416 classic knapsack

Up-bottom:

Dfs without memo(TLE)

```
class Solution {
    public boolean canPartition(int[] nums) {
        int sum = 0;
        for(int n:nums) sum += n;
        if(sum % 2 != 0) return false;
        int target = sum / 2;
        return dfs(nums, 0, target);
    }
    public boolean dfs(int[] nums, int index, int target) {
        if(target < 0) return false;
        if(index == nums.length && target == 0) return true;
        if(index == nums.length) return false;
        return dfs(nums, index + 1, target) || dfs(nums, index + 1, target - nums[index]);
    }
}
```

Dfs with memo:

```
class Solution {
    Boolean[][] memo;
    public boolean canPartition(int[] nums) {
        int sum = 0;
        for(int n:nums) sum += n;
        if(sum % 2 != 0) return false;
        int target = sum / 2;
        memo = new Boolean[nums.length + 1][target + 1];
        return dfs(nums, 0, target);
    }
    public boolean dfs(int[] nums, int index, int target) {
        if(target < 0) return false;
        if(index == nums.length && target == 0) return true;
        if(index == nums.length) return false;
        if(memo[index][target] != null) return memo[index][target];
        memo[index][target] = dfs(nums, index + 1, target) || dfs(nums, index + 1, target - nums[index]);
        return memo[index][target];
    }
}
```

注意要用Boolean而不是boolean 用Boolean才能判断是否等于null。
`boolean`是基本数据类型
`Boolean`是它的封装类，和其他类一样，有属性有方法，可以`new`，例如：
`Boolean flag = new Boolean("true");  // boolean` 则不可以！

Bottom-up:
0-1背包问题 要么选要么不选
A good explaination:

This problem is essentially let us to find whether there are several numbers in a set which are able to sum to a specific value (in this problem, the value is sum/2).

Actually, this is a 0/1 knapsack problem, for each number, we can pick it or not. Let us assume dp[i][j] means whether the specific sum j can be gotten from the first i numbers. If we can pick such a series of numbers from 0-i whose sum is j, dp[i][j] is true, otherwise it is false.

Base case: dp[0][0] is true; (zero number consists of sum 0 is true)

Transition function: For each number, if we don't pick it, dp[i][j] = dp[i-1][j], which means if the first i-1 elements has made it to j, dp[i][j] would also make it to j (we can just ignore nums[i]). If we pick nums[i]. dp[i][j] = dp[i-1][j-nums[i]], which represents that j is composed of the current value nums[i] and the remaining composed of other previous numbers. Thus, the transition function is dp[i][j] = dp[i-1][j] || dp[i-1][j-nums[i]]

```java
class Solution {
    public boolean canPartition(int[] nums) {
        int sum = 0;
        for(int n:nums) sum += n;
        if(sum % 2 != 0) return false;
        int target = sum / 2;
        boolean[][] dp = new boolean[nums.length + 1][target + 1];
        for(int i = 0; i <= nums.length; i++) {
            dp[i][0] = true;
        }
        for(int i = 1; i <= nums.length; i++) {
            for(int j = 1; j <= target; j++) {
                dp[i][j] = dp[i - 1][j];//not choose
                if(j - nums[i - 1] >= 0) {
                    dp[i][j] = dp[i][j] || dp[i - 1][j - nums[i - 1]];//choose
                }
            }
        }
        return dp[nums.length][target];
    }
}
```

474 ones and zeros

每个字符串可以选择或不选择 有两个限制条件 字符串中0最多m个 1最多n个

top-bottom

Dfs without memo TLE

```java
class Solution {
    Map<String, int[]> map;
    public int findMaxForm(String[] strs, int m, int n) {
        map = new HashMap<>();
        for(String s: strs) {
            map.put(s, count(s));
        }
        return dfs(strs, m, n, 0);
    }
    public int dfs(String[] strs, int m, int n, int start) {
        if(start >= strs.length || m < 0 || n < 0) return 0;
        int max = Integer.MIN_VALUE;
        for(int i = start; i < strs.length; i++) {
            int[] count = map.get(strs[i]);
            int couldAddThisOne = (m - count[0] >= 0 && n - count[1] >= 0)? 1:0;
            //注意choose当前的时候 可能选了就超过限制了 所以这里判断是否可以选
            max = Math.max(max, couldAddThisOne + dfs(strs, m - count[0], n - count[1], i + 1));//choose
            max = Math.max(max, dfs(strs, m, n, i + 1));//not choose
        }
        return max;
    }

    public int[] count(String s) {
        char[] c = s.toCharArray();
        int[] res = new int[2];
        for(int i = 0; i < c.length; i++) {
            if(c[i] == '0') res[0]++;
            else res[1]++;
        }
        return res;
    }
}
```

