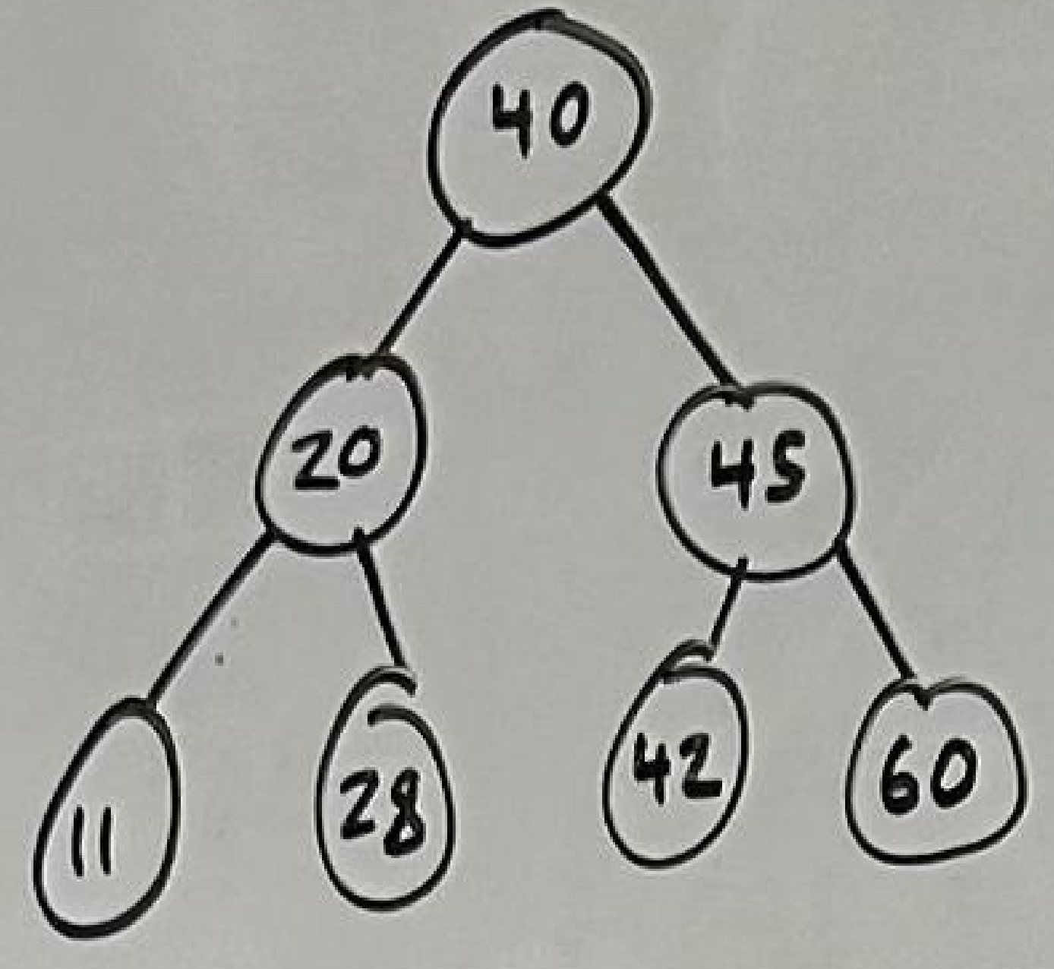
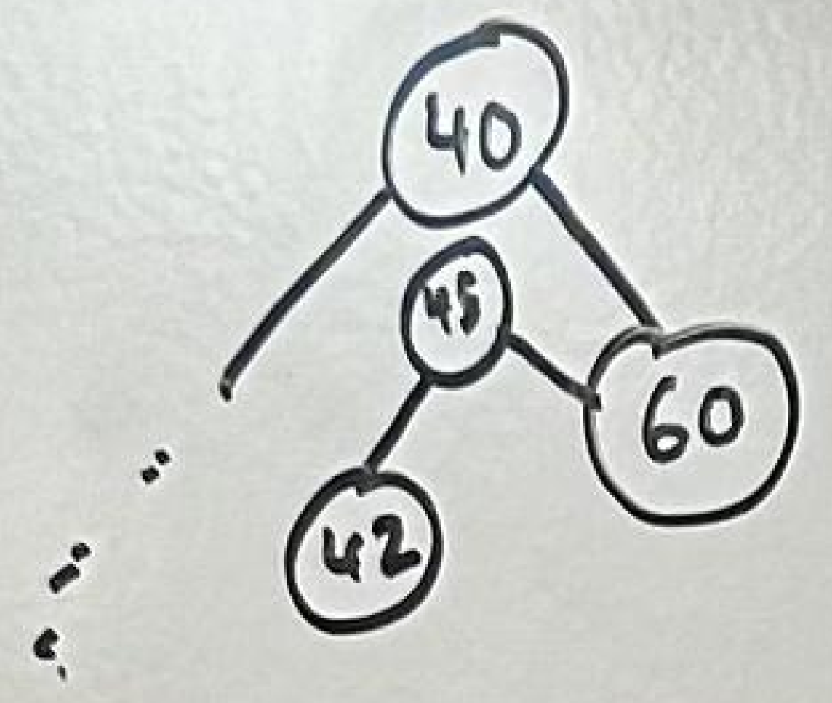
```
Transplant(T, u, v)
    if u.p == NIL
        T.root = v
    else if u = u.p.left
        u.p.left = v
    else
        u.p.right = v
    if v ≠ Nil
        v.p = u.p
```

