http://cs101.openjudge.cn/

print(f'{ans},{res:.1f}')print是可以带sep和end参数的

样例输入：

5

0 0

1 1

2 2

3 3

4 4

使用贪心算法解寒假问题

n = int(input())  
s = [[int(x) for x in input().split()]for \_ in range(n)]  
s.sort(key = lambda x:x[1])  
  
m = s[0][1]  
a = 1  
for i in range(n):  
 if s[i][0] > m:  
 a += 1  
 m = s[i][1]  
  
print(a)

全排列输入

3

输出

1 2 3

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

def dfs(idx, n, used, temp, result):

if idx == n + 1:

result.append(temp[:])

return

for i in range(1, n + 1):

if not used[i]:

temp.append(i)

used[i] = True

dfs(idx + 1, n, used, temp, result)

used[i] = False

temp.pop()

def generate\_permutations(n):

result = []

used = [False] \* (n + 1)

dfs(1, n, used, [], result)

for perm in result:

print(" ".join(map(str, perm)))

n = int(input())

generate\_permutations(n)

递归

maxn = 11hashTable = [False] \* maxn # 当整数i已经在数组 P中时为 true

#@recvizdef increasing\_permutaions(n, prefix=[]):

if len(prefix) == n: # 递归边界，已经处理完排列的1~位

return [prefix]

result = []

for i in range(1, n + 1):

if hashTable[i]:

continue

hashTable[i] = True # 记i已在prefix中

# 把i加入当前排列，处理排列的后续号位

result += increasing\_permutaions(n, prefix + [i])

hashTable[i] = False # 处理完为i的子问题，还原状态

return result

n = int(input())result = increasing\_permutaions(n)for r in result:

print(' '.join(map(str,r)))

Lake counting代码：

import sys

# input = sys.stdin.read

sys.setrecursionlimit(20000)

def dfs(x,y):

field[x][y] = "."

for dx,dy in directions:

nx,ny = x+dx,y+dy

if 0 <= nx < m and 0 <= ny < n and field[nx][ny] == "W":

dfs(nx,ny)

m,n = map(int,input().split())

directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]

field = [list(input()) for \_ in range(m)]

cnt = 0

for i in range(m):

for j in range(n):

if field[i][j] == "W":

dfs(i,j)

cnt += 1

print(cnt)

Bfs代码模板

from collections import deque

def bfs(start, end):

q = deque([(0, start)]) # (step, start)

in\_queue = {start}

while q:

step, front = q.popleft() # 取出队首元素

if front == end:

return step # 返回需要的结果，如：步长、路径等信息

数字操作

从整数1开始，每轮操作可以选择将上轮结果加1或乘2。问至少需要多少轮操作才能达到指定整数。

from collections import deque

def bfs(n):

inq = set()

inq.add(1)

q = deque()

q.append((0, 1))

while q:

step, front = q.popleft()

if front == n:

return step

if front \* 2 <= n and front \* 2 not in inq:

inq.add(front \* 2)

q.append((step + 1, front \* 2))

if front + 1 <= n and front + 1 not in inq:

inq.add(front + 1)

q.append((step + 1, front + 1))

n = int(input())

print(bfs(n))

sy321迷宫最短路径

from collections import deque

# 声明方向变化的数组，代表上下左右移动

MAX\_DIRECTIONS = 4

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

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

def is\_valid\_move(x, y, n, m, maze, in\_queue):

return 0 <= x < n and 0 <= y < m and maze[x][y] == 0 and (x, y) not in in\_queue

def bfs(start\_x, start\_y, n, m, maze):

queue = deque()

queue.append((start\_x, start\_y))

in\_queue = set()

prev = [[(-1, -1)] \* m for \_ in range(n)]

in\_queue.add((start\_x, start\_y))

while queue:

x, y = queue.popleft()

if x == n - 1 and y == m - 1:

return prev

for i in range(MAX\_DIRECTIONS):

next\_x = x + dx[i]

next\_y = y + dy[i]

if is\_valid\_move(next\_x, next\_y, n, m, maze, in\_queue):

prev[next\_x][next\_y] = (x, y)

in\_queue.add((next\_x, next\_y))

queue.append((next\_x, next\_y))

return None

def print\_path(prev, end\_pos):

path = []

while end\_pos != (-1, -1):

path.append(end\_pos)

end\_pos = prev[end\_pos[0]][end\_pos[1]]

path.reverse()

for pos in path:

print(pos[0] + 1, pos[1] + 1)

if \_\_name\_\_ == '\_\_main\_\_':

n, m = map(int, input().split())

maze = [list(map(int, input().split())) for \_ in range(n)]

prev = bfs(0, 0, n, m, maze)

if prev:

print\_path(prev, (n - 1, m - 1))

else:

print("No path found")

