常用代码总结

----------数据结构----------

【栈】

List.append()

List.pop()

【队列】

```
from collections import deque
queue = deque()
```

入队： `queue.append()`

出队： `v = queue.popleft()`

【双端队列】

```
class Deque_on_list:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return self.items == []
    def addFront(self, item):
        self.items.append(item)
    def addRear(self, item):     # 0 位置当尾巴
        self.items.insert(0, item)
    def removeFront(self):
        return self.items.pop()
    def removeRear(self):
        return self.items.pop(0)
    def size(self):
        return len(self.items)
```

【优先队列】

```
import heapq
class PriorityQueue:

  def __init__(self):
    self._queue = []
    self._index =0

  def push(self, item, priority):
    # 传入两个参数，一个是存放元素的数组，另一个是要存储的元素，这里是一个元组。
    # 由于 heap 内部默认有小到大排，所以对 priority 取负数
    heapq.heappush(self._queue, (-priority, self._index, item))
    self._index += 1

  def pop(self):
    return heapq.heappop(self._queue)[-1]
```

----------算法----------

【二分查找】

```
def bsearch(nums, target):
    left = 0
    right = len(nums) - 1
    while left <= right:
        mid = (right + left) // 2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1 – left
```

【KMP 算法】

```
def partial(pattern):
    ret = [0,0]
    for i in range(1, len(pattern)):
        j = ret[i]
        while j > 0 and pattern[j] != pattern[i]:
            j = ret[j]
        ret.append(j+1 if pattern[j] == pattern[i] else 0)
return ret

def indexKMP(S, P, pos=0):
    i=0                             #P 的读写头
    j=pos                           #S 的读写头
    part=partial(P)                 #计算 P 的 partial
    while i<len(P) and j<len(S):
        if P[i] == S[j]:        #两个读写头下的字符相等
            i += 1
            j += 1
        else:                       #不等
            if i == 0:
                j += 1
            else:
                i = part[i]
    else:
        if i == len(P):         #找到了一个匹配
            return j-i
        else:
            return None
```

----------排序----------

【快速排序】

```
def qsort(nums: [int]) -> [int]:
    def recursion(begin, end):
        if end - begin <= 1:
            return
        mid = (begin + end) // 2
        pivot = nums[mid]
        nums[begin], nums[mid] = nums[mid], nums[begin]
        i = begin + 1
        j = end - 1
        while i <= j:
            while i < end and nums[i] < pivot:
                i += 1
            while j >= begin + 1 and nums[j] >= pivot:
                j -= 1
            if i < j:
                nums[i], nums[j] = nums[j], nums[i]
        nums[j], nums[begin] = nums[begin], nums[j]
        recursion(begin, j)
        recursion(j + 1, end)
    recursion(0, len(nums))
    return nums
//另一个版本快排
def qsort(nums: [int]) -> [int]:
    if len(nums)==0 or len(nums)==1:
        return nums
    else:
        qsortHelper(nums,0,len(nums)-1)
    return nums

def qsortHelper(nums,first,last):
    if first<last:
        splitpoint=partition(nums,first,last)
        qsortHelper(nums,first,splitpoint-1)
        qsortHelper(nums,splitpoint+1,last)

def partition(nums,first,last):
    median=(first+last+1)//2
    if nums[median]<nums[first]:
        temp=nums[median]
        nums[median]=nums[first]
        nums[first]=temp
    if nums[first]>nums[last]:
```

```
            temp=nums[last]
            nums[last]=nums[first]
            nums[first]=temp
        if nums[median]>nums[last]:
            temp=nums[last]
            nums[last]=nums[first+1]
            nums[first+1]=temp

        temp=nums[first+1]
        nums[first+1]=nums[median]
        nums[median]=temp

        pivotvalue=nums[first+1]
        leftmark=first+2
        rightmark=last
        done=False
        while not done:
            while leftmark<=rightmark and nums[leftmark]<=pivotvalue:
                leftmark+=1
            while rightmark>=leftmark and nums[rightmark]>=pivotvalue:
                rightmark-=1
            if rightmark<leftmark:
                done=True
            else:
                temp=nums[leftmark]
                nums[leftmark]=nums[rightmark]
                nums[rightmark]=temp

        nums[first+1]=nums[rightmark]
        nums[rightmark]=pivotvalue

        return rightmark
```

