Ch.5.7-5.10 Trees 參考答案

A little more about Disjoint Set

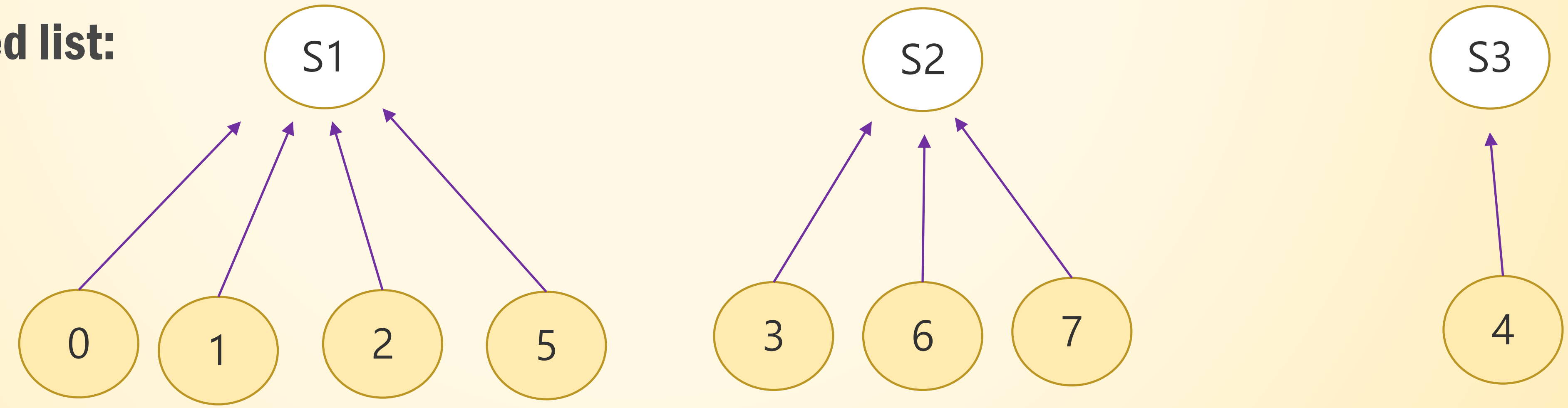
- Quick Find
- Simple Union with Simple Find
- Weighted Union with Collapsing Find

Quick Find

- Using array or linked list
  - Eg. DS = {0, 1, 2, 5}, {3, 6, 7}, {4}
  - Array:

id	0	1	2	3	4	5	6	7
Set	S1	S1	S1	S2	S3	S1	S2	S2

  - Linked list:

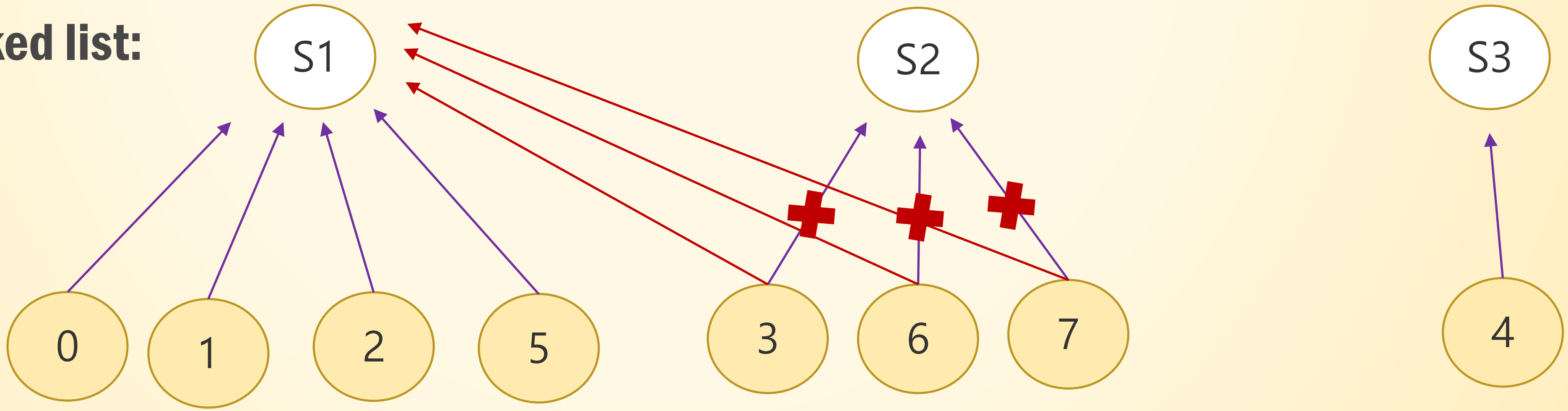


Union with Quick Find

- Using array or linked list
  - Eg. DS = {0, 1, 2, 5}, {3, 6, 7}, {4}
  - Array:

id	0	1	2	3	4	5	6	7
Set	S1	S1	S1	S1	S3	S1	S1	S1

  - Linked list:



Union with Quick Find

- Time complexity:
  - Union → O(n)
  - Find → O(1)

- How to improve the union(i, j) operation?

