



> StartBig

Second Practical

2023

MESA

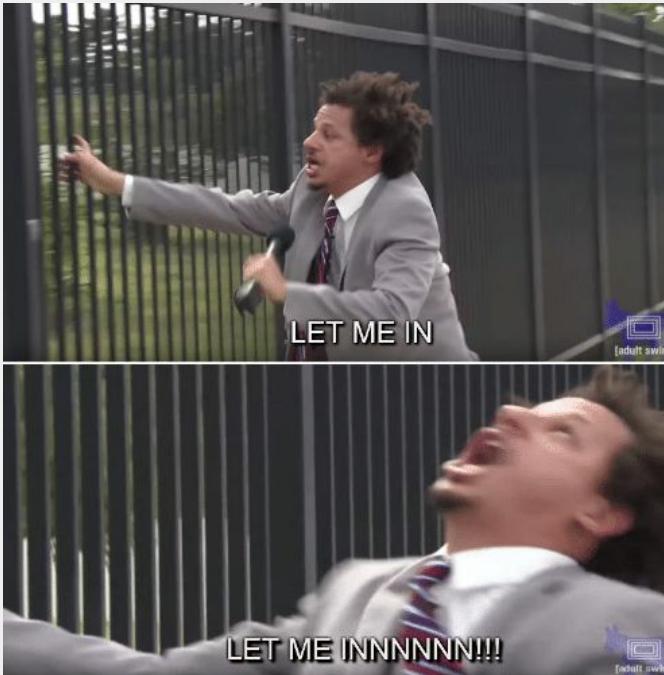


**Meeting is starting
soon...**

Enjoy some popcorns!



■ Receiving the Emails



Presence form



Submit here.

WHO IS THIS GUY?



Giovanni Manfredi

- MESA active Member since October 2022
- Participant at IT Sprint 2023 in Skopje, North Macedonia
- HO and creator of StartBig
- Tech savvy since I was 6 years old

 AGENDA

- **MESA** - What is MESA?
- What **does** MESA do?
- How do I **join**?
- What about **StartBig**?

■ What is MESA?

Local Association



**Milan Engineering
Student Association**

European Association



Electrical Engineering **STudent
European **AssoCiation****

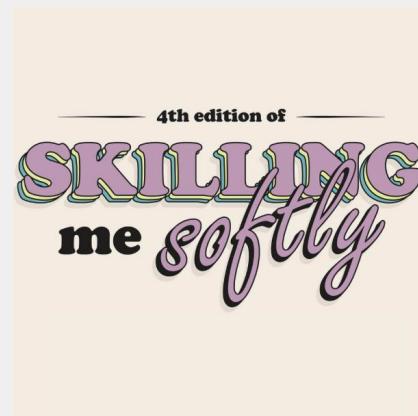
■ What does MESA do?

EVENTS!

Hard Skills



Soft Skills



An International Network



24 countries



43 universities



5000+ people



5 regions

■ How do I join?



Scan the QR and
click on the section Join us

■ What about StartBig?

>startBig

Initiating coding interview preparation in process...



CareerService session

How do we get to do a coding interview?



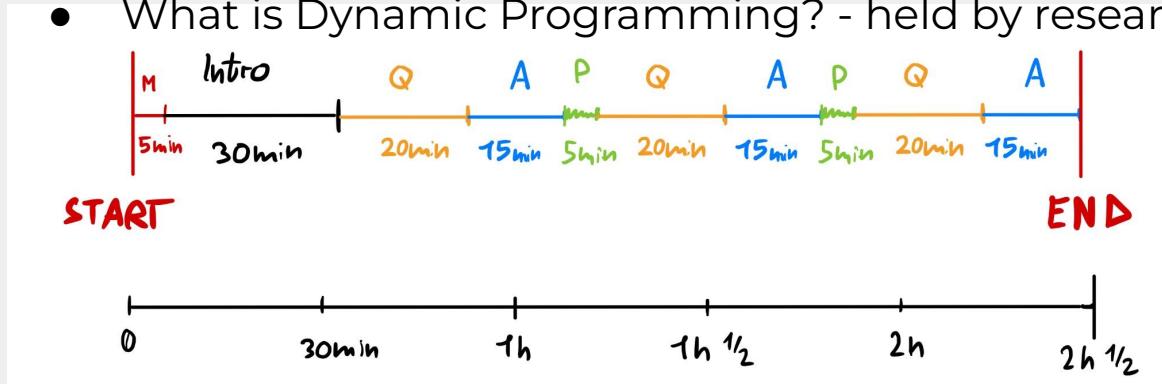
CareerService is here to help us find that out!

■ 3 Practical sessions

How can we prepare for a coding interview?

Content of the sessions:

- Introduction to coding interview questions - held by me
- Trees & Graphs (DSF & BSF algorithms) - held by researcher Davide Yi Xian Hu
- What is Dynamic Programming? - held by researcher Nicolò Felicioni



■ Company session

How is the true experience behind coding interviews?

Andrea from **Oracle** will help us understand that!

There will be a small ice-breaker interview and then open questions from you!

■ Tutors here to help!



■ STAGE TO ...

Davide Yi Xian Hu

PhD Student in
Information Technology



POLITECNICO
MILANO 1863

TREES AND GRAPHS

DAVIDE YI XIAN Hu

EMAIL: DAVIDEYI.HU@POLIMI.IT



POLITECNICO
MILANO 1863



INTRO PRESENTATION

DAVIDE YI XIAN Hu

👉 Ph.D. STUDENT INFORMATION TECHNOLOGY

👉 MAIL: DAVIDEYi.Hu@POLIMI.IT

👉 RESEARCH INTERESTS:

🔍 DEEP LEARNING TESTING

🔍 COMPUTER VISION APPLICATIONS



□ TREES

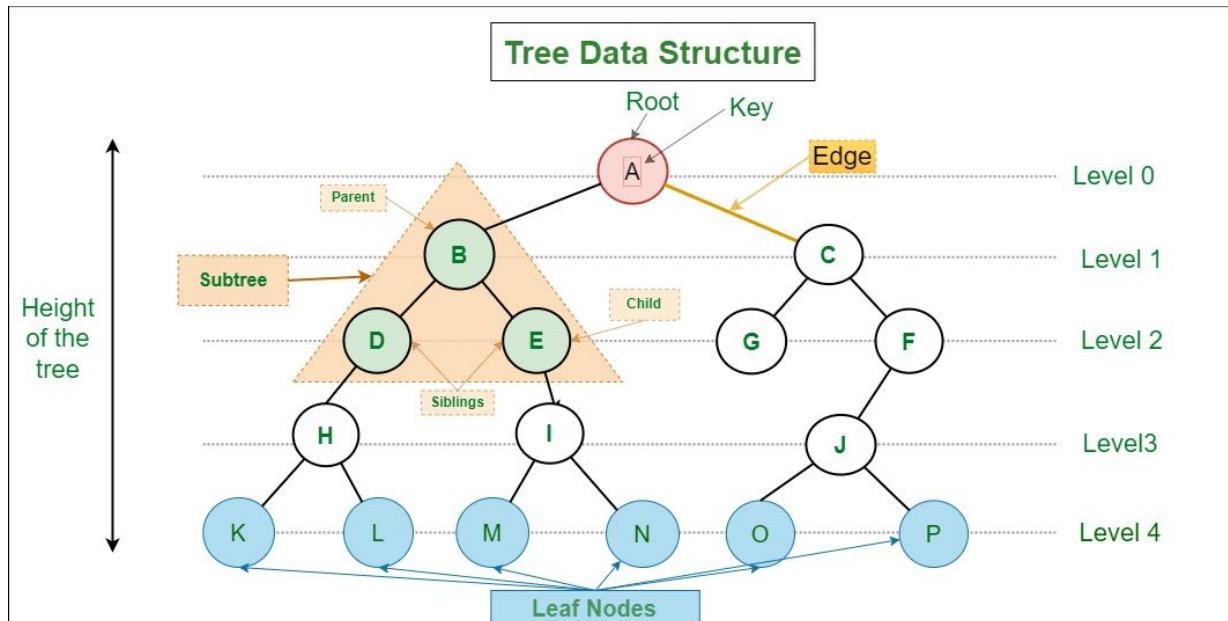


TREE DEFINITION

TREE DATA STRUCTURE

- 👉 A **TREE**  IS A **HIERARCHICAL** DATA STRUCTURE WITH A SET OF NODES
- 👉 EACH NODE CAN HAVE **MANY CHILDREN**
- 👉 EACH NODE HAVE **EXACTLY ONE PARENT**
(EXCEPT FOR THE ROOT NODE)
- 👉 **No CYCLES/LOOPS**

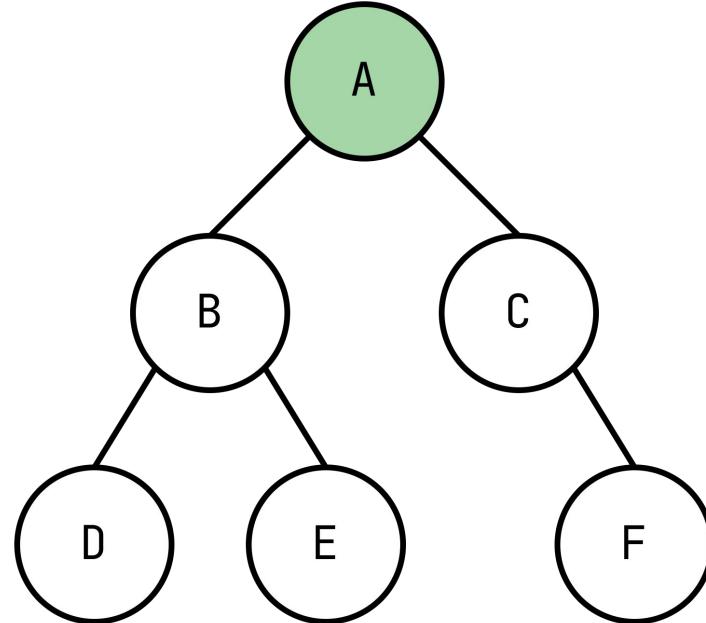
EXAMPLE OF TREE



BINARY TREE



- 👉 ONE OF THE **MOST COMMONLY** USED TYPE OF TREES
- 👉 EACH NODE CAN HAVE AT MOST **TWO CHILDREN** (K-ARY TREE WITH K=2)

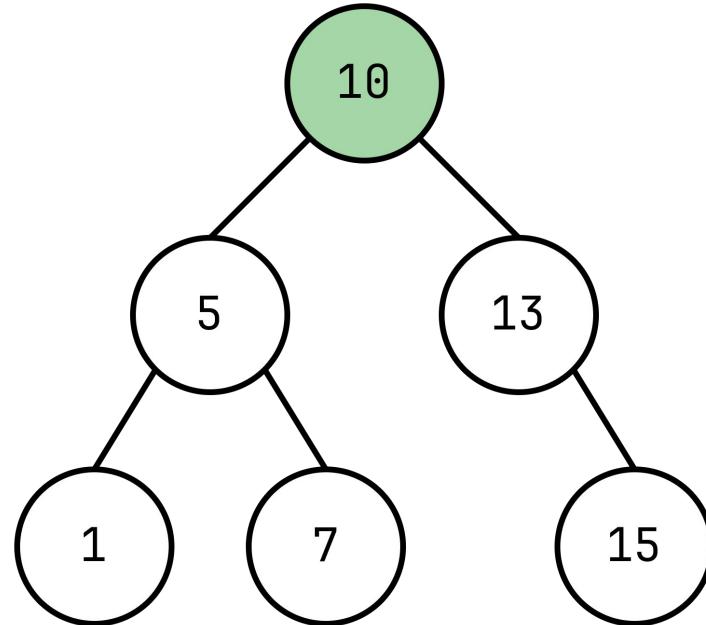


BINARY SEARCH TREE



THE **KEY** OF A NODE IS:

- 👉 GREATER THAN THE KEYS IN
THE LEFT SUBTREE.
- 👉 LESS THAN THE KEYS IN
THE RIGHT SUBTREE.