【归并排序】

```
def merge_sort(data_list):
    if len(data_list)<=1:
        return data_list
    middle=int(len(data_list)/2)
    left=merge_sort(data_list[:middle])
    right=merge_sort(data_list[middle:])
    merged=[]
    while left and right:
        merged.append(left.pop(0) if left[0]<=right[0] else right.pop(0))
    merged.extend(right if right else left)
```

```
    return merged
```

【桶排序】

```
def BucketSort_counter(alist, ceiling, key=lambda x:x):   #key 的取值范围是[0,ceiling)
    blist = [None]*len(alist)                #临时数组
    count = [0]*ceiling                      #初始化计数器
    for i in alist:
        count[key(i)] += 1                   #统计每个 key 出现的次数
    #print(count)
    for i in range(1, len(count)):
        count[i] += count[i-1]               #统计累计计数的 key 次数(<=key)
                                             #其实就是对应元素应该的排位
    #print(count)
    for i in range(len(blist)-1, -1, -1):    #从尾部开始保持稳定性
        count[key(alist[i])] -= 1
        blist[count[key(alist[i])]] = alist[i]          #重新排位
    return blist

def BucketSort_container(alist, ceiling, key=lambda x:x):   #key 的取值范围是[0,ceiling)
    #container = [[]]*ceiling                #为什么这样不行？
    container = [ [] for _ in range(ceiling)]
    for i in alist:
        container[key(i)].append(i)     #分配
    blist = []
    for bucket in container:
        blist.extend(bucket)                #回收
    return blist
```

【基数排序】

```
def base_sort(alist, base, code_num, BucketSort = BucketSort_container):
    for i in range(code_num):
        print(alist)
        #从低位开始对每一个排序码，调用 BucketSort()
        alist = BucketSort(alist, base, key=lambda x:x//base**i%base)
    return alist
//另一种基数排序
def radixsort(sortlist):
    l=[]
    main=sortlist
    j=0
    for i in range(10):
        l.append(Queue())
    remain=[]
    while len(remain)<len(main):
```

```
            remain=[]
            for i in main:
                if i>=(10**j):
                    l[i//(10**j)%10].enqueue(i)
                else:
                    remain.append(i)
            main=remain[:]
            for i in range(10):
                while not l[i].isEmpty():
                    main.append(l[i].dequeue())
            j+=1
        return [str(x) for x in main]

print(' '.join(radixsort([int(x) for x in input().split()]
```

----------栈、队列应用----------

【中缀转后缀】

```
prec = {"*":3,"/":3,"+":2,"-":2}
def infixToPostfix(infixexpr):
    opStack = Stack()
    postfixList = []
    tokenList = infixexpr.split()
    for token in tokenList:
        if token in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" \
           or token in "0123456789":   #操作数的处理
            postfixList.append(token)
        elif token == '(':              #标记子表达式开始
            opStack.push(token)
        elif token == ')':              #子表达式结束
            while opStack.peek() != '(':
                postfixList.append(opStack.pop())
            else:
                opStack.pop()           #弹出'('
        else:                           #操作符
            while (not opStack.isEmpty()
                   and opStack.peek() != '('
                   and prec[opStack.peek()] >= prec[token]):
                postfixList.append(opStack.pop())
            opStack.push(token)       #所有操作符都必须进栈等待
    while not opStack.isEmpty():
        postfixList.append(opStack.pop())
    return " ".join(postfixList)

print(infixToPostfix("A + B * 5"))
print(infixToPostfix("( A + B ) * 5"))
```

【中缀转前缀】

```
def infixToPrefix(infixexpr):
    prec={'*':3,'/':3,'+':2,'-':2,')':1}
    opStack=Stack()
    postfixList=[]
    tokenList=infixexpr.split()
    retokenList=reversed(tokenList)
    for token in retokenList:
        if token in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' or token in '0123456789':
            postfixList.append(token)
        elif token==')':
            opStack.push(token)
        elif token=='(':
```

```
                topToken=opStack.pop()
                while topToken!=')':
                    postfixList.append(topToken)
                    topToken=opStack.pop()
            else:
                while (not opStack.isEmpty()) and (prec[opStack.peek()]>prec[token]):
                    postfixList.append(opStack.pop())
                opStack.push(token)

        while not opStack.isEmpty():
            postfixList.append(opStack.pop())
        return ' '.join(reversed(postfixList))

print(infixToPrefix(input()))
```

