Cheat Sheet

基本问题

1.筛法：

埃氏筛：

```
def underp(t):
    ints=[True]*(t+1)
    ints[0]=ints[1]=False
    for x in range(2,int(t**0.5)+1):
        if ints[x]:
            for m in range(x**2,int(t)+1,x):
                ints[m]=False
    return [num for num,prime in enumerate(ints) if prime]
```

2.输入输出方式改造

```
import sys
from math import gcd
input,output=sys.stdin.read,sys.stdout.write
data,ans=input().strip().split("\n"),[]
for lines in data: #也可以直接建立迭代器
    a,b=map(int,lines.split())
    ans.append(gcd(a,b))
output("\n".join(map(str,ans))+"\n")
```

3.全排列函数

```
from itertools import permutations
n=int(input())
a=[i+1 for i in range(n)]
lst=tuple(map(int,input().split()))
result=list(permutations(a))
print(result.index(lst))
```

4.特殊函数语法

find函数/count函数/upper函数

```
s=input().lower()
text=" "+input().lower()+" "
indx=text.find(" "+s+" ")
cnt=text.split().count(s)
if indx==-1:
    print(-1)
else:
    print(cnt,indx)
```

算法问题

1.排序算法：

自定义排序：

```
from functools import cmp_to_key
def compare(x,y):
    if 2*x+1>y:
        return 1 #交换二者的位置
    else:
        return -1
a=[1,2,3]
a.sort(key=cmp_to_key(compare))
```

逆向二重排序：

```
a=[(1,1),(2,3),(2,1),(4,1)]
a.sort(keys=lambda x:(x[0],-x[1]))
```

2.dfs搜索模板

递归写法：

```
import sys
n,m=map(int,input().split())
sys.setrecursionlimit(20000) #更改递归深度，如果不够会报错RE
pond=[["."for _ in range(m+2)] for _ in range(n+2)]
tags=[[True for _ in range(m+2)] for _ in range(n+2)]
for i in range(1,n+1):
    lines=list(str(input()))
    pond[i][1:m+1]=lines
nums=0
def findlakes(x,y):
    tags[x][y]=False #设置标签
    for dirx in [-1,0,1]:
        for diry in [-1,0,1]:
            if tags[x+dirx][y+diry] and pond[x+dirx][y+diry]=="W": #保护圈这里可以直接利用连等号判断
                findlakes(x+dirx,y+diry)
    return
for i in range(1,n+1):
    for j in range(1,m+1):
        if tags[i][j] and pond[i][j]=="W":
            nums+=1
            findlakes(i,j)
print(nums)
```

栈写法：

```
n=int(input())
dir=[(1,0),(0,1)] #设置方向向量
maze=[[1 for _ in range(n+2)]for _ in range(n+2)]
for i in range(1,n+1):
```

```
    maze[i][1:n+1]=list(map(int,input().split()))
stack=[(1,1)]
while stack: #栈非空则有位置可以继续遍历
    x,y=map(int,stack.pop())
    for xmove,ymove in dir:
        if maze[x+xmove][y+ymove]==1:
            continue
        maze[x+xmove][y+ymove]=1
        stack.append((x+xmove,y+ymove))#对于更一般的情形，可以像bfs那样建立一个集合存储所有走过的位置
    if x==y==n:
        print("Yes")
        break
else:
    print("No")
```

3.bfs搜索模板：

```
from collections import deque
direction=[(1,0),(-1,0),(0,1),(0,-1)]
def bfs(chac,stx,sty,edx,edy):
    queue=deque([(stx,sty,chac,0)])
    pathset=set([(stx,sty,chac)])#多维度集合
    while queue:
        xnow,ynow,chacnow,t=queue.popleft()
        for dx,dy in direction:
            xnew=xnow+dx
            ynew=ynow+dy
            if xnew==edx and ynew==edy:
                return t+1
            if 0<=xnew<m and 0<=ynew<n:
                if (xnew,ynew,chacnow) not in pathset:
                    if trace[xnew][ynew]=="#":
                        if chacnow>0:
                            queue.append((xnew,ynew,chacnow-1,t+1))
                            pathset.add((xnew,ynew,chacnow-1))
                    else:
                        queue.append((xnew,ynew,chacnow,t+1))
                        pathset.add((xnew,ynew,chacnow))
```

