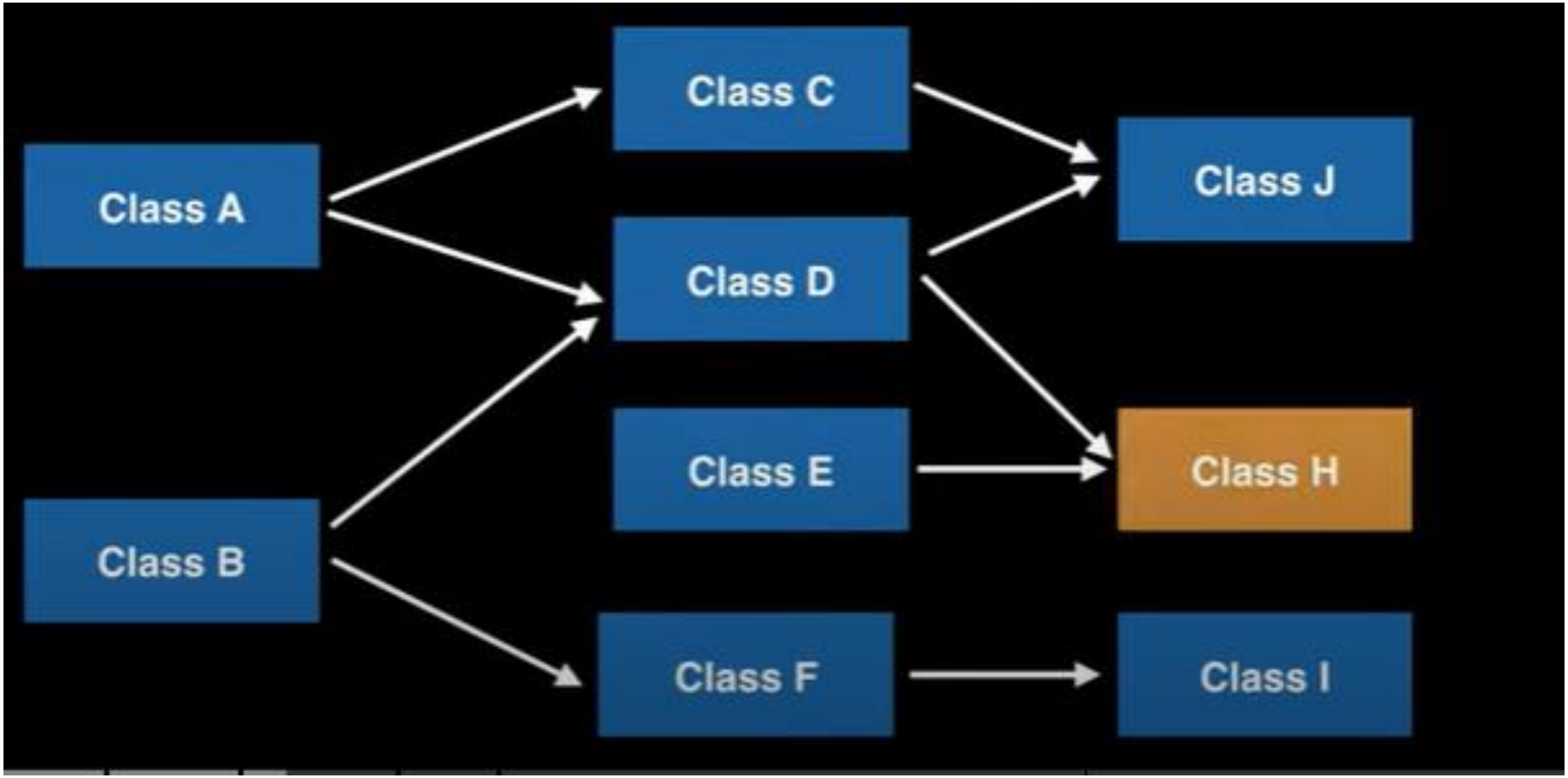
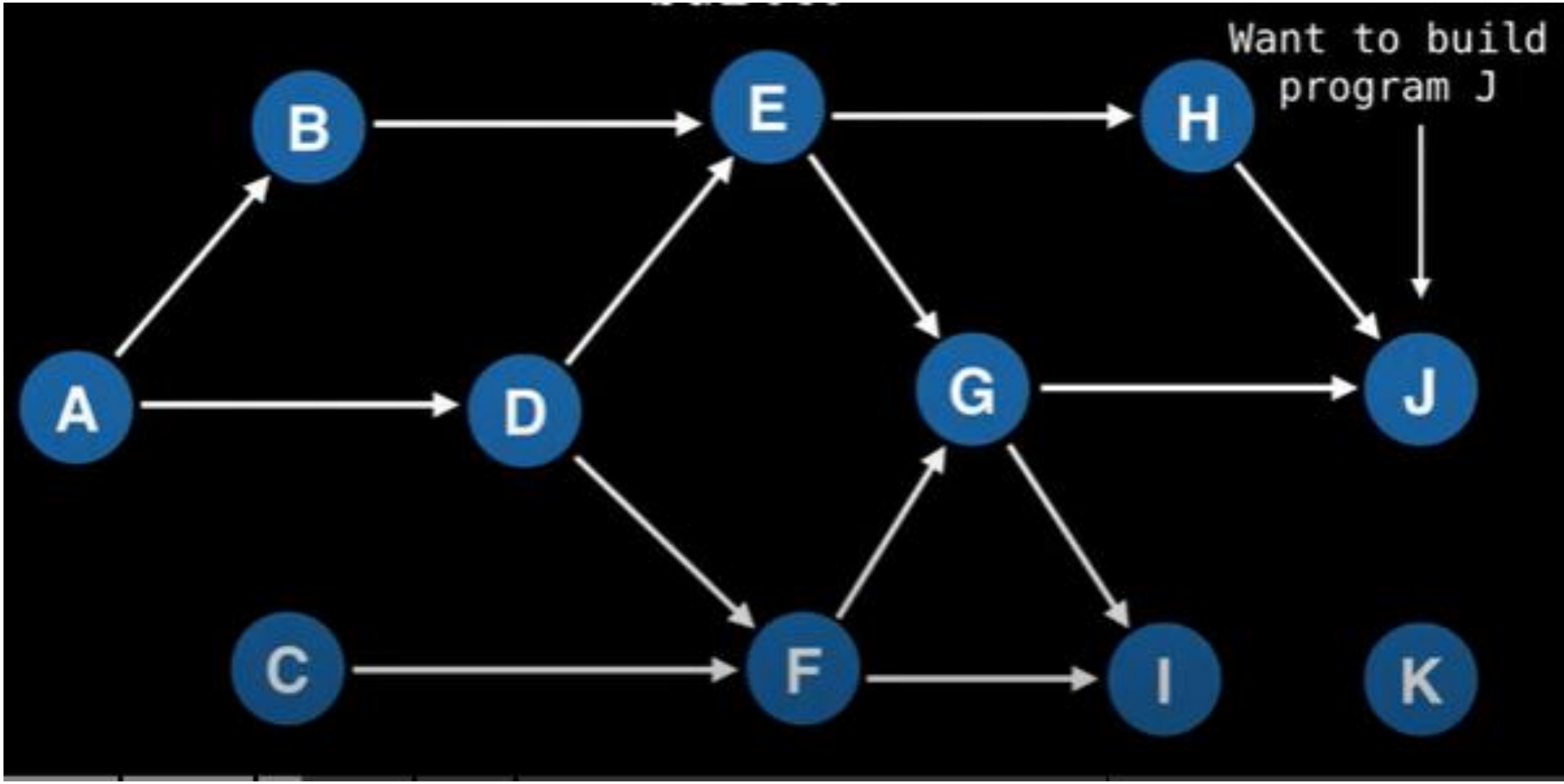
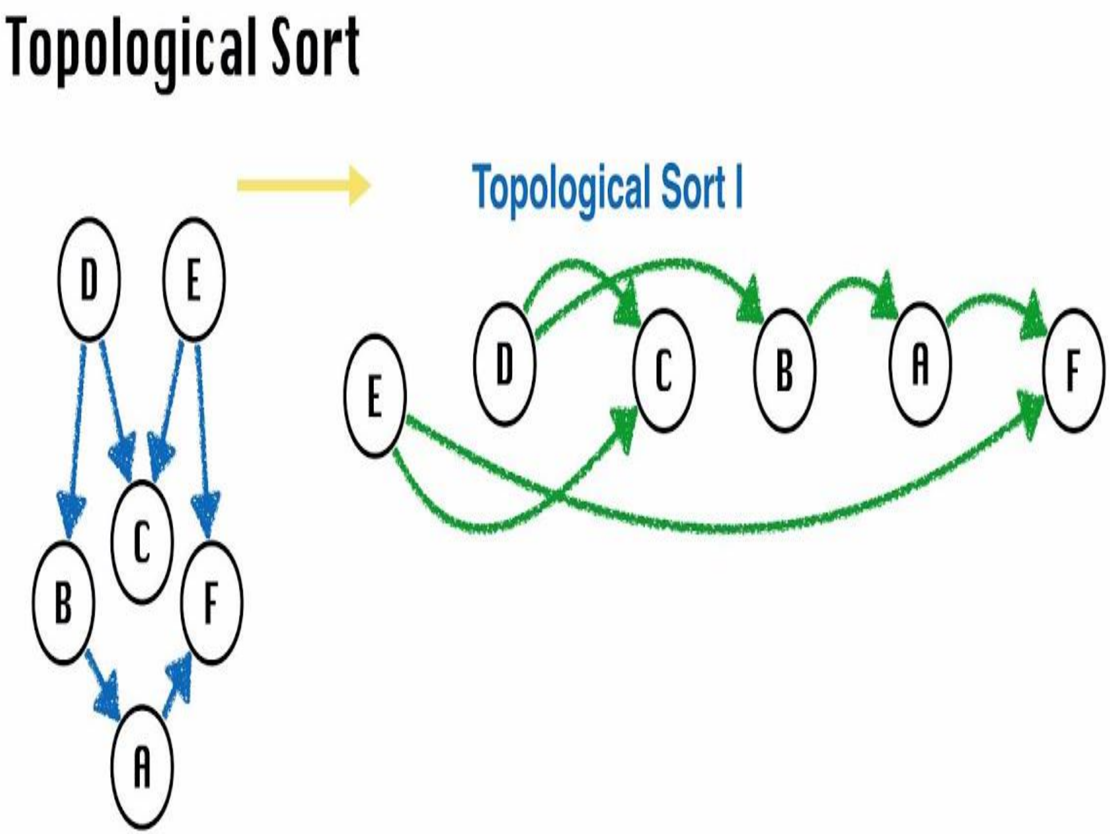
Topological Sorting

