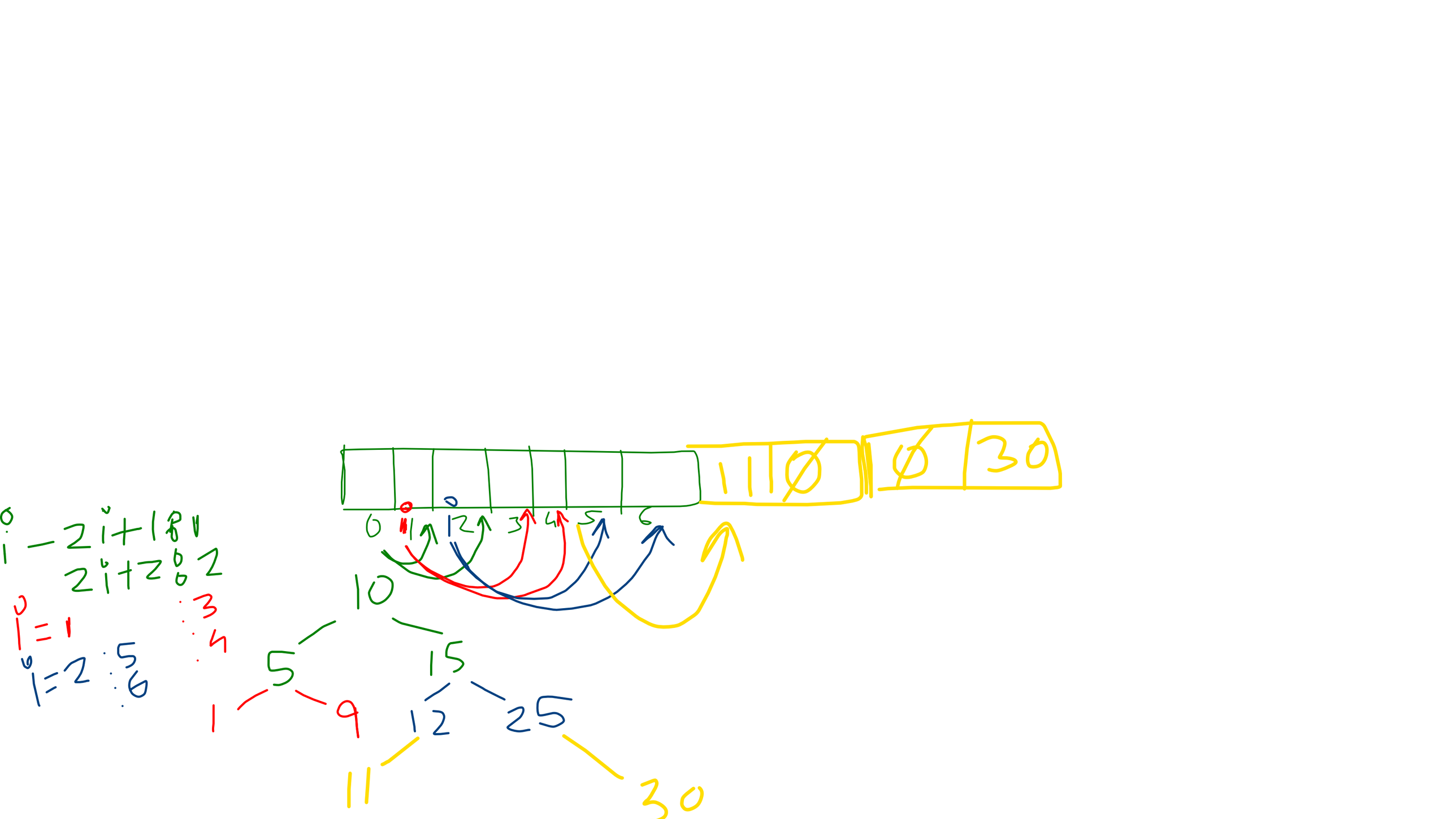
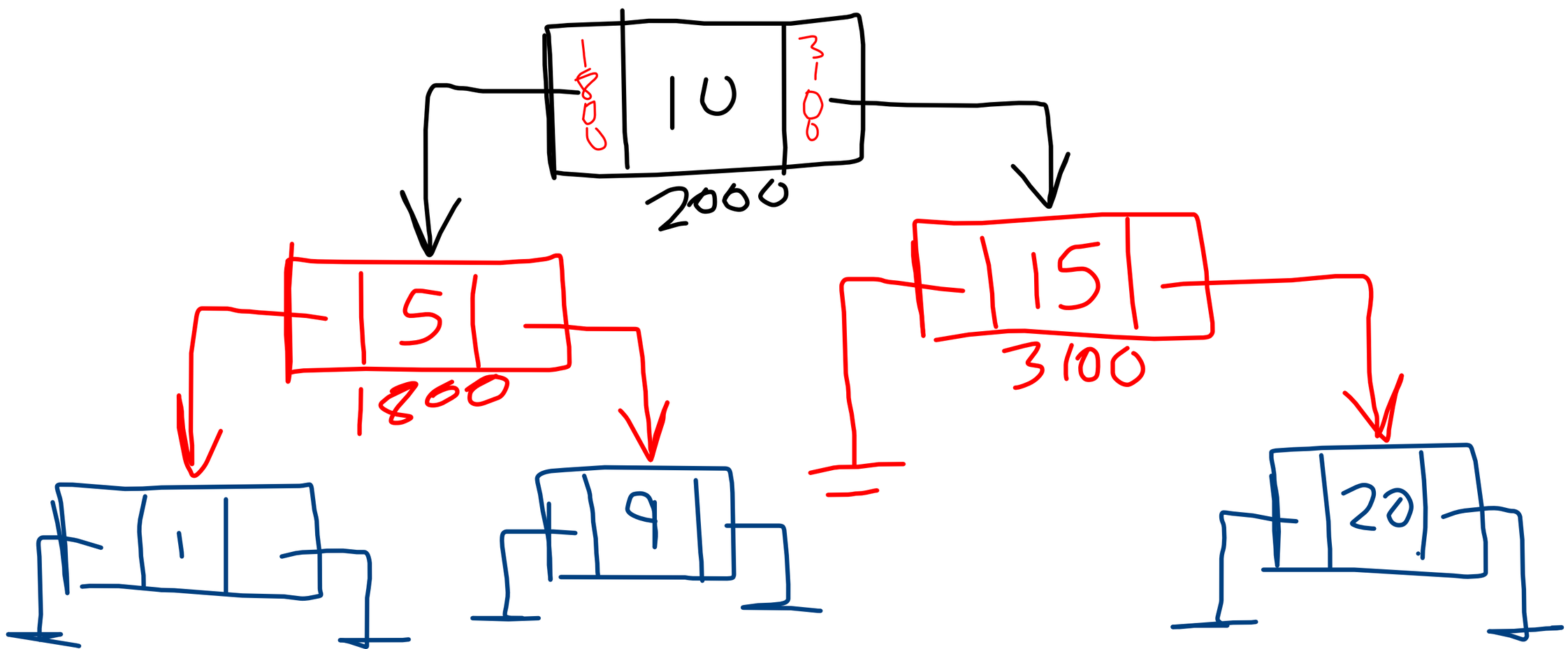