Dfs with memo:

加了memo依然TLE

```
class Solution {
    Map<String, int[]> map;
    Integer[][][] memo;
    public int findMaxForm(String[] strs, int m, int n) {
        map = new HashMap<>();
        memo = new Integer[m + 1][n + 1][strs.length + 1];

        for(String s: strs) {
            map.put(s, count(s));
        }
        return dfs(strs, m, n, 0);
    }
    public int dfs(String[] strs, int m, int n, int start) {
        if(start >= strs.length || m < 0 || n < 0) return 0;
        int max = Integer.MIN_VALUE;
        if(memo[m][n][start] != null) return memo[m][n][start];
        for(int i = start; i < strs.length; i++) {
            int[] count = map.get(strs[i]);
            int couldAddThisOne = (m - count[0] >= 0 && n - count[1] >= 0)? 1:0;
            //注意choose当前的时候 可能选了就超过限制了 所以这里判断是否可以选
            max = Math.max(max, couldAddThisOne + dfs(strs, m - count[0], n - count[1], i + 1));//choose
            max = Math.max(max, dfs(strs, m, n, i + 1));//not choose
        }
        memo[m][n][start] = max;
        return max;
    }

    public int[] count(String s) {
        char[] c = s.toCharArray();
        int[] res = new int[2];
        for(int i = 0; i < c.length; i++) {
            if(c[i] == '0') res[0]++;
            else res[1]++;
        }
        return res;
    }
```

dp (bottom to top)

```java
class Solution {
    Map<String, int[]> map;
    public int findMaxForm(String[] strs, int m, int n) {
        map = new HashMap<>();
        for(String s: strs) {
            map.put(s, count(s));
        }
        int[][][] dp = new int[m + 1][n + 1][strs.length + 1];
        for(int i = 0; i <= m; i++) {
            for(int j = 0; j <= n; j++) {
                for(int index = 1; index <= strs.length; index++) {
                    dp[i][j][index] = dp[i][j][index - 1];
                    int[] count = map.get(strs[index - 1]);
                    if(i - count[0] >= 0 && j - count[1] >= 0) {
                        dp[i][j][index] = Math.max(dp[i][j][index], 1 + dp[i - count[0]][j - count[1]][index
- 1]);
                    }
                }
            }
        }
        //dp[m][n][index] = max(dp[m][n][index - 1], 1 + dp[m - count[0]][n - count[1]][index - 1])
        return dp[m][n][strs.length];
    }

    public int[] count(String s) {
        char[] c = s.toCharArray();
        int[] res = new int[2];
        for(int i = 0; i < c.length; i++) {
            if(c[i] == '0') res[0]++;
            else res[1]++;
        }
        return res;
    }
}
```

494. Target sum 每个可以选择加或减

Top-bottom: dfs without memo:

```java
class Solution {
    public int findTargetSumWays(int[] nums, int S) {
        return dfs(nums, 0, S);
    }
    public int dfs(int[] nums, int index, int target) {
        if(index == nums.length && target == 0) return 1;
        if(index == nums.length) return 0;
        int res = dfs(nums, index + 1, target - nums[index]) + dfs(nums, index + 1, target + nums[index]);
        return res;
    }
}
```

Dfs with memo(target may less than 0):

```java
class Solution {
    Integer[][] memo;
    public int findTargetSumWays(int[] nums, int S) {
        if(S > 1000) return 0;
        memo = new Integer[nums.length + 1][2000 + S + 1];
        return dfs(nums, 0, S);
    }
    public int dfs(int[] nums, int index, int target) {
        if(index == nums.length && target == 0) return 1;
        if(index == nums.length) return 0;
        if(memo[index][1000 + target] != null) return memo[index][1000 + target];
        int res = dfs(nums, index + 1, target - nums[index]) + dfs(nums, index + 1, target + nums[index]);
        memo[index][1000 + target] = res;
        return res;
    }
}
```