【热土豆（约瑟夫）】

```
def hotPotato(namelist, num):
    que =Queue()
    for name in namelist:
        que.enqueue(name)
    while que.size() > 1:
        for i in range(num-1):
            que.enqueue(que.dequeue())
        print(que.dequeue())   #杀掉一个
    return que.dequeue()
```

【双端队列回文词】

```
def isPalindromic(str):
    dq = Deque()
    for c in str:
        dq.addFront(c)
    while dq.size()>1:
        if dq.removeFront() != dq.removeRear():
            return False
    else:
        return True
```

【单向栈】

```
class Solution:
    def removeKdigits(self, num: str, k: int) -> str:
        n=len(num)
        stack=[]
        stack.append(int(num[0]))
        delete=0
```

```
for i in num[1:]:
        while stack and stack[-1]>int(i) and delete<k:
                stack.pop()
                delete+=1
        stack.append(int(i))
stack=stack[0:n-k]
return ''.join([str(x) for x in stack]).lstrip('0') or "0"
```

----------输入输出----------

【矩阵输入】

```
rowA,colA=map(int,input().split())
A=[[int(x) for x in input().split()] for i in range(rowA)]
board=[[0]*(m+2)]+[[0]+[int(x) for x in input().split()]+[0] for j in range(n)]+[[0]*(m+2)]
```

【初始化】

```
slots=[[] for _ in len(self.table_size)]
l=[[-1]*(n+2)]+[[-1]+[0]*n+[-1] for j in range(n)]+[[-1]*(n+2)]
A=[[0]*n for i in range(n)]
A=[float('inf')]*n
```

【矩阵输出】

```
for y in range(n):
    print(' '.join([str(x) for x in output[y]]))
```

【矩阵提取个别行列】

```
D=[A[r][j:j+q] for r in range(i,i+p)]
```

【小数格式输出】

```
print('{:.2f}'.format(x))
print('{1} {1} {0}'.format('hello','world'))
```

【进制转换】

```
print('{0:b}'.format(3))
```

b-二进制 d-十进制 o-八进制 x-十六进制

【定义四周的函数】

```
dx=[0,0,1,-1]
dy=[1,-1,0,0]

dx=[0,0,1,1,1,-1,-1,-1]
dy=[1,-1,1,0,-1,1,0,-1]
```

【捕获结束】

```
l=input()
while l!='0 0 0 0 0 0':

    l=input()
```

【不知道何时结束程序】

```
while True:
    try:
        n=int(input())
    except EOFError:
```

break

【排序】
l.sort(key=lambda x:(x[0],-x[1]))
l.sort(key=lambda x: -x[0]/x[1])

【深拷贝】
import copy
board=copy.deepcopy(mat)

【字典】
按照值大小返回键：d_key = sorted(d, key=lambda k: d[k])
d_key = max (d, key=lambda k: d[k])
指定值返回键：list(s.keys())[list(s.values()).index(value)]
Zip 实现值和键的翻转，注意内容只能使用一次：zip(prices.values(), prices.keys())

【位运算符 二进制的相关运算】
~ 按位反 & 按位与 | 按位或 ^ 按位异或 << 左移位 >> 右移位
~x 补码
x&y 都是 1 取 1，此外取 0
x|y 都是 0 取 0，此外取 1
x^y 相同取 0，不同取 1
x<<2 左移 1 位，等于十进制下×2 x>>2 右移 1 位，等于十进制下÷2

