语法

常用库与常用函数

1. `Collections`:
   - `Counter(hashable)`:加法，减法，`elements()`，`most_common()`，`total()`，`update()`
   - `OrderedDict`，`defaultdict`，`deque`，`namedtuple(typename, fieldname)`
2. `functools`:
   - `partial`，`cmp_to_key`，`@lru_cache(max_size=None)`
3. `itertools`:
   - `accumulate(p, func)`，`combinations(iterable, r)`，`permutation()`，`combinations_with_replacement()`，`groupby(iterable, key)`
4. `heapq`:`heapify() -> None`
5. `bisect` 注意其中`key`参数在Openjudge上不可用

```
def bisect_right(a, x, lo=0, hi=None, *, key=None):
    """Return the index where to insert item x in list a, assuming a is sorted.

    The return value i is such that all e in a[:i] have e <= x, and all e in
    a[i:] have e > x.  So if x already appears in the list, a.insert(i, x) will
    insert just after the rightmost x already there.

    Optional args lo (default 0) and hi (default len(a)) bound the
    slice of a to be searched.

    A custom key function can be supplied to customize the sort order.
    """

    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
```

`bisect_left`的差别在于，循环部分的代码为

```
        while lo < hi:
            mid = (lo + hi) // 2
            if a[mid] < x:
                lo = mid + 1
            else:
                hi = mid
```

套用以上代码时，得到的结果最后有时候可能需要减1

关键字、内置函数与其他语法

1. `sorted() -> List`
2. `yield`
3. `[0 if i < 2 else 1 for i in range(n)]`
4. `f-string`
    - `f"{a:.2f}"`，`f"{a:#<10}"`(或使用`str.rjust(n, char)`)

算法

贪心

区间问题

- 按左端点排：区间合并、区间覆盖、区间分组（将右端点放入小顶堆）
- 按右端点排：不相交区间、区间选点

例题：充实的寒假生活（不相交区间）

- 描述
  寒假马上就要到了，龙傲天同学获得了从第0天开始到第60天结束为期61天超长寒假，他想要尽可能丰富自己的寒假生活。
  现提供若干个活动的起止时间，请计算龙同学这个寒假至多可以参加多少个活动？注意所参加的活动不能有任何时间上的重叠，在第x天结束的活动和在第x天开始的活动不可同时选择。
- 输入
  第一行为整数n，代表接下来输入的活动个数(n < 10000)
  紧接着的n行，每一行都有两个整数，第一个整数代表活动的开始时间，第二个整数代表全结束时间
- 输出
  输出至多参加的活动个数

```
n = int(input())
act = sorted([list(map(int, input().split())) for i in range(n)], key = lambda
t: t[1])
cnt = 0
start = -1
for a in act:
    if a[0] > start:
        cnt += 1
        start=a[1]
print(cnt)
```

其他例题

排队

- 描述
  有 N 名同学从左到右排成一排，第 i 名同学的身高为 hi。现在张老师想改变排队的顺序，他能进行任意多次（包括0次）如下操作：如果两名同学相邻，并且他们的身高之差不超过 D，那么老师就能交换他俩的顺序。
  请你帮张老师算一算，通过以上操作，字典序最小的所有同学（从左到右）身高序列是什么?
- 输入
  第一行包含两个正整数 N, D (1<=N<=105, 1<=D<=109)。
  接下去 N 行，每行一个正整数 hi (1<=hi<=109) 表示从左到右每名同学的身高。
- 输出
  输出 N 行，第 i 行表示答案中第 i 名同学的身高。

```
from collections import deque
def main():
    N, D = map(int, input().split())
    stu = deque(int(input()) for i in range(N))

    ans = []
    while stu:
        premin = stu[0]
        premax = stu[0]
        free = []
        for i in range(len(stu)):
            h = stu.popleft()
            if h - premin <= D and premax - h <= D:
                free.append(h)
            else:
                stu.append(h)
            premax = max(premax, h)
            premin = min(premin, h)
        ans += sorted(free)
    print(*ans, sep = "\n")
```

