蓝桥杯笔记

以后做事：切记切记切记，不要一直做新题，及时复习是最重要的。

一定一定拿一个国奖，6~7月准备两个项目，八月开始投简历。

真题一定要全部过一遍

一、基础复习阶段 3.4~3.10

二、刷题 3.10~3.20

三、刷真题，搞懂每一道真题 3.20~4.10

程序易错点：

1. 变量未声明
2. i,j,k变量写错了
3.

11+12+16+15+12+12 = 78

七夕祭

$$
\begin{gathered}
p[i]：i给i+1\ \ p[i]个糖果 \\
则ans=\sum_{i=1}^{n}|p[i]| \\
p[1]=a[1]-avg \\
递推\ \ p[2]=p[1]+a[2]-avg \\
p[3]=p[2]+a[3]-avg=p[1]+a[2]-avg+a[3]-avg \\
p[4]=p[3]+a[4]-avg=p[2]+p[3]-avg+a[4]-avg=p[1]+a[2]-avg+a[3]-avg+a[4]-avg \\
=\sum_{i=2}^{4}(a[i]-avg)-p[1] \\
ans=
\end{gathered}
\tag{1}
$$

```python
from collections import defaultdict

r = defaultdict(int)
c = defaultdict(int)
s = defaultdict(int)

def get_ans(a, n):
    ans = 0
    avg = sum(a.values()) // n
    for i in a:
        a[i] -= avg
    s[1] = 0
    prev_sum = 0
    sorted_values = sorted(a.values())
    mid = sorted_values[n // 2]
    for i in range(2, n + 1):
        prev_sum += a[i]
        s[i] = prev_sum
    for i in s.values():
        ans += abs(i - mid)
    return ans

```

```python
n, m, t = map(int, input().split())
for _ in range(t):
    x, y = map(int, input().split())
    r[x] += 1
    c[y] += 1

if t % n != 0 and t % m != 0:
    print("impossible")
elif t % n == 0 and t % m == 0:
    print("both", get_ans(r, n) + get_ans(c, m))
elif t % n == 0:
    print("row", get_ans(r, n))
else:
    print("column", get_ans(c, m))
```

模板

基础+杂项

快速排序

```python
def quick_sort(q, l, r):
    if l>=r:
        return
    i,j,x=l-1,r+1,q[(l+r)>>1]
    while i<j:
        i+=1
        while q[i]<x:
            i+=1
        j-=1
        while q[j]>x:
            j-=1
        if i<j:
            q[i], q[j] = q[j], q[i]
    quick_sort(q, l, j)
    quick_sort(q, j+1, r)
n=int(input())
arr=list(map(int, input().split()))
quick_sort(arr,0,n-1)
print(" ".join(map(str, arr)))
```

归并排序

j = mid+1 !!!

```python
def merge_sort(q,l,r):
    if l>=r:
        return
    mid = (l+r)>>1
    merge_sort(q, l, mid)
    merge_sort(q, mid+1, r)
    i,j,k = l, mid+1, 0
    tmp = [0]*(r-l+1)
    while i<=mid and j<=r:
        if q[i]<q[j]:
            tmp[k]=q[i]
            k+=1
            i+=1
        else:
            tmp[k]=q[j]
```

```
            k+=1
            j+=1
    while i<=mid:
        tmp[k]=q[i]
        i+=1
        k+=1
    while j<=r:
        tmp[k]=q[j]
        j+=1
        k+=1
    for i in range(l, r+1):
        q[i]=tmp[i-l]
n=int(input())
arr=list(map(int, input().split()))
merge_sort(arr, 0, n-1)
print(" ".join(map(str, arr)))
```

二分

```
def check(x):
    # 检查 x 是否满足某种性质
    pass  # 这里需要根据具体的情况实现

def bsearch_1(l, r):
    while l < r:
        mid = (l + r) // 2
        if check(mid):
            r = mid
        else:
            l = mid + 1
    return l

def bsearch_2(l, r):
    while l < r:
        mid = (l + r + 1) // 2
        if check(mid):
            l = mid
        else:
            r = mid - 1
    return l

# 示例用法
# 首先定义 check 函数来检查性质
# 然后使用 bsearch_1 或 bsearch_2 来进行二分搜索
```

浮点数二分

```python
def check(x):
    # 检查x是否满足某种性质
    pass  # 这里需要根据具体情况实现check函数

def bsearch_3(l, r):
    eps = 1e-6  # eps 表示精度，取决于题目对精度的要求
    while r - l > eps:
        mid = (l + r) / 2
        if check(mid):
            r = mid
        else:
            l = mid
    return l
```

一维前缀和

```python
def prefixSum(arr):
    n = len(arr)
    prefixSum = [0] * n
    prefixSum[0] = arr[0]

    for i in range(1, n):
        prefixSum[i] = prefixSum[i-1] + arr[i]

    for i in range(n):
        print(prefixSum[i], end=" ")

arr = [1, 2, 3, 4, 5]
prefixSum(arr)
```

二维前缀和

```python
def prefixSum2D(arr):
    n = len(arr)
    m = len(arr[0])
    prefixSum = [[0] * m for _ in range(n)]

    # 计算第一行的前缀和
    prefixSum[0][0] = arr[0][0]
    for j in range(1, m):
        prefixSum[0][j] = prefixSum[0][j-1] + arr[0][j]

    # 计算第一列的前缀和
    for i in range(1, n):
        prefixSum[i][0] = prefixSum[i-1][0] + arr[i][0]

    # 计算其他部分的前缀和
    for i in range(1, n):
        for j in range(1, m):
            prefixSum[i][j] = prefixSum[i-1][j] + prefixSum[i][j-1] - prefixSum[i-1][j-1] + arr[i][j]

    return prefixSum

def submatrixSum(prefixSum, x1, y1, x2, y2):
    return prefixSum[x2][y2] - prefixSum[x1-1][y2] - prefixSum[x2][y1-1] + prefixSum[x1-1][y1-1]

arr = [[1, 2, 3, 4],
       [5, 6, 7, 8],
```

```
        [9, 10, 11, 12]]

prefixSum = prefixSum2D(arr)
print(prefixSum)

x1, y1 = 1, 1
x2, y2 = 2, 2
submatrixSum = submatrixSum(prefixSum, x1, y1, x2, y2)
print(submatrixSum)
```

一维差分

差分和前缀和是逆运算。

需要计算某一段区间+−操作时，运用差分操作 `updateRange` 需要先构造差分数组

$s[i] = s[i-1] + a[i]$ 逆运算 $d[i] = a[i] - a[i-1]$ （原数组看作为前缀和数组）

```
def updateRange(B, l, r, c):
    B[l] += c
    B[r + 1] -= c

def printArray(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()

n= 5
B = [0] * (n + 1)

updateRange(B, 1, 3, 2)
updateRange(B, 2, 4, 3)

printArray(B)
```

二维差分

```
def updateSubmatrix(S, x1, y1, x2, y2, c):
    S[x1][y1] += c
    S[x2 + 1][y1] -= c
    S[x1][y2 + 1] -= c
    S[x2 + 1][y2 + 1] += c

def printMatrix(S):
    rows = len(S)
    cols = len(S[0])
    for i in range(rows):
        for j in range(cols):
            print(S[i][j], end=" ")
        print()

rows = 3
cols = 3
S = [[0] * cols for _ in range(rows + 1)]

updateSubmatrix(S, 0, 0, 1, 1, 1)
updateSubmatrix(S, 1, 1, 2, 2, 2)

printMatrix(S)
```

双指针

```
for i in range(n):
    j = 0
    while j < i and check(j, i):
        j += 1
    # 具体问题的逻辑

# 常见问题分类：
# (1) 对于一个序列，用两个指针维护一段区间
# (2) 对于两个序列，维护某种次序，比如归并排序中合并两个有序序列的操作
```

位运算

```
原码，反码，补码
求n二进制表示中第k位数字: n >> k & 1
返回n的最后一位1：lowbit(n) = n & -n 树状数组基本操作
```

离散化

```
alls = []  # 存储所有待离散化的值

# 将所有值排序
alls.sort()

# 去掉重复元素
alls = list(set(alls))

# 二分求出x对应的离散化的值
def find(x):
    l, r = 0, len(alls) - 1
    while l < r:
        mid = (l + r) // 2
        if alls[mid] >= x:
            r = mid
        else:
            l = mid + 1
    return r + 1  # 映射到1, 2, ...n

```

区间合并

```
def merge(segs):
    segs.sort()  # 区间左端点排序
    res = []
    st, ed = -2e9, -2e9
    for seg in segs:
        if ed < seg[0]:
            if st != -2e9:
                res.append((st, ed))
            st, ed = seg[0], seg[1]
        else:
            ed = max(ed, seg[1])
    if st != -2e9:
        res.append((st, ed))
    segs[:] = res
```

数据结构

单链表

```
N = int(1e5+10)
e=[0]*N
ne=[0]*N
head=-1
idx=1

def insert(x):
    global idx, head
    e[idx]=x
    ne[idx]=head
    head=idx
    idx+=1

def add(k, x):
    global idx
    e[idx]=x
    ne[idx]=ne[k]
    ne[k]=idx
    idx+=1

def remove(k):
    global idx, head
    if k==0:
        head = ne[head]
    else:
        ne[k]=ne[ne[k]]
n = int(input())
for _ in range(n):
    s = input().split()
    op=s[0]
    if op=='H':
        insert(int(s[1]))
    elif op=='I':
        add(int(s[1]), int(s[2]))
    else:
        remove(int(s[1]))

i=head
while i!=-1:
    print(e[i], end=' ')
    i=ne[i]
```

双链表

```
N = int(1e5+10)
e=[0]*N
l=[0]*N
r=[0]*N
idx=0
def init():
    global idx
    r[2]=1
    l[1]=2
    idx=3
def insert(k, x):
    global idx
    e[idx]=x
    l[idx]=k
    r[idx]=r[k]
```

```
    l[r[k]]=idx
    r[k]=idx
    idx+=1
def remove(k):
    l[r[k]]=l[k]
    r[l[k]]=r[k]
init()
m=int(input())
for _ in range(m):
    s=input().split()
    if s[0]=='L':
        x=int(s[1])
        insert(2,x)
    elif s[0]=='R':
        x=int(s[1])
        insert(l[1],x)
    elif s[0]=='D':
        k=int(s[1])+2
        remove(k)
    elif s[0]=='IL':
        k=int(s[1])+2
        x=int(s[2])
        insert(l[k],x)
    elif s[0]=='IR':
        k=int(s[1])+2
        x=int(s[2])
        insert(k,x)
i=2
while i!=0:
    if i==2 or i==1:
        i=r[i]
        continue
    print(e[i],end=" ")
    i=r[i]
```

栈

```
N = int(1e5+10)  # 假设N的值为100

stk = [0] * N
tt = 0

# 向栈顶插入一个数
tt += 1
stk[tt] = x

# 从栈顶弹出一个数
tt -= 1

# 栈顶的值
stk[tt]

# 判断栈是否为空
if tt > 0:
    pass
```