、

斐波拉契数列  
def fibonacci(n):

if n == 1 or n == 2:

return 1

else:

return fibonacci(n-1) + fibonacci(n-2)

n = int(input())print(fibonacci(n))

动态规划

def f(n):

if n <= 2:

return 1

if dp[n] != -1:

return dp[n]

else:

dp[n] = f(n-1)+f(n-2)

return dp[n]

dp = [-1]\*21

n = int(input())

ans = []

for \_ in range(n):

num = int(input())

ans.append(f(num))

print('\n'.join(map(str, ans)))

汉诺塔

def moveHanoi(n, from\_rod, to\_rod, mid\_rod):

if n == 0:

return

moveHanoi(n - 1, from\_rod, mid\_rod, to\_rod)

print(f"{from\_rod}->{to\_rod}")

moveHanoi(n - 1, mid\_rod, to\_rod, from\_rod)

n = int(input())

print(2\*\*n - 1)

moveHanoi(n, 'A', 'C', 'B')

圆盘个数增加后需要增加移动步数，如果每次都计算将是很庞大的计算量，所以需要使用DP(Dynamic Programming，动态规划法)求解。

d = [0] \* 15f = [float('inf')] \* 15

d[1] = 1for i in range(2, 13):

d[i] = d[i - 1] \* 2 + 1

f[1] = 1for i in range(2, 13):

for j in range(1, i):

f[i] = min(f[i], f[i - j] \* 2 + d[j])

for i in range(1, 13):

print(f[i]

八皇后

from itertools import permutations

def solve\_n\_queens(n):

solutions = []

cols = range(n)

# 生成每一行皇后位置的排列

for perm in permutations(cols):

# 检查是否有两个皇后在同一对角线上

if n == len(set(perm[i] + i for i in cols)) == len(set(perm[i] - i for i in cols)):

# 如果满足条件，加入解

solutions.append(perm)

return solutions

solutions = solve\_n\_queens(8)

for \_ in range(int(input())):

n = int(input())

queen\_string = ''.join(str(col + 1) for col in solutions[n - 1])

print(queen\_string)

最大连续子序列

n = int(input())

\*a, = map(int, input().split())

dp = [0]\*n

dp[0] = a[0]

for i in range(1, n):

dp[i] = max(dp[i-1]+a[i], a[i])

print(max(dp))

连续上升子序列

n = int(input())

\*b, = map(int, input().split())

dp = [1]\*n

for i in range(1, n):

for j in range(i):

if b[j] < b[i]:

dp[i] = max(dp[i], dp[j]+1)

print(max(dp))

背包问题

样例输入

3 4

3000 2000 1500

4 3 1

样例输出

3500

def knapsack(n, c, w, p):

cell = [[0 for j in range(c+1)]for i in range(n+1)]

for j in range(c+1):

#第0行全部赋值为0，物品编号从1开始.为了下面赋值方便

cell[0][j] = 0

for i in range(1, n+1):

for j in range(1, c+1):

#生成了n\*c有效矩阵，以下公式w[i-1],p[i-1]代表从第一个元素w[0],p[0]开始取。

if j >= w[i-1]:

cell[i][j] = max(cell[i-1][j], p[i-1] + cell[i-1][j - w[i-1]])

else:

cell[i][j] = cell[i-1][j]

return cell

goodsnum, bagsize = map(int, input().split())#goodsnum, bagsize = 3, 4\*value, = map(int, input().split())\*weight, = map(int, input().split())#value, weight = [1500, 3000, 2000], [1, 4, 3] # guitar, stereo, laptop

cell = knapsack(goodsnum, bagsize, weight, value)print(cell[goodsnum][bagsize])

