一. dfs 和 bfs 模板

```
def location(n):
    dp = [[False] * 8 for _ in range(8)]
    answer = []
    def backtrack(row, path):
        if row == 8:
            answer.append(list(path))
            return
        for col in range(8):
            if is_safe(row, col, path):
                dp[row][col] = True
                path.append(col)
                backtrack(row + 1, path)
                path.pop()
                dp[row][col] = False
    def is_safe(row, col, path):
        for i in range(row):
            if dp[i][col] or path[i] == col or abs(row - i) == abs(col - path[i]):
                return False
        return True

    backtrack( row: 0,  path: [])
    answer.sort()
    return answer[n - 1] if n - 1 < len(answer) else []
t=int(input())
for _ in range(t):
    n = int(input())
    result = location(n)
    for i in range(len(result)):
        result[i] = int(result[i]) + 1
    num=""
    for i in range(len(result)):
        num+=str(result[i])
    print(int(num))
```

可以用来 debug 所有和枚举有关的 dfs

```
import heapq
m,n,p=map(int,input().split())
maps=[list(input().split()) for _ in range(m)]
dire=[(0,1),(0,-1),(1,0),(-1,0)]
for _ in range(p):
    sx,sy,ex,ey=map(int,input().split())
    if maps[sx][sy]=="#" or maps[ex][ey]=="#":
        print("NO")
        continue
    in_queue=set()
    ans=[]
    heap=[]
    heapq.heappush(heap, _item: (0,sx,sy))
    in_queue.add((sx,sy,-1))
    while heap:
        energy,x,y=heapq.heappop(heap)
        if x==ex and y==ey:
            ans.append(energy)
        for i in range(4):
            nx=x+dire[i][0]
            ny=y+dire[i][1]
            if 0<=nx<m and 0<=ny<n and maps[nx][ny]!="#" and (nx,ny,i) not in in_queue:
                in_queue.add((nx,ny,i))
                heapq.heappush(heap, _item: (energy+abs(int(maps[nx][ny])-int(maps[x][y])),nx,ny))
    print(min(ans) if ans else "NO")
```

迪杰斯特拉专属 heapq

注意在一般 bfs 的时候不要用

```python
dire=[(0,1),(0,-1),(1,0),(-1,0)]
def dfs(x1,y1,x2,y2,maps,visited):
    if maps[x1][y1]==9 or maps[x2][y2]==9:
        return True
    for i in dire:
        nx1=x1+i[0]
        ny1=y1+i[1]
        nx2=x2+i[0]
        ny2=y2+i[1]
        if 0 <= nx1 < n and 0 <= ny1 < n and 0 <= nx2 < n and 0 <= ny2 < n:
            if maps[nx1][ny1] != 1 and maps[nx2][ny2] != 1:
                if (nx1,ny1,nx2,ny2) not in visited:
                    visited.add((nx1, ny1, nx2, ny2))
                    if dfs(nx1,ny1,nx2,ny2,maps,visited):
                        return True
    return False

n=int(input())
maps = [list(map(int, input().split())) for _ in range(n)]
start = None
for i in range(n):
    for j in range(n):
        if maps[i][j] == 5:
            if start is None:
                start = (i, j, -1, -1)
            else:
                start = (start[0], start[1], i, j)
                break

visited=set()
visited.add(start)
print("yes"if dfs(start[0],start[1],start[2],start[3],maps,visited) else "no")
```

