416. 分割等和子集

```
func canPartition(nums []int) bool {
    n := len(nums)
    sum := 0
    for i := 0; i < n; i++ {
        sum += nums[i]
    }
    if sum % 2 == 1 {return false}
    sum /= 2
    dp := make([][]bool, n)
    for i := 0; i < len(dp); i++ {
        dp[i] = make([]bool, sum+1)
    }
    dp[0][0] = true
    if nums[0] <= sum {
        dp[0][nums[0]] = true
    }
    for i := 1; i < n; i++ {
        for j := 0; j <= sum; j++ {
            if j-nums[i] >= 0 {
                dp[i][j] = dp[i-1][j] || dp[i-1][j-nums[i]]
            } else {
                dp[i][j] = dp[i-1][j]
            }
        }
    }
    return dp[n-1][sum]
}
```

494. 目标和

```
func findTargetSumWays(nums []int, target int) int {
    if target > 1000 || target < -1000 {return 0}
    n := len(nums)
    offset := 1000
    w := 2000
    dp := make([][]int, n)
    for i := 0; i < len(dp); i++ {
        dp[i] = make([]int, w+1)
    }
    dp[0][offset-nums[0]] += 1
    dp[0][offset+nums[0]] += 1

    for i := 1; i < n; i++ {
        for j := 0; j <= w; j++ {
            if j-nums[i] >= 0 && j-nums[i] <= w {
                dp[i][j] = dp[i-1][j-nums[i]]
            }
            if j+nums[i] >= 0 && j+nums[i] <= w {
                dp[i][j] += dp[i-1][j+nums[i]]
            }
        }
    }
    return dp[n-1][target+1000]
}
```

322. 零钱兑换

```go
func coinChange(coins []int, amount int) int {
    n := len(coins)
    dp := make([][]int, n)
    for i := 0; i < n; i++ {
        dp[i] = make([]int, amount+1)
        for j := 0; j <= amount; j++ {
            dp[i][j] = math.MaxInt32
        }
    }
    for c := 0; c <= amount/coins[0]; c++ {
        dp[0][c*coins[0]] = c
    }
    for i := 1; i < n; i++ {
        for j := 0; j <= amount; j++ {
            k := j/coins[i]
            for c := 0; c <= k; c++ {
                if dp[i-1][j-c*coins[i]] != math.MinInt32 &&
                    dp[i-1][j-c*coins[i]] + c < dp[i][j] {
                    dp[i][j] = dp[i-1][j-c*coins[i]] + c
                }
            }
        }
    }
    if dp[n-1][amount] == math.MaxInt32 {return -1}
    return dp[n-1][amount]
}
```

518. 零钱兑换 II

```go
func change(amount int, coins []int) int {
    n := len(coins)
    dp := make([][]int, n)
    for i := 0; i < len(dp); i++ {
        dp[i] = make([]int, amount+1)
    }
    for c := 0; c <= amount/coins[0]; c++ {
        dp[0][c*coins[0]] = 1
    }
    for i := 1; i < n; i++ {
        for j := 0; j <= amount; j++ {
            k := j/coins[i]
            for c := 0; c <= k; c++ {
                dp[i][j] += dp[i-1][j-c*coins[i]]
            }
        }
    }
    return dp[n-1][amount]
}
```

64. 最小路径和

```go
func minPathSum(grid [][]int) int {
    m := len(grid)
    n := len(grid[0])
    dp := make([][]int, m)
    for i := 0; i < m; i++ {
        dp[i] = make([]int, n)
    }
    len := 0
    for i := 0; i < m; i++ {
```

```
        len += grid[i][0]
        dp[i][0] = len
    }
    len = 0
    for j := 0; j < n; j++ {
        len += grid[0][j]
        dp[0][j] = len
    }
    for i := 1; i < m; i++ {
        for j := 1; j < n; j++ {
            dp[i][j] = int(math.Min(float64(dp[i-1][j]), float64(dp[i][j-1]))) +
grid[i][j]
        }
    }
    return dp[m-1][n-1]
}
```

