***11.12 极小化最大问题***

猜数字大小：[1,n]中间有一个数需要去猜出，但是每次猜数字x 时，都需要花费x；给定n，不管需要猜的数是多少，我们需要求出最小的花费来确保能猜到这个数字。

题目解释：最小最大问题，不管需要搜索到哪个数字，在所有的搜索方法中找出最小的现金数目。

解题思路：动态规划问题，F(i,j)表示从i到j猜数字所花费的最小，目标F(1,n)

将F(1,n) 划分为F(1,n) = max{ F(1,x-1) + F(x+1,n)} +x 其中x从1 到n

对F( i , j ) = max{F(i, k-1),F(k+1,j)} +k

代码框架如下：

for i=n-1; n>=1; n--

for j=i+1; j<=n; j++

for k=i ; i<j; i++

minCost = min(minCost, max{dp[i][k-1], dp[k+1][j]} + k )

dp[i][j] = minCost

***11.13 字典树问题，找前缀！***

问题描述，实现一个字典键值对，实现insert可以插入字典键值对，实现sum可以找出key中含特定字符串对应的val总和。

因为要遍历寻找所有字符串的前缀情况，用字典树实现最好，go的字典树实现：

type TrieNode struct{

children [26]\*TrieNode

val int

}

type RootNode struct{  
 root \*TrieNode

}

func Constructor()RootNode{

return RootNode{&TrieNode{}}  
}

1. insert实现（字典树插入实现）：

func insert(…){

…

node = this.root

for \_,char :=range key{

if node.children[char-'26']==nil{

//如果node对应的子树为空，代表该路径还未有过，

//则新建节点，

node.children[char-'26'] = &TrieNode{}

}

//移动node到下一层的节点，并为其赋值

node = node.children[char-'26']

node.val += val

}

}

1. sum实现（字典树搜索实现）：

func find(…){  
 node = this.root

for \_,char:=range key{

if node.children[char-'26']==nil{

return 0

}

node = node.children[char-'26']

}

return node.val  
}

11.15灯泡问题

题目描述：每过一轮，序号为i的倍数的灯泡切换开关状态。总共n个灯泡，总共n轮，求问n轮后几个灯泡是亮的

问题转换：每一轮中，序号为伦次倍数的灯泡切换，最终切换奇次的灯泡亮灯。问题转换为n个灯泡中约数为奇数的个数。

再者，由于约数成对出现，所以只有平方数的约数为奇数，所以问题转换为求n以内平方数的个数。

11.18字符串比较相同字符（位运算问题）

题目描述，给定字符串数组,找出数组中两个最长字符串，且两个字符串没有相同字符。

问题转换：由于只有小写字母（题目中声明的，如果不只小写字母，需要考虑转反码的方式），总共26个字符，每个字符串可以用一个int型（64位）的低26位来表示各个字母的出现情况,最后用一个切片来表示各个字符串的字母出现情况：具体代码如下：

listStr := make([ ]int, len(words))

for str :=range words{

tmp :=0

for ch :=range str{

tmp |= (1<<(ch-‘a’))

//这个或|用于将所有字符的存在情况进行汇总

}

}

统计完成后用一个双for循环寻找max (len(str1),str(2)) && listStr[str1]&listStr[str2] == 0

// 代表两个字符串没有公共字母

2.25

**1.有序数组查找两数之和**

（1）笨办法：使用二分查找

for val:= range nums{

find(nums,target-val)

}使用二分查找解决

（2）好方法：借用二分框架，i=0; j= len(nums) -1;

i++和j—来缩小搜索区间，直接一次遍历即可。

二分查找注意点1： l<=r 查找区间为闭区间[ l,r] 如果不带等号会漏掉[2,2]的情况

二分查找注意点2：使用搜索区间的方法：[l ,r ]判断区间开闭

二分查找注意点3：寻找左边界的二分：使用区间[ l,r） for l<r ；==target :mid = right

最后返回left. 最后left == len(nums) 代表没有target

二分查找注意点3：寻找右边界的二分： for l<r ；==target :mid =left + 1 ,最后返回right-1, 最后right == 0 代表没有target.