Dfs 的模板，建议在写各种走迷宫的时候一定要加上 visited 这样一个数据集而不是单纯的保护圈，这样就可以避免像这道题目中的同时占据很多个格子以及前面拿到题目中的方向问题

```python
from collections import deque
T=int(input())
dire=[(0,1),(0,-1),(1,0),(-1,0)]
for _ in range(T):
    R,C,K=map(int,input().split())
    maps=[list(input().split()) for _ in range(R)]
    location_start=[]
    location_end=[]
    for i in range(R):
        for j in range(C):
            if maps[i][0][j]=="S":
                location_start.append((i,j))
            elif maps[i][0][j]=="E":
                location_end.append((i,j))
    heap=deque()
    in_queue=set()
    ans=[]
    sx,sy=location_start[0][0],location_start[0][1]
    ex,ey=location_end[0][0],location_end[0][1]
    heap.append((0,sx,sy))
    in_queue.add((sx,sy,0))
    while heap:
        time,x,y=heap.popleft()
        if x==ex and y==ey:
            ans.append(time)
        for i in range(4):
            nx,ny=x+dire[i][0],y+dire[i][1]
            if 0<=nx<R and 0<=ny<C and (nx,ny,(time+1)%K) not in in_queue:
                if maps[nx][0][ny]!="#" or (time+1)%K==0:
                    heap.append((time+1,nx,ny))
                    in_queue.add((nx,ny,(time+1)%K))
    print(min(ans) if ans else "Oop!")
```

其实变换的迷宫就是一个需要考虑事件影响的 bfs，所以在写 bfs 的时候可以不像寻宝那样 step+=1 这个样子，而是直接把时间这个变量带入 deque 之中

通过使用栈来模拟递归，可以避免因递归过深导致的栈溢出问题。

```
def dfs(x, y):
    stack = [(x, y)]
    while stack:
        x, y = stack.pop()
        if field[x][y] != 'W':
            continue
        field[x][y] = '.'  # 标记当前位置为已访问
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < n and 0 <= ny < m and field[nx][ny] == 'W':
                stack.append((nx, ny))

# 读取输入
n, m = map(int, input().split())
field = [list(input()) for _ in range(n)]

# 初始化8个方向
directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1,

# 计数器
cnt = 0

# 遍历地图
for i in range(n):
    for j in range(m):
        if field[i][j] == 'W':
            dfs(i, j)
            cnt += 1

print(cnt)
```

```
import sys
sys.setrecursionlimit(3000)
m,n=map(int,input().split())
maps=[list(input().split()) for _ in range(m)]
dire=[(0,1),(0,-1),(1,0),(-1,0),(1,1),(-1,-1),(1,-1),(-1,1)]
def counting(m,n,maps):
    visited=set()
    lakes=[]
    def lake_counting(x,y,island,visited):
        island.append([x,y])
        visited.add((x,y))
        for i in range(8):
            nx,ny=x+dire[i][0],y+dire[i][1]
            if 0<=nx<m and 0<=ny<n and (nx,ny) not in visited and maps[nx][0][ny]=="W":
                lake_counting(nx,ny,island,visited)
    for i in range(m):
        for j in range(n):
            if maps[i][0][j]=="W" and (i,j) not in visited:
                island=[]
                lake_counting(i,j,island,visited)
                lakes.append(island)
    return(len(lakes))
print(counting(m,n,maps))
```

二. dp 和贪心

这里主要总结一些特殊的 dp 题型

（1）二分查找问题

河中跳房子和 aggressive cow

#47838948提交状态

状态: Accepted

源代码

```
L,n,M=map(int,input().split())
rock=[0]
for _ in range(n):
    D=int(input())
    rock.append(D)
rock.append(L)
def check(x):
    num=0
    now=0
    for i in range(1,n+2):
        if rock[i]-now<x:
            num+=1
        else:
            now=rock[i]
    if num>M:
        return True
    else:
        return False
le,re=0,L+1
ans=-1
while le<re:
    mid=(le+re)//2
    if check(mid):
        re=mid
    else:
        ans=mid
        le=mid+1
print(ans)
```


二分查找的方法解决贪心其实更像是穷举，而二分查找就是找到哪一个合适。

（2）dp 问题中的背包问题

0—1 背包：

```
def max_value(n,b,values,weights):
    dp=[0]*(b+1)
    for i in range(n):
        for w in range(b,weights[i]-1,-1):
            if dp[w]<dp[w-weights[i]]+values[i]:
                dp[w]=dp[w-weights[i]]+values[i]
    return max(dp)
n,b=map(int,input().split())
values=list(map(int,input().split()))
weights=list(map(int,input().split()))
print(max_value(n,b,values,weights))
```