剑指 Offer 47. 礼物的最大价值

```
func maxValue(grid [][]int) int {
    n := len(grid)
    m := len(grid[0])
    dp := make([][]int, n)
    for i := 0; i < len(dp); i++ {
        dp[i] = make([]int, m)
    }
    sum := 0
    for j := 0; j < m; j++ {
        sum += grid[0][j]
        dp[0][j] = sum
    }
    sum = 0
    for i := 0; i < n; i++ {
        sum += grid[i][0]
        dp[i][0] = sum
    }
    for i := 1; i < n; i++ {
        for j := 1; j < m; j++ {
            dp[i][j] = int(math.Max(float64(dp[i-1][j]), float64(dp[i][j-1]))) +
grid[i][j]
        }
    }
    return dp[n-1][m-1]
}
```

120. 三角形最小路径和

```
func minimumTotal(triangle [][]int) int {
    n := len(triangle)
    dp := make([][]int, n)
    for i := 0; i < len(dp); i++ {
        dp[i] = make([]int, n)
    }
    dp[0][0] = triangle[0][0]
    for i := 1; i < n; i++ {
        dp[i][0] = dp[i-1][0] + triangle[i][0]
        for j := 1; j < i; j++ {
            dp[i][j] = int(math.Min(float64(dp[i-1][j]), float64(dp[i-1][j-1]))) +
triangle[i][j]
```

```
        }
        dp[i][i] = dp[i-1][i-1] + triangle[i][i]
    }
    res := math.MaxInt32
    for j := 0; j < n; j++ {
        if dp[n-1][j] < res {res = dp[n-1][j]}
    }
    return res
}
```

62. 不同路径

```
func uniquePaths(m int, n int) int {
    dp := make([][]int, m)
    for i, _ := range dp {
        dp[i] = make([]int, n)
    }
    for i := 0; i < m; i++{
        dp[i][0] = 1
    }
    for i := 0; i < n; i++ {
        dp[0][i] = 1
    }
    for i := 1; i < m; i++ {
        for j := 1; j < n;j++ {
            dp[i][j] = dp[i-1][j] + dp[i][j-1]
        }
    }
    return dp[m-1][n-1]
}
```

63. 不同路径 II

```
func uniquePathsWithObstacles(obstacleGrid [][]int) int {
    m := len(obstacleGrid)
    n := len(obstacleGrid[0])
    dp := make([][]int, m)
    for i, _ := range dp {
        dp[i] = make([]int, n)
    }
    if obstacleGrid[0][0] == 1 {
        dp[0][0] = 0
    } else {
        dp[0][0] = 1
    }
    for j := 1; j < n; j++ {
        if obstacleGrid[0][j] == 1 {
            dp[0][j] = 0
        } else {
            dp[0][j] = dp[0][j-1]
        }
    }
    for i := 1; i < m; i++ {
        if  obstacleGrid[i][0] == 1 {
            dp[i][0] = 0
        } else {
            dp[i][0] = dp[i-1][0]
        }
    }
    for i := 1; i < m; i++ {
```

```
        for j := 1; j < n; j++ {
            if obstacleGrid[i][j] == 1 {
                dp[i][j] = 0
            } else {
                dp[i][j] = dp[i-1][j]  + dp[i][j-1]
            }
        }
    }
    return dp[m-1][n-1]
}
```

198. 打家劫舍

```
func rob(nums []int) int {
    if len(nums) == 0 {return 0}
    n := len(nums)
    dp := make([][]int, n)
    for i := 0; i < len(dp); i++ {
        dp[i] = make([]int, 2)
    }
    dp[0][0] = 0
    dp[0][1] = nums[0]
    for i := 1; i < n; i++ {
        dp[i][0] = int(math.Max(float64(dp[i-1][0]), float64(dp[i-1][1])))
        dp[i][1] = dp[i-1][0] + nums[i]
    }
    return int(math.Max(float64(dp[n-1][0]), float64(dp[n-1][1])))
}
```