2.两数之和：、

考虑使用hashmap ，map[val]index 的形式， 一遍循环，判断map[target -val]是否有值

3. 前缀和，求数组中子数组和为定值的情况。

for {  
 sum[i+1] = val+sum[i]

for j=i; j>0; j--{  
 if target[ j ] -sum[ i ] == k

res++

}

}

2.26 三数之和

func threeSum(nums []int) [][]int {

    sort.Ints(nums)// 排序

    for i\_1:=0;i\_1<n;i\_1++{ //一层循环

        if i\_1>0&&nums[i\_1-1]==nums[i\_1] //跳过相同的

            continue

        i\_3:=n-1

        for i\_2:=i\_1+1;i\_2<n;i\_2++ //二层循环[i1,n]

            if i\_2>i\_1+1&&nums[i\_2-1]==nums[i\_2]{ //跳过相同的

                continue

            for i\_2<i\_3&&nums[i\_1] + nums[i\_2] > -nums[i\_3]

                    i\_3-- //三层循环 i\_2<i\_3且和<target

            if i\_3==i\_2 //i\_2 == i\_3时结束

                break

            if nums[i\_1] + nums[i\_2] == -nums[i\_3]

                res = append(res,[]int{nums[i\_1],nums[i\_2],nums[i\_3]})

    return res

最大子数组和

[-2,1,-3,4,-1,2,1,-5,4]

func maxSubArray(nums []int) int {

    var res = nums[0]

    var pre = nums[0]

for index,val:=range nums{

        if val <0{

            pre = res +val

        }else{

            res = max(res,pre+val)

            if pre<0{

                pre = val

            }

        }

    }

    return res

}

**动态规划问题总结**

1. 分割等子集（求定值的0-1背包问题）

dp[i][j] 代表nums中前i个数能够凑齐j 与否 bool值

i from[ 1, n-1] j from[1,sum/2 ]

dp[i][j] = dp[i-1][j] ||dp[i][j-nums[i]] j>=nums[i]

dp[i-1][j] j<nums[i]

dp[k][0] = true而dp[0][nums[0]] =true 因为无论nums如何，总能实现和为0的情况。

改进：

由于每一行的dp之和上一行的dp有关，可以简化为一维数组：

dp[ j] = dp[ j] || dp[j-nums[i]] 左边的是这一行dp,右边是上一行的dp

且第二层循环需要从右到左，防止右边不是上一行的值。

1. 零钱兑换：

for i [ 1, amount]:

dp[ i ] = amount +1

for coin in coins:

dp[ i] = min(dp[i], dp[i-coin]+1)

1. 01背包

转移方程：dp[i][V] = max(dp[i-1][V] , dp[i-1][V-c[i]]+w[i])

循环情况：

外层： for i [ 0,N]

内层： for v [ 0,V ]

如果要优化空间，每次内循环用同一个数组保存当前 i件物品下的各种容量对应的最大价值。v [0,V]是从左到右的，可以用v [V,0]来保证 求解转移方程时，方程左边的变量是上一次外循，也即i-1的值

内层循环优化为下：

for v [V，0]

f[v] = max(f[v],f[v-c[i]]+w[i])

初始化状态设置：

1. 背包必须装满，f[0]=0，其余为MinInt （2）背包可以不满：f[0,n]=0

理解：必须装满，只有容量0的背包满足能满足要求。不装满则都满足。

栈相关  
1.下一个更大元素

对nums，建立单调栈，每个元素入栈前，将栈内小于该元素的元素全部出栈并建立键值对hashmap,以此保证栈内始终单调有序。

如果是求与下一个更大元素的距离，则改成序号入栈即可。