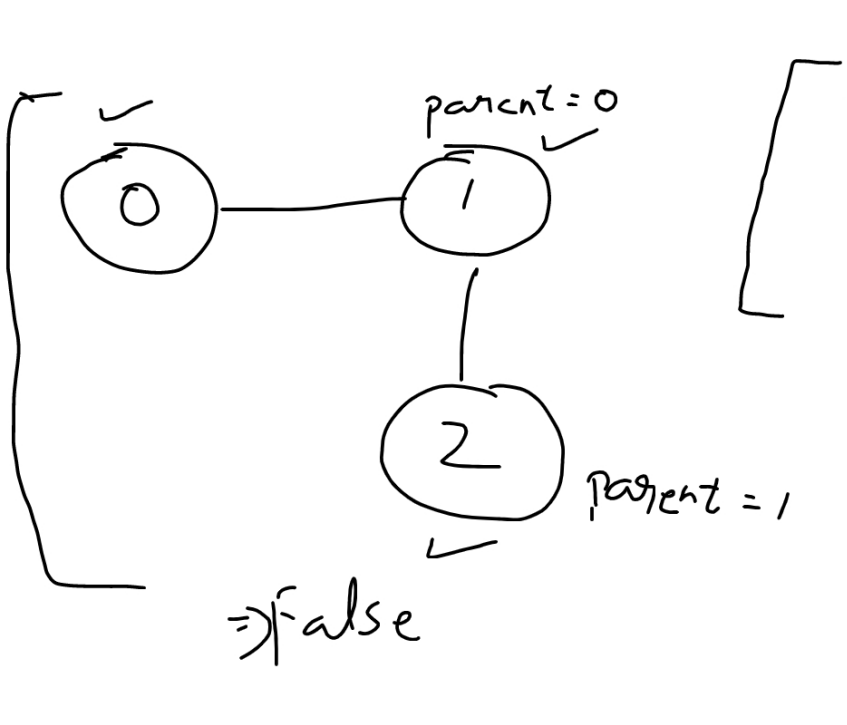
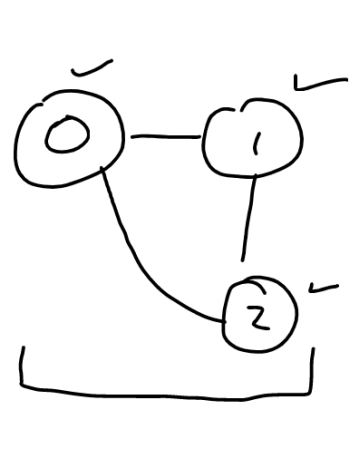
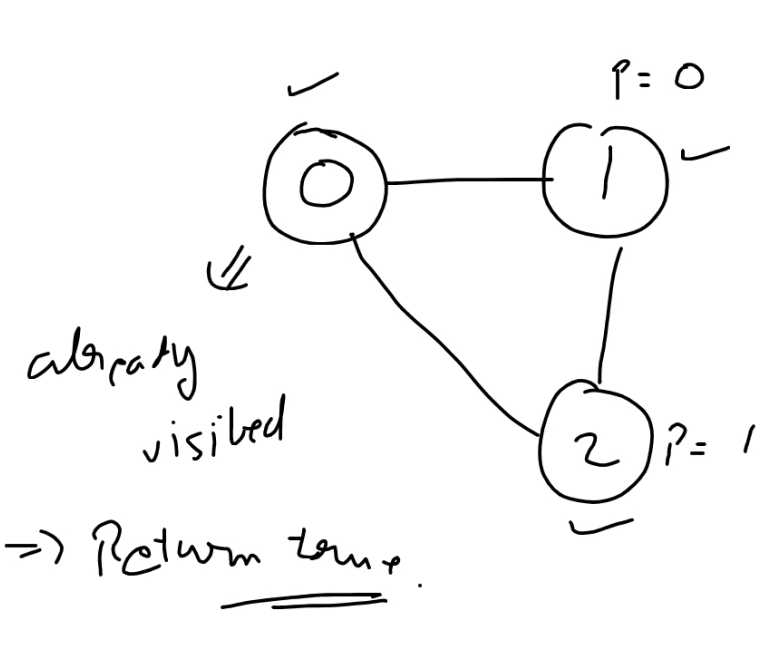