dijkstra算法

记录距离版本

```
import heapq
direction=[(1,0),(-1,0),(0,1),(0,-1)]
def dijkstra(xs,ys,xe,ye):
    global hill
    queue=[(0,xs,ys)] # 注意距离要放在前面
    distance=[[float("inf") for _ in range(m+2)] for _ in range(n+2)] #初始化每个点到出发点的最小距离
    distance[xs][ys]=0
    while queue:
        dis,xnow,ynow=heapq.heappop(queue) #弹出最小值
        if xnow==xe and ynow==ye:
```

```
            return dis
        for dir in direction:
            xnew=xnow+dir[0]
            ynew=ynow+dir[1] #邻接位置的判断仍旧采用方向向量的方式
            if hill[xnew][ynew]!="#": #判断是否能够到达该位置
                consume=dis+abs(int(hill[xnew][ynew])-int(hill[xnow][ynow]))
                if distance[xnew][ynew]>consume:
                    distance[xnew][ynew]=consume
                    heapq.heappush(queue,(consume,xnew,ynew))
    return "NO"
```

注意集合要在到达后更新/记录其他消费版本

```
def dijkstra(stpoint,citynum,money):
    queue=[(0,stpoint,0)]
    heapq.heapify(queue)
    while queue:
        dis,st,cost=heapq.heappop(queue)
        visited[st-1]=cost
        if st==citynum:
            return dis
        for next,path,expense in graph[st]:
            if expense+cost<=money:
                if visited[next-1]>=expense+cost:
                    heapq.heappush(queue,(dis+path,next,expense+cost))
    return -1
```

4.背包模板

01背包

```
dp,choice=[0]*(v+1),[[] for _ in range(v+1)]
for i in range(n):
    for j in range(v,weight[i]-1,-1):
        cpy=choice[j-weight[i]]+[i+1]
        if dp[j-weight[i]]+prices[i]>dp[j]:
            dp[j]=dp[j-weight[i]]+prices[i]
            choice[j]=cpy
        elif dp[j-weight[i]]+prices[i]==dp[j]:
            if cpy<choice[j]:
                choice[j]=cpy
```

完全背包(无序)（物品在外）

```
dp=[1]+[0]*(num)
for i in pow2:
    if i>num:
        break
    for j in range(i,num+1):
        dp[j]+=dp[j-i]
```

完全背包(有序)+反扣除（调换内外层顺序 位置在外）

```
for i in range(1,n+1):
    for j in range(1,k+1):
        if j>i:
            break
        dp[i]=(dp[i-j]+dp[i])
for i in range(1,n+1):
    for j in range(1,d):
        if j>i:
            break
        dpp[i]=(dpp[i-j]+dpp[i])
```

多维度限制背包（大的做值）

```
dp=[[0]*m]+[[float("inf") for _ in range(m)] for _ in range(k)]
for i in range(1,k+1):
    cost,attack=map(int,input().split())
    for j in range(i,0,-1):
        for s in range(attack,m):
            if dp[j-1][s-attack]+cost<=n:
                dp[j][s]=min(dp[j-1][s-attack]+cost,dp[j][s])
for q in range(k,0,-1):
    for s in range(m):
        if dp[q][s]!=float("inf"):
            print(q,m-s)
```

多重背包最值、可否问题

```
for i in range(m):
        cost=prices[i]
        nums=dic[cost]
        k=1
        while nums>=k:
            nums-=k
            costeff=cost*k
            numseff=k
            for i in range(n,costeff-1,-1):
                if dp[i-costeff]:
                    dp[i]=1
            k*=2
        if nums>0:
            costeff=cost*nums
            numseff=nums
            for i in range(n,costeff-1,-1):
                if dp[i-costeff]:
                    dp[i]=1
```

多重背包计数问题：

```
for count,money in types:
    for j in range(target,0,-1):
        for k in range(1,min(count,j//money)+1):
            dp[j]+=dp[j-money*k]
        dp[j]=dp[j]%(pow(10,9)+7)
return dp[-1]
```

