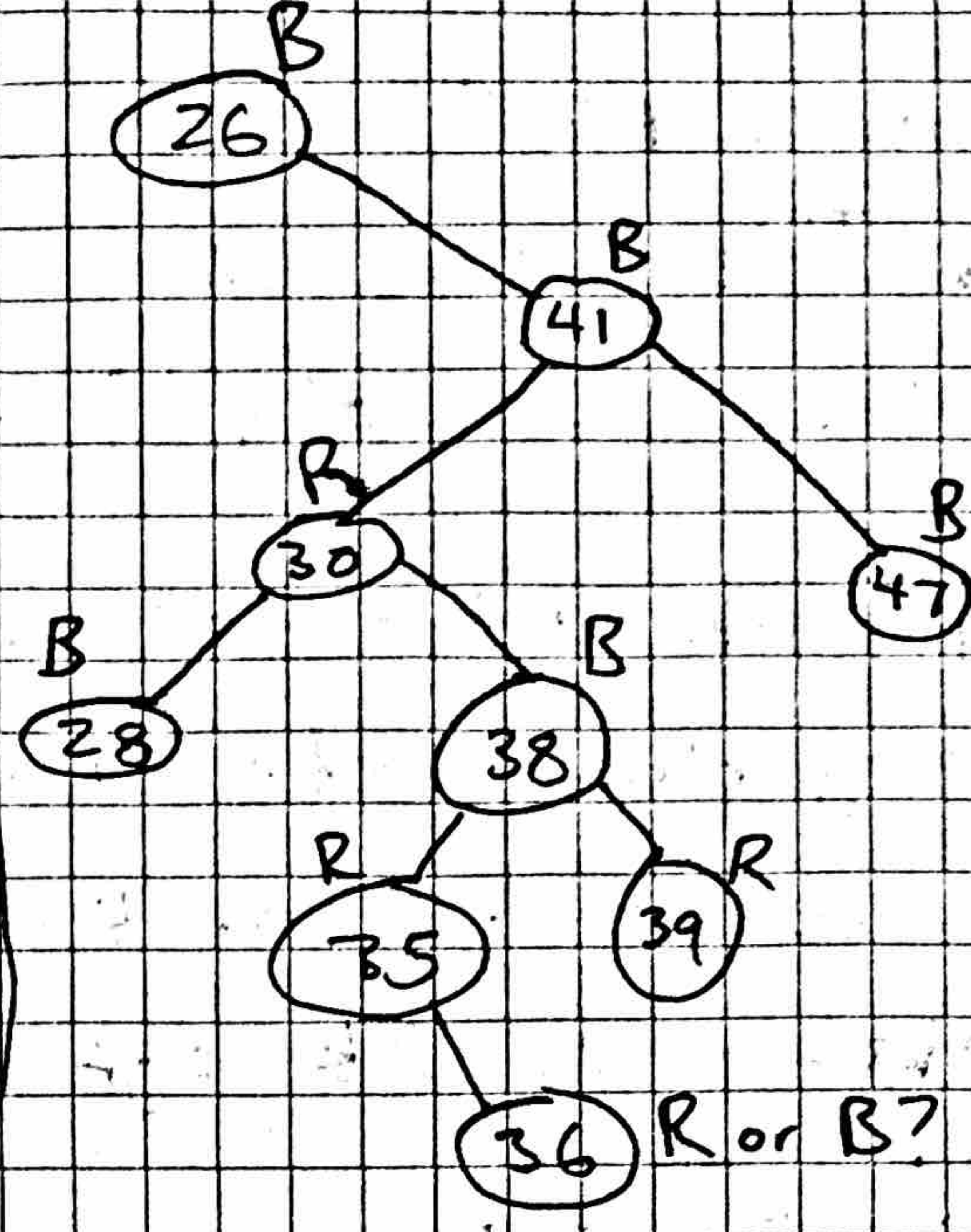
13.1-2)