----------递归（greedy）----------
贪心算法很多的思路在于要先排序！
l.sort(key=lambda x:(x[0],-x[1]))
l.sort(key=lambda x: -x[0]/x[1])
d_key = sorted(d, key=lambda k: d[k])（字典）
【汉诺塔】递归
```
def move(n, a, b, c):
    if(n == 1):
        print(a,"->",c)
        return
    move(n-1, a, c, b)
    move(1, a, b, c)
    move(n-1, b, a, c)
move(3, "a", "b", "c")

def movetower(height,fromPloe,withPole,toPole):
    if height >=1:
        movetower(height-1,fromPloe,toPole,withPole)
        moveDisk(height,fromPloe,toPole)
        movetower(height-1,withPole,fromPloe,toPole)

def moveDisk(disk,fromPole,toPole):
    print(f'Move disk[{disk}] from {fromPole} to {toPole}')
```
【装箱问题】greedy
```
l=input()
while l!='0 0 0 0 0 0':
    a,b,c,d,e,f=map(int,l.split())
    d1={0:0,1:5,2:3,3:1}
    s=d+e+f-(-c)//4
    b1=max(0,b-5*d-d1[c%4])
    s=s-(-b1)//9-min((-4*b-9*c-16*d-25*e-36*f+36*(s-(-b1)//9)-a),0)//36
    print(s)
    l=input()
```
【greedy】打怪兽-注意边界处理！！！
```
cases=int(input())
for i in range(cases):
    n,m,b=map(int,input().split())
    l=[]
    for j in range(n):
        l.append([int(x) for x in input().split()])
    l.sort(key=lambda x:(x[0],-x[1]))
    k=1
```

```
        b-=l[0][1]
        for j in range(1,n):
            if b<=0:
                break
            if l[j][0]==l[j-1][0]:
                k+=1
            else:
                k=1
            if k<=m:
                b-=l[j][1]
            if j==n-1 and b<=0:
                j+=1
        print(l[j-1][0] if b<=0 else 'alive')
```

【greedy】送外卖

```
n=int(input())
output=[]
for i in range(n):
    m=int(input())
    a=[int(x) for x in input().split()]
    b=[int(x) for x in input().split()]
    l=[([0]*2) for i in range(m)]
    for j in range(m):
        l[j][0]=a[j]
        l[j][1]=b[j]
    l.sort(key=lambda x:x[0],reverse=True)
    t=l[0][0]
    s=0
    for j in range(m-1):
        s+=l[j][1]
        t=min(t,max(s,l[j+1][0]))
    output.append(str(min(t,s+l[m-1][1])))
print('\n'.join(output))
```

【greedy】熄灯（注意保存列表）

```
n,m=map(int,input().split())
l=[int(x) for x in input().split()]
j=0
b=[0]*(n+1)
b[0]=l[0]
for i in range(0,n-1):
    b[i+1]=b[i]+(-1+2*(i%2))*(l[i+1]-l[i])
b[-1]=b[n-1]+(1-2*(n%2))*(m-l[-1])
print(max((b[-1]+m)//2,(m+2*max(b)-b[-1]-1)//2))
```

【greedy，Huffman】剪绳子

```
#CS101 18164
N=int(input())
l=sorted([int(x) for x in input().split()])
ans=0
for i in range(N-1):
    remain=l[0]+l[1]
    ans+=remain
    l.remove(l[0])
    l.remove(l[0])
    l.append(remain)
    l.sort()
print(ans)
```

【质数筛法】

```
l=[True]*1000001
l[0]=l[1]=False
for i in range(1001):
    if l[i]==True:
        for j in range(2*i,1000001,i):
            l[j]=False
n=int(input())
k=[int(x) for x in input().split()]
for i in range(n):
    if k[i]**0.5!=int(k[i]**0.5):
        print('NO')
    else:
        print('YES' if l[int(k[i]**0.5)]==True else 'NO')
```

【直方图最大矩形—单项栈/greedy】

```
class Solution:
    def largestRectangleArea(self, heights: List[int]) -> int:
        n = len(heights)
        left, right = [0] * n, [0] * n

        mono_stack = list()
        for i in range(n):
            while mono_stack and heights[mono_stack[-1]] >= heights[i]:
                mono_stack.pop()
            left[i] = mono_stack[-1] if mono_stack else -1
            mono_stack.append(i)

        mono_stack = list()
```

```
for i in range(n - 1, -1, -1):
    while mono_stack and heights[mono_stack[-1]] >= heights[i]:
        mono_stack.pop()
    right[i] = mono_stack[-1] if mono_stack else n
    mono_stack.append(i)

ans = max((right[i] - left[i] - 1) * heights[i] for i in range(n)) if n > 0 else 0
return ans
```