队列

```python
N = 100  # 假设N的值为100

q = [0] * N
hh = 0
tt = -1

# 向队尾插入一个数
tt += 1
q[tt] = x

# 从队头弹出一个数
hh += 1

# 队头的值
q[hh]

# 判断队列是否为空
if hh <= tt:
    pass

N = 100  # 假设N的值为100

q = [0] * N
hh = 0
tt = 0

# 向队尾插入一个数
q[tt] = x
tt += 1
if tt == N:
    tt = 0

# 从队头弹出一个数
hh += 1
if hh == N:
    hh = 0

# 队头的值
q[hh]

# 判断队列是否为空
if hh != tt:
    pass
```

单调栈

```python
tt = 0
stk = [0] * (n + 1)

for i in range(1, n + 1):
    while tt and check(stk[tt], i):
        tt -= 1
    stk[tt + 1] = i
    tt += 1
```

单调队列

```python
n = 10  # 假设n的值为10

hh = 0
tt = -1
q = [0] * n

for i in range(n):
    while hh <= tt and check_out(q[hh]):
        hh += 1
    while hh <= tt and check(q[tt], i):
        tt -= 1
    q[tt + 1] = i
    tt += 1

N = int(1e6+10)
q=[0 for _ in range(N)]
n,k=map(int, input().split())
a=[0]+[int(x) for x in input().split()]

hh,tt=0,-1
for i in range(1,n+1):
    if hh<=tt and i-q[hh]+1>k:
        hh+=1
    while hh<=tt and a[q[tt]] >= a[i]:
        tt-=1
    tt+=1
    q[tt]=i
    if i >= k:
        print(a[q[hh]], end=" ")

print()
hh,tt=0,-1
for i in range(1,n+1):
    if hh<=tt and i-q[hh]+1>k:
        hh+=1
    while hh<=tt and a[q[tt]] <= a[i]:
        tt-=1
    tt+=1
    q[tt]=i
    if i>=k:
        print(a[q[hh]], end=" ")
```

KMP

```python
m = len(p)  # 假设p为模板串，长度为m
n = len(s)  # 假设s为模式串，长度为n

ne = [0] * (m + 1)  # 初始化ne数组

# 求Next数组
j = 0
for i in range(2, m + 1):
    while j and p[i] != p[j + 1]:
        j = ne[j]
    if p[i] == p[j + 1]:
        j += 1
    ne[i] = j

# 匹配
```

```python
j = 0
for i in range(1, n + 1):
    while j and s[i] != p[j + 1]:
        j = ne[j]
    if s[i] == p[j + 1]:
        j += 1
    if j == m:
        j = ne[j]
        # 匹配成功后的逻辑
```

Tire

```python
N = 100010
son = [[0] * 26 for _ in range(N)]
cnt = [0] * N
idx = 0

# 0号点既是根节点，又是空节点
# son[][]存储树中每个节点的子节点
# cnt[]存储以每个节点结尾的单词数量
# 插入一个字符串
def insert(s):
    global idx
    p = 0
    for i in range(len(s)):
        u = ord(s[i]) - ord('a')
        if not son[p][u]:
            idx += 1
            son[p][u] = idx
        p = son[p][u]
    cnt[p] += 1

# 查询字符串出现的次数
def query(s):
    p = 0
    for i in range(len(s)):
        u = ord(s[i]) - ord('a')
        if not son[p][u]:
            return 0
        p = son[p][u]
    return cnt[p]

```

并查集

```python
N = 1000005  # 假设N的值为1000005

p = [0] * N  # 初始化p数组

# 返回x的祖宗节点
def find(x):
    if p[x] != x:
        p[x] = find(p[x])
    return p[x]

# 初始化，假定节点编号是1~n
for i in range(1, n + 1):
    p[i] = i

# 合并a和b所在的两个集合
p[find(a)] = find(b)
```

维护size信息

```python
# Python中没有类似于C++的数组声明方式，我们直接使用列表来代替
p = [i for i in range(N)]
size = [1] * N

# 返回x的祖宗节点
def find(x):
    if p[x] != x:
        p[x] = find(p[x])
    return p[x]

# 初始化，假定节点编号是1~n
n = N # 假设n是提前定义好的
for i in range(1, n + 1):
    p[i] = i
    size[i] = 1

# 合并a和b所在的两个集合
p[find(a)] = find(b)
size[b] += size[a]
```

维护到祖宗节点距离的并查集

```python
# Python中没有类似于C++的数组声明方式，我们直接使用列表来代替
p = [i for i in range(N)]
d = [0] * N

# 返回x的祖宗节点
def find(x):
    if p[x] != x:
        u = find(p[x])
        d[x] += d[p[x]]
        p[x] = u
    return p[x]

# 初始化，假定节点编号是1~n
n = N # 假设n是提前定义好的
for i in range(1, n + 1):
    p[i] = i
    d[i] = 0

# 合并a和b所在的两个集合
p[find(a)] = find(b)
d[find(a)] = distance  # 根据具体问题，初始化find(a)的偏移量
```

堆

```python
# h[N] 存储堆中的值，h[1]是堆顶，x的左儿子是2x，右儿子是2x + 1
# ph pos[k] 存储第k个插入的点在堆中的位置
# hp ord[k] 存储堆中下标是k的点是第几个插入的

N = 1000005  # 假设N的值为1000005

h = [0] * N  # 初始化h数组
ph = [0] * N  # 初始化ph数组
hp = [0] * N  # 初始化hp数组
size = 0  # 初始化size为0

# 交换两个点，及其映射关系
```

```python
def heap_swap(a, b):
    ph[hp[a]], ph[hp[b]] = ph[hp[b]], ph[hp[a]]
    hp[a], hp[b] = hp[b], hp[a]
    h[a], h[b] = h[b], h[a]


def down(u):
    t = u
    if u * 2 <= size and h[u * 2] < h[t]:
        t = u * 2
    if u * 2 + 1 <= size and h[u * 2 + 1] < h[t]:
        t = u * 2 + 1
    if u != t:
        heap_swap(u, t)
        down(t)

def up(u):
    while u // 2 and h[u] < h[u // 2]:
        heap_swap(u, u // 2)
        u >>= 1

# O(n)建堆
for i in range(n // 2, 0, -1):
    down(i)

def add_element(x):
    global size
    size += 1
    h[size] = x
    ph[size] = size
    hp[size] = size
    up(size)

def get_heap_top():
    return h[1]

def delete_element(k):
    global size
    heap_swap(k, size)
    size -= 1
    down(k)
    up(k)

def update_element(k, x):
    h[k] = x
    down(k)
    up(k)

# 添加元素
add_element(x)

# 堆顶元素
heap_top = get_heap_top()

# 删除第size个节点
delete_element(1)

# 删除第k个节点
delete_element(k)

# 将第k个元素赋值为x
update_element(k, x)
```

哈希

(1)拉链法

```python
N = 1000005  # 假设N的值为1000005

h = [-1] * N  # 初始化h数组为-1
e = [0] * N  # 初始化e数组
ne = [0] * N  # 初始化ne数组
idx = 0  # 初始化idx为0

# 向哈希表中插入一个数
def insert(x):
    k = (x % N + N) % N
    e[idx] = x
    ne[idx] = h[k]
    h[k] = idx
    idx += 1

# 在哈希表中查询某个数是否存在
def find(x):
    k = (x % N + N) % N
    i = h[k]
    while i != -1:
        if e[i] == x:
            return True
        i = ne[i]
    return False
```

(2) 开放寻址法

```python
N = 1000005  # 假设N的值为1000005

h = [0] * N  # 初始化h数组

# 如果x在哈希表中，返回x的下标；如果x不在哈希表中，返回x应该插入的位置
def find(x):
    t = (x % N + N) % N
    while h[t] != 0 and h[t] != x:
        t += 1
        if t == N:
            t = 0
    return t
```

字符串哈希

```python
N = 1000005  # 假设N的值为1000005
P = 131  # 或者可以设置为13331

h = [0] * N  # 初始化h数组
p = [0] * N  # 初始化p数组

# 初始化
p[0] = 1
for i in range(1, n + 1):
    h[i] = h[i - 1] * P + ord(str[i])
    p[i] = p[i - 1] * P

# 计算子串 str[l ~ r] 的哈希值
def get(l, r):
```

```
    return h[r] - h[l - 1] * p[r - l + 1]
```

图论

树的存储

邻接矩阵

```python
# 创建一个二维列表表示邻接矩阵
n = 10  # 顶点数量
g = [[0] * n for _ in range(n)]

# 添加一条边a->b
def add_edge(a, b):
    g[a][b] = 1

# 初始化
g = [[0] * n for _ in range(n)]
```

邻接表

```python
# 创建一个列表表示邻接表
n = 10  # 顶点数量
h = [-1] * n
e = [0] * n
ne = [0] * n
idx = 0

# 添加一条边a->b
def add_edge(a, b):
    global idx
    e[idx] = b
    ne[idx] = h[a]
    h[a] = idx
    idx += 1

# 初始化
idx = 0
h = [-1] * n
```

树和图的存储

```python
# 邻接表表示的图
N = 100010  # 根据具体需求设置合适的最大节点数量

# 对于每个点k，开一个单链表，存储k所有可以走到的点。h[k]存储这个单链表的头结点
h = [-1] * N
# 存储边的目标节点
e = [0] * N
# 存储下一条边的索引
ne = [0] * N
# 边的索引
idx = 0

# 添加一条边a->b
def add(a, b):
    global idx
    e[idx] = b
    ne[idx] = h[a]
    h[a] = idx
```

```
    idx += 1

# 初始化
idx = 0
for i in range(N):
    h[i] = -1

```

树和图的遍历

DFS

```
def dfs(u):
    st[u] = True  # st[u] 表示点u已经被遍历过
    for i in range(h[u], -1, -1):
        j = e[i]
        if not st[j]:
            dfs(j)
```

BFS

```
from queue import Queue

q = Queue()
st[1] = True  # 表示1号点已经被遍历过
q.put(1)

while not q.empty():
    t = q.get()
    for i in range(h[t], -1, -1):
        j = e[i]
        if not st[j]:
            st[j] = True  # 表示点j已经被遍历过
            q.put(j)
```

拓扑排序

```
def topsort():
    hh = 0
    tt = -1
    # d[i] 存储点i的入度
    for i in range(1, n + 1):
        if d[i] == 0:
            q.append(i)
            tt += 1
    while hh <= tt:
        t = q[hh]
        hh += 1

        for i in range(h[t], -1, -1):
            j = e[i]
            d[j] -= 1
            if d[j] == 0:
                q.append(j)
                tt += 1
    # 如果所有点都入队了，说明存在拓扑序列；否则不存在拓扑序列。
    return tt == n - 1
```

LCA

```python
def lca(x,y):
    if dep[x] < dep[y]:
        x,y = y,x
    d = dep[x]-dep[y]
    while d:  # 循环直到深度差为 0
        v = d & -d  # 获取 d 的最低位的 1 所在的位置
        i = v.bit_length() - 1  # 计算最低位的位置索引
        x = fa[i][x]  # 将节点 x 上移到和节点 y 同一深度
        d -= v  # 更新深度差
    if x==y:
        return x
    for k in range(K-1, -1, -1):
        if fa[k][x] != fa[k][y]:
            x = fa[k][x]
            y = fa[k][y]
    return fa[0][x]