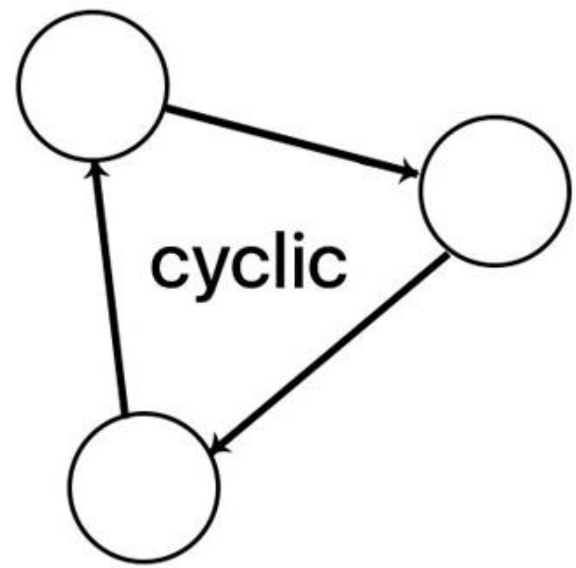
- Many real world situations can be modelled as a graph with directed edges where some events must occur before the others.
- School class prerequisites
- Program dependencies
- Event scheduling
- Assembly instructions
- Job scheduling