----------动态规划----------

【找硬币】

```
def dpMC(coinValueList, change, minCoins, coinsUsed):
    for cents in range(change+1):
        # minCoins[less than cents] ==> minCoins[cents]
        ll = [(1+minCoins[cents-c],c) for c in coinValueList if c <= cents]
        minCoins[cents], coinsUsed[cents] = min(ll, key=lambda x:x[0], default=(cents, 1))
    return minCoins[change]
def printCoins(coinsUsed, change):
    coin = change
    while coin > 0:
        thisCoin = coinsUsed[coin]
        print(thisCoin, end=' ')
        coin -= thisCoin
    print()
```

【构造列表式】Sereja and Suffixes

```
n,m=map(int,input().split())
l=[int(x) for x in input().split()]
l.reverse()
s={l[0]}
l2=[1]*n
for i in range(n-1):
    if l[i+1] not in s:
        s.add(l[i+1])
        l2[i+1]=l2[i]+1
    else:
        l2[i+1]=l2[i]
for i in range(m):
    print(l2[n-int(input())])
```

【构造列表式】Ilya and Queries

```
s=input()
n=len(s)
l1=[0]*n
for i in range(n-1):
    if s[i]==s[i+1]:
        l1[i+1]+=1
for i in range(n-2):
    l1[i+2]+=l1[i+1]
m=int(input())
for i in range(m):
    l,r=map(int,input().split())
    print(l1[r-1]-l1[l-1])
```

【剪丝带】完全背包

```
inf = 1e9 + 7
n,a,b,c = map(int,input().split()) dp = [0]+[-inf]*n
for i in range(1,n+1):
    for j in (a,b,c):
if i >= j:
#dp[i] = max(dp[i-j], dp[i-j] + 1, dp[i]) dp[i] = max(dp[i-j] + 1, dp[i])
print(dp[n])
```

【boredom】

```
n=int(input())
l1=[int(x) for x in input().split()]
l2=[0]*(max(l1)+1)
for i in l1:
    l2[i]+=1 f=[0]*(max(l1)+1)
for i in range(max(l1)+1):
    f[i]=max(f[i-1],f[i-2]+i*l2[i])
print(f[max(l1)])
```

【滑雪】

```
r,c=map(int,input().split())
l=[[10001]*(c+2)]+[[10001]+[int(x) for x in input().split()]+[10001] for i in range(r)]+[[10001]*(c+2)]
output=[[0]*(c+2) for i in range(r+2)]
dx=[0,0,-1,1]
dy=[1,-1,0,0]

def dp(i,j):
    if output[i][j]>0:
        return output[i][j]
    for s in range(4):
        if l[i][j]>l[i+dx[s]][j+dy[s]]:
            output[i][j]=max(output[i][j],dp(i+dx[s],j+dy[s])+1)
    return output[i][j]

ans=0
for i in range(1,r+1):
    for j in range(1,c+1):
        ans=max(ans,dp(i,j))
print(ans+1)
```

【最大上升子序列和】

```
n=int(input())
```

```
l=[int(x) for x in input().split()]
s=l[:]
for i in range(1,n):
    for j in range(i):
        if l[i]>l[j]:
            s[i]=max(s[j]+l[i],s[i])
print(max(s))
```

【最长上升子序列】

```
n=int(input())
l=[int(x) for x in input().split()]
s=[1]*n
for i in range(1,n):
    for j in range(i):
        if l[i]>l[j]:
            s[i]=max(s[j]+1,s[i])
print(max(s))
```

【三角形】

```
n=int(input())
l1=[int(x) for x in input().split()]
for i in range(n-1):
    l2=[int(x) for x in input().split()]
    l3=[l2[0]+l1[0]]+[max(l1[i],l1[i+1])+l2[i+1] for i in range(len(l2)-2)]+[l2[-1]+l1[-1]]
    l1=l3
print(max(l1))
```