```

最短路

单元最短路

所有边权为正：

朴素版Dijkstra $O(n^2)$,堆优化版的Dijkstra $O(mlogn)$,

存在负权边

Bellman-Ford $O(nm)$, SPFA 队列优化的Bellman-Ford, 一般情况：$O(m)$ 最坏情况：$O(nm)$

多元汇最短路：

Floyed $O(n^3)$

朴素Dijkstra

```python
N = int(5e2)+10
INF = 0x3f3f3f3f
g = [[INF]*N for _ in range(N)]
#g = defaultdict(lambda:defaultdict(lambda:INF)) 同样的效果
dis = [INF]*N
st = [False]*N

def dijkstra():
    dis[1]=0
    for i in range(n-1):
        t=-1
        for j in range(1,n+1):
            if not st[j] and (t==-1 or dis[j]<dis[t]):
                t=j
        for j in range(1,n+1):
            dis[j]=min(dis[j], dis[t] + g[t][j])
        st[t]=True
    if dis[n]==INF:
        return -1
    return dis[n]

n, m = map(int, input().split())
for _ in range(m):
    x, y, z = map(int ,input().split())
    g[x][y] = min(g[x][y], z)

```

```
26
27 print(dijkstra())
```

堆优化版Dijkstra

```
1  from heapq import *
2  import sys
3  input = lambda:sys.stdin.readline().strip()
4  N = 150010
5  INF = 0x3f3f3f3f
6  e, ne, head, w, dis= [0]*N, [0]*N, [-1]*N, [0]*N, [INF]*N
7  st = [False]*N
8  idx=0
9
10 def add(a, b, x):
11     global idx
12     e[idx]=b
13     w[idx]=x
14     ne[idx]=head[a]
15     head[a]=idx
16     idx+=1
17
18 def dijkstra():
19     dis[1]=0
20     h=[]
21     heappush(h, (0,1))
22     while h:
23         dist, ver = heappop(h)
24         if st[ver]: continue
25         st[ver]=True
26         i=head[ver]
27         while i!=-1:
28             j=e[i]
29             if not st[j] and dis[j]>dist+w[i]:
30                 dis[j]=dist+w[i]
31                 heappush(h, (dis[j], j))
32             i=ne[i]
33     if dis[n]==INF:
34         print(-1)
35     else:
36         print(dis[n])
37
38 n, m = map(int, input().split())
39 for _ in range(m):
40     x, y, z = map(int, input().split())
41     add(x,y,z)
42
43 dijkstra()
```

另一种写法

```
1  from heapq import *
2  from collections import defaultdict
3  import sys
4  input=lambda:sys.stdin.readline().strip()
5  N, INF = 150010, 0x3f3f3f3f
6  dis, st = [INF]*N, [False]*N
7
8  g=defaultdict(list)
9
10
11 def dijkstra():
```

```
    dis[1]=0
    h=[]
    heappush(h, (0, 1))
    while h:
        dist, ver = heappop(h)
        if st[ver]:
            continue
        st[ver]=True
        for y, z in g[ver]:
            if dis[y]>dis[ver]+z:
                dis[y]=dis[ver]+z
                heappush(h, (dis[y], y))
    if dis[n]==INF:
        print(-1)
    else:
        print(dis[n])

n, m = map(int, input().split())
for _ in range(m):
    x,y,z = map(int, input().split())
    g[x].append((y, z))

dijkstra()
```

Bellman-Ford

```
# n表示点数，m表示边数
dist = [float('inf')] * (n + 1)  # dist[x]存储1到x的最短路距离
# 边，a表示出点，b表示入点，w表示边的权重
edges = []

# 求1到n的最短路距离，如果无法从1走到n，则返回-1。
def bellman_ford():
    dist[1] = 0
    # 如果第n次迭代仍然会松弛三角不等式，就说明存在一条长度是n+1的最短路径，由抽屉原理，路径中至少存在两个相同的点，说明图中存在负权回路。
    for i in range(n):
        for j in range(m):
            a, b, w = edges[j]['a'], edges[j]['b'], edges[j]['w']
            if dist[b] > dist[a] + w:
                dist[b] = dist[a] + w
    if dist[n] > float('inf') / 2:
        return -1
    return dist[n]
```

SPFA

```
from collections import defaultdict, deque
import sys
input = lambda:sys.stdin.readline().strip()

N, INF = int(1e5+10), 0x3f3f3f3f

dis, st = [INF]*N, [False]*N
g=defaultdict(list)

def spfa():
    q=deque()
    q.append(1)
    st[1]=True
    dis[1]=0
```

```
    while q:
        x=q.popleft()
        st[x]=False
        for y, z in g[x]:
            if dis[y]>dis[x]+z:
                dis[y]=dis[x]+z
                if not st[y]:
                    st[y]=True
                    q.append(y)
    if dis[n]==INF:
        print('impossible')
    else:
        print(dis[n])

n, m = map(int, input().split())
for _ in range(m):
    x, y, z = map(int, input().split())
    g[x].append((y, z))

spfa()
```

判断负环

```
from collections import deque

N = int(1e5+10)
INF = 0x3f3f3f3f

n = 0  # 总点数
h = [-1] * N  # 邻接表存储所有边
e, ne, w = [0] * N, [0] * N, [0] * N
idx = 0
dist, cnt = [INF] * N, [0] * N  # dist[x]存储1号点到x的最短距离，cnt[x]存储1到x的最短路中经过的点数
st = [False] * N  # 存储每个点是否在队列中

# 如果存在负环，则返回True，否则返回False。
def spfa():
    # 不需要初始化dist数组
    # 原理：如果某条最短路径上有n个点（除了自己），那么加上自己之后一共有n+1个点，由抽屉原理一定有两个点相同，所以存在环
    q = deque()
    for i in range(1, n + 1):
        q.append(i)
        st[i] = True
    while q:
        t = q.popleft()
        st[t] = False
        i = h[t]
        while i != -1:
            j = e[i]
            if dist[j] > dist[t] + w[i]:
                dist[j] = dist[t] + w[i]
                cnt[j] = cnt[t] + 1
                if cnt[j] >= n:
                    return True  # 如果从1号点到x的最短路中包含至少n个点（不包括自己），则说明存在环
                if not st[j]:
                    q.append(j)
                    st[j] = True
            i = ne[i]
    return False

```

```
# Example usage:
# n = 5
# h = [-1, 2, 1, 4, 3, -1]
# e = [0, 2, 1, 4, 3, 0]
# ne = [1, -1, 3, -1, -1, 2]
# w = [0, 1, 2, 3, 4, 5]
# if spfa():
#     print("Exist negative cycle")
# else:
#     print("No negative cycle")
```

floyd

```
INF = float('inf')

# 初始化距离矩阵d，d[a][b]表示a到b的最短距离
def initialize(n):
    d = [[0 if i == j else INF for j in range(n)] for i in range(n)]
    return d

# Floyd算法求解最短路径
def floyd(d, n):
    for k in range(n):
        for i in range(n):
            for j in range(n):
                d[i][j] = min(d[i][j], d[i][k] + d[k][j])
    return d

# Example usage:
# n = 5
# d = initialize(n)
# d = floyd(d, n)
# print(d)
```

Prim

```
INF = float('inf')

# Prim算法求解最小生成树的权重之和
def prim(n, g):
    dist = [INF] * (n + 1)
    st = [False] * (n + 1)
    res = 0
    for i in range(n):
        t = -1
        for j in range(1, n + 1):
            if not st[j] and (t == -1 or dist[t] > dist[j]):
                t = j
        if i and dist[t] == INF:
            return INF
        if i:
            res += dist[t]
        st[t] = True
        for j in range(1, n + 1):
            dist[j] = min(dist[j], g[t][j])
    return res

# Example usage:
# n = 5
# g = [[0] * (n + 1) for _ in range(n + 1)]
# dist = prim(n, g)
```

```
# print(dist)

```

Kruskal

```
INF = float('inf')

# 并查集的查找操作
def find(x, p):
    if p[x] != x:
        p[x] = find(p[x], p)
    return p[x]

# Kruskal算法求解最小生成树的权重之和
def kruskal(n, m, edges):
    edges.sort(key=lambda x: x[2])  # 按照边权重对边进行排序
    p = [i for i in range(n + 1)]   # 初始化并查集的父节点数组
    res = 0
    cnt = 0
    for edge in edges:
        a, b, w = edge
        a = find(a, p)
        b = find(b, p)
        if a != b:  # 如果两个连通块不连通，则将这两个连通块合并
            p[a] = b
            res += w
            cnt += 1
    if cnt < n - 1:
        return INF
    return res

# Example usage:
# n, m = 5, 7
# edges = [(1, 2, 2), (1, 3, 5), (1, 4, 6), (2, 3, 1), (2, 4, 3), (3, 4, 4), (4, 5, 7)]
# min_spanning_tree_weight = kruskal(n, m, edges)
# print(min_spanning_tree_weight)
```

染色法

```
# 定义全局变量
N = 10005  # 根据需要修改
M = 20005  # 根据需要修改
h = [-1] * N  # 邻接表头
e, ne = [0] * M, [0] * M  # 邻接表存储图
idx = 0  # 邻接表索引
color = [-1] * N  # 表示每个点的颜色，-1表示未染色，0表示白色，1表示黑色

# 深度优先搜索进行着色
def dfs(u, c):
    color[u] = c
    i = h[u]
    while i != -1:
        j = e[i]
        if color[j] == -1:
            if not dfs(j, 1 - c):
                return False
        elif color[j] == c:
            return False
        i = ne[i]
    return True

```

```python
# 检查图是否是二分图
def check(n):
    flag = True
    for i in range(1, n + 1):
        if color[i] == -1:
            if not dfs(i, 0):
                flag = False
                break
    return flag

# Example usage:
# n = 5
# h = [-1, 2, -1, 4, -1, 4]  # 邻接表头
# e = [0, 3, 0, 1, 0, 4]  # 邻接表存储图
# ne = [1, -1, -1, 2, 5, -1]  # 邻接表存储图
# if check(n):
#     print("Graph is a bipartite graph.")
# else:
#     print("Graph is not a bipartite graph.")
```

匈牙利算法

```python
N = 1005  # 根据需要修改
M = 100005  # 根据需要修改
h = [-1] * N  # 邻接表头
e, ne = [0] * M, [0] * M  # 邻接表存储所有边，只存储从第二个集合指向第一个集合的边
idx = 0  # 邻接表索引
match = [0] * N  # 存储第二个集合中的每个点当前匹配的第一个集合中的点是哪个
st = [False] * N  # 表示第二个集合中的每个点是否已经被遍历过

# 匈牙利算法中的深度优先搜索寻找增广路
def find(x):
    for i in range(h[x]):
        j = e[i]
        if not st[j]:
            st[j] = True
            if match[j] == 0 or find(match[j]):
                match[j] = x
                return True
    return False

# 求最大匹配数
def hungarian(n1, n2):
    res = 0
    for i in range(1, n1 + 1):
        st = [False] * N  # 重置st数组
        if find(i):
            res += 1
    return res

# Example usage:
# n1, n2 = 5, 5
# h = [-1, 2, 3, 4, 0, 0]  # 邻接表头
# e = [1, 2, 3, 4, 0, 0]  # 邻接表存储所有边，只存储从第二个集合指向第一个集合的边
# ne = [1, 2, 3, 4, -1, -1]  # 邻接表存储所有边，只存储从第二个集合指向第一个集合的边
# max_matching = hungarian(n1, n2)
# print(max_matching)
```