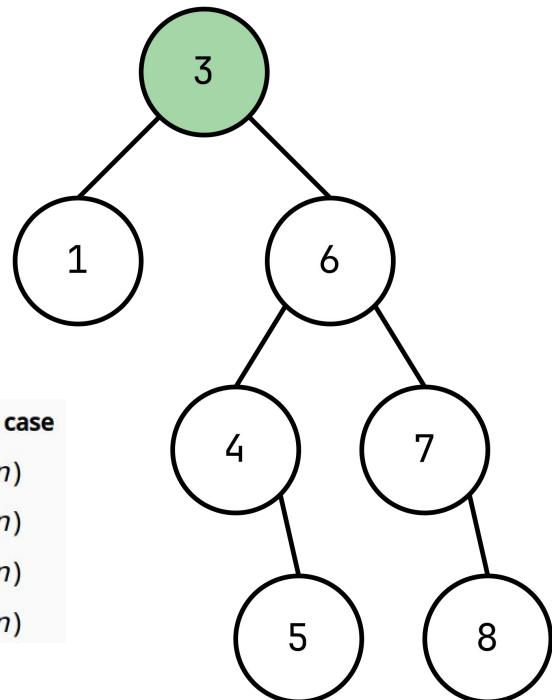


BALANCED TREES

BINARY SEARCH TREES
ARE **NOT BALANCED**

BALANCED TREE:
THE HEIGHT OF THE
LEFT AND THE RIGHT
SUBTREE FOR EACH NODE
IS EITHER ZERO OR ONE

Algorithm	Average	Worst case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$



BALANCED TREES

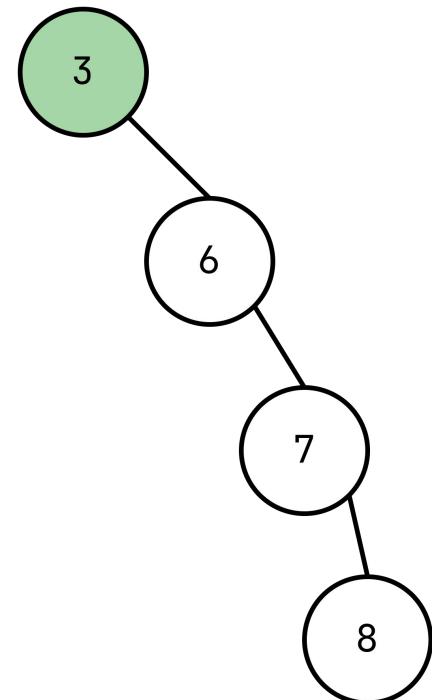


BINARY SEARCH TREES
ARE **NOT BALANCED**

BALANCED TREE:

THE HEIGHT OF THE
LEFT AND THE RIGHT
SUBTREE FOR EACH NODE
IS EITHER ZERO OR ONE

Algorithm	Average	Worst case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$



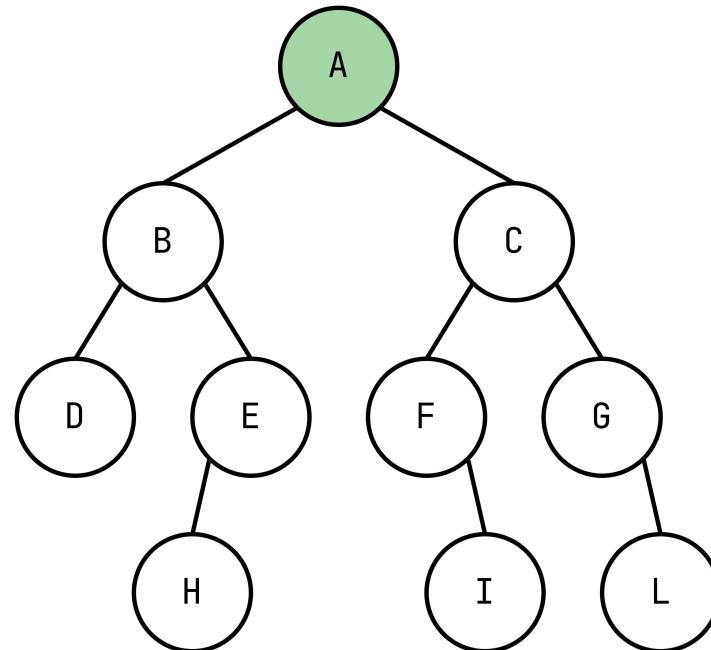
BALANCED SEARCH TREES



EXAMPLES OF BALANCED TREES:

- 👉 AVL (BINARY) TREE
- 👉 RED-BLACK (BINARY) TREE
- 👉 B-TREE (K-ARY)

SEARCH, INSERTION AND DELETION
COST **O(LOG N)**



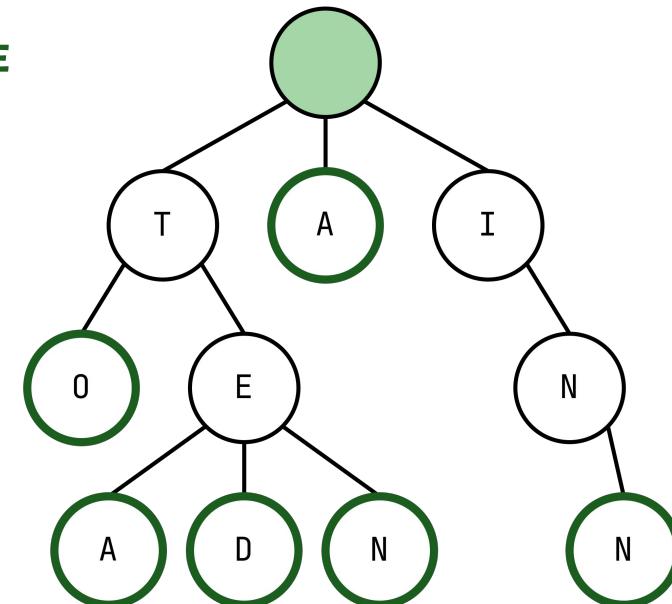
TRIE

KNOWN ALSO AS **DIGITAL TREE** OR **PREFIX TREE**

STRING-INDEXED LOOKUP DATA STRUCTURE

APPLICATIONS:

- 👉 AUTOCOMPLETE
- 👉 FULL-TEXT SEARCH
- 👉 BLOCKCHAIN (MERKLE-PATRICIA)



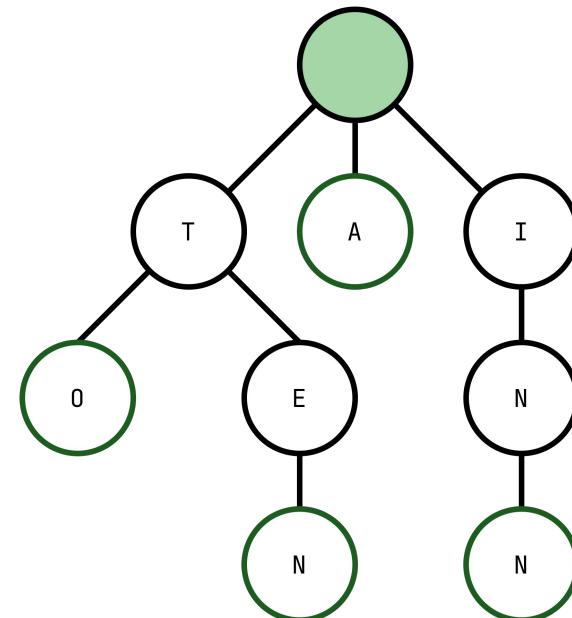
TRIE

KNOWN ALSO AS **DIGITAL TREE** OR **PREFIX TREE**

STRING-INDEXED LOOKUP DATA STRUCTURE

APPLICATIONS:

- 👉 AUTOCOMPLETE
- 👉 FULL-TEXT SEARCH
- 👉 BLOCKCHAIN (MERKLE-PATRICIA)



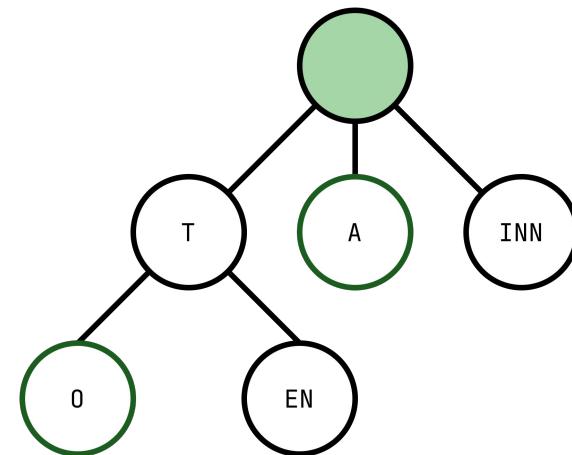
TRIE

KNOWN ALSO AS **DIGITAL TREE** OR **PREFIX TREE**

STRING-INDEXED LOOKUP DATA STRUCTURE

APPLICATIONS:

- 👉 AUTOCOMPLETE
- 👉 FULL-TEXT SEARCH
- 👉 BLOCKCHAIN (MERKLE-PATRICIA)



COMPRESSED TRIE





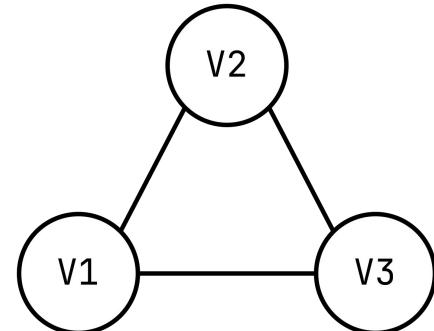
□GRAPHS



GRAPHS DEFINITION

GRAPH DATA STRUCTURE

- 👉 A **GRAPH** 🍇 IS A DATA STRUCTURE WITH A SET OF VERTICES AND EDGES
- 👉 GRAPHS CAN BE **DIRECTED (DiGRAPH)** OR **UNDIRECTED (GRAPH)**
- 👉 BOTH VERTICES AND EDGES CAN HOLD VALUES
- 👉 TREES ARE A PARTICULAR TYPE OF GRAPH



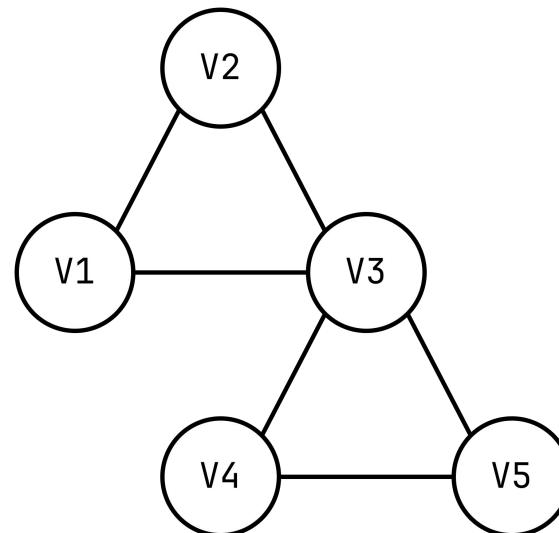
GRAPH IMPLEMENTATION

GRAPHS CAN BE IMPLEMENTED AS:

👉 **ADJACENCY LIST**

👉 **ADJACENCY MATRIX**

👉 **INCIDENCE MATRIX**

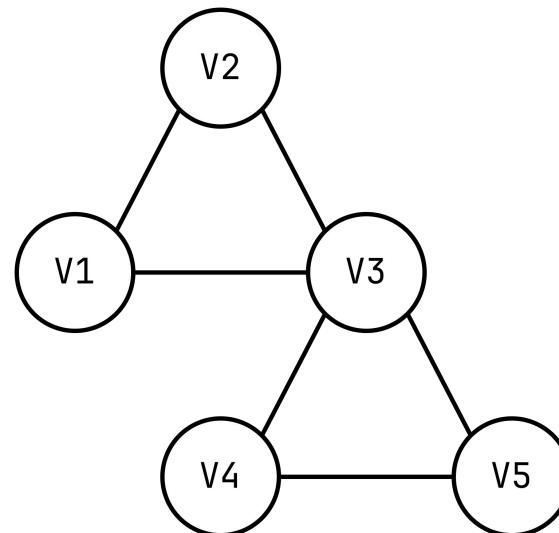


GRAPH IMPLEMENTATION

ADJACENCY LIST

LIST OF VERTICES AND
EACH VERTEX HAS A LIST OF
ADJACENT VERTICES

```
[ v1 [v2, v3],  
  v2 [v1, v3],  
  v3 [v1, v2, v3, v4],  
  v4 [v3, v5]  
  v5 [v3, v4]  
 ]
```



GRAPH IMPLEMENTATION

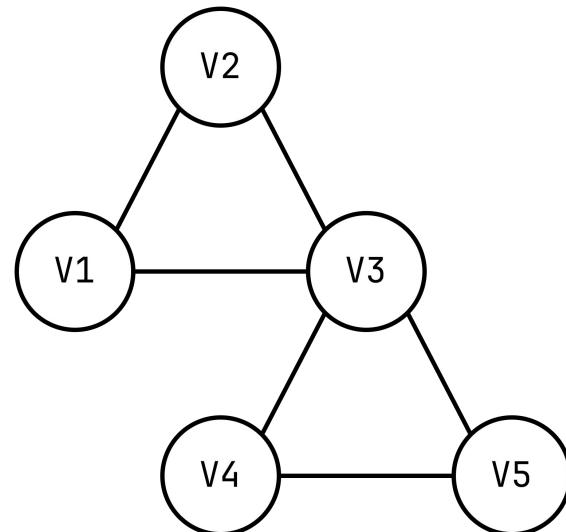
ADJACENCY MATRIX

TWO-DIMENSIONAL MATRIX

ROWS REPRESENTS **SOURCE** VERTICES

COLUMNS REPRESENTS **DESTINATION** VERTICES

	V1	V2	V3	V4	V5
V1	-	X	X	0	0
V2	X	-	X	0	0
V3	X	X	-	X	X
V4	0	0	X	-	X
V5	0	0	X	X	-



GRAPH IMPLEMENTATION

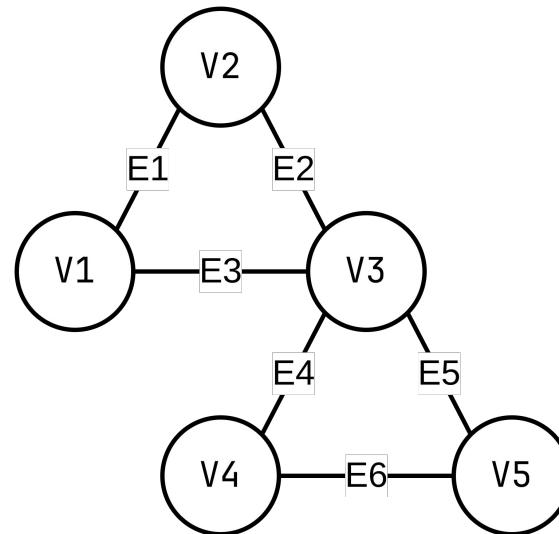
INCIDENCE MATRIX

TWO-DIMENSIONAL MATRIX

ROWS REPRESENTS VERTICES

COLUMNS REPRESENTS EDGES

	E1	E2	E3	E4	E5	E6
V1	X	0	X	0	0	0
V2	X	X	0	0	0	0
V3	0	X	X	X	0	X
V4	0	0	0	X	X	0
V5	0	0	0	0	X	X





GRAPH IMPLEMENTATION

	Adjacency list	Adjacency matrix	Incidence matrix
Store graph	$O(V + E)$	$O(V ^2)$	$O(V \cdot E)$
Add vertex	$O(1)$	$O(V ^2)$	$O(V \cdot E)$
Add edge	$O(1)$	$O(1)$	$O(V \cdot E)$
Remove vertex	$O(E)$	$O(V ^2)$	$O(V \cdot E)$
Remove edge	$O(V)$	$O(1)$	$O(V \cdot E)$
Are vertices x and y adjacent (assuming that their storage positions are known)?	$O(V)$	$O(1)$	$O(E)$
Remarks	Slow to remove vertices and edges, because it needs to find all vertices or edges	Slow to add or remove vertices, because matrix must be resized/copied	Slow to add or remove vertices and edges, because matrix must be resized/copied



GRAPH COMPRESSED REPRESENTATION

SPARSE MATRIX

IS THE MOST EFFICIENT WAY TO REPRESENT
A MATRIX WITH LOT OF ZEROS

FAST CONSTRUCTION:

LIL - LIST OF LISTS (ADJACENCY LIST)

DOK - DICTIONARY OF KEYS

COO - COORDINATE LIST

(FAST DATA ACCESS):

CSR - COMPRESSED SPARSE ROW

CSC - COMPRESSED SPARSE COLUMN

GRAPH COMPRESSED REPRESENTATION

LIL

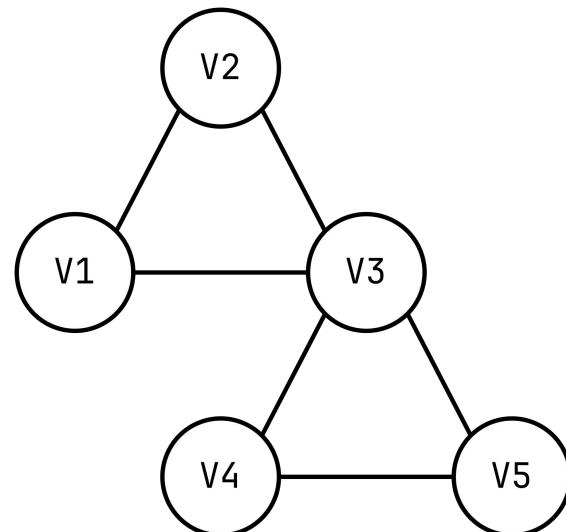
```
[ v1 [v2, v3],  
  v2 [v1, v3],  
  v3 [v1, v2, v3, v4],  
  v4 [v3, v5]  
  v5 [v3, v4]  
]
```

COO

```
[ (v1, v2),  
  (v1, v3),  
  (v2, v3),  
  (v3, v4),  
  (v3, v5),  
  (v4, v5)]
```

DOK

```
{ (v1, v2),  
  (v1, v3),  
  (v2, v3),  
  (v3, v4),  
  (v3, v5),  
  (v4, v5)}
```





SEARCH ALGORITHMS



SEARCH ALGORITHMS

SEARCH ALGORITHMS

- 👉 THEY ARE USED TO TRAVERSE OR EXPLORE A GRAPH OR TREE IN ORDER TO FIND A SPECIFIC ELEMENT OR PATH.
- 👉 THE MOST POPULAR ALGORITHMS ARE:
DFS (DEPTH FIRST SEARCH) AND
BFS (BREADTH FIRST SEARCH).



DFS

DEPTH FIRST SEARCH

DEPTH FIRST SEARCH

- 👉 USES A STACK DATA STRUCTURE.
- 👉 VISITS NODES DEPTH-FIRST (I.E., GOES AS DEEP AS POSSIBLE ALONG EACH BRANCH BEFORE BACKTRACKING).
- 👉 GOOD FOR TRAVERSING AN ENTIRE TREE OR GRAPH.
- 👉 CAN GET STUCK IN AN INFINITE LOOP IF THE GRAPH HAS CYCLES.

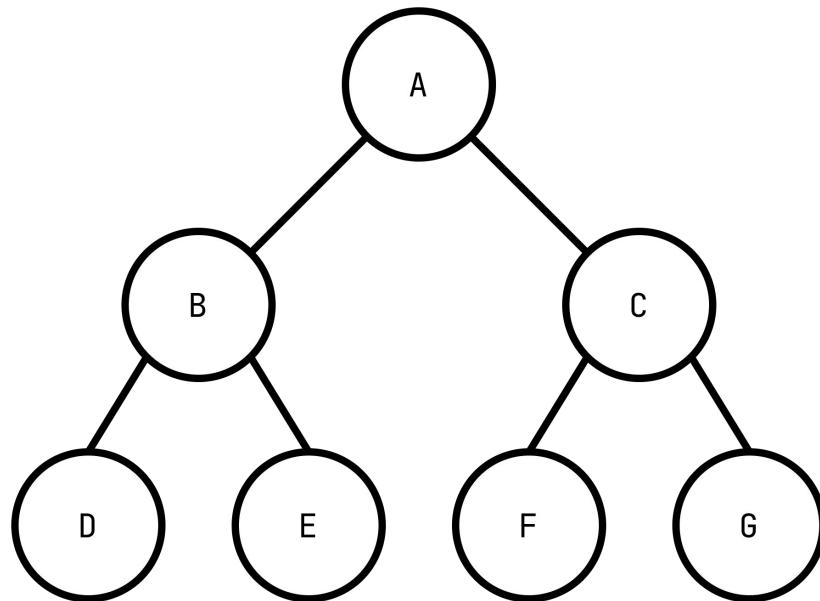
DEPTH FIRST SEARCH

EXAMPLE:

FIND A PATH FROM A TO F

VISITED: []

STACK: []



DEPTH FIRST SEARCH

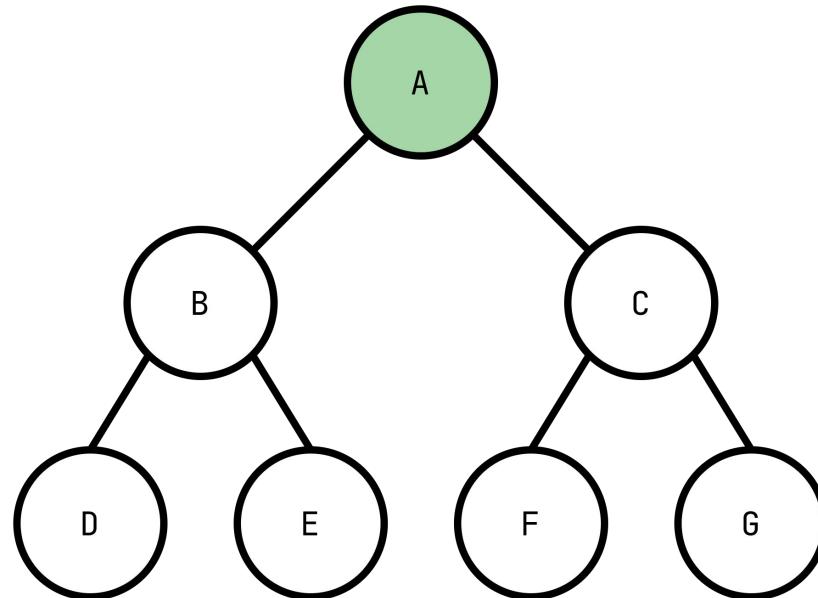
EXAMPLE:

FIND A PATH FROM A TO F

VISIT NODE A

VISITED: [A]

STACK: [B, C]