多重背包：

```
N=int(input())
if N%50!=0:
    print("Fail")
    exit()
resting=list(map(int,input().split()))
tickets=[1]*resting[0]+[2]*resting[1]+[5]*resting[2]+[10]*resting[3]+[2(
n=len(tickets)
dp=[0]+[999999]*(N//50)
for i in range(n-1,-1,-1):
    for j in range(N//50,tickets[i]-1,-1):
        dp[j]=min(dp[j],dp[j-tickets[i]]+1)
if dp[N//50]==999999:
    print("Fail")
else:
    print(dp[N//50])
```

背包

以背包容量为 dp 数组长度逆向递归

但是这段代码对于 python3 来说是超时的，所以我们可以采用二进制优化。

```
# 蒋子轩23工学院
# 多重背包中的最优解问题
n = int(input())
if n % 50 != 0:
    print('Fail')
    exit()
n //= 50
nums = list(map(int, input().split()))
price = [1, 2, 5, 10, 20, 50, 100]
dp = [float('inf')] * (n + 1)
dp[0] = 0
for i in range(7):
#for i in range(6, -1, -1):
    cur_price = price[i]
    cur_num = nums[i]
    k = 1
    while cur_num > 0:  #二进制分组优化，时间缩短了将近两个数量级。
                                                    #相同物品避免重复工作，「二进制分组」提高效率
        use_num = min(cur_num, k)
        cur_num -= use_num
        for j in range(n, cur_price * use_num - 1, -1):
            dp[j] = min(dp[j], dp[j - cur_price * use_num] + use_num)
        k *= 2
if dp[-1] == float('inf'):
    print('Fail')
else:
    print(dp[-1])
```

二进制优化的原理：

我们知道任何一个数都可以用 2，4，8，等一系列数表示

所以在每一次的多重背包之中对于每一个元素都按照这个方法讨论其取法

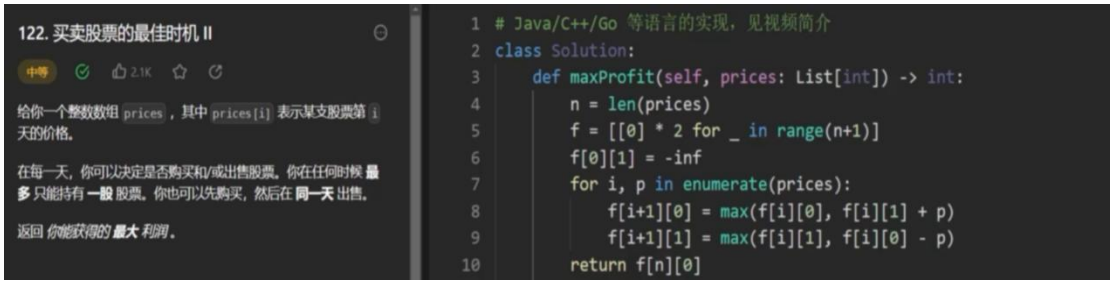
完全背包

```python
n, m = map(int, input().split())
coins = list(map(int, input().split()))
dp = [float("inf")] * (m + 1)
dp[0] = 1
for i in coins:
    dp[i] = 1
for i in range(1, m + 1):
    for coin in coins:
        if i - coin >= 0:
            dp[i] = min(dp[i], dp[i - coin] + 1)

#print(dp)
if dp[m] == float("inf"):
    print(-1)
else:
    print(dp[m])
```

（注：
这里我们不排除考试的时候会在我们到底要输出哪个上面做文章，如果是恰好 k 块钱，那么就输出 dp[k]就可以，如果是至少或者至多。我们要知道，如果 k+-我们背包中最大的量，那么结果就是等差数列因此无意义。所以我们把范围控制在这个区间就可以了）

3. 买卖股票问题——状态机




```python
class Solution:
    def maxProfit(self, k: int, prices: List[int]) -> int:
        n = len(prices)
        f = [[[-inf] * 2 for _ in range(k+2)] for _ in range(n+1)]
        for j in range(1, k+2):
            f[0][j][0] = 0
        for i, p in enumerate(prices):
            for j in range(1, k+2):
                f[i+1][j][0] = max(f[i][j][0], f[i][j-1][1] + p)
                f[i+1][j][1] = max(f[i][j][1], f[i][j][0] - p)
        return f[n][k+1][0]
```