【组合乘积】

```
T=int(input())
l=[int(x) for x in input().split()]
copyl=l[:]
for i in copyl:
    if T%i!=0 or i==1:
        l.remove(i)
ans=set()
for i in l:
    ans.add(i)
    copyans=list(ans)
    for j in copyans:
        ans.add(i*j)
print('YES' if T in ans else 'NO')
```

【合唱团】（最大上升子序列变体）

```
n=int(input())
```

```
performance=list(input().split())
kd=list(input().split())
k=int(kd[0])
d=int(kd[1])
dp=[[0]*n for _ in range(k)]
dp1=[[0]*n for _ in range(k)]
for j in range(n):
    performance[j]=int(performance[j])
    dp[0][j]=performance[j]
    dp1[0][j]=performance[j]
for i in range(1,k):
    for j in range(n):
        for k in range(max(j-d,0),j):
            dp[i][j]=max(dp[i][j],dp1[i-1][k]*performance[j],dp[i-1][k]*performance[j])
            dp1[i][j]=min(dp1[i][j],dp1[i-1][k]*performance[j],dp[i-1][k]*performance[j])
print(max(dp[-1]))
```

【最长公共子序列】

```
def LCS(string1,string2):
    len1 = len(string1)
    len2 = len(string2)
    res = [[0 for i in range(len1+1)] for j in range(len2+1)]
    for i in range(1,len2+1):
        for j in range(1,len1+1):
            if string2[i-1] == string1[j-1]:
                res[i][j] = res[i-1][j-1]+1
            else:
                res[i][j] = max(res[i-1][j],res[i][j-1])
    return res,res[-1][-1]
print(LCS("helloworld","loop"))
```

【最长公共子串】

```
def LCstring(string1,string2):
    len1 = len(string1)
    len2 = len(string2)
    res = [[0 for i in range(len1+1)] for j in range(len2+1)]
    result = 0
    for i in range(1,len2+1):
        for j in range(1,len1+1):
            if string2[i-1] == string1[j-1]:
                res[i][j] = res[i-1][j-1]+1
                result = max(result,res[i][j])
    return result
print(LCstring("helloworld","loop"))
```

【简单的整数划分】

```
def GPC3(n):
    if n < 0:
        return 0
    dp = [1] + [0]*n
    for num in range(1,n+1):
        for i in range(num,n+1):
            dp[i] += dp[i-num]
    return dp[-1]
```

【01 背包问题】

```
T,M=map(int,input().split())
l=[0]*(T+1)
copyl=l[:]
for i in range(M):
    t,m=map(int,input().split())
    if t<=T:
        for j in range(t,T+1):
            l[j]=max(copyl[j-t]+m,l[j])
        copyl=l[:]
print(l[T])
```

【矩阵转移】最小距离

```
l1='algorithm'
l2='alligator'
m=len(l1)
n=len(l2)
edit=[list(range(0,(n+1)*20,20))]+[[(i+1)*20]+[0]*(n) for i in range(m)]
editProcedure=[['']+[('replicate '+l2[i]) for i in range(n)]]+[['replicate '+l1[i]]+[' ']*(n) for i in
range(m)]
for i in range(2,m+1):
    editProcedure[i][0]=editProcedure[i-1][0]+' '+editProcedure[i][0]
for i in range(2,n+1):
    editProcedure[0][i]=editProcedure[0][i-1]+' '+editProcedure[0][i]
for i in range(1,m+1):
    for j in range(1,n+1):
        if l1[i-1]==l2[j-1]:
            edit[i][j]=edit[i-1][j-1]+5
            editProcedure[i][j]=editProcedure[i-1][j-1]+' replicate '+l1[i-1]
        else:
            edit[i][j]=min(edit[i-1][j]+20,edit[i][j-1]+20)
            if edit[i-1][j]<=edit[i][j-1]:
                editProcedure[i][j]=editProcedure[i-1][j]+' delete '+l1[i-1]
            else:
```

```
                editProcedure[i][j]=editProcedure[i][j-1]+' insert '+l2[j-1]
print('最小编辑距离得分为',edit[i][j])
print('编辑过程为',editProcedure[i][j])
```

