# Tree 树

✓1. build123()

```
struct node * build123()
{
    easy
}
```

---

✓2. size()

```
int size(struct node *node)
{   if (!node)
        return 0;
    if (node->left)
        return size(node->left) +1;
    if (node->right)
        return size(node->right)+1;
}
```

return (size(node->left) + node(node->right) +1)

3 maxDepth()

```
int maxDepth(struct node* node)
{
    if(!node) return 0;

    return max(maxDepth(node->left),
               maxDepth(node->right))
           + 1
}
```

4. minValue() BST

```
int minValue(struct node* node)
{
    node* current = node
    while (node->left)
        current
        node = node->left;
        current
    return node->val;
}
```

5. printTree()  BST

```
void printTree(struct node *node)
{
    if (!node) return;
    printTree(node->left)
    if (node->left) return(node->left)
    print node->val;
    if (node->right) return(node->right)
    printTree(node->right)
}
```

6. printPostorder() 强调问结合

```
void printPostorder (struct node *node)
{ if(!node) return;
  if(node->left)
    printTree(node->left)
    PrintPostorder(node->left)
  if(node->right)
    printTree(node->right)
    PrintPostorder(node->right)
  print(node->val)
}
```

7. hadPathSum()

```
int hasPathSum(struct node*
    node, int sum)
{ if (!node) return sum==0;
  if(node->left)
return(hasPathSum(node->left, sum-node->val)
  if(node->right)
return || hasPathSum(node->right, sum-node->val));
}
```

8. printPaths()

```
void printPaths(struct node* node)
{ vector<int> v;
  printPathRecur(node, v);
}
void printPathRecur(node *node, vector<int>& v)
{ if(!node->left && !node->right) // leaf node
  { for(auto i: v)
      cout<<i;
    cout<<endl;
  }
  if(node->left)
  { v.push_back(node->val);
    printPathRecur(node->left, v);
  }
```

```
if (node->right)
    { v.push_back(node->val);
      printPathsRecur(node->right, v)
    }
}
```

9. mirror()

```
void mirror(struct node *node)
{ if(node->left || node->right)
      swap(node->left, node->right)
  if (node->left)
     mirror(node->left)
  if (node->right)
     mirror (node->right)
}
```

```
9. void mirror(struct node* node)
{  if (!node) return;
   else
   {   mirror(node->left)
       mirror(node->right)
       Swap(node->left, node->right)
   }
}
```

10. doubleTree()

```
void doubleTree(struct node* node)
{
    if (!node) return;

    node newnode = new node(node->val);

    newnode->left = node->left;

    node->left = newnode;

    doubleTree(newnode->left);
    doubleTree(node->right)
}
```

11. sameTree()

```
int sameTree(struct node *a, struct node *b)
{ if(!a && !b) return true;
  else if(a && b)
  { if(a->val != b->val) return false;
    return [a->val == b->val &&
    SameTree(a->left, b->left) &&
    SameTree(a->right, b->right));
  }return true;
  else // 一个空一个非空
    return false;
}
```

12. countTree()

?

13.