213. 打家劫舍 II

```
func rob(nums []int) int {
    n := len(nums)
    if n == 1 {return nums[0]}
    if n == 2 {return int(math.Max(float64(nums[0]), float64(nums[1])))}
    max1 := rob_dp(nums, 1, n-1)
    max2 := nums[0] + rob_dp(nums, 2, n-2)
    return int(math.Max(float64(max1), float64(max2)))
}

func rob_dp(nums []int, p, r int) int {
    n := len(nums)
    dp := make([][]int, n)
    for i := 0; i < len(dp); i++ {
        dp[i] = make([]int, 2)
    }
    dp[p][0] = 0
    dp[p][1] = nums[p]
    for i := p+1; i <= r; i++ {
        dp[i][0] = int(math.Max(float64(dp[i-1][0]), float64(dp[i-1][1])))
        dp[i][1] = dp[i-1][0] + nums[i]
    }
    return int(math.Max(float64(dp[r][0]), float64(dp[r][1])))
}
```

337. 打家劫舍 III

```
func rob(root *TreeNode) int {
    money := postorder(root)
```

```
    return int(math.Max(float64(money[0]), float64(money[1])))
}

func postorder(root *TreeNode) []int {
    if root == nil {return []int{0, 0}}
    leftMoney := postorder(root.Left)
    rightMoney := postorder(root.Right)
    money := make([]int, 2)
    money[0] = int(math.Max(float64(leftMoney[0]), float64(leftMoney[1]))) +
int(math.Max(float64(rightMoney[0]), float64(rightMoney[1])))
    money[1] = (leftMoney[0] + rightMoney[0]) + root.Val
    return money
}
```

714. 买卖股票的最佳时机含手续

```
func maxProfit(prices []int, fee int) int {
    n := len(prices)
    dp := make([][]int, n)
    for i := 0; i < len(dp); i++ {
        dp[i] = make([]int, 2)
    }
    dp[0][0] = -prices[0] //第 i 天持有股票
    dp[0][1] = 0 //第 i 天不持有股票
    for i := 1; i < n; i++ {
        dp[i][0] = int(math.Max(float64(dp[i-1][0]), float64(dp[i-1][1]-prices[i])))
        dp[i][1] = int(math.Max(float64(dp[i-1][0]+prices[i]-fee), float64(dp[i-
1][1])))
    }
    return int(math.Max(float64(dp[n-1][0]), float64(dp[n-1][1])))
}
```

309. 最佳买卖股票时机含冷冻期

```
func maxProfit(prices []int) int {
    if len(prices) == 0 {return 0}
    n := len(prices)
    dp := make([][]int, n)
    for i := 0; i < len(dp); i++ {
        dp[i] = make([]int, 4)
    }
    dp[0][0] = -prices[0]
    dp[0][1] = 0
    dp[0][2] = 0
    dp[0][3] = 0
    for i := 1; i < n; i++ {
        dp[i][0] = max3(dp[i-1][0], dp[i-1][2]-prices[i], dp[i-1][3]-prices[i])
        dp[i][1] = dp[i-1][0]+prices[i]
        dp[i][2] = dp[i-1][1]
        dp[i][3] = int(math.Max(float64(dp[i-1][2]), float64(dp[i-1][3])))
    }
    return max4(dp[n-1][0], dp[n-1][1], dp[n-1][2], dp[n-1][3])
}

func max3(a, b, c int) int{
    max := a
    if b > max {
        max = b
    }
```

```
    if c > max {
        max = c
    }
    return max
}

func max4(a, b, c, d int) int{
    max := a
    if b > max {
        max = b
    }
    if c > max {
        max = c
    }
    if d > max {
        max = d
    }
    return max
}
```