后者是恰好交易 k 次
或许状态机 dp 也是一种双 dp

4. 正难则反：剪绳子

```python
n = int(fin())
import heapq
a = list(map(int, fin().split()))
heapq.heapify(a)
ans = 0
for i in range(n-1):
    x = heapq.heappop(a)
    y = heapq.heappop(a)
    z = x + y
    heapq.heappush(a, z)
    ans += z
print(ans)
```

把问题反过来看，剪绳子变成拼绳子

有 `n` 个气球，编号为`0` 到 `n - 1`，每个气球上都标有一个数字，这些数字存在数组 `nums` 中。

现在要求你戳破所有的气球。戳破第 `i` 个气球，你可以获得 `nums[i - 1] * nums[i] * nums[i + 1]` 枚硬币。 这里的 `i - 1` 和 `i + 1` 代表和 `i` 相邻的两个气球的序号。如果 `i - 1`或 `i + 1` 超出了数组的边界，那么就当它是一个数字为 `1` 的气球。

求所能获得硬币的最大数量。

```python
class Solution:
    def maxCoins(self, nums: List[int]) -> int:
        n = len(nums)
        rec = [[0] * (n + 2) for _ in range(n + 2)]
        val = [1] + nums + [1]

        for i in range(n - 1, -1, -1):
            for j in range(i + 2, n + 2):
                for k in range(i + 1, j):
                    total = val[i] * val[k] * val[j]
                    total += rec[i][k] + rec[k][j]
                    rec[i][j] = max(rec[i][j], total)

        return rec[0][n + 1]
```

这道题目就是再 i，j 之间不断戳。最后就是返回再 0 和 n+1 之间随便戳。

1. **区间问题**:从目标区间反向分解为子区间
2. **路径问题**:从终点反向推导到起点，利用已有结果
3. **子序列问题**:从末尾回溯，逐步构建解

一个路径问题的实例：

```
class Solution:
    def calculateMinimumHP(self, dungeon: List[List[int]]) -> int:
        n, m = len(dungeon), len(dungeon[0])
        BIG = 10**9
        dp = [[BIG] * (m + 1) for _ in range(n + 1)]
        dp[n][m - 1] = dp[n - 1][m] = 1
        for i in range(n - 1, -1, -1):
            for j in range(m - 1, -1, -1):
                minn = min(dp[i + 1][j], dp[i][j + 1])
                dp[i][j] = max(minn - dungeon[i][j], 1)

        return dp[0][0]
```

5. 双 dp

双 dp 感觉就可以看成一种状态机 dp，比如说土豪购物，其中的状态就是仍不扔掉那一个东西。

用状态机和双 dp 写的土豪购物：

```
prices=list(map(int,input().split(",")))
n=len(prices)
f=[[-float('inf')]*(n+1) for _ in range(2)]
for j in range(2):
    f[j][0]=0
for i in range(n):
    f[0][i+1]=max(f[0][i]+prices[i],prices[i])
    f[1][i+1]=max(f[1][i]+prices[i],f[0][i],prices[i])
max_answer=-float("inf")
for i in range(1,n+1):
    max_answer=max(max_answer,f[1][i])
print(max_answer)
```

源代码

```
prices=list(map(int,input().split(",")))
dp1=[0]*len(prices)
dp2=[0]*len(prices)
for i in range(len(prices)):
    dp1[i] = max(dp1[i - 1] + prices[i], prices[i])
    dp2[i] = max(dp1[i - 1], dp2[i - 1] + prices[i], prices[i])
print(max(dp2))
```