parent = 0
parent = 1

0 → 1
1 → 0,2
2 → 1

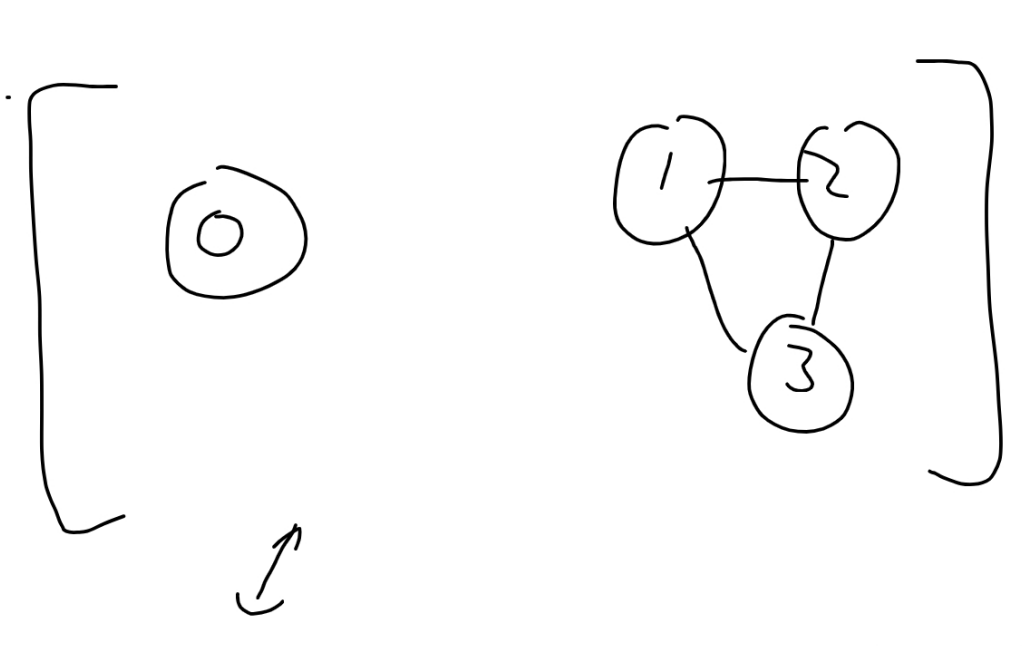
⇒ false




P = 0
P = 1
already visited

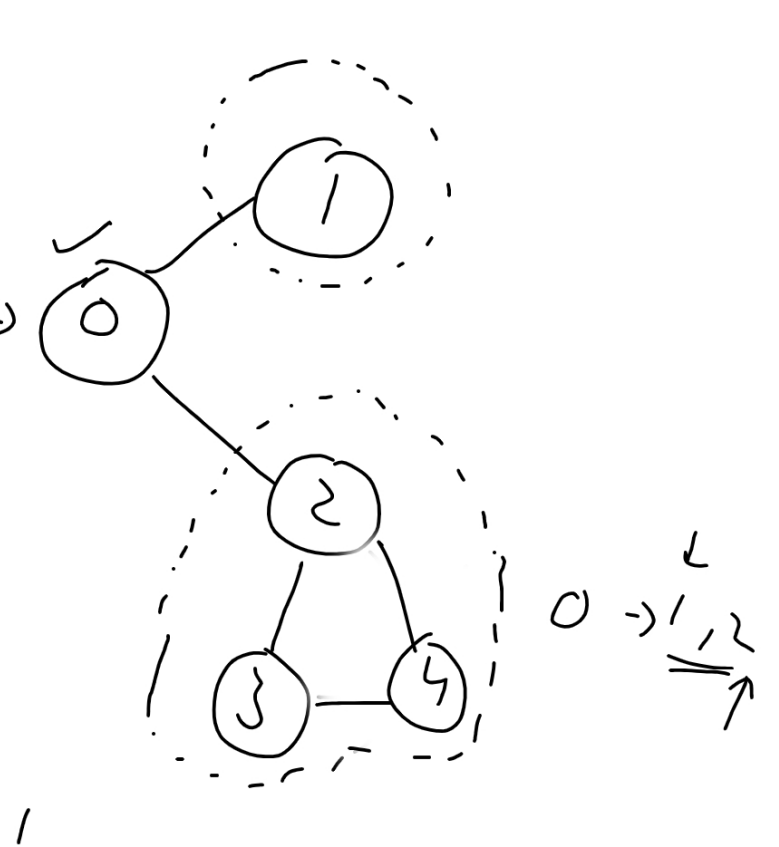
⇒ Return true.