Your previous code was restored from your local storage. Reset to default

Dp bottom-up:

```java
class Solution {
    public int findTargetSumWays(int[] nums, int S) {
        //dp[i][j] = dp[i - 1][j - nums[i - 1]]  + dp[i - 1][j + nums[i - 1]]
        //dp[i][j]: to index i, num of combinations that could sum to j
        if(S > 1000 || S < -1000) return 0;
        int[][] dp = new int[nums.length + 1][S + 1 + 2000];
        dp[0][1000] = 1;
        for(int i = 1; i <= nums.length; i++) {
            for(int j = 0; j < S + 2001; j++) {
                if(j - nums[i - 1] >= 0 && j - nums[i - 1] < 2001 + S) dp[i][j] += dp[i - 1][j - nums[i - 1]];
                if(j + nums[i - 1] >= 0 && j + nums[i - 1] < 2001 + S) dp[i][j] += dp[i - 1][j + nums[i - 1]];
            }
        }
        return dp[nums.length][1000 + S];
    }
}
```

Your previous code was restored from your local storage. Reset to default

322 coin change

Dfs without memo:

```
class Solution {
    public int coinChange(int[] coins, int amount) {
        long res = dfs(coins, 0, amount);
        return res == Integer.MAX_VALUE? -1:(int)res;
    }
    public long dfs(int[] coins, int index, int target) {
        if(target < 0) return Integer.MAX_VALUE;
        if(target == 0) return 0;
        long min = Integer.MAX_VALUE;
        for(int i = index; i < coins.length; i++) {
            min = Math.min(min, 1 + dfs(coins, i, target - coins[i]));

        }
        min = Math.min(min, Integer.MAX_VALUE);
        return min;
    }
}
```

Dfs + memo:

```
class Solution {
    Integer[][] memo;
    public int coinChange(int[] coins, int amount) {
        memo = new Integer[coins.length + 1][amount + 1];
        long res = dfs(coins, 0, amount);
        return res == Integer.MAX_VALUE? -1:(int)res;
    }
    public long dfs(int[] coins, int index, int target) {
        if(target < 0) return Integer.MAX_VALUE;
        if(target == 0) return 0;
        if(memo[index][target] != null) return memo[index][target];
        long min = Integer.MAX_VALUE;
        for(int i = index; i < coins.length; i++) {
            min = Math.min(min, 1 + dfs(coins, i, target - coins[i]));

        }
        min = Math.min(min, Integer.MAX_VALUE);
        memo[index][target] = (int)min;
        return min;
    }
```

Dp:(bottom to up)

```java
class Solution {
    public int coinChange(int[] coins, int amount) {
        //dp[index][amount]: to the index, min coins sum to this amount
        //dp[i][j] = min(1 + dp[i][j - coins[i - 1]], dp[i - 1][j])
        //dp[any][0]
        long[][] dp = new long[coins.length + 1][amount + 1];
        for(int i = 0; i <= coins.length; i++) {
            Arrays.fill(dp[i], Integer.MAX_VALUE);
        }
        for(int i = 0; i <= coins.length; i++) {
            dp[i][0] = 0;
        }
        for(int i = 1; i <= coins.length; i++) {
            for(int j = 1; j <= amount; j++) {
                dp[i][j] = dp[i - 1][j];
                if(j - coins[i - 1] >= 0) {
                    dp[i][j] = Math.min(dp[i][j], 1 + dp[i][j - coins[i - 1]]);
                }
            }
        }
        dp[coins.length][amount] = Math.min(dp[coins.length][amount], Integer.MAX_VALUE);
        return dp[coins.length][amount] == Integer.MAX_VALUE? -1:(int)dp[coins.length][amount];
    }
}
```

Coin change II

Dfs without memo:

```java
class Solution {
    public int change(int amount, int[] coins) {
        if(coins.length == 0 && amount == 0) return 1;
        if(coins.length == 0) return 0;
        return dfs(coins, 0, amount);
    }
    public int dfs(int[] coins, int index, int target) {
        if(target == 0 && index < coins.length) return 1;
        if(target < 0 || index >= coins.length) return 0;
        int sum = 0;
        for(int i = index; i < coins.length; i++) {
            sum += dfs(coins, i, target - coins[i]);
        }
        return sum;
    }
}
```