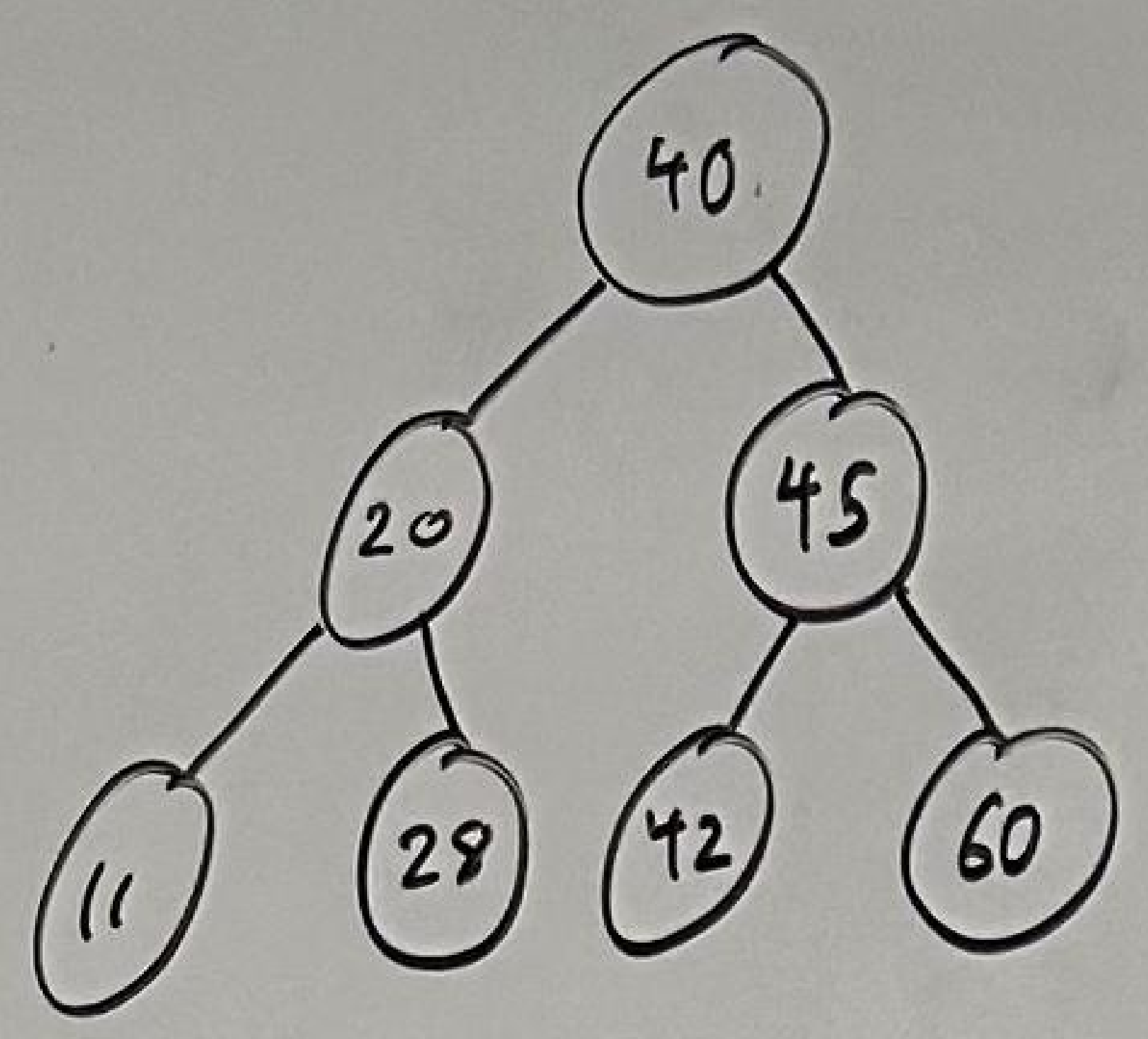
```
Tree-Delete(T, z)
    if z.left == NiL
        Transplant(T, z, z.right)
    else if z.right == Nil
        Transplant(T, z, z.left)
    else
        y = Tree-Minimum(z.right)
        if y ≠ z.right
            Transplant(T, y, y.right)
            y.right = z.right
            y.right.p = y
        Transplant(T, z, y)
        y.left = z.left
        y.left.p = y
```