【开餐馆】

变形 01 背包，状态方程

f[i] = max(f[i],f[j] + c[i])，其中 j 与 i 距离大于临界值，对于 j 要遍历

【最长回文子串】

```
s=input()
n=len(s)
dp=[[False]*n for _ in range(n)]
maxL=0
mini=0
for i in range(n):
    dp[i][i]=True
    if i>0:
        dp[i-1][i]=(s[i]==s[i-1])
    if dp[i-1][i]==True:
        maxL=1
        mini=i-1
for i in range(n-2):
    for j in range(0,n-2-i):
        if s[j]==s[j+i+2]:
            dp[j][j+2+i]=dp[j+1][j+i+1]
            if dp[j][j+2+i]==True and i+2>maxL:
                maxL=i+2
                mini=j
print(s[mini:mini+maxL+1])
```

【解码方法】

```
class Solution:
    def numDecodings(self, s: str) -> int:
        n = len(s)
        # a = f[i-2], b = f[i-1], c = f[i]
        a, b, c = 0, 1, 0
        for i in range(1, n + 1):
            c = 0
            if s[i - 1] != '0':
                c += b
            if i > 1 and s[i - 2] != '0' and int(s[i-2:i]) <= 26:
                c += a
            a, b = b, c
        return c
```

----------图----------

注意剪枝，每一个节点可能访问多次，必须在最小值更新时才更新

【dfs-池塘】

```
dx=[0,0,1,1,1,-1,-1,-1]
dy=[1,-1,1,0,-1,1,0,-1]
count=0
def dfs(i,j):
    global count
    if l[i][j]!='W':
        return
    l[i][j]='M'
    count+=1
    for s in range(8):
        dfs(i+dx[s],j+dy[s])

T=int(input())
for i in range(T):
    N,M=map(int,input().split())
    l=[[0]*(M+2)]+[[0]+list(input())+[0] for i in range(N)]+[[0]*(M+2)]
    ans=0
    for i in range(1,N+1):
        for j in range(1,M+1):
            count=0
            if l[i][j]=='W':
                dfs(i,j)
                ans=max(ans,count)
    print(ans)
```

【封闭岛屿数量-dfs】

```
n=10
maps=[]
visited=[[-1]*n for _ in range(n)]
for i in range(n):
    maps.append(list(input().split(',')))

dx=[0,0,-1,1]
dy=[1,-1,0,0]

def dfs_mark(x,y):
    maps[x][y]='X'
    visited[x][y]=0
    for i in range(4):
        if x+dx[i]>=0 and x+dx[i]<n and y+dy[i]>=0 and y+dy[i]<n:
            if maps[x+dx[i]][y+dy[i]]=='0' and visited[x+dx[i]][y+dy[i]]==-1:
```

```
                    dfs_mark(x+dx[i],y+dy[i])

count=0
for i in [0,n-1]:
    for j in range(n):
        if maps[i][j]=='0':
            dfs_mark(i,j)
for j in [0,n-1]:
    for i in range(n):
        if maps[i][j]=='0':
            dfs_mark(i,j)
for i in range(n):
    for j in range(n):
        if maps[i][j]=='0':
            dfs_mark(i,j)
            count+=1
print(count)
```