DEPTH FIRST SEARCH

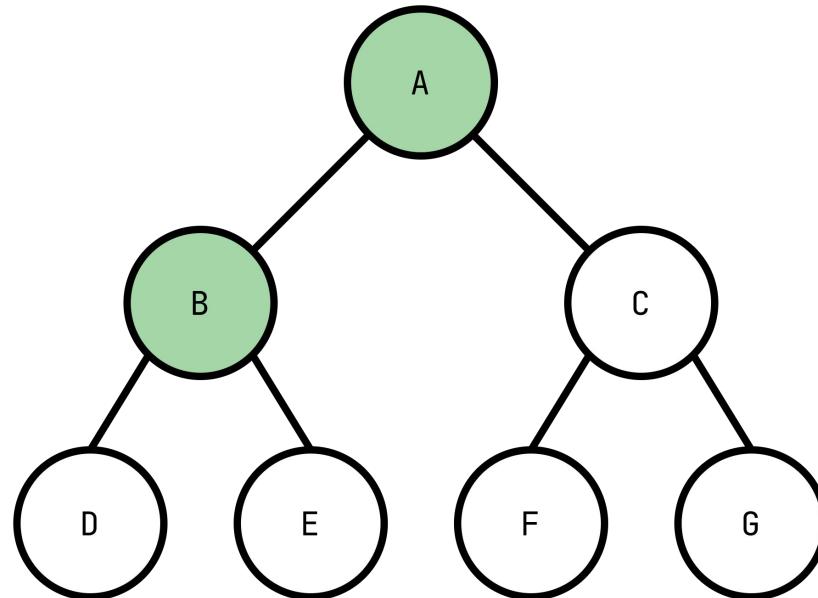
EXAMPLE:

FIND A PATH FROM A TO F

VISIT NODE B

VISITED: [A, B]

STACK: [D, E, C]



DEPTH FIRST SEARCH

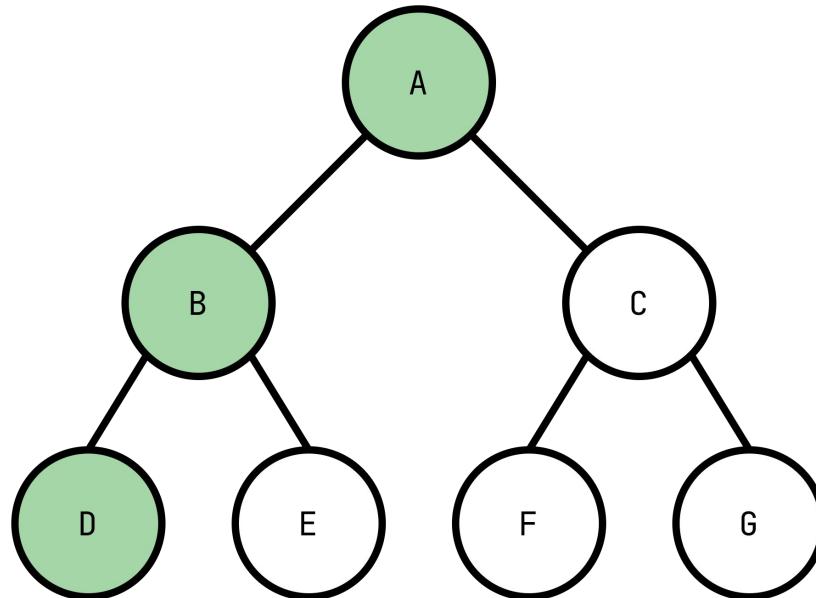
EXAMPLE:

FIND A PATH FROM A TO F

VISIT NODE D

VISITED: [A, B, D]

STACK: [E, C]



DEPTH FIRST SEARCH

EXAMPLE:

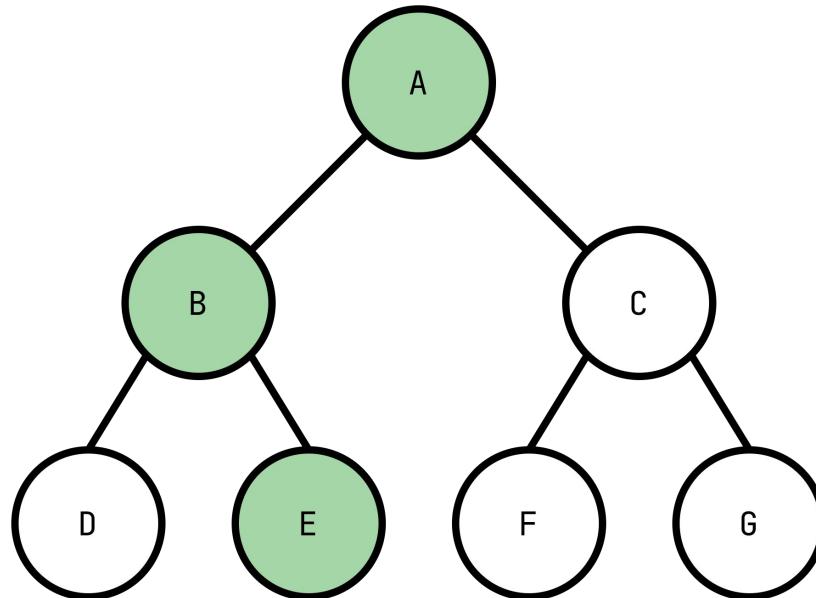
FIND A PATH FROM A TO F

BACKTRACKING

AND VISIT NODE E

VISITED: [A, B, D, E]

STACK: [C]



DEPTH FIRST SEARCH

EXAMPLE:

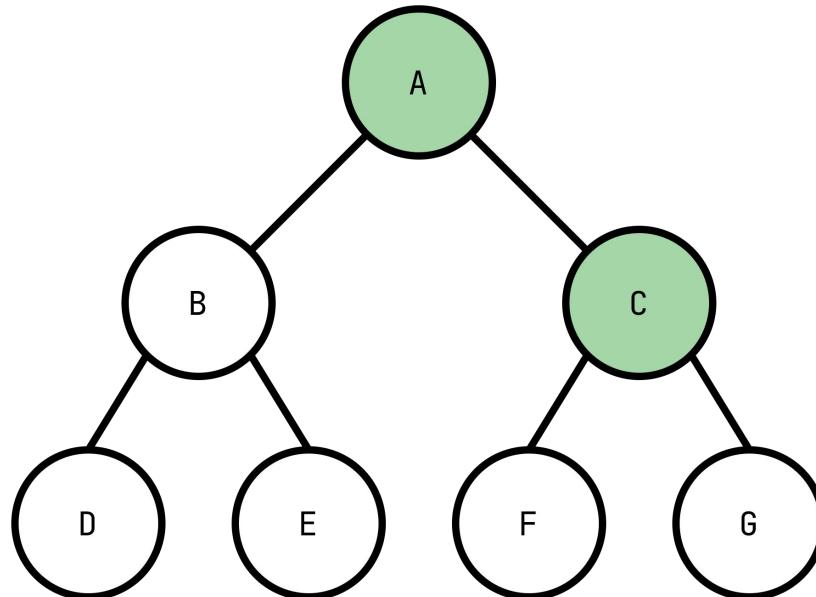
FIND A PATH FROM A TO F

BACKTRACKING

AND VISIT NODE C

VISITED: [A, B, D, E, C]

STACK: [F, G]



DEPTH FIRST SEARCH

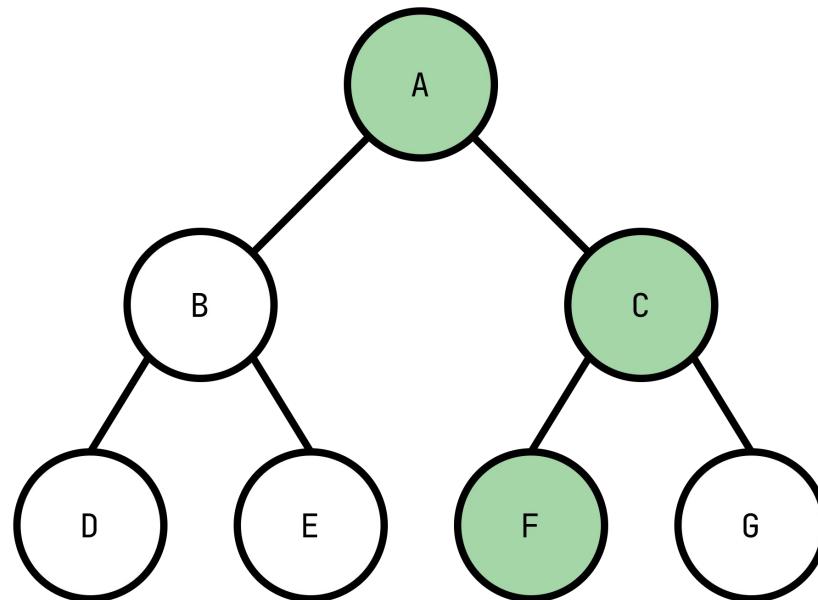
EXAMPLE:

FIND A PATH FROM A TO F

VISIT NODE F

VISITED: [A, B, D, E, C, F]

STACK: [G]





BFS

BREADTH FIRST SEARCH

BREADTH FIRST SEARCH

- 👉 USES A QUEUE DATA STRUCTURE.
- 👉 VISITS NODES BREADTH-FIRST (I.E., EXPLORES ALL THE NEIGHBORS OF A NODE BEFORE MOVING ON TO THE NEXT LEVEL).
- 👉 GOOD FOR FINDING THE SHORTEST PATH BETWEEN TWO NODES.
- 👉 MAY USE MORE MEMORY THAN DFS IF THE GRAPH IS LARGE.
- 👉 MAY BE SLOWER THAN DFS FOR TRAVERSING THE ENTIRE TREE OR GRAPH.

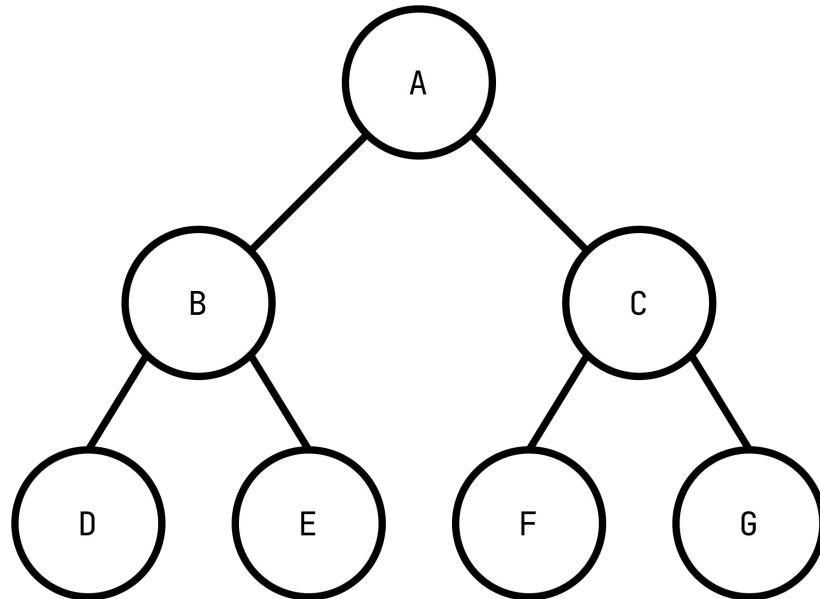
BREADTH FIRST SEARCH

EXAMPLE:

FIND A PATH FROM A TO F

VISITED: []

QUEUE: []



