70. 爬楼梯

```
func climbStairs(n int) int {
    if n <= 2 {return n}
    dp := make([]int, n+1)
    dp[1] = 1
    dp[2] = 2
    for i := 3; i <= n; i++ {
        dp[i] = dp[i-1] + dp[i-2]
    }
    return dp[n]
}
```

```
func climbStairs(n int) int {
    dp := make([]int, n+1)
    dp[0] = 1
    for i := 1; i <= n; i++ {
        if i-1 >= 0 {dp[i] += dp[i-1]}
        if i-2 >= 0 {dp[i] += dp[i-2]}
    }
    return dp[n]
}
```

322. 零钱兑换

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

剑指 Offer 14- I. 剪绳子

```
func cuttingRope(n int) int {
    if n == 1 {return 1}
    if n == 2 {return 1}
    if n == 3 {return 2}
    dp := make([]int, n+1)
    dp[0] = 1
    for i := 1; i <= n; i++ {
        for j := 1; j <= i; j++ {
            if dp[i] < j * dp[i-j] {
                dp[i] = j*dp[i-j]
            }
        }
```

```
    }
    return dp[n]
}
```

剑指 Offer 46. 把数字翻译成字符串

```
func translateNum(num int) int {
    if num <= 9 {return 1}
    //把十进制数转化成数字数组
    digitlist := make([]int, 0)
    for num != 0 {
        digitlist = append(digitlist, num%10)
        num/=10
    }
    n := len(digitlist)
    digits := make([]int, n)
    for i := 0; i < n; i++ {
        digits[i] = digitlist[n-i-1]
    }
    dp := make([]int, n+1)
    dp[0] = 1
    for i := 1; i <= n ; i++{
        dp[i] = dp[i-1]
        if i-2 >= 0 && isValid(digits[i-2], digits[i-1]) {
            dp[i] += dp[i-2]
        }
    }
    return dp[n]
}

func isValid(a, b int) bool{
    if a == 1 {return true}
    if a == 2 && b >= 0 && b <= 5 {return true}
    return false
}
```

139. 单词拆分

```
func wordBreak(s string, wordDict []string) bool {
    n := len(s)
    dp := make([]bool, n+1)
    dp[0] = true
    for i := 1; i <= n; i++ {
        for _, word := range wordDict {
            len := len(word)
            startp := i-len

            if startp >= 0 && startsWith(s, word, startp) && dp[i-len] {
                dp[i] = true
                break
            }
        }
    }
    return dp[n]
}

func startsWith(s, word string, start int) bool{
    return s[start:start+len(word)] == word
}
```

1143. 最长公共子序列

```
func longestCommonSubsequence(text1 string, text2 string) int {
    n := len(text1)
    m := len(text2)
    t1 := []byte(text1)
    t2 := []byte(text2)
    dp := make([][]int, n+1)
    for i := 0; i < len(dp); i++ {
        dp[i] = make([]int, m+1)
    }
    for j := 0; j <= m; j++ {
        dp[0][j] = 0
    }
    for i := 0; i <= n; i++ {
        dp[i][0] = 0
    }
    for i := 1; i <= n; i++ {
        for j := 1; j <= m; j++ {
            if t1[i-1] == t2[j-1] {
                dp[i][j] = max3(dp[i-1][j-1]+1, dp[i-1][j], dp[i][j-1])
            } else {
                dp[i][j] = max3(dp[i-1][j-1], dp[i-1][j], dp[i][j-1])
            }
        }
    }
    return dp[n][m]
}

func max3(a, b, c int) int{
    maxv := a
    if maxv < b {maxv = b}
    if maxv < c {maxv = c}
    return maxv
}
```

72. 编辑距离

```
func minDistance(word1 string, word2 string) int {
    n := len(word1)
    m := len(word2)
    if n == 0 {return m}
    if m == 0 {return n}
    w1 := []byte(word1)
    w2 := []byte(word2)
    dp := make([][]int, n+1)
    for i := 0; i < len(dp); i++ {
        dp[i] = make([]int, m+1)
    }
    for j := 0; j <= m; j++ {
        dp[0][j] = j
    }
    for i := 0; i <= n; i++ {
        dp[i][0] = i
    }
    for i := 1; i <= n; i++ {
        for j := 1; j <= m; j++ {
            if w1[i-1] == w2[j-1] {
```

```
                dp[i][j] = min3(dp[i-1][j]+1,  dp[i][j-1]+1, dp[i-1][j-1])
            } else {
                dp[i][j] = min3(dp[i-1][j]+1,  dp[i][j-1]+1, dp[i-1][j-1]+1)
            }
        }
    }
    return dp[n][m]
}

func min3(n1, n2, n3 int) int{
    return int(math.Min(float64(n1), math.Min(float64(n2), float64(n3))))
}
```

300. 最长递增子序列

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

//解法 2
func lengthOfLIS2(nums []int) int {
    n := len(nums)
    lisToMinV := make([]int, n+1)
    k := 0
    dp := make([]int, n)
    for i := 0; i < n; i++ {
        len := bsearch(lisToMinV, k, nums[i])
        if len == -1 {
            dp[i] = 1
        } else {
            dp[i] = len+1
        }
        if dp[i] > k {
            k = dp[i]
            lisToMinV[dp[i]] = nums[i]
        } else if lisToMinV[dp[i]] > nums[i] {
            lisToMinV[dp[i]] = nums[i]
        }
    }
    result := 0
    for i := 0; i < n; i++ {
        if dp[i] > result {result = dp[i]}
    }
    return result
}
```

```go
//查找最后一个比 target 小的元素位置
func bsearch(a []int, k, target int) int {
    low := 1
    high := k
    for low <= high {
        mid := (low+high)/2
        if a[mid]<target {
            if mid == k || a[mid+1] >= target {
                return mid
            } else {
                low = mid + 1
            }
        } else {
            high = mid-1
        }
    }
    return -1
}
```