Cheeting Paper

施广熠 生命科学学院 计算概论B

1、函数

```
#二进制八进制与十六进制
ans=[2,12,34,99,-1,-7]
for k in range(len(ans)):
    print(bin(ans[k]),oct(ans[k]),hex(ans[k]))
#注意输出的数为“0b1100”，“0o14”，”0xc“

print(int("1010",2)) #int函数可将字符串类型的某进制数（参数即为数的进制）转化为十进制整型
```

```
#需要赋值的函数：upper、lower、sorted、split、pop
#不需要赋值的函数：add、append、remove、sort

ord()  #返回ASC码值
chr()  #ASC码返回字符串
```

```
print("string","list",sep=".",end="?") #控制间隔/结尾
#end=的默认值为"/n" 即换行 "/t" 为制表

#round()保留小数
print(round(3.123456789,5)# 3.12346
```

```
#指定位数的小数保留
num=3.1415926
print(format(num, ".5f"), end="")
#如何在字符串中穿插变量
st1="hello world"
print(f"say:{st1}")
```

```
#list.sort()的key参数
input()
lt = input().split()
max_len = len(max(lt, key = lambda x:len(x))) #x可替换为任意变量
lt.sort(key = lambda x: tuple([int(i) for i in x]) * ceil(max_len/len(x)))
lt1 = lt[::-1]
print(''.join(lt1),''.join(lt))
```

```
#try/except
#邮箱验证
a=0
while True:
    try:
        e=input()
        if "@" in e and "@." not in e :
            a+=1
        if ".@" not in e and "." in e[e.find("@"):]:
```

```
            a+=1
        if e[0]!="@" and e[0]!=".":
            a+=1
        if e[-1]!="@" and e[-1]!=".":
            a+=1
        if e.count("@")==1:
            a+=1
        if a==5:
            a=0
            print("YES")
        else:
            a=0
            print("NO")
    except EOFError: #检测到EOFError表面测试数据结束
        break
```

2、数组

```
#字符串
#字符串-大小写转化
letter="L" #判断字母大小写
print(letter.isupper(),letter.islower())
string="Good!"  #仅将小写字母转为大写/仅将大写字母转为小写
print(string.lower(),string.upper())
sent="it is SO GOOD!"  #转换为首字母大写/大小写转换
print(sent.capitalize(),sent.swapcase())

#题目【多项式的时间复杂度】
#字符串处理常用方法：截取[:]字符串，split("?")分割，+直接相加
strings=input().split('+')
tuple_strings=[(s.split('n^')) for s in strings]#同时存储系数和次数
maximum=0
for string in tuple_strings:
    if string[0]!='0':
        maximum=max(maximum,int(string[1]))#迭代器遍历，程序更舒服
print('n^'+str(maximum))
```

```
#字典相关
d={"k1":1} #新建一个字典
d["k2"]=2  #修改字典“k2”键对应的值，如无对应的键，则创建新的键值对
k1=d["k1"] #访问k1键对应的值
k2=d.get("k2","NO")
k3=d.get("k3","NO")  #get()访问时，如果没有找到键，则返回默认值用","加在后面
l1=d.values()  #返回所有值的列表（直接print会有问题），同理还有keys/items
print(2 in l1)
#遍历方法：
for key,value in d.items():
    print(key,value)
del d["k1"]
a=d.pop("k2")  #弹出给定键的相应值，在字典中删除键值对
d={"k1":4,"k2":5,"k3":6}
b=d.popitem()  #弹出最后一个键值对（以元组形式）
```

```
#集合相关
#空集合用s=set()
SET={1,3,2,4,5,4}  #区别于元组，集合既不能切片，也不能直接相加
print(SET)   #
SET.add(4)   #集合不能重复，自动归到同一个数字
SET.add(6)
SET.remove(2) #或用discard()，不会报错
SET.pop()  #从最右端弹出
print(SET)
set0={0,1,2,3}
print(set0 & SET,set0 | SET,set0 - SET)
#分别为交集/并集/补集

#题目【校门口的树】
L,n=map(int,input().split())
s=set()     #空集合为s=set();s={}为创建一个空的字典
for i in range(n):
    start,end=map(int,input().split())
    se={i for i in range(start,end+1)}  #集合生成式的使用
    s=s.union(se)  #union函数表示取并集，返回一个集合
print(L-len(s)+1)
```

```
#双向链表 （增删复杂度为O（1））
from collections import deque
l=[2,3,4,5]
q=deque(l)  #创建双向列表
#appendleft()/append() 向队头/队尾添加元素,添加序列则为extendlrft()/extend()
q.insert(2,3.5) #向指定位置插入元素（前一个参数为index）
#popleft()/pop() 从队头/队尾弹出元素
#index() count()均可使用
```

```
#堆排序
import heapq
heap=[3,22,5,5,1,3]
heapq.heapify(heap)  #原地转化，不需要赋值
heapq.heappush(32)  #增添元素，保持堆的合法性
heapq.heappop()  #创建heap时所有元素取反，弹出之后再取反就可以pop最大值
heapq.nlargest(2,heap)  #弹出最大的n个，最小n个用nsmallest()
#heapreplace(heap,item)先弹出最小元素，再添加item
#heappushpop(heap,item)先添加item，再弹出最小元素

#12.剪绳子
import heapq  #用堆的结构，效率非常高
n=int(input())
num=[int(i) for i in input().split()]
heapq.heapify(num)
su=0
while len(num)>1:
    num1=heapq.heappop(num)
    num2=heapq.heappop(num)
    su+=num1+num2
    heapq.heappush(num,num1+num2)
print(su)
```

```python
#栈 题目【波兰表达式】
# 用栈解决，更好理解
expression = input().split()
stack = []
ind=len(expression)-1
for i in range(1,len(expression)+1):
    a = expression[-i]
    if a in ['+', '-', '*', '/']:
        c = stack.pop(-1)
        d = stack.pop(-1)
        if a == '+':
            stack.append(c + d)
        elif a == '-':
            stack.append(c - d)
        elif a == '*':
            stack.append(c * d)
        else:
            stack.append(c / d)
    else:
        stack.append(float(a))

print("{:.6f}".format(stack[0]))
```