值得注意的是这里的状态机其实和前面讲的不太一样，少了一个维度。

原因是如果你丢掉了一个你就不能再丢掉了。

然后另一点就是我们说的“至多”。

6. dp 加上搜索

```
R,C=map(int,input().split())
maps=[list(map(int,input().split())) for i in range(R)]
dire=[(0,1),(0,-1),(1,0),(-1,0)]
dp=[[-1]*C for _ in range(R)]
def dfs(x,y,h):
    if dp[x][y]!=-1:
        return dp[x][y]
    max_path=1
    for i in range(4):
        nx=x+dire[i][0]
        ny=y+dire[i][1]
        if 0<=nx<R and 0<=ny<C and maps[nx][ny]<h:
            if dp[nx][ny]==dp[x][y]+1:
                continue
            else:
                max_path=max(max_path,dfs(nx,ny,maps[nx][ny])+1)
    dp[x][y]=max_path
    return dp[x][y]
max_length=0
for i in range(R):
    for j in range(C):
        max_length=max(max_length,dfs(i,j,maps[i][j]))
print(max_length)
```

注意如何剪枝

三. 语法问题等

Euler筛 得到 `1<=i<=n` 的素数(以列表查找的形式为例,截至 `10 ** 6`)

```
is_prime = [True] * 1000001
for i in range(2, int(1000000 ** 0.5) + 1):  # 只需查找到开根得到的数
    if is_prime[i]: # 如果小的数是素数
        for j in range(i * i, 1000001, i):  #所有i*j(j=i, i+1, ……)都不是素数
            is_prime[j] = False
```

求最大公约数(gcd):辗转相除法(辗转相除直至余数为 `0`)(省事可以 `from math import gcd` , `ans = gcd(a, b)`)

```
def gcd(a, b):
  while b:
    a, b = b, a%b
  return a
```

滑动窗口问题

```python
class Solution:
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
        n = len(nums)
        window = deque()
        ans = []

        for i in range(n):
            while window and nums[window[-1]] < nums[i]:
                window.pop()
            window.append(i)
            while window[0] <= i - k:
                window.popleft()
            if i >= k - 1:
                ans.append(nums[window[0]])
        return ans
```

螺旋矩阵（洋葱类似问题）

```python
n = int(input())
s = [[401]*(n+2)]
mx = s + [[401] + [0]*n + [401] for _ in range(n)] + s

dirL = [[0,1], [1,0], [0,-1], [-1,0]]

row = 1
col = 1
N = 0
drow, dcol = dirL[0]

for j in range(1, n*n+1):
    mx[row][col] = j
    if mx[row+drow][col+dcol]:
        N += 1
        drow, dcol = dirL[N%4]

    row += drow
    col += dcol

for i in range(1, n+1):
    print(' '.join(map(str, mx[i][1:-1])))
```

单调栈问题

```python
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
```

```python
class Solution:
    def trap(self, height: List[int]) -> int:
        stack = []  # 初始化一个空栈，用于存储柱子的索引
        water = 0  # 初始化水的总量为0
        for i in range(len(height)):  # 遍历每个柱子
            while stack and height[i] > height[stack[-1]]:  # 当栈不为空且当前柱子高于栈顶柱子时
                top = stack.pop()  # 弹出栈顶柱子的索引
                if not stack:  # 如果栈为空，说明没有左边的边界，无法形成水坑
                    break
                distance = i - stack[-1] - 1  # 计算左右边界之间的距离
                bounded_height = min(height[i], height[stack[-1]]) - height[top]  # 计算水坑的高度
                water += distance * bounded_height  # 计算水坑的容量，并累加到总水量中
            stack.append(i)  # 将当前柱子的索引压入栈中
        return water  # 返回总的水容量
```

一定要善用字典和集合！！！！

源代码

```python
nCases=int(input())
for _ in range(nCases):
    n,m,b=map(int,input().split())
    skills=[tuple(map(int,input().split())) for _ in range(n)]
    skills.sort()
    skill_t={}
    for i in skills:
        if i[0] not in skill_t:
            skill_t[i[0]]=[i[1]]
        else:
            skill_t[i[0]].append(i[1])
    for i in skill_t:
        skill_t[i]=sorted(skill_t[i],reverse=True)
        b-=sum(skill_t[i][:m])
        if b<=0:
            print(i)
            break
    if b>0:
        print("alive")
```

```
输出：
法一：f"{value:.nf}"  #其中n是希望保留的小数位数
百分数
print(f"Percentage: {percentage:.2%}")  # 输出：85.00%
科学计数法：
print(f"Scientific notation: {large_number:.2e}")  # 输出：1.23e+06
```