0 → 1,2
1 → 0,2
2 → 0,1



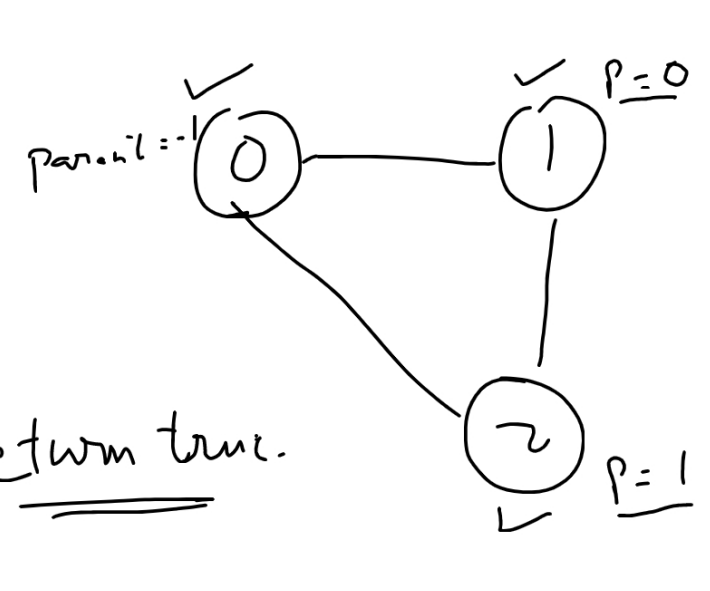
0 →
1 → 2, 3
2 → 1, 3
3 → 1, 2

current = 0



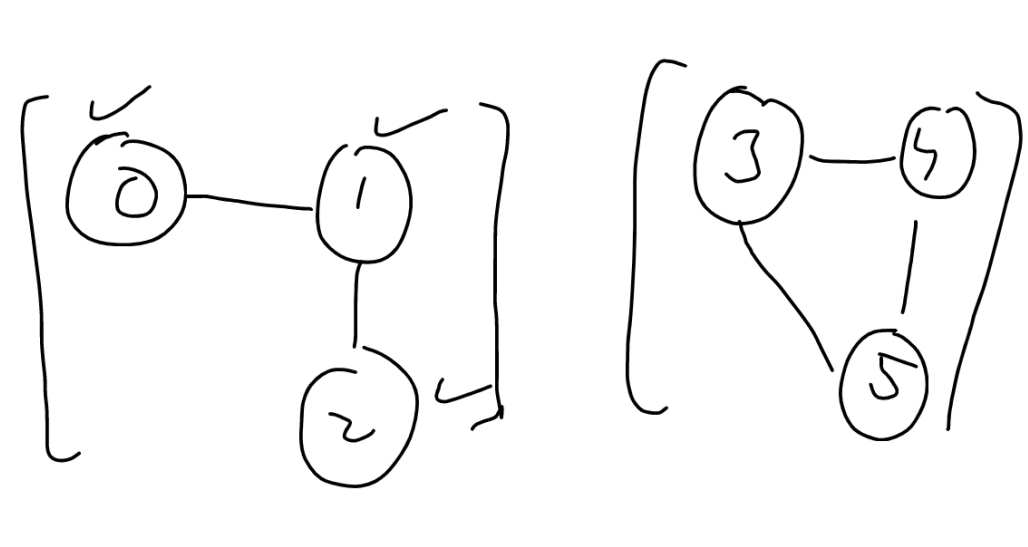
j = 1

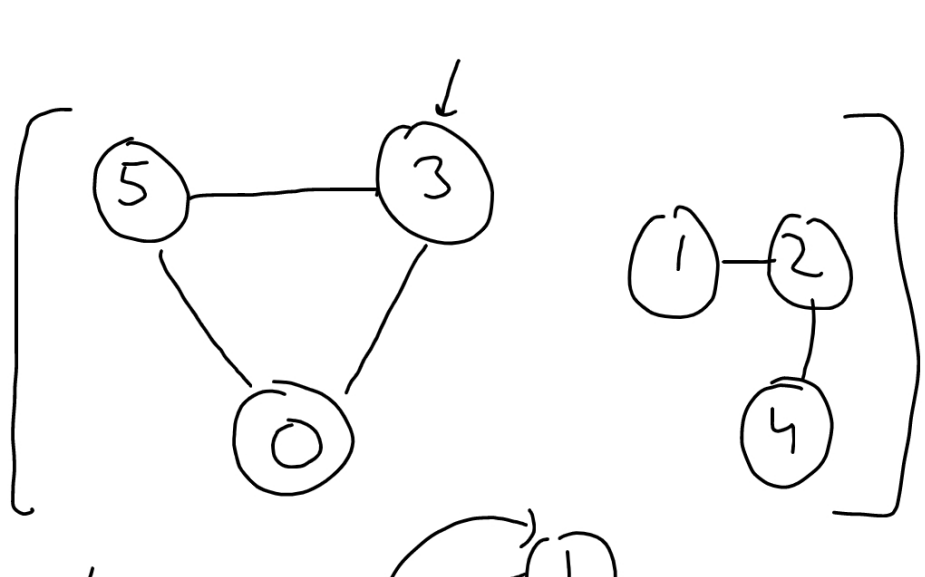
0 → 1,2
1 → 0
2 → 0,3,4
3 → 2,4
4 → 2,3


parent = -1
P = 0
P = 1

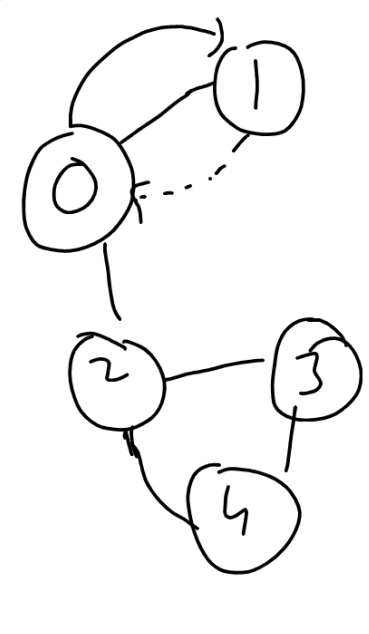
Return true.

