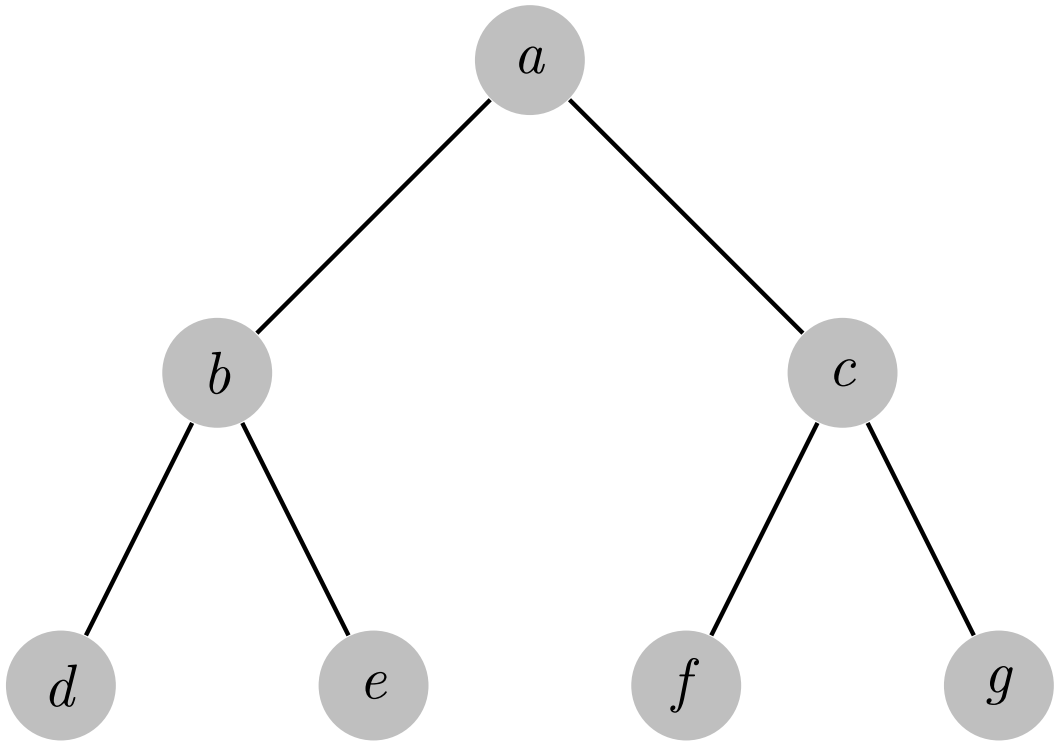
Path Construction from Traversing Algorithm

First, to differentiate between traversal and path, we revisit the same tree as before; the ordering of the visited nodes is a, b, d, e, c, f, g.



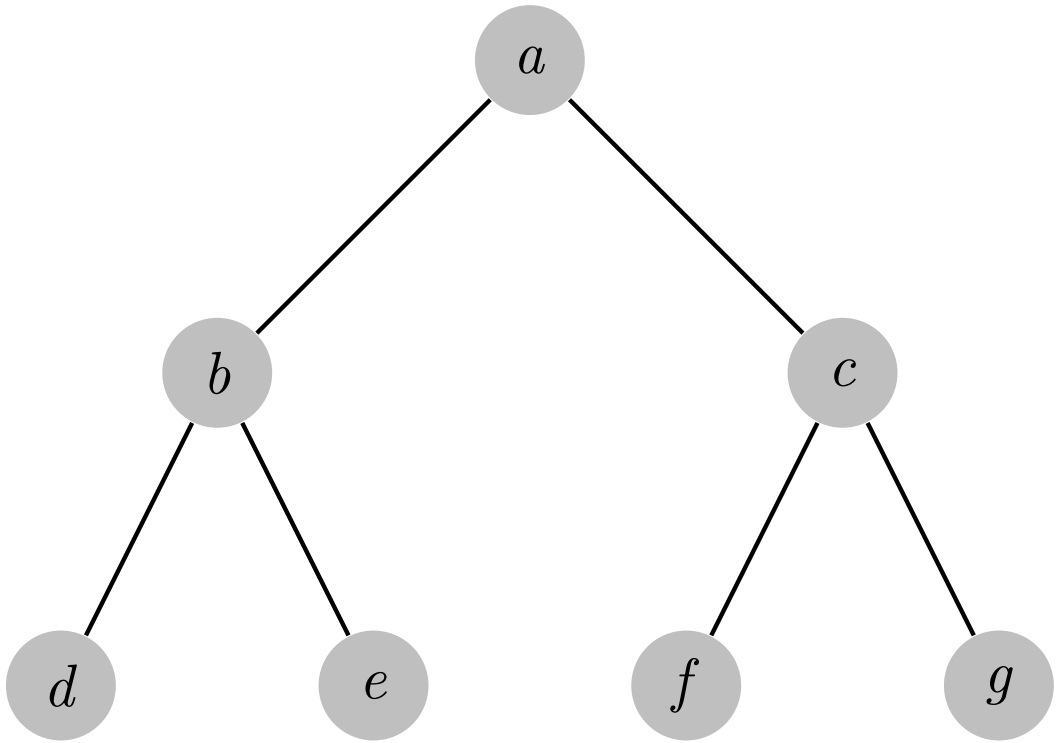
Path Construction from Traversing Algorithm

the ordering of the visited nodes is a, b, d, e, c, f, g. Does this mean that the **path** according to our algorithm from $a \rightarrow f$ is a, b, d, e, c, f?



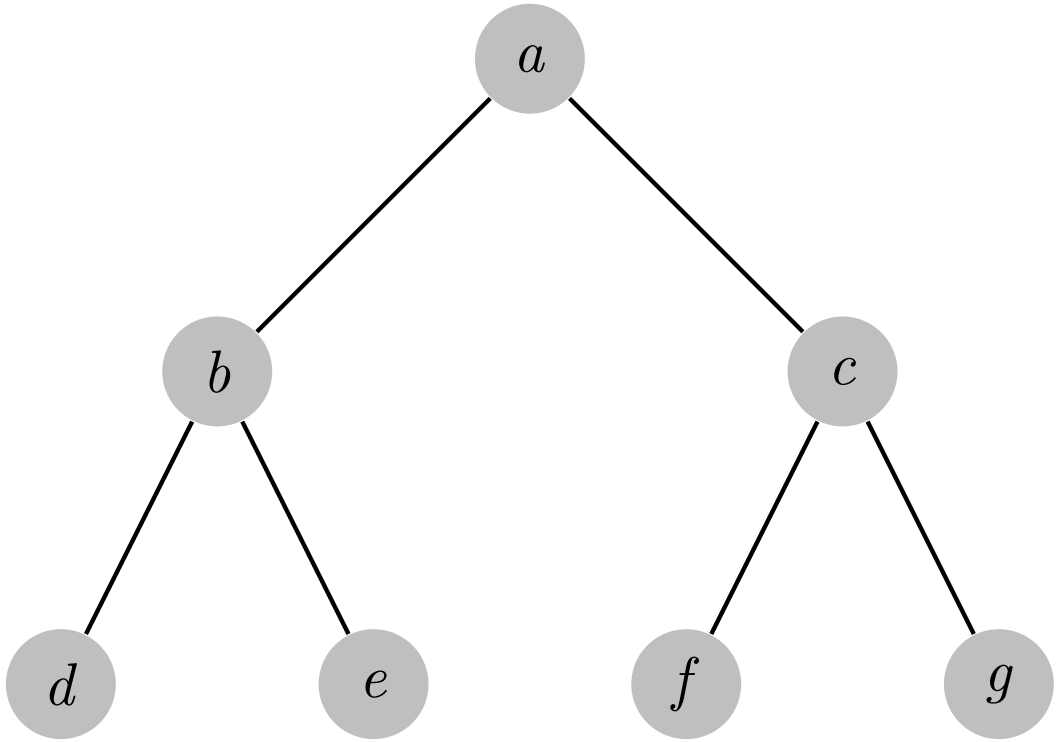
Path Construction from Traversing Algorithm

the ordering of the visited nodes is a, b, d, e, c, f, g. Does this mean that the **path** according to our algorithm from $a \to f$ is a, b, d, e, c, f? well, **No**. This is just a traversal order.



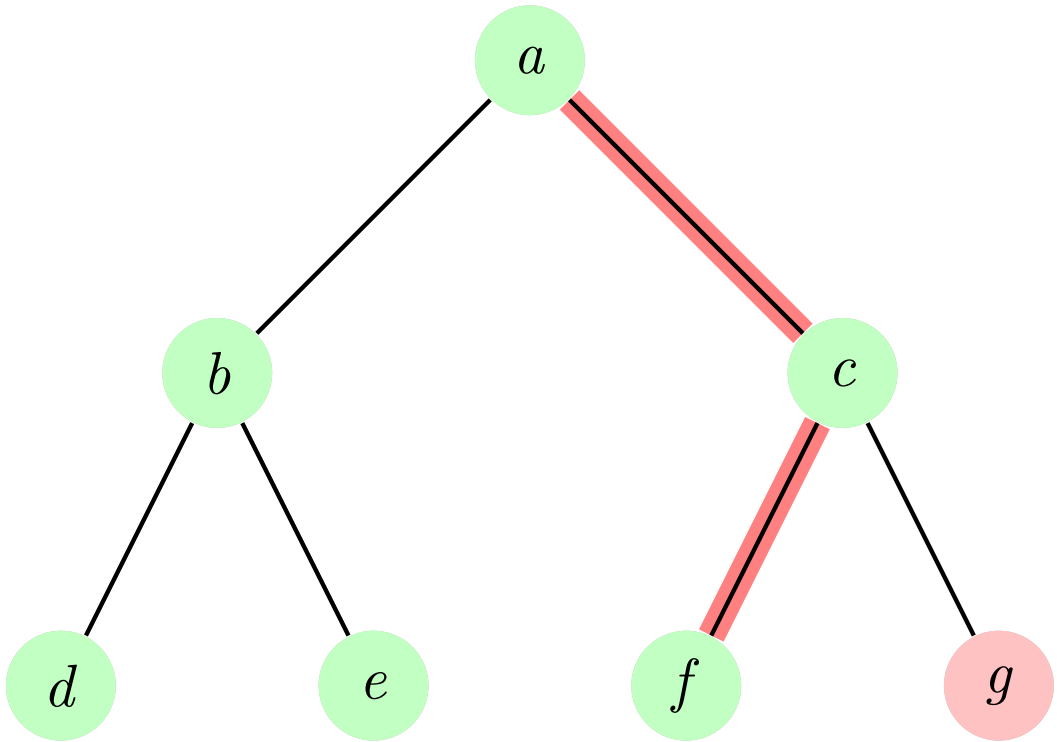
Path Construction from Traversing Algorithm

In that case, what is the path constructed from $a \rightarrow f$ according to our algorithm?



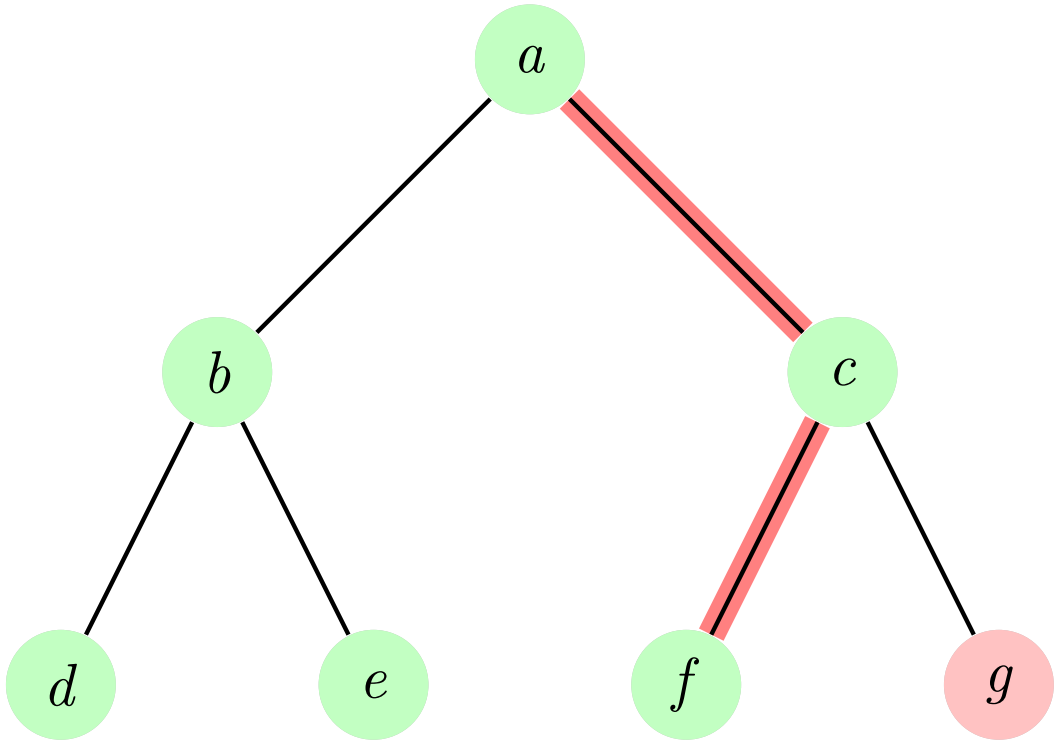
Path Construction from Traversing Algorithm

In that case, what is the path constructed from $a \rightarrow f$ according to our algorithm? It is a, c, f as we see from our previous execution.