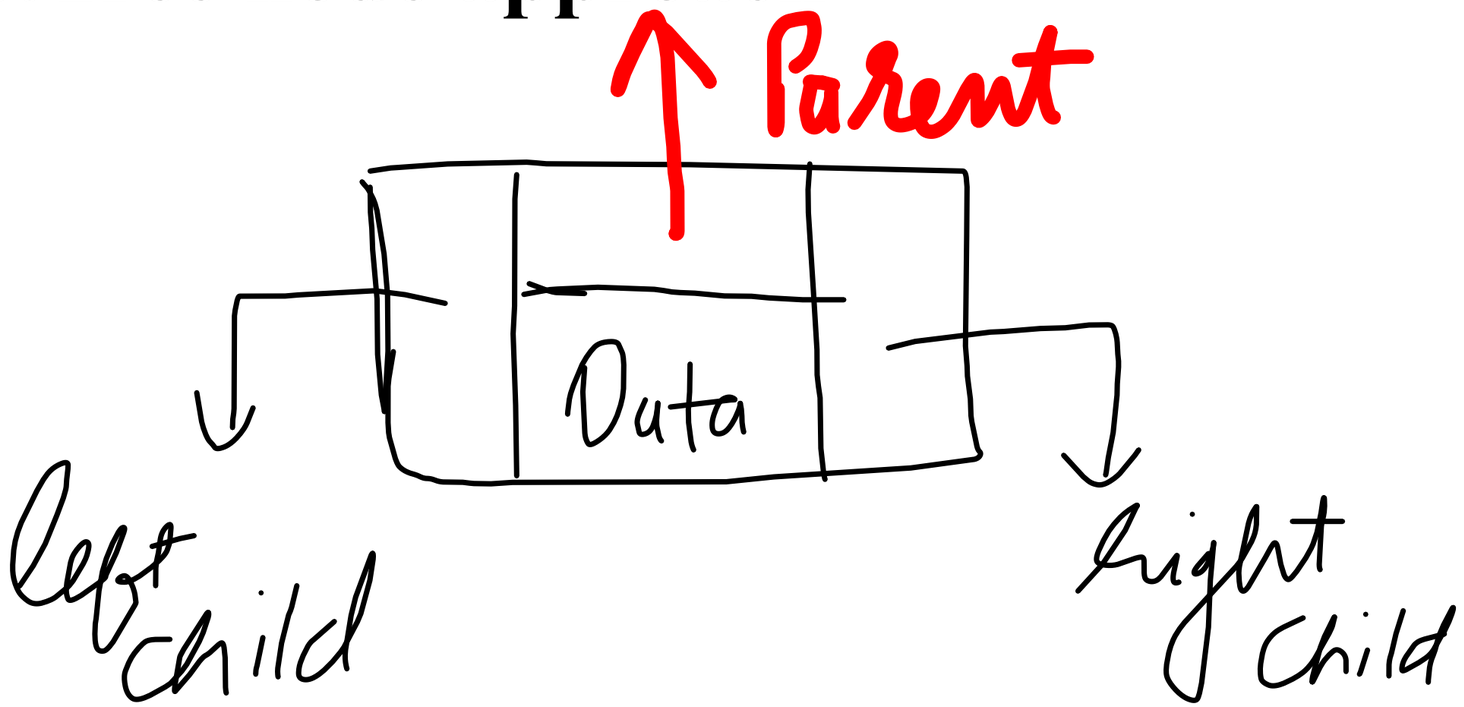

0
1 - 2i+1 || 2i+2 || 2
1 = 1
1 = 2
3
4
5
6
0 1 2 3 4 5 6
10
5
15
1
9
12
25
11
30
1 1 ∅
∅ 30

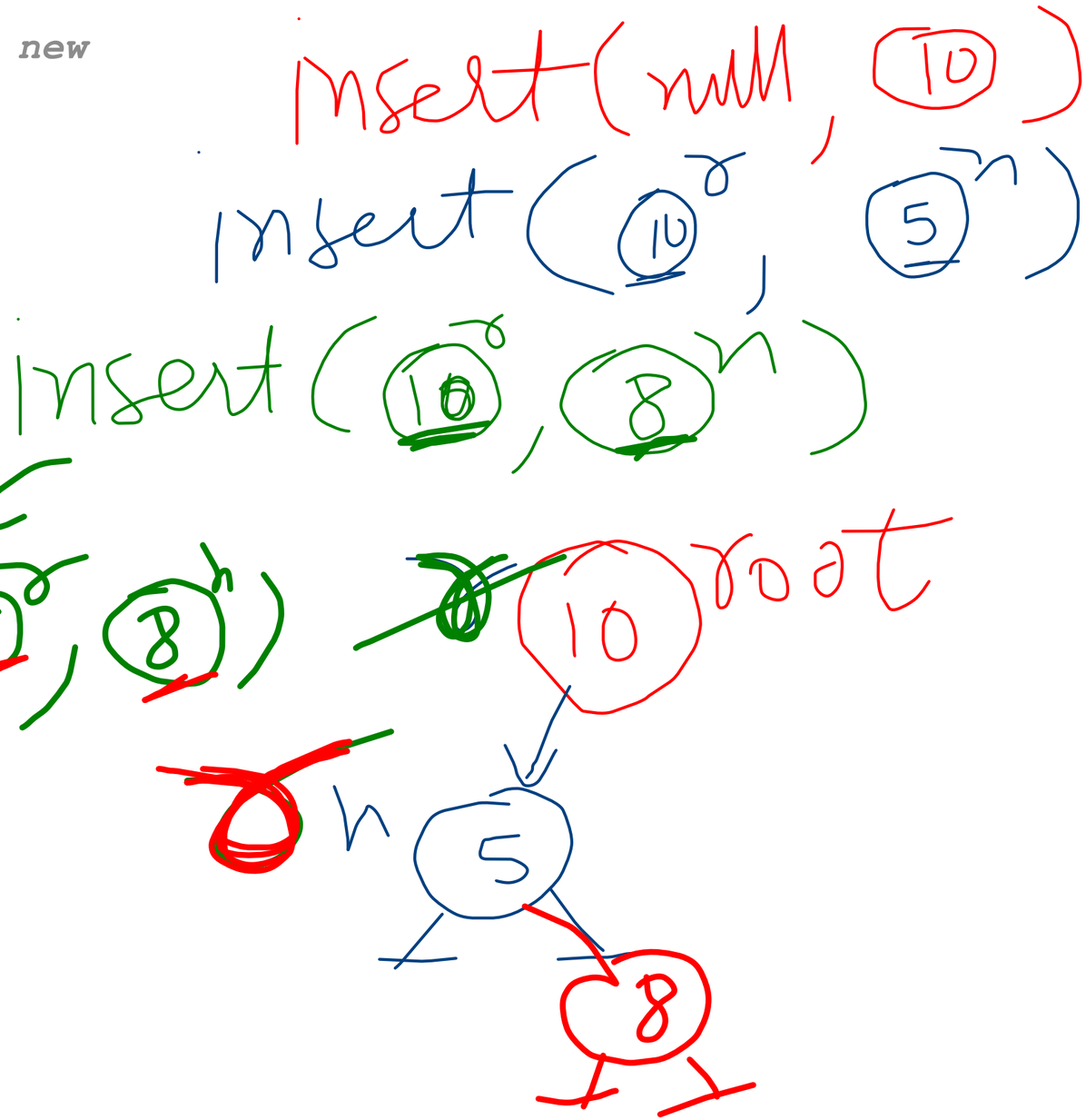

10
1800
3100
2000
5
1800
15
3100
1
9
20

3.Dynamic:Tree node approach

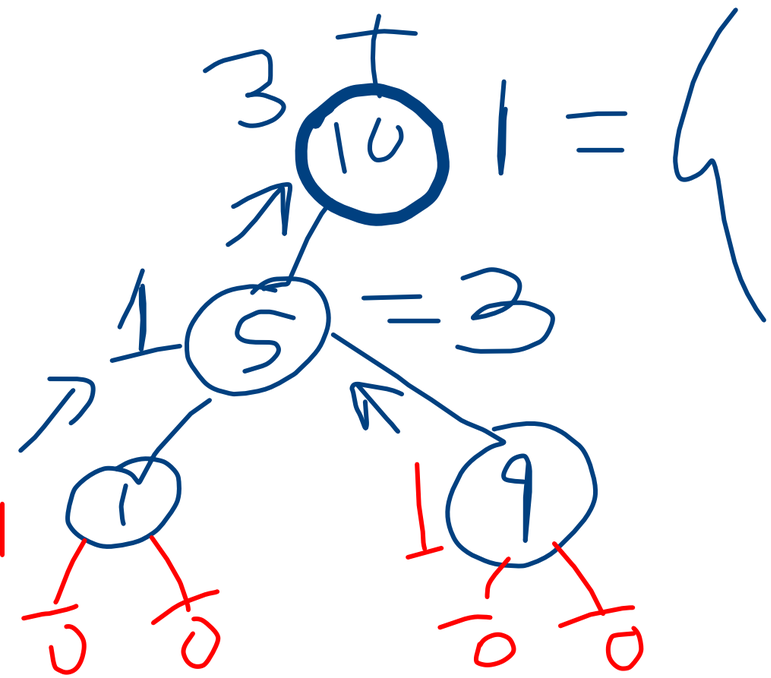


+ve:To use recursion, can use sequential normal coding for traversal of tree.
-ve:Too complex to implement and use due to multi-pointer system.

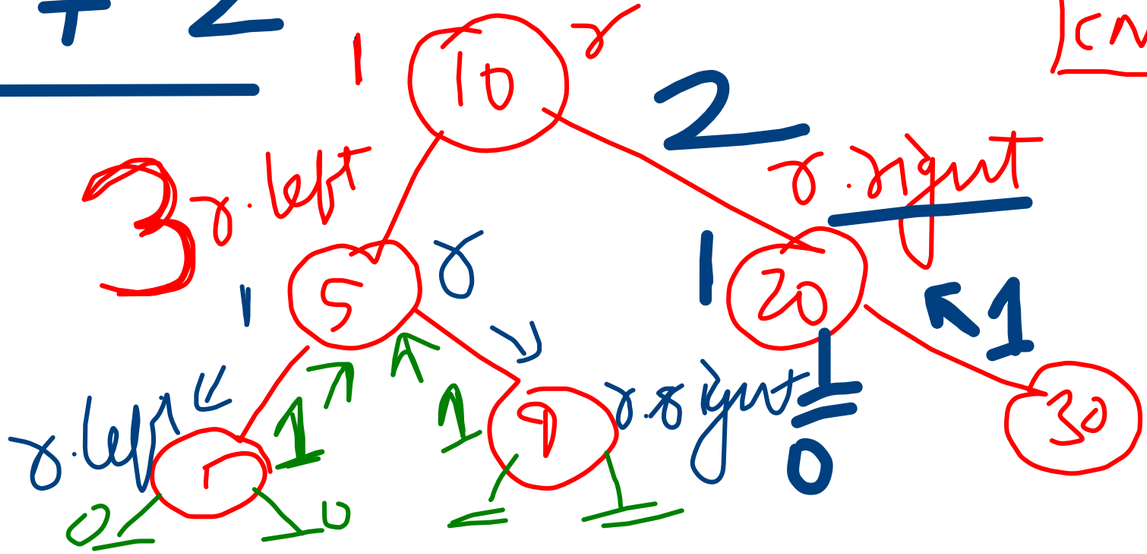
```
void insert_node(Dnode r,Dnode n)//root---->r     new
node---->n
{
    if (root == null)//root not init
        root = n;
    else {
        if (n.data < r.data) {
            if (r.left == null) {
                r.left = n;
                System.out.println("\n"+n.data+"
inserted");
            }
            else
                insert_node(r.left, n);

        }
        else//right
        {
            if (r.right == null) {
                r.right = n;
                System.out.println("\n"+n.data+"
inserted");
            }
            else
                insert_node(r.right, n);
        }
    }
}
```

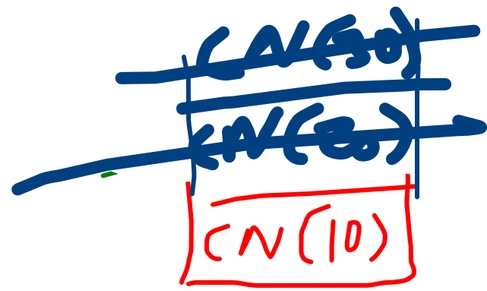


```
static int count_nodes(Dnode r)
{
    if(r==null)
        return 0;
    return 1+count_nodes(r.left)+count_nodes(r.right);
}
```

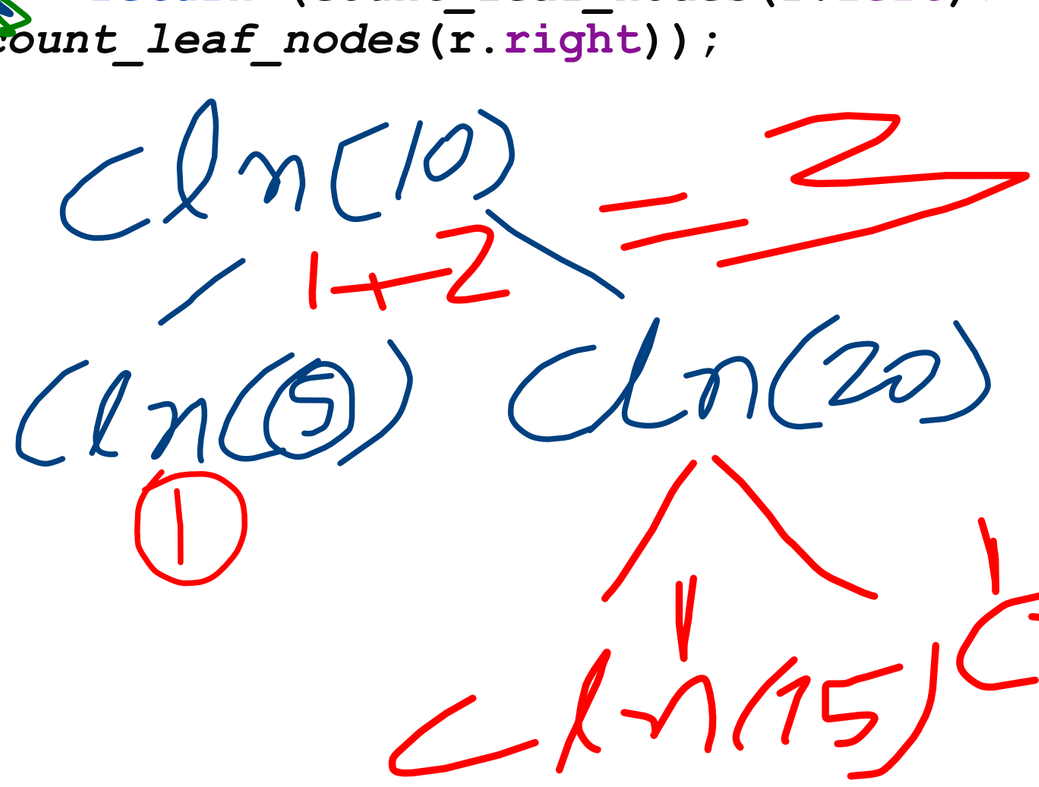


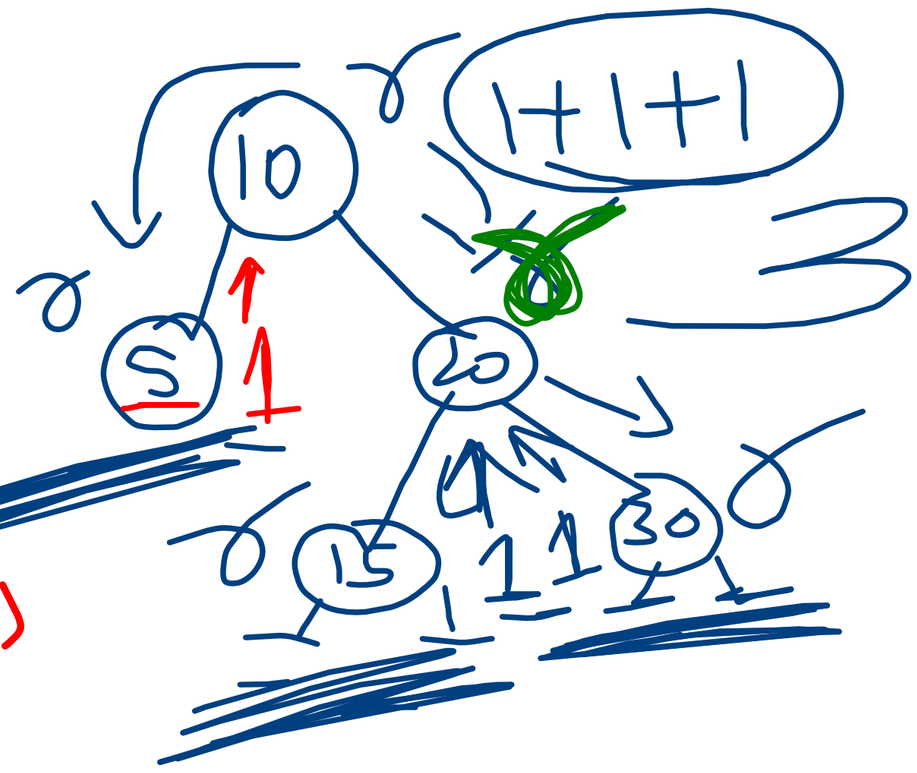
$$1 + \frac{3 + 2}{6}$$





```
static int count_leaf_nodes(Dnode r)
{
    if(r==null)
        return 0;
    if (r.left==null && r.right==null)
        return 1;
    return (count_leaf_nodes(r.left)+
count_leaf_nodes(r.right));
}
```





5+10+15+30+20

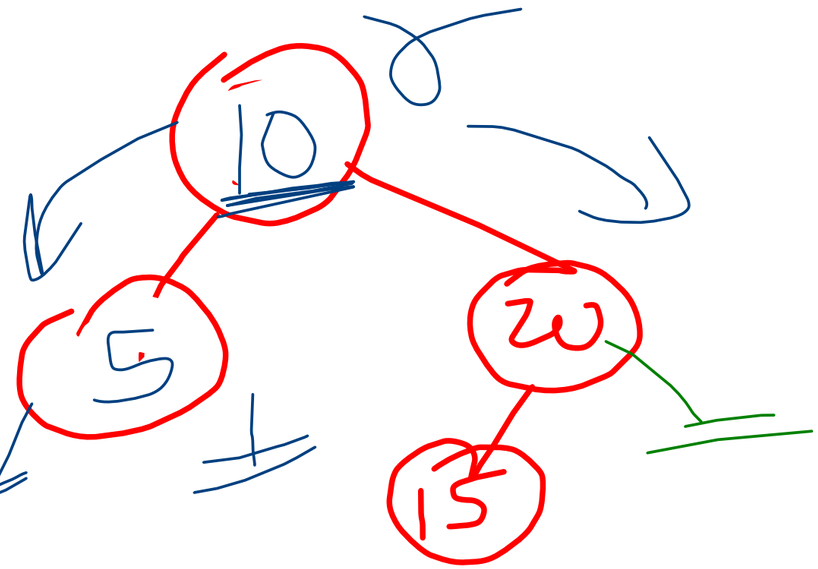
```
static int sum_of_elements(Dnode r)//return total of all
element values
{
    if(r==null)
        return 0;
    return r.data+sum_of_elements(r.left)
+sum_of_elements(r.right);
}
```

S(10)

10+S(5)+S(20)

↓5

5+0+0 | 25



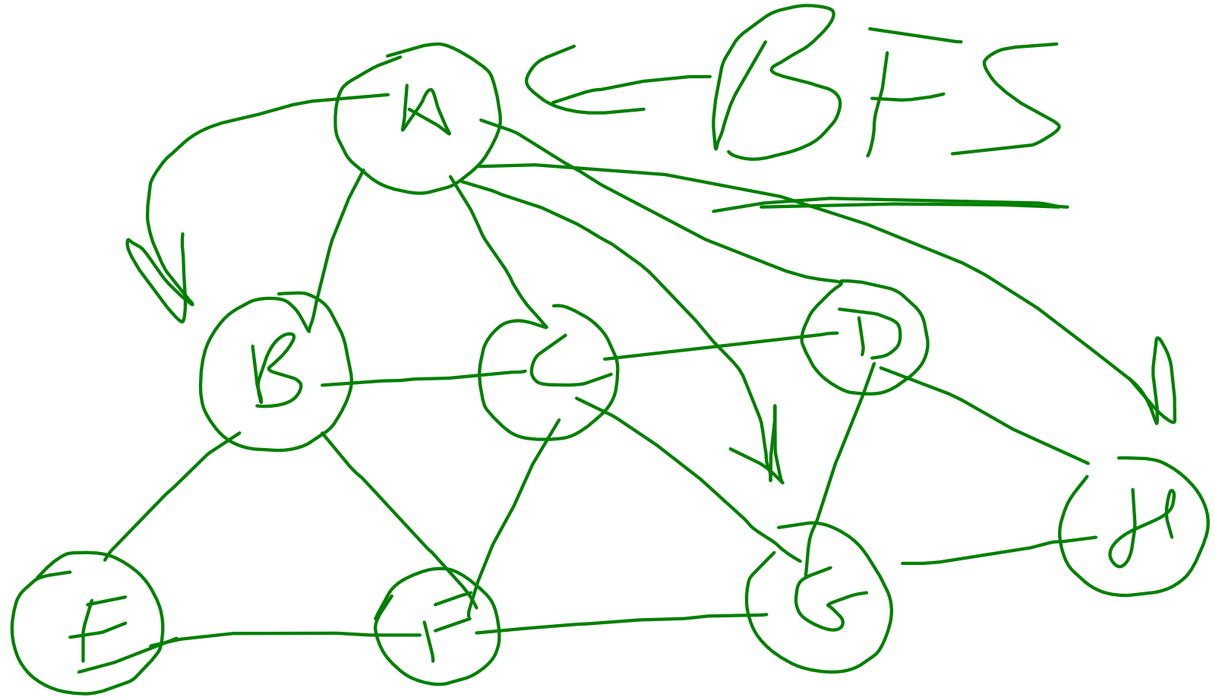
20+S(15)+0

10+5+25

40

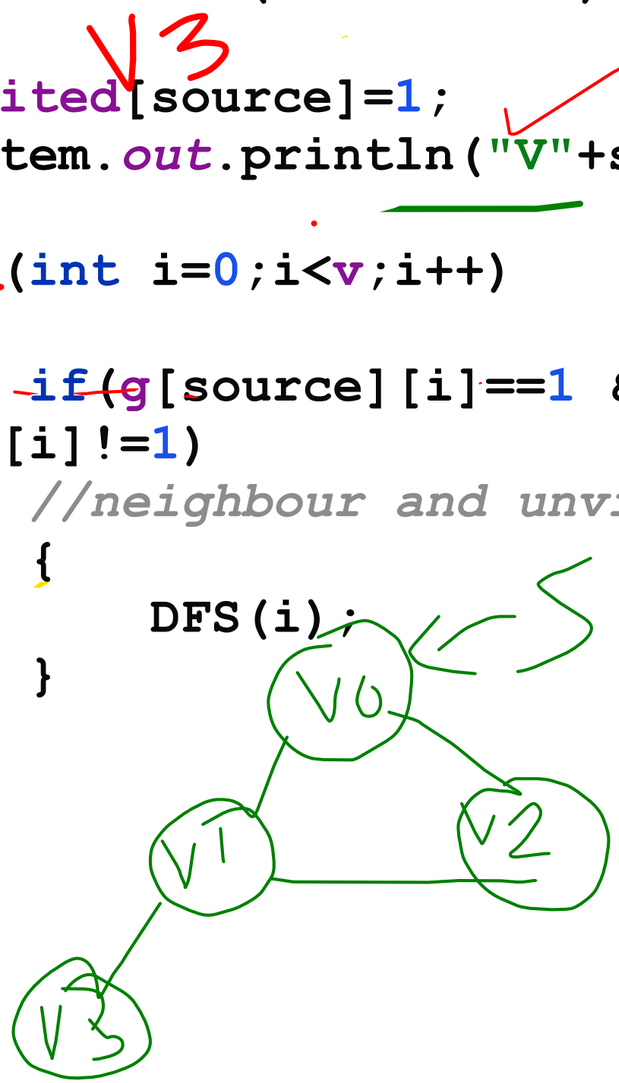
BFS

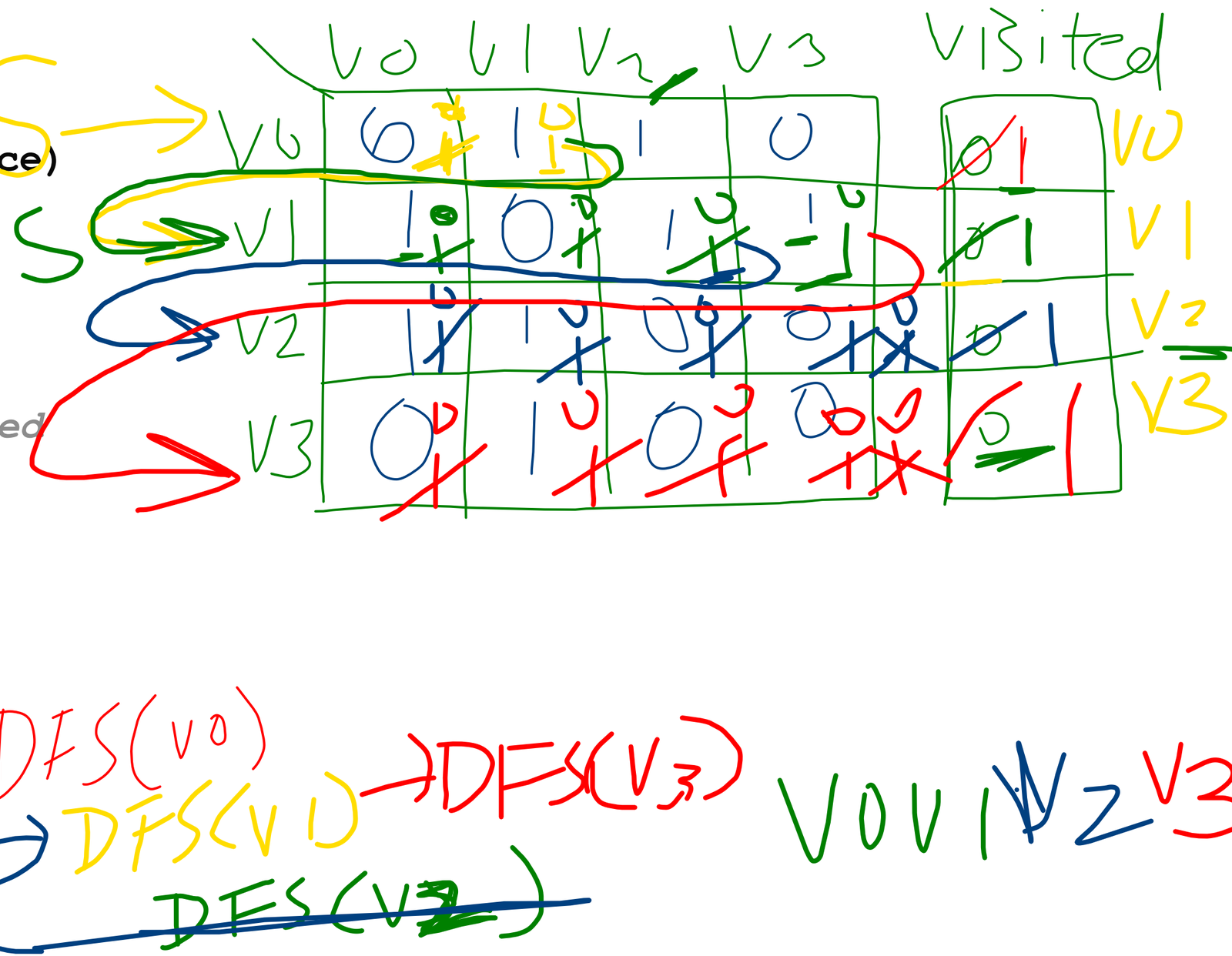
1. Start by putting source on queue and marking it visited.
2. Search for all unvisited neighbors of element at queue front ,mark them visited and put them on queue.
3. Remove element at queue front
4. Continue 2,3 till queue is not empty



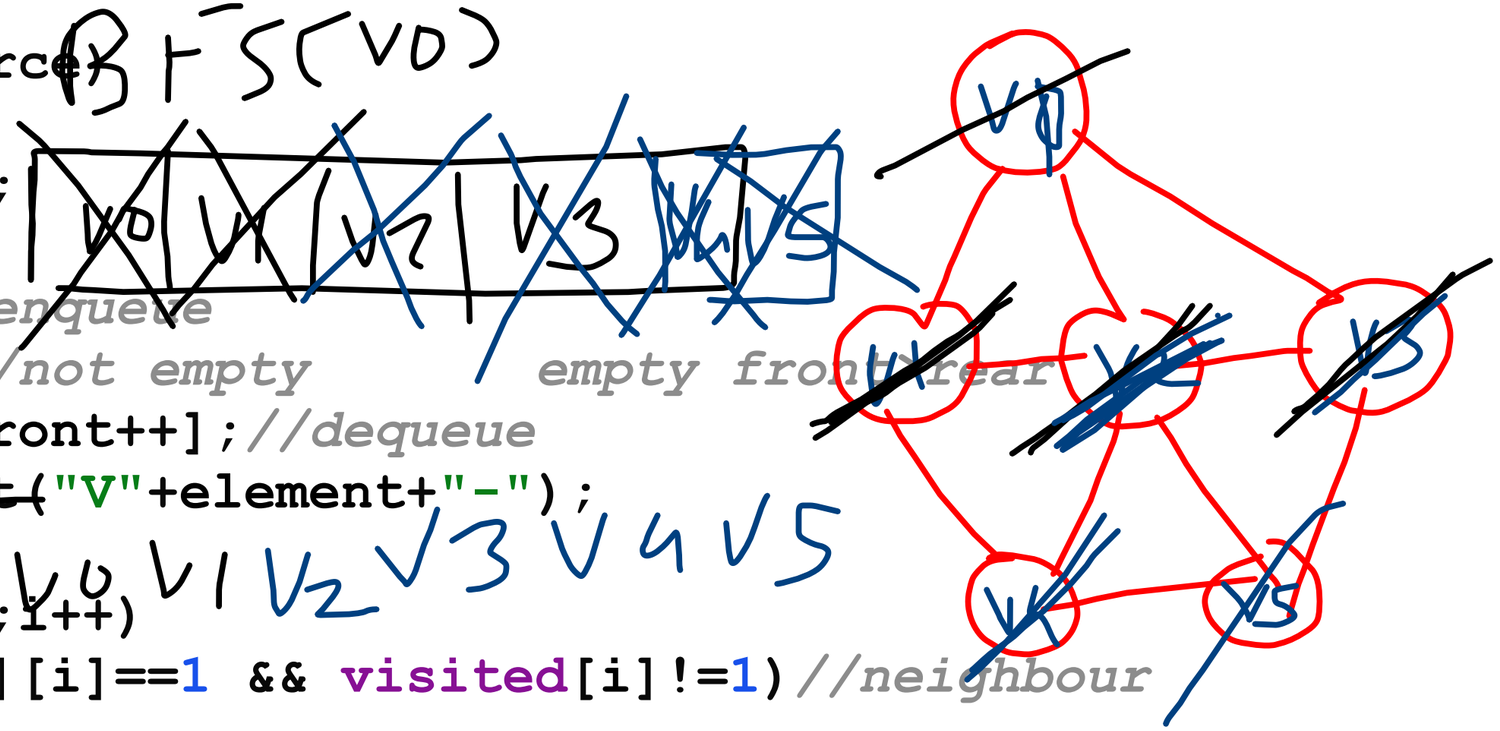
A B G H C D E F

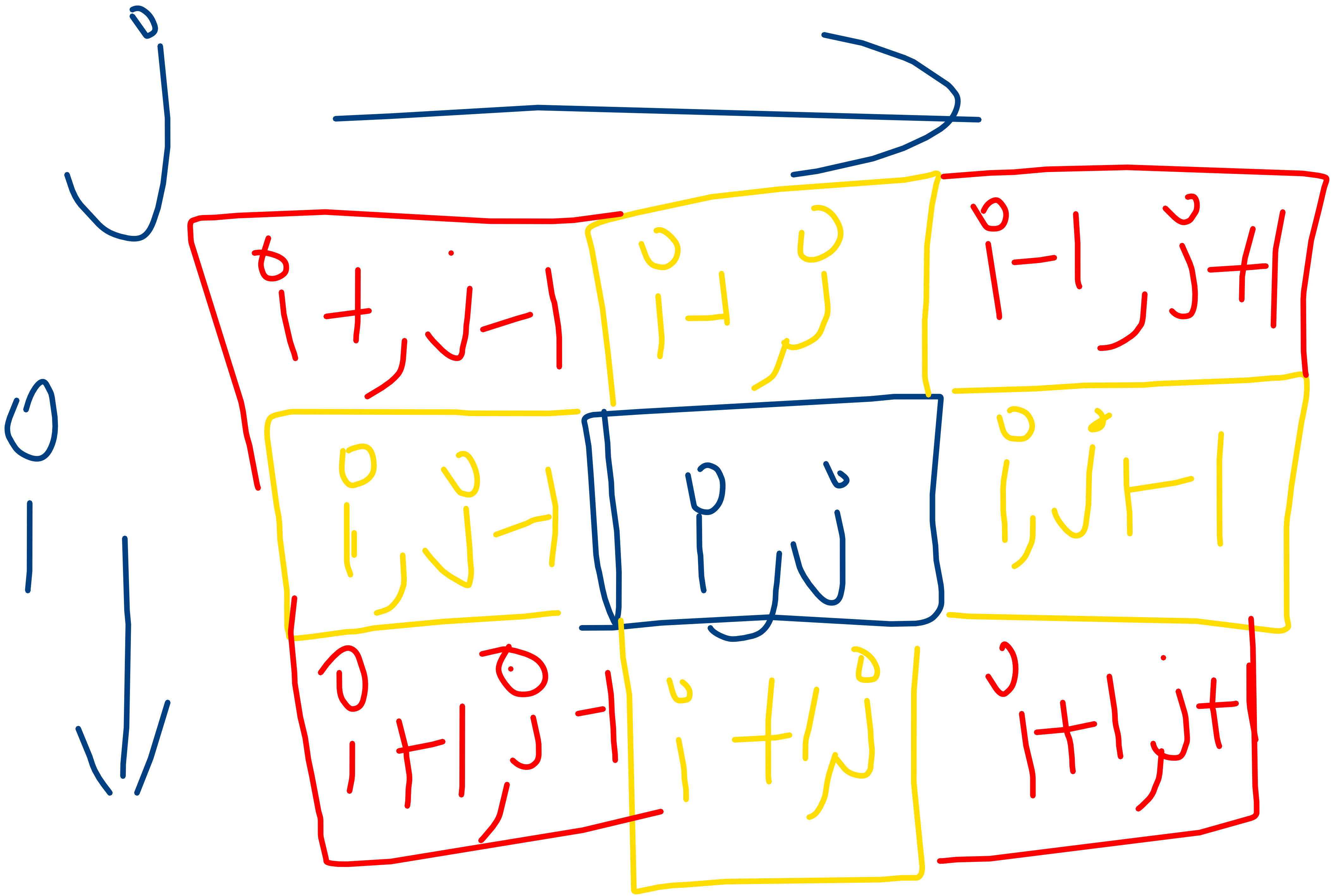
```
public void DFS(int source)
{
     visited[source]=1;
     System.out.println("V"+source)
;
     for(int i=0;i<v;i++)
     {
          if(g[source][i]==1 &&
visited[i]!=1)
          //neighbour and unvisited
          {
               DFS(i);
          }
     }
}
```





```
public void BFS(int source)
{    int q[]=new int[v];
     int front=0,rear=-1;
     visited[source]=1;
     q[++rear]=source;//enqueue
     while(front<=rear)//not empty
     {    int element=q[front++];//dequeue
          System.out.print("V"+element+"-");

          for(int i=0;i<v;i++)
          {  if(g[element][i]==1 && visited[i]!=1)//neighbour
and unvisited
             {    visited[i]=1;//visited
                  q[++rear]=i;//enqueue
             }
          }
     }
}
```




j
i
i,j-1
i-1,j
i-1,j+1
i,j-1
i,j
i,j+1
i+1,j-1
i+1,j
i+1,j+1