Replaces sub-tree rooted at u with subtree rooted at v

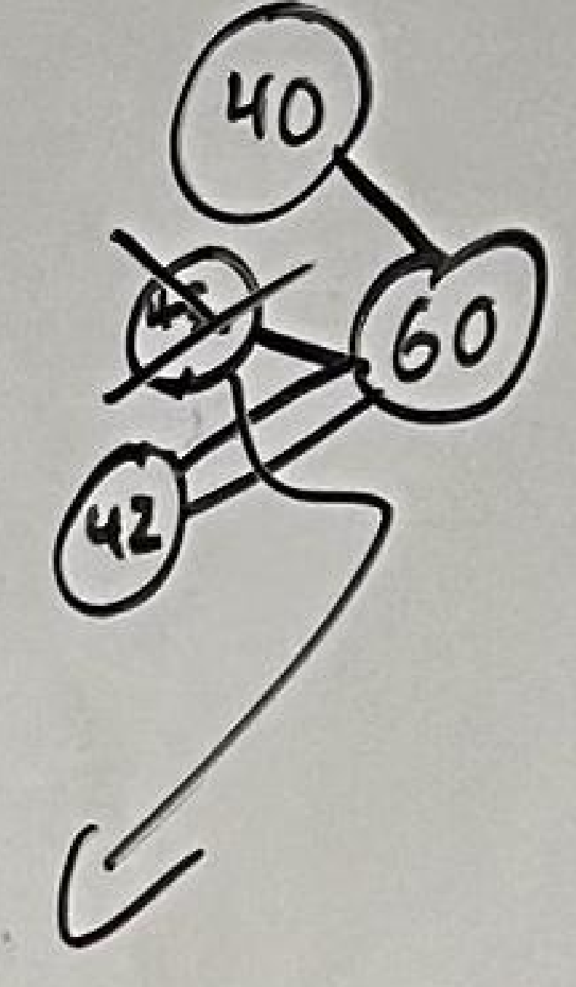


Transplant(T, 45, 60)
U = 45, v = 60