Path Construction from Traversing Algorithm

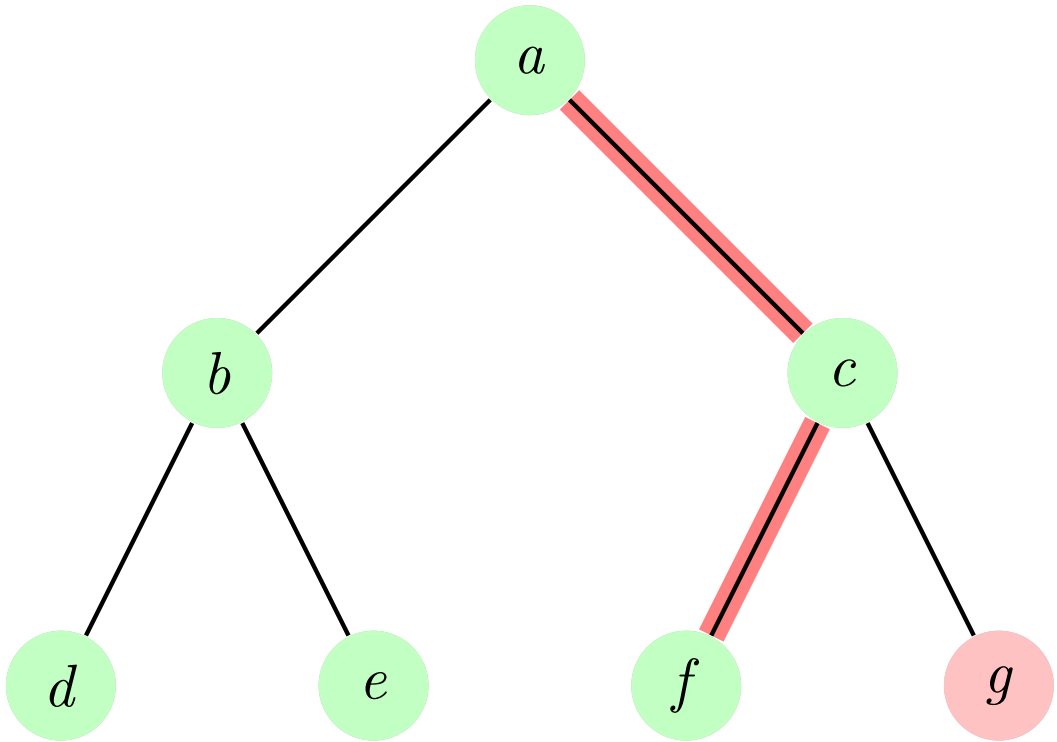
so, how could we construct such path from our algorithm?



Path Construction from Traversing Algorithm

so, how could we construct such path from our algorithm?

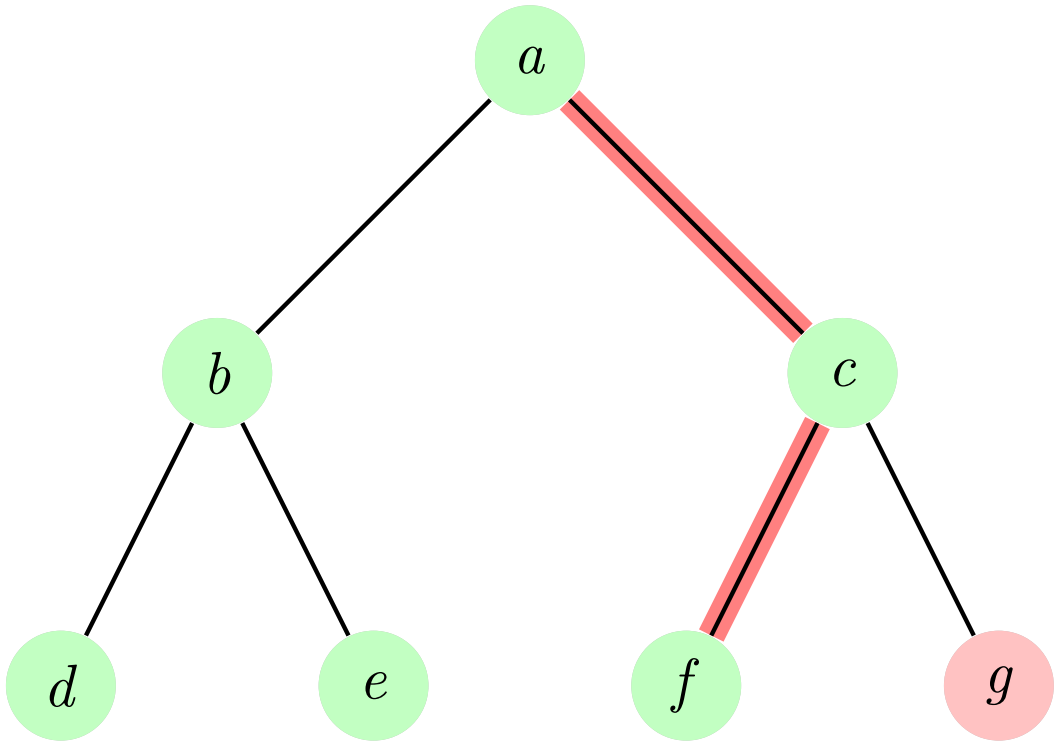
- store the whole paths, instead of just nodes



Path Construction from Traversing Algorithm

so, how could we construct such path from our algorithm?

- store the whole paths, instead of just nodes; Storage hungry



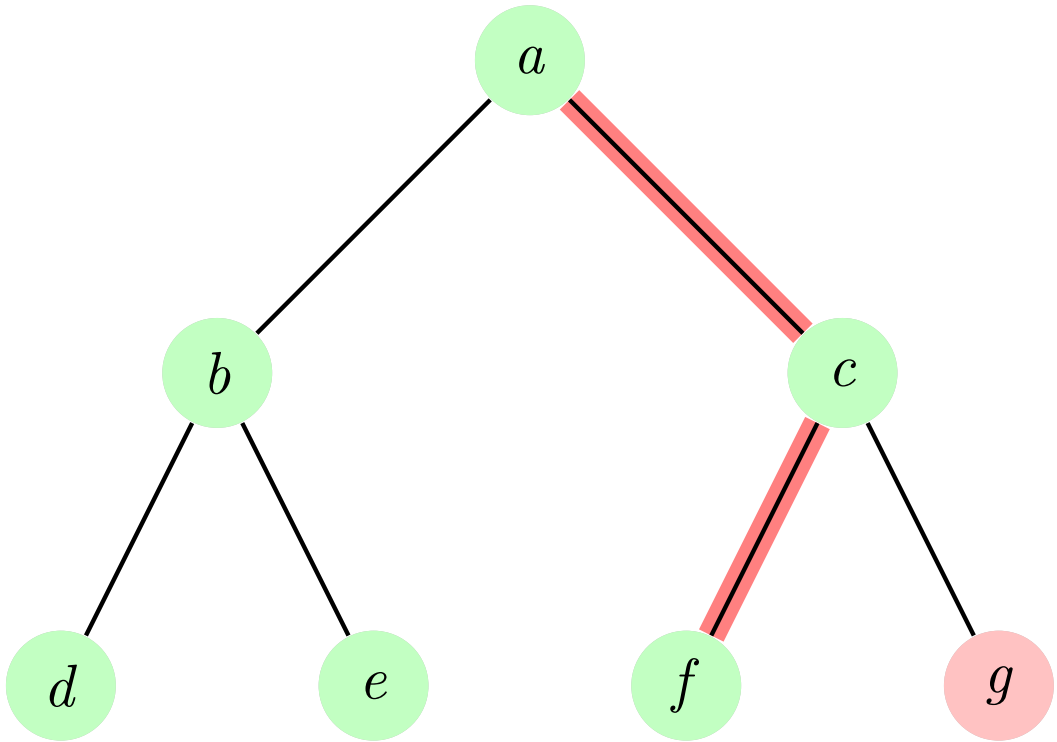
Path Construction from Traversing Algorithm

so, how could we construct such path from our algorithm?

- using **parent-child** structure.

Child $\longleftarrow$ Parent

$b \longleftarrow a$
$d \longleftarrow b$
$e \longleftarrow b$
$c \longleftarrow a$
$f \longleftarrow c$
$g \longleftarrow c$

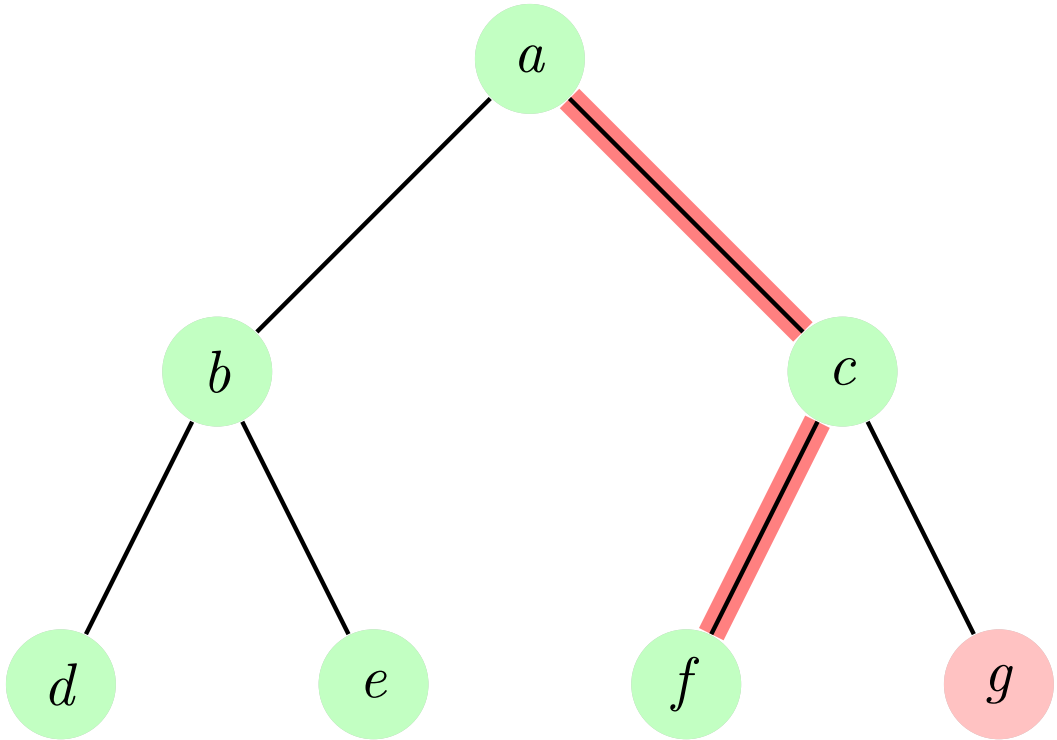


Path Construction from Traversing Algorithm

start from your goal, f, and move backward until getting your start.

Child $\longleftarrow$ Parent

$b \longleftarrow a$
$d \longleftarrow b$
$e \longleftarrow b$
$c \longleftarrow a$
$f \longleftarrow c$
$g \longleftarrow c$

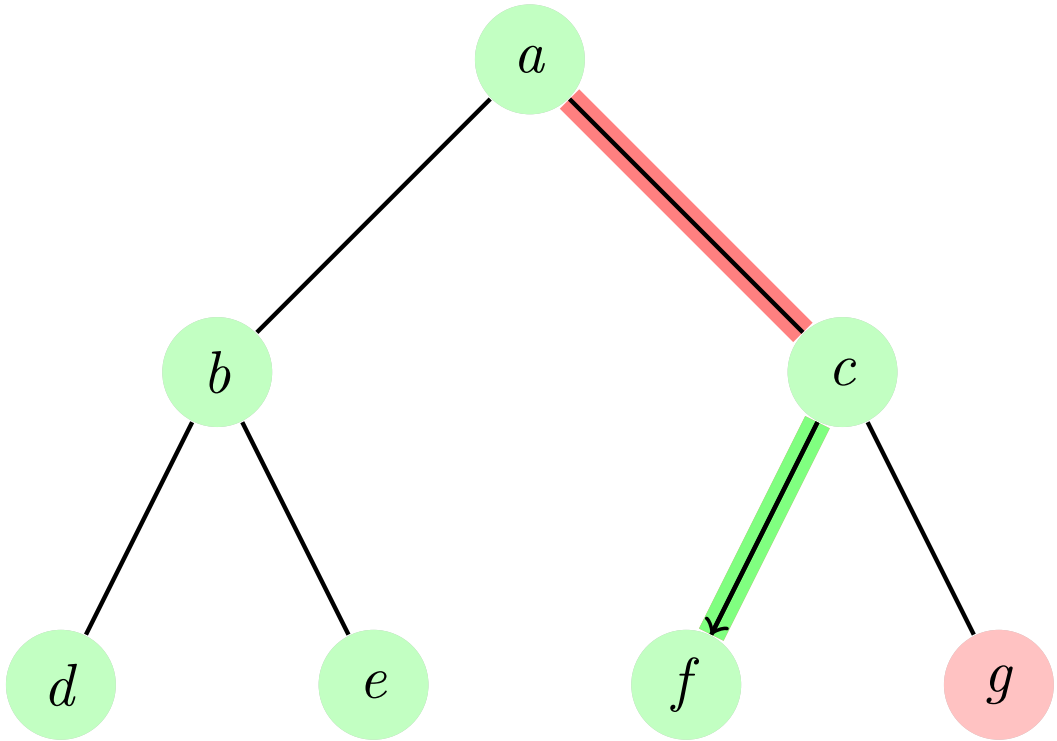


Path Construction from Traversing Algorithm

Path: $c \rightarrow f$

Child $\longleftarrow$ Parent

$b \longleftarrow a$
$d \longleftarrow b$
$e \longleftarrow b$
$c \longleftarrow a$
$f \longleftarrow c$
$g \longleftarrow c$