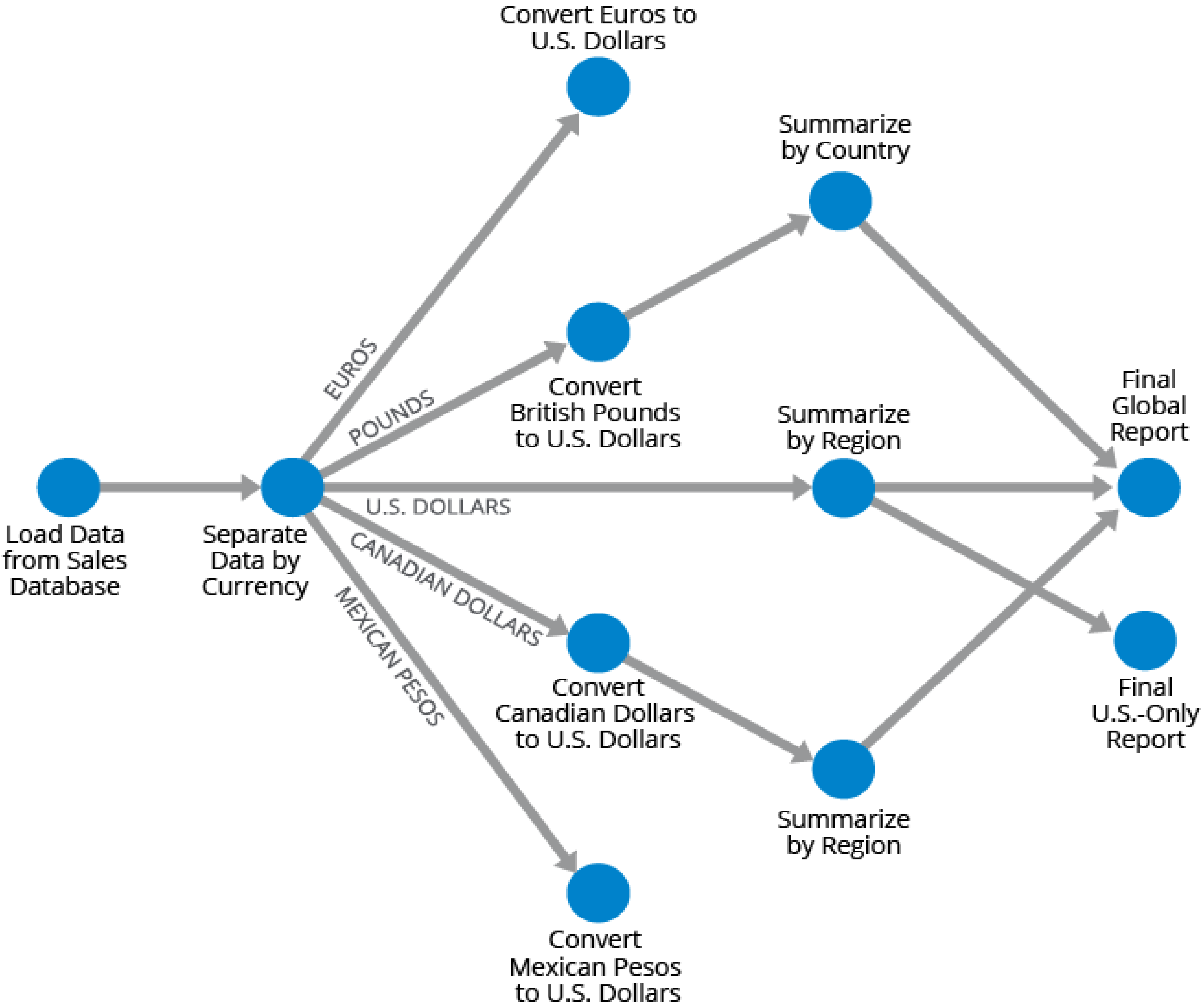

Class C
Class A
Class J
Class D
Class E
Class H
Class B
Class F
Class I

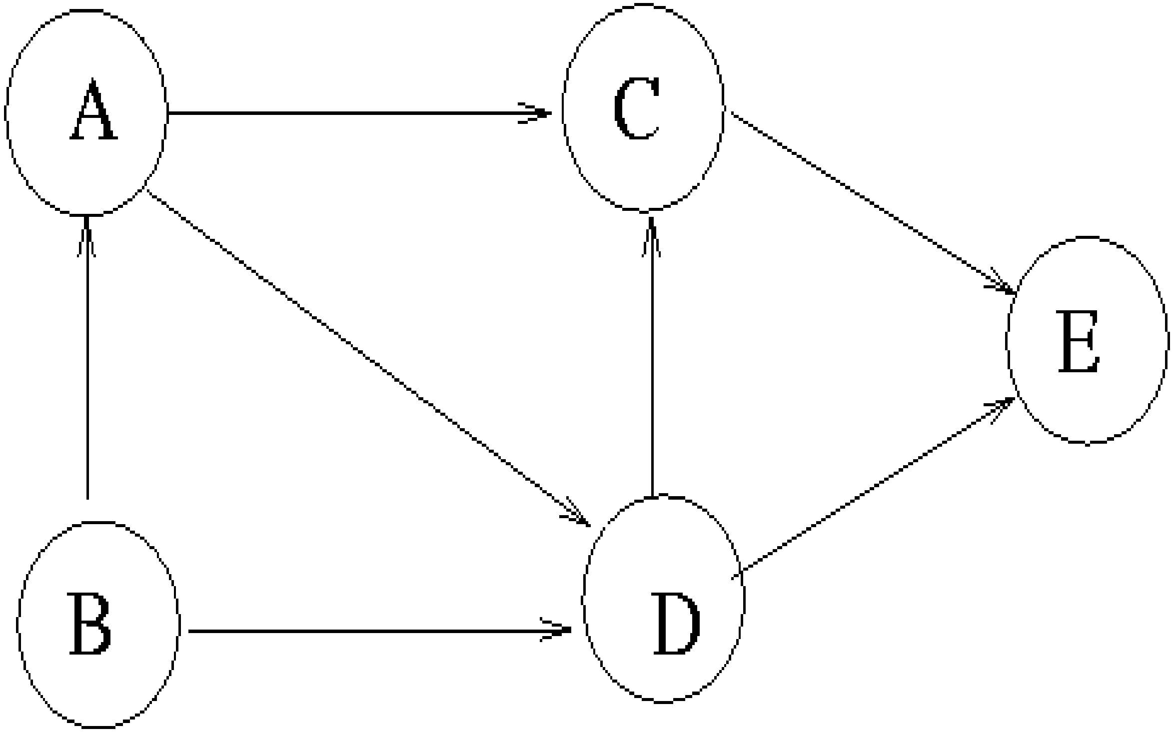

Want to build program J
A
B
C
D
E
F
G
H
I
J
K

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

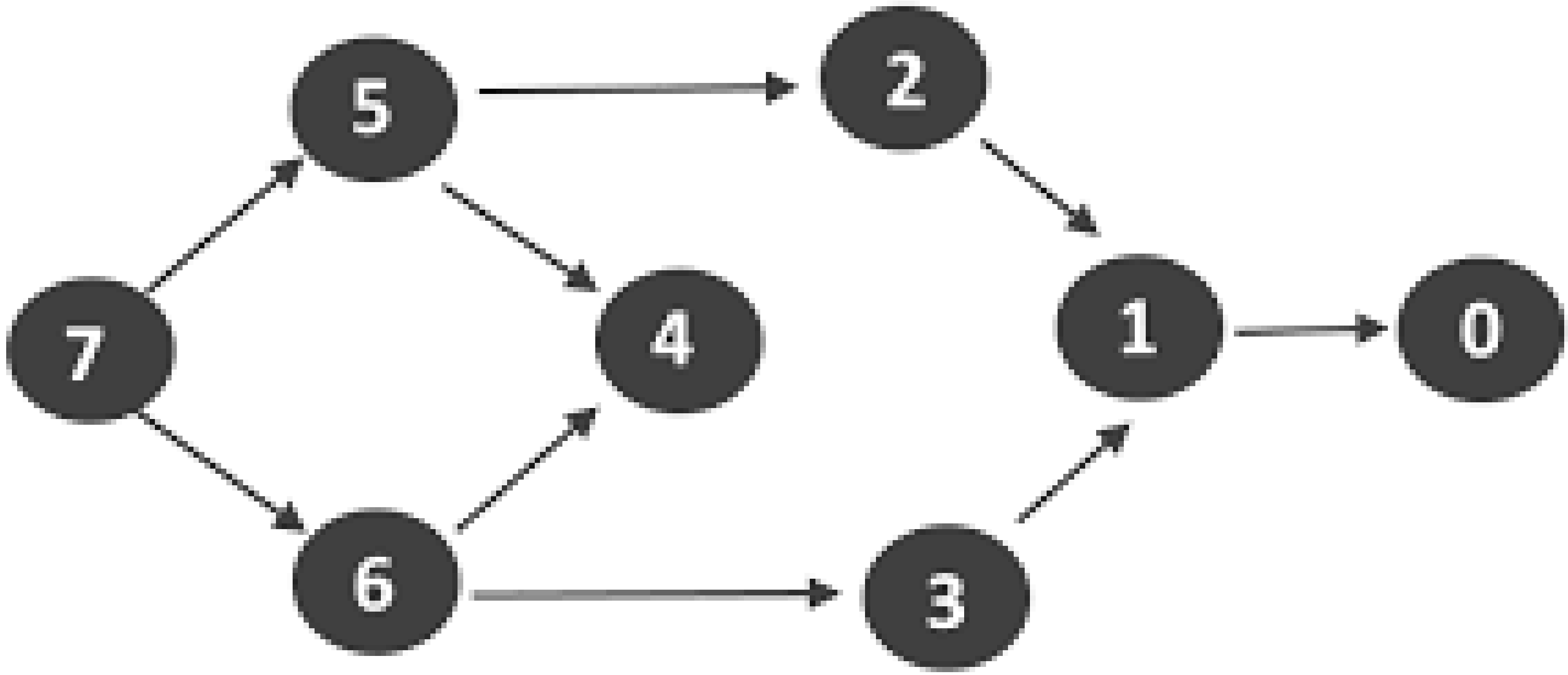




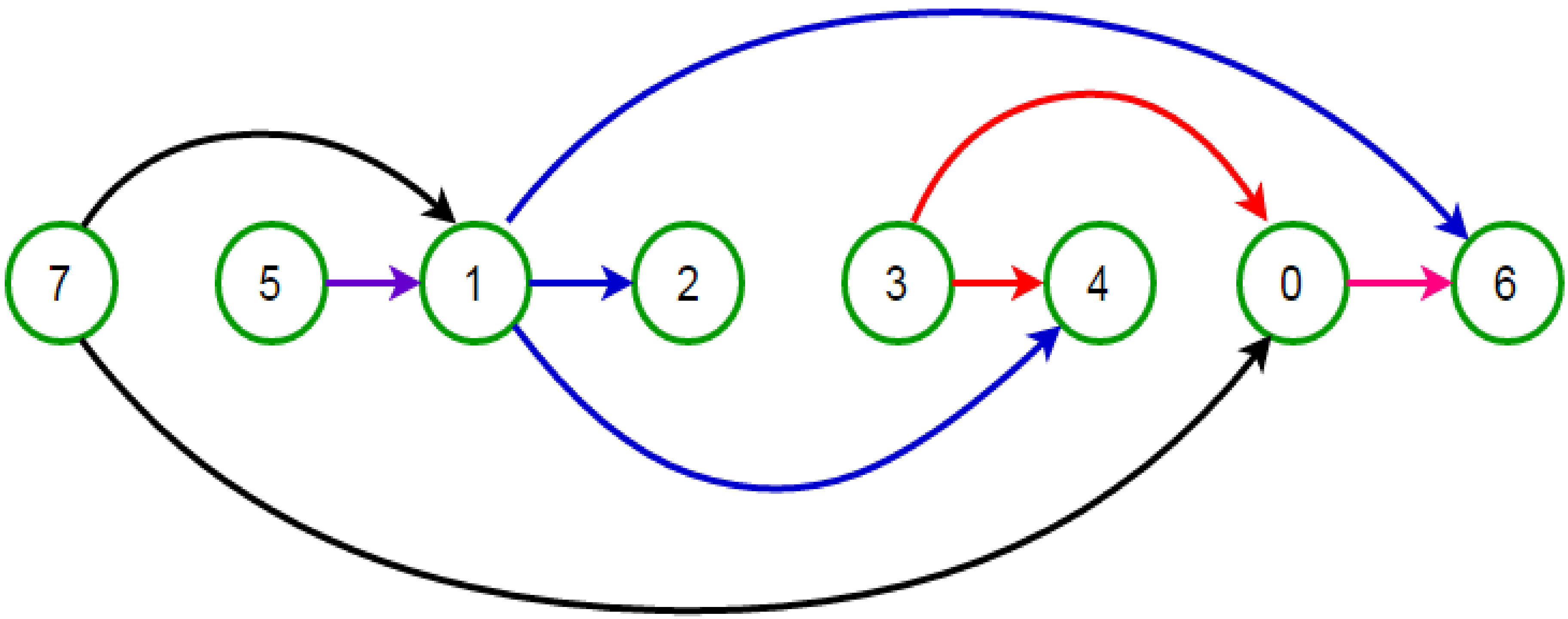

Convert Euros to U.S. Dollars
Summarize by Country
EUROS
POUNDS
Convert British Pounds to U.S. Dollars
Summarize by Region
Final Global Report
Load Data from Sales Database
Separate Data by Currency
U.S. DOLLARS
CANADIAN DOLLARS
MEXICAN PESOS
Convert Canadian Dollars to U.S. Dollars
Final U.S.-Only Report
Summarize by Region
Convert Mexican Pesos to U.S. Dollars


A
C
E
B
D

```
printf("\nThe topological order is:\n");

    for(i=1;i<=n;i++){
      j=1;
      while(j<=n){
        if(flag[j]==0 &&indeg[j]==0)
            {
                        printf("%d ",j);
        flag [j]=1;



      for(k=1;k<=n;k++)
        if(a[j][k]!=0)
          indeg[k]--;
          break;
      }
      j++;
      }
      if(j==n+1)
        {printf("Not possible Cycle Exist");break;
    }
    }
    }
```