最大最小整数

- 描述
  假设有n个正整数，将它们连成一片，将会组成一个新的大整数。现需要求出，能组成的最大最小整数。
  比如，有4个正整数，23，9，182，79，连成的最大整数是97923182，最小的整数是18223799。
- 输入
  第一行包含一个整数n，1<=n<=1000。
  第二行包含n个正整数，相邻正整数间以空格隔开。
- 输出
  输出为一行，为这n个正整数能组成的最大的多位整数和最小的多位整数，中间用空格隔开。
- 提示
  位数不同但前几位相同的时候。例如： 898 8987，大整数是898+8987，而不是8987+898

```
# 两倍长度是正确的。
from math import ceil
input()
lt = input().split()

max_len = len(max(lt, key = lambda x:len(x)))
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))
```

动态规划

背包问题

0-1背包

- 题意概要：有n个物品和一个容量为W的背包，每个物品有重量$w{i}$ *和价值*$v{i}$两种属性，要求选若干物品放入背包使背包中物品的总价值最大且背包中物品的总重量不超过背包的容量。

```
for i in range(n):
    for l in range(w, w[i] - 1, -1):
        f[l] = max(f[l], f[l - w[i]] + v[i])
```

完全背包

- 完全背包模型与 0-1 背包类似，与 0-1 背包的区别仅在于一个物品可以选取无限次，而非仅能选取一次。

```
n,m=map(int,input().split())
a=list(map(int,input().split()))
dp=[0]+[10000]*m
for i in range(1,m+1):
    for x in a:
        if i>=x:
            dp[i]=min(dp[i-x]+1,dp[i])
print(dp[m])
```

多重背包

- 多重背包也是 0-1 背包的一个变式。与 0-1 背包的区别在于每种物品有k_i个，而非一个。

```
for i in range(n):
    for l in range(w, w[i] - 1, -1):
        for k in range(1, cnt[i] + 1):
            if k * w[i] <= l:
                dp[l] = max(dp[l], dp[weight - k * w[i]] + k * v[i])
            else:
                break
```

二维费用背包

- 有n个任务需要完成，完成第i个任务需要花费t_i 分钟，产生c_i 元的开支。现在有T 分钟时间，W 元钱来处理这些任务，求最多能完成多少任务。

```
for i in range(n):
    for x in range(T, t[i] - 1, -1):
        for y in range(W, w[i] - 1, -1):
            dp[x][y] = max(dp[x][y], dp[x - t[i]][y - w[i]])
```

- 例题：宠物小精灵之收服

```
L = [[-1]*(M+1) for i in range(K+1)]
L[0][M] = N
for i in range(K):
    cost, dmg = map(int, input().split())
    for p in range(M):
        for q in range(i+1, 0, -1):
            if p+dmg <= M and L[q-1][p+dmg] != -1:
                L[q][p] = max(L[q][p], L[q-1][p+dmg]-cost)


def find():
    for i in range(K, -1, -1):
        for j in range(M, -1, -1):
            if L[i][j] != -1:
                return [str(i), str(j)]


print(' '.join(find()))
```

最长不降子序列

- 给定一个长度为n 的序列 A（$n \leq 5000$），求出一个最长的 A 的子序列，满足该子序列的后一个元素不小于前一个元素。

```
from bisect import bisect_right
n = int(input())
h = list(map(int, input().split()))
testing = [-1] * (n + 1)
for i in h:
    testing[bisect_right(testing, i) - 1] = i
print(testing[::-1].index(-1))
```