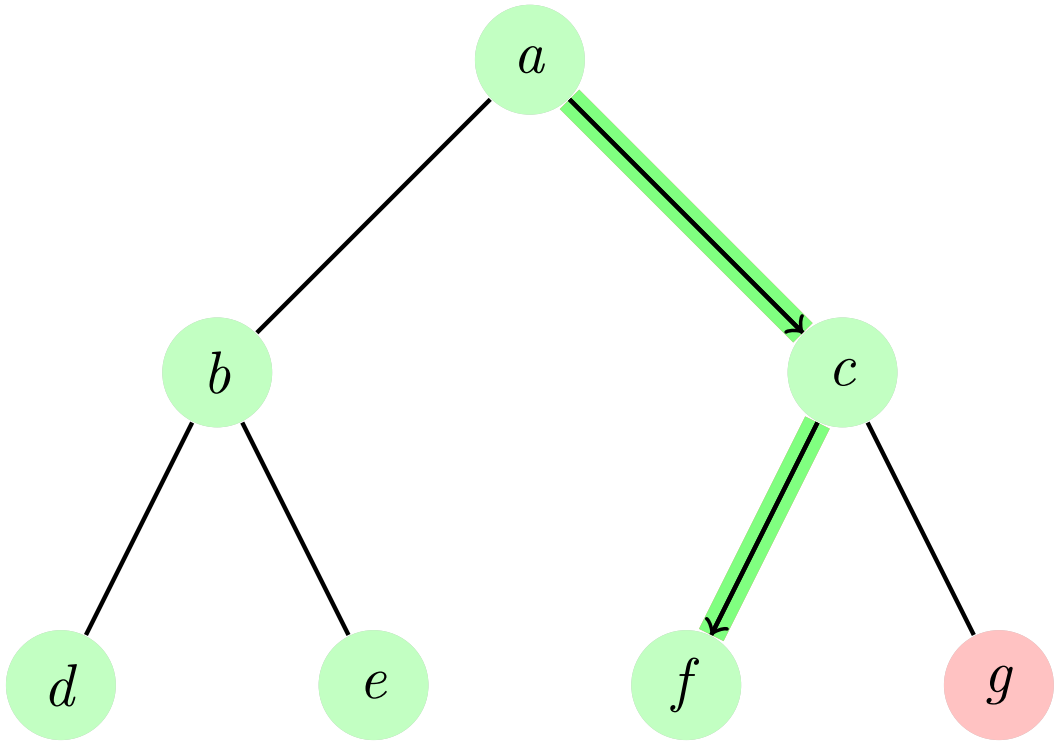


Path Construction from Traversing Algorithm

Path: $a \rightarrow c \rightarrow f$

Child $\longleftarrow$ Parent

$b \longleftarrow a$
$d \longleftarrow b$
$e \longleftarrow b$
$c \longleftarrow a$
$f \longleftarrow c$
$g \longleftarrow c$

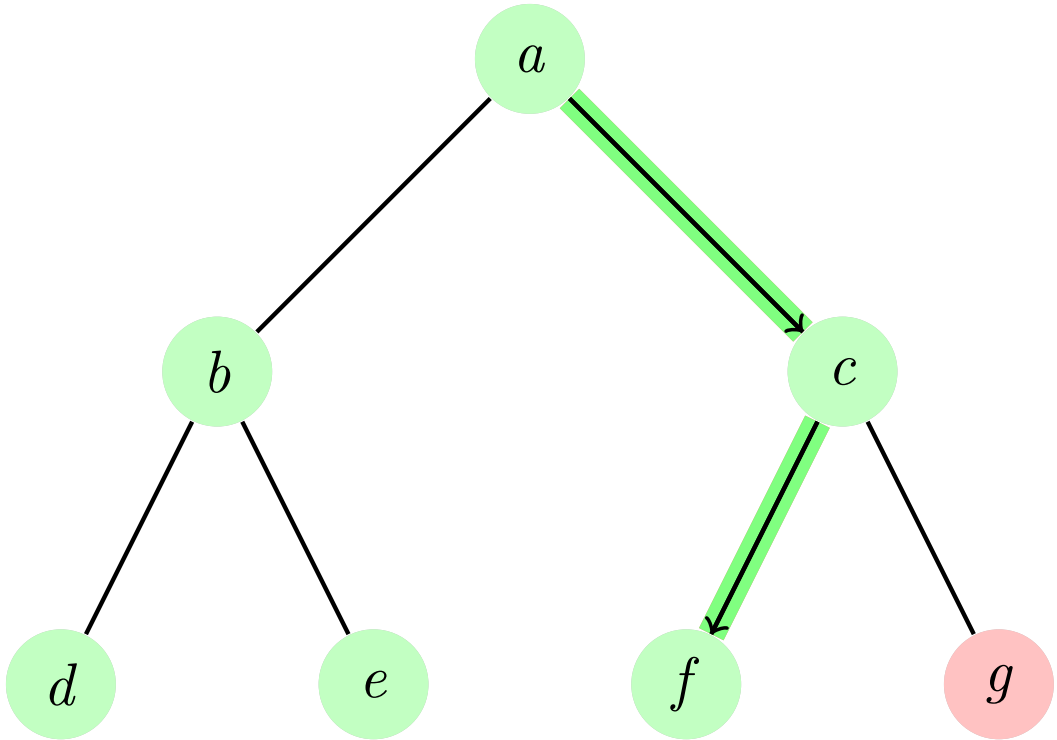


Path Construction from Traversing Algorithm

We can this technique for all the traversing algorithms, mentioned within this module.

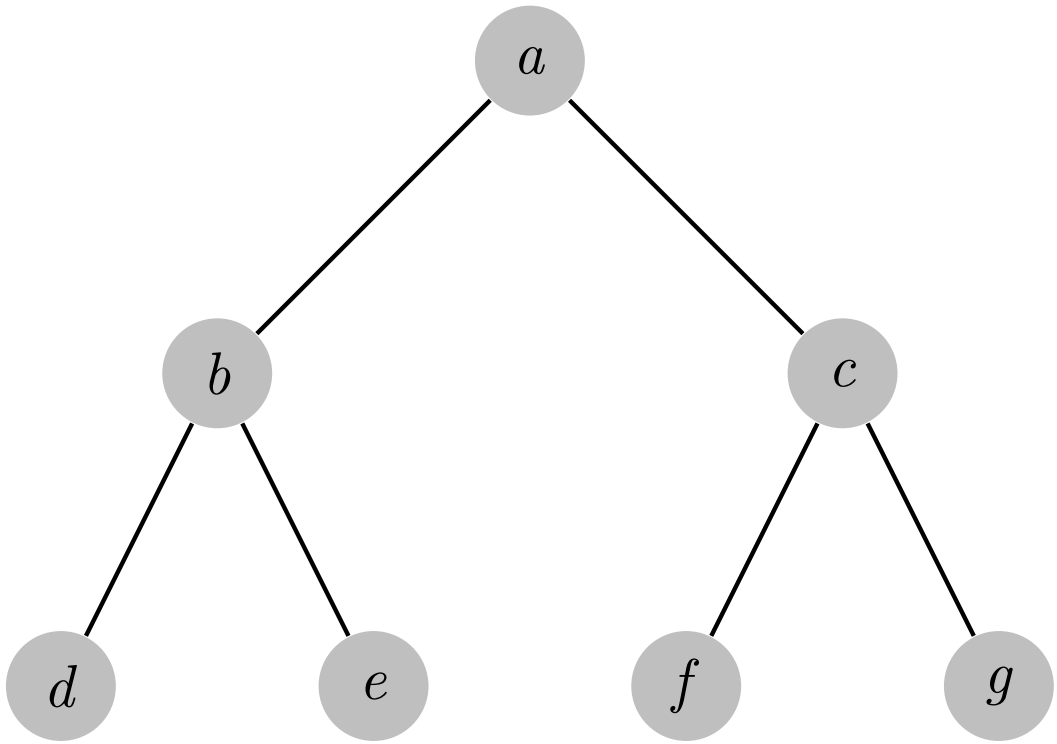
Child $\longleftarrow$ Parent

$b \longleftarrow a$
$d \longleftarrow b$
$e \longleftarrow b$
$c \longleftarrow a$
$f \longleftarrow c$
$g \longleftarrow c$



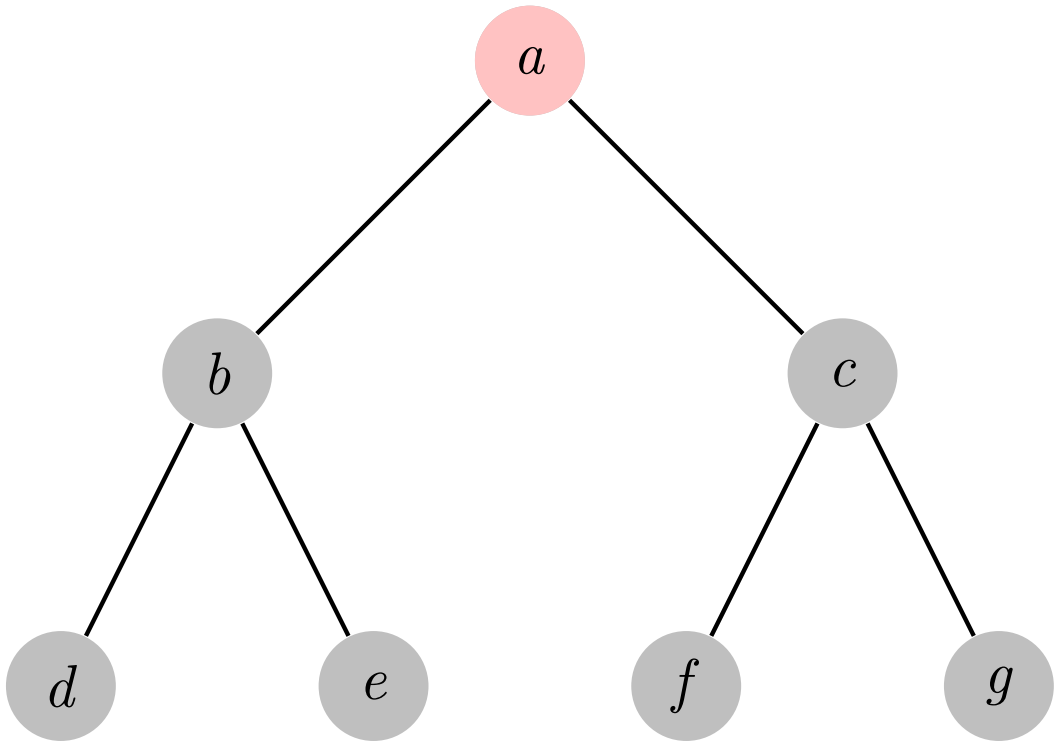
Breadth First Search (BFS)

Breadth has the max priority



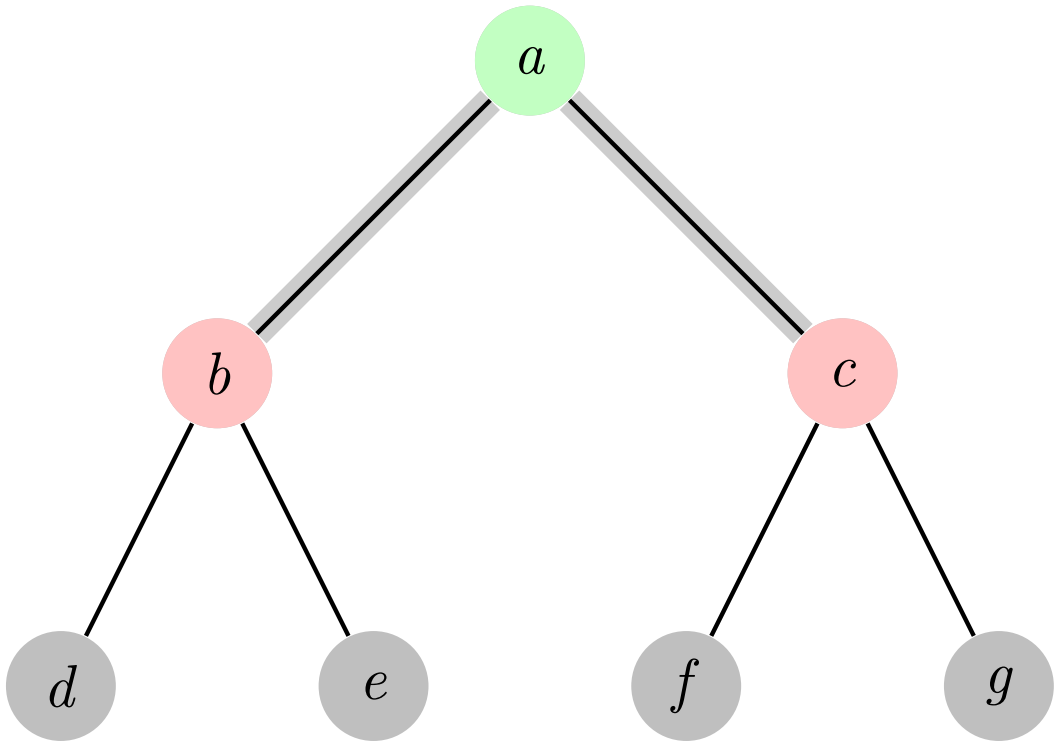
Breadth First Search (BFS)