完全背包

n, a, b, c = map(int, input().split())dp = [0]+[float('-inf')]\*n

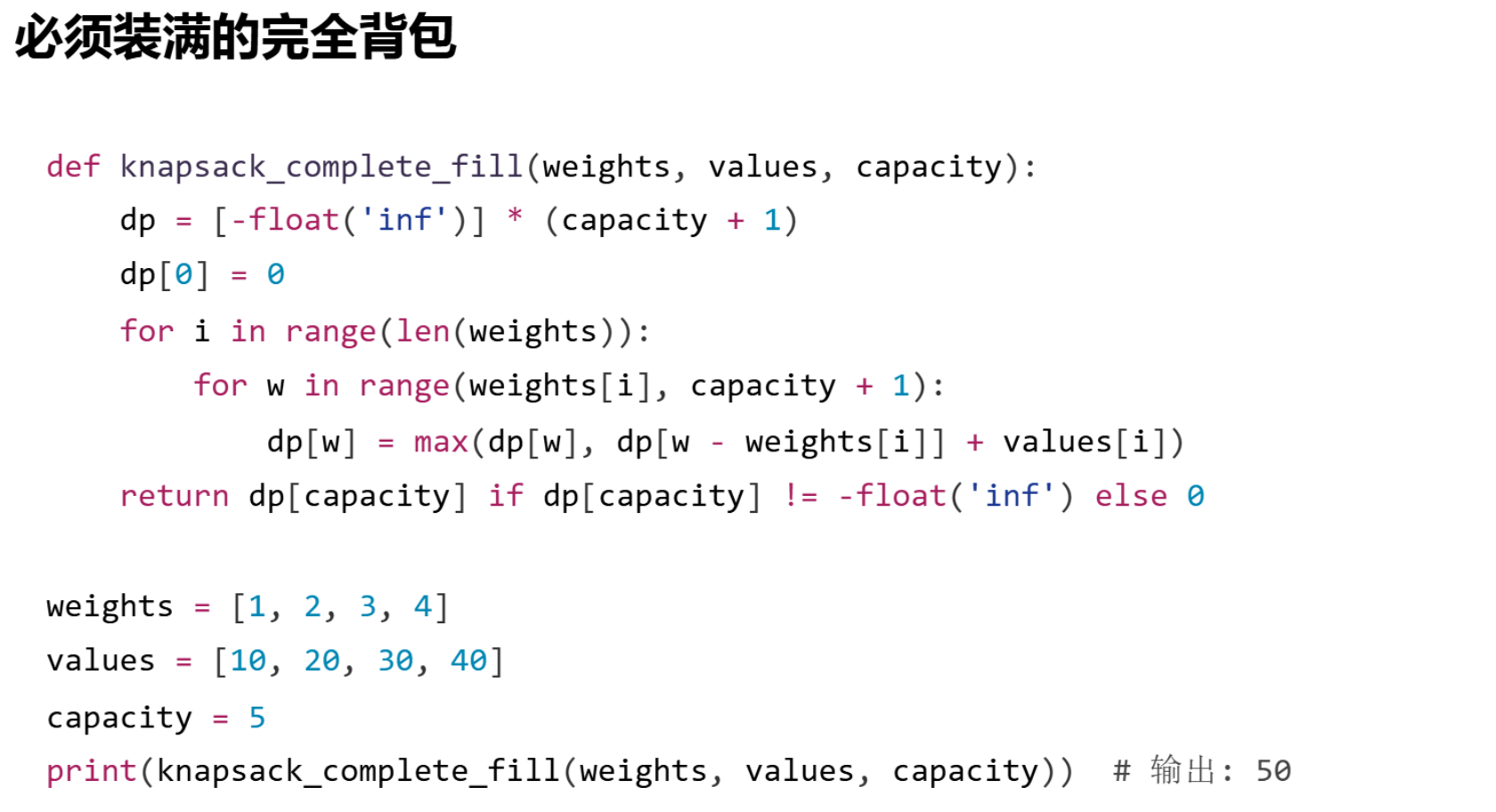
for i in range(1, n+1):

for j in (a, b, c):

if i >= j:

dp[i] = max(dp[i-j] + 1, dp[i])

print(dp[n])