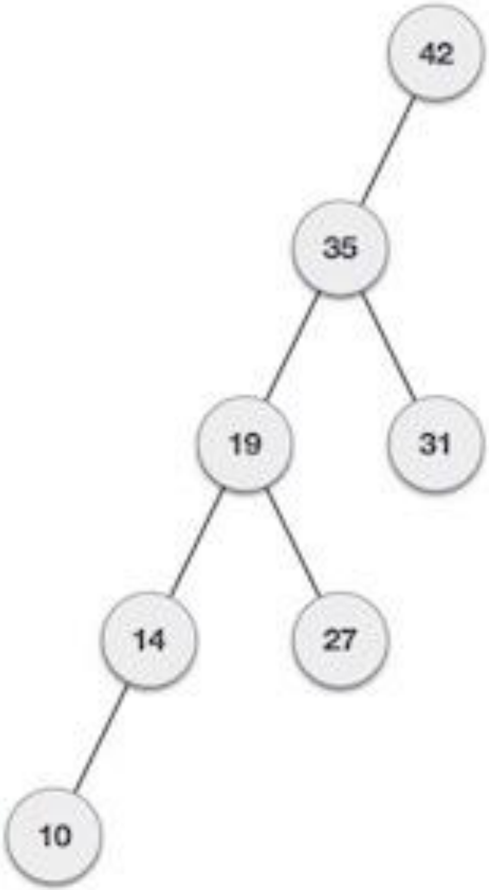


Topological Sort : 7 6 5 4 3 2 1 0

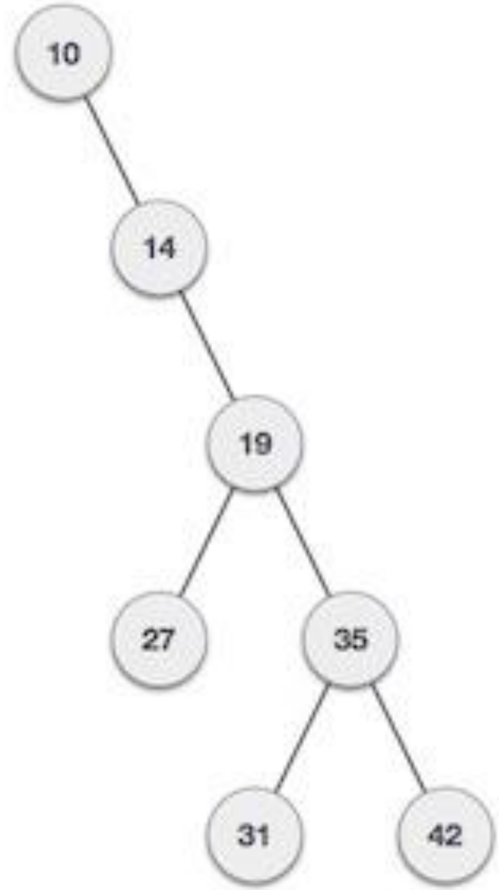


Topological Order

Binary Search Tree



If input 'appears' non-increasing manner



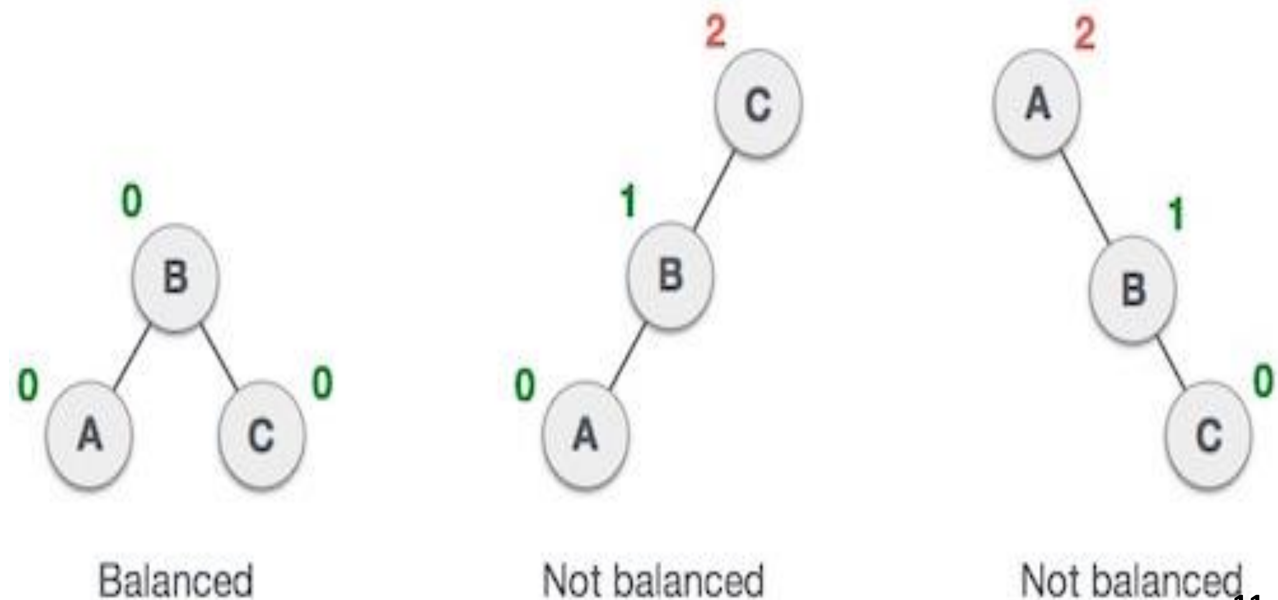
If input 'appears' in non-decreasing manner

AVL Tree

Named after their

inventor **Adelson**, **Velski** & **Landis**, **AVL trees** are height balancing binary search tree. AVL tree checks the height of the left and the right sub-trees and assures that the difference is not more than 1. This difference is called the **Balance Factor**.

- Here we see that the first tree is balanced and the next two trees are not balanced
- In the second

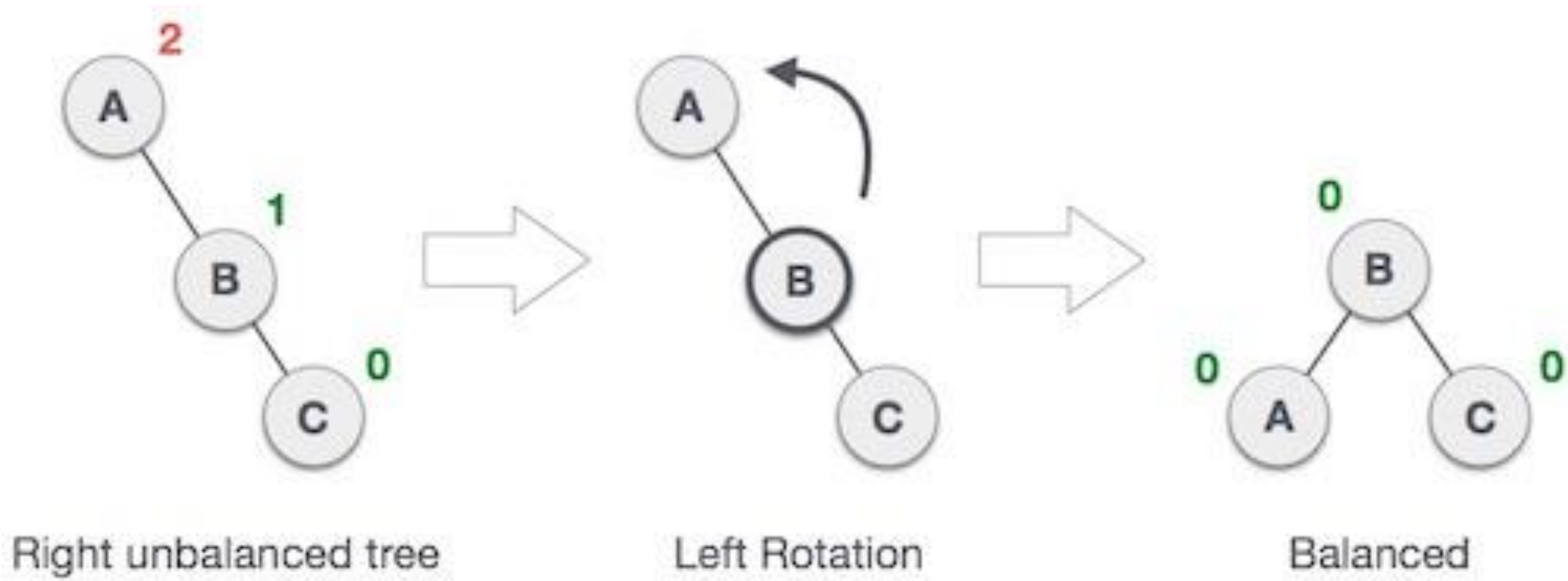


AVL Rotations

- To balance itself, an AVL tree may perform the following four kinds of rotations –
- Left rotation
- Right rotation
- Left-Right rotation
- Right-Left rotation