Breadth has the max priority



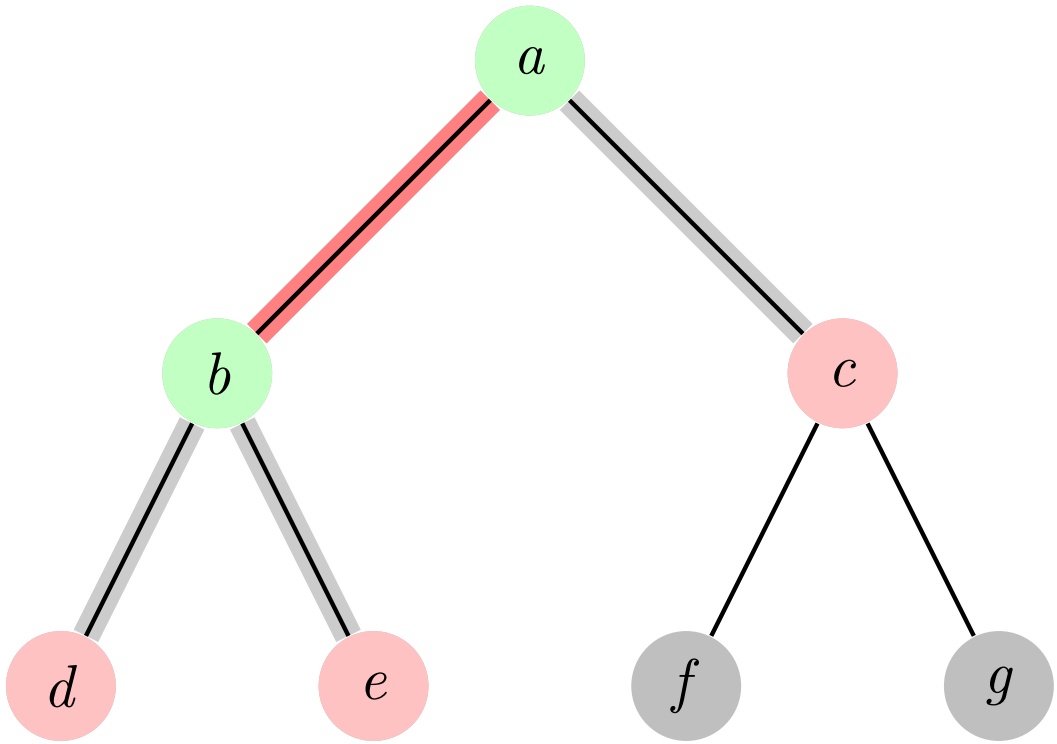
Breadth First Search (BFS)

Breadth has the max priority



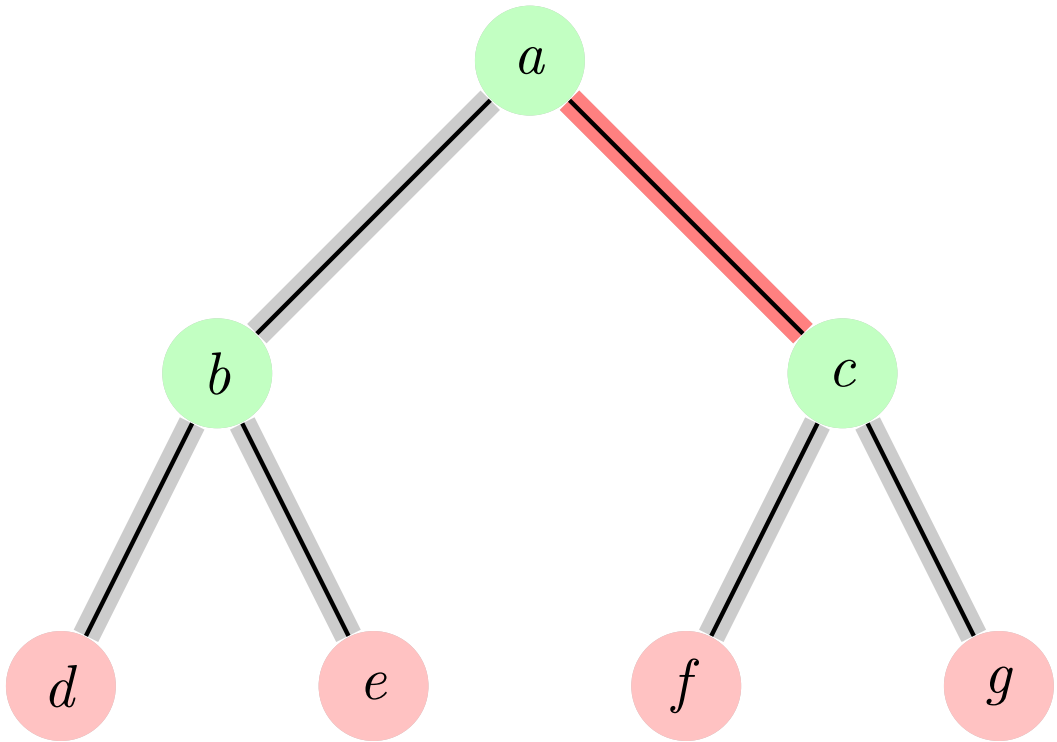
Breadth First Search (BFS)

Breadth has the max priority



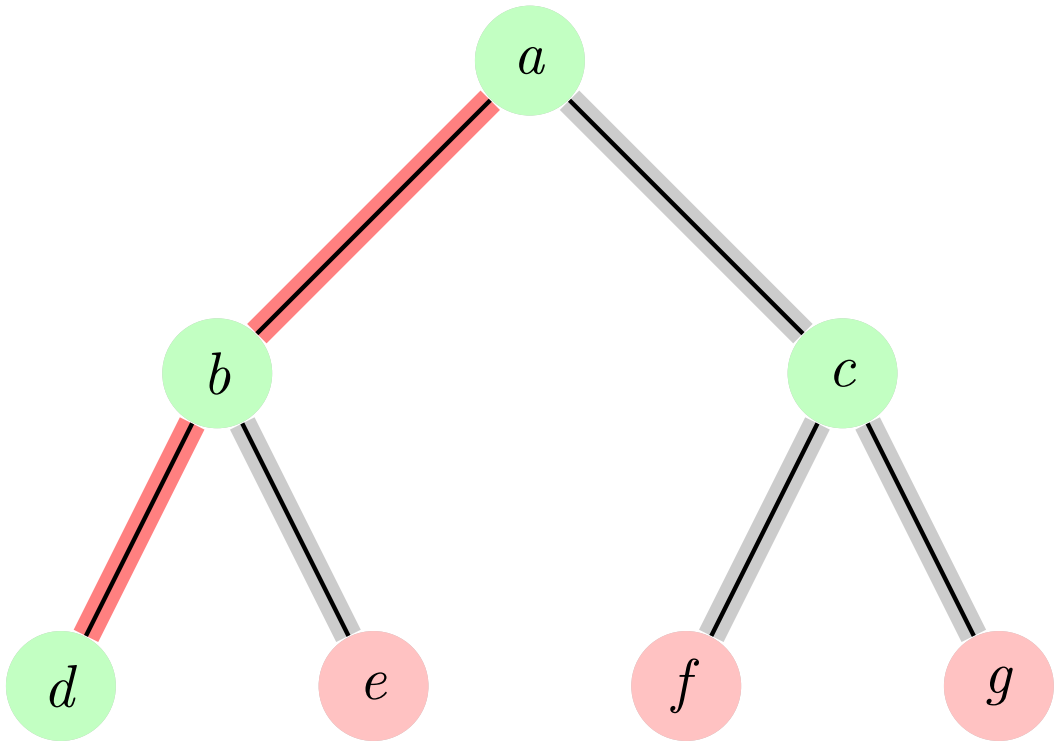
Breadth First Search (BFS)

Breadth has the max priority



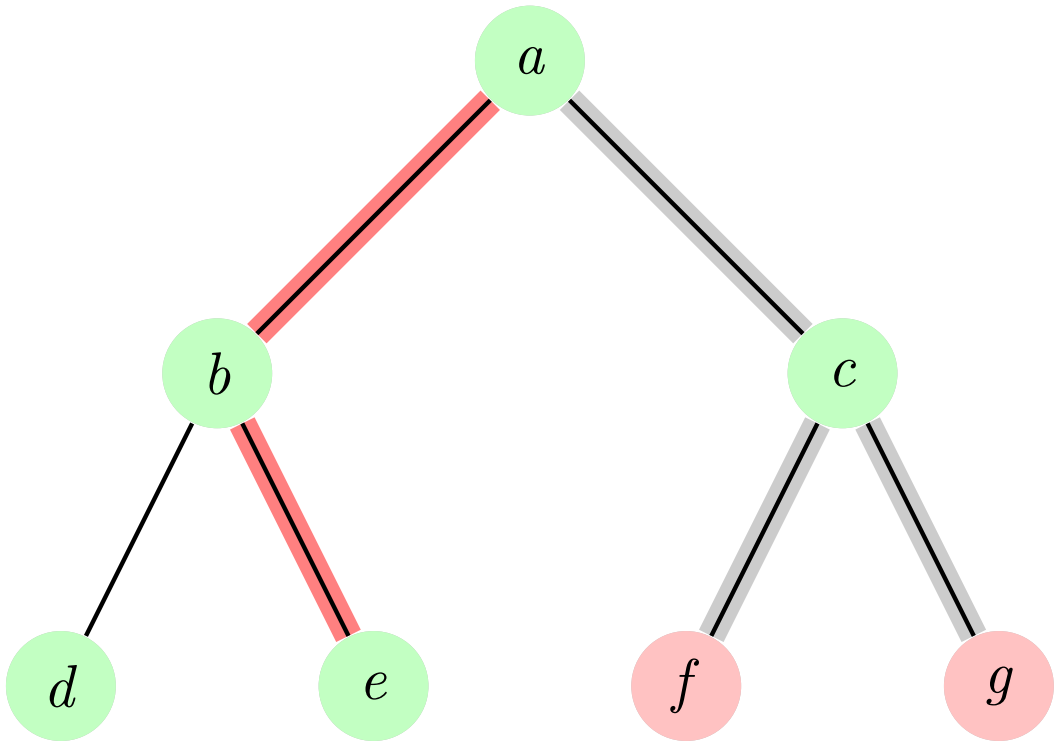
Breadth First Search (BFS)

Breadth has the max priority



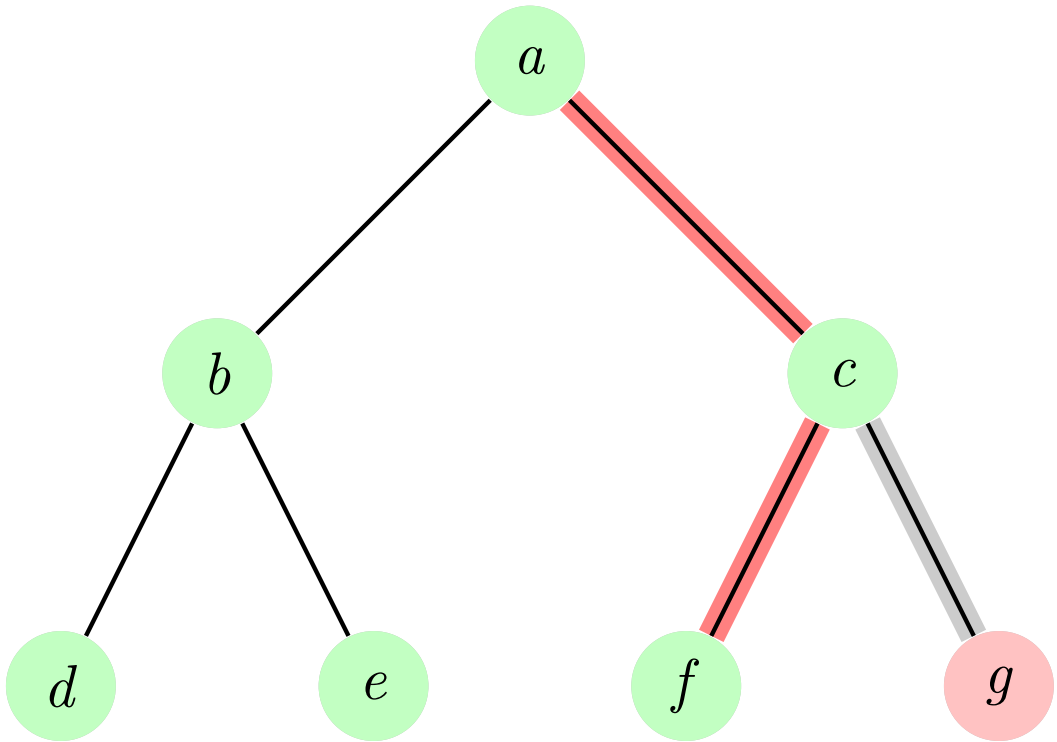
Breadth First Search (BFS)