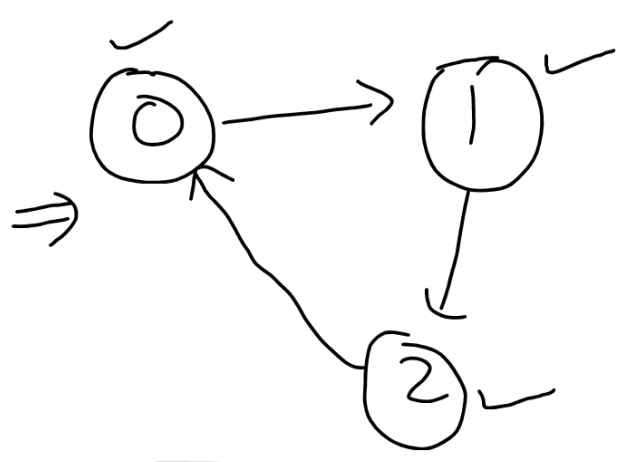
0 → 1,2
1 → 0,2
2 → 0, 1





0 → 1,2

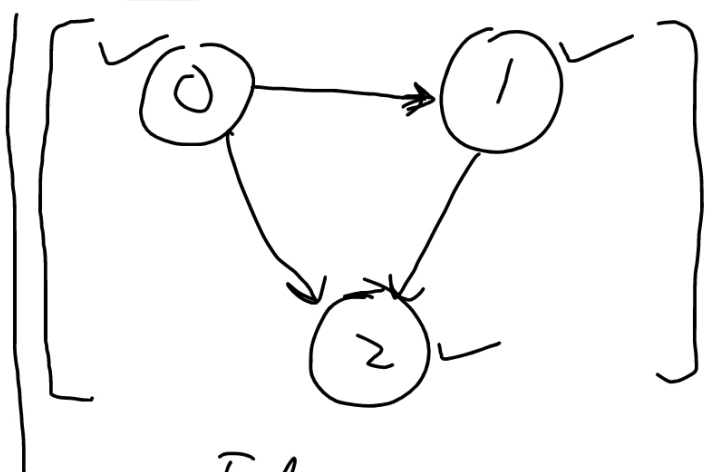




True

0 → 1
1 → 2
2 → 0

2
1
0



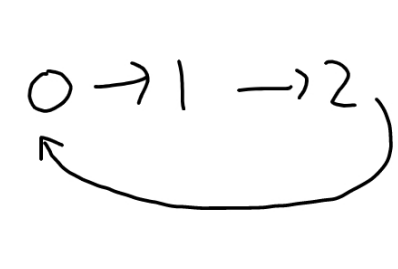
True ~~X~~

False.

0 → 1,2 (2,1)
1 → 2
2 → ~~—~~

~~2~~
~~1~~
0

2
1
0



⇒ 0 → 1 → 2

0
2
1
0

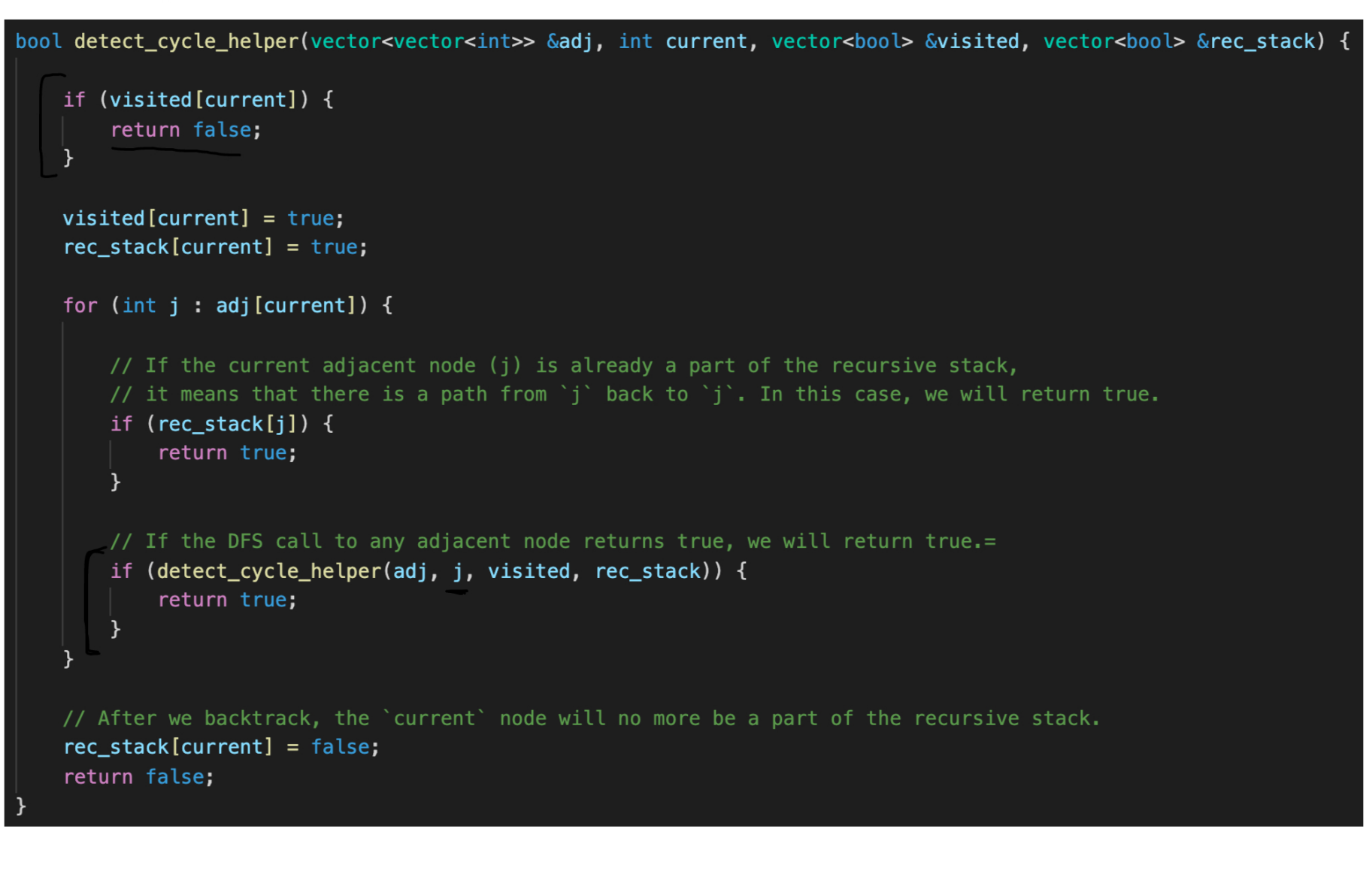
[0 → 1 → 2 → 0]

[True.]