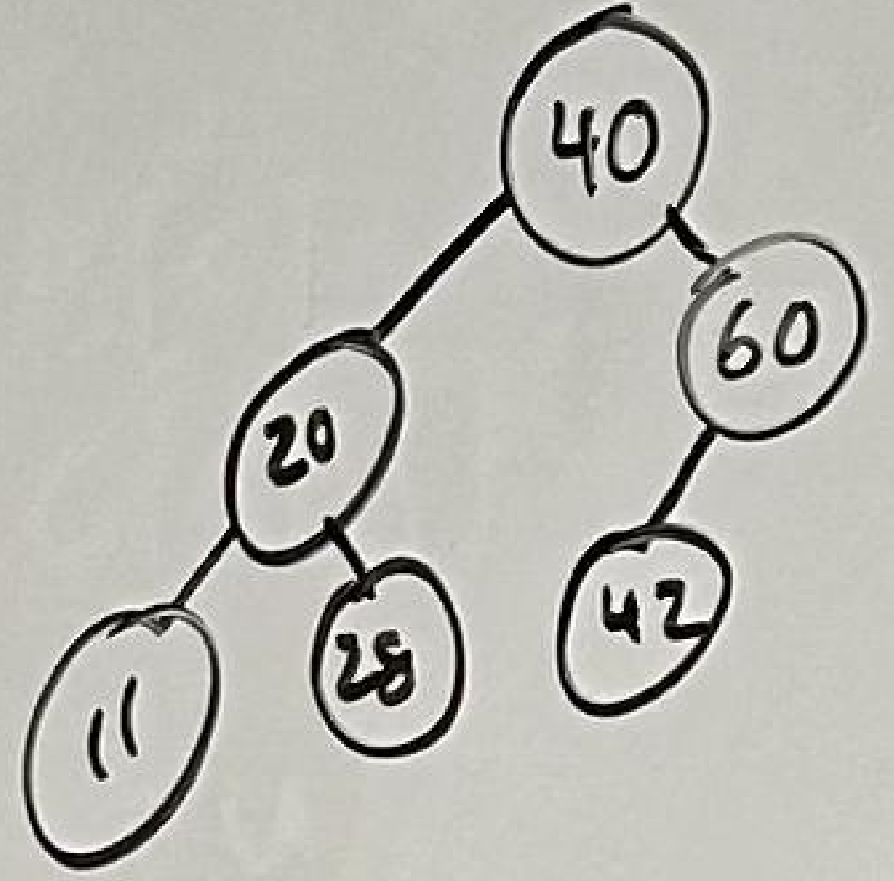


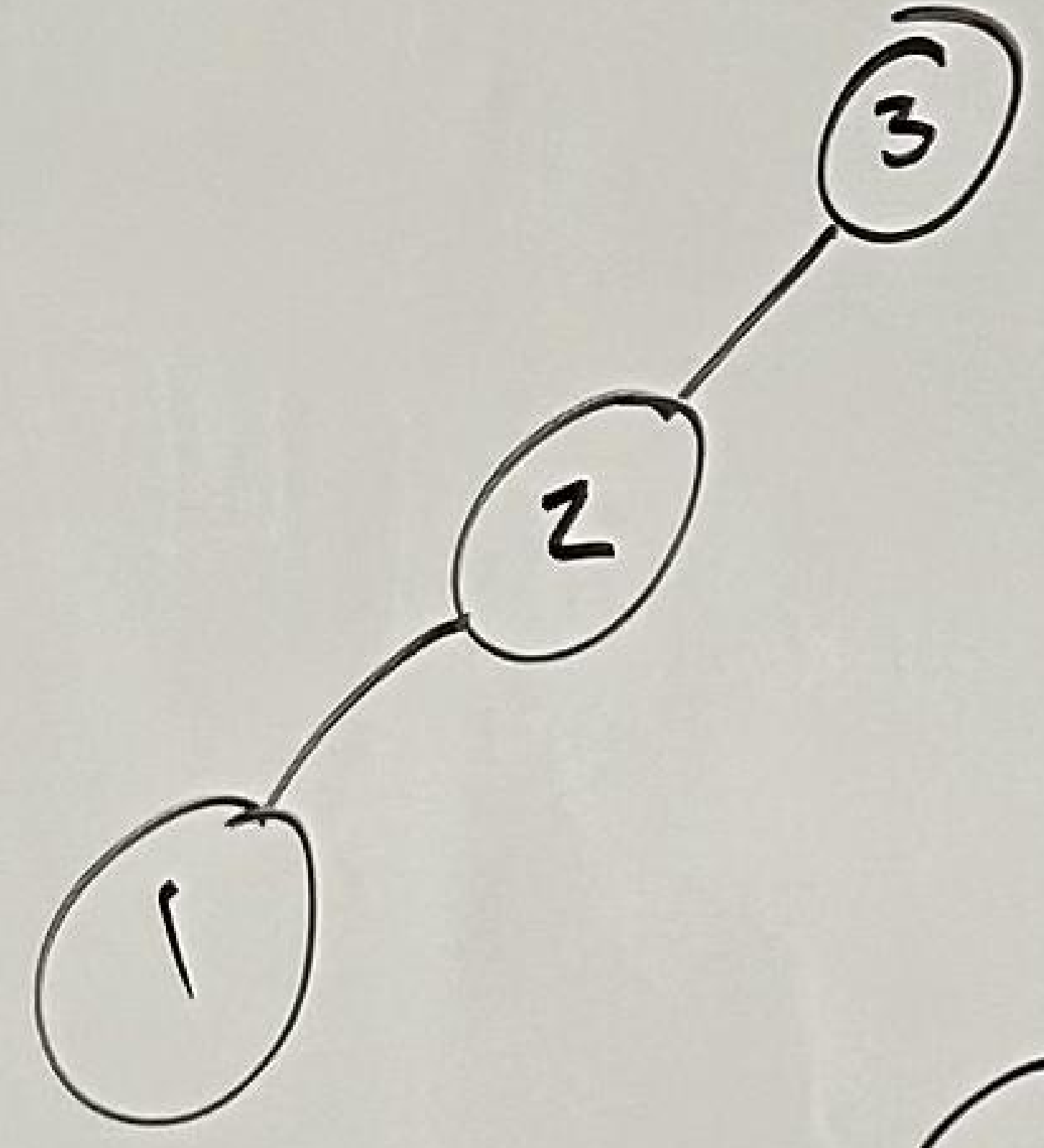


Tree-Delete(T, 45)
y = 60
Transplant(T, 45, 60)