Breadth has the max priority



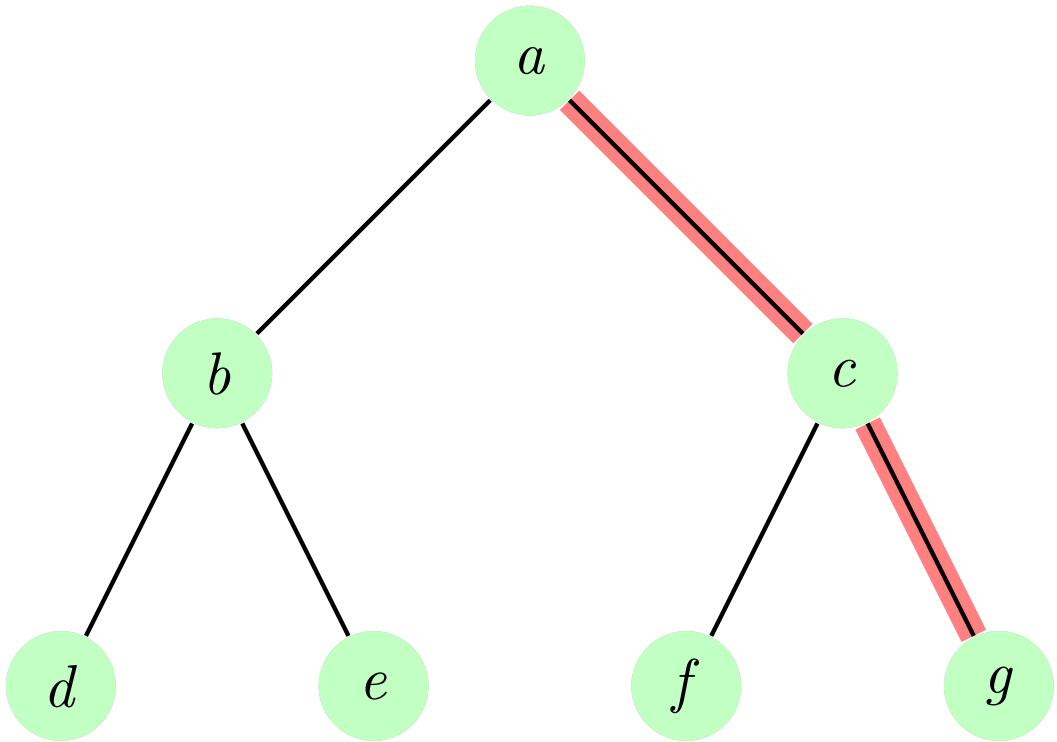
Breadth First Search (BFS)

Breadth has the max priority



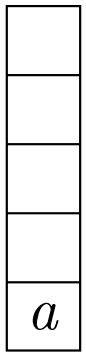
Breadth First Search (BFS)

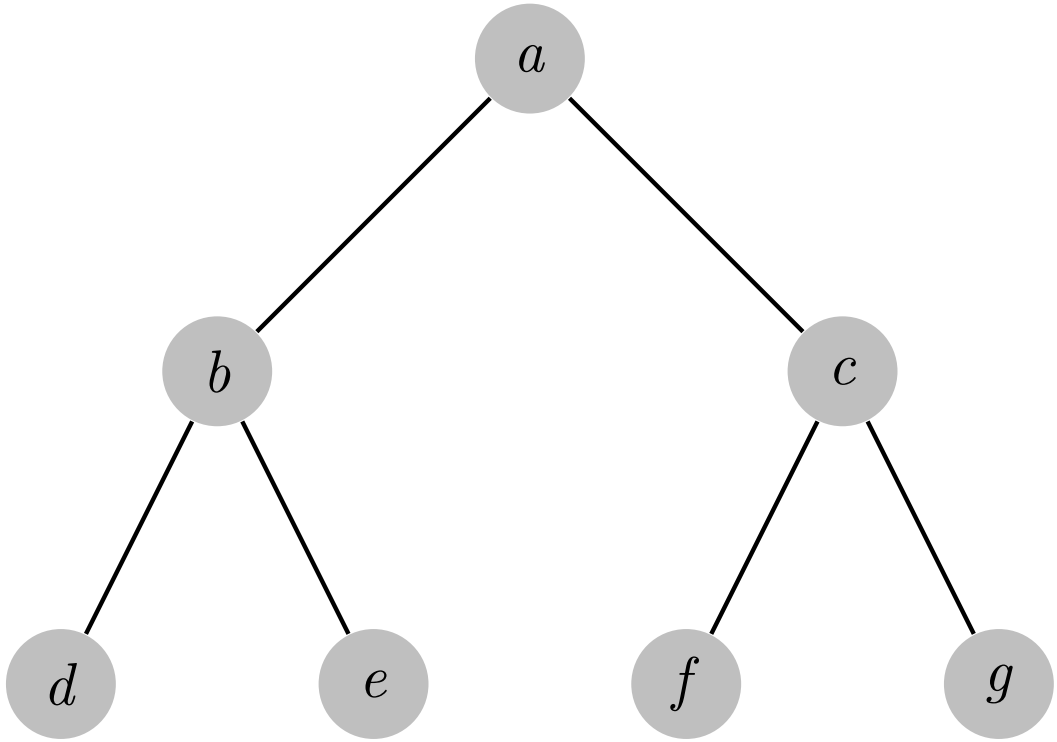
Breadth has the max priority



Breadth First Search (BFS)

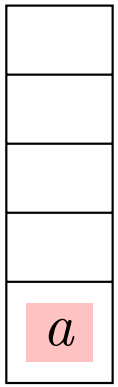
Let's do it again, to notice the **pattern** of nodes to be visited

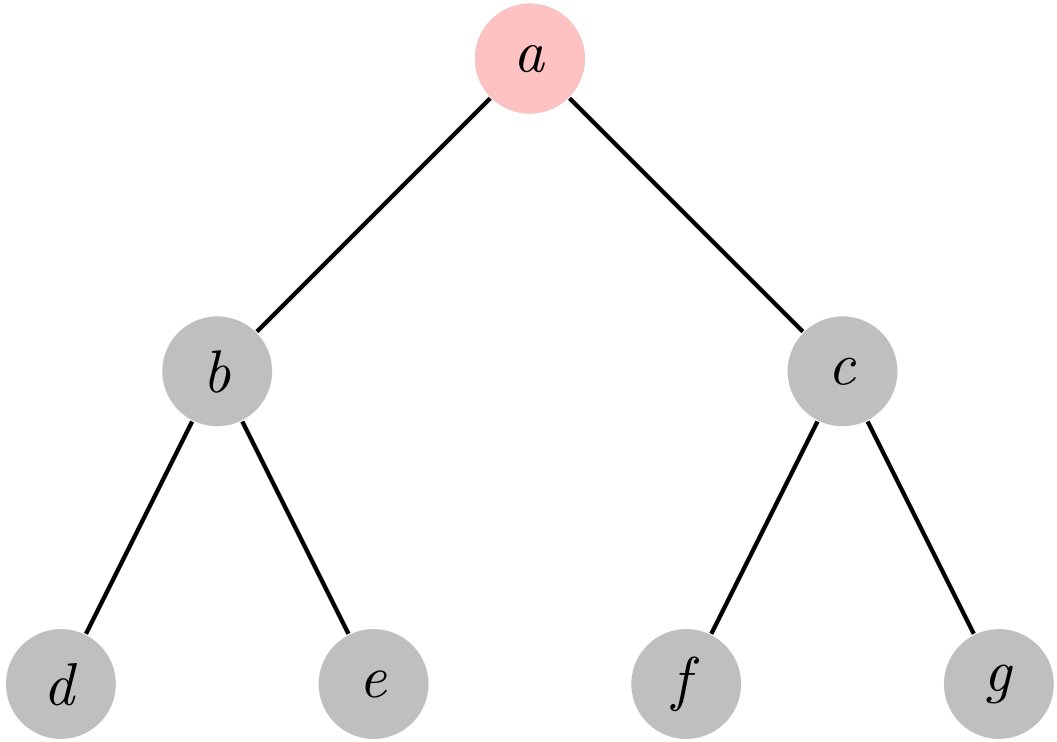




Breadth First Search (BFS)

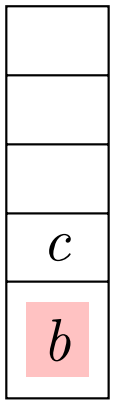
Let's do it again, to notice the **pattern** of nodes to be visited

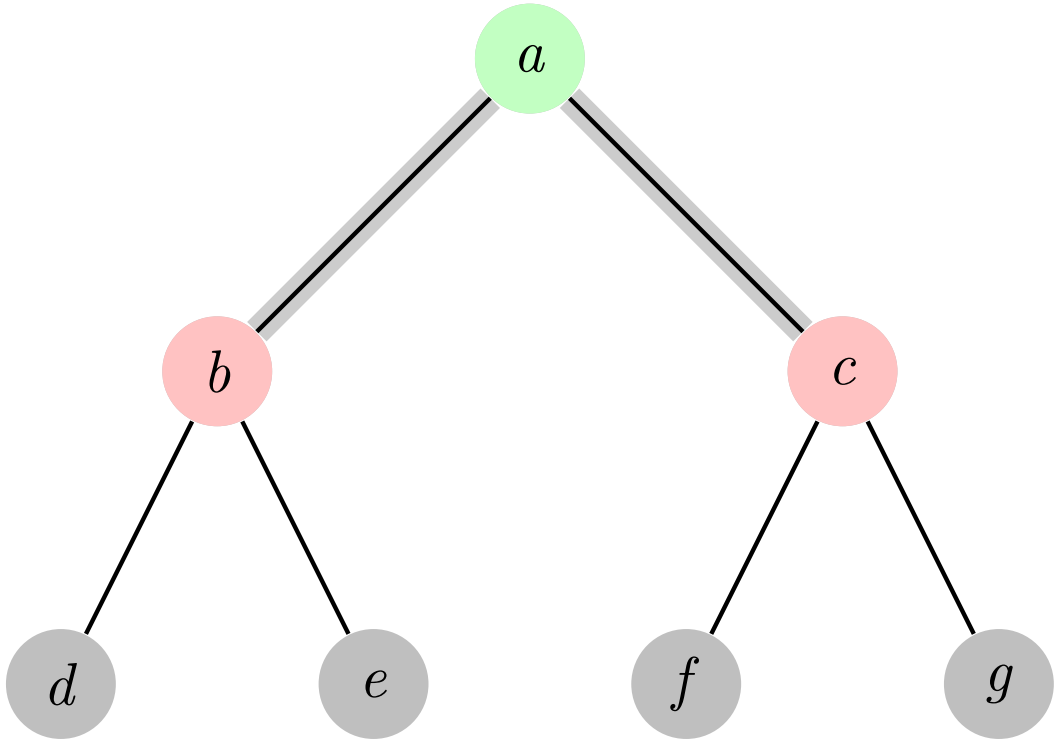




Breadth First Search (BFS)

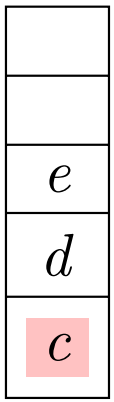
Let's do it again, to notice the **pattern** of nodes to be visited

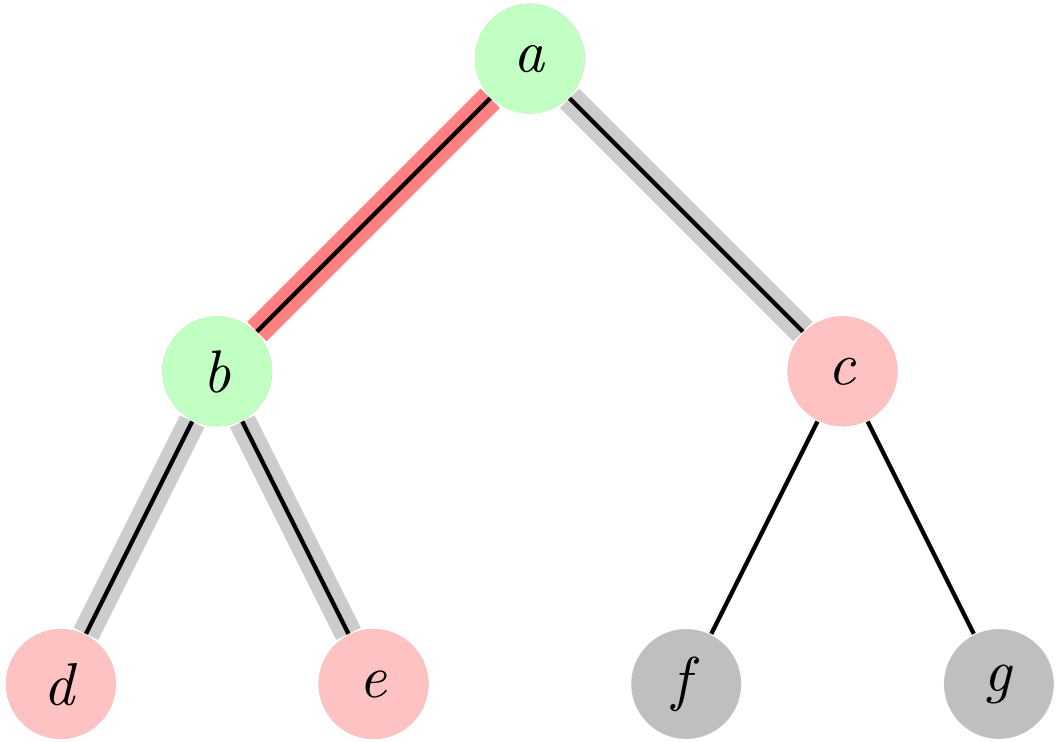




Breadth First Search (BFS)

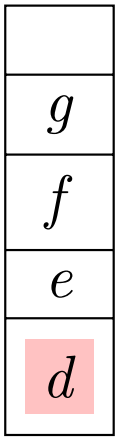
Let's do it again, to notice the **pattern** of nodes to be visited