数学

试除法判定质数

```python
# 判断一个数是否是素数
def is_prime(x):
    if x < 2:
        return False
    for i in range(2, int(x ** 0.5) + 1):
        if x % i == 0:
            return False
    return True

# Example usage:
# result = is_prime(17)
# print(result)  # Output: True
```

试除法分解质因数

```python
# 因数分解函数
def divide(x):
    i = 2
    while i <= x ** 0.5:
        if x % i == 0:
            s = 0
            while x % i == 0:
                x //= i
                s += 1
            print(i, s)
        i += 1
    if x > 1:
        print(x, 1)
    print()

# Example usage:
# divide(36)
```

朴素筛法求素数

```python
N = 1000005  # 根据需要修改
primes = []  # 存储所有素数
st = [False] * N  # st[x]存储x是否被筛掉

# 筛素数函数
def get_primes(n):
    global primes
    global st
    for i in range(2, n + 1):
        if not st[i]:
            primes.append(i)
            for j in range(i, n + 1, i):
                st[j] = True

# Example usage:
# get_primes(100)
# print(primes)
```

线性筛法求素数

```python
N = 1000005  # 根据需要修改
primes = []  # 存储所有素数
st = [False] * N  # st[x]存储x是否被筛掉

# 筛素数函数
def get_primes(n):
    global primes
    global st
    for i in range(2, n + 1):
        if not st[i]:
            primes.append(i)
        for j in range(len(primes)):
            if primes[j] * i > n:
                break
            st[primes[j] * i] = True
            if i % primes[j] == 0:
                break

# Example usage:
# get_primes(100)
# print(primes)
```

试除法求所有约数

```python
# 获取因数函数
def get_divisors(x):
    res = []
    i = 1
    while i <= x ** 0.5:
        if x % i == 0:
            res.append(i)
            if i != x // i:
                res.append(x // i)
        i += 1
    res.sort()
    return res

# Example usage:
# divisors = get_divisors(36)
# print(divisors)
```

约数个数和约数之和

```
如果 N = p1^c1 * p2^c2 * ... *pk^ck
约数个数：  (c1 + 1) * (c2 + 1) * ... * (ck + 1)
约数之和：  (p1^0 + p1^1 + ... + p1^c1) * ... * (pk^0 + pk^1 + ... + pk^ck)
```

gcd

```python
def gcd(a, b):
    return gcd(b, a % b) if b else a
```

求欧拉函数

```python
# 计算欧拉函数
def phi(x):
    res = x
    i = 2
    while i <= x ** 0.5:
        if x % i == 0:
            res = res // i * (i - 1)
            while x % i == 0:
                x //= i
        i += 1
    if x > 1:
        res = res // x * (x - 1)
    return res

# Example usage:
# result = phi(36)
# print(result)
```

筛法求欧拉函数

```python
N = 1000005  # 根据需要修改
primes = []  # 存储所有素数
euler = [0] * N  # 存储每个数的欧拉函数
st = [False] * N  # st[x]存储x是否被筛掉

# 获取欧拉函数数组
def get_eulers(n):
    global primes
    global euler
    global st
    euler[1] = 1
    for i in range(2, n + 1):
        if not st[i]:
            primes.append(i)
            euler[i] = i - 1
        for j in range(len(primes)):
            if primes[j] * i > n:
                break
            t = primes[j] * i
            st[t] = True
            if i % primes[j] == 0:
                euler[t] = euler[i] * primes[j]
                break
            euler[t] = euler[i] * (primes[j] - 1)

# Example usage:
# get_eulers(100)
# print(euler)
```

快速幂

```python
# 快速幂函数
def qmi(m, k, p):
    res, t = 1 % p, m
    while k:
        if k & 1:
            res = res * t % p
        t = t * t % p
        k >>= 1
    return res

# Example usage:
# result = qmi(2, 10, 1000000007)
# print(result)
```

拓展欧几里得

```python
# 求 x, y，使得 ax + by = gcd(a, b)
def exgcd(a, b, x, y):
    if b == 0:
        x[0], y[0] = 1, 0
        return a
    d = exgcd(b, a % b, y, x)
    y[0] -= (a // b) * x[0]
    return d

# Example usage:
# x = [0]
# y = [0]
# gcd = exgcd(30, 20, x, y)
# print("x:", x[0], "y:", y[0], "gcd:", gcd)
```

高斯消元

```python
eps = 1e-8  # 根据需要调整

# a 是增广矩阵，n 是矩阵维度
def gauss(a, n):
    c, r = 0, 0
    for c in range(n):
        t = r
        for i in range(r, n):
            if abs(a[i][c]) > abs(a[t][c]):
                t = i
        if abs(a[t][c]) < eps:
            continue
        for i in range(c, n + 1):
            a[r][i], a[t][i] = a[t][i], a[r][i]
        for i in range(n, c - 1, -1):
            a[r][i] //= a[r][c]
        for i in range(r + 1, n):
            if abs(a[i][c]) > eps:
                for j in range(n, c - 1, -1):
                    a[i][j] -= a[r][j] * a[i][c]
        r += 1

    if r < n:
        for i in range(r, n):
            if abs(a[i][n]) > eps:
                return 2  # 无解
        return 1  # 有无穷多组解
```

```
    for i in range(n - 1, -1, -1):
        for j in range(i + 1, n):
            a[i][n] -= a[i][j] * a[j][n]
    return 0  # 有唯一解

# Example usage:
# a = [[2, 1, -1, 8], [-3, -1, 2, -11], [-2, 1, 2, -3]]
# n = 3
# result = gauss(a, n)
# print(result)
```

递归法求组合数

```
N = 1005  # 根据需要调整
mod = 1000000007  # 根据需要调整
c = [[0] * N for _ in range(N)]  # 初始化二维数组

# 计算组合数
for i in range(N):
    for j in range(i + 1):
        if j == 0:
            c[i][j] = 1
        else:
            c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % mod

# Example usage:
# result = c[5][2]  # 获取从5个苹果中选2个的方案数
# print(result)
```

通过预处理逆元的方式求组合数

```
mod = 1000000007  # 根据需要调整
N = 1005  # 根据需要调整
fact = [0] * N  # 存储阶乘的余数
infact = [0] * N  # 存储阶乘逆元的余数

# 快速幂模板
def qmi(a, k, p):
    res = 1
    while k:
        if k & 1:
            res = (res * a) % p
        a = (a * a) % p
        k >>= 1
    return res

# 预处理阶乘的余数和阶乘逆元的余数
fact[0] = infact[0] = 1
for i in range(1, N):
    fact[i] = (fact[i - 1] * i) % mod
    infact[i] = (infact[i - 1] * qmi(i, mod - 2, mod)) % mod

# 计算组合数
def C(n, m):
    if m > n:
        return 0
    return (fact[n] * infact[m] % mod * infact[n - m] % mod)

# Example usage:
# result = C(5, 2)  # 获取组合数 C(5, 2)
```

```python
# print(result)
```

Lucas定理

```python
p = 1000000007  # 根据需要调整

# 快速幂模板
def qmi(a, k):
    res = 1
    while k:
        if k & 1:
            res = (res * a) % p
        a = (a * a) % p
        k >>= 1
    return res

# 通过定理求组合数C(a, b)
def C(a, b):
    res = 1
    for i in range(1, b + 1):
        res = (res * (a - i + 1)) % p
        res = (res * qmi(i, p - 2)) % p
    return res

# Lucas定理计算组合数
def lucas(a, b):
    if a < p and b < p:
        return C(a, b)
    return (C(a % p, b % p) * lucas(a // p, b // p)) % p

# Example usage:
# result = lucas(10, 5)  # 计算组合数 C(10, 5)
# print(result)
```

分解质因数法求组合数

```python
# 获取素数列表
def get_primes(n):
    primes = []
    st = [False] * (n + 1)
    for i in range(2, n + 1):
        if not st[i]:
            primes.append(i)
        for j in range(len(primes)):
            if primes[j] * i > n:
                break
            st[primes[j] * i] = True
            if i % primes[j] == 0:
                break
    return primes

# 获取n!中p的次数
def get(n, p):
    res = 0
    while n:
        res += n // p
        n //= p
    return res

# 高精度乘法
def mul(a, b):
```

```
    res = [0] * (len(a) + len(b))
    for i in range(len(a)):
        t = 0
        for j in range(len(b)):
            t += res[i + j] + a[i] * b[j]
            res[i + j] = t % 10
            t //= 10
        res[i + len(b)] += t
    while len(res) > 1 and res[-1] == 0:
        res.pop()
    return res

# 计算组合数
def calc_combination(a, b):
    primes = get_primes(a)
    sum = [0] * len(primes)
    for i in range(len(primes)):
        p = primes[i]
        sum[i] = get(a, p) - get(b, p) - get(a - b, p)
    res = [1]
    for i in range(len(primes)):
        for j in range(sum[i]):
            res = mul(res, [primes[i]])
    return res

# Example usage:
# result = calc_combination(10, 5)  # 计算组合数 C(10, 5)
# print(''.join(map(str, result[::-1])))
```

题目

日期差值

```
mm = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30,31, 30, 31]

def day(x):
    y = int(x/10000)
    m = int((x/100)%100)
    d = x%100
    mm[2]=29 if (y%4==0 and y%100!=0) or y%400==0 else 28
    for i in range(1, m):
        d+=mm[i]
    for i in range(1, y):
        d+=366 if (i%4==0 and i%100!=0) or i%400==0 else 365
    return d

while True:
    try:
        x=int(input())
        y=int(input())
        print( abs(day(x)-day(y))+1 )
    except:
        break

```

特殊排序

```python
# Forward declaration of compare API.
# def compare(a, b):
# @param a, b int
# @return bool
# return bool means whether a is less than b.

class Solution(object):
    def specialSort(self, N):
        """
        :type N: int
        :rtype: List[int]
        """
        a = [1]
        for i in range(2, N+1):
            l, r = 0, len(a)-1
            while l<r:
                mid = (l+r)>>1
                if compare(i, a[mid]):
                    r=mid
                else:
                    l=mid+1
            a[r+1:]=a[r:]
            a[r]=i
            if compare(a[r+1],a[r]):
                a[r], a[r+1] = a[r+1], a[r]
        return a
```

单链表

```python
N = int(1e5+10)
h, idx, e, ne = -1, 1, [0]*N, [0]*N

def insert(x):
    global h, idx
    e[idx]=x
    ne[idx]=h
    h=idx
    idx+=1

def add(k, x):
    global h,idx
    e[idx]=x
    ne[idx]=ne[k]
    ne[k]=idx
    idx+=1

def remove(k):
    global h
    if k==0:
        h = ne[h]
    else:
        ne[k]=ne[ne[k]]

n=int(input())
for _ in range(n):
    op = input().split()
    if op[0]=='H':
        insert(int(op[1]))
    elif op[0]=='D':
```

```
        remove(int(op[1]))
    elif op[0]=='I':
        add(int(op[1]), int(op[2]))
i=h
while i!=-1:
    print(e[i], end=" ")
    i=ne[i]
```

DP

数字三角形

```
f=[]
n=int(input())
for _ in range(n):
    f.append([int(x) for x in input().split()])

for i in range(n-2,-1,-1):
    for j in range(i+1):
        f[i][j]=max(f[i+1][j], f[i+1][j+1])+f[i][j]
print(f[0][0])
```