Dfs with memo:

```java
class Solution {
    Integer[][] memo;
    public int change(int amount, int[] coins) {
        if(coins.length == 0 && amount == 0) return 1;
        if(coins.length == 0) return 0;
        memo = new Integer[coins.length + 1][amount + 1];
        return dfs(coins, 0, amount);
    }
    public int dfs(int[] coins, int index, int target) {
        if(target == 0 && index < coins.length) return 1;
        if(target < 0 || index >= coins.length) return 0;
        if(memo[index][target] != null) return memo[index][target];
        int sum = 0;
        for(int i = index; i < coins.length; i++) {
            sum += dfs(coins, i, target - coins[i]);
        }
        memo[index][target] = sum;
        return sum;
    }
}
```

Dp:

```java
class Solution {
    public int change(int amount, int[] coins) {
        //dp[index][amount]: to this index, num of ways to reach this amount
        //dp[i][j] = dp[i - 1][j] + dp[i][j - coins[i - 1]]
        //dp[any][0] = 1
        if(coins.length == 0 && amount == 0) return 1;
        if(coins.length == 0) return 0;
        int[][] dp = new int[coins.length + 1][amount + 1];
        for(int i = 0; i < coins.length + 1; i++) {
            dp[i][0] = 1;
        }
        for(int i = 1; i < coins.length + 1; i++) {
            for(int j = 1; j < amount + 1; j++) {
                dp[i][j] += dp[i - 1][j];
                if(j - coins[i - 1] >= 0) {
                    dp[i][j] += dp[i][j - coins[i - 1]];
                }
            }
        }
        return dp[coins.length][amount];
    }
}
```

198. House robber

```java
class Solution {
    public int rob(int[] nums) {
        //dp[i]: maximum to this index
        //dp[i] = Math.max(nums[i - 1] + dp[i - 2], dp[i - 1])
        //dp[0] = 0, dp[1] = nums[1]
        if(nums.length == 0) return 0;
        if(nums.length == 1) return nums[0];
        int[] dp = new int[nums.length + 1];
        dp[0] = 0; dp[1] = nums[0];
        for(int i = 2; i <= nums.length; i++) {
            dp[i] = Math.max(dp[i - 1], nums[i - 1] + dp[i - 2]);
        }
        return dp[nums.length];
    }
}
```

213. House robber II

```java
class Solution {
    public int rob(int[] nums) {
        //dp[i] = Math.max(dp[i - 2] + nums[i - 1], dp[i - 1])  if(i - 2 != (i + 1)%nums.length)
        if(nums.length == 0) return 0;
        if(nums.length == 1) return nums[0];
        return Math.max(robHelper(Arrays.copyOfRange(nums, 0, nums.length - 1)),
robHelper(Arrays.copyOfRange(nums, 1, nums.length)));
    }

    public int robHelper(int[] nums) {
        if(nums.length == 0) return 0;
        if(nums.length == 1) return nums[0];
        int[] dp = new int[nums.length + 1];
        dp[0] = 0;
        dp[1] = nums[0];
        for(int i = 2; i <= nums.length ; i++) {
            dp[i] = Math.max(dp[i - 1], dp[i - 2] + nums[i - 1]);
        }
        return dp[nums.length];
    }
}
```

337. House robber III

Postorder traversal, for each node, save a maximum number in map
When we consider each root node, we have 2 choices: 1-choose root, 2-not choose node

```
class Solution {
    Map<TreeNode, Integer> map;
    public int rob(TreeNode root) {
        //dp[root] = Math.max(dp[leftChild] + dp[rightChild], dp[root] + Math.max(dp[leftChild.leftChild],
dp[leftChild.right]) + Math.max(dp[rightChild.left], dp[rightChild.right]))
        //if(root.left == null && root.right == null) dp[root] = root.val;
        //map.put(null, 0)
        map = new HashMap<>();
        map.put(null, 0);
        dfs(root);
        return map.get(root);
    }
    public void dfs(TreeNode root) {
        if(root == null) return;
        if(root.left == null && root.right == null) {
            map.put(root, root.val);
            return;
        }
        dfs(root.left);
        dfs(root.right);
        int notChoose = map.get(root.left) + map.get(root.right);
        int chooseRoot = root.val;
        if(root.left != null) {
            chooseRoot += (map.get(root.left.left) + map.get(root.left.right));
        }
        if(root.right != null) {
            chooseRoot += (map.get(root.right.left) + map.get(root.right.right));
        }
        map.put(root, Math.max(notChoose, chooseRoot));
        return;
    }
}
```