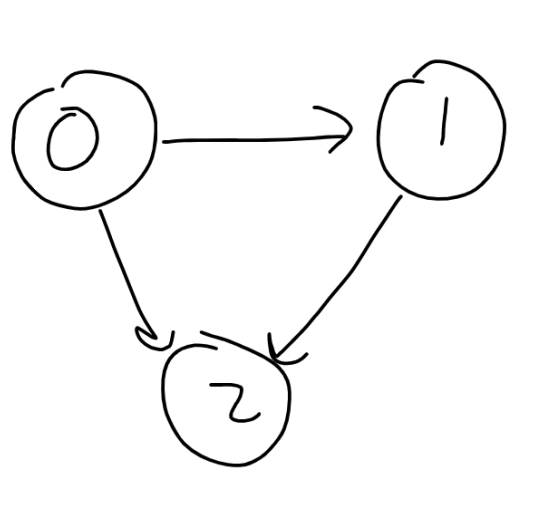
~~2~~
~~1~~
~~2~~
~~0~~

⇒ [0 → 2]
⇒ [0 → 1 → 2]

False.



```
bool detect_cycle_helper(vector<vector<int>> &adj, int current, vector<bool> &visited, vector<bool> &rec_stack) {

    if (visited[current]) {
        return false;
    }

    visited[current] = true;
    rec_stack[current] = true;

    for (int j : adj[current]) {

        // If the current adjacent node (j) is already a part of the recursive stack,
        // it means that there is a path from `j` back to `j`. In this case, we will return true.
        if (rec_stack[j]) {
            return true;
        }

        // If the DFS call to any adjacent node returns true, we will return true.=
        if (detect_cycle_helper(adj, j, visited, rec_stack)) {
            return true;
        }
    }

    // After we backtrack, the `current` node will no more be a part of the recursive stack.
    rec_stack[current] = false;
    return false;
}
```