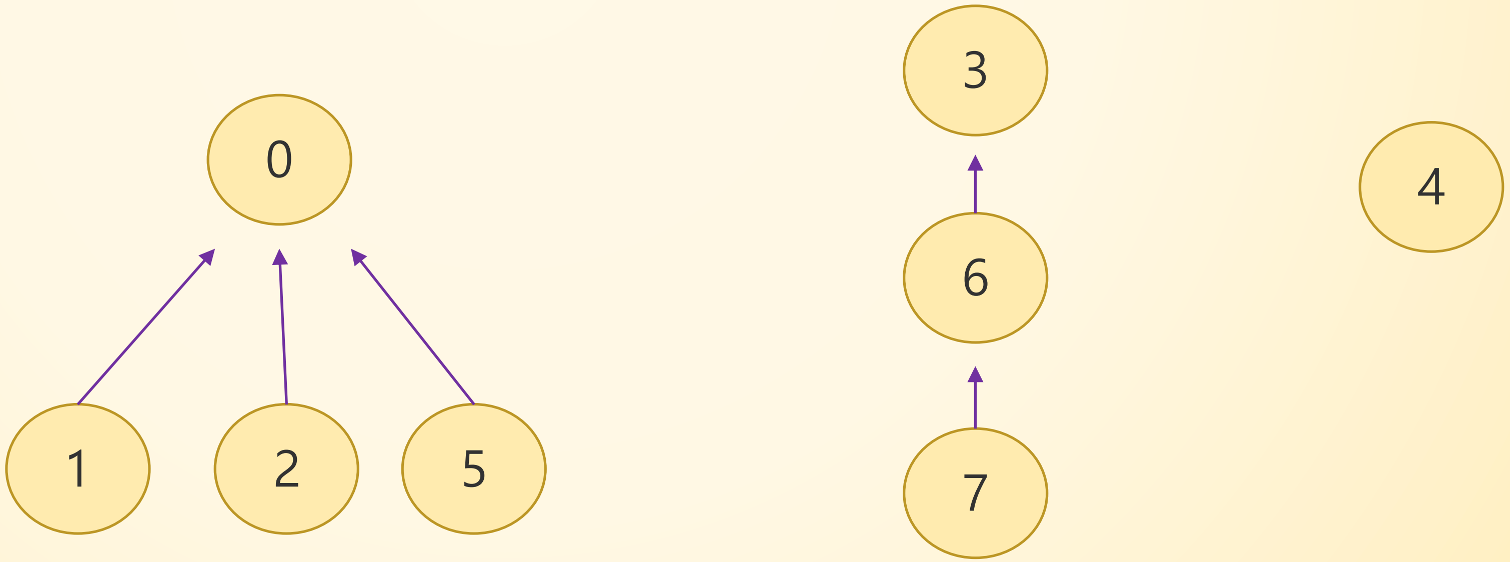
  e.g. Simple Union and Simple Find

Simple Find

- Using array or linked list
  - Eg. DS = {0, 1, 2, 5}, {3, 6, 7}, {4}
  - Array:

id	0	1	2	3	4	5	6	7
parent	-1	0	0	-1	-1	0	3	6

  - Linked list:



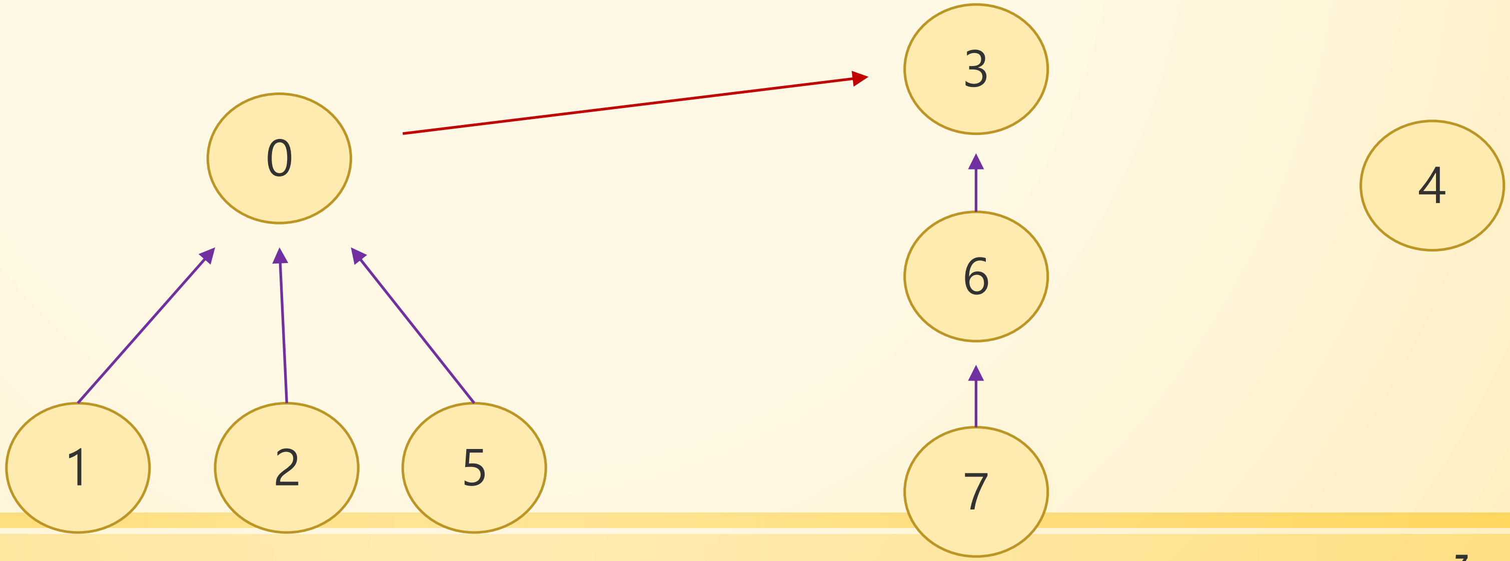

```
SimpleFind(int i)
{
    while (parent[i]>=0)
        i = parent[i];
    return i;
}
```

Simple Union

- Using array or linked list
  - Eg. DS = {0, 1, 2, 5}, {3, 6, 7}, {4}
  - Array:

id	0	1	2	3	4	5	6	7
parent	3	0	0	-1	-1	0	3	6

  - Linked list:




```
SimpleUnion(int i, int j)
{
    parent[i] = j;
}
```

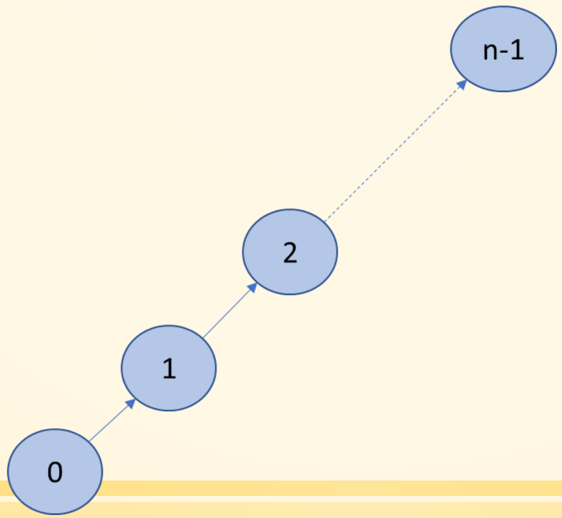
Simple Union

Consider the following situation:

Union(0,1)➔Union(1,2) ➔Union(2,3)➔.....➔Union(n-2,n-1)

In this situation, find(x) operation would have O(n) time in average case.

Random union may cause performance degradation of find operations.



Simple Find with Simple Union

- Time complexity:
  - Union → O(1)
  - Find → O(n)

- Can we provide a win-win solution for find(i) and union(i, j) operations ?

  e.g. Weighted Union

Weighted Union

- "Union by weight"
  - if the **number** of nodes in the tree with root i is less than the **number** in the tree with root j, then make j the parent of i; otherwise make i the parent of j.

- Alternative: "Union by height"
  - if the **height** of the tree with root i is less than the **height** of the tree with root j, then make j the parent of i; otherwise make i the parent of j.