256. paint house

```
class Solution {
    public int minCost(int[][] costs) {
        //dp[i][j]: minimum cost of painting room i with color j
        if(costs.length == 0 || costs[0].length == 0) return 0;
        int n = costs.length;
        int[][] dp = new int[n + 1][3];
        for(int i = 1; i <= n; i++) {
            dp[i][0] = Math.min(dp[i - 1][1], dp[i - 1][2]) + costs[i - 1][0];
            dp[i][1] = Math.min(dp[i - 1][0], dp[i - 1][2]) + costs[i - 1][1];
            dp[i][2] = Math.min(dp[i - 1][0], dp[i - 1][1]) + costs[i - 1][2];
        }
        int res = Math.min(dp[n][0], dp[n][1]);
        res = Math.min(res, dp[n][2]);
        return res;
    }
}
```

以precolor & index做dp：

```
class Solution {
    public int minCost(int[][] costs) {
        //dp[i][j]:precolor is j, index i, the minimum cost
        //dp[i][j] = costs[i - 1][j] + Math.min(dp[i - 1][(j + 1) % 3, dp[i - 1][(j + 2) % 3]])
        if(costs.length == 0 || costs[0].length == 0) return 0;
        int[][] dp = new int[costs.length + 1][3];
        for(int i = 1; i <= costs.length; i++) {
            for(int j = 0; j <3; j++) {
                dp[i][j] = costs[i - 1][j] + Math.min(dp[i - 1][(j + 1) % 3], dp[i - 1][(j + 2) % 3]);
            }
        }
        int min = Math.min(dp[costs.length][0], dp[costs.length][1]);
        min = Math.min(min, dp[costs.length][2]);
        return min;
    }
}
```

265. Paint house II

Dp, just remember the smallest 2 of last house

```
class Solution {
    public int minCostII(int[][] costs) {
        //dp[i][j]: minimum cost for painting house i and precolor is j
        //dp[i][j] = if(smallColor != j) costs[i - 1][j] + small, else = costs[i - 1][j] + big
        //update small , big, smallColor, bigColor
        if(costs.length == 0 || costs[0].length == 0) return 0;
        int small = 0, big = 0, smallColor = -1, bigColor = -1;
        for(int i = 1; i <= costs.length; i++) {
            int s = Integer.MAX_VALUE, b = Integer.MAX_VALUE, sColor = -1, bColor = -1;
            for(int j = 0; j < costs[0].length; j++) {
                int curr = costs[i - 1][j];
                if(smallColor != j) {
                    curr += small;
                } else {
                    curr += big;
                }
                if(curr < s) {
                    b = s;
                    bColor = sColor;
                    s = curr;
                    sColor = j;
                } else if(curr < b) {
                    b = curr;
                    bColor = j;
                }
            }
            small = s; big = b; smallColor = sColor; bigColor = bColor;
        }
        return small;
    }
}
```

1473. Paint house III

```java
class Solution {
    public int minCost(int[] houses, int[][] cost, int m, int n, int target) {
        //dp[index][precolor][target] = cost[index - 1][color] + min(dp[index - 1][preColor][target], dp[index - 1][otherColor][target - 1])
        //if(houses[index - 1] != 0) dp[index][precolor][target] = min(dp[index - 1][preColor][target], dp[index - 1][otherColor][target - 1])
        //dp[0][any][0] = 0
        long[][][] dp = new long[m + 1][n + 1][target + 1];
        for(int i = 0; i <= m; i++) {
            for(int j = 0; j <= n; j++) {
                Arrays.fill(dp[i][j], Integer.MAX_VALUE);
            }
        }
        for(int i = 0; i <= n; i++) {
            dp[0][i][0] = 0;
        }
        for(int i = 1; i <= m; i++) {
            for(int j = 1; j <= n; j++) {
                for(int k = 1; k <= target; k++) {
                    if(houses[i - 1] == 0) {
                        dp[i][j][k] = cost[i - 1][j - 1];
                        long min = dp[i - 1][j][k];
                        for(int preColor = 1; preColor <= n; preColor++) {
                            if(preColor != j) {
                                min = Math.min(min, dp[i - 1][preColor][k - 1]);
                            }
                        }
                        dp[i][j][k] += min;
                    } else {
                        int curColor = houses[i - 1];
                        long min = dp[i - 1][curColor][k];
                        for(int preColor = 1; preColor <= n; preColor++) {
                            if(preColor != curColor) {
                                min = Math.min(min, dp[i - 1][preColor][k - 1]);
                            }
                        }
                        dp[i][curColor][k] = min;
                    }
                }
            }
        }
        long res = Integer.MAX_VALUE;
        for(int i = 0; i <= n; i++) {
            res = Math.min(dp[m][i][target], res);
        }
        return res == Integer.MAX_VALUE? -1:(int)res;
    }
}
```