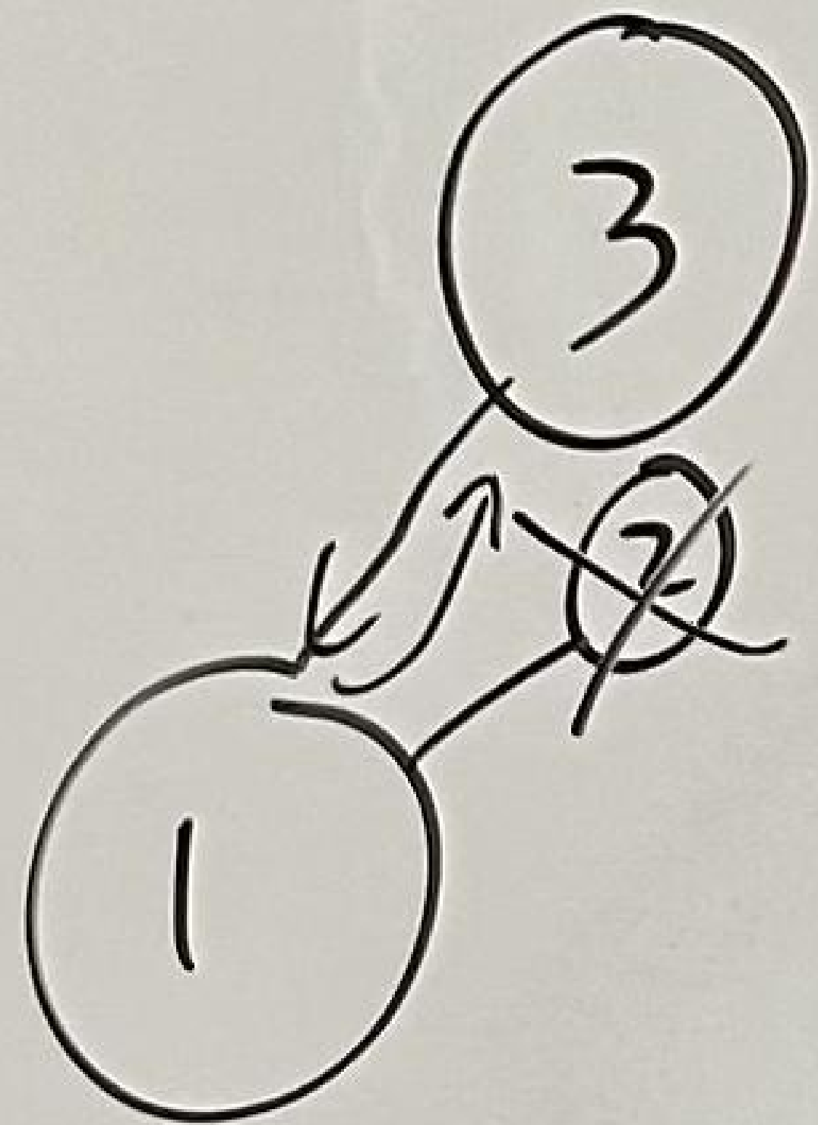


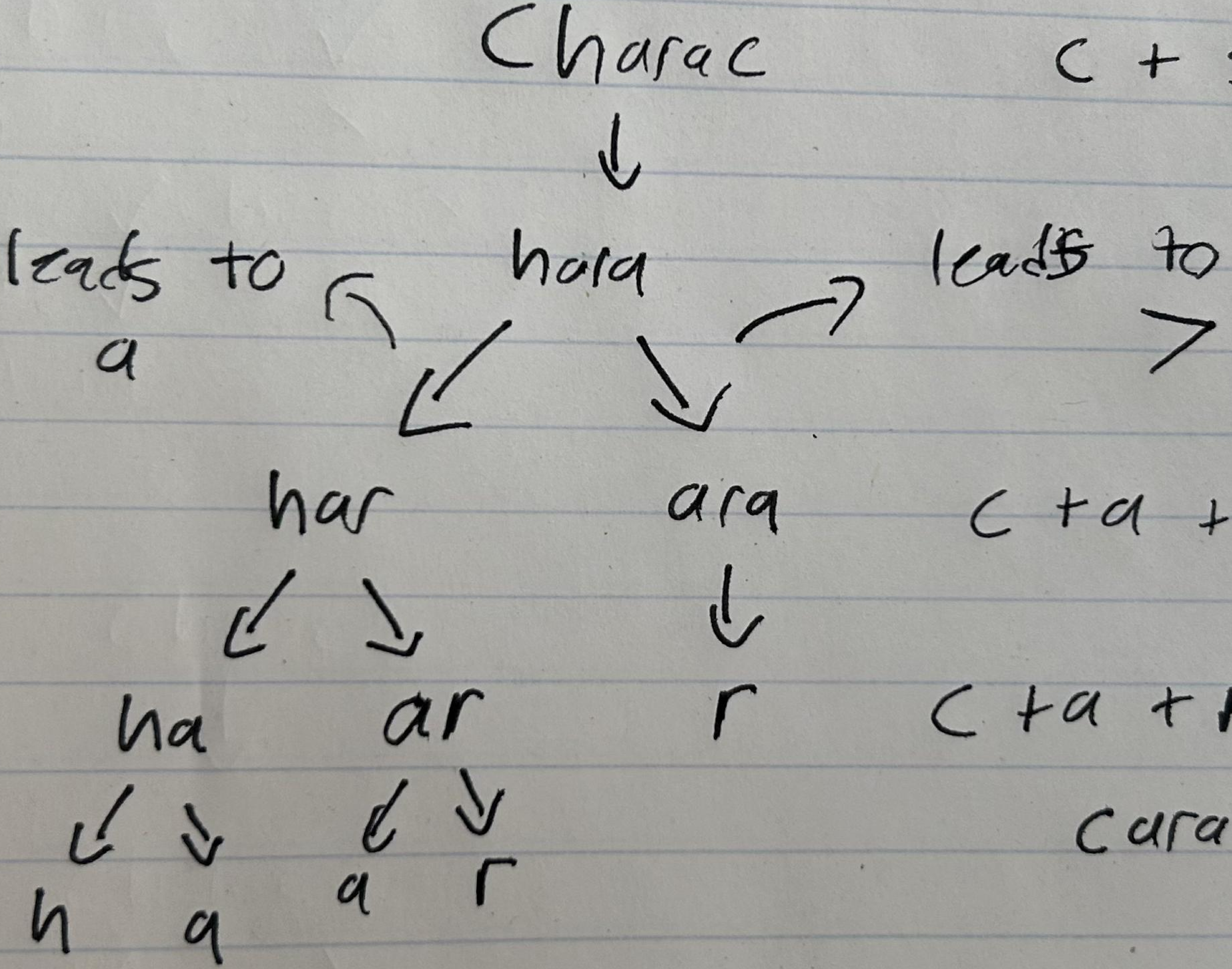
Nullify





TD(T, 2)
TP(T, 2, 1)