Weighted Union with Simple Find

- Time complexity:
  - Union: O(1)
  - Find: O(log n)

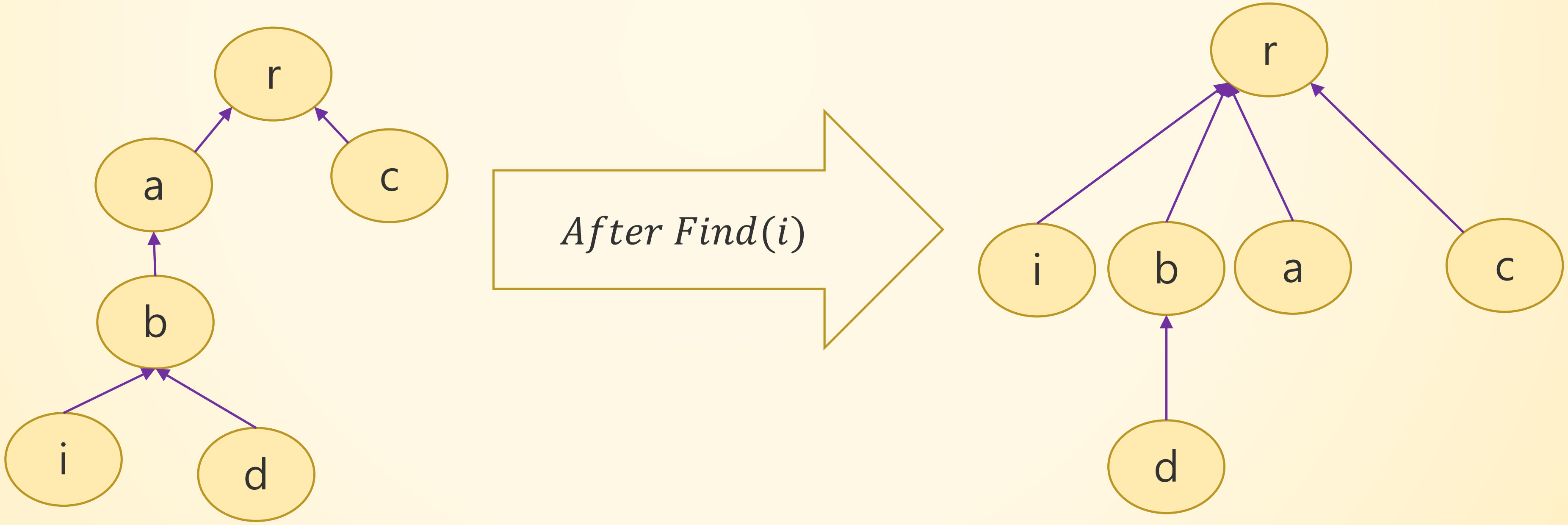
- Is it possible to improve further?

  e.g. Weighted Union with **Collapsing Find**

Collapsing Find

- **path compression:**

  After $Find(i)$,
  if(j *is a node on the path from i to its root*) **and** $(parent[i] \neq root(i))$, **then** $parent[j]$ *is set to root*.



Weighted Union with Collapsing Find

- Time complexity:
  - Union: O(1)
  - **Find: O($\alpha(n)$)**

- Inverse Ackermann: α
  - $\alpha(n) = \frac{1}{A(n,n)}$ (in practice, this value is at most 4)

Exercises

Q1:

Construct a BST from the given preorder traversal result: 15, 4, 1, 12, 10, 20, 30. Explain your approach.

A1:

Preorder : 15, 4, 1, 12, 10, 20, 30

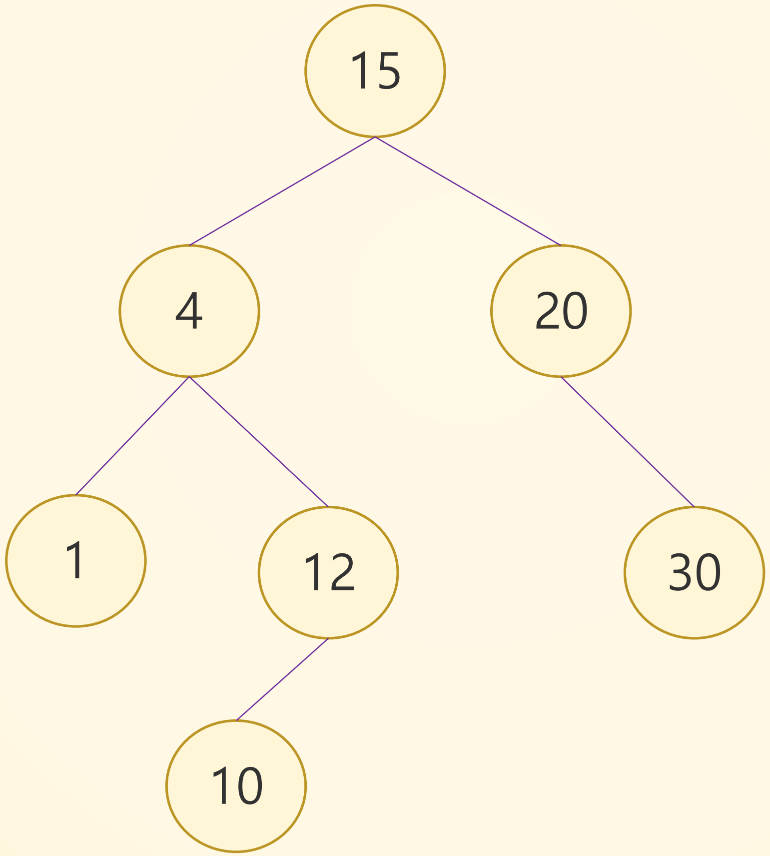
➔15 is the root of this binary search tree

Due to binary search tree

➔{4, 1, 12, 10} is the left subtree, {20, 30} is right subtree

And so on....

A1:



Q2:

Construct a BST from the given postorder traversal result: 2, 6, 4, 9, 13, 11, 7. Explain your approach.

A2:

Postorder : 2, 6, 4, 9, 13, 11, 7

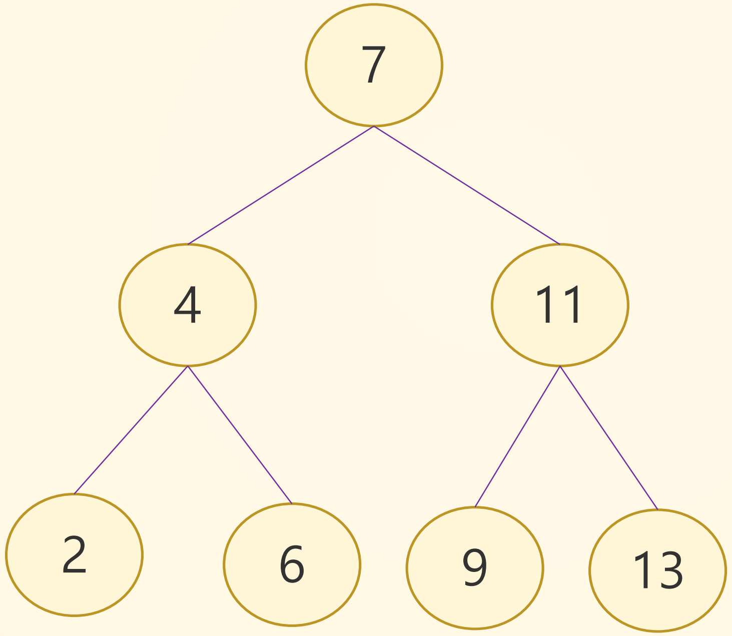
➔7 is the root of this binary search tree

Due to binary search tree

➔{2, 6, 4} is the left subtree, {9, 13, 11} is right subtree

And so on….