5.其他dp类问题

双dp（土豪购物）

```
dp1=[lst[0]]+[0]*(l-1)
dp2=[max(lst[0],0)]+[0]*(l-1)
for i in range(1,l):
    dp1[i]=max(dp1[i-1]+lst[i],lst[i])
    dp2[i]=max(dp1[i-1],dp1[i],dp2[i-1]+lst[i])
if all(lst[i]<0 for i in range(l)):
    print(max(lst))
else:
    print(max(dp2))
```

不下降子序列（拦截导弹）

```
for i in range(n):#非单调
    p=bisect.bisect_right(dp,-lst[i])
    dp[p]=-lst[i]
pend=bisect.bisect_right(dp,pow(10,8))
```

```
for i in range(n):#单调
    p=bisect.bisect_left(dp,-lst[i])
    dp[p]=-lst[i]
pend=bisect.bisect_right(dp,pow(10,8))
```

回文+编辑距离

```
s=input()
l=len(s)
dp=[[0]*i+[float("inf") for _ in range(l-i)] for i in range(1,l+1)]
for y in range(1,l):
    for pl in range(0,l-y):
        ynew=y+pl
        xnew=pl
        if s[xnew]==s[ynew]:
            dp[xnew][ynew]=dp[xnew+1][ynew-1]#
        else:
            dp[xnew][ynew]=min(dp[xnew+1][ynew],dp[xnew][ynew-1],dp[xnew+1][ynew-
1])+1
print(dp[0][-1])
```

其他综合性dp：最大整数

```
def change(x,y):
    if int(x+y)>int(y+x):
```

```
        return -1
    else:
        return 1
m=int(input())
n=int(input())
lst=list(map(str,input().split()))
lst.sort(key=cmp_to_key(change))
dp=["0"]+[-1]*m
for i in range(n):
    for j in range(m,len(lst[i])-1,-1):
        if dp[j-len(lst[i])]!=-1:
            dp[j]=str(max(int(dp[j-len(lst[i])]+lst[i]),int(dp[j])))
```

6.二分查找模板

应用之一：归并排序

```
def mergesort(x,n):
    if n==1:
        return x,0
    fr=n//2
    lst1,re1=mergesort(x[:fr],fr)
    lst2,re2=mergesort(x[fr:],n-fr)
    sre,re,p1,p2,t=[],0,0,0,re1+re2
    while re<n:
        if p1==fr:
            while p2<n-fr:
                sre.append(lst2[p2])
                p2+=1
            break
        if p2==n-fr:
            while p1<fr:
                sre.append(lst1[p1])
                p1+=1
            break
        if lst2[p2]<lst1[p1]:
            t+=(fr-p1)
            sre.append(lst2[p2])
            p2+=1
        else:
            sre.append(lst1[p1])
            p1+=1
        re+=1
    return sre,t
n=int(input())
lst=list(map(int,input().split()))
x,p=mergesort(lst,n)
print(p)
```

应用之二：月度开销 从答案出发二分

```
import math
def check(put):
    global m
    cnt,bucket=0,1
```

```
    for i in range(n):
        if cost[i]>put:
            return False
        if cnt+cost[i]<=put:
            cnt+=cost[i]
        else:
            cnt=cost[i]
            bucket+=1
    if bucket<=m:
        return True
    else:
        return False
n,m=map(int,input().split())
cost=[]
for _ in range(n):
    cost.append(int(input()))
su=sum(cost)
lim=math.ceil(su/m)
ans=cost[0]
bucket=[0]*m
while lim<=su:
    mid=(lim+su)//2
    ju=check(mid)
    if ju:
        ans=mid
        su=mid-1
    else:
        lim=mid+1
print(ans)
```

标准二分：

```
import bisect
a=[(1,2),(3,1),(4,4),(2,7)]
b=[1,3,3,5,6,7]
print(bisect.bisect_left(a,(2,5),key=lambda x:(x[0],x[1])))#输出1
bisect.insort_left(b,4)
print(b)#输出[1, 3, 3, 4, 5, 6, 7]
print(bisect.bisect_left(b,3))#输出1
print(bisect.bisect_right(b,3))#输出3
```