背包

空间优化成1维之后，只有完全背包问题的体积是从小到大循环的

01背包

```
N = int(1e3+10)
f=[ 0 for _ in range(N) ]
n,v=map(int,input().split())
for i in range(n):
    vi,wi=map(int,input().split())
    for j in range(v, vi-1,-1):
        f[j]=max(f[j],f[j-vi]+wi)
print(f[v])
```

多重背包

单调队列

```
MN = int(2e4+10)
f=[0 for _ in range(MN)]
q=[0 for _ in range(MN)]
g=[0 for _ in range(MN)]

N,V = map(int, input().split())

for i in range(N):
    v,w,s=map(int, input().split())
    g=f[:]
    for j in range(v):
        hh,tt=0,-1
        for k in range(j,V+1,v):
            while hh<=tt and q[hh]<k-s*v:
                hh+=1
            while hh<=tt and g[q[tt]]+(k-q[tt])//v*w <= g[k]:
                tt-=1
            tt+=1
```

```
            q[tt]=k
            f[k]=g[q[hh]]+(k-q[hh])//v*w

print(f[V])
```

二维费用背包

```
N = int(1e2+10)
f=[[0]*N for _ in range(N)]

n,V,M = map(int , input().split())

for i in range(n):
    v,m,w=map(int , input().split())
    for j in range(V,v-1,-1):
        for k in range(M, m-1, -1):
            f[j][k]=max(f[j][k], f[j-v][k-m]+w)

print(f[V][M])
```

宠物小精灵

```
N = int(1e3+10)
M = int(5e2+10)
f=[[0]*M for _ in range(N)]

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

for i in range(kk):
    v1,v2=map(int, input().split())
    for j in range(n, v1-1, -1):
        for k in range(m-1,v2-1,-1):
            f[j][k]=max(f[j][k], f[j-v1][k-v2]+1)

print(f[n][m-1], end=" ")
t=m-1
while t>0 and f[n][m-1]==f[n][t-1]:
    t-=1

print(m-t)
```

潜水

最多

恰好

最少

```
N = int(1e2+10)
INF = 0x3f3f3f3f
f=[ [INF]*N for _ in range(N) ]
f[0][0]=0

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

for i in range(k):
```

```
    a,b,c=map(int, input().split())
    for j in range(m,-1,-1):
        for k in range(n,-1,-1):
            f[j][k]=min(f[j][k], f[max(j-a, 0)][max(k-b, 0)]+c)

print(f[m][n])
```

庆功会

```
N = int(6e3+10)
f=[0 for _ in range(N)]
n,m = map(int, input().split())

for i in range(n):
    v,w,s=map(int, input().split())
    for j in range(1,s+1):
        for k in range(m, v-1, -1):
            f[k]=max(f[k], f[k-v]+w)


print(f[m])
```

分组背包

分组背包的顺序:

1. **物品组**
2. **体积**
3. **决策**

```
N = int(1e2+10)
f=[0 for _ in range(N)]
v=[0 for _ in range(N)]
w=[0 for _ in range(N)]

N,V = map(int, input().split())
for i in range(N): # 物品组
    s=int(input())
    for j in range(s):
        v[j],w[j]=map(int, input().split())
    for j in range(V,-1,-1): #体积
        for k in range(s): #决策
            if j>=v[k]:
                f[j]=max(f[j], f[j-v[k]]+w[k])

print(f[V])
```

机器分配

多重背包变种

```
N = 100
f=[0 for _ in range(N)]
w=[[0]*N for _ in range(N)]
c=[[0]*N for _ in range(N)]

n,m=map(int, input().split())
for i in range(n):
    w[i]=[0]+[int(x) for x in input().split()]
```

```
9
10 for i in range(n):
11     for j in range(m,-1,-1):
12         for k in range(1,j+1):
13             if f[j-k]+w[i][k] > f[j]:
14                 f[j]=f[j-k]+w[i][k]
15                 c[i][j]=k
16
17 print(f[m])
18
19 t=m
20 for i in range(n-1,-1,-1):
21     print(i+1, c[i][t])
22     t-=c[i][t]
23
```

金明的预算方案

```
1  N = 80
2  M = 32010
3  f=[0 for _ in range(M)]
4  mas=[[0,0] for _ in range(N)]
5  ser=[[] for _ in range(N)]
6
7  n,m=map(int,input().split())
8
9  for i in range(1,m+1):
10     v,p,q=map(int, input().split())
11     if q==0:
12         mas[i]=[v,v*p]
13     else:
14         ser[q].append((v, v*p))
15
16
17 for i in range(1,m+1):
18     for j in range(n, -1, -1):
19         for k in range( (1 << len(ser[i]) )):
20             v,p=mas[i]
21             for l in range(len(ser[i])):
22                 if k>>l & 1:
23                     v+=ser[i][l][0]
24                     p+=ser[i][l][1]
25
26             if j>=v:
27                 f[j]=max(f[j], f[j-v]+p)
28
29 print(f[n])
```

摘花生

```python
N = 105
f=[ [0]*N for _ in range(N)]
t=int(input())
for _ in range(t):
    r,c=map(int, input().split())
    for i in range(1,r+1):
        f[i] = [0] + [int(x) for x in input().split()]
        for j in range(1,c+1):
            f[i][j]+=max(f[i-1][j], f[i][j-1])
    print(f[r][c])
```

最小通行费

```python
N = int(1e3+10)
INF = 0x3f3f3f3f
f = [[INF]*N for _ in range(N)]
f[1][0]=f[0][1]=0
n=int(input())
for i in range(1,n+1):
    f[i]=[INF]+[int(x) for x in input().split()]
for i in range(1,n+1):
    for j in range(1,n+1):
        f[i][j]+=min(f[i-1][j],f[i][j-1])
print(f[n][n])
```

方格

```python
N = 15
g = [[0]*N for _ in range(N)]
f = [[[[0]*N for _ in range(N)] for _ in range(N)] for _ in range(N)]
n=int(input())
while True:
    r,c,x=map(int,input().split())
    if r==0 and c==0 and x==0:
        break
    g[r][c]=x
for i in range(1,n+1):
    for j in range(1,n+1):
        for k in range(1,n+1):
            for l in range(1,n+1):
                if i==k and j==l:
                    f[i][j][k][l] = max(f[i-1][j][k-1][l], f[i-1][j][k][l-1], f[i][j-1][k-1][l], f[i][j-1][k][l-1]) + g[i][j]
                else:
                    f[i][j][k][l] = max(f[i-1][j][k-1][l], f[i-1][j][k][l-1], f[i][j-1][k-1][l], f[i][j-1][k][l-1]) + g[i][j] + g[k][l]
print(f[n][n][n][n])
```

```python
N = 55
g=[[0]*N for _ in range(N)]
f=[[[[0]*N for _ in range(N)] for _ in range(N)] for _ in range(N)]

m,n=map(int, input().split())
for i in range(1,m+1):
    g[i]=[0]+[int(x) for x in input().split()]

for i in range(1,m+1):
    for j in range(1,n+1):
        for k in range(1,m+1):
            for l in range(1,n+1):
```

```
                if i==k and j==l:
                    f[i][j][k][l]=max(f[i-1][j][k-1][l], f[i-1][j][k][l-1], f[i][j-1][k-1][l], f[i][j-1][k][l-1])+g[i][j]
                else:
                    f[i][j][k][l]=max(f[i-1][j][k-1][l], f[i-1][j][k][l-1], f[i][j-1][k-1][l], f[i][j-1][k][l-1])+g[i][j]+g[k][l]
print(f[m][n][m][n])
```

LIS

```
N=int(1e3+10)
f=[1 for _ in range(N)]
a=[]
n=int(input())
a=[0]+[int(x) for x in input().split()]
for i in range(1,n+1):
    for j in range(1,i):
        if a[i]>a[j]:
            f[i]=max(f[i],f[j]+1)
ans = 0
for i in range(1,n+1):
    ans = max(ans, f[i])
print(ans)
```

LCS

```
N = int(1e3+10)
f=[[0]*N for _ in range(N)]
a=""
b=""
n,m=map(int, input().split())
a=input()
b=input()
for i in range(n):
    for j in range(m):
        f[i][j]=max(f[i-1][j], f[i][j-1])
        if a[i]==b[j]:
            f[i][j]=f[i-1][j-1]+1
print(f[n-1][m-1])
```

最大上升子序列和

```
import copy
a=[]
f=[]
ans=0

n=int(input())
a=[int(x) for x in input().split()]
f=copy.deepcopy(a)
for i in range(n):
    for j in range(i):
        if a[i]>a[j]:
            f[i]=max(f[i], f[j]+a[i])

for i in range(n):
    ans = max(ans, f[i])
print(ans)

N = int(1e3+10)
f=[0 for _ in range(N)]
```

```
ans = 0

n=int(input())
a=[0]+[int(x) for x in input().split()]

for i in range(1,n+1):
    f[i]=a[i]
    for j in range(1,i):
        if a[i]>a[j]:
            f[i]=max(f[i], f[j]+a[i])
    ans = max(ans, f[i])

print(ans)
```

最大上升子序列II

```
N = int(1e5+10)
INF = 0x3f3f3f3f
q=[INF for _ in range(N)]
ans=0

n=int(input())
a=[0]+[int(x) for x in input().split()]

for i in range(1,n+1):
    l,r=0,i
    while l<r:
        mid = (l+r+1)>>1
        if q[mid]<a[i]:
            l=mid
        else:
            r=mid-1
    ans = max(ans, l+1)
    q[l+1]=min(q[l+1], a[i])

print(ans)

```

```
N = int(1e5+10)
q=[0 for _ in range(N)]
len = 0

n=int(input())
a=[0]+[int(x) for x in input().split()]

for i in range(1, n+1):
    l,r=0,len
    while l<r:
        mid = (l+r+1)>>1
        if q[mid]<a[i]:
            l=mid
        else:
            r=mid-1
    len=max(len, l+1)
    q[l+1]=a[i]

print(len)
```

```
N = int(1e5+10)
q=[0 for _ in range(N)]
len = 0
```

```
n=int(input())
a=[0]+[int(x) for x in input().split()]
for i in range(1,n+1):
    l,r=0,len
    while l<r:
        mid = (l+r+1)>>1
        if q[mid]>=a[i]:
            r=mid-1
        else:
            l=mid
    len = max(len, l+1)
    q[l+1]=a[i]

print(len)
```

二分原则：

有单调性，并且二分之后能保持单调性

怪盗基德的滑翔翼

```
N = int(1e3+10)
f=[0 for _ in range(N)]

t=int(input())
while t:
    t-=1
    ans=0
    n=int(input())
    a=[0]+[int(x) for x in input().split()]
    for i in range(1,n+1):
        f[i]=1
        for j in range(1,i):
            if a[i]>a[j]:
                f[i]=max(f[i], f[j]+1)
        ans = max(ans, f[i])
    for i in range(n,0,-1):
        f[i]=1
        for j in range(n,i,-1):
            if a[i]>a[j]:
                f[i]=max(f[i], f[j]+1)
        ans = max(ans, f[i])
    print(ans)
```