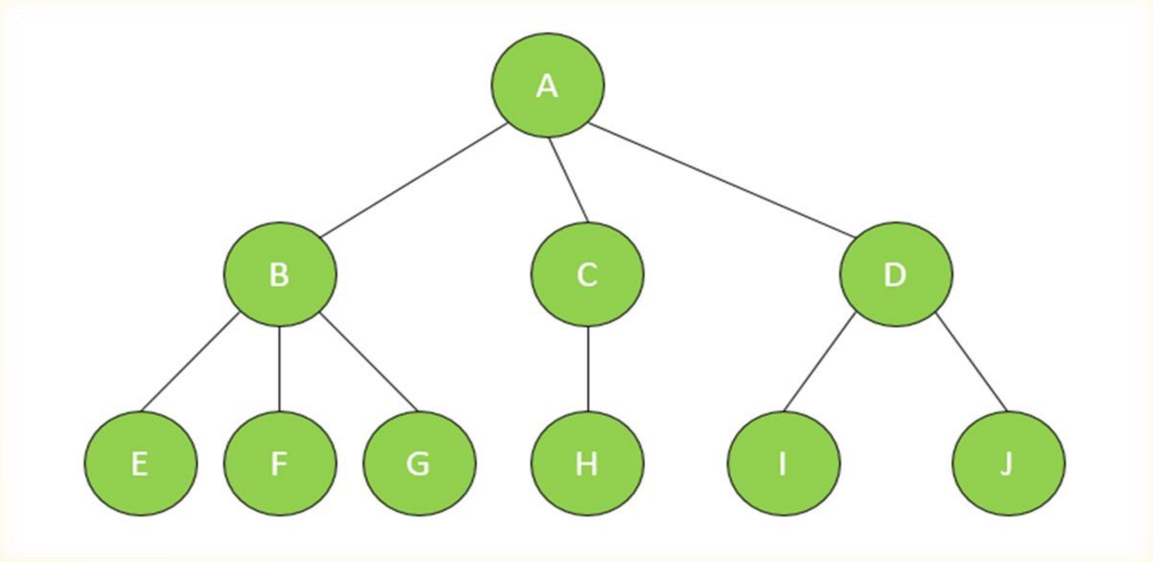
A2:



Q3 (1) :

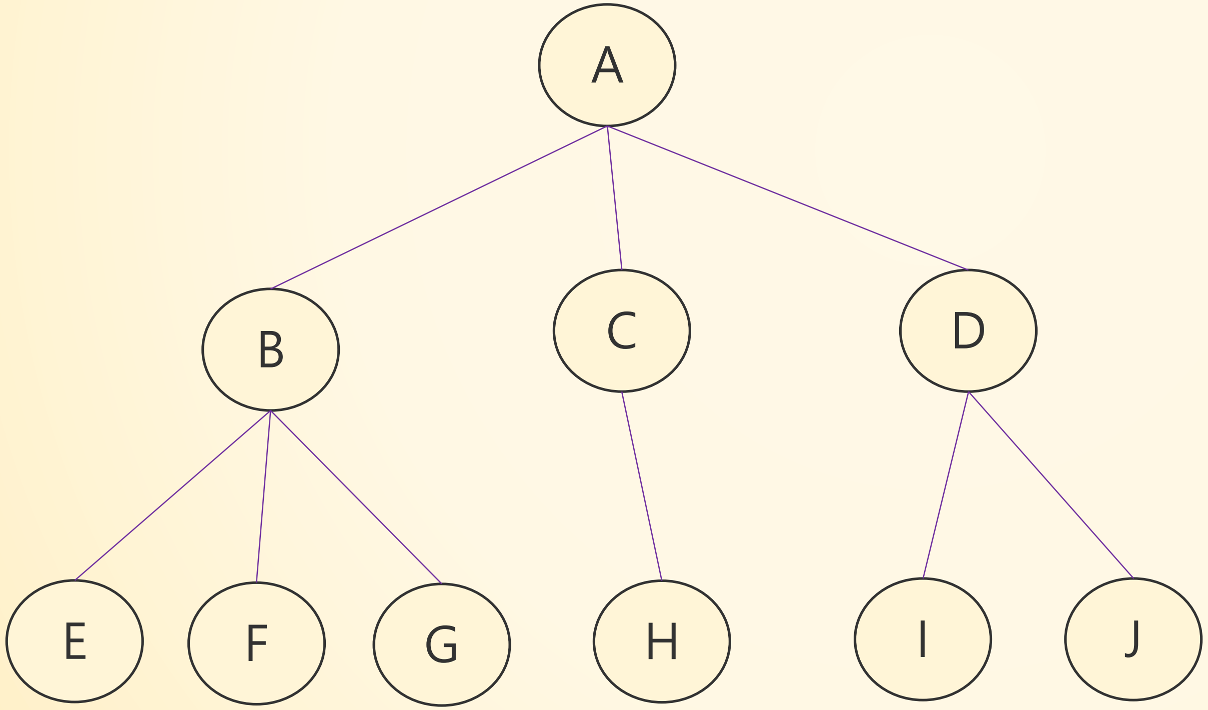
Please propose and explain your solutions to the following processes.