反序二分/拦截导弹变形：

```
from typing import List
import bisect
class Solution:
    def maxPathLength(self, coordinates: List[List[int]], k: int) -> int:
        target,l=coordinates[k],len(coordinates)
        coordinates.sort(key=lambda x:(x[1],-x[0]))
        idx=coordinates.index(target)
        dp=[pow(10,11)]*idx+[target[0]]+[pow(10,11)]*(l-idx-1)
        for i in range(l):
            if i<idx:
                if coordinates[i][0]<target[0]:
```

```
                    p1=bisect.bisect_left(dp,coordinates[i][0])
                    dp[p1]=coordinates[i][0]
            elif i>idx:
                if coordinates[i][0]>target[0]:
                    p2=bisect.bisect_left(dp,coordinates[i][0])
                    dp[p2]=coordinates[i][0]
        cnt=0
        for x in dp:
            if x!=pow(10,11):
                cnt+=1
        return cnt
```

7.递归模板

全排列

```
def arrange(numlst,path):
    if len(path)==len(numlst):
        print(*path)
        return
    for i in range(len(numlst)):
        if numlst[i]:
            para=numlst[i]
            numlst[i]=False
            path.append(para)
            arrange(numlst,path)
            numlst[i]=para
            path.pop()
n=int(input())
nums=[i for i in range (1,n+1)]
arrange(nums,[])
```

回溯/返回值问题：k个等和子序列

```
lru_cache(maxsize=None)
def recursion(nums,su,k):
    nums=list(nums)
    su=list(su)
    if nums:
        for i in range(k):
            if nums[0]+su[i]<=target:
                su[i]+=nums[0]
                if recursion(tuple(nums[1:]),tuple(su),k):
                    return True
                su[i]-=nums[0]
            else:
                continue
        return False
    else:
        for i in range(k):
            if su[i]!=target:
                return False
        return True
```

n皇后问题：列表append时深拷贝/更新变量时的位置

```
def nqueen(x,num,n):
    if x==0:
        scpy=copy.deepcopy(s)
        res.append(scpy)
        return
    for i in range(n):
        ju=0#注意，每轮要重新初始化，想清楚！
        for snum in range(num):
            if abs(num-snum)==abs(i-s[snum]) or i==s[snum]:
                ju=1
                break
        if ju==0:
            s.append(i)
            nqueen(x-1,num+1,n)
            s.pop()
nqueen(n,0,n)
relst=[]
for x in res:
    for y in range(n):
        queenboard[y][x[y]]="Q"
        queenboard[y]="".join(queenboard[y])
    relst.append(queenboard)
    queenboard=[["." for _ in range(n)]for _ in range(n)]
return relst
```

8.贪心算法

交换类型贪心

```
from functools import cmp_to_key
def swap(a,b):
    if max(a[0]+a[1],a[0]+b[1]+b[0])>max(b[0]+b[1],b[0]+a[1]+a[0]):
        return 1
    else:
        return -1
l,st,ed=len(growTime),-1,0
totalgrow=[(plantTime[i],growTime[i]) for i in range(l)]
totalgrow.sort(key=cmp_to_key(swap))
for i in range(l):
    st+=totalgrow[i][0]
    ed=max(ed,st+totalgrow[i][1]+1)
return ed
```

反悔类贪心

```
stations.sort()
l=len(stations)
stations=[(0,0)]+stations
oil=[(-stations[i][1],stations[i][0]-stations[i-1][0]) for i in range(1,l+1)]+
[(0,target-stations[-1][0])]
move,rem,cnt=[],startFuel,0
for i in range(l+1):
    rem-=oil[i][1]
    while rem<0:
```

```
        if move:
            energy,pos=heapq.heappop(move)
            rem+=-energy
            cnt+=1
        else:
            return -1
    heapq.heappush(move,oil[i])
return cnt
```

队列贪心/栈贪心

```
from collections import deque
n,k=map(int,input().split())
lst=list(map(int,input().split()))
queue=deque()
for i in range(n):
    while queue and queue[0]<=i-k:
        queue.popleft()
    while queue and lst[queue[-1]]<lst[i]:
        queue.pop()
    queue.append(i)
    if i>=k-1:
        print(lst[queue[0]],end=" ")
```