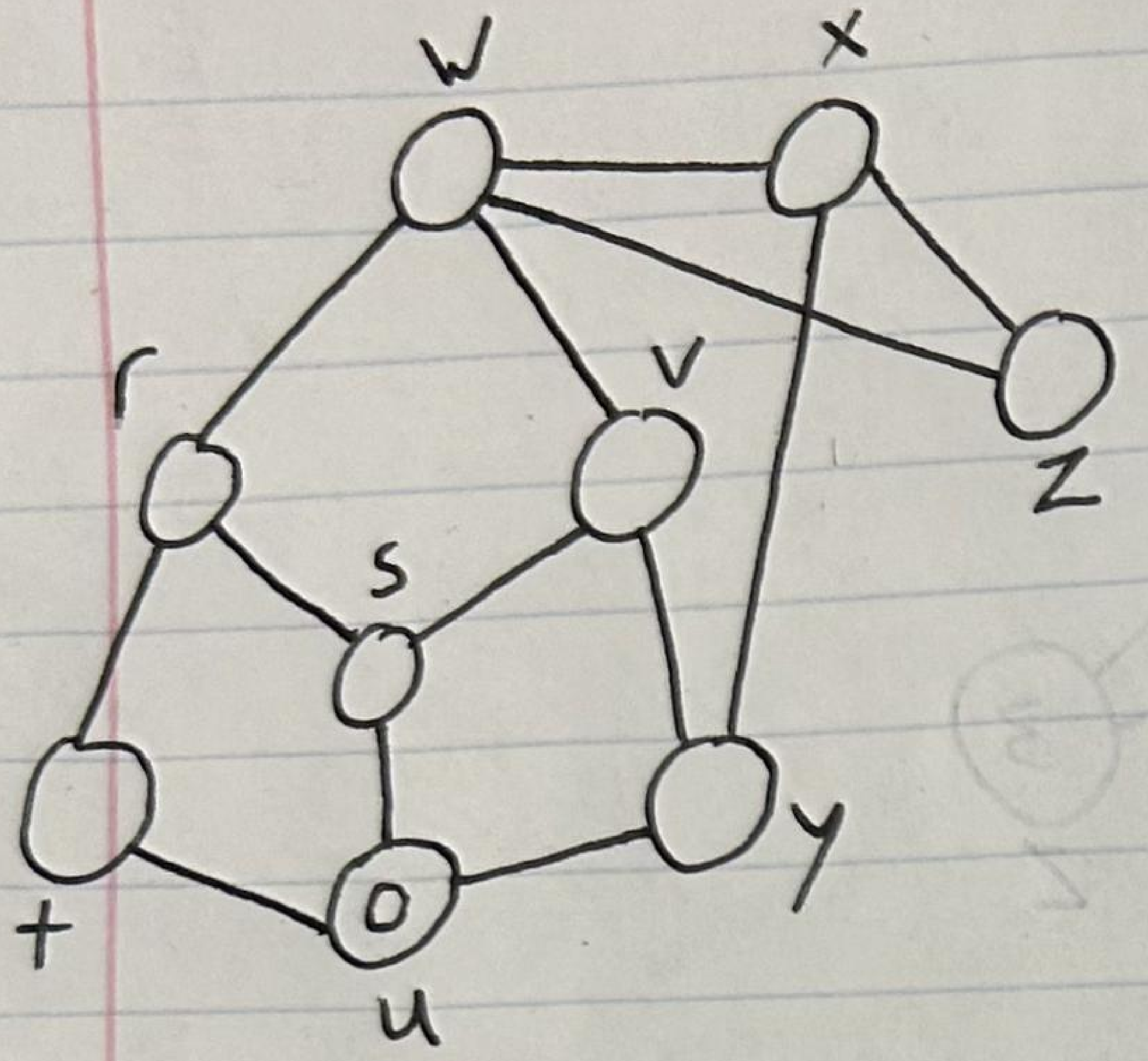
c + sub + c

c + a + sub + a + c

c + a + r + a + c

carac

Q = { }



Q = {u}

u.adj = [t, s, y]