质数：

欧拉筛（线性筛）

def oula(a):

zhishu=[]

zhishu1=[True]\*(a+1)

for i in range(2,a+1):

if zhishu1[i]:

zhishu.append(i)

for h in zhishu:

if h\*i<=a:

zhishu1[h\*i]=False

zhishu=set(zhishu)

return zhishu

dp

最长公共子序列

for i in range(len(A)):

for j in range(len(B)):

if A[i] == B[j]:

dp[i][j] = dp[i-1][j-1]+1

else:

dp[i][j] = max(dp[i-1][j],dp[i][j-1])

最长单调子序列

dp = [1]\*n

for i in range(1,n):

for j in range(i):

if A[j]<A[i]:

dp[i] = max(dp[i],dp[j]+1)

ans = sum(dp)

背包问题

0-1背包

考虑取前i个物品用t时间所能得到的最大值，枚举第i个物品是否取完成转移。注意这里加上时间参数t，因为转移过程中t的限定可能会变。“加参数”是DP问题中最重要的技巧之一。

dp = [0]\*T

for i in range(n):

for t in range(T,time[i]-1,-1):

dp[t] = max(dp[t],dp[t-time[i]]+value[i])

ans = dp[T]

这里采用“滚动数组”的方法将二维数组压缩成一维，是DP问题中常用的技巧。这基于选前i个物品的状态仅依赖于选前i-1个物品的状态。注意内层循环要倒着遍历！

完全背包

将0-1背包中内层循环改为正着遍历即可（这里其实就利用了先前已经得到的信息来简化转移：在先前的转移中物品i可能已经用过若干次了）

多重背包

最简单的思路是将多个同样的物品看成多个不同的物品，从而化为0-1背包。稍作优化：可以改善拆分方式，譬如将m个1拆成x\_1,x\_2,……,x\_t个1，只需要这些x\_i中取若干个的和能组合出1至m即可。最高效的拆分方式是尽可能拆成2的幂，也就是所谓“二进制优化”

dp = [0]\*T

for i in range(n):

all\_num = nums[i]

k = 1

while all\_num>0:

use\_num = min(k,all\_num) #处理最后剩不足2的幂的情形

for t in range(T,use\_num\*time[i]-1,-1):

dp[t] = max(dp[t-use\_num\*time[i]]+use\_num\*value[i],dp[t])

k \*= 2

all\_num -= use\_nume

注：背包问题的DP解法需要时间T不太大，因为要遍历每个可能的T。如果T很大而物品数量很少，采用DFS枚举物品的选法有时是更好的选择。

二分查找模板

bisect\_left:

lo,hi=0,len(a)while lo < hi:

mid = (lo + hi) // 2

if a[mid] < x:

lo = mid + 1

else:

hi = mid

bisect\_right:

lo,hi=0,len(a)while lo < hi:

mid = (lo + hi) // 2

if x < a[mid]:

hi = mid

else:

lo = mid + 1

八皇后

list1 = []  
  
def queen(s):  
 if len(s) == 8:  
 list1.append(s)  
 return  
 for i in range(1, 9):  
 if all(str(i) != s[j] and abs(len(s) - j) != abs(i - int(s[j])) for j in range(len(s))):  
 queen(s + str(i))  
  
queen('')  
samples = int(input())  
for k in range(samples):  
 print(list1[int(input()) - 1])

direction = [(1,0),(-1,0),(0,1),(0,-1)]

def dfs(x,y):

for dx, dy in direction:

global cnt

nx = x + dx

ny = y + dy

if maze[nx][ny] == 'e':

cnt += 1

continue

if maze[nx][ny] == 0:

maze[x][y] = 1

dfs(nx,ny)

maze[x][y] = 0

return

n,m = map(int,input().split())

maze = []

maze.append([-1 for i in range(m+2)])

for \_ in range(n):

maze.append([-1] + [int(\_) for \_ in input().split()] + [-1])

maze.append([-1 for i in range(m+2)])

maze[1][1] = 's'

maze[n][m] = 'e'

cnt = 0

dfs(1,1)

print(cnt)