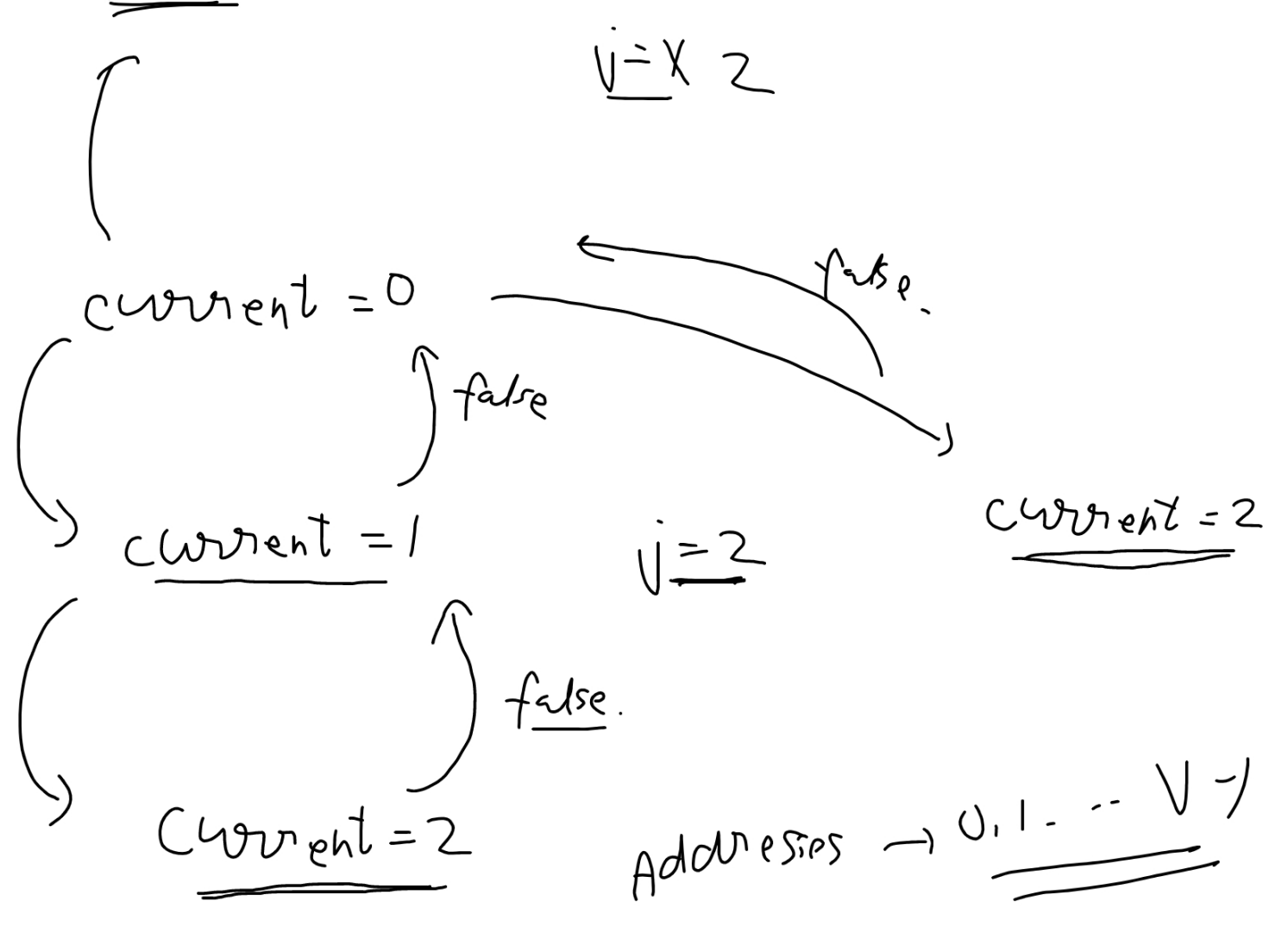


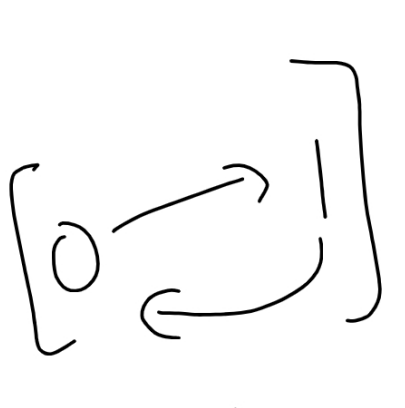
	0	1	2
visited:	~~F~~ T	~~F~~ T	~~F~~ T
rec_stack:	~~F~~ ~~T~~ F	~~F~~ ~~T~~ F	~~F~~ ~~T~~ F

[0 → 1, 2
1 → 2
2 →]

~~2~~
~~1~~
0

j = ~~1~~ 2


false.
current = 0
false
false.
current = 1
j = 2
current = 2
false.
Current = 2



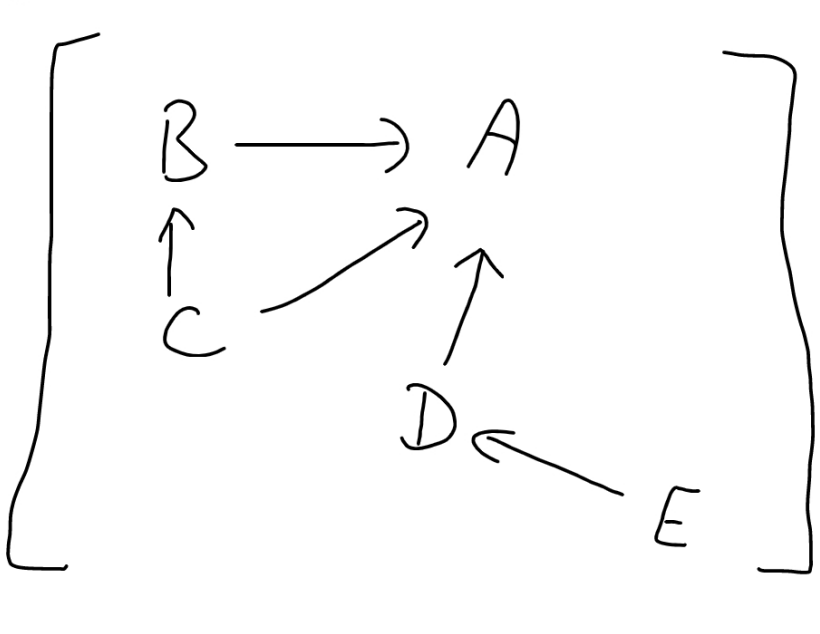
True.

[0 — 1]