55. Jump game

```
class Solution {
    public boolean canJump(int[] nums) {
        int max = 0;
        for(int i = 0; i < nums.length; i++) {
            if(max >= i) {
                max = Math.max(max, nums[i] + i);
            } else break;
        }
        return max >= (nums.length - 1);
    }
}
```

45. Jump game II

Dp TLE

```
class Solution {
    public int jump(int[] nums) {
        //dp[i]: min steps to this index
        int[] dp = new int[nums.length];
        Arrays.fill(dp, Integer.MAX_VALUE);
        dp[0] = 0;
        for(int i = 0; i < nums.length; i++) {
            for(int j = 0; j < i; j++) {
                if(nums[j] + j >= i) {
                    dp[i] = Math.min(dp[i], dp[j] + 1);
                }
            }
        }
        return dp[nums.length - 1];
    }
}
```

Has to O(n), bfs

```java
class Solution {
    public int jump(int[] nums) {
        //dp[i]: min steps to this index
        int far = nums[0], steps = 0, index = 1;
        while(index < nums.length) {
            steps++;
            int nextFar = 0;
            while(index < nums.length && far >= index) {
                nextFar = Math.max(nextFar, nums[index] + index);
                index++;
            }
            far = nextFar;
        }
        return steps;
    }
}
```

Simple code:

Java ● Autocomplete

```java
class Solution {
    public int jump(int[] nums) {
        //dp[i]: min steps to this index
        int end = 0, far = 0, step = 0;
        for(int i = 0; i < nums.length - 1; i++) {
            far = Math.max(far, i + nums[i]);
            if(i == end) {
                step++;
                end = far;
            }
        }
        return step;
    }
}
```

1306. Jump game III

bfs

```
class Solution {
    public boolean canReach(int[] arr, int start) {
        boolean[] visited = new boolean[arr.length];
        Queue<Integer> q = new LinkedList<>();
        q.add(start);
        visited[start] = true;
        while(!q.isEmpty()) {
            int size = q.size();
            for(int i = 0; i < size; i++) {
                int curr = q.poll();
                if(arr[curr] == 0) return true;
                int left = curr - arr[curr];
                int right = curr + arr[curr];
                if(left >= 0 && left < arr.length && visited[left] == false) {
                    q.add(left);
                    visited[left] = true;
                }
                if(right >= 0 && right < arr.length && visited[right] == false) {
                    q.add(right);
                    visited[right] = true;
                }
            }
        }
        return false;
    }
}
```

Recursive

```
class Solution {
    public boolean canReach(int[] arr, int start) {
        if(start >= 0 && start < arr.length) {
            int tmp = arr[start];
            arr[start] += arr.length;
            return tmp == 0 || canReach(arr, start + tmp) || canReach(arr, start - tmp);
        }
        return false;
    }
}
```

1345. Jump game IV

bfs

```java
class Solution {
    public int minJumps(int[] arr) {
        Map<Integer, List<Integer>> map = new HashMap<>();
        for(int i = 0; i < arr.length; i++) {
            map.computeIfAbsent(arr[i], k -> new ArrayList<>()).add(i);
        }
        boolean[] visited = new boolean[arr.length];
        Queue<Integer> q = new LinkedList<>();
        q.add(0);
        visited[0] = true;
        int step = 0;
        while(!q.isEmpty()) {
            int size = q.size();
            for(int i = 0; i < size; i++) {
                int curr = q.poll();
                if(curr == arr.length - 1) {
                    return step;
                }
                List<Integer> list = map.get(arr[curr]);
                list.add(curr + 1);
                list.add(curr - 1);
                for(int each:list) {
                    if(each == curr) continue;
                    if(each >= 0 && each < arr.length && visited[each] == false) {
                        q.add(each);
                        visited[each] = true;
                    }
                }
                list.clear();//must clear !!!
            }
            step++;
        }
        return 0;
    }
}
```