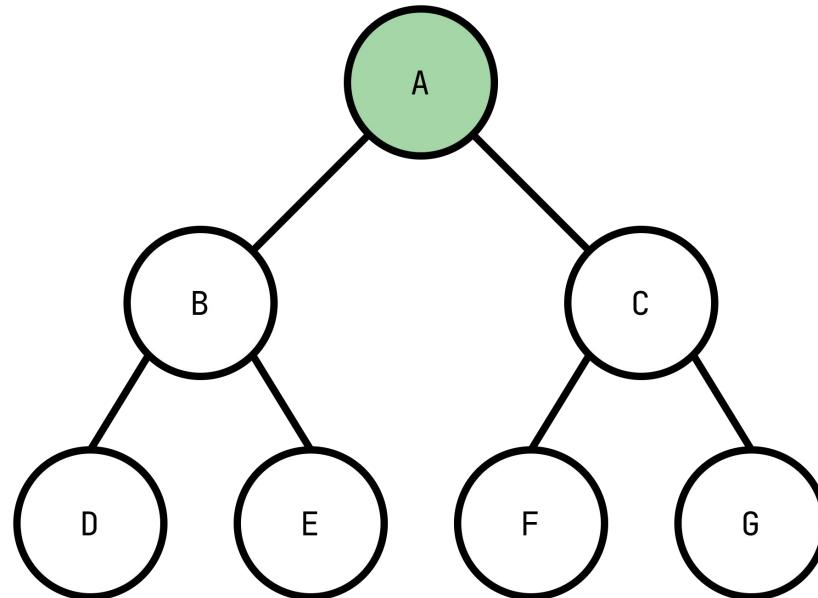
BREADTH FIRST SEARCH

EXAMPLE:

FIND A PATH FROM A TO F

VISITED: [A]

QUEUE: [B, C]



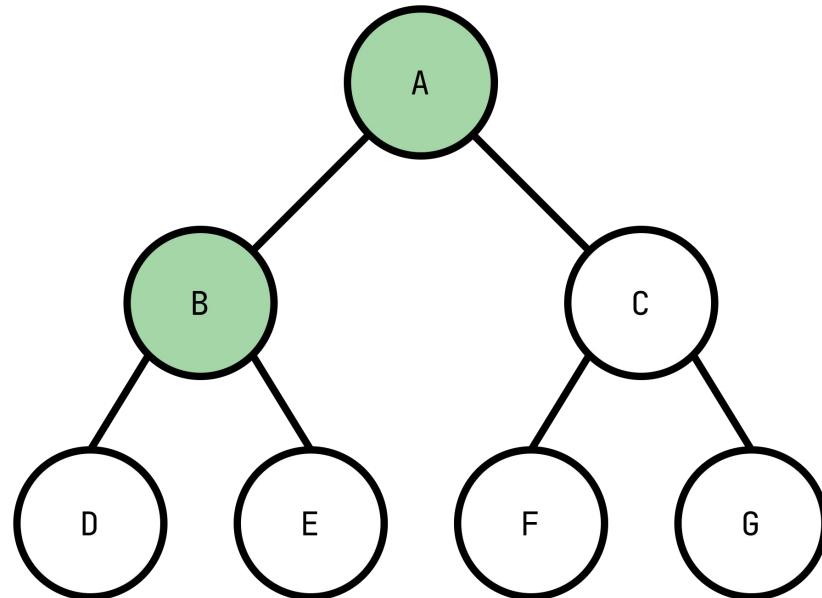
BREADTH FIRST SEARCH

EXAMPLE:

FIND A PATH FROM A TO F

VISITED: [A, B]

QUEUE: [C, D, E]



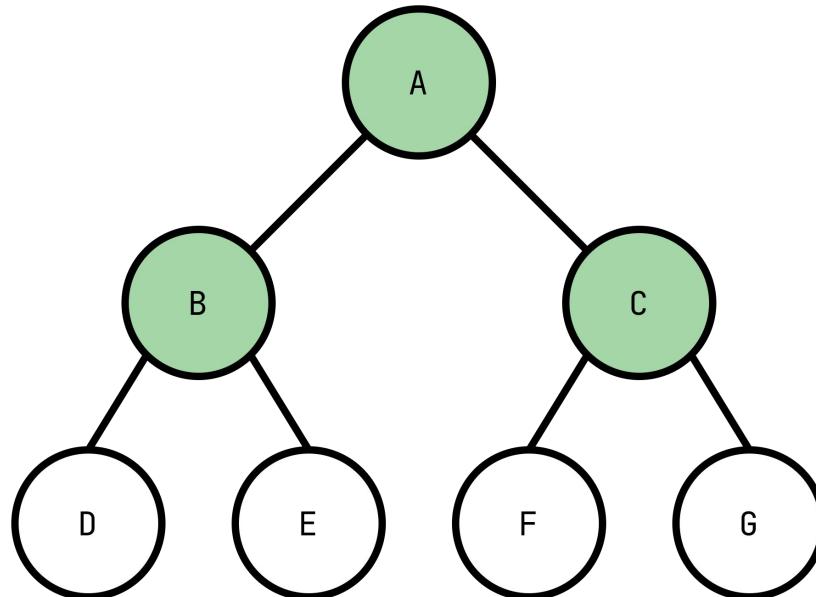
BREADTH FIRST SEARCH

EXAMPLE:

FIND A PATH FROM A TO F

VISITED: [A, B, C]

QUEUE: [D, E, F, G]



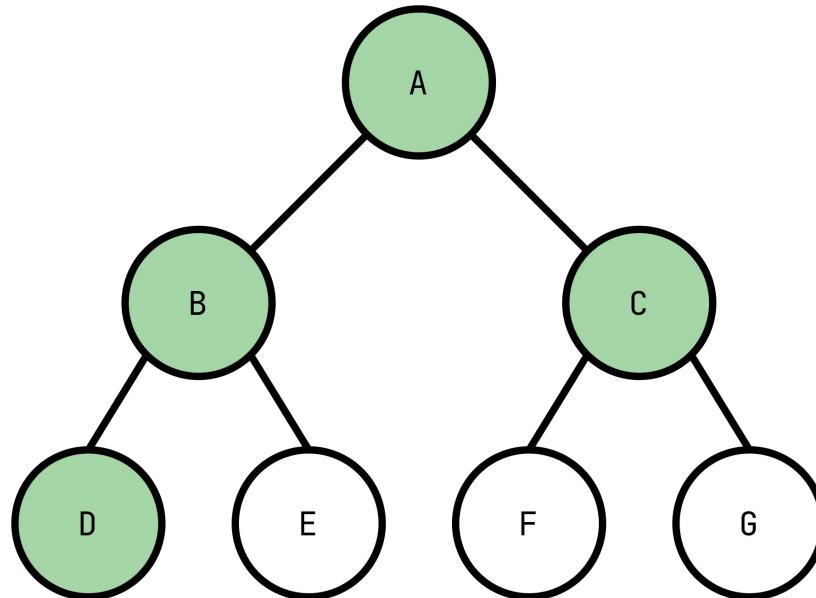
BREADTH FIRST SEARCH

EXAMPLE:

FIND A PATH FROM A TO F

VISITED: [A, B, C, D]

QUEUE: [E, F, G]



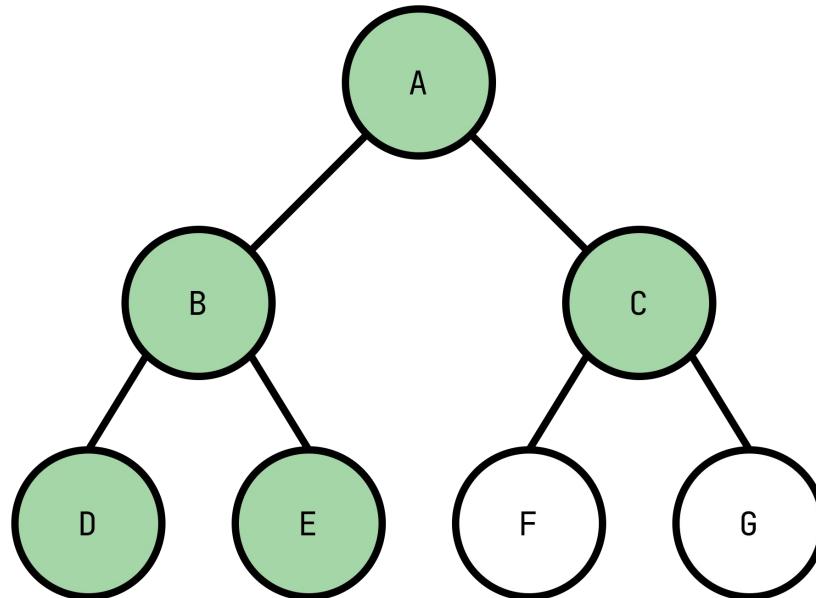
BREADTH FIRST SEARCH

EXAMPLE:

FIND A PATH FROM A TO F

VISITED: [A, B, C, D, E]

QUEUE: [F, G]



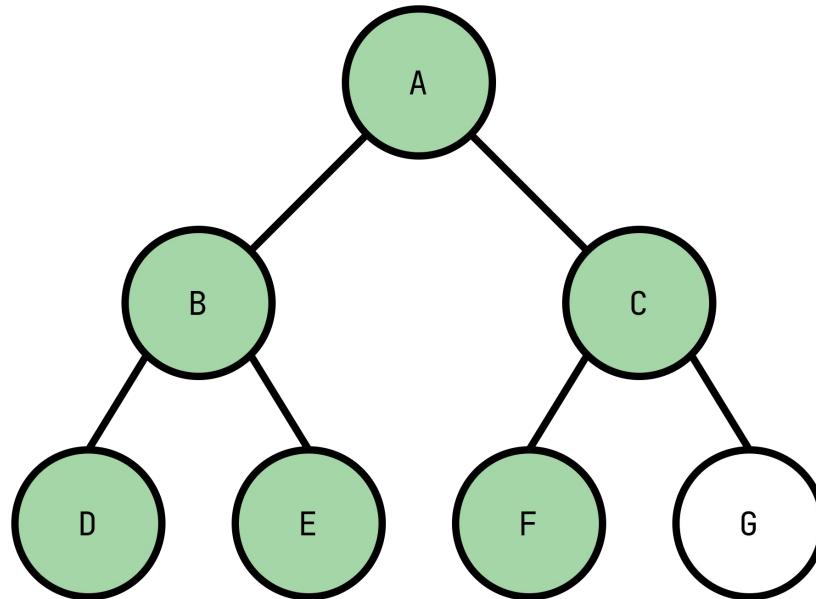
BREADTH FIRST SEARCH

EXAMPLE:

FIND A PATH FROM A TO F

VISITED: [A, B, C, D, E, F]

QUEUE: [G]





SEARCH ALGORITHMS

COMPLEXITY AND OPTIMALITY



WORST CASE COMPLEXITIES

OF BOTH ALGORITHMS ARE THE SAME.



TIME COMPLEXITY: $O(|V| + |E|)$



SPACE COMPLEXITY: $O(|V|)$

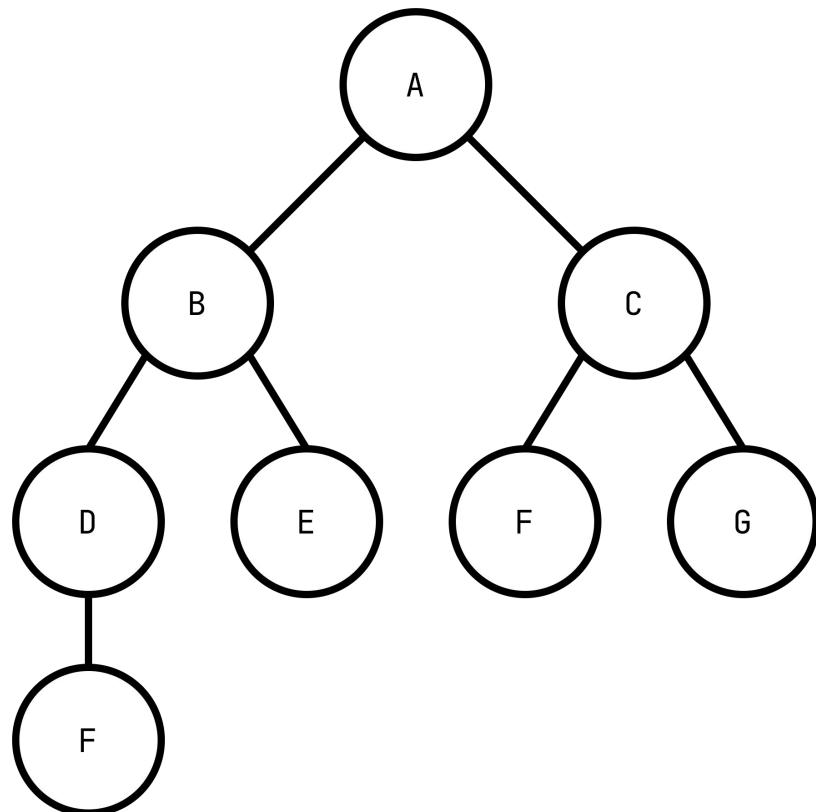


BFS IS OPTIMAL IF THE PATH COST IS A
NON-DECREASING FUNCTION OF D(DEPTH).