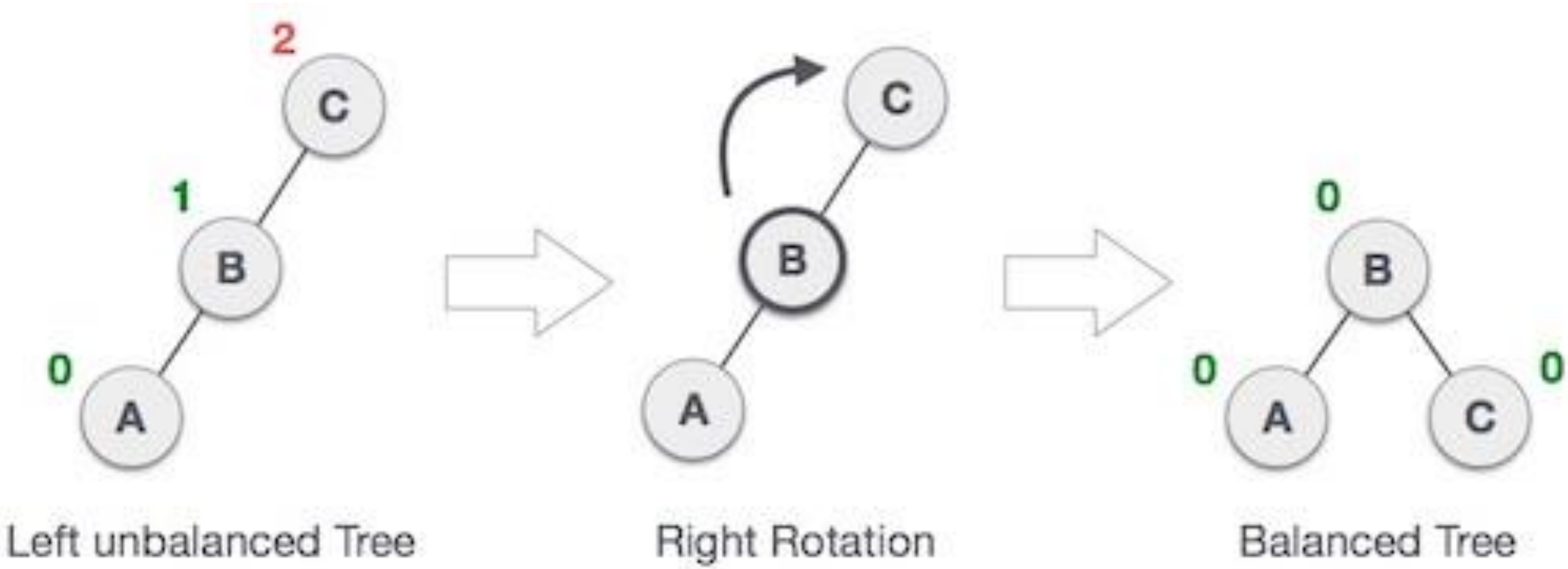
Left Rotation

If a tree becomes unbalanced, when a node is inserted into the right sub-tree of the right sub-tree, then we perform a single left rotation –