(1) Transform the tree below into a binary tree



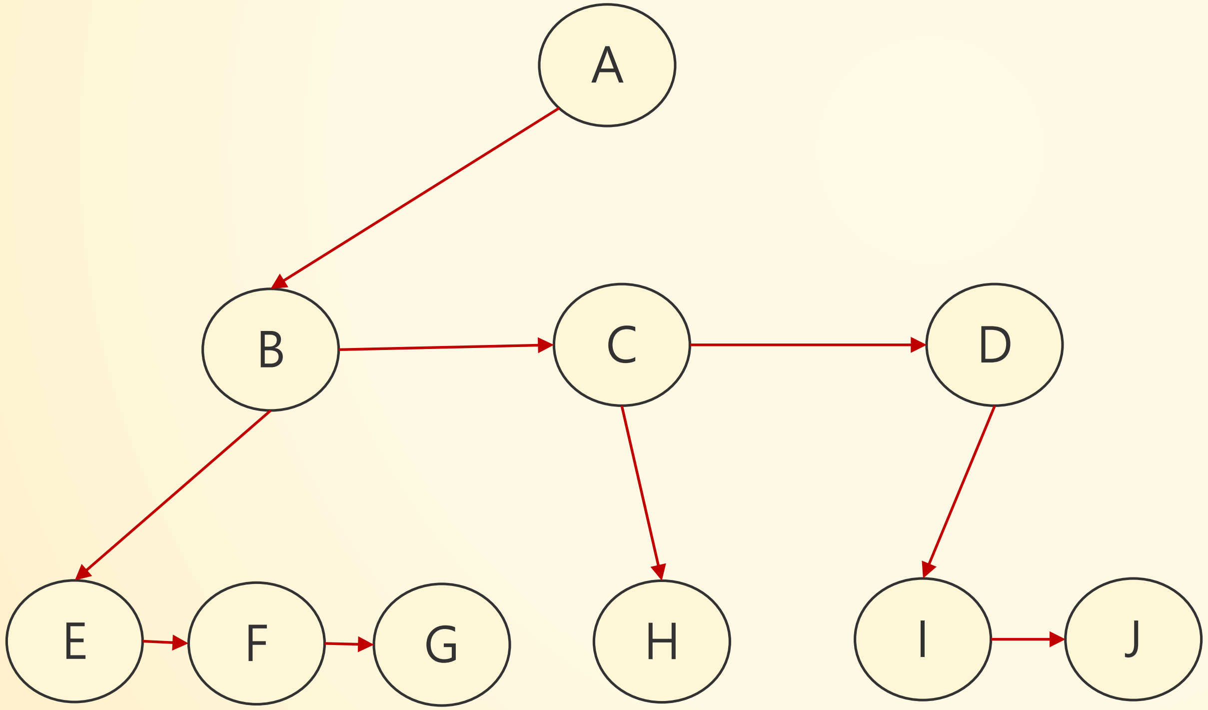
A3 (1):

Using “left-child right-sibling”



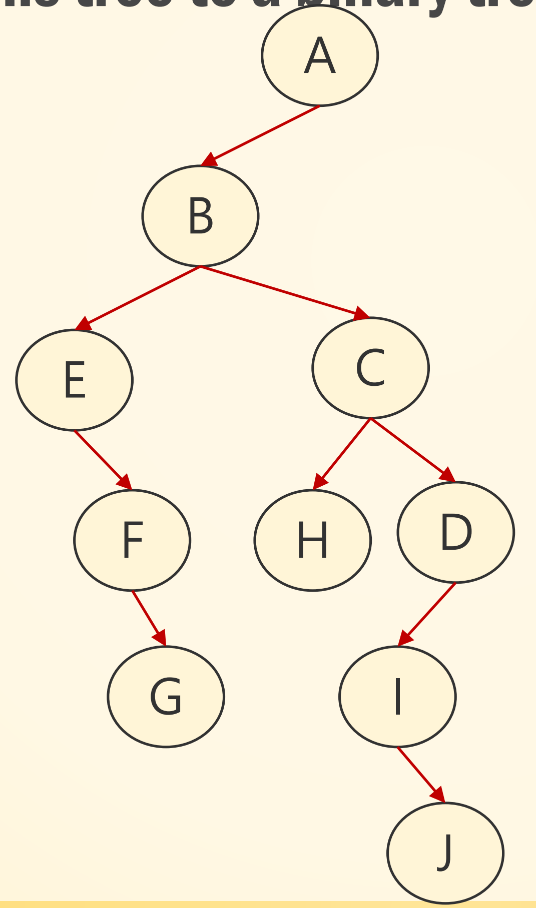
A3 (1):

Using “left-child right-sibling”



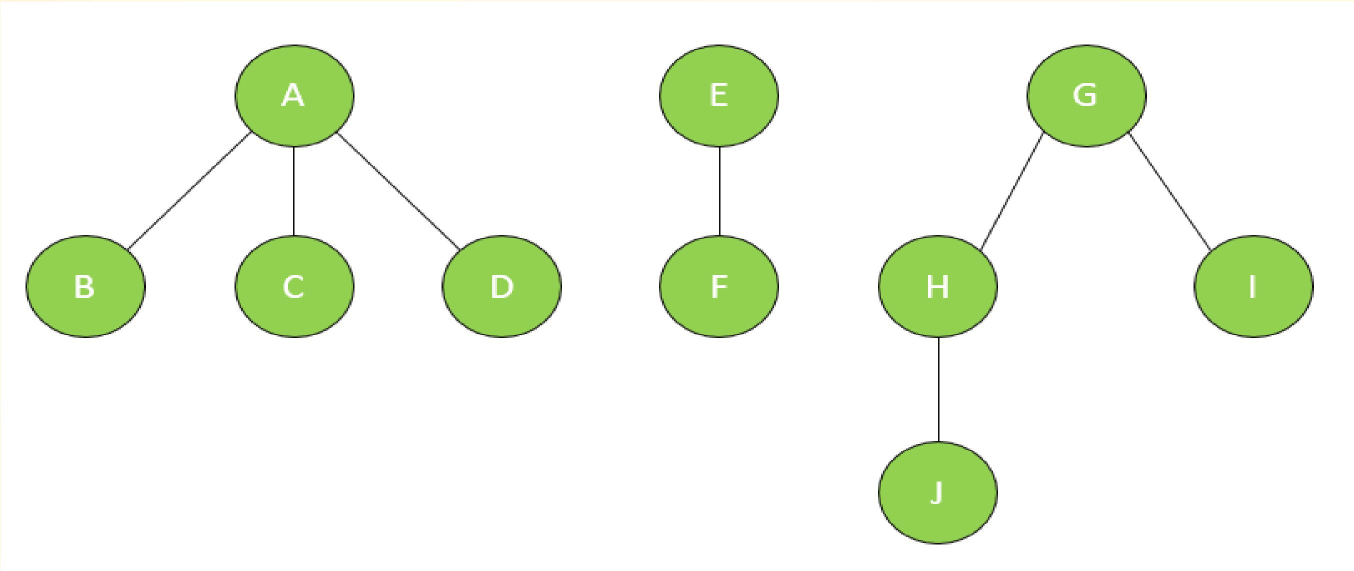
A3 (1):

Or we can transform this tree to a binary tree by turning each sibling 45° clockwise.