3、工具

ASCII码表

二进制	十进制	十六进制	图形	二进制	十进制	十六进制	图形	二进制	十进制	十六进制	图形
0010 0000	32	20	(space)	0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
0011 0011	51	33	3	0101 0011	83	53	S	0111 0011	115	73	s
0011 0100	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t
0011 0101	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
0011 0110	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
0011 0111	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
0011 1000	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
0011 1001	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
0011 1010	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
0011 1011	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
0011 1100	60	3C	<	0101 1100	92	5C	\	0111 1100	124	7C	|
0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0011 1111	63	3F	?	0101 1111	95	5F	_				

```python
#埃氏筛
def prime(n):
    s=set()
    ans=0
    ans=[True]*(n+1)
    ans[0]=ans[1]=False
    for i in range(2,int(n**0.5)+1):
        if ans[i]:
            for j in range(i*i,n+1,i):
                ans[j]=False
    for k in range(2,n+1):
        if ans[k]:
            s.add(k)
    return s
```

```python
#二分查找与插入
import bisect
sorted_list = [1,3,5,7,9] #[(0)1, (1)3, (2)5, (3)7, (4)9]
position = bisect.bisect_left(sorted_list, 6)
print(position) # 输出：3，因为6应该插入到位置3，才能保持列表的升序顺序
```

```
bisect.insort_left(sorted_list, 6)
print(sorted_list) # 输出：[1, 3, 5, 6, 7, 9]，6被插入到适当的位置以保持升序顺序

sorted_list=(1,3,5,7,7,7,9) #left为小于等于x的第一个索引，right为大于x的第一个索引
print(bisect.bisect_left(sorted_list,7))
print(bisect.bisect_right(sorted_list,7))
# 输出：3 6

#二分查找源码
def bisect_right(a, x, lo=0, hi=None, *, key=None):

    if lo < 0:
        raise ValueError('lo must be non-negative')
    if hi is None:
        hi = len(a)
    # Note, the comparison uses "<" to match the
    # __lt__() logic in list.sort() and in heapq.
    if key is None:
        while lo < hi:
            mid = (lo + hi) // 2
            if x < a[mid]:
                hi = mid
            else:
                lo = mid + 1
    else:
        while lo < hi:
            mid = (lo + hi) // 2
            if x < key(a[mid]):
                hi = mid
            else:
                lo = mid + 1
    return lo
```

```
#calendar（improt calendar)
1. calendar.month(年, 月) : 返回一个月份的日历字符串。它接受年份和月份作为参数，并以多行字符串的形式返回该月份的日历。
2. calendar.calendar(年) : 返回一个年份的日历字符串。这个函数生成整个年份的日历，格式化为多行字符串。
3. calendar.monthrange(年, 月) : 返回两个整数，第一个是该月第一天是周几（0-6表示周一到周日），第二个是该月的天数。
4. calendar.weekday(年, 月, 日) : 返回给定日期是星期几。0-6的返回值分别代表星期一到星期日。
5. calendar.isleap(年) : 返回一个布尔值，指示指定的年份是否是闰年。
6. calendar.leapdays(年1, 年2) : 返回在指定范围内的闰年数量，不包括第二个年份。
7. calendar.monthcalendar(年, 月) : 返回一个整数矩阵，表示指定月份的日历。每个子列表表示一个星期；天数为0表示该月份此天不在该星期内。
8. calendar.setfirstweekday(星期) : 设置日历每周的起始日。默认情况下，第一天是星期一，但可以通过这个函数更改。
9. calendar.firstweekday() : 返回当前设置的每周起始日。
```

```
#Counter
from collections import Counter
# O(n)
# 创建一个待统计的列表
data = ['apple', 'banana', 'apple', 'orange', 'banana', 'apple']
# 使用Counter统计元素出现次数
counter_result = Counter(data) # 返回一个字典类型的东西
# 输出统计结果
print(counter_result) # Counter({'apple': 3, 'banana': 2, 'orange': 1})
print(counter_result["apple"]) # 3
```

```
#排列与组合
from itertools import permutations as per
elements = [1, 2, 3]
permutations = list(per(elements))
#[(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)]

from itertools import combinations as com
elements = ['A', 'B', 'C', 'D']
# 生成所有长度为2的组合
combinations = list(com(elements, 2))
#[('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'C'), ('B', 'D'), ('C', 'D')]
```