登山

```
N = int(1e3+10)
f=[1 for _ in range(N)]
g=[1 for _ in range(N)]
ans = 0

n=int(input())
a=[0]+[int(x) for x in input().split()]

for i in range(1,n+1):
    for j in range(1,i):
        if a[i]>a[j]:
            f[i]=max(f[i], f[j]+1)

for i in range(n,0,-1):
    for j in range(n,i,-1):
```

```
        if a[i]>a[j]:
            g[i]=max(g[i], g[j]+1)

for i in range(1,n+1):
    ans = max(ans, f[i]+g[i]-1)

print(ans)
```

合唱队形

```
N = int(1e3+10)
f=[1 for _ in range(N)]
g=[1 for _ in range(N)]
ans = 0

n=int(input())
a=[0]+[int(x) for x in input().split()]

for i in range(1,n+1):
    for j in range(1,i):
        if a[i]>a[j]:
            f[i]=max(f[i], f[j]+1)

for i in range(n,0,-1):
    for j in range(n,i,-1):
        if a[i]>a[j]:
            g[i]=max(g[i], g[j]+1)

for i in range(1,n+1):
    ans = max(ans, f[i]+g[i]-1)

print(n-ans)
```

友好城市

```
N = int(5e3+10)
a=[(-1,-1)]
f=[1 for _ in range(N)]
ans = 0

n=int(input())
for i in range(n):
    x,y=map(int, input().split())
    a.append((x,y))

a.sort(key=lambda x:x[0])

for i in range(1,n+1):
    for j in range(1,i):
        if a[i][1]>a[j][1]:
            f[i]=max(f[i], f[j]+1)
    ans = max(f[i], ans)

print(ans)
```

拦截导弹

贪心证明

```
N = int(1e3+10)
f=[1 for _ in range(N)]
```

```
g=[0 for _ in range(N)]
ans = 0
cnt = 0

a=[0]+[int(x) for x in input().split()]
n=len(a)

for i in range(n-1,0,-1):
    for j in range(n-1,i,-1):
        if a[i]>=a[j]:
            f[i]=max(f[i], f[j]+1)
    ans = max(ans, f[i])

print(ans)

for i in range(1, n):
    k=0
    while k<cnt and g[k]<a[i]:
        k+=1
    if k>=cnt:
        cnt+=1
    g[k]=a[i]

print(cnt)
```

导弹防御系统

```
N = int(1e2+10)
up=[0 for _ in range(N)]
down=[0 for _ in range(N)]
ans=0

def dfs(u, su, sd):
    global ans
    if su + sd>=ans:
        return
    if u==n:
        ans=su+sd
        return

    k=0
    while k<su and up[k]>=a[u]:
        k+=1
    t=up[k]
    up[k]=a[u]
    if k>=su:
        dfs(u+1, su+1, sd)
    else:
        dfs(u+1, su, sd)
    up[k]=t

    k=0
    while k<sd and down[k]<=a[u]:
        k+=1
    t=down[k]
    down[k]=a[u]
    if k>=sd:
        dfs(u+1, su, sd+1)
    else:
        dfs(u+1, su, sd)
    down[k]=t

```

```
while True:
    n=int(input())
    ans = n
    if n==0:
        break
    a=[int(x) for x in input().split()]


    dfs(0,0,0)
    print(ans)
```

最长公共上升子序列

```
N = int(3e3+10)
f=[[0]*N for _ in range(N)]
ans=0

n=int(input())
a=[0]+[int(x) for x in input().split()]
b=[0]+[int(x) for x in input().split()]

for i in range(1,n+1):
    maxv=1
    for j in range(1,n+1):
        f[i][j]=f[i-1][j]
        if a[i]==b[j]:
            f[i][j]=max(f[i][j], maxv)
        if a[i]>b[j]:
            maxv=max(maxv, f[i][j]+1)
for i in range(1,n+1):
    ans = max(ans, f[n][i])

print(ans)
```

状态机模型

闫氏DP分析法——状态机分析法

```
N = int(1e5+10)


t=int(input())
while t:
    t-=1
    f=[[0]*2 for _ in range(N)]
    n=int(input())
    a=[0]+[int(x) for x in input().split()]
    f[1][0]=0
    f[1][1]=a[1]
    for i in range(2,n+1):
        f[i][0]=max(f[i-1][1],f[i-1][0])
        f[i][1]=max(f[i-1][0], f[i-2][1])+a[i]

    print(max(f[n][0], f[n][1]))
```

买卖股票 IV

```
K = 110
INF = 0x3f3f3f3f
f=[[[-INF]*2 for _ in range(K)] for _ in range(2)]

```

```
n,k = map(int ,input().split())
w=[0]+[int(x) for x in input().split()]

f[1][0][0]=0
t=0

for i in range(1,n+1):
    for j in range(k+1):
        f[t][j][0]=max(f[t^1][j][0], f[t^1][j][1]+w[i])
        f[t][j][1]=max(f[t^1][j][1], f[t^1][j-1][0]-w[i])
        t^=1

res =max(f[t^1][i][0] for i in range(k+1))
print(res)
```

股票交易V

```
N = int(1e5+10)
INF = 0x3f3f3f3f
f=[[-INF]*3 for _ in range(N)]

n=int(input())
w=[0]+[int(x) for x in input().split()]
f[0][2]=0

for i in range(1,n+1):
    f[i][0]=max(f[i-1][0], f[i-1][2]-w[i])
    f[i][1]=f[i-1][0]+w[i]
    f[i][2]=max(f[i-1][1], f[i-1][2])


print(max(f[n][1], f[n][2]))
```

状态压缩DP

小国王

```
N = 13
M = 1<<N
K = 110
state=[]
h=[[] for _ in range(M)]
cnt = [0]*M
f=[[[0]*M for _ in range(K)] for _ in range(N)]

def check(x):
    global n
    for i in range(n):
        if (x>>i)&1 and (x>>i+1)&1:
            return False
    return True

def count(x):
    global n
    cnt=0
    for i in range(n):
        if (x>>i)&1:
            cnt+=1
    return cnt

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

```
25
26  for i in range(1<<n):
27      if check(i):
28          state.append(i)
29          cnt[i]=count(i)
30
31  for i in range(len(state)):
32      for j in range(len(state)):
33          a = state[i]
34          b = state[j]
35          if (a&b)==0 and check(a|b):
36              h[i].append(j)
37
38  f[0][0][0]=1
39  for i in range(1,n+2):
40      for j in range(m+1):
41          for k in range(len(state)):
42              for t in h[k]:
43                  c = cnt[state[k]]
44                  if j>=c:
45                      f[i][j][state[k]]+=f[i-1][j-c][state[t]]
46
47
48  print(f[n+1][m][0])
49
```

愤怒的小鸟

```
1   eps = 1e-6
2   N = 18
3   M = 1<<18
4   INF = 0x3f3f3f3f
5
6   def cmp(a,b):
7       if abs(a-b)<eps:
8           return 0
9       if a>b:
10          return 1
11      return -1
12
13  t = int(input())
14
15  for _ in range(t):
16      f = [INF]*M
17      path = [[0]*N for _ in range(N)]
18      q = [0]*N
19
20      n,m = map(int, input().split())
21
22      for i in range(n):
23          x,y = map(float, input().split())
24          q[i]=(x,y)
25
26      for i in range(n):
27          path[i][i]=1<<i #关键点：与其他点的抛物线可能都不合法，所以需要独立出一条抛物线
28          for j in range(n):
29              x1,y1 = q[i]
30              x2,y2 = q[j]
31
32              if cmp(x1,x2)==0:
33                  continue
34
```

```
            a = (y1/x1-y2/x2)/(x1-x2)
            if cmp(a,0)>=0:
                continue
            b = y1/x1-a*x1

            state=0
            for k in range(n):
                x3,y3 = q[k]
                if cmp(a*x3*x3+b*x3, y3) == 0:
                    state+=(1<<k)
            path[i][j]=state

    f[0]=0
    for i in range(1<<n):
        for j in range(n):
            if not (i>>j)&1:
                x=j
                break
        for j in range(n):
            f[i | path[x][j]] = min(f[i | path[x][j]], f[i]+1)

    print(f[(1<<n)-1])
```

集合类状态压缩DP

最短Hamilton距离

```
N = 22
M = 1<<20
INF = 0x3f3f3f3f
f=[[INF]*N for _ in range(M)]
w=[[]*N for _ in range(N)]

n=int(input())
for i in range(n):
    w[i]=[int(_) for _ in input().split()]

f[1][0]=0
for i in range(1<<n):
    for j in range(n):
        if i>>j&1:
            for k in range(n):
                if i>>k&1:
                    f[i][j]=min(f[i][j], f[i-(1<<j)][k]+w[k][j])

print(f[(1<<n)-1][n-1])
```

区间DP

石子合并

```
N = 1010
INF = 0x3f3f3f3f
f=[[INF]*N for _ in range(N)]
s=[0]*N
n = int(input())
a=[0]+[int(_) for _ in input().split()]

for i in range(1,n+1):
    f[i][i]=0
    s[i]=s[i-1]+a[i]
```

```
11
12  for l in range(2,n+1):
13      for i in range(1,n-l+2):
14          j=i+l-1
15          for k in range(i,j):
16              f[i][j] = min(f[i][j], f[i][k] + f[k+1][j] + s[j]-s[i-1])
17
18  print(f[1][n])
```

环形石子合并

```
1   N = 410
2   w=[0]*N
3   s=[0]*N
4   INF = 0x3f3f3f3f
5   f=[[-INF]*N for _ in range(N)]
6   g=[[INF]*N for _ in range(N)]
7   n=int(input())
8
9   a= [0]+[int(_) for _ in input().split()]
10
11  for i in range(1,n+1):
12      w[i] = w[i+n] = a[i]
13
14  for i in range(1,n*2+1):
15      s[i]=s[i-1]+w[i]
16
17
18  for l in range(1,n+1):
19      for i in range(1,n*2-l+2):
20          j=i+l-1
21          if l==1:
22              f[i][j]=g[i][j]=0
23          for k in range(i,j):
24              f[i][j] = max(f[i][j], f[i][k]+f[k+1][j]+s[j]-s[i-1])
25              g[i][j] = min(g[i][j], g[i][k]+g[k+1][j]+s[j]-s[i-1])
26
27
28  minv, maxv = INF, -INF
29
30  for i in range(1,n+1):
31      minv = min(minv, g[i][i+n-1])
32      maxv = max(maxv, f[i][i+n-1])
33
34  print(minv, maxv, sep='\n')
```

能量项链

```
1   N = 210
2   w=[0]*N
3   f=[[0]*N for _ in range(N)]
4
5   n = int(input())
6   a=[0]+[int(_) for _ in input().split()]
7
8   for i in range(1,n+1):
9       w[i]=w[i+n]=a[i]
10
11
12  for l in range(3,n+2):
13      for i in range(1,n*2-l+2):
```