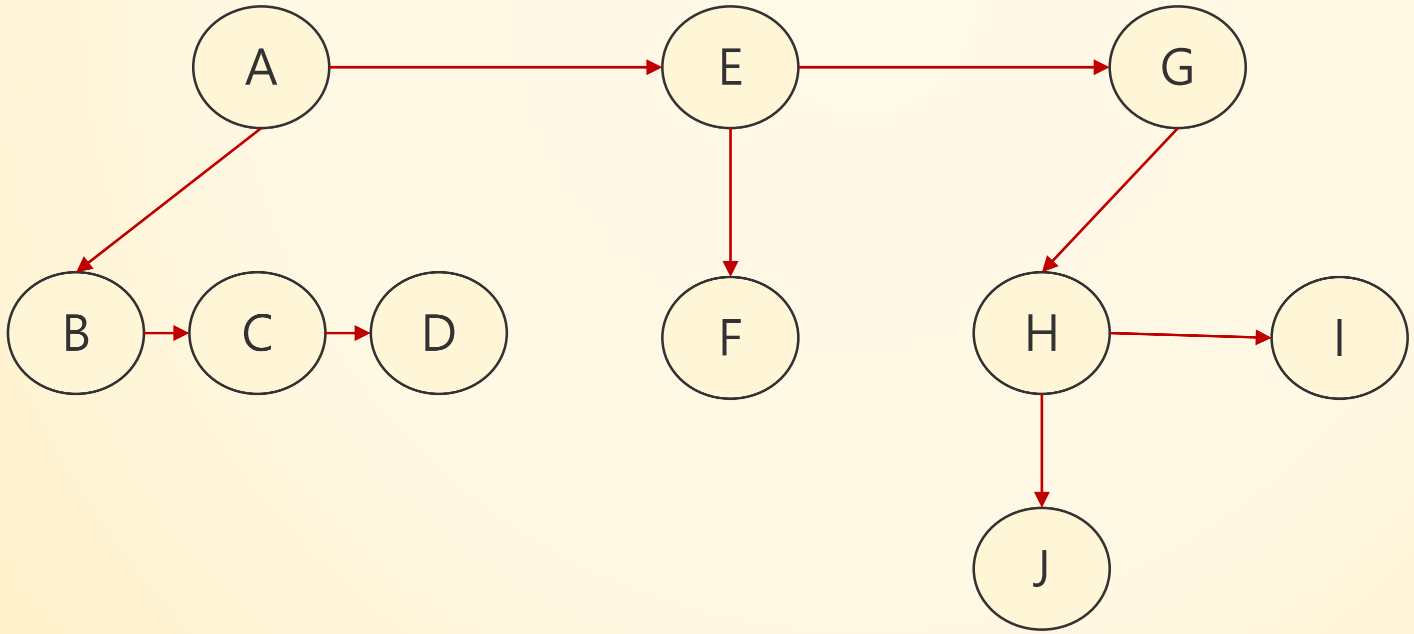
Q3 (2):

Please propose an algorithm that can transform a forest into binary tree.



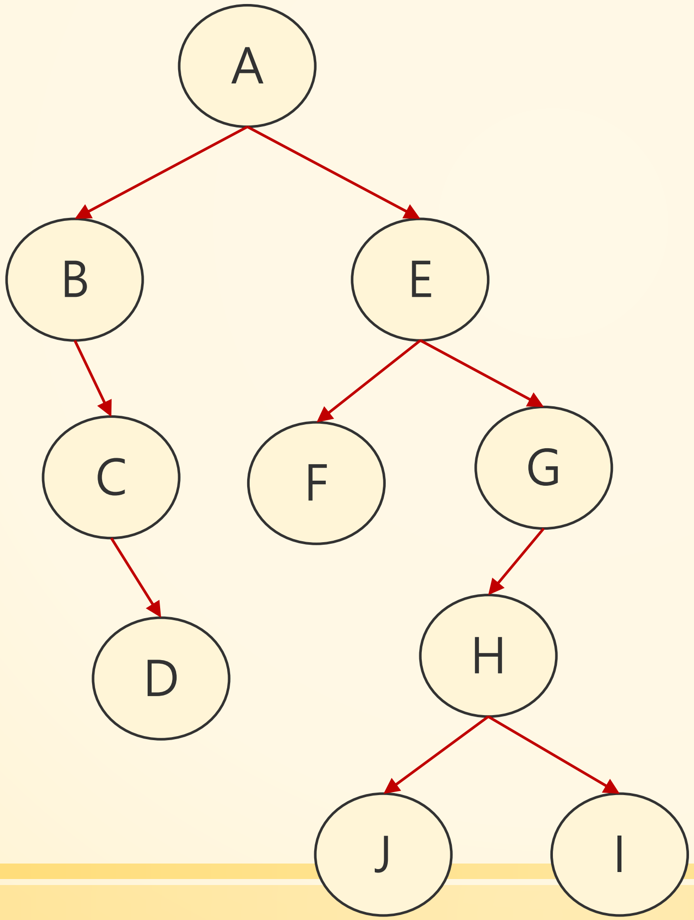
A3 (2):

Using “left-child right-sibling”



A3 (2):

Or we can transform this forest to a binary tree by turning each sibling 45° clockwise.



Q4:

We want to construct a set which includes the elements $x_1, x_2 \dots x_n$, starting from an empty set, i.e., to put $x_1, x_2 \dots x_n$ in a set)

a. Calculate the total number of operations (make-set and union) needed to establish the set.

b. What is the worst-case time complexity? Please explain your analysis process.

A4:

A. n make-set and (n-1) union

B. Union with quick find:

Union(0,1)→Union(1,2) →.....→Union(n-2, n-1)

1. Union(0,1)

id	0	1	2	3	…	…	…	n-1
Set	S1	S1	S2	S3	…	…	…	Sn-1

2. Union(1,2)

id	0	1	2	3	…	…	…	n-1
Set	S2	S2	S2	S3	…	…	…	Sn-1

And so on....

→$O(n^2)$

Q5:

According to the previous exercise, can you think of any other approach to speed up the union function? If yes, please describe how dose it work and analyze the time complexity.

A5:

- Simple Find with Simple Union
- Weighted Union with Collapsing Find

Q6:

Given the disjoint set DS = {0, 1, 2, 5}, {3, 6, 7}, {4}, please show how you may use an "array" to represent this disjoint set. First describe your data structure and then show the result.