【最小距离—队列实现 bfs】

```
class Queue:
    def __init__(self):
        self.lst = []
        self.head = 0
    def push(self, obj):
        self.lst.append(obj)
    def pop(self):
        self.head += 1
    def top(self):
        return self.lst[self.head]
    def empty(self):
        return (self.head >= len(self.lst))
class Pos:
    def __init__(self, x, y):
        self.x = x
        self.y = y
n = int(input())
maps = [None for i in range(n)] #存储地图
for i in range(n):
    string = list(input())
    maps[i] = string
visit = [[-1 for i in range(n)] for j in range(n)] #存储距离，未访 问则为-1
queue = Queue()
dx = [1, 0, -1, 0]
dy = [0, 1, 0, -1]
def dfs_mark(x, y): #选择一座岛屿作为起点并标记为 “X”
```

```
        maps[x][y] = 'X'
        visit[x][y] = 0
        for i in range(4):
            newx = x + dx[i]
            newy = y + dy[i]
            if newx < n and newx >= 0 and newy < n and newy >= 0:
                if maps[newx][newy]=='1' and visit[newx][newy]==-1:
                    dfs_mark(newx, newy)
                elif maps[newx][newy] == '0' and visit[newx][newy] == -1:
                    queue.push(Pos(newx,newy))
                    visit[newx][newy] = 1 #与起点距离为 1 的点进入搜索队列

breakflag = 0
for i in range(n):
    for j in range(n):
        if maps[i][j] == '1':
            dfs_mark(i, j)
            breakflag = 1
            break
    if breakflag:
        break
breakflag = 0
while not queue.empty(): #从距离为 1 的点开始计算距离，使用队列进行广度优先搜索
    tmp = queue.top()
    x = tmp.x
    y = tmp.y
    queue.pop()
    for i in range(4):
        newx = x + dx[i]
        newy = y + dy[i]
        if newx < n and newx >= 0 and newy < n and newy >= 0 and visit[newx][newy]
== -1:
            queue.push(Pos(newx, newy))
            visit[newx][newy] = visit[x][y] + 1
            if(maps[newx][newy] == '1'):
                print(visit[newx][newy] - 1)
                breakflag = 1
                break
    if breakflag:
        break
```

【仙岛求药（迷宫问题）--bfs】

```
maze=[]  #创建迷宫
visited=[]  #访问过的结点
```

```
dis=[]
nx = [[1, 0], [-1, 0], [0, -1], [0, 1]] #移动范围
n,m=map(int,input().split())   #输入行与列
for i in range(n):
    temp = list(map(str, input()))
    maze.append(temp)
dis = [[float('inf') for i in range(m)] for i in range(n)]
for temp in maze:
    if "@" in temp:
        start=(maze.index(temp),temp.index("@"))
    if "*" in temp:
        end = (maze.index(temp), temp.index("*"))
def bfs():
    dis[start[0]][start[1]] = 0
    q = []
    node = (start[0],start[1])
    q.append(node)
    visited.append(node)
    while len(q)>0:
        point = q.pop(0)
        if (point[0] == end[0] and point[1] == end[1]):     #终点位置
            break
        for i in range(4): #下上左右
            dx = point[0] + nx[i][0]
            dy = point[1] + nx[i][1]
            if (0 <= dx < n and 0 <= dy < m and maze[dx][dy] != "#" and (dx,dy) not in
visited):
                newPoint = (dx, dy)
                visited.append(newPoint)
                q.append(newPoint)
                dis[dx][dy] = dis[point[0]][point[1]] + 1
if __name__ == '__main__':
    bfs()
    if dis[end[0]][end[1]] != float("inf"):
        print(dis[end[0]][end[1]])
    else:
        print(-1)
```

【最大联通子图--bfs】

```
def dfs(graph,node,visited):
    if node!=-1 and node not in visited:
        visited.append(node)
        if node not in graph:
            return visited
```

```
            for nei in graph[node]:
                dfs(graph,nei,visited)
        return visited

graph={}
ids=set()
n=int(input())
for i in range(n):
        l=input().split(' : ')
        ids.add(int(l[0]))
        if int(l[0]) not in graph:
            graph[int(l[0])]=[int(x) for x in l[1].split()]

maxp=0
for i in ids:
        dfs_path=dfs(graph,i,[])
        maxp=max(maxp,len(dfs_path))
print(maxp)
```

最短路径（dfs/bfs，每一个节点可能访问多次，必须在最小值更新时才更新）
2-动态规划（O（n））
3-二分查找
4-栈/DP
5-贪心（优先队列）

----------常见报错原因----------
Runtime Error
①除以零
②数组越界：int a[3]; a[10000000]=10;
③指针越界：int * p; p=(int *)malloc(5 * sizeof(int)); *(p+1000000)=10;
④使用已经释放的空间：int * p; p=(int *)malloc(5 * sizeof(int));free(p); *p=10;
⑤数组开得太大，超出了栈的范围，造成栈溢出：（例如剪枝不充分）
⑥输入数据类型不匹配