1340. Jump game V

Dfs without memo

```java
class Solution {
    //stack over flow 问题说明循环没有停下来
    //TLE
    public int maxJumps(int[] arr, int d) {
        if(arr.length == 0 || d == 0) return 0;
        int max = 0;
        for(int i = 0; i < arr.length; i++) {
            max = Math.max(max, dfs(arr, i, d));
        }
        return max + 1;
    }
    public int dfs(int[] arr, int index, int d) {
        int max = 0;
        for(int i = index + 1; i <= Math.min(arr.length - 1, index + d); i++) {
            if(arr[i] >= arr[index]) break;
            max = Math.max(max, 1 + dfs(arr, i, d));
        }
        for(int i = index - 1; i >= Math.max(0, index - d); i--) {
            if(arr[i] >= arr[index]) break;
            max = Math.max(max, 1 + dfs(arr, i, d));
        }
        return max;
    }
}
```

Dfs with memo:

```java
class Solution {
    //stack over flow 问题说明循环没有停下来
    //TLE
    int[] memo;
    public int maxJumps(int[] arr, int d) {
        if(arr.length == 0 || d == 0) return 0;
        int max = 0;
        memo = new int[arr.length + 1];
        Arrays.fill(memo, -1);
        for(int i = 0; i < arr.length; i++) {
            max = Math.max(max, dfs(arr, i, d));
        }
        return max + 1;
    }
    public int dfs(int[] arr, int index, int d) {
        int max = 0;
        if(memo[index] != -1) return memo[index];
        for(int i = index + 1; i <= Math.min(arr.length - 1, index + d); i++) {
            if(arr[i] >= arr[index]) break;
            max = Math.max(max, 1 + dfs(arr, i, d));
        }
        for(int i = index - 1; i >= Math.max(0, index - d); i--) {
            if(arr[i] >= arr[index]) break;
            max = Math.max(max, 1 + dfs(arr, i, d));
        }
        memo[index] = max;
        return max;
    }
}
```

877 stone game

Dfs with memo:

```java
class Solution {
    int[][] memo;
    public boolean stoneGame(int[] piles) {
        memo = new int[piles.length + 1][piles.length + 1];
        for(int i = 0; i < piles.length + 1; i++) {
            Arrays.fill(memo[i], -1);
        }
        int alexOverLee = dfs(piles, 0, piles.length - 1);
        return alexOverLee > 0;
    }

    public int dfs(int[] piles, int i, int j) {
        if(i == j) return piles[i];
        if(memo[i][j] != -1) return memo[i][j];
        memo[i][j] = Math.max(piles[i] - dfs(piles, i + 1, j), piles[j] - dfs(piles, i, j - 1));
        return memo[i][j];
    }
}
```

dp(bottom to up):

```
class Solution {
    public boolean stoneGame(int[] piles) {
        //dp[i][j]: pick from i and j, the sum over next person
        int[][] dp = new int[piles.length][piles.length];
        for(int i = 0; i < piles.length; i++) {
            dp[i][i] = piles[i];
        }
        for(int i = 1; i < piles.length; i++) {
            for(int j = 0; j < piles.length - i; j++) {
                dp[j][j + i] = Math.max(piles[j] - dp[j + 1][j + i], piles[j + i] - dp[j][j + i - 1]);
            }
        }
        return dp[0][piles.length - 1] > 0;
    }
}
```

1140 stone game II

Dfs + memo:
Dfs的max初始值为min_value 一开始写成0就错了 过程中可能出现负值

```
class Solution {
    int[][] memo;
    public int stoneGameII(int[] piles) {
        memo = new int[piles.length + 1][101];
        for(int i = 0; i <= piles.length; i++) {
            Arrays.fill(memo[i], -1);
        }
        int alexOverLee = dfs(piles, 0, 1);
        int sum = 0;
        for(int p:piles) sum += p;
        return (sum + alexOverLee) / 2;
    }
    public int dfs(int[] piles, int index, int M) {
        if(index >= piles.length) return 0;
        int max = Integer.MIN_VALUE;
        if(memo[index][M] != -1) return memo[index][M];
        for(int i = 1; i <= 2 * M; i++) {
            int currGet = 0;
            for(int j = index; j < Math.min(piles.length, index + i); j++) {
                currGet += piles[j];
            }
            max = Math.max(max, currGet - dfs(piles, index + i, Math.max(M, i)));
        }
        memo[index][M] = max;
        return max;
    }
}
```