NORMALLY, BFS IS APPLIED WHEN ALL THE ACTIONS
HAVE THE SAME COST.

OPTIMALITY COMPARISON

- 👉 ASSUME THERE ARE TWO PATHS TO REACH F
- 👉 ASSUME EACH STEP HAVE THE SAME COST (FOR SIMPLICITY 1)



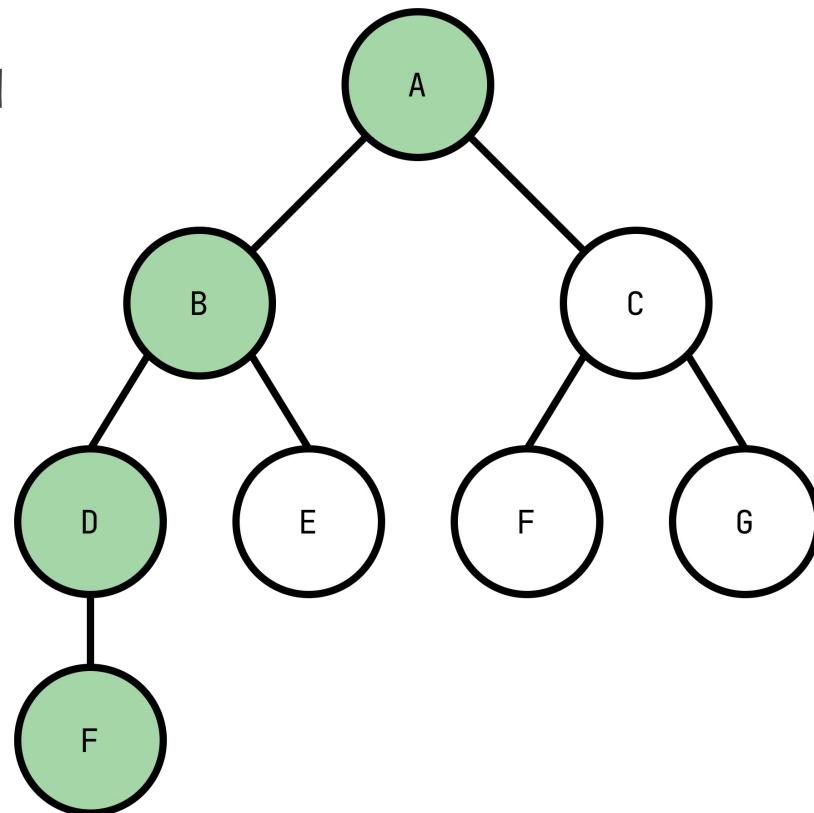
OPTIMALITY COMPARISON

DEPTH FIRST SEARCH

PATH FOUND:

[A → B → D → F]

PATH COST: 4



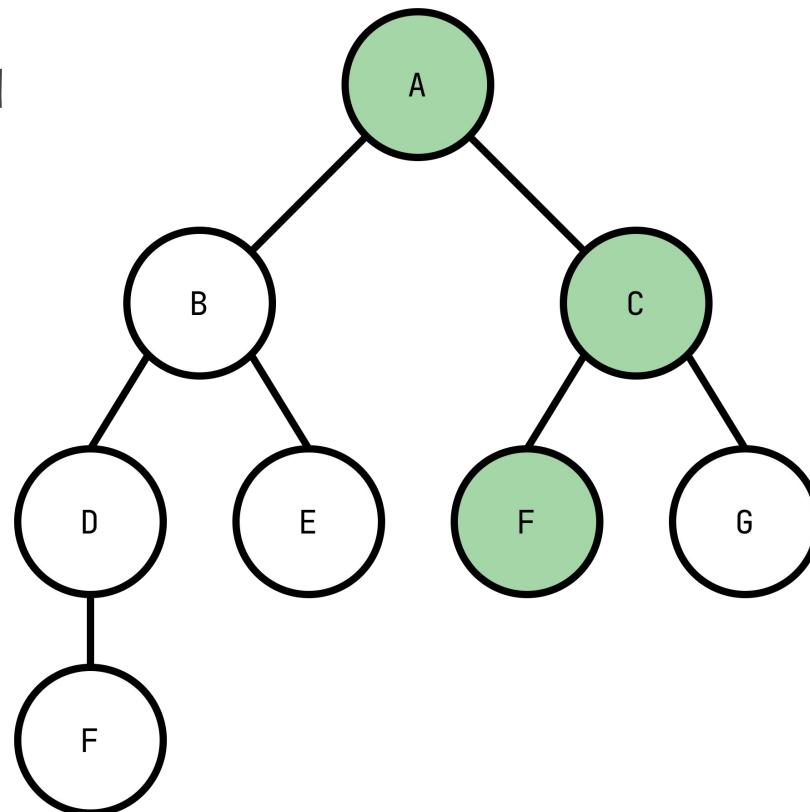
OPTIMALITY COMPARISON

BREADTH FIRST SEARCH

PATH FOUND:

[A → C → F]

PATH COST: 3





LEETCODE

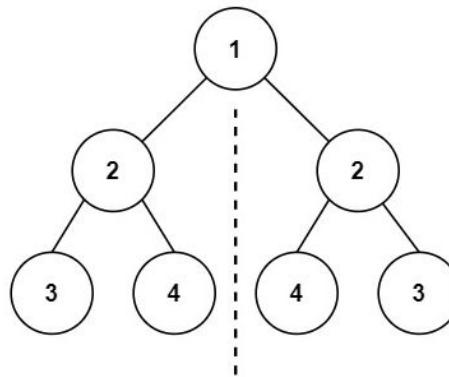


LEETCODE PROBLEM 1

101 SYMMETRIC TREE

<https://leetcode.com/problems/symmetric-tree/>

GIVEN THE ROOT OF A BINARY TREE, CHECK WHETHER IT IS A MIRROR OF ITSELF (I.E., SYMMETRIC AROUND ITS CENTER).

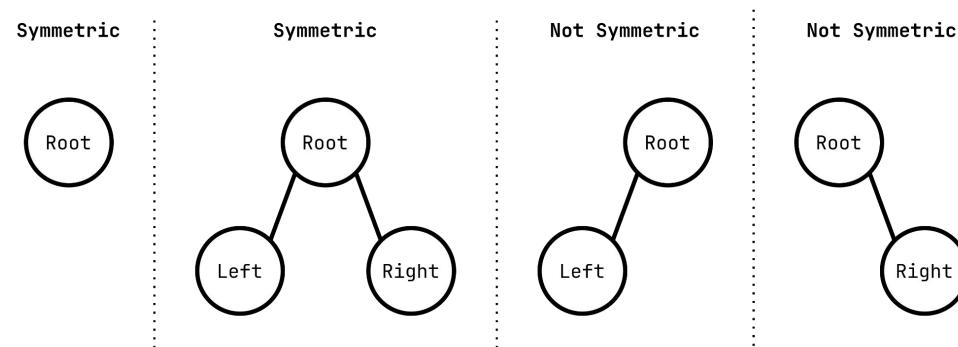




LEETCODE SOLUTION 1

101 SYMMETRIC TREE

[HTTPS://LEETCODE.COM/PROBLEMS/SYMMETRIC-TREE/](https://leetcode.com/problems/symmetric-tree/)

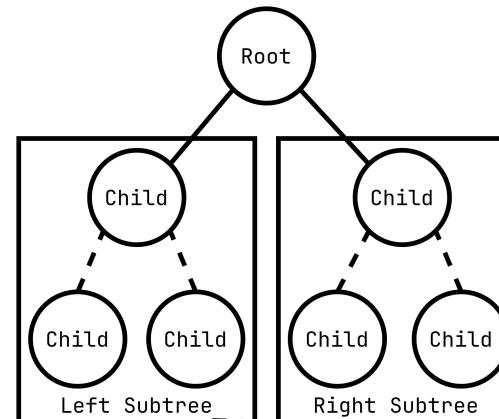




LEETCODE SOLUTION 1

101 SYMMETRIC TREE

[HTTPS://LEETCODE.COM/PROBLEMS/SYMMETRIC-TREE/](https://leetcode.com/problems/symmetric-tree/)



Are Left and Right Subtrees Symmetric?



LEETCODE PROBLEM 2

98 VALIDATE BINARY TREE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

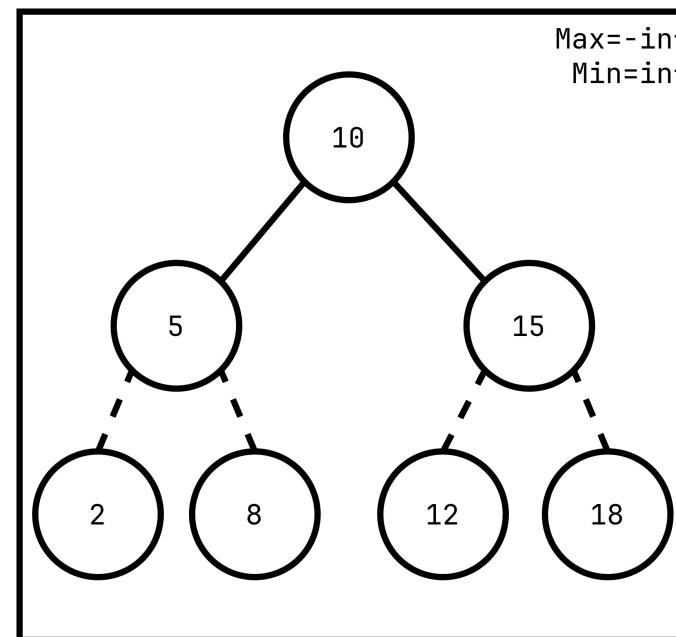
GIVEN THE ROOT OF A BINARY TREE, DETERMINE IF IT IS A VALID BINARY SEARCH TREE (BST). A VALID BST IS DEFINED AS FOLLOWS:

- 👉 THE LEFT SUBTREE OF A NODE CONTAINS ONLY NODES WITH KEYS LESS THAN THE NODE'S KEY.
- 👉 THE RIGHT SUBTREE OF A NODE CONTAINS ONLY NODES WITH KEYS GREATER THAN THE NODE'S KEY.
- 👉 BOTH THE LEFT AND RIGHT SUBTREES MUST ALSO BE BINARY SEARCH TREES.