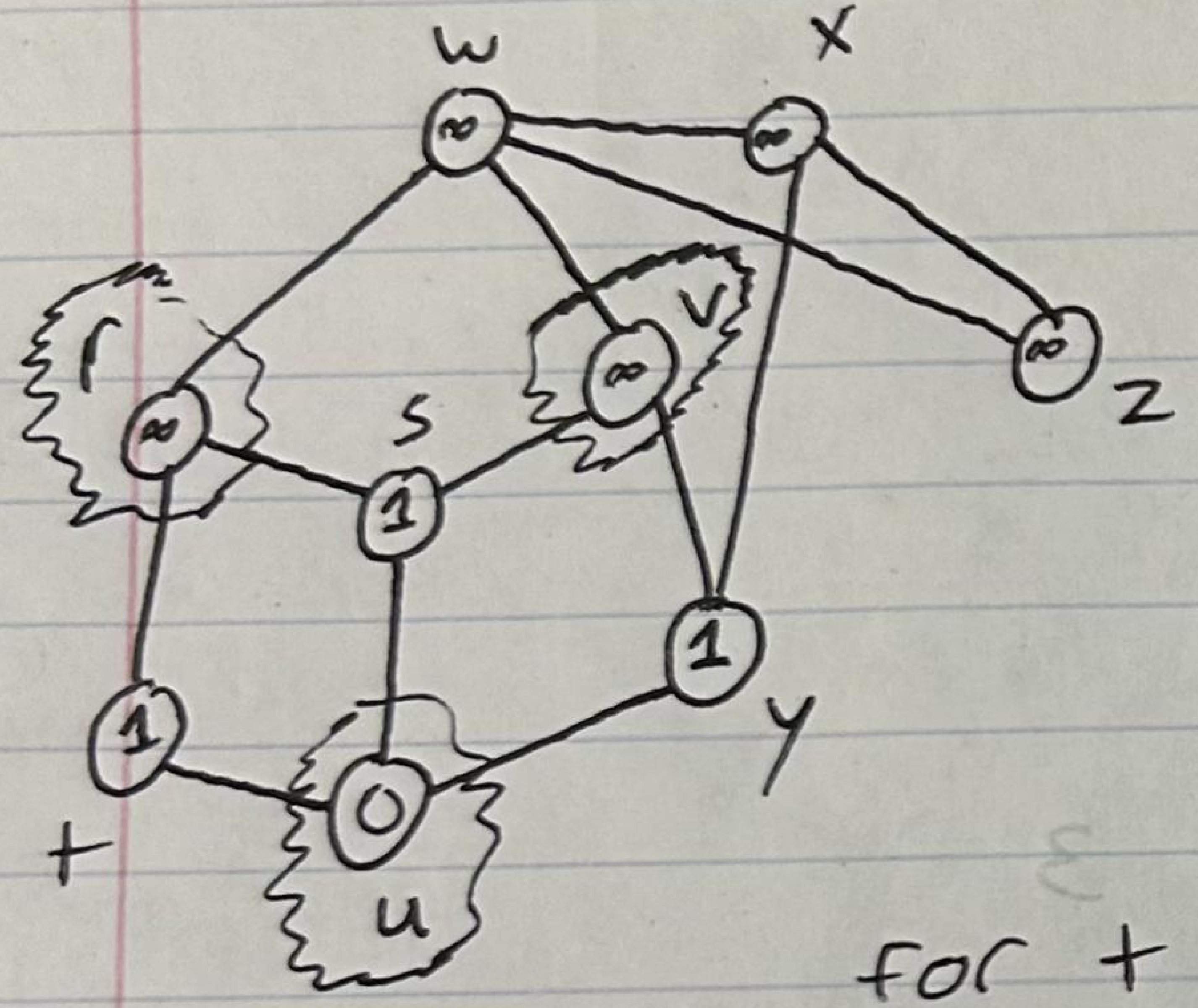
Set all depths to 1

Mark as visited

enque in alphabetical order

Set prev accordingly

Q = {s, t, y)