1406. Stone game III
Dfs + memo:

```java
class Solution {
    Integer[] memo;
    public String stoneGameIII(int[] stoneValue) {
        int sum = 0;
        memo = new Integer[stoneValue.length + 1];
        for(int s:stoneValue) sum += s;
        int aliceOverBob = dfs(stoneValue, 0);
        int alice = (sum + aliceOverBob) / 2;
        int Bob = sum - alice;
        if(alice > Bob) return "Alice";

        else if(alice < Bob) return "Bob";
        else return "Tie";
    }
    public int dfs(int[] stone, int index) {
        if(index >= stone.length) return 0;
        if(memo[index] != null) return memo[index];
        int max = Integer.MIN_VALUE;
        for(int i = 1; i <= 3; i++) {
            int currGet = 0;
            for(int j = index; j < Math.min(index + i, stone.length); j++) {
                currGet += stone[j];//Math.min()记得！！！不然会超界
            }
            max = Math.max(max, currGet - dfs(stone, index + i));
        }
        memo[index] = max;
        return max;
    }
}
```

121. Best time to buy and sell stock

```java
class Solution {
    public int maxProfit(int[] prices) {
        if(prices.length <= 1) return 0;
        int min = prices[0];
        int max = 0;
        for(int i = 1; i < prices.length; i++) {
            max = Math.max(max, prices[i] - min);
            min = Math.min(min, prices[i]);
        }
        return max;
    }
}
```

122. Best time to buy and sell stock II

```java
class Solution {
    public int maxProfit(int[] prices) {
        int min = prices[0];
        int sum = 0;
        for(int i = 1; i < prices.length; i++) {
            if(prices[i] > min) {
                sum += (prices[i] - min);
                min = prices[i];
            } else {
                min = Math.min(min, prices[i]);
            }
        }
        return sum;
    }
}
```

123. Best Time to Buy and Sell Stock III

Straight forward solution: (not efficient)

```
class Solution {
    public int maxProfit(int[] prices) {
        int max = 0;
        if(prices.length < 3) return helper(prices);
        for(int i = 1; i <= prices.length - 2; i++) {
            max = Math.max(max, helper(Arrays.copyOfRange(prices, 0, i + 1)) + helper(Arrays.copyOfRange(prices, i,
prices.length)));
        }
        return max;
    }

    public int helper(int[] prices) {
        if(prices.length <= 1) return 0;
        int min = prices[0];
        int max = 0;
        for(int i = 1; i < prices.length; i++) {
            max = Math.max(max, prices[i] - min);
            min = Math.min(min, prices[i]);
        }
        return max;
    }
```

Dfs + memo TLE:

```
class Solution {
    Integer[][] memo;
    public int maxProfit(int[] prices) {
        if(prices.length <= 1) return 0;
        if(prices.length == 2) return Math.max(0, prices[1] - prices[0]);
        memo = new Integer[prices.length + 1][3];
        return Math.max(dfs(prices, 0, 2), 0);
    }

    public int dfs(int[] prices, int index, int target) {
        if(target == 0 || index >= prices.length - 1) return 0;
        if(memo[index][target] != null) return memo[index][target];
        int max = Integer.MIN_VALUE;
        for(int i = index; i < prices.length; i++) {
            for(int j = i + 1; j < prices.length; j++) {
                max = Math.max(max, Math.max(0, prices[j] - prices[i]) + Math.max(0, dfs(prices, j, target - 1)));
            }
        }
        memo[index][target] = max;
        return max;
    }
```

se Run Code Result Debugger

dp(bottom to up)

```java
class Solution {
    //dp[i][j]: maxprofit in day i, with left trasaction j
    //dp[i][j] = max(dp[i - 1][j] (no trasaction on day j), prices[i - 1] - prices[k - 1](k = 0...i - 2) + dp[k][j - 1]
    //dp[0][any] = 0
    public int maxProfit(int[] prices) {
        if(prices.length <= 1) return 0;
        int[][] dp = new int[prices.length + 1][3];
        for(int i = 0; i < 3; i++) {
            dp[0][i] = 0;
        }
        for(int i = 1; i <= prices.length; i++) {
            for(int j = 1; j < 3; j++) {
                dp[i][j] = dp[i - 1][j];
                for(int k = 1; k <= i - 1; k++) {
                    dp[i][j] = Math.max(dp[i][j], prices[i - 1] - prices[k - 1] + dp[k][j - 1]);
                }
            }
        }
        return dp[prices.length][2];
    }
```