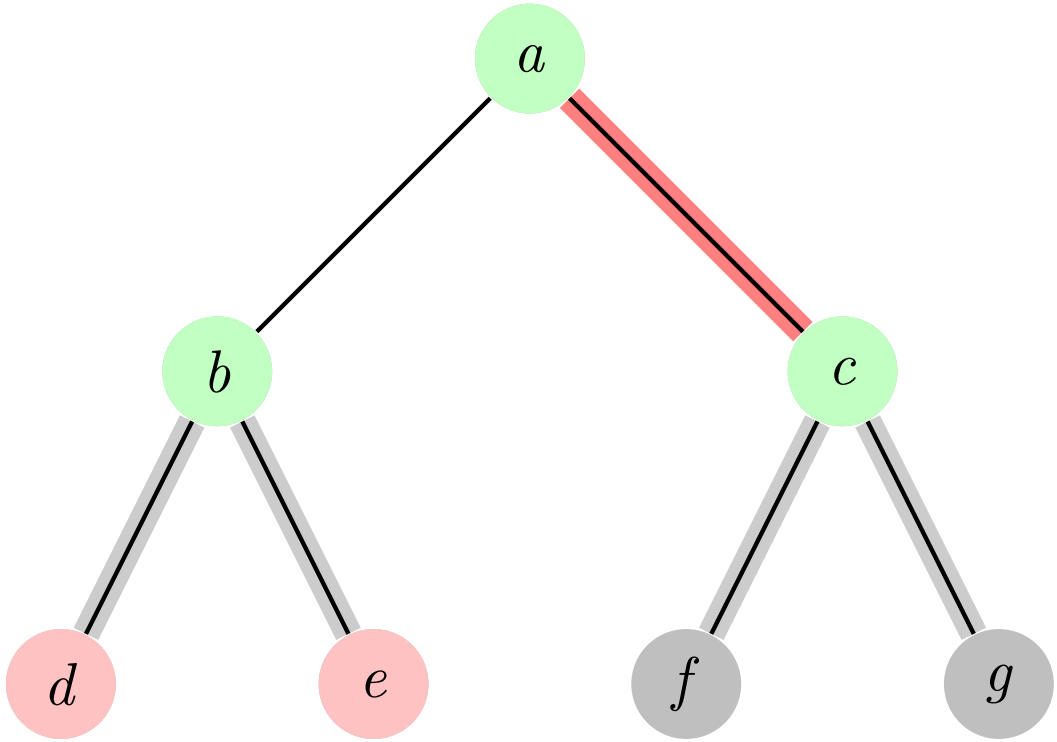




Breadth First Search (BFS)

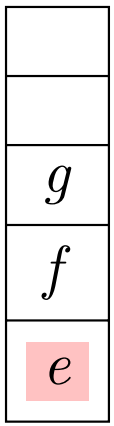
Let's do it again, to notice the **pattern** of nodes to be visited

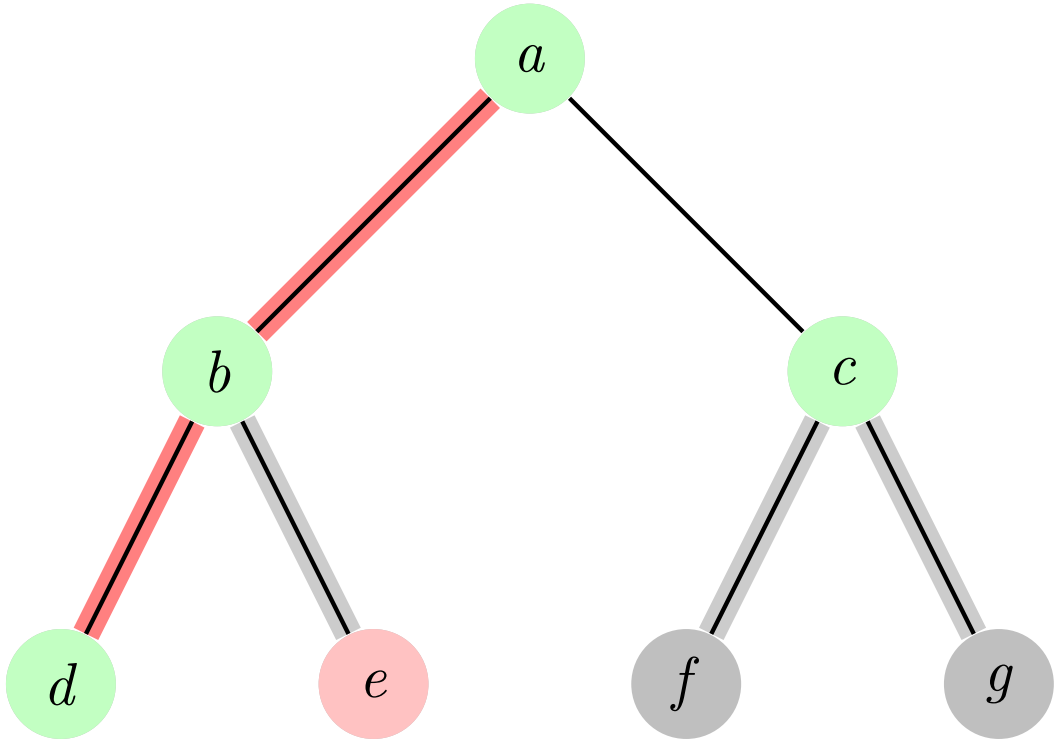




Breadth First Search (BFS)

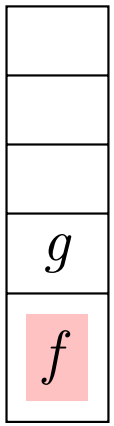
Did you get the pattern of nodes to be visited?

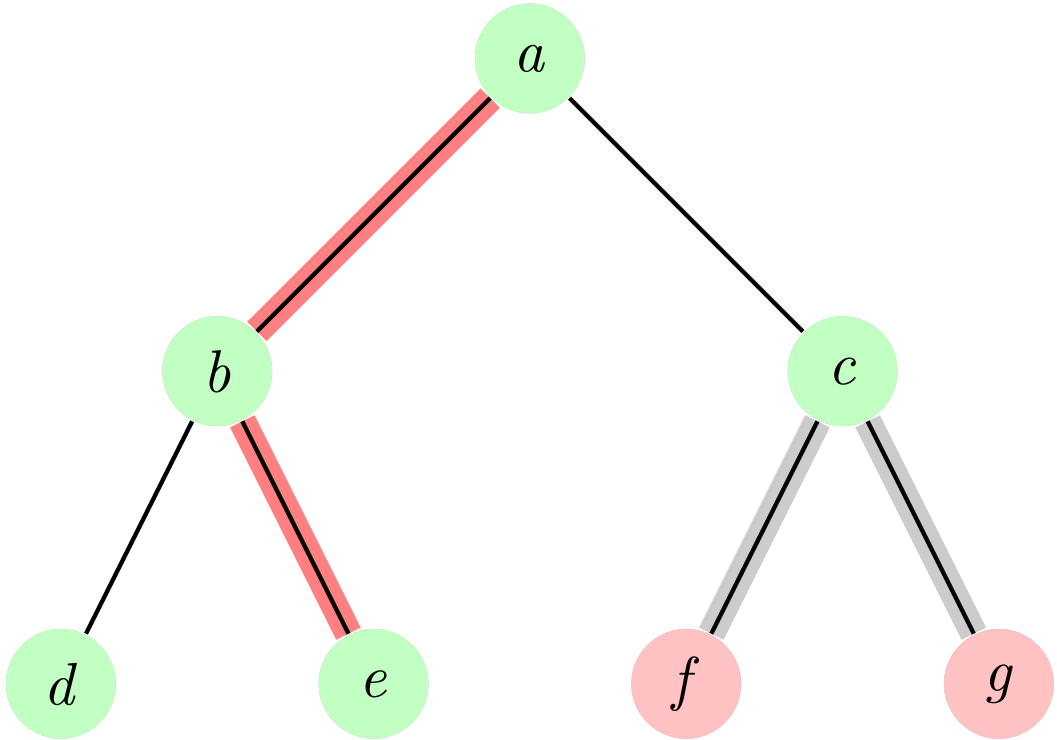




Breadth First Search (BFS)

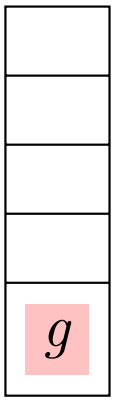
Did you get the pattern of nodes to be visited? **First** inserted element is **first** to explore.

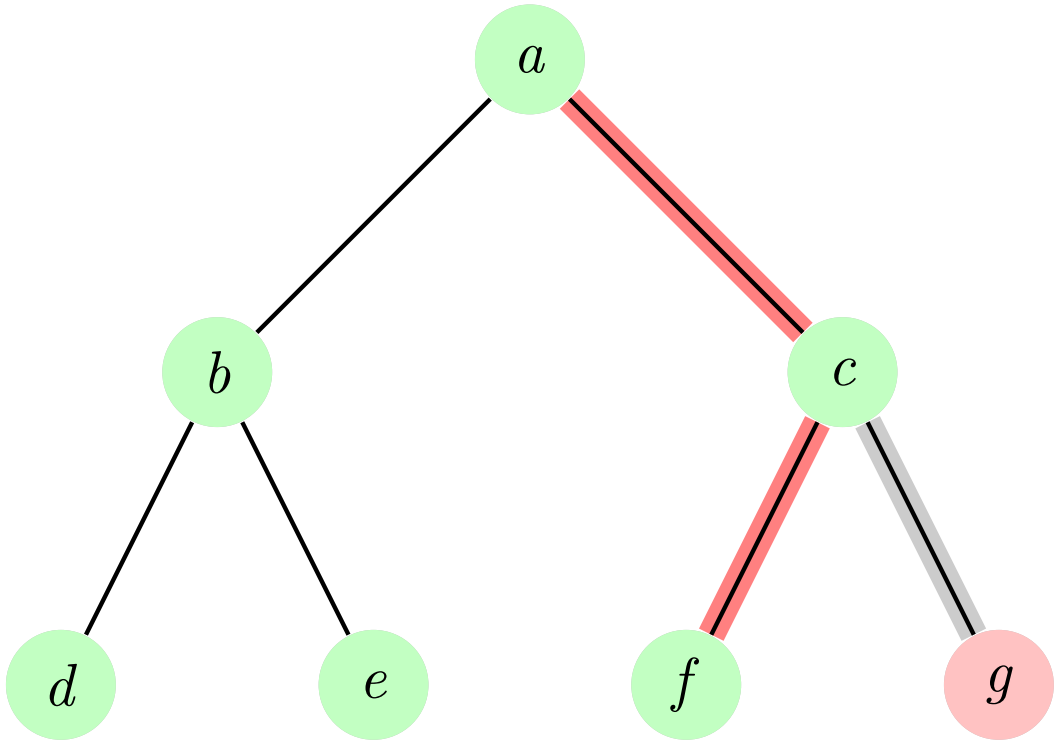




Breadth First Search (BFS)

Did you notice what might be this structure? Hint: First-in First-out

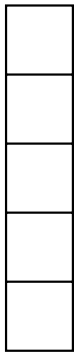


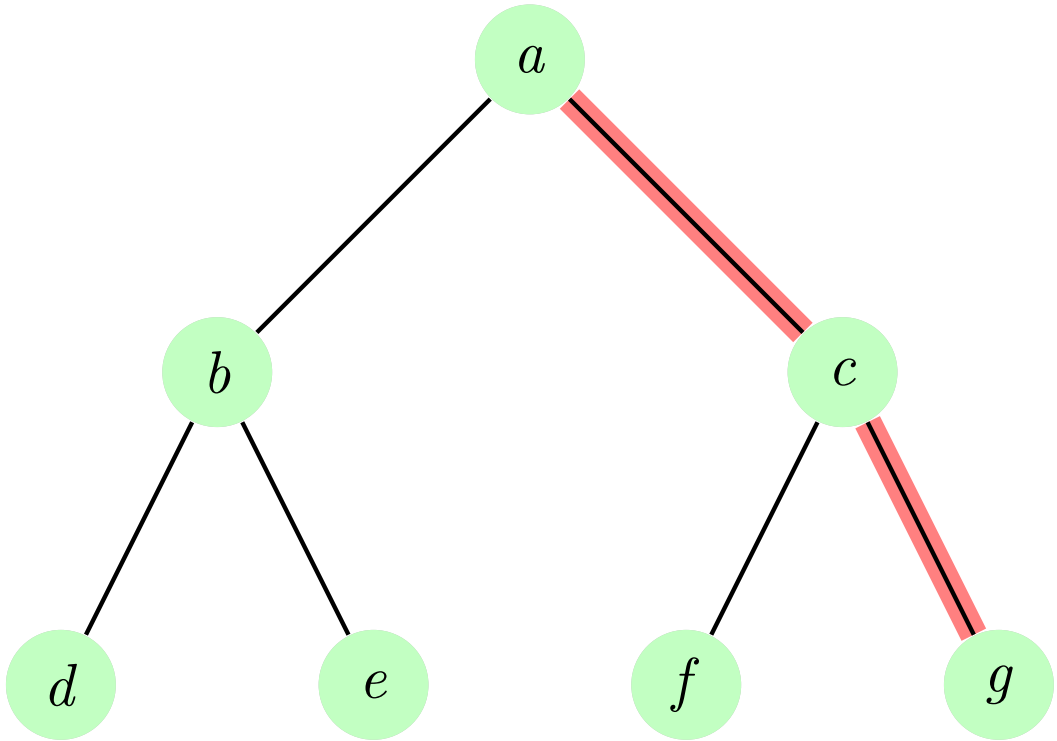


Breadth First Search (BFS)

Did you notice what might be this structure? Hint: First-in First-out

Yes, it is *Queue*





Breadth First Search (BFS)

Algorithm 2: BREADTH-FIRST($root$)

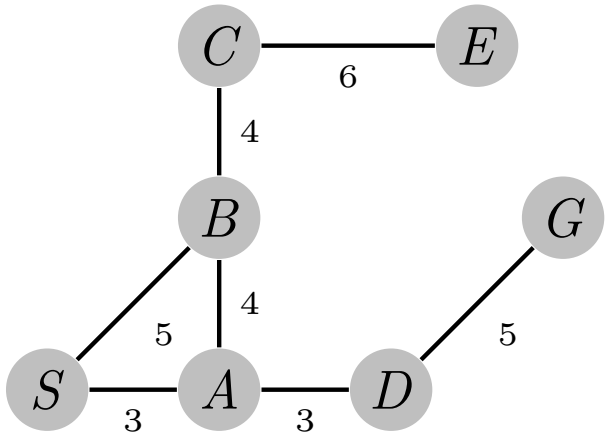
```
def S to be Queue;
visited ⟵ {};
S.enqueue(root);
while S ≠ ϕ do
    node ⟵ S.dequeue();
    if node ∉ visited then
        visited ⟵ visited ∪ {node};
        for n ∈ adjacent(node) do
            S.push(n);
        end
    end
end
```

Branch and Bound

Right now we have weights, now what should we prioritize?