A6:

- **Method 1:**

id	0	1	2	3	4	5	6	7
parent	-1	0	0	-1	-1	0	3	3

Note: -1 means root

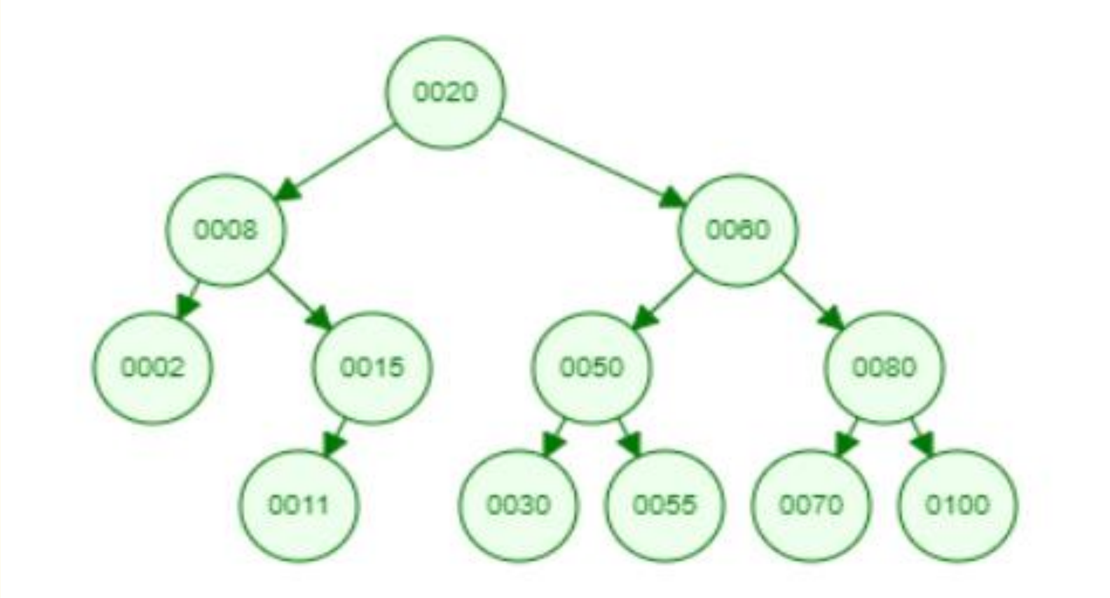
- **Method 2:**

id	0	1	2	3	4	5	6	7
Set	S1	S1	S1	S2	S3	S1	S2	S2

Q7:

Please describe how to delete a node for the following binary tree and show the result.

- Case 1 : Delete 0002 node
- Case 2 : Delete 0015 node
- Case 3 : Delete 0050 node



To Delete a node in a BST

1) ***Node to be deleted is leaf:*** Simply remove from the tree.

```
              50                            50
           /     \         delete(20)      /   \
          30      70       --------->    30     70
         /  \    /  \                     \    /  \
       20   40  60   80                   40  60   80
```

2) ***Node to be deleted has only one child:*** Copy the child to the node and delete the child

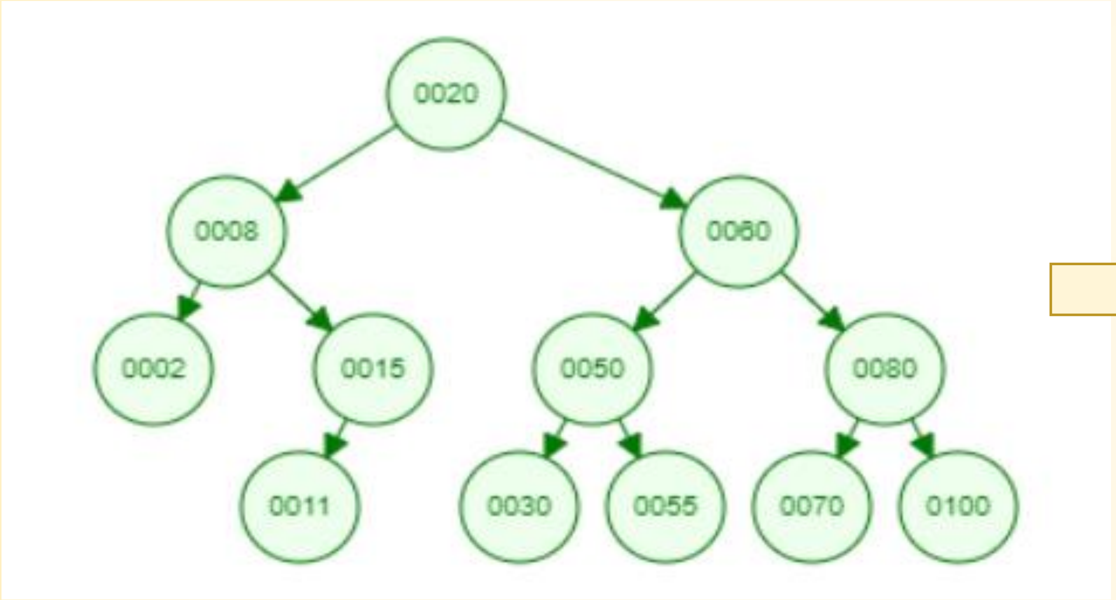
```
              50                            50
           /     \         delete(30)      /   \
          30      70       --------->    40     70
            \    /  \                          /  \
            40  60   80                       60   80
```

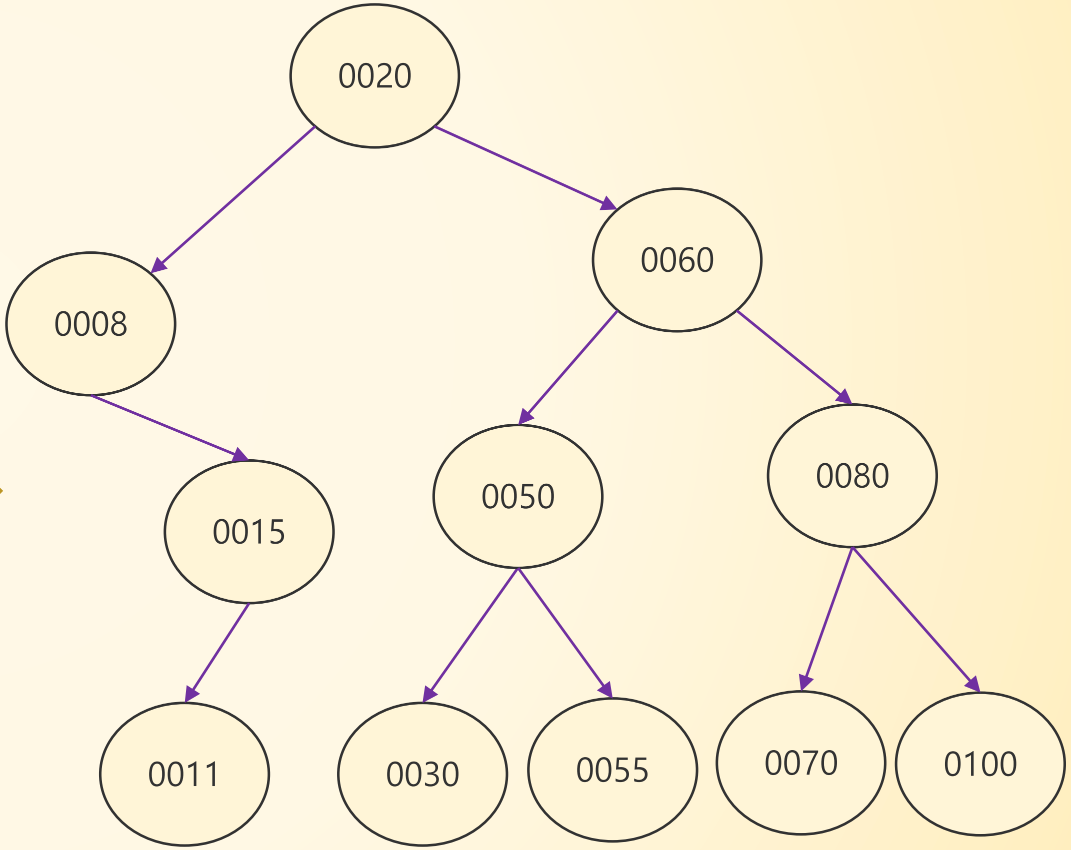
3) ***Node to be deleted has two children:*** Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.

```
              50                            60
           /     \         delete(50)      /   \
          40      70       --------->    40    70
                 /  \                            \
                60   80                           80
```