4、模板与例题

dp:

```
#采药-01背包-一维形式
#背包问题可以从二维本质出发
T,M=map(int,input().split())
l=[]
dp=[0]*(T+1)
for i in range(M):
    t,v=map(int,input().split())
    l.append((t,v))
for k in range(M):
    for j in range(T,0,-1):
        if j>=l[k][0]:
            dp[j]=max(dp[j],l[k][1]+dp[j-l[k][0]])
print(dp[-1])

#18.最佳凑单 #01背包问题，理解dp[i,j]的意义
n,t=map(int,input().split())
value_list=[int(_) for _ in input().split()]
dp=[999999999]*(t+1)
for i in range(n):
    for j in range(t,-1,-1):
        if value_list[i]>=j:
            dp[j]=min(dp[j],value_list[i]-j)
        else:
            dp[j]=min(dp[j],dp[j-value_list[i]])
if dp[-1]==999999999:
```

```
    print(0)
else:
    print(dp[-1]+t)
```

```
#完全背包(初始版本/一维数组优化/状态转移方程优化）
#多重背包就对每个i以最后一个个数参数增加一次循环
n,v=map(int,input().split())
staff=[]
for i in range(n):
    V,W=map(int, input().split())
    staff.append((V,W))
dp=[[0]*(v+1) for i in range(n+1)]
for i in range(1,n+1):
    for j in range(1,v+1):
        if j<staff[i-1][0]:
            dp[i][j]=dp[i-1][j]
        else:
            k=1
            while j>=k*staff[i-1][0]:
                dp[i][j]=max(dp[i-1][j-k*staff[i-1][0]]+k*staff[i-1][1],dp[i-1]
[j],dp[i][j])
                k+=1
print(dp[-1][-1])

n,v=map(int,input().split())
staff=[]
for i in range(n):
    V,W=map(int, input().split())
    staff.append((V,W))
dp=[0]*(v+1)
for i in range(1,n+1):
    for j in range(v,0,-1):
        k=0
        while j>=k*staff[i-1][0]:
            dp[j]=max(dp[j-k*staff[i-1][0]]+k*staff[i-1][1],dp[j])
            k+=1
print(dp[-1])

n,v=map(int,input().split())
staff=[]
for i in range(n):
    V,W=map(int, input().split())
    staff.append((V,W))
dp=[[0]*(v+1) for i in range(n+1)]
for i in range(1,n+1):
    for j in range(1,v+1):
        if j>=staff[i-1][0]:  #相当于取了某些个数的i，先返回同行位置，取不了了再回到i-1
            dp[i][j]=max(dp[i][j-staff[i-1][0]]+staff[i-1][1],dp[i-1][j])
        else:
            dp[i][j]=dp[i-1][j-1]
print(dp[-1][-1])
```

```
#10.剪彩
n,a,b,c=map(int,input().split())
dp=[0]+[-4000]*n  #保证只能从0发出
for i in range(n+1):
    for j in (a,b,c):
        if i>=j:
            dp[i]=max(dp[i],dp[i-j]+1)
print(dp[n])
```

```
#3.合唱队形  寻找最长递增/递减子列
#一维dp，dp[i]表示包含i的从一端开始的最长递增子列，便于得出状态转移方程
n=int(input())
height=[int(i) for i in input().split()]
dp_l=[1]*n
for i in range(1,n):
    for j in range(i):
        if height[i]>height[j]:
            dp_l[i]=max(dp_l[i],dp_l[j]+1)
dp_r=[1]*n
for i in range(n-2,-1,-1):
    for j in range(n-1,i,-1):
        if height[i]>height[j]:
            dp_r[i]=max(dp_r[i],dp_r[j]+1)
ma=0
for i in range(n):
    ma=max(ma,dp_l[i]+dp_r[i])
print(n-ma+1)
```