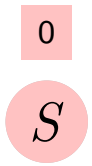
Branch and Bound

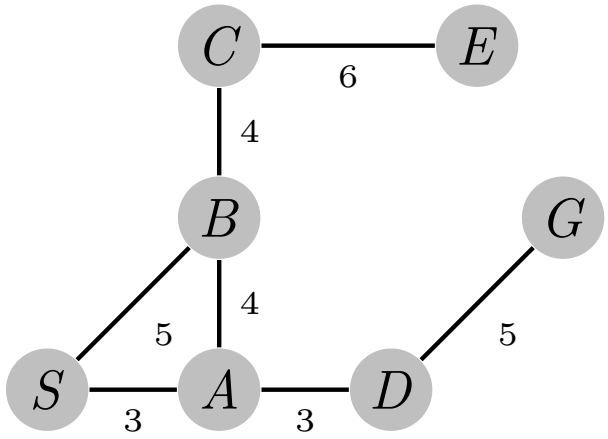
Right now we have weights, now what should we prioritize? Min/Max **weights**



Branch and Bound

Right now we have weights, now what should we prioritize? Min/Max **weights**

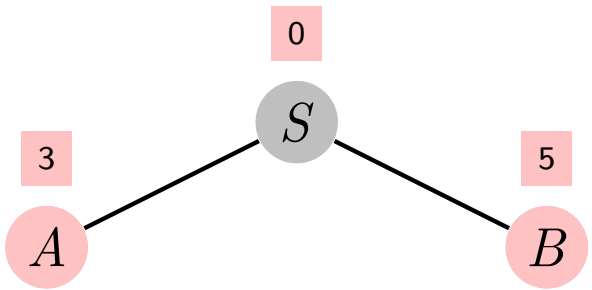


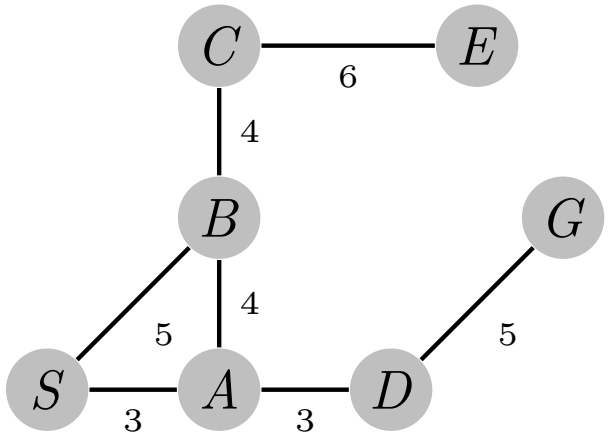


$\boxed{S:0}$

Branch and Bound

Right now we have weights, now what should we prioritize? Min/Max **weights**

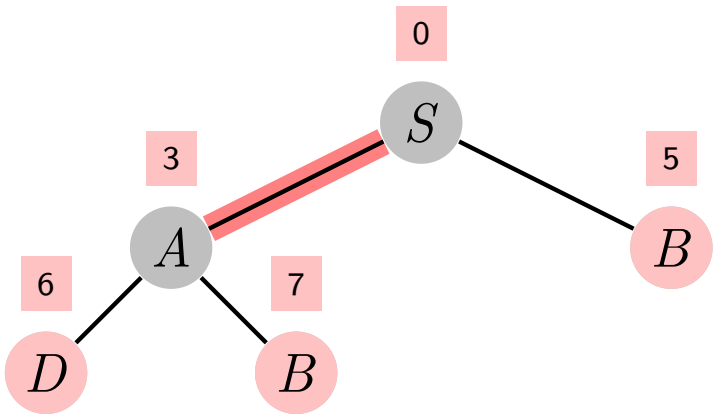


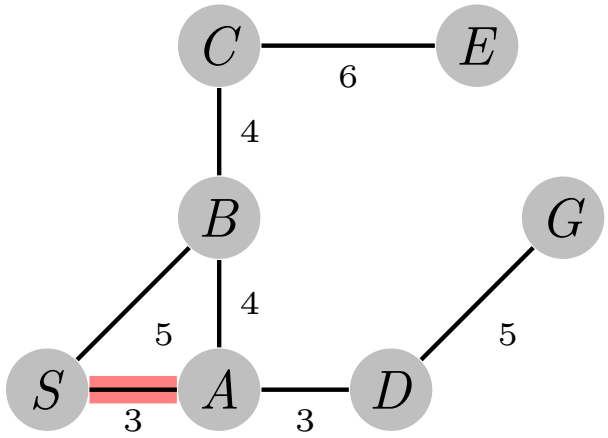


$A:3$	$B:5$

Branch and Bound

Right now we have weights, now what should we prioritize? Min/Max **weights**

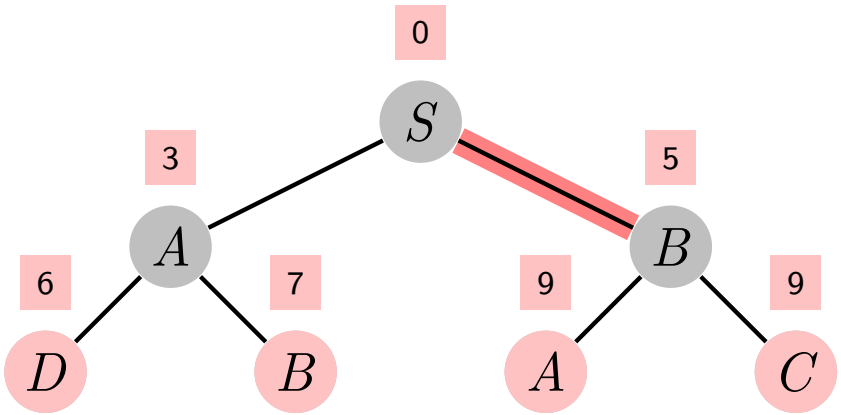


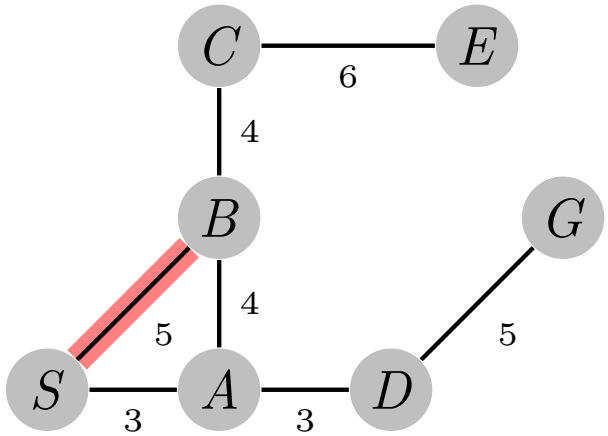


$B:5$	$D:6$	$B:7$

Branch and Bound

Can you notice the pattern of the structure?

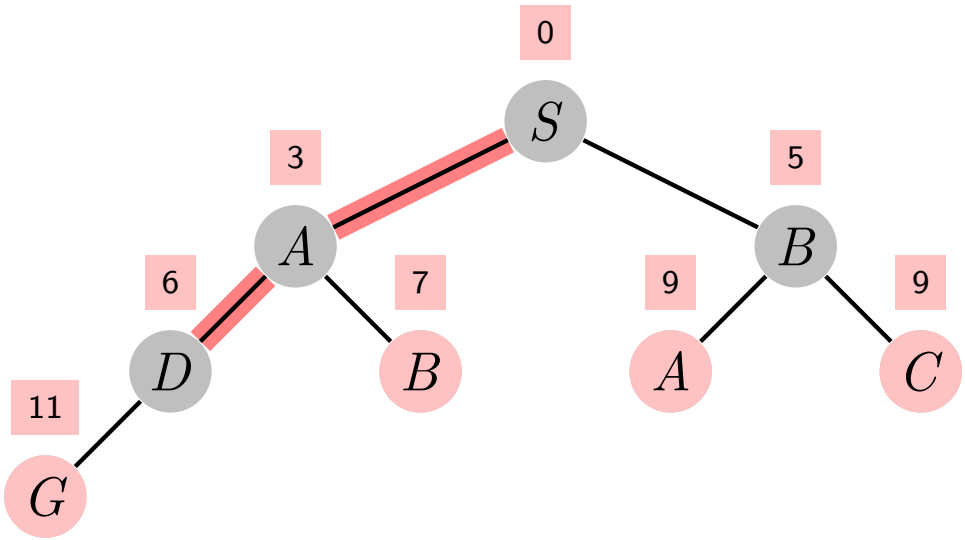


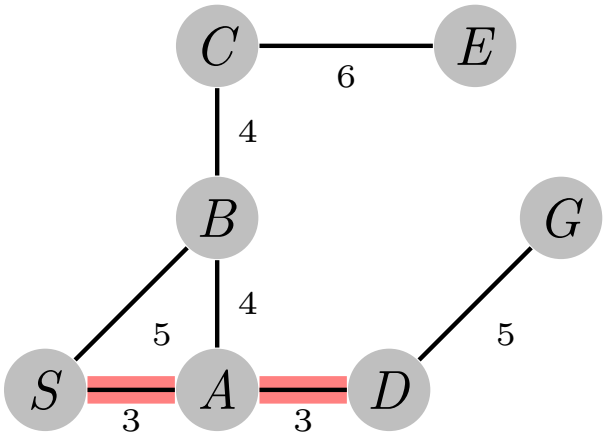


$D:6$	$B:7$	$A:9$	$C:9$

Branch and Bound

Can you notice the pattern of the structure?

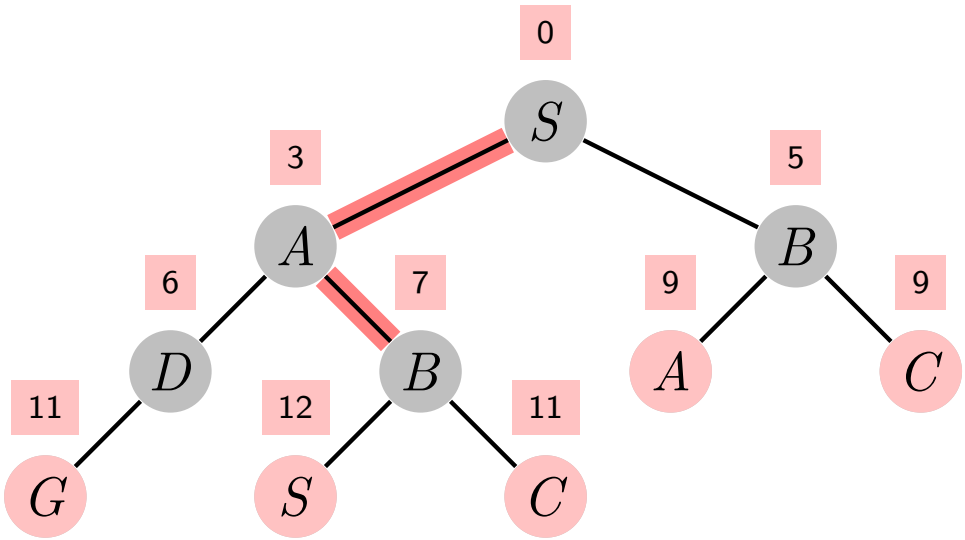


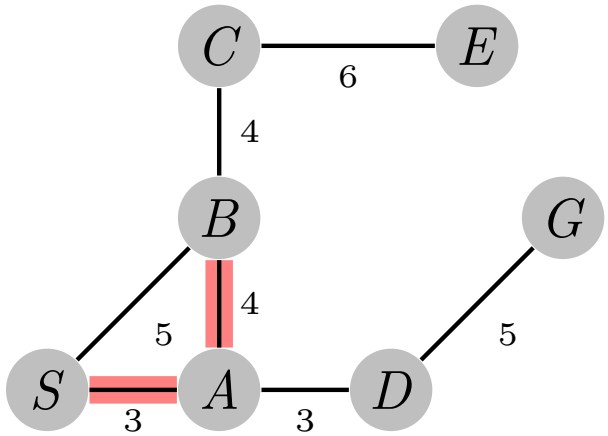


$B:7$	$A:9$	$C:9$	$G:11$

Branch and Bound

Have you noticed what happened?