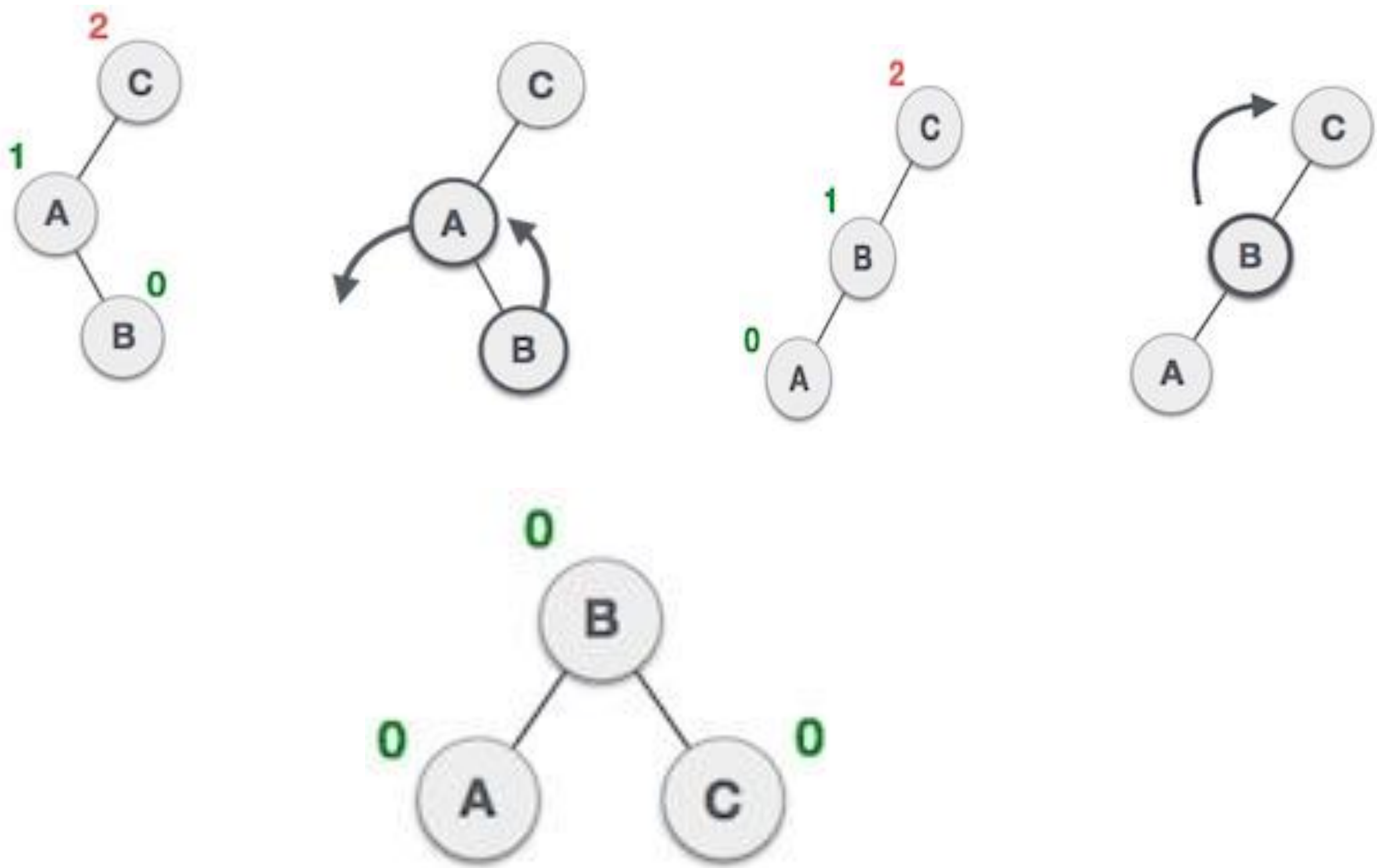


Right Rotation

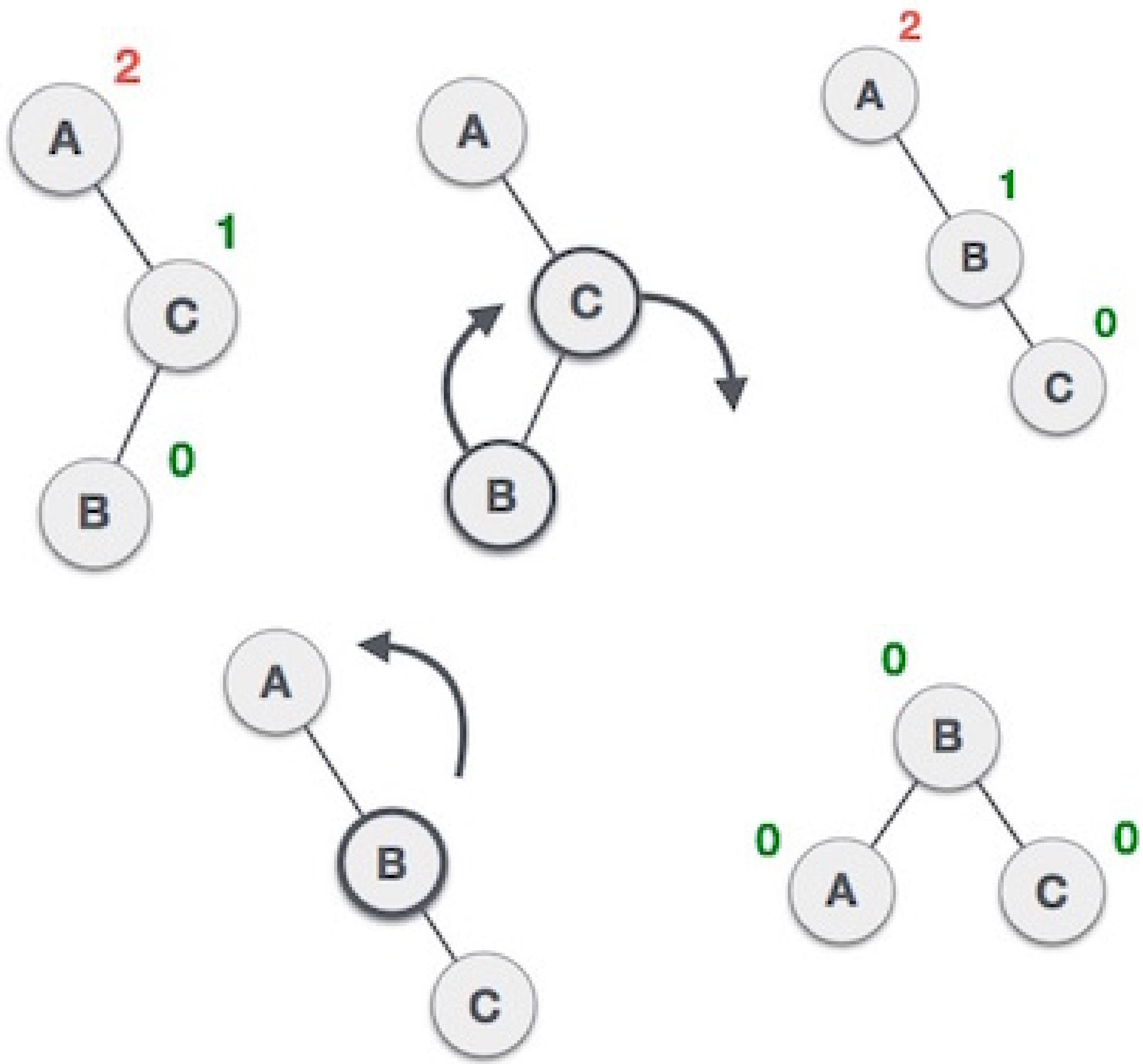
AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.



Left-Right Rotation



Right-Left Rotation

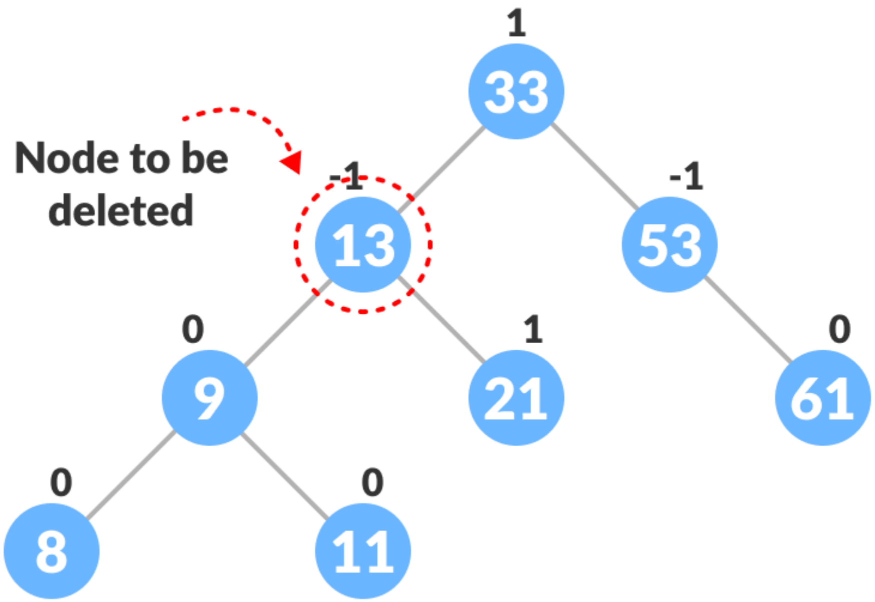


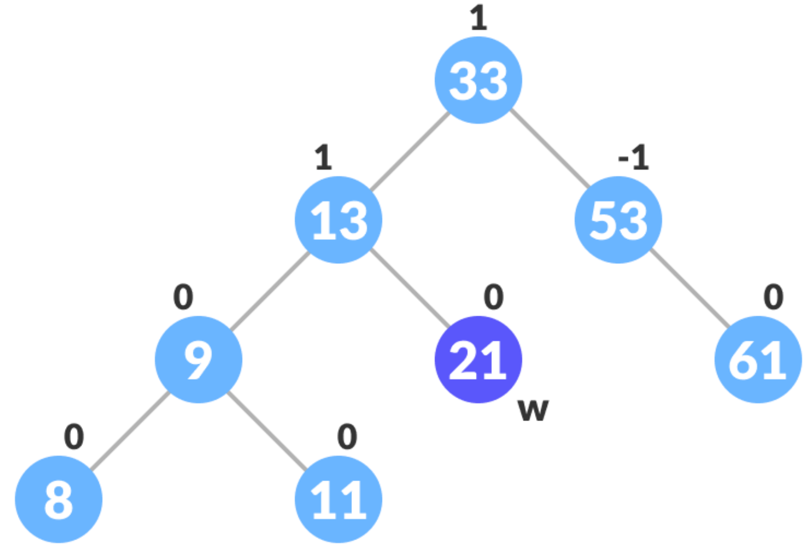
Problem-

Construct AVL Tree for the following sequence of numbers-
50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