土豪购物

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

```
s=list(input())
n=len(s)
dp1=[0]*n
dp2=[0]*n
if s[0]=="B":
    dp1[0]=1
elif s[0]=="R":
    dp2[0]=1
for i in range(1,n):
    if s[i]=="R":
        dp1[i]=dp1[i-1]
        dp2[i]=min(1+dp1[i-1],1+dp2[i-1])
    elif s[i]=="B":
        dp2[i]=dp2[i-1]
        dp1[i]=min(1+dp1[i-1],1+dp2[i-1])
print(dp1[n-1])
```

红蓝玫瑰

Boredom。

```
from collections import Counter
n = int(input())
cnt = dict(Counter(map(int, input().split())))
n = max(cnt.keys())
dp = [[0 for i in range(n+1)] for j in range(2)]
a = [0 for i in range(n+2)]
for k in cnt:
    a[k] = cnt[k]
dp[1][1] = a[1]
dp[0][2] = a[1]
dp[1][2] = a[2] * 2
for i in range(3, n+1):
    dp[0][i] = max(dp[1][i-1], dp[1][i-2])
    dp[1][i] = dp[0][i-1] + i*a[i]
print(max(dp[0][n], dp[1][n]))
```

搜索

深度优先搜索

```
n=int(input())
zong=[]
a=list(input())
b=list(input())
c=list(input())
d=list(input())
zong.append(a)
zong.append(b)
zong.append(c)
zong.append(d)
def dfs(t,i,bol,step):
    if step==len(t):
        return "YES"
    if i==len(t) and step!=len(t):
        return "NO"
    for j in range(4):
        if t[i] in zong[j] and not bol[j]:
            bol[j]=True
            if dfs(t,i+1,bol,step+1)=="YES":
                return "YES"
            else:
                bol[j]=False
    return "NO"
for _ in range(n):
    t=list(input())
    bol=[False]*4
    print(dfs(t,0,bol,0))
```

```
directions=[[0,1],[0,-1],[1,0],[-1,0]]
m,n=map(int,input().split())
a=[list(map(int,input().split())) for _ in range(m)]
dp=[[0]*(n+1) for _ in range(m+1)]
def dfs(i,j):
    if dp[i][j]>0:
        return dp[i][j]
    for dx,dy in directions:
            nx=i+dx;ny=j+dy
            if 0<=nx<m and 0<=ny<n and a[nx][ny]<a[i][j]:
                dp[i][j]=max(dp[i][j],dfs(nx,ny)+1)
    return dp[i][j]
result=0
for i in range(m):
    for j in range(n):
        result=max(result,dfs(i,j))
print(result+1)
```

```
directions=[[2,1],[1,2],[-1,2],[1,-2],[-1,-2],[2,-1],[-2,1],[-2,-1]]
cnt=0
def valued(n,m,x,y,bol):
    return 0<=x<n and 0<=y<m and not bol[x][y]
def dfs(n,m,x,y,bol):
    global cnt
```

```
    bol[x][y]=True
    if bol==[[True]*m for _ in range(n)]:
        cnt+=1
    bol[x][y]=False
    for dx,dy in directions:
        nx=x+dx;ny=y+dy
        if valued(n,m,nx,ny,bol):
            bol[x][y]=True
            dfs(n,m,nx,ny,bol)
            bol[x][y]=False
    return cnt
for _ in range(int(input())):
    a,b,c,d=map(int,input().split())
    bol=[[False]*b for _ in range(a)]
    cnt=0
    result=dfs(a,b,c,d,bol)
    print(result)
```

广度优先搜索

```
from collections import deque
def bfs(s, e):
    inq = set()
    inq.add(s)
    q = deque()
    q.append((0, s))
    while q:
        now, top = q.popleft() # 取出队首元素
        if top == e:
            return now # 需要返回的结果
        # 将 top 的下一层节点中未曾入队的节点全部入队，并加入集合inq设置为已入队
```