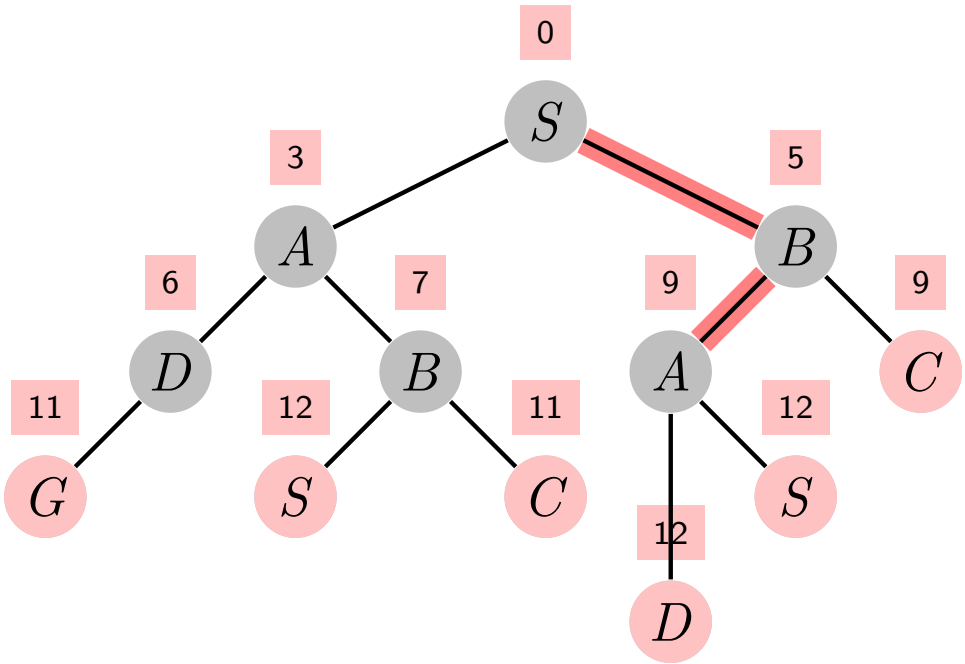


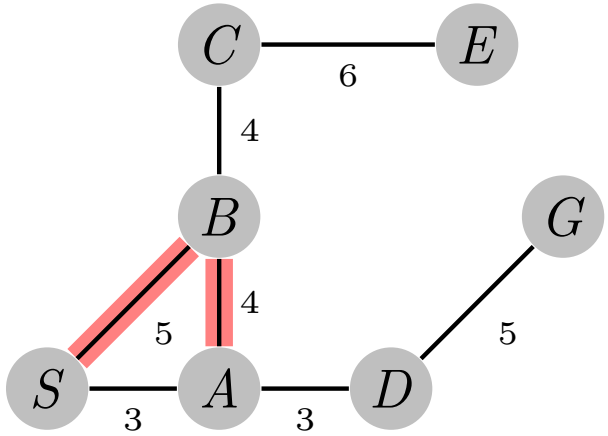


$A:9$	$C:9$	$C:11$	$G:11$	$S:12$

Branch and Bound

Now, what is such structure?

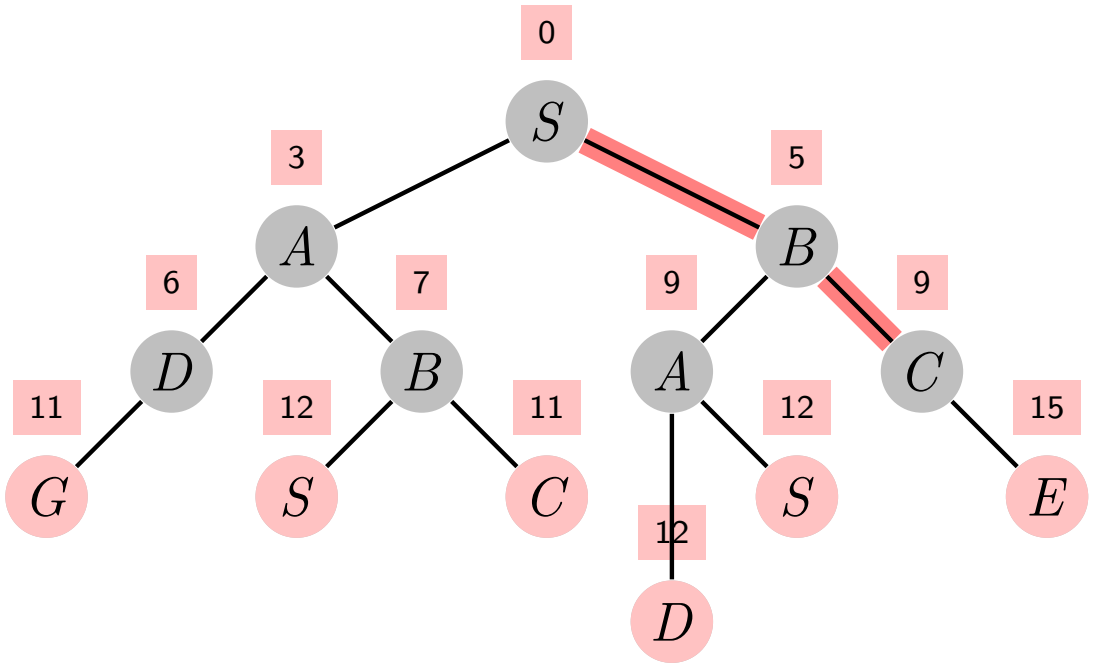


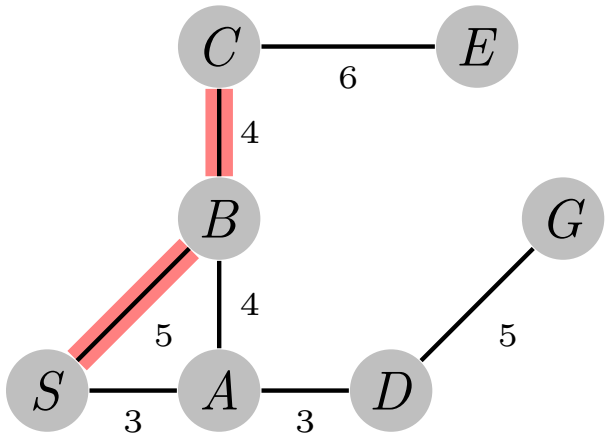


$C:9$	$C:11$	$G:11$	$S:12$	$D:12$	$S:12$

Branch and Bound

Now, what is such structure?

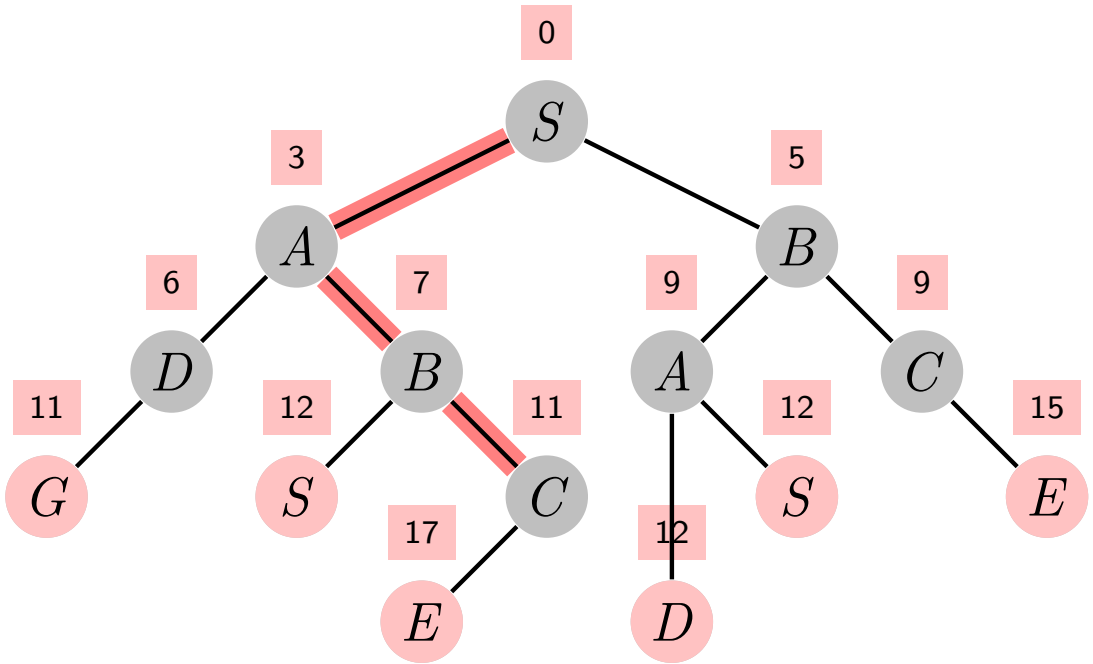


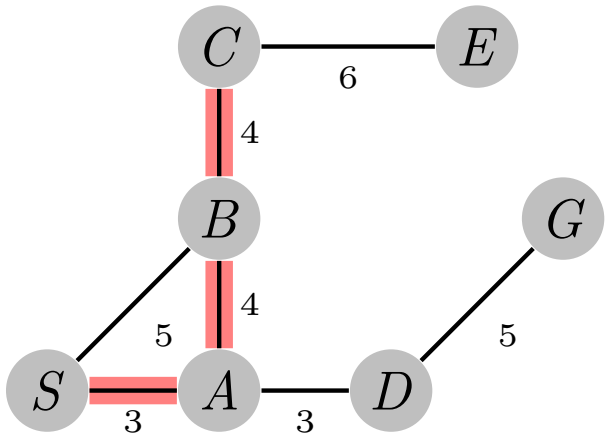


$C:11$	$G:11$	$S:12$	$D:12$	$S:12$	$E:15$

Branch and Bound

Now, what is such structure?

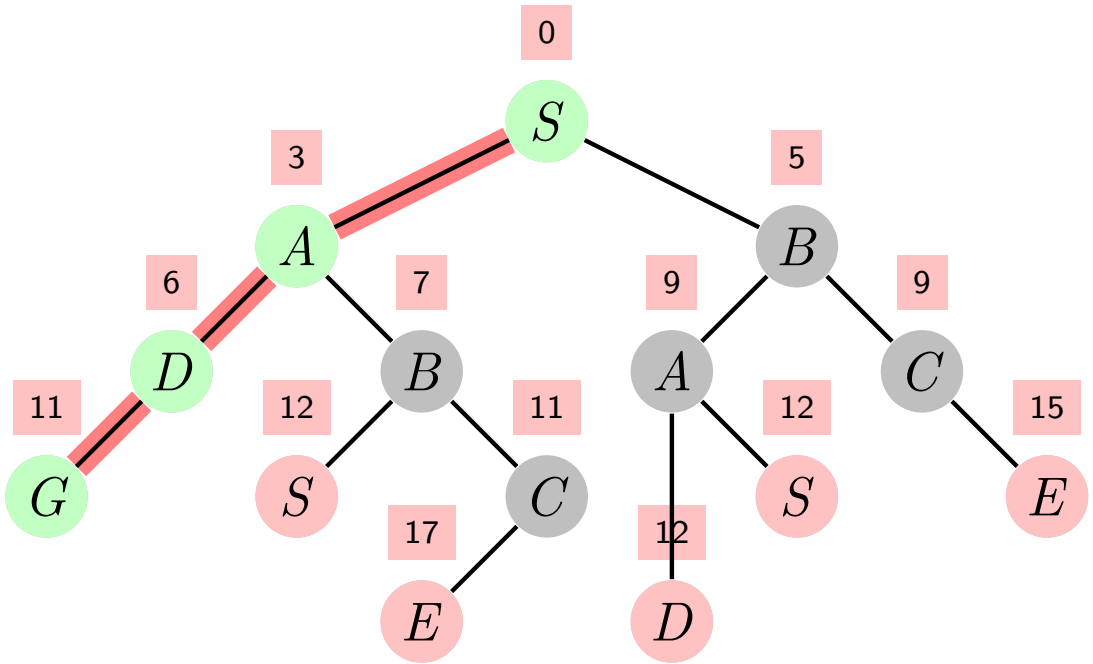


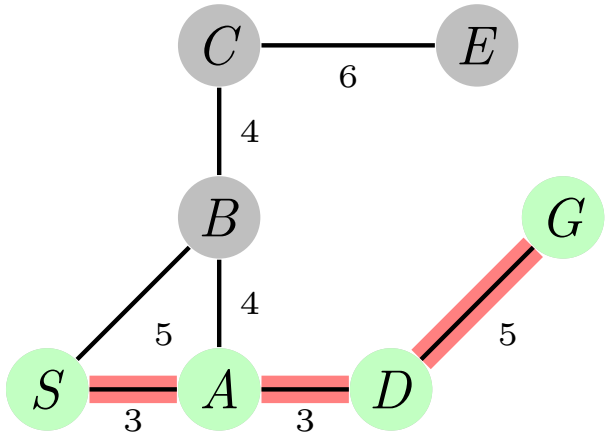


$G:11$	$S:12$	$D:12$	$S:12$	$E:15$	$E:17$

Branch and Bound

Yes, it is a **Priority Queue**, where the priority is the overall path cost.





$S:12$	$D:12$	$S:12$	$E:15$	$E:17$

Thanks!