A7:

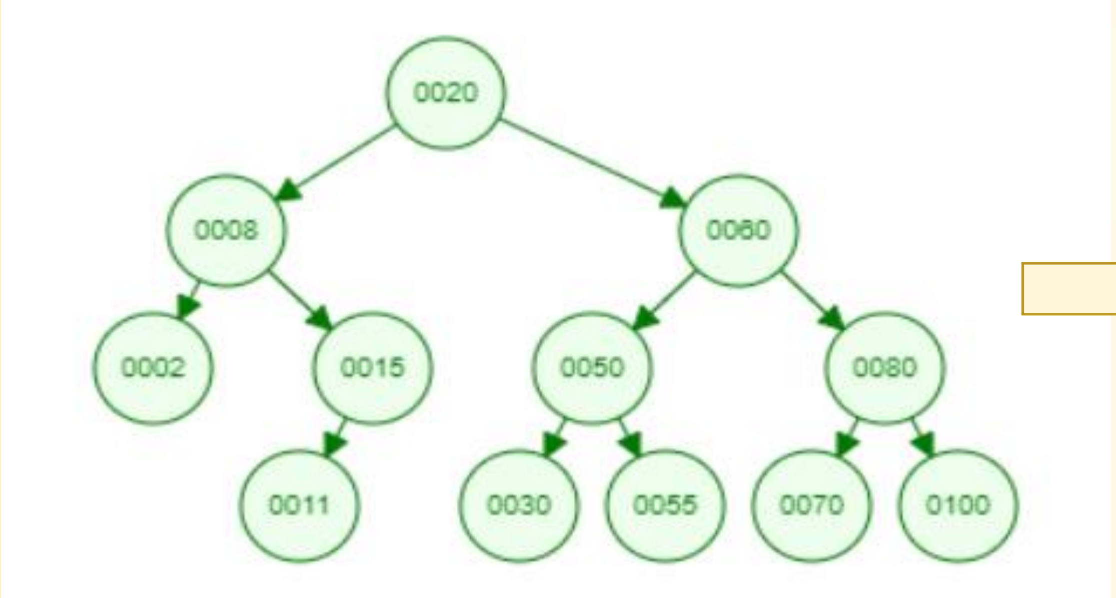
- **Case 1 : Delete 0002 node**

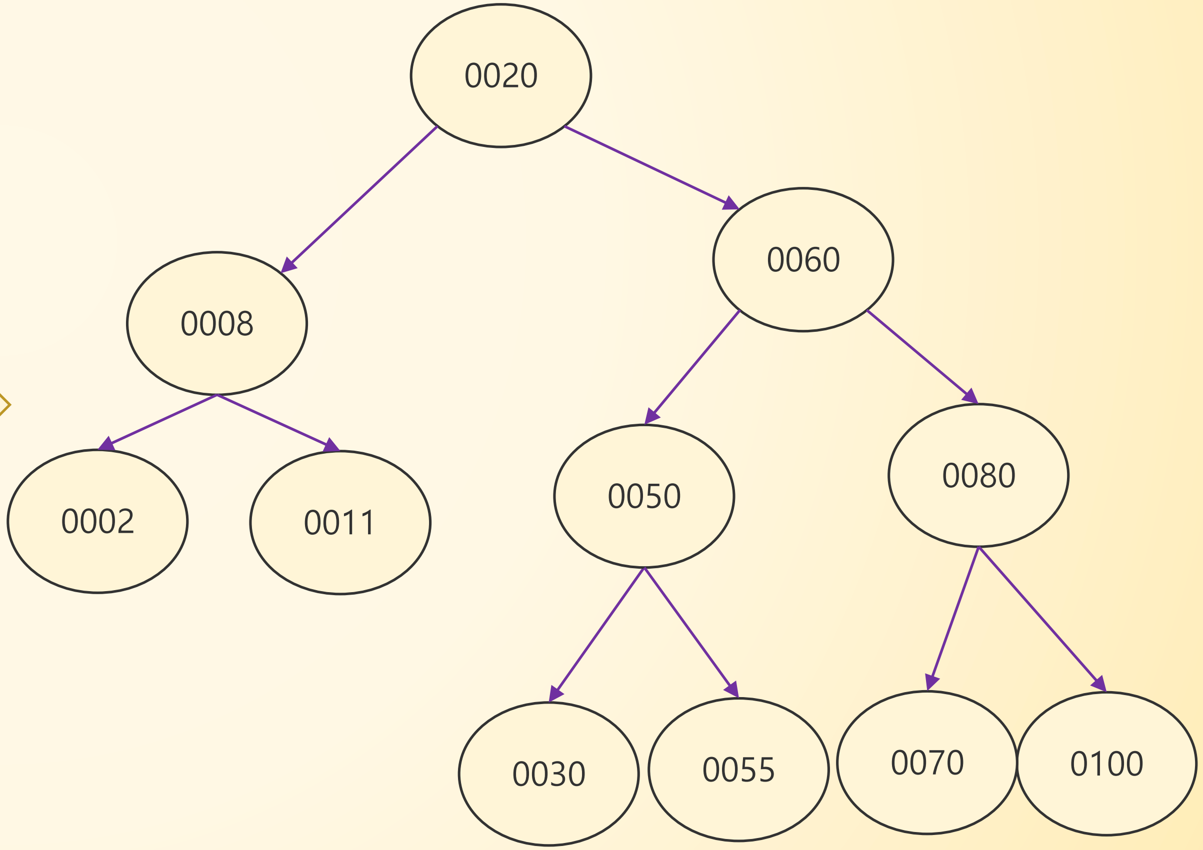




A7:

- **Case 2 : Delete 0015 node**





A7:

- **Case 3 : Delete 0050 node**