数据结构问题

1.滑动窗口/defaultdict/enumerate

字典确定位置

```
class Solution(object):
    def lengthOfLongestSubstring(self, s):
        from collections import defaultdict
        nearest,st,mx=defaultdict(lambda:-1),0,0 #设定defaultdict的默认值
        for i,char in enumerate(s): #迭代器和元素全部遍历
            if nearest[char]>=st:
                st=nearest[char]+1
            nearest[char]=i
            mx=max(mx,i-st+1) #更新最大值
        return mx
```

滑动窗口最大值：双端队列的应用 单调优先队列

```
queue=deque()
for i in range(n):
    while queue and queue[0]<=i-k:
        queue.popleft()
    while queue and lst[queue[-1]]<lst[i]:
        queue.pop()
    queue.append(i)
    if i>=k-1:
        print(lst[queue[0]],end=" ")
```

2.栈

波兰表达式：从后向前入栈

```
sign=["+","-","*","/"]
lst=list(input().split())
stack,auxstack=[],[]
for i in range(len(lst)-1,-1,-1):
    if lst[i] not in sign:
        stack.append(float(lst[i]))
    else:
        a,b=stack.pop()
        stack.append(eval(str(a)+lst[i]+str(b)))
print(f"{float(stack[0]):.6f}")
```

解密字符串：从前向后入栈，每逢]向前弹弹弹

```
for i in range(l):
    if s[i]=="]":
        while stack[-1]!="[":
            sget.append(stack.pop())
        stack.pop()
        sget=sget[-1::-1]
        while stack and stack[-1] in nums:
            numget+=stack.pop()
        numget=numget[-1::-1]
        sget*=int(numget)
        stack+=sget
        numget,sget="",[]
    else:
        stack.append(s[i])
```

指标栈：计算柱状图中最大子矩形

```
from typing import List
class Solution:
    def largestRectangleArea(self, heights: List[int]) -> int:
        l=len(heights)
        stack,mxarea=[],0
        for i in range(l):
            while stack and heights[stack[-1]]>heights[i]:
                h=heights[stack.pop()]
                w=i if not stack else i-1-stack[-1]
                mxarea=max(mxarea,h*w)
            stack.append(i)
        while stack:
            h=heights[stack.pop()]
            w=l if not stack else l-1-stack[-1]
            mxarea=max(mxarea,h*w)
        return mxarea
```

3.集合

利用排异性 最短的愉悦旋律

```
n,m=map(int,input().split())
notes=list(map(int,input().split()))
lyrics,cnt=set(),1
for i in range(n):
    lyrics.add(notes[i])
    if len(lyrics)==m:
        cnt+=1
        lyrics.clear()
        print(cnt)
```

4.优先队列

滑动窗口最大值

```
from collections import deque
n,m=map(int,input().split())
lst=list(map(int,input().split()))
queue=deque()
for i in range(n):
    while queue and queue[0]<i-m+1:
        queue.popleft()
    while queue and lst[queue[-1]]<lst[i]:
        queue.pop()
    queue.append(i)
    if i>=m-1:
        print(lst[queue[0]],end=" ")
```

5.堆

遇见堆不要死脑筋！有的时候排序也可以解决。

```
from collections import Counter
from typing import List
import heapq
class Solution:
    def topKFrequent(self, words: List[str], k: int) -> List[str]:
        countlst=Counter(words)
        queue=[(nums,item) for item,nums in countlst.items()]
        ktex=heapq.nlargest(k,queue,lambda x:(x[0],x[1]))
        return [y for x,y in ktex]
```

特殊定理以及其他类问题

1.dilworth定理的应用

```
for _ in range(t):
    n=int(input())
    numset,cnt,lstsum=set(),0,[0]*(n+1)
    lst=[0]+list(map(int,input().split()))
    for i in range(1,n+1):
        lstsum[i]=lstsum[i-1]+lst[i]
    for i in range(n+1):
        if lstsum[i] not in numset:
            numset.add(lstsum[i])
        else:
            numset.clear()
            numset.add(lstsum[i])
            cnt+=1
```