```
        j=i+l-1
        for k in range(i+1,j):
            f[i][j] = max(f[i][j], f[i][k]+f[k][j]+w[i]*w[k]*w[j])

maxv = -1
for i in range(1,n+1):
    maxv = max(maxv, f[i][i+n])

print(maxv)
```

凸多边形的划分

```
N = 55
INF = 1e30
f=[[INF]*N for _ in range(N)]

n = int(input())
w=[0]+[int(_) for _ in input().split()]

for i in range(1,n+1):
    f[i][i+1]=0

for l in range(3,n+1):
    for i in range(1, n-l+2):
        j=i+l-1
        for k in range(i+1,j):
            f[i][j] = min(f[i][j], f[i][k]+f[k][j]+w[i]*w[k]*w[j])

print(f[1][n])
```

加分二叉树

```
N = 50

def dfs(l, r):
    if l>r:
        return
    k=root[l][r]
    print(k,end=' ')
    dfs(l,k-1)
    dfs(k+1,r)


n = int(input())
w = [0]+[int(_) for _ in input().split()]
f=[[0]*N for _ in range(N)]
root=[[0]*N for _ in range(N)]

for l in range(1,n+1):
    for i in range(1,n-l+2):
        j=i+l-1
        for k in range(i,j+1):
            left = 1 if k==i else f[i][k-1]
            right = 1 if k==j else f[k+1][j]
            s = left*right+w[k]
            if i==j:
                s=w[k]
            if f[i][j]<s:
                f[i][j]=s
                root[i][j]=k

```

```
print(f[1][n])
dfs(1,n)
```

树形DP

树的最长路径

注意注意再注意，while循环链不符合要求是需要 i=ne[i]

```
import sys
sys.setrecursionlimit(int(1e5+10))
N = 10010*2
e = [0]*N
ne = [0]*N
w = [0]*N
h = [-1]*N
idx = 1
ans = -1

def add(a, b, c):
    global idx
    e[idx]=b
    w[idx]=c
    ne[idx]=h[a]
    h[a]=idx
    idx+=1

def dfs(root, father):
    global ans
    dis, d1, d2 = 0,0,0
    i=h[root]
    while i!=-1:
        j = e[i]
        if j==father:
            i=ne[i] # 重点
            continue
        d = dfs(j, root)+w[i]
        dis = max(dis, d)
        if d>d1:
            d2=d1
            d1=d
        else:
            d2 = max(d2, d)
        i=ne[i]
    ans = max(ans, d1+d2)
    return dis

n = int(input())

for i in range(n-1):
    a,b,c = map(int, input().split())
    add(a,b,c)
    add(b,a,c)

dfs(1,-1)
print(ans)
```

树的中心

```python
N = 10010*2
e = [0]*N
ne=[0]*N
h=[-1]*N
w=[0]*N
d1=[0]*N
d2=[0]*N
s1=[0]*N
up=[0]*N
idx = 1

def add(a,b,c):
    global idx
    e[idx]=b
    w[idx]=c
    ne[idx]=h[a]
    h[a]=idx
    idx+=1

def dfs1(u, father):
    i=h[u]
    while i!=-1:
        j=e[i]
        if j==father:
            i=ne[i]
            continue
        dfs1(j,u)
        dis = d1[j]+w[i]
        if dis>d1[u]:
            d2[u]=d1[u]
            d1[u], s1[u]=dis, j
        elif dis>d2[u]:
            d2[u]=dis
        i=ne[i]

def dfs2(u, father):
    i=h[u]
    while i!=-1:
        j=e[i]
        if j==father:
            i=ne[i]
            continue
        if s1[u]==j:
            up[j]=max(up[u], d2[u])+w[i]
        else:
            up[j]=max(up[u], d1[u])+w[i]
        i=ne[i]
        dfs2(j, u)

n = int(input())

for i in range(n-1):
    a,b,c = map(int ,input().split())
    add(a,b,c)
    add(b,a,c)

dfs1(1,-1)
dfs2(1,-1)
ans = 1e10
for i in range(1,n+1):
```

```
    ans = min(ans, max(up[i], d1[i]))
print(ans)
```

数字转换

```
N = int(5e4+10)
s=[0]*N
st=[False]*N
e=[0]*N
ne=[0]*N
h=[-1]*N
idx=1
ans = -1

def add(a,b):
    global idx
    e[idx]=b
    ne[idx]=h[a]
    h[a]=idx
    idx+=1

def dfs(u):
    global ans
    d1,d2=0,0
    i=h[u]
    while i!=-1:
        j=e[i]
        dis = dfs(j)+1
        if dis>d1:
            d2,d1=d1,dis
        elif dis>d2:
            d2=dis
        i=ne[i]
    ans = max(ans, d1+d2)
    return d1

n = int(input())

for i in range(1,n+1):
    for j in range(2,n+1):
        if i>n//j:
            break
        s[i*j]+=i

for i in range(2,n+1):
    if s[i]<i:
        add(s[i], i)
        st[i]=True

# for i in range(1,n+1):
#     if not st[i]:
#         dfs(i)

dfs(1)
print(ans)
```

没有上司的舞会

```python
import sys
sys.setrecursionlimit(int(1e4))

N = int(7e3)

v=[[] for _ in range(N)]
f=[[0]*2 for _ in range(N)]
st=[False]*N
h=[0]*N
root=0

def dfs(u):
    f[u][1]+=h[u]
    for i in v[u]:
        dfs(i)
        f[u][1]+=f[i][0]
        f[u][0]+=max(f[i][0], f[i][1])

n = int(input())

for i in range(1,n+1):
    h[i]=int(input())

for i in range(n-1):
    l,k = map(int, input().split())
    v[k].append(l)
    st[l]=True

for i in range(1,n+1):
    if not st[i]:
        root=i

dfs(root)

print(max(f[root][0], f[root][1]))
```

数位DP

1081

```python
from typing import List
N = 35
def init() -> List[List[int]]:
    f = [[0] * N for _ in range(N)]
    for i in range(N):
        for j in range(i + 1):
            if j == 0:
                f[i][j] = 1
            else:
                f[i][j] = f[i - 1][j] + f[i - 1][j - 1]
    return f

def dp(n: int, k: int, b: int, f: List[List[int]]) -> int:
    if n == 0:
        return 0
    nums = []
    while n:
        nums.append(n % b)
        n //= b
```

```
    res = 0
    last = 0
    for i in range(len(nums) - 1, -1, -1):
        x = nums[i]
        if x > 0:
            res += f[i][k - last]
            if x > 1:
                if k - last - 1 >= 0:
                    res += f[i][k - last - 1]
                break
            else:
                last += 1
                if last > k:
                    break

        if i == 0 and last == k:
            res += 1

    return res

def solve(l: int, r: int, k: int, b: int) -> int:
    f = init()
    return dp(r, k, b, f) - dp(l - 1, k, b, f)

if __name__ == "__main__":
    l, r, k, b = map(int, input().split())
    print(solve(l, r, k, b))
```

记忆化

```
import heapq

class Node:
    def __init__(self, i, j, num):
        self.i = i
        self.j = j
        self.num = num

def main():
    n, m = map(int, input().split())
    f = [[1] * (m + 2) for _ in range(n + 2)]  # distance
    g = [[0] * (m + 2) for _ in range(n + 2)]  # store heights

    pq = []
    for i in range(1, n + 1):
        nums = list(map(int, input().split()))
        for j, num in enumerate(nums, 1):
            g[i][j] = num
            heapq.heappush(pq, Node(i, j, num))

    ma = -1
    while pq:
        t = heapq.heappop(pq)
        i, j, nu = t.i, t.j, t.num
        if g[i - 1][j] < nu:
            f[i][j] = max(f[i][j], f[i - 1][j] + 1)
        if g[i + 1][j] < nu:
            f[i][j] = max(f[i][j], f[i + 1][j] + 1)
        if g[i][j - 1] < nu:
            f[i][j] = max(f[i][j], f[i][j - 1] + 1)
        if g[i][j + 1] < nu:
            f[i][j] = max(f[i][j], f[i][j + 1] + 1)
```

```
        ma = max(ma, f[i][j])

    print(ma)

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

数论

试除法求约数

```
def get(x):
    ans = []
    for i in range(1,x+1):
        if i>x//i:
            break
        if x%i==0:
            ans.append(i)
            if i!=x//i:
                ans.append(x//i)
    ans.sort()
    return ans

n = int(input())
for i in range(n):
    x = int(input())
    ans = get(x)
    for i in ans :
        print(i, end=' ')
    print()
```

树状数组

楼兰图腾

```
import sys
input=lambda:sys.stdin.readline()

M = 200010
suml, sumg = 0, 0
tr = [0]*M
n = int(input())
a = [0]+[int(_) for _ in input().split()]

def lowbit(x):
    return x&-x
def add(x,v):
    while x<M:
        tr[x]+=v
        x+=lowbit(x)

def query(x):
    res = 0
    while x:
        res+=tr[x]
        x-=lowbit(x)
    return res

for i in range(1,n+1):
    y=a[i]
    lw = query(y-1)
```

```
    lwr = y-1-lw # the lower element on the right side
    gr = query(n)-query(y)
    grr = (n-y) - gr
    suml += (lw*lwr)
    sumg += (gr*grr)
    add(y,1)

print(sumg, suml)
```

线段树

```
# 定义树节点，l,r, val表示该节点记录的是区间[l, r]的最大值是val
class Tree():
    def __init__(self):
        self.l = 0
        self.r = 0
        self.lazy = 0
        self.val = 0

# 二叉树是堆形式，可以用一维数组存储，注意数组长度要开4倍空间
tree = [Tree() for i in range(10*4)]

# 建树，用cur<<1访问左子树，cur<<1|1访问右子树，位运算操作很方便
def build(cur, l, r):
    tree[cur].l, tree[cur].r, tree[cur].lazy, tree[cur].val = l, r, 0, 0
    # 当l==r的时候结束递归
    if l < r:
        mid = l + r >> 1
        build(cur<<1, l, mid)
        build(cur<<1|1, mid+1, r)

# 当子节点计算完成后，用子节点的值来更新自己的值
def pushup(cur):
    tree[cur].val = max(tree[cur<<1].val, tree[cur<<1|1].val)

# 单点更新
def add(cur, x, v):
    if tree[cur].l == tree[cur].r:
        tree[cur].val += v
    else:
        mid = tree[cur].r + tree[cur].l >> 1
        if x > mid:
            add(cur>>1|1, x, v)
        else:
            add(cur<<1, x, v)
        pushup(cur)

# 将lazy标记向下传递一层
def pushdown(cur):
    if tree[cur].lazy:
        lazy = tree[cur].lazy
        tree[cur<<1].lazy += lazy
        tree[cur<<1|1].lazy += lazy
        tree[cur<<1].val += lazy
        tree[cur<<1|1].val += lazy
        tree[cur].lazy = 0

# 区间更新
def update(cur, l, r, v):
    if l <= tree[cur].l and tree[cur].r <= r:
        tree[cur].lazy += v
        tree[cur].val += v
```

```
        return
    if r < tree[cur].l or l > tree[cur].r:
        return
    if tree[cur].lazy:
        pushdown(cur)
    update(cur<<1, l, r, v)
    update(cur<<1|1, l, r, v)
    pushup(cur)

# 区间查询
def query(cur, l, r):
    if l <= tree[cur].l and tree[cur].r <= r:
        return tree[cur].val
    if tree[cur].l > r or tree[cur].r < l:
        return 0
    if tree[cur].lazy:
        pushdown(cur)
    return max(query(cur<<1, l, r), query(cur<<1|1))

# 测试
# -----
#        ---
#  -------
#   --
#         --

build(1, 1, 10)
update(1, 1, 5, 1)
update(1, 7, 10, 1)
update(1, 2, 8, 1)
update(1, 3, 4, 1)
update(1, 9, 10, 1)
print(query(1, 1, 10))
```

```
def pushup(u):
    tr[u] = tr[u << 1] + tr[u << 1 | 1]

def build(u, l, r):
    if l == r:
        tr[u] = 0
    else:
        mid = (l + r) >> 1
        build(u << 1, l, mid)
        build(u << 1 | 1, mid + 1, r)
        pushup(u)

def query(u, l, r, ql, qr):
    if l >= ql and r <= qr:
        return tr[u]
    mid = (l + r) >> 1

    if mid==l and mid==r:
        return 0

    res = 0
    if ql <= mid:
        res = query(u << 1, l, mid, ql, qr)
    if qr > mid:
        res += query(u << 1 | 1, mid + 1, r, ql, qr)
    return res

def modify(u, x, l, r, val):
```

```
    if l == r:
        tr[u] += val
    else:
        mid = (l + r) >> 1
        if x <= mid:
            modify(u << 1, x, l, mid, val)
        else:
            modify(u << 1 | 1, x, mid + 1, r, val)
        pushup(u)
```