小游戏

- 描述
  游戏在一个分割成w * h个正方格子的矩形板上进行，每个正方格子上可以有一张游戏卡片，当然也可以没有。 当下面的情况满足时，我们认为两个游戏卡片之间有一条路径相连： 路径只包含水平或者竖直的直线段。路径不能穿过别的游戏卡片。但是允许路径临时的离开矩形板。

```
from collections import deque
directions = [[0, 1], [1, 0], [-1, 0], [0, -1]]
def bfs(w, h, a, x, y, c, d):
    a[c][d]=" "
    q = deque()
    q.append((x,y,1,0))
    q.append((x, y, -1, 0))
    q.append((x, y, 0, 1))
    q.append((x, y, 0, -1))
    inq = set()
    inq.add((x, y))
    step=0
    while q:
        for _ in range(len(q)):
            s, t,ds,dt=q.popleft()
            if  s==c and t==d:
                a[c][d]="X"
```

```
                return step
            for dx, dy in directions:
                if dx!=ds and dy!=dt:
                    for k in range(1,max(w,h)+2):
                        nx, ny = s + k*dx, t + k*dy
                        if 0 <= nx <= h + 1 and 0 <= ny <= w + 1 and a[nx][ny]
== " " and (nx, ny) not in inq:
                           q.append((nx,ny,dx,dy))
                           inq.add((nx, ny))
                        else:
                            break
        step+=1
    return "No"
o_1 = 0
while True:
    o_1 += 1
    w, h = map(int, input().split())
    if w == 0 and h == 0:
        break
    a = [[" "] * (w + 2)]
    a += [[" "] + list(input()) + [" "] for _ in range(h)]
    a += [[" "] * (w + 2)]
    print(f"Board #{o_1}:")
    o_2 = 0
    while True:
        o_2 += 1
        y, x, d, c = map(int, input().split())
        if x == 0 and y == 0 and c == 0 and d == 0:
            break
        l = bfs(w, h, a, x, y, c, d)
        if l == "No":
            print(f"Pair {o_2}: impossible.")
        else:
            print(f"Pair {o_2}: {l} segments.")
    print()
```

数学问题

中国剩余定理

![[中国剩余定理.png]]

生理周期

```
# def exgcd(a, b):
#   if b == 0:
#       x = 1
#       y = 0
#       return x, y
#   x1, y1 = exgcd(b, a%b)
#   x = y1
#   y = x1 - a//b*y1
#   return x, y

def main():
    for cnt in range(1, int(1e9)):
```

```
        p, e, i, d = map(int, input().split())
        if i == -1:
            break
        x = (1288*i - 6831*e+ 5544*p) % 21252 - d
        print(f"Case {cnt}: the next triple peak occurs in {(x-1) % 21252 + 1}
days.")

if __name__ == '__main__':
    main()
```

欧拉筛法

```
def euler(n):
    b=[True]*(n+1)
    t=[]
    for i in range(2,n+1):
        if b[i]:
            t.append(i)
        for p in t:
            if p*i>n:
              break
            b[p*i]=False
            if i%p==0:
                break
    return t
```

Narayana Pandita算法

1. Find the highest index `i` such that `s[i] < s[i+1]`. If no such index exists, the permutation is the last permutation.
2. Find the highest index `j > i` such that `s[j] > s[i]`. Such a `j` must exist, since `i+1` is such an index.
3. Swap `s[i]` with `s[j]`.
4. Reverse the order of all of the elements after index i till the last element.