bfs:

```
#寻宝 bfs版，bfs寻找最短路径长度
from collections import deque
def valid(x,y):
    if 0<=x<m and 0<=y<n:
        if treasure[x][y]!=2 and not inque[x][y]:
            return True
        else:
            return False
    else:
        return False

def bfs():
    q=deque()
    q.append((0,0,0)) #将目前格子的步数和位置一同记录
    while q:
        t=q.popleft()
        for i in range(4):
            x,y,cnt=t[0]+dx[i],t[1]+dy[i],t[2]
            if valid(x,y):
                inque[x][y]=True
                q.append((x,y,cnt+1)) #在前一格的基础上cnt+1
                if treasure[x][y]==1:
```

```
                    print(cnt+1)
                    return
    print("NO")

m,n=map(int,input().split())
treasure=[]
for i in range(m):
    x=[int(_) for _ in input().split()]
    treasure.append(x)
inque=[[False]*n for i in range(m)]
dx=[0,1,0,-1]
dy=[1,0,-1,0]
if treasure[0][0]==1:
    print(0)
    exit()
if treasure[0][0]==2:
    print("NO")
    exit()
bfs()
```

```
#最大连通域面积-bfs模板-面积计算
from collections import deque
dx=[1,-1,0,0,1,1,-1,-1]
dy=[0,0,1,-1,1,-1,-1,1]
max_total=0
def valid(x,y):
    if 0<=x<n and 0<=y<m:
        if matrix[x][y]=="W" and not inq[x][y]:
            return True
        else:
            return False
    else:
        return False
def bfs(x,y):
    global max_total  #多组数据时全局变量一定不要忘记归零!
    q=deque()
    q.append((x,y))
    inq[x][y]=True
    cnt=1
    while q:
        bounce=q.popleft()
        for r in range(8):
            nx=bounce[0]+dx[r]
            ny=bounce[1]+dy[r]
            if valid(nx,ny):
                q.append((nx,ny))
                inq[nx][ny]=True
                cnt+=1
    max_total=max(max_total,cnt)

t=int(input())
for i in range(t):
    n,m=map(int,input().split())
    matrix=[]
    for i0 in range(n):
```

```
        matrix.append(input())
    inq=[]
    for i1 in range(n):
        inq.append([False]*m)
    for k1 in range(n):
        for k2 in range(m):
            if valid(k1,k2):
                bfs(k1,k2)
    print(max_total)
    max_total=0
```

```
#迷宫最短路径
#如何在bfs中记录路径
from queue import Queue
MAXN = 100
MAXD = 4
dx = [0, 0, 1, -1]
dy = [1, -1, 0, 0]
def canVisit(x, y):
    return x >= 0 and x < n and y >= 0 and y < m and maze[x][y] == 0 and not
inQueue[x][y]
def BFS(x, y):
    q = Queue()
    q.put((x, y))
    inQueue[x][y] = True
    while not q.empty():
        front = q.get()
        if front[0] == n - 1 and front[1] == m - 1:
            return
        for i in range(MAXD):
            nextX = front[0] + dx[i]
            nextY = front[1] + dy[i]
            if canVisit(nextX, nextY):
                pre[nextX][nextY] = (front[0], front[1]) #pre中每个点都记录前一个点的
位置
                inQueue[nextX][nextY] = True
                q.put((nextX, nextY))
def printPath(p):
    prePosition = pre[p[0]][p[1]]
    if prePosition == (-1, -1):
        print(p[0] + 1, p[1] + 1)
        return
    printPath(prePosition)  #类似回溯，推到起始点时开始从头输出
    print(p[0] + 1, p[1] + 1)

n, m = map(int, input().split())
maze = []
for _ in range(n):
    row = list(map(int, input().split()))
    maze.append(row)
inQueue = [[False] * m for _ in range(n)]
pre = [[(-1, -1)] * m for _ in range(n)]
BFS(0, 0)
printPath((n - 1, m - 1))
```