LEETCODE SOLUTION 2

98 VALIDATE BINARY TREE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

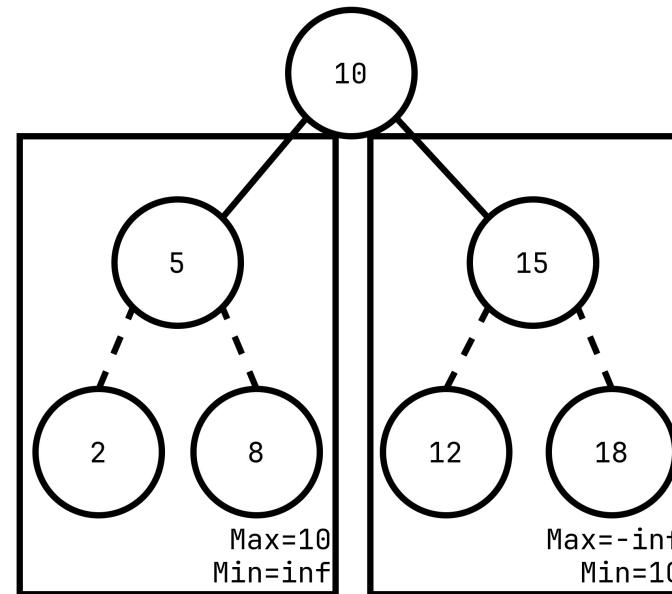




LEETCODE SOLUTION 2

98 VALIDATE BINARY TREE

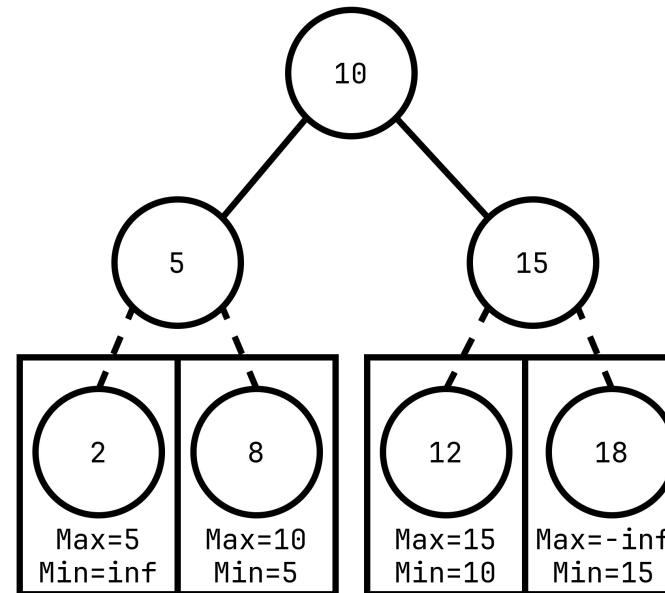
[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)



LEETCODE SOLUTION 2

98 VALIDATE BINARY TREE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)





LEETCODE PROBLEM 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

THERE ARE A TOTAL OF `NUMCOURSES` COURSES YOU HAVE TO TAKE, LABELED FROM 0 TO `NUMCOURSES - 1`. YOU ARE GIVEN AN ARRAY `PREREQUISITES` WHERE `PREREQUISITES[i] = [AI, BI]` INDICATES THAT YOU MUST TAKE COURSE `BI` FIRST IF YOU WANT TO TAKE COURSE `AI`.

FOR EXAMPLE, THE PAIR `[0, 1]`, INDICATES THAT TO TAKE COURSE 0 YOU HAVE TO FIRST TAKE COURSE 1.

RETURN TRUE IF YOU CAN FINISH ALL COURSES. OTHERWISE, RETURN FALSE.



LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

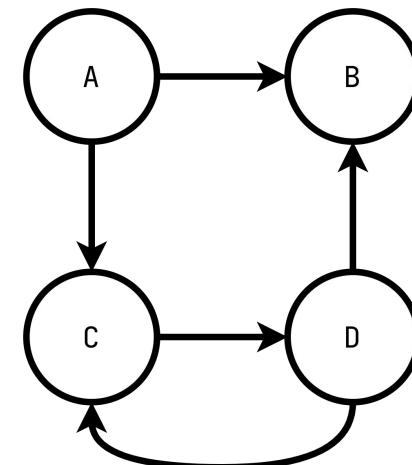
DEPTH FIRST SEARCH

WITH VISITED AND STACK

VISIT ALL NODES BY DEPTH

A → B: FINISH A TO DO B

VISIT - CHECK - ADD -
ADJACENT - REMOVE





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)
DEPTH FIRST SEARCH

MARK THE CURRENT NODE AS VISITED AND ALSO MARK THE INDEX IN THE RECURSION STACK.

ITERATE A LOOP FOR ALL THE VERTICES AND FOR EACH VERTEX, CALL THE RECURSIVE FUNCTION IF IT IS NOT YET VISITED.

(THIS STEP IS DONE TO MAKE SURE THAT IF THERE IS A FOREST OF GRAPHS, WE ARE CHECKING EACH FOREST)



LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)
DEPTH FIRST SEARCH

IN EACH RECURSION CALL, FIND ALL THE ADJACENT VERTICES OF THE CURRENT VERTEX WHICH ARE NOT VISITED:

- 👉 IF AN ADJACENT VERTEX IS ALREADY MARKED IN THE RECURSION STACK THEN RETURN TRUE.
- 👉 OTHERWISE, CALL THE RECURSIVE FUNCTION FOR THAT ADJACENT VERTEX.



LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

WHILE RETURNING FROM THE RECURSION CALL, UNMARK THE CURRENT NODE FROM THE RECURSION STACK, TO REPRESENT THAT THE CURRENT NODE IS NO LONGER A PART OF THE PATH BEING TRACED.

IF ANY OF THE FUNCTIONS RETURNS TRUE, STOP THE FUTURE FUNCTION CALLS AND RETURN TRUE AS THE ANSWER.



LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

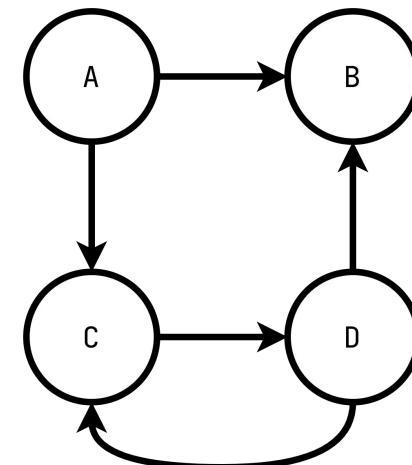
VISIT A

CURRENT: [A]

VISITED: []

STACK: []

ADJLIST: [[], [A, D], [A, D], [C]]



LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

CHECK A →

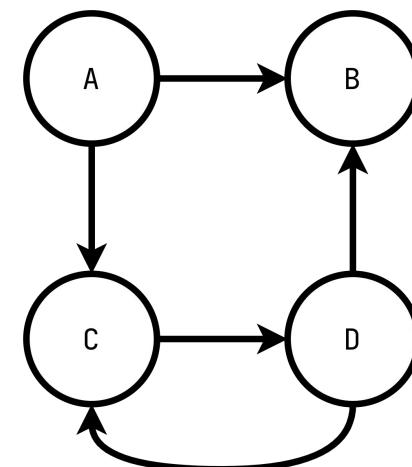
NOT VISITED AND NOT STACK

CURRENT: [A]

VISITED: []

STACK: []

ADJLIST: [[], [A, D], [A, D], [C]]



LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

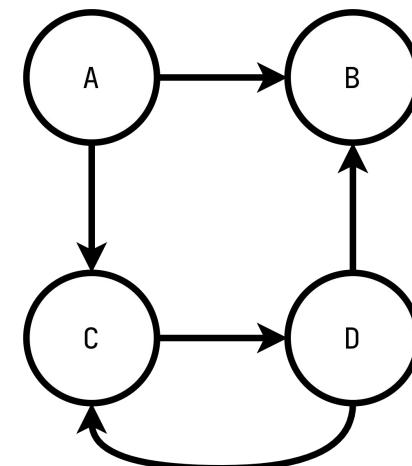
ADD A TO VISITED AND STACK

CURRENT: [A]

VISITED: [A]

STACK: [A]

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

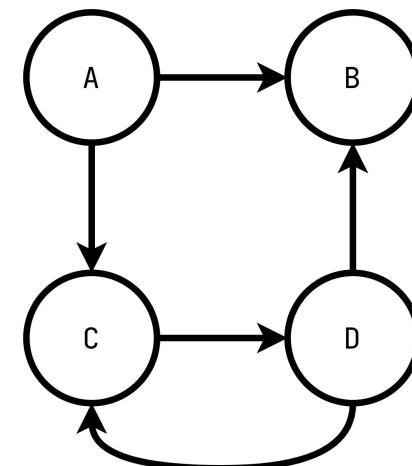
VISIT ADJLIST OF A ([])

CURRENT: [A]

VISITED: [A]

STACK: [A]

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

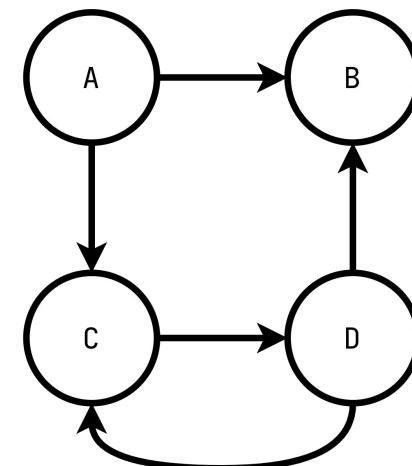
REMOVE A FROM STACK

CURRENT: [A]

VISITED: [A]

STACK: []

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

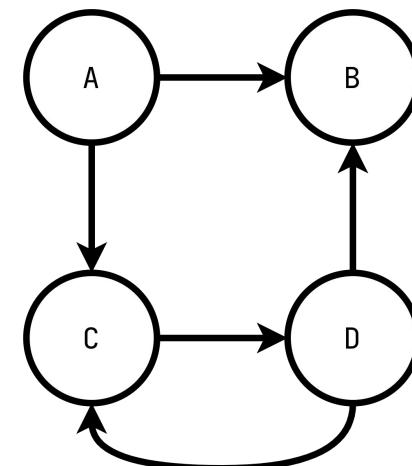
VISIT B

CURRENT: [B]

VISITED: [A]

STACK: []

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

CHECK B →

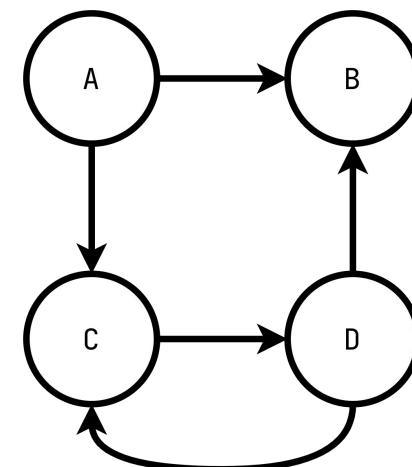
NOT VISITED AND NOT STACK

CURRENT: [B]

VISITED: [A]

STACK: []

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

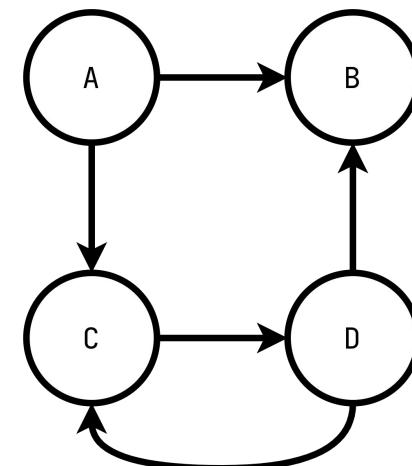
ADD B TO VISITED AND STACK

CURRENT: [B]

VISITED: [A, B]

STACK: [B]

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

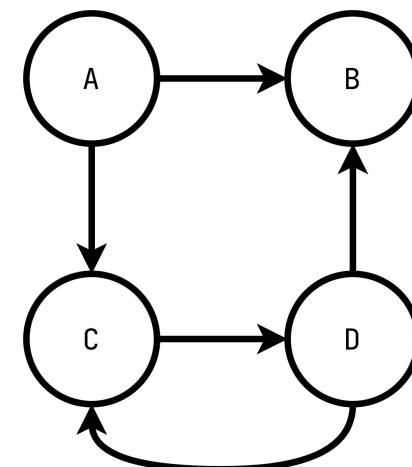
VISIT ADJLIST OF B ([A, D])