Addresses → 0,1 . . . V-1

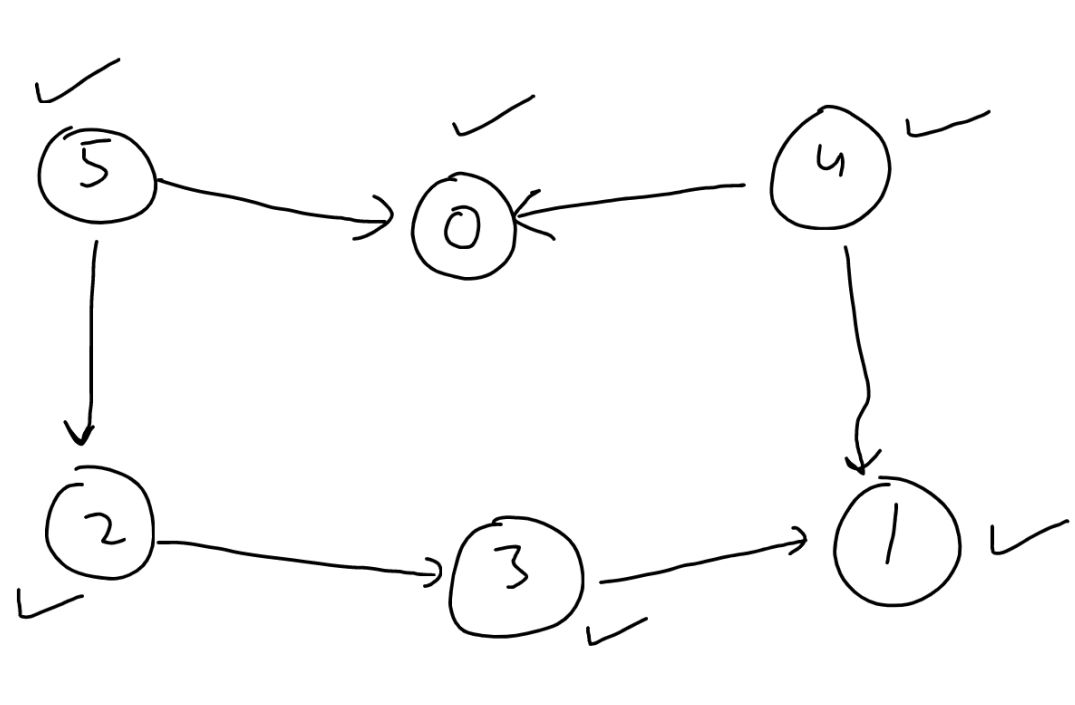
---

courses	Prerequisites
A	B, C, D
B	C
D	E



A →
B → A
C → A, B
D → A
E → D

[E C B D A] ordering of courses



0 →
1 →
2 → 3
⇒ 3 → 1
4 → 0, 1
5 → 0, 2

[[ ]
[ ]
[3]
[1]
[0, 1]
[0, 2]]

helper.

~~5~~
~~4~~
~~2~~
~~3~~
~~1~~
~~0~~

Step-1 topo(0)

Step-2 topo(1)

Step-3 topo(2)
topo(3)

Step-4 topo(4)

Step-5 topo(5)

[5, 4, 2, 3, 1, 0] ✓

topological sort

U → V

U
V

[U, V]

Node {
string ___ ,
}

0 → A
1 → B
2 → C
3 → D
4 → E
5 → F

0, 1, a, 10a

[unordered_map <string, vector <string>>
adj

A → [B, C, D]
B → [C]
D → [E]

unordered_map <string, bool>
visited

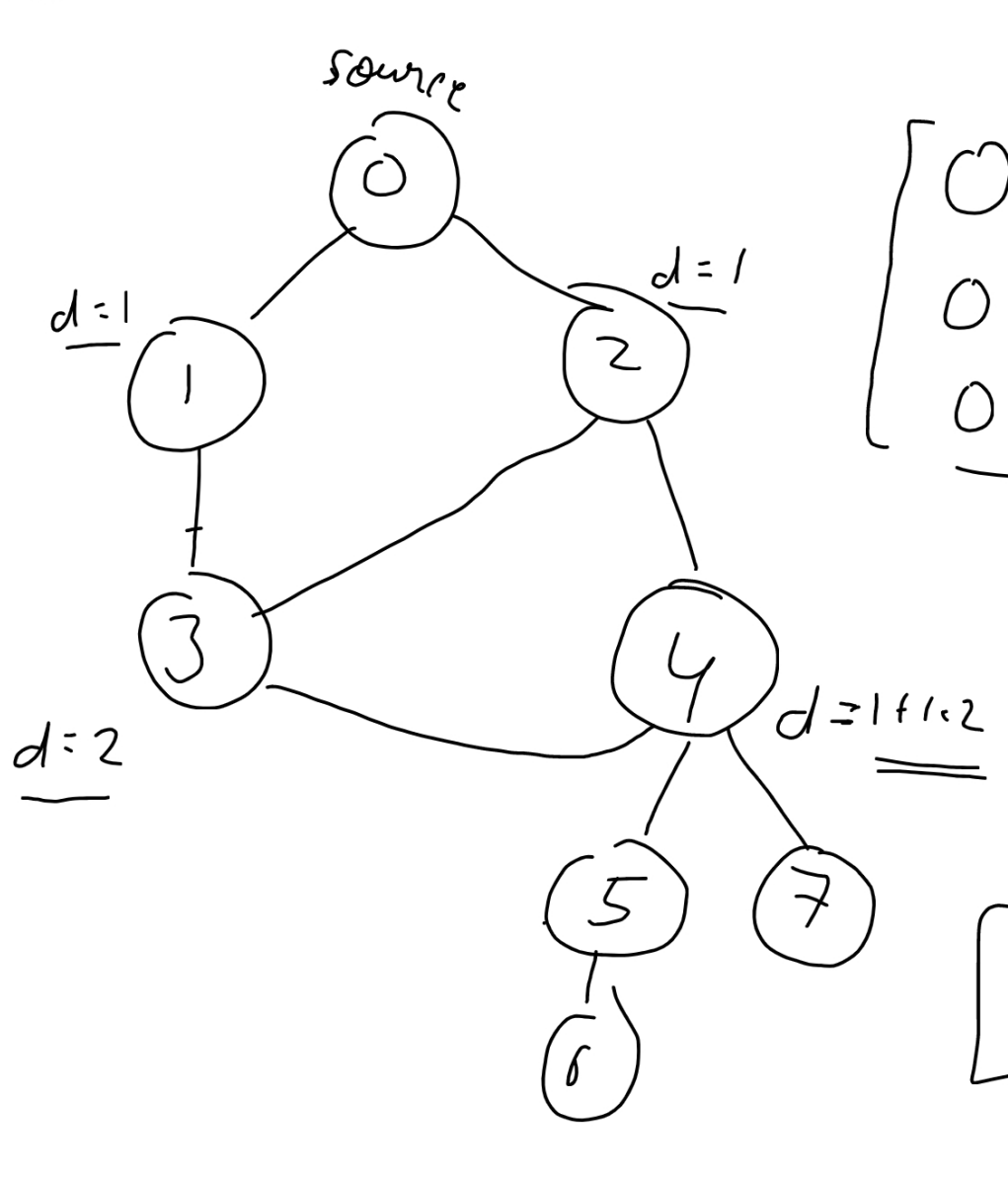
[unorder_map <int, vector <int>>]
unorder_map <int, bool> vis;

0 → 1 → 2
3

[current = 0] j = 1
current = 1 j = 2
current = 2

1
2

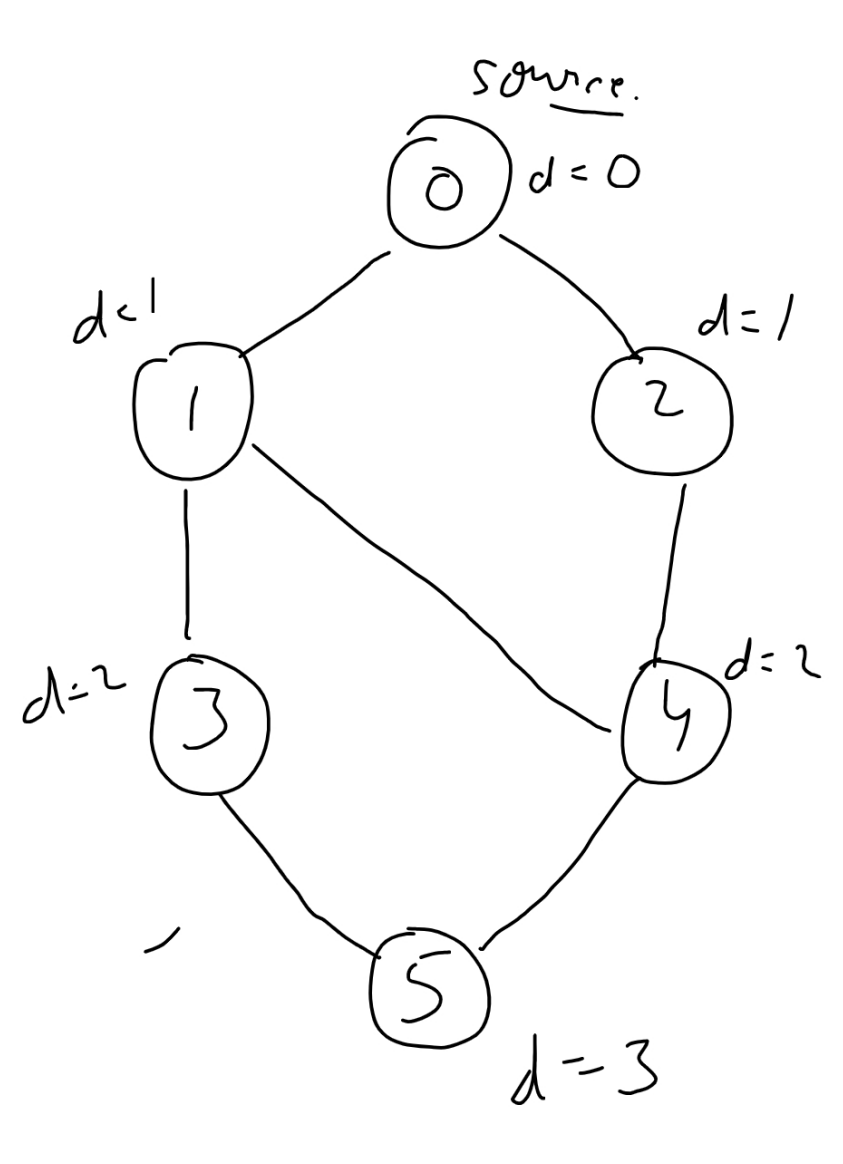
[1, 2]


source
d = 1
d = 1
d = 2
d = 1+1 = 2

[0, 1, 3, ~~3~~ 2, 4
0, 1, 3, 4
0, 2, 4]
dist = 2

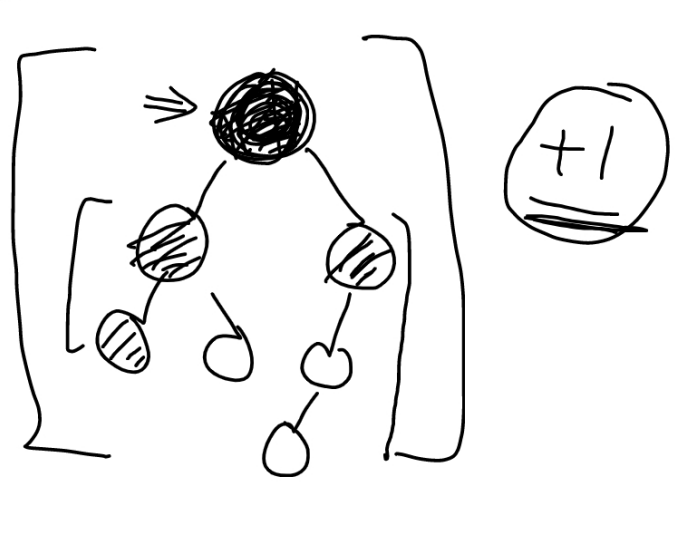
BFS

SSSP
[Single Source Shortest Path]


source.
d = 0
d = 1
d = 1
d = 2
d = 2
d = 3

0 → 1, 2
1 → 0, 3, 4
2 → 0, 4
3 → 1, 5
4 → 1, 2, 5
5 → 3, 4

```
int height (Node * root) {
    if (root == NULL) return 0;
⇒ [int lh = height(root -> left);
   int rh = height (root -> right);]
  ⇒return max (lh, rh) + 1;
}
```



+1