dfs:

```
#寻宝 dfs版
def dfs(x,y):
    dx=[1,0,-1,0]
    dy=[0,1,0,-1]
    global cnt
    global cnt_min
    global treasure
    for i in range(4):
        if treasure[x+dx[i]][y+dy[i]]!=2:
            treasure[x][y]=2
            cnt+=1
            if treasure[x+dx[i]][y+dy[i]]==1:
                cnt_min=min(cnt,cnt_min)
                treasure[x][y]=0  #注意计数器和图都要恢复回溯
                cnt-=1
                break
            else:
                dfs(x+dx[i],y+dy[i])
                treasure[x][y]=0
                cnt-=1
    return

cnt_min=99999
cnt=0
n,m=map(int,input().split())
treasure=[[2]*(m+2)]
for i in range(n):
    treasure.append([2]+[int(j) for j in input().split()]+[2])
treasure.append([2]*(m+2))
if treasure[1][1]==1:
    print(0)
    exit()
dfs(1,1)
if cnt_min==99999:
    print("NO")
else:
    print(cnt_min)
```

```
#八皇后问题
def dfs(cur):
    global solution
    for i0 in range(8):
        flag=True
        for j0 in range(len(solution)): #用列表solution直接储存某一次的结果，因为八皇后可以直接根据前面的路径判断后面的位置是否可访问，不需要在图上修改
            if cur+i0==j0+solution[j0] or cur-i0==j0-solution[j0] or i0 in solution:
                flag=False
        if flag:
            solution+=[i0]
            if cur==7:
                su=0
```

```
                for i1 in range(8):
                    su+=(solution[i1]+1)*10**(7-i1)
                ans.append(su)
                solution.pop(-1)
            else:
                dfs(cur+1)
                solution.pop(-1)
        else:
            continue

ans=[]
solution=[]
dfs(0)
n=int(input())
for i in range(n):
    num=int(input())
    print(ans[num-1])
```

```
#组合乘积
def dfs(T,l): #dfs解决非图问题
    for i in range(len(l)):
        if l[i]!=0:
            if T%l[i]==0 and l[i]>0:
                T=T//l[i]
                temp=l[i]
                l[i]=-1  #防止路径重复
                if T==1:
                    print("YES")
                    exit()
                else:
                    dfs(T,l)
                    l[i]=temp  #零时储存前一个值便于回溯
                    T=T*temp

t=int(input())
num=[int(i) for i in input().split()]
if t==0:
    if 0 in num:
        print("YES")
    else:
        print("NO")
else:
    dfs(t,num)
    print("NO")
```

```
#迷宫最大权值和
dx = [-1, 0, 1, 0]
dy = [ 0, 1, 0, -1]
maxValue = -9999
def dfs(maze, x, y, nowValue):
    global maxValue
    if x==n and y==m:
        if nowValue > maxValue:
            maxValue = nowValue
```

```
            return
    for i in range(4):
        nx = x + dx[i]
        ny = y + dy[i]
        if maze[nx][ny] == 0:
            maze[nx][ny] = -1
            tmp = w[x][y]
            w[x][y] = -9999
            nextValue = nowValue + w[nx][ny]
            dfs(maze, nx, ny, nextValue)
            maze[nx][ny] = 0
            w[x][y] = tmp

n, m = map(int, input().split())
maze = []
maze.append( [-1 for x in range(m+2)] )
for _ in range(n):
    maze.append([-1] + [int(_) for _ in input().split()] + [-1])
maze.append([-1 for x in range(m + 2)])
w = []
w.append([-9999 for x in range(m + 2)])
for _ in range(n):
    w.append([-9999] + [int(_) for _ in input().split()] + [-9999])
w.append([-9999 for x in range(m + 2)])
dfs(maze, 1, 1, w[1][1])
print(maxValue)
```

双指针：

```
#三数之和，排序后用双指针解决
def threeSum(nums):
    nums.sort()  # 先对数组排序
    result = []
    n = len(nums)

    for i in range(n - 2):
        # 跳过重复的元素
        if i > 0 and nums[i] == nums[i - 1]:
            continue

        # 双指针
        left = i + 1
        right = n - 1

        while left < right:
            total = nums[i] + nums[left] + nums[right]

            if total < 0:
                left += 1
            elif total > 0:
                right -= 1
            else:
                result.append([nums[i], nums[left], nums[right]])

                # 跳过重复的元素
```

```
                while left < right and nums[left] == nums[left + 1]:
                    left += 1
                while left < right and nums[right] == nums[right - 1]:
                    right -= 1

                left += 1
                right -= 1

    return len(result)

nums = [int(_) for _ in input().split()]
count = threeSum(nums)
print(count)

#朵拉找子列 子列末端都不是整个子段的最大值或者最小值
N=int(input())
for i in range(N):
    n=int(input())
    l=[int(j) for j in input().split()]
    if n<4:
        print(-1)
    else:
        left=0
        right=n-1  #因题目所给排列的特殊性，可以确认最大值和最小值，用双指针向中间取
                   #尤其是最大的子列，是从全体中筛选出来的一段
        M=n
        m=1
        while right-left>=3:
            if l[left]==M:
                M-=1
                left+=1
            elif l[right]==M:
                M-=1
                right-=1
            elif l[left]==m:
                m+=1
                left+=1
            elif l[right]==m:
                m+=1
                right-=1
            else:
                print(left+1,right+1)
                exit()
        print(-1)
```

逆向思维：

```
#垃圾炸弹 mixtra
#反向思维，注意到垃圾的点数很少，可以将垃圾的值加到周围d*d的点上
d=int(input())
n=int(input())
square = [[0]*1025 for _ in range(1025)]
for _ in range(n):
```

```
    x,y,k=map(int, input().split())
    for i in range(max(x-d, 0), min(x+d+1, 1025)):
        for j in range(max(y-d, 0), min(y+d+1, 1025)):
            square[i][j] += k
res = max_point = 0
for i in range(0, 1025):
    for j in range(0, 1025):
        if square[i][j] > max_point:
            max_point = square[i][j]
            res=1
        elif square[i][j] == max_point:
            res+=1
print(res, max_point)
```

```
#河中跳房子 用确定的interval长度反向确定移走的石头数
#注意二分查找的写法
n,m = map(int, input().split())
expenditure = []
for _ in range(n):
    expenditure.append(int(input()))

def check(x):
    num, s = 1, 0
    for i in range(n):
        if s + expenditure[i] > x:
            s = expenditure[i]
            num += 1
        else:
            s += expenditure[i]

    return [False, True][num > m]

lo = max(expenditure)
# hi = sum(expenditure)
hi = sum(expenditure) + 1
ans = 1
while lo < hi:
    mid = (lo + hi) // 2
    if check(mid):       # 返回True，是因为num>m，是确定不合适
        lo = mid + 1     # 所以lo可以置为 mid + 1。
    else:
        ans = mid    # 如果num==m，mid就是答案
        hi = mid

#print(lo)
print(ans)
```