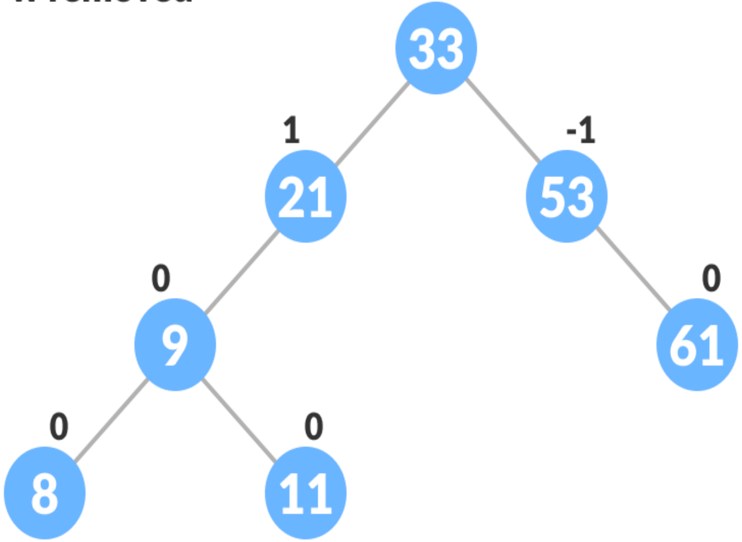
Delete Operation

- There are three cases for deleting a node:
- If nodeToBeDeleted is the leaf node (ie. does not have any child), then remove nodeToBeDeleted.
- If nodeToBeDeleted has one child, then substitute the contents of nodeToBeDeleted with that of the child. Remove the child.
- If nodeToBeDeleted has two children, find the inorder successor w of nodeToBeDeleted (ie. node with a minimum value of key in the right subtree).

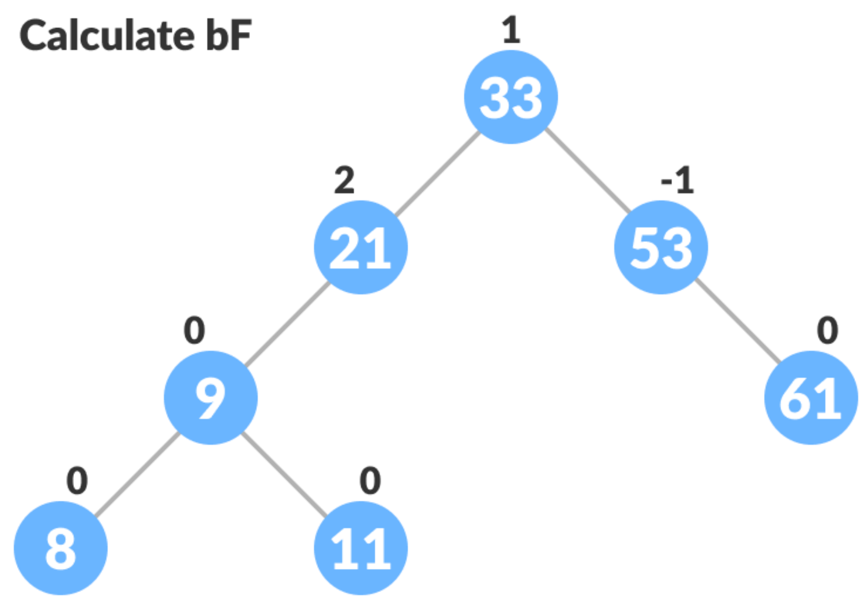


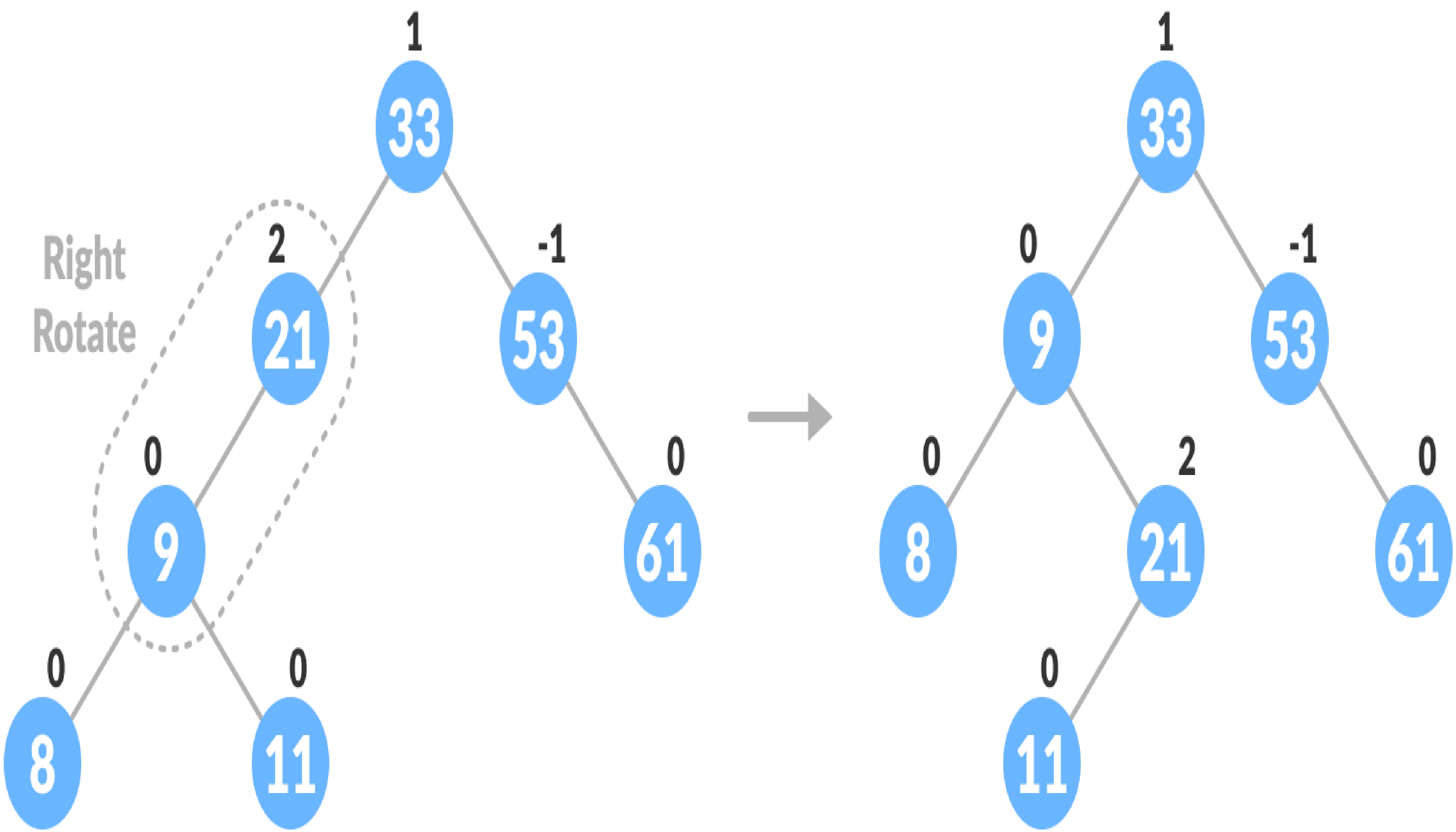


w removed



Calculate bF





Right
Rotate
1
33
2
21
-1
53
0
9
0
61
0
8
0
11
1
33
0
9
-1
53
0
8
2
21
0
61
0
11