其他问题

螺旋矩阵

洋葱

- 描述
  我们将洋葱抽象为一个n*n的矩阵，绕着矩阵最外侧环绕一圈，即得到一个“层”，然后将这个“层”中所有元素去掉，得到一个子矩阵，重复上述操作，即可得到若干个“层”。显然，若n为奇数，则位于其中心的那一个元素也构成一个“层”。
  现在给小明一个n*n的矩阵，小明想找到这个矩阵的“最大层”。“最大层”即为该矩阵每个“层”中数字之和最大的那个。

```
DIRECTIONS = ((0, 1), (1, 0), (0, -1), (-1, 0))
n = int(input())
N = 0
onion = [[-1e9 for i in range(n + 2)]] + [[-1e9] + list(map(int,
input().split())) + [-1e9] for i in range(n)] + [[-1e9 for i in range(n + 2)]]
dx, dy = DIRECTIONS[0]
```

```
x, y = 1, 0
layer = [0 for i in range(n // 2 + 1)]
for i in range(1, 1 + n * n):
    if onion[x + dx][y + dy] == -1e9:
        N += 1
        dx, dy = DIRECTIONS[N % 4]
    x, y = x + dx, y + dy
    layer[N // 4] += onion[x][y]
    onion[x][y] = -1e9
print(max(layer))
```

二分查找

月度开销

```
n,m=map(int,input().split())
a=[]
for _ in range(n):
    t=int(input())
    a.append(t)
p=sum(a)
def panduan(length):
    sum1=0
    shu=0
    if all(i<=length for i in a):
     for i in range(len(a)):
        if sum1+a[i]>length:
            shu+=1
            sum1=a[i]
        else:
            sum1+=a[i]
     if shu+1<=m:
        return True
     else:
        return False
    else:
        return False
start=0;end=p+1
result=0
while start<end:
    mid=(start+end)//2
    if panduan(mid):
        end=mid
        result=mid
    else:
        start=mid+1
print(result)
```

归并排序

```
def MergeSort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = MergeSort(arr[:mid])
    right = MergeSort(arr[mid:])
```

```
    return merge(left, right)

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result
```

求排列的逆序数

数据结构

并查集

- 按秩合并

```
def find(i):
    if parent[i] == i:
        return i
    else:
        result = find(parent[i])
        parent[i] = result
        return result
def union(left, right):
    lrep = find(left)
    rrep = find(right)
    if lrep == rrep:
        return
    if rank[lrep] < rank[rrep]:
        parent[lrep] = rrep
    elif rank[lrep] > rank[rrep]:
        parent[rrep] = lrep
    else:
        parent[rrep] = lrep
        rank[lrep] += 1
```

- 极简版

```
def find(i):
    if parent[i] != i:
        return find(parent[i])
    return i
def union(i, j)
    parent[find(i)] = find(j)
```

例题

- 现有一个学校，学校中有若干个班级，每个班级中有若干个学生，每个学生只会存在于一个班级中。如果学生 A 和学生 B 处于一个班级，学生 B 和学生 C 处于一个班级，那么我们称学生 A 和学生 C 也处于一个班级。 现已知学校中共 n 个学生（编号为从 1 到 n ），并给出 m 组学生关系（指定两个学生处于一个班级），问总共有多少个班级，并按降序给出每个班级的人数。

```
def find(x):
    if parent[x] != x:
        parent[x] = find(parent[x])
    return parent[x]

def union(x, y):
    root_x = find(x)
    root_y = find(y)
    if root_x != root_y:
        parent[root_x] = root_y
        size[root_y] += size[root_x]

n, m = map(int, input().split())
parent = list(range(n + 1))
size = [1] * (n + 1)

for _ in range(m):
    a, b = map(int, input().split())
    union(a, b)

classes = [size[x] for x in range(1, n + 1) if x == parent[x]]
print(len(classes))
print(' '.join(map(str, sorted(classes, reverse=True))))
```

排列

```
aa=[1]
c=1
for i in range(1,1025):
    c=c*i
    aa.append(c)
for _ in range(int(input())):
    n,k=map(int,input().split())
    cc=list(map(int,input().split()))
    bb=[i for i in range(1,n+1)]
    d=0
    l=n-1
    for j in cc:
        d+=bb.index(j)*aa[l]
        bb.remove(j)
        l-=1
    d+=k
    while d>=aa[n]:
        d-=aa[n]
    dd=[]
    bb=[i for i in range(1,n+1)]
    for p in range(n-1,-1,-1):
        t=d//aa[p]
        dd.append(bb[t])
```

```
        del(bb[t])
        d-=t*aa[p]
    print(*dd)
```