连续修改：

```
#猫咪派对
#使用一个数组 f 来记录每种颜色出现的次数，使用另一个数组 cnt 来统计每个次数的颜色数量。
#通过迭代颜色列表，并根据不同的条件判断，计算并更新最长的连续天数 ans。
n = int(input())
colors = list(map(int, input().split()))
```

```
N = 10**5 + 10
ans = 0
mx = 0
f = [0] * N
cnt = [0] * N

for i in range(1, n + 1):
    color = colors[i - 1]
    cnt[f[color]] -= 1
    f[color] += 1
    cnt[f[color]] += 1
    mx = max(mx, f[color])
    ok = False
    if cnt[1] == i:  # every color has occurrence of 1
        ok = True
    elif cnt[i] == 1:  # only one color has the maximum occurrence and the
occurrence is i
        ok = True
    elif cnt[1] == 1 and cnt[mx] * mx == i - 1:  # one color has occurrence of 1
and other colors have the same occurrence
        ok = True
    elif cnt[mx - 1] * (mx - 1) == i - mx and cnt[mx] == 1:  # one color has the
occurrence 1 more than any other color
        ok = True
    if ok:
        ans = i

print(ans)
```

余数判断整体：

```
#XXXXX
line=int(input())
for i in range(line):
    n,x=map(int,input().split())
    l=input().split()
    l1=[]
    a=0
    for j in range(len(l)):
        l1.append(int(l[j]))
        if int(l[j])%x==0:
            a+=1
    if a==len(l1):
        print(-1)
    else:
        if sum(l1)%x!=0:
            print(n)
        else:
            for k in range(n):
               if l1[k]%x!=0 or l1[-(k+1)]%x!=0:
                   print(n-k-1)
                   break
```

线性最优：

```
#世界杯只因
n=int(input())
zy=[int(i) for i in input().split()]
rang=[(max(0,i-zy[i]),min(n-1,i+zy[i])) for i in range(n)]  #对监控的覆盖范围进行排序
rang.sort()
end=-1
r=0
num=0
while end<n-1:
    ma=0
    while rang[r][0]<=end+1:
        ma=max(ma,rang[r][1])  #找出覆盖不断情况下的最远覆盖末端（第一次要求从0开始，之后仅要求从上一次新增大的范围中选择最远覆盖末端）
        r+=1
        if r==n:
            break
    r=max(end,0) #从新一次的起点开始遍历
    end=ma
    num+=1
print(num)
```

```
#雷达安装
cnt=0
while True:
    cnt+=1
    n,d=map(int,input().split())
    if n==0:
        exit()
    island=[]
    interval=[]
    for i in range(n):
        a,b=map(int,input().split())
        island.append((a,b))
    flag=True
    #判断是否能覆盖
    for j in range(n):
        if island[j][1]>d:
            flag=False
    if not flag:
        print(f"Case {cnt}: -1")
        input()
        continue
    #####
    for i in range(n):  #将雷达位置转化为线性坐标上的interval区间
        s=island[i][0]-(d*d-island[i][1]**2)**0.5
        e=island[i][0]+(d*d-island[i][1]**2)**0.5
        interval.append((s,e))
    interval.sort()  #先进行排序，排序可以保证局部的处理方法是最优解
    su=1
    pos=[interval[0][0],interval[0][1]]
    for j in range(n):
```

```
        if interval[j][0]<=pos[1]:
            pos[1]=min(interval[j][1],pos[1])
        else:
            pos=[interval[j][0],interval[j][1]]
            su+=1
    print(f"Case {cnt}: {su}")
    input()
```

子列：

```
#15.分发糖果
n=int(input())  #本质上就是寻找单调递增或递减的子序列
l1=[int(i) for i in input().split()]
l2=[1]*n
l3=[1]*n
for i in range(1,n):
    if l1[i]>l1[i-1]:
        l2[i]=l2[i-1]+1
    else:
        l2[i]=1
for i in range(n-2,-1,-1):
    if l1[i]>l1[i+1]:
        l3[i]=l3[i+1]+1
    else:
        l3[i]=1
su=0
for i in range(n):
    su+=max(l2[i],l3[i])
print(su)
```

5、注意事项

1.边界值和特殊情况

2.整除和除余符号不搞混

3.dfs bfs等循环数多的，注意循环开头的几行代码

4.注意重新审题，题目中给出的限制

5.检查输入和输出格式是否完全一致，主要是输出

6.减少逻辑问题要在写的时候保持冷静