搜索深入

池塘计数

```
import sys
from collections import deque
input = lambda:sys.stdin.readline().strip()

N = int(1e3+10)
M = N*N
g = [0]*N
cnt=0
vis = [ [False]*N for _ in range(N) ]

def bfs(x, y):
    q=deque()
    q.append( (x,y) )
    while q:
        tx, ty = q[0]
        q.popleft()
        for i in range(-1, 2):
            for j in range(-1, 2):
                if i==0 and j==0:
                    continue
                xx, yy = tx+i, ty+j
                if xx<0 or xx>=n or yy<0 or yy>=m or vis[xx][yy] or g[xx][yy]=='.':
                    continue
                vis[xx][yy]=True
                q.append( (xx,yy) )

n, m = map(int, input().split())
for i in range(n):
    g[i]=input()

for i in range(n):
    for j in range(m):
        if g[i][j]!='W' or vis[i][j]:
            continue
        bfs(i, j)
        cnt+=1
print(cnt)
```

城堡问题

```
import sys
from collections import deque
input = lambda:sys.stdin.readline().strip()
N = 55
g = []
vis = [[False]*N for _ in range(N)]
area = 0
```

```
cnt=0

def bfs(x, y):
    vis[x][y]=True
    q = deque()
    q.append((x, y))
    dx, dy = [0, -1, 0, 1], [-1, 0, 1, 0]
    ans=1
    while q:
        tx, ty = q.popleft()
        for i in range(4):
            xx = tx+dx[i]
            yy = ty+dy[i]
            if xx<0 or xx>=n or yy<0 or yy>=m or vis[xx][yy]:
                continue
            if (g[tx][ty]>>i)&1:
                continue
            ans+=1
            vis[xx][yy]=True
            q.append((xx,yy))
    return ans

n, m = map(int, input().split())
for _ in range(n):
    g.append(list(map(int, input().split())))

for i in range(n):
    for j in range(m):
        if vis[i][j]:
            continue
        area = max(area, bfs(i, j))
        cnt+=1
print(cnt)
print(area)
```

山峰和山谷

```
import sys
from collections import deque
N = 1010
g = []
vis = [[0]*N for _ in range(N)]
pek, val = 0, 0

def bfs(x, y):
    global higher, lower
    vis[x][y]=True
    q = deque()
    q.append((x,y))
    while q:
        tx,ty = q.popleft()
        for i in range(-1, 2):
            for j in range(-1, 2):
                if i==0 and j==0:
                    continue
                xx, yy = tx+i, ty+j
                if xx<0 or xx>=n or yy<0 or yy>=n:
                    continue
                if g[xx][yy]!=g[tx][ty]:
                    if g[xx][yy]>g[tx][ty]:
                        higher=True
                    elif g[xx][yy]<g[tx][ty]:
```

```
                        lower=True
                elif not vis[xx][yy]:
                    vis[xx][yy]=True
                    q.append((xx,yy))

higher, lower = False, False
n = int(input())
for _ in range(n):
    g.append(list(map(int, input().split())))

# print(g)
for i in range(n):
    for j in range(n):
        if vis[i][j]:
            continue
        higher, lower = False, False
        bfs(i, j)
        if not higher:pek+=1
        if not lower:val+=1
print(pek, val)
```

Python特点

IDLE 使用

输入输出

输出列表：

```
print（*a） # 输出列表中的所有数，用空格分隔
print（*a，sep="\n"） #每个数单独放一行
```

*运算符

1. 解包运算符：
   当`*`运算符用于可迭代对象（如列表、元组、集合等）前面时，它可以将可迭代对象解包为多个元素。例如：

   ```
   a = [1, 2, 3]
   print(*a)  # 解包并打印出每个元素：1 2 3
   ```

2. 可变参数：
   当`*`运算符用于函数定义时，它表示接受任意数量的参数，并将它们作为元组传递给函数。这种用法通常称为可变参数。例如：

   ```
   def my_func(*args):
       for arg in args:
           print(arg)

   my_func(1, 2, 3)  # 打印出每个参数：1 2 3
   ```

3. 扩展运算符：
   当`*`运算符用于可迭代对象前面时，它可以将可迭代对象的元素扩展到另一个可迭代对象中。这种用法通常称为扩展运算符。例如：

```python
a = [1, 2, 3]
b = [4, 5, 6
c = [*a, *b]  # 扩展a和b的元素到c中
print(c)  # 输出：[1, 2, 3, 4, 5, 6]
```

4. 乘法运算符：
   当 `*` 运算符用于数字和可迭代对象之间时，它表示重复该可迭代对象的元素。例如：

```python
a = [1, 2, 3]
b = a * 3  # 重复a的元素3次
print(b)  # 输出：[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

*和**

1. `*` 和 `**` 在函数定义中的使用：
   - `*args` 用于接收任意数量的位置参数，并将它们作为元组传递给函数。
   - `**kwargs` 用于接收任意数量的关键字参数，并将它们作为字典传递给函数。
2. `*` 和 `**` 在函数调用中的使用：
   - 在函数调用时，`*` 用于解包可迭代对象，并将其作为位置参数传递给函数。
   - 在函数调用时，`**` 用于解包字典，并将其作为关键字参数传递给函数。

栈模拟递归

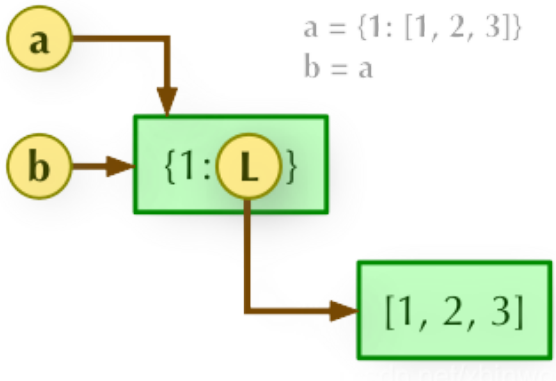
```python
from collections import deque
def dfs(idx,p):
    q = deque()
    q.append((idx,p))
    while q:
        idx,p = q.pop()
        D[idx] = D[p] + V[idx]
        for u in A[idx]:
            if u == p: continue
            q.append((u,idx))

```

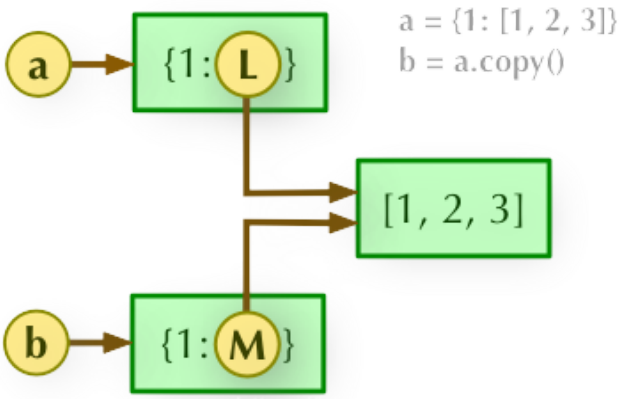
引用赋值、浅拷贝和深拷贝

Python赋值、浅拷贝、深拷贝的区别
[:]和.copy()都属于“浅拷贝”，只拷贝最外层元素，外层元素是独立内存；内层嵌套元素则通过引用方式共享，而非独立分配内存。使用 copy 模块的 copy.copy （ 浅拷贝 ）和 copy.deepcopy（深拷贝），其中deepcopy是构建了一个完全独立的对象。

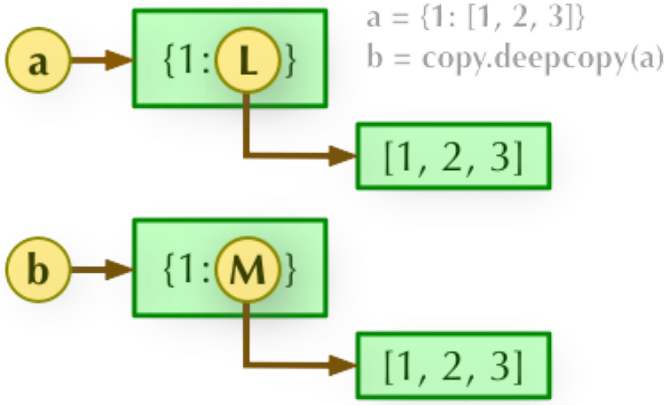
1、b = a: 赋值引用，a 和 b 都指向同一个对象。



2、b = a.copy(): 浅拷贝, a 和 b 是一个独立的对象，但他们的子对象（内层嵌套对象）还是指向统一对象（是引用）。



3、b = copy.deepcopy(a): 深度拷贝, a 和 b 完全拷贝了父对象及其子对象，两者是完全独立的。



例子:

引用赋值

```
>>>a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>b = a[:]
>>>print(b)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>print(id(a)) #41946376
>>>print(id(b)) #41921864
或
>>>b = a.copy()
>>>print(b)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>print(id(a)) #39783752
>>>print(id(b)) #39759176
```

浅拷贝

```
>>>a = [1,2,['A','B']]
>>>print('a={}'.format(a))
>>>b = a[:]
>>>b[0] = 9 #修改b的最外层元素，将1变成9
>>>b[2][0] = 'D' #修改b的内嵌层元素
>>>print('a={}'.format(a))
>>>print('b={}'.format(b))
>>>print('id(a)={}'.format(id(a)))
>>>print('id(b)={}'.format(id(b)))
a=[1, 2, ['A', 'B']] #原始a
a=[1, 2, ['D', 'B']] #b修改内部元素A为D后，a中的A也变成了D，说明共享内部嵌套元素，但外部元素1没变。
b=[9, 2, ['D', 'B']] #修改后的b
```

```
id(a)=38669128
id(b)=38669192

```

深拷贝

```python
import copy
a = [1, 2, 3, 4, ['a', 'b']] #原始对象

b = a                        #赋值，传对象的引用
c = copy.copy(a)             #对象拷贝，浅拷贝
d = copy.deepcopy(a)         #对象拷贝，深拷贝

a.append(5)                  #修改对象a
a[4].append('c')             #修改对象a中的['a', 'b']数组对象

print( 'a = ', a )
print( 'b = ', b )
print( 'c = ', c )
print( 'd = ', d )

#输出：
'a = ', [1, 2, 3, 4, ['a', 'b', 'c'], 5]
'b = ', [1, 2, 3, 4, ['a', 'b', 'c'], 5]
'c = ', [1, 2, 3, 4, ['a', 'b', 'c']]
'd = ', