124. 二叉树中的最大路径和

```
class Solution {
       int res = Intege r.M IN_ VALUE;
        public int maxPat hSu m(T reeNode root) {
                get One Sid eMa x(r oot);
                return res;
       }
        public int getOne Sid eMa x(T reeNode root) {
               if (root == null) {
                        return 0;
               }
               int left = Math.m ax(0, getOne Sid eMa x(r oot.le ft));
               int right = Math.m ax(0, getOne Sid eMa x(r oot.ri ght));
               res = Math.m ax(left + right + root.val, res);
                return Math.m ax( left, right) + root.val;
       }
}
```

230. 二叉搜索树中第K小的元素

```
class Solution {
       int k = 0;
       int res = 0;
        public int kthSma lle st( Tre eNode root, int k) {
                this.k = k;
                tra ver se( root);
                return res;
       }
        public void traver se( Tre eNode root) {
               if (root == null) {
                        return;
               }
                tra ver se( roo t.l eft);
               k--;
               if (k == 0) {
                       res = root.val;
                        return;
               }
                tra ver se( roo t.r ight);
       }
}
```

322. 零钱兑换

```
/*
        开辟一 个动态 数组， 基底d p[0]=0。
        dp[i] = num。
        i代表 目标值 ，nu m代表 硬币个数。
        num 初值为 amt +1。 代表不可能。
        每个d p[i ]都与 dp[ i-c oin ]的结 果相关联。
*/
class Solution {
        public int coinCh ang e(int[] coins, int amount) {

                int[] dp = new int[amount + 1];
                Arr ays.fi ll(dp, amount + 1);
                dp[0] = 0;
               for (int i = 1; i < dp.length; i++) {
                       for (int coin : coins) {
                               if (i - coin >= 0) {
                                       dp[i] = Math.m in(dp[i - coin] + 1, dp[i]);
                               }
                       }
               }
                return dp[dp.l ength - 1] == amount + 1? -1 : dp[dp.l ength - 1];
       }
}
```

46. 全排列

```
class Solution {
        boo lean[] used;
        Lis t<L ist <In teg er>> res;
        public List<L ist <In teg er>> permut e(int[] nums) {
               res = new Linked Lis t<>();
               used = new boolea n[n ums.le ngth];
                bac ktr ack (nums, new Linked Lis t<>());
                return res;
       }
        public void backtr ack (int[] nums, Linked Lis t<I nte ger> track) {
// 临时trac k大小满的时候
               if (track.size() == nums.l ength) {
                        res.ad d(new Linked Lis t(t rack));
                        return;
               }
               for (int i = 0; i < nums.l ength; i++) {
                       if (used[i]) continue;
```

46. 全排列 (cont)

```
> // 做选择，添加，下探结果
        used[i] = true;
        track.add(nums[i]);
        backtrack(nums, track);
        used[i] = false;
        track.removeLast();
    }
  }
}
```

105. 从前序与中序遍历序列构造二叉树

```
class Solution {
        Has hMa p<I nteger, Intege r> inorde rVa lMapIdx = new HashMa p<>();
        public TreeNode buildT ree (int[] preorder, int[] inorder) {
               for (int i = 0; i < inorde r.l ength; i++) {
                        ino rde rVa lMa pId x.p ut( ino rde r[i], i);
               }
                return buildT ree (pr eorder, inorder, 0, preord er.l ength - 1, 0, inorde r.l ength - 1);
       }
       // pre: 3 9 20 15 7
       // T L R
       // in : 9 3 15 20 7
       // L T R
        public TreeNode buildT ree (int[] preorder, int[] inorder, int preStart, int preEnd, int inStart,
int inEnd) {
               if (preStart > preEnd) {
                        return null;
               }
               int rootVal = preord er[ pre Start];
               int inorde rRo otIdx = inorde rVa lMa pId x.g et( roo tVal);
                Tre eNode root = new TreeNo de( roo tVal);
                roo t.left = buildT ree (pr eorder, inorder, preStart + 1, preStart + inorde rRo otIdx -
inStart, inStart, inorde rRo otIdx - 1);
                roo t.right = buildT ree (pr eorder, inorder, preStart + inorde rRo otIdx - inStart + 1,
preEnd, inorde rRo otIdx + 1, inEnd);
                return root;
       }
}
```