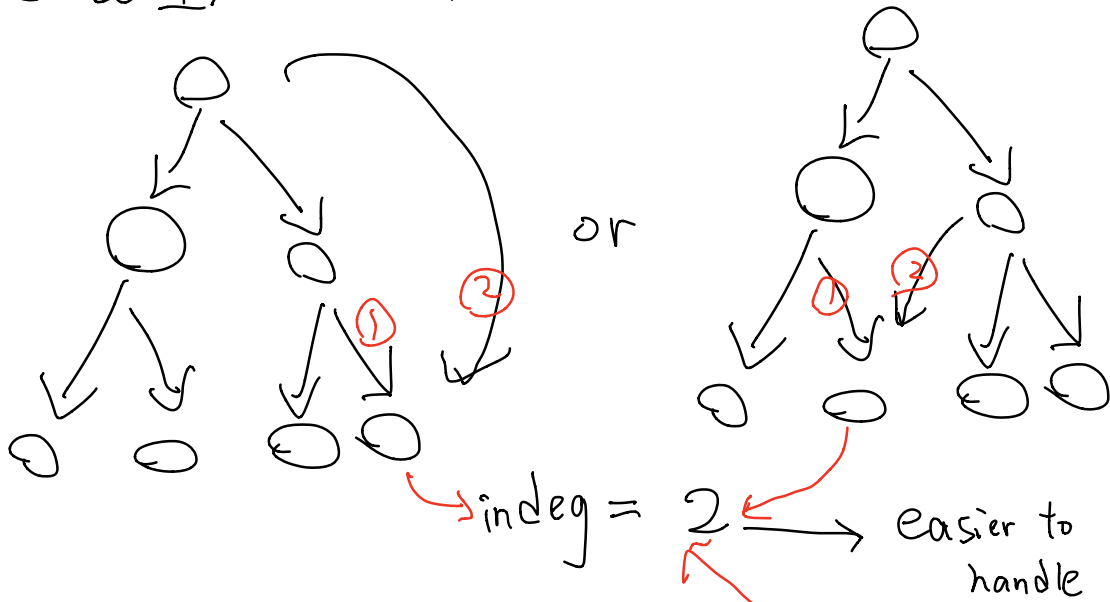
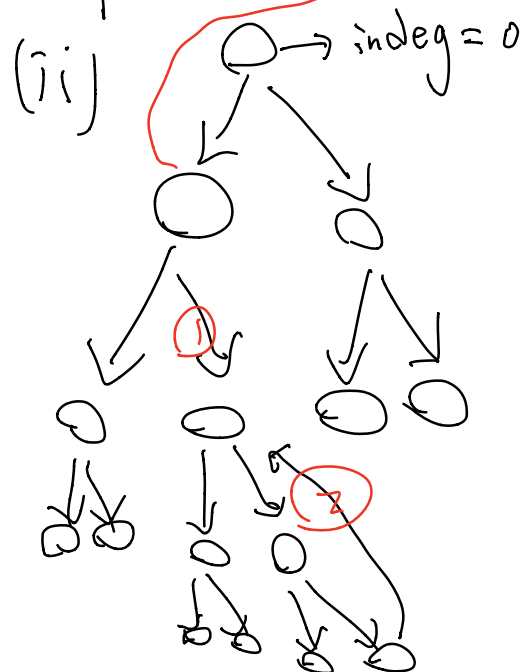
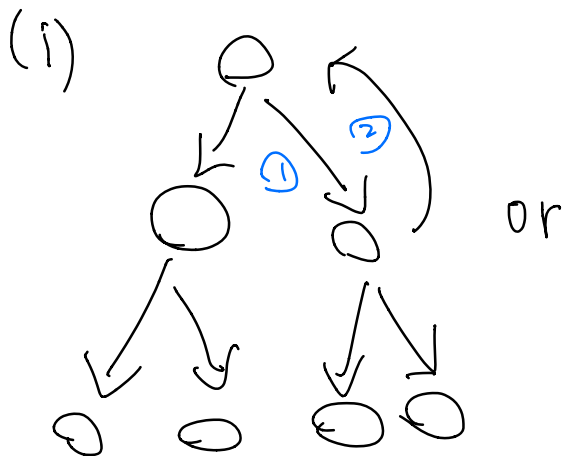


Case 1: \downarrow : parent points to a descendant



Case 2: \uparrow a descendant points to a parent



Either ① or ②
can be removed

(i)

```

graph TD
    Root(( )) --> L1L(( ))
    Root --> L1R(( ))
    L1L --> L2L1(( ))
    L1L --> L2L2(( ))
    L1R --> L2R1(( ))
    L1R --> L2R2(( ))
    L2L1 --> L3L1(( ))
    L2L1 --> L3L2(( ))
    L2L2 --> L3L3(( ))
    L2L2 --> L3L4(( ))
    L2R1 --> L3R1(( ))
    L2R1 --> L3R2(( ))
    L2R2 --> L3R3(( ))
    L2R2 --> L3R4(( ))
  
```

The diagram shows a binary tree structure. The root node is at the top. It has two children. The left child has two children of its own. The right child has two children. The tree is not a full binary tree. Some nodes are circled in blue and labeled with numbers: the root's left child is labeled '2', the root's right child is labeled '1', and the left child of the root's left child is labeled '3'. Arrows indicate the flow from parent to child.

Either ①, ②, ③ can be removed

Algorithm:-

For Case 1, Case 2(ii),

we only have ①, ② to choose.

And because Case 2 (ii), we don't know which edge should we remove.

⇒ Just try removing (1) and do an union find,
if we don't see a loop ⇒ remove (1)
else ⇒ remove (2)

For case 2 (i),
every edge on the loop can be
removed

⇒ Do an union find

⇒ return the edge that causes a loop

#