If 36 is black and inserted

28 bh = 3

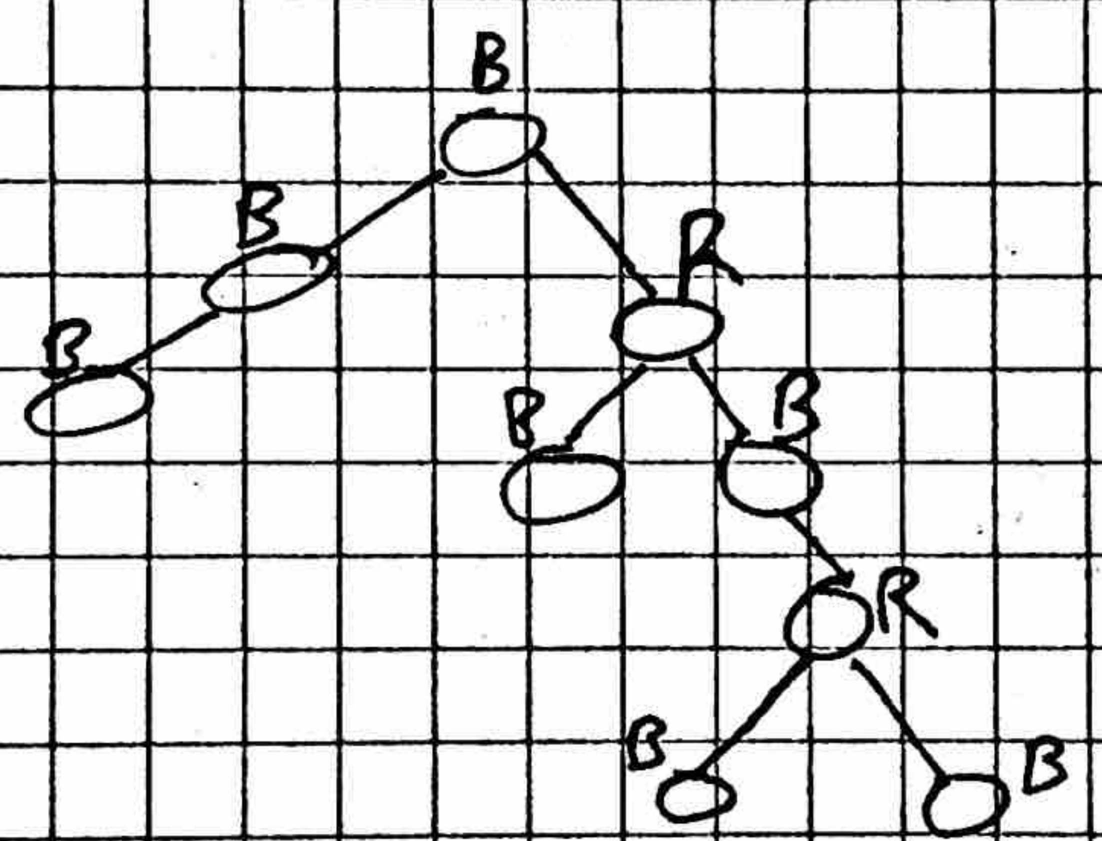
47 bh = 3

if 36 Black then bh = 4

breaks Prop. 5

If 36 is red then that breaks prop. 4 for node 35. It doesn't have 2 children so 35 should not be red.

13.1-5)



the longest simple path can be twice as long as the shortest simple path.

Shortest simple path occurs when there is only B nodes on that path.

Longest simple path occurs when every B node has a R child to maximize the red children on a path.

Since a red parent can't have a red child then the alternating Red and Black tree scheme would produce the longest simple path which is twice as long as an only black node simple path

13.1-6) smallest possible number of nodes is 2^k-1. This occurs when there is a complete binary tree with only Black nodes

largest possible number of nodes occurs when every black node has a red child in a complete tree. Since k is black height then the total height is 2k so the largest possible # of internal nodes is $2^{2k}-1$.

13.2-1) Right-Rotate(T, x)

```
y = x.left
x.left = y.right
if y.right ≠ T.nil
    y.right.p = x
y.p = x.p
if x.p == T.nil
   T.root = y
else if x == x.p.right
     x.p.right = y
else x.p.left = y
y.right = x
x.p = y
```

13.2-4)

If take an n-node binary search tree and call right rotate on the root node until the roots left child is NIL. This results in an right-going chain. Since rotate takes O(1) time and n-rotations occured then it takes O(n) rotations to transform the tree. If we take this right going chain tree and call left rotate on the root until root.right is NIL then we would have to rotate n-1 nodes which is O(n) rotations.

13.3-1) if z's color is black then that messes up Property 5 of r-b trees. When a red node is inserted the black height doesn't change and RB-Insert-Fixup is used to rebalance the tree and re color the tree nodes based on their properties. By choosing to use red nodes instead of black nodes Property 4 gets broken which is more easily fixed than using black nodes and trying to fix Property 5 every insertion call. Property 5 is broken every time a black node is inserted while Property 4 isn't necessarily broken every time a red node is inserted.

22.1-1) The in degree of a vertex is the # of edges having that vertex as terminus. It's outdegree is the number of edges having that vertex as origin.

a - origin

b - terminus

Outdegree takes $O(E)$ time to compute the out degree of every vertex. Following all the edges will give you the outdegree

indegree takes $O(E+V)$ time to compute every vertex. Following every vertex to check for edges will find the indegree of every vertex

22.1-3) adjacency-list

1) make a temp adjacency-list that's the same size as G
2) iterate V-list
3) iterate U-list
4) store G(V,U) in temp(U,V)
5) return temp

to iterate through a directed graph with an adjacency-list represtation takes $\boxed{O(V+E)}$

adjacency-matrix

1) Make a temp matrix of size G
2) iterater columns
3) iterate rows
4) store G(columns, rows) in temp(rows, columns)
5) return temp

to iterate through a adjacency-matrix represtation of a directed graph takes $\boxed{O(V*V)}$

Derrick DeBose

22.2-3) Single bit 0 or 1, Black or white.

lines 5 and 14 aren't very useful for BFS because line 13 only checks to see if the vertex's color is white. Without having the color gray the vertex will still get enqueued if the color is white. The loop invariant still holds for each vertex getting enqueued and dequeued exactly one time.

22.2-4)

```
BFS(G, s)
for each x ∈ V-{s}
    color[x] = white
    d[x] = ∞
    π[x] = NIL
color[s] = gray
d[s] = 0
π[s] = NIL
Q = ∅
Enqueue(Q, s)
while Q ≠ ∅
    x = Dequeue(Q)
    for y = 1 to |V|
        if color(y) == white
            color[y] = Gray
            d[y] = d[x] + 1
            π[y] = x
            Enqueue(Q, y)
    color[x] = black
```