2.区间问题

区间选点问题：（倒序排列）

```
proce.sort(key=lambda x:(x[1],x[0]))
    edtime,cnt=proce[0][1],1
    for i in range(1,n):
        if proce[i][0]>edtime:
            cnt+=1
            edtime=proce[i][1]
```

区间覆盖问题：

```
clips.sort(key=lambda x:(x[0],-x[1]))
l=len(clips)
st,ed,i,cnt=0,time,0,0
while st<ed and i<l:
    maxrange=0
    while i<l and clips[i][0]<=st:
        maxrange=(max(maxrange,clips[i][1]))
        i+=1
    if maxrange<=st:
        return -1
    st=maxrange
    cnt+=1
    if st>=time:
        break
```

区间分组问题：

```
cows.sort(key=lambda x:(x[0],x[1]))
cnt,feed=1,[0]*n
feed[cows[0][2]]=1
mnqueue=[(cows[0][1],1)]
heapq.heapify(mnqueue)
for j in range(1,n):
    last,idx=heapq.heappop(mnqueue)
    if cows[j][0]>last:
        feed[cows[j][2]]=idx
        heapq.heappush(mnqueue,(cows[j][1],idx))
```

```
    else:
        cnt+=1
        heapq.heappush(mnqueue,(last,idx))
        heapq.heappush(mnqueue,(cows[j][1],cnt))
        feed[cows[j][2]]=cnt
```

3.类约瑟夫问题

```
from typing import List
from collections import deque
class Solution:
    def deckRevealedIncreasing(self, deck: List[int]) -> List[int]:
        deck.sort()
        idx=deque([range(len(deck))])
        ans=[0]*len(deck)
        for x in deck:
            ans[idx.popleft()]=x
            if idx:
                idx.append(idx.popleft())
        return ans
```

约瑟夫问题变式

```
def joseph(m,p, n):
    kids = list(range(1, m + 1))
    pos = p-1  # 从位置0开始（Python中列表的索引是从0开始的）
    order=[]

    while len(kids) > 0:  # 只要剩下一个人，就结束循环
        pos = (pos + n - 1) % len(kids)  # 计算下一个删除的位置
        order.append(kids.pop(pos))  # 删除该位置的元素
    return order  # 返回最后一个剩下的孩子

while True:
    m, p,n = map(int, input().split())
    if m == n ==p== 0:
        break
    print(*joseph(m, p,n),sep=",")
```

4.双指针

计算和为0的三个数的组合个数

```
for i in range(l):
    if nums[i]>0:
        break
    if i>0 and nums[i]==nums[i-1]:
        continue
    p1,p2=i+1,l-1
    while p1<p2:
        while nums[p1]+nums[p2]+nums[i]<0:
            p1+=1
        while nums[p1]+nums[p2]+nums[i]>0:
            p2-=1
```

```
        if p1<p2 and nums[p1]+nums[p2]+nums[i]==0:
            threenum.append([nums[p1],nums[p2],nums[i]])
            p1+=1
            p2-=1
            while p1<p2 and nums[p1]==nums[p1-1]:
                p1+=1
            while p2>p1 and nums[p2]==nums[p2+1]:
                p2-=1
```

5.懒更新

提前建立字典/列表记录

```
import heapq
from collections import defaultdict
n=int(input())
dicleft=defaultdict(int)
dicright=defaultdict(int)
left,right=[],[]
heapq.heapify(left)
heapq.heapify(right)
for _ in range(n):
    s,a,b=input().split()
    a,b=int(a),int(b)
    heapq.heappush(left,-a)
    heapq.heappush(right,b)
    if s=="+":
        dicleft[a]+=1
        dicright[b]+=1
        if -left[0]>right[0]:
            print("YES")
        else:
            print("NO")
    else:
        dicleft[a]-=1
        dicright[b]-=1
        while left and dicleft[-left[0]]==0:
            heapq.heappop(left)
        while right and dicright[right[0]]==0:
            heapq.heappop(right)
        if len(left)==0 or -left[0]<=right[0]:
            print("NO")
        else:
            print("YES")
```