CURRENT: [B]

VISITED: [A, B]

STACK: [B]

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

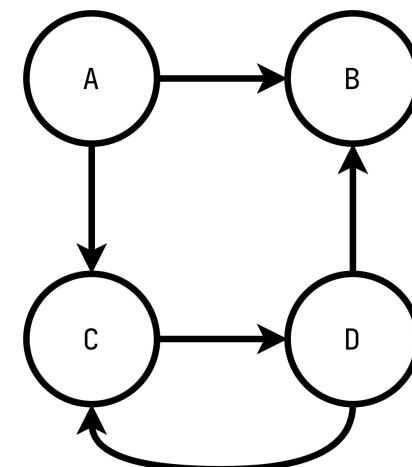
VISIT A

CURRENT: [A]

VISITED: [A, B]

STACK: [B]

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

CHECK A →

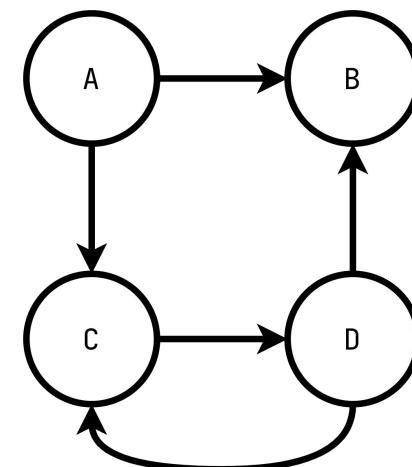
VISITED AND NOT STACK → FALSE

CURRENT: [A]

VISITED: [A, B]

STACK: [B]

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

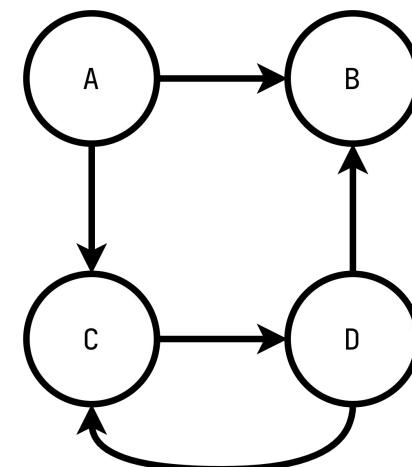
VISIT D

CURRENT: [D]

VISITED: [A, B]

STACK: [B]

ADJLIST: [[], [A, D], [A, D], [C]]



LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

CHECK D →

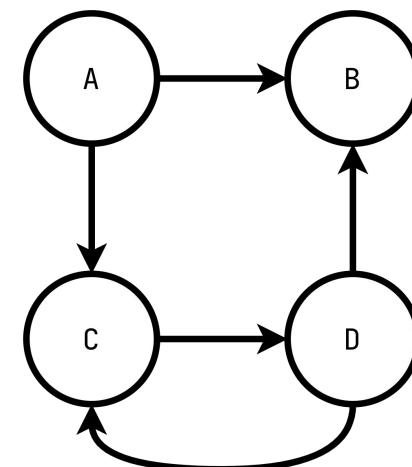
NOT VISITED AND NOT STACK

CURRENT: [D]

VISITED: [A, B]

STACK: [B]

ADJLIST: [[], [A, D], [A, D], [C]]



LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

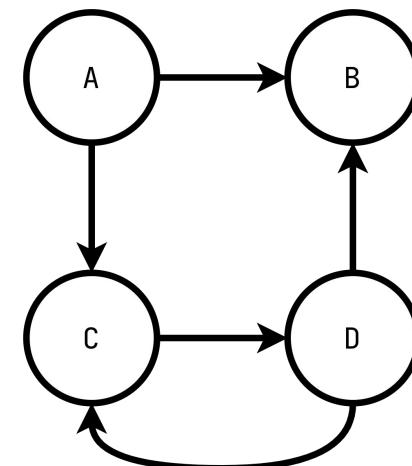
ADD D TO VISITED AND STACK

CURRENT: [D]

VISITED: [A, B, D]

STACK: [B, D]

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

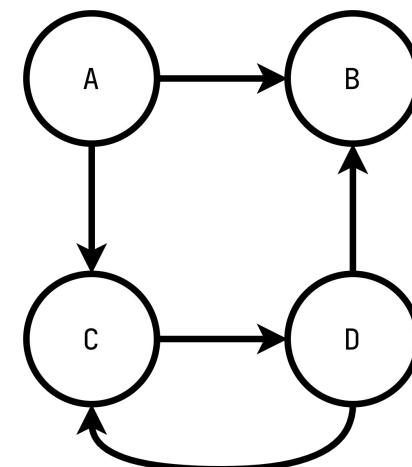
VISIT ADJLIST OF D ([C])

CURRENT: [D]

VISITED: [A, B, D]

STACK: [B, D]

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

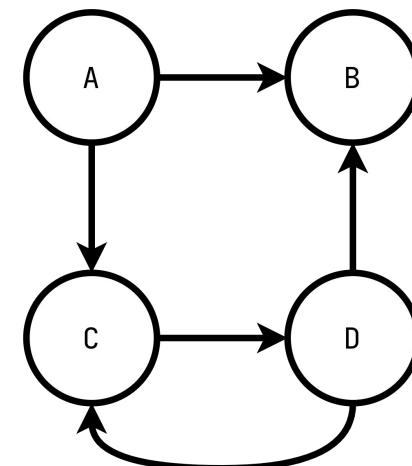
VISIT C

CURRENT: [C]

VISITED: [A, B, D]

STACK: [B, D]

ADJLIST: [[], [A, D], [A, D], [C]]



LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

CHECK C →

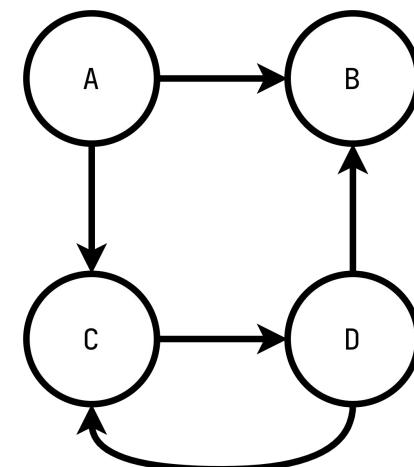
NOT VISITED AND NOT STACK

CURRENT: [C]

VISITED: [A, B, D]

STACK: [B, D]

ADJLIST: [[], [A, D], [A, D], [C]]



LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

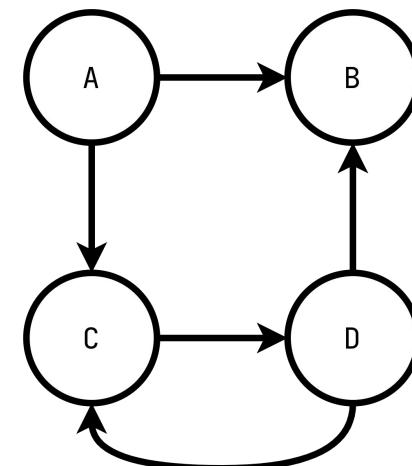
ADD C TO VISITED AND STACK

CURRENT: [C]

VISITED: [A, B, D, C]

STACK: [B, D, C]

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

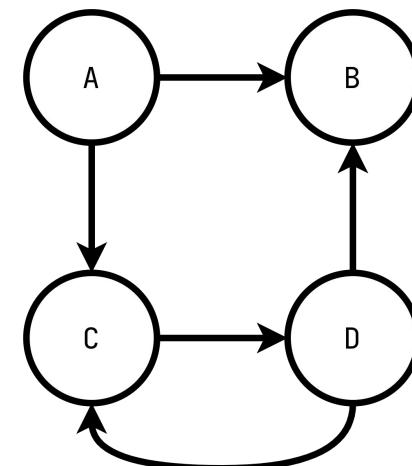
VISIT ADJLIST OF C ([A, D])

CURRENT: [C]

VISITED: [A, B, D, C]

STACK: [B, D, C]

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

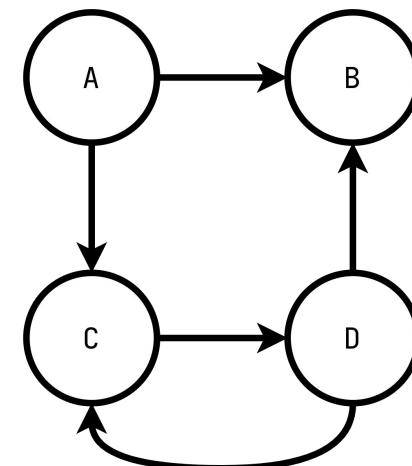
VISIT A

CURRENT: [A]

VISITED: [A, B, D, C]

STACK: [B, D, C]

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

CHECK A →

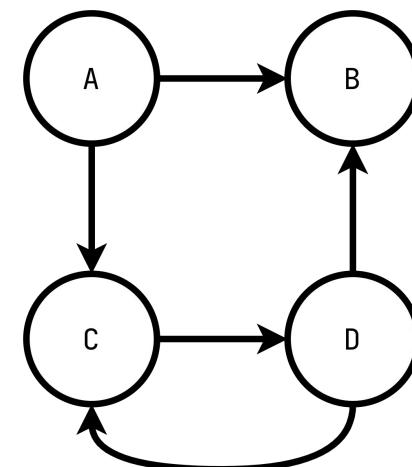
VISITED AND NOT STACK → FALSE

CURRENT: [A]

VISITED: [A, B, D, C]

STACK: [B, D, C]

ADJLIST: [[], [A, D], [A, D], [C]]





LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

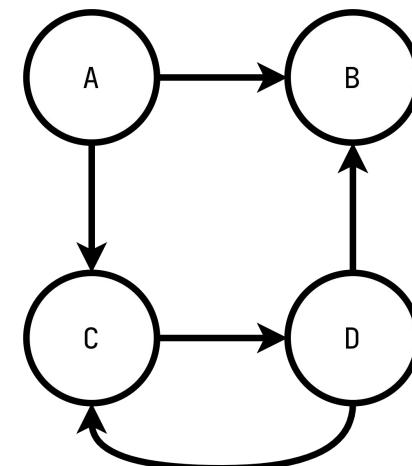
VISIT D

CURRENT: [D]

VISITED: [A, B, D, C]

STACK: [B, D, C]

ADJLIST: [[], [A, D], [A, D], [C]]



LEETCODE SOLUTION 3

207 COURSE SCHEDULE

[HTTPS://LEETCODE.COM/PROBLEMS/VALIDATE-BINARY-SEARCH-TREE/](https://leetcode.com/problems/validate-binary-search-tree/)

DEPTH FIRST SEARCH

CHECK D →

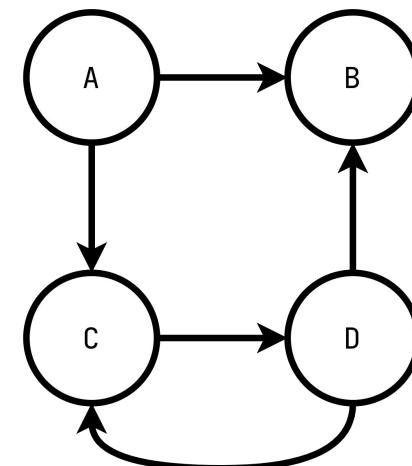
VISITED AND STACK → TRUE

CURRENT: [D]

VISITED: [A, B, D, C]

STACK: [B, D, C]

ADJLIST: [[], [A, D], [A, D], [C]]



■ Feedback form



Submit here.

■ T.HANKS everyone!



T.HANKS A LOT!