旅行家的预算

```
d1,c,d2,P,N=map(float,input().split())
a=[(0,P)]
for _ in range(int(N)):
    d,p=map(float,input().split())
    a.append((d,p))
a.sort()
start=0
cost=c*P
lenghth=c*d2
youjia=P
while start+lenghth<d1:
    d=[]
    for i in a:
        if start<i[0]<=start+lenghth:
            d.append(i)
    if not d:
        print("No Solution")
        exit(0)
    else:
        d.sort(key=lambda x:x[1])
        m,n=d[0]
        if n>=youjia:
            haoyou = (m - start) / d2
            cost+=haoyou*n
            youjia=n
            start=m
        else:
            d.sort()
            for i in d:
                if i[1]<youjia:
                    m=i[0];n=i[1]
                    break
            haoyou = (m - start) / d2
            cost+=c*n-(c-haoyou)*youjia
            youjia=n
            start=m
while True:
    if all((i[0] >= start and i[1] >= youjia) or(i[0]<start) for i in a):
        cost-=((lenghth-(d1-start))/d2)*youjia
        print(f"{cost:.2f}")
        break
    else:
     for i in a:
        if i[0]>=start and i[1]<youjia:
            haoyou=(i[0]-start)/d2
            cost += c * i[1] - (c - haoyou) * youjia
            youjia =i[1]
            start=i[0]
```

走山路

```
import heapq
m,n,p=map(int,input().split())
a=[list(input().split()) for _ in range(m)]
directions=[[-1,0],[1,0],[0,1],[0,-1]]
def dijkstra(u,v,s,t):
    pq=[(0,u,v)]
    visited=set()
    dis=[[float("inf")]*n for _ in range(m)]
    dis[u][v]=0
    if a[u][v]=="#" or a[s][t]=="#":
        return "NO"
    while pq:
        liang,x,y=heapq.heappop(pq)
        if x==s and y==t:
            return liang
        if (x,y) in visited:
            continue
        visited.add((x,y))
        for dx,dy in directions:
            nx=x+dx;ny=y+dy
            if 0<=nx<m and 0<=ny<n and (nx,ny) not in visited and a[nx]
[ny]!="#":
                new_liang=liang+abs(int(a[nx][ny])-int(a[x][y]))
                if new_liang<dis[nx][ny]:
                    dis[nx][ny]=new_liang
                    heapq.heappush(pq,(new_liang,nx,ny))
    return "NO"
for _ in range(p):
    x,y,o,q=map(int,input().split())
    print(dijkstra(x,y,o,q))
```

最大子矩阵

```
def max_submatrix(matrix):
    def kadane(arr):
        # max_ending_here 用于追踪到当前元素为止包含当前元素的最大子数组和。
        # max_so_far 用于存储迄今为止遇到的最大子数组和。
        max_end_here = max_so_far = arr[0]
        for x in arr[1:]:
            # 对于每个新元素，我们决定是开始一个新的子数组（仅包含当前元素 x），
            # 还是将当前元素添加到现有的子数组中。这一步是 Kadane 算法的核心。
            max_end_here = max(x, max_end_here + x)
            max_so_far = max(max_so_far, max_end_here)
        return max_so_far

    rows = len(matrix)
    cols = len(matrix[0])
    max_sum = float('-inf')

    for left in range(cols):
        temp = [0] * rows
        for right in range(left, cols):
            for row in range(rows):
                temp[row] += matrix[row][right]
            max_sum = max(max_sum, kadane(temp))
    return max_sum
```

```
n = int(input())
nums = []

while len(nums) < n * n:
    nums.extend(input().split())
matrix = [list(map(int, nums[i * n:(i + 1) * n])) for i in range(n)]

max_sum = max_submatrix(matrix)
print(max_sum)
```