First visit s

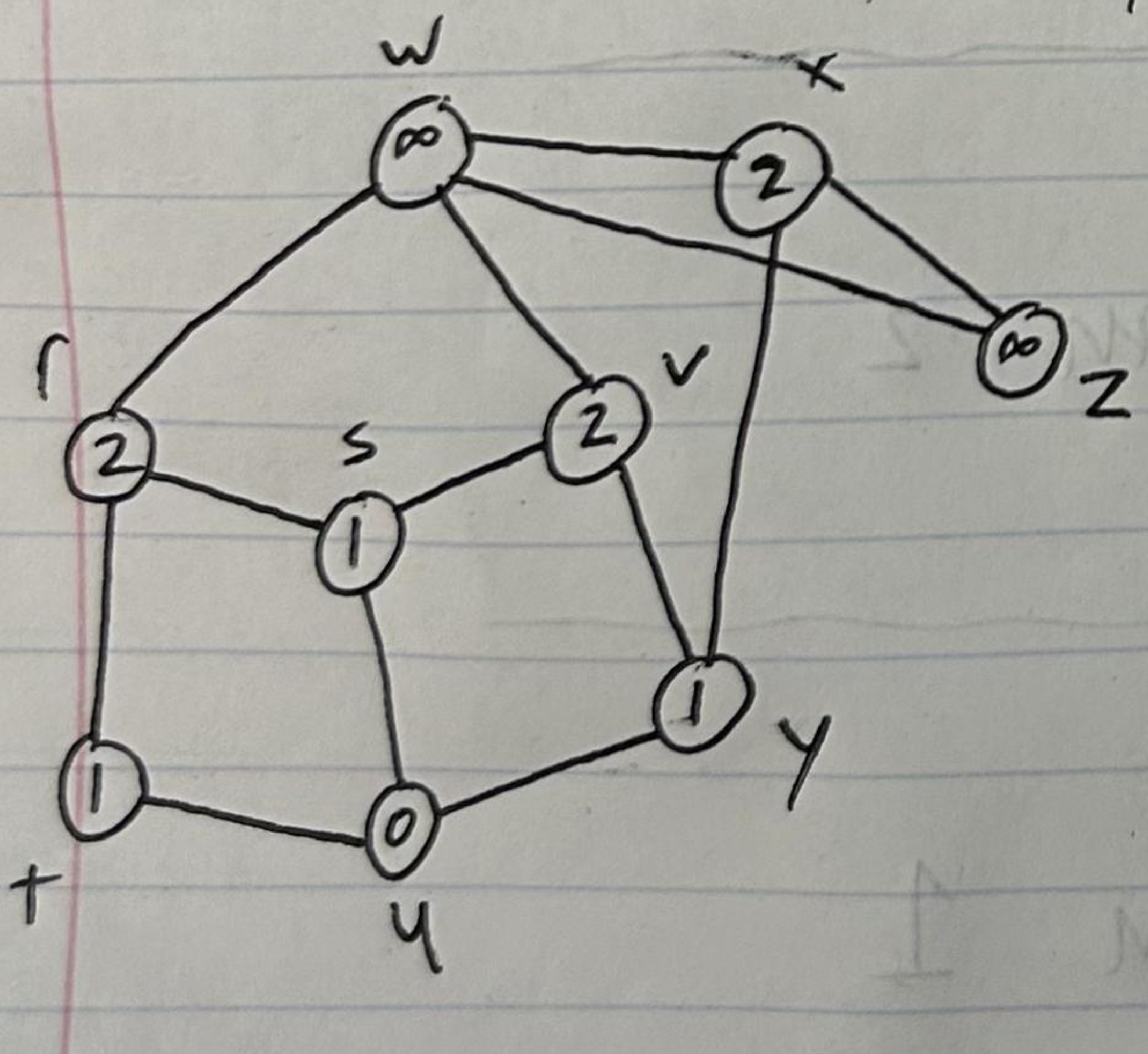
s.adj = [r, u, v] but u visited

So focus on [r, v]

Set depths to s.depth (1) + 1 = 2

and mark as visited

carry out same process for t / y, leading to x having depth 2

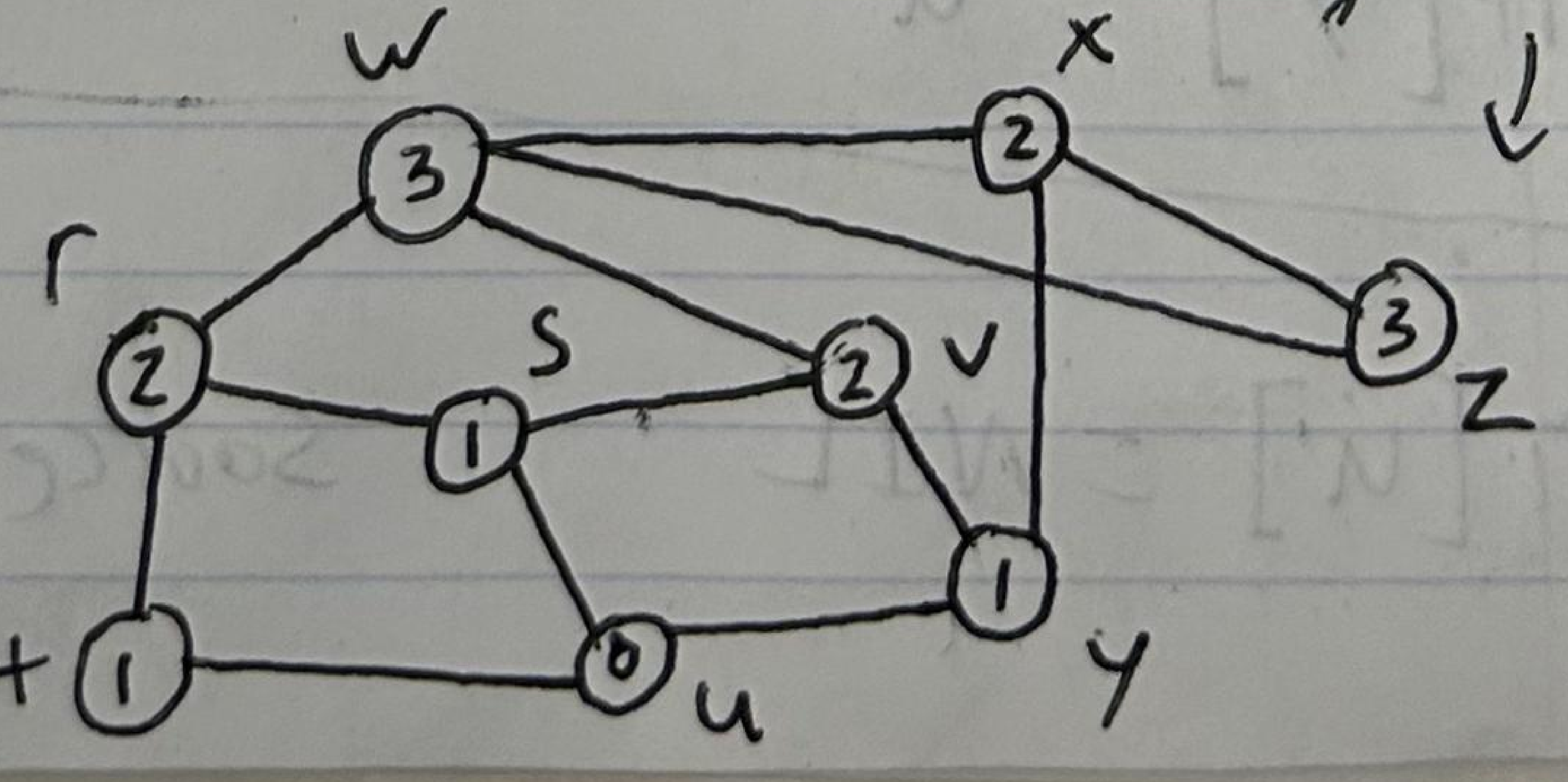


Q = {x}

x.adj = [w, z]

increment depths

final Q = {w, z}, but no unvisited neighbors


w
x
3
2
3
z
r
2
s
2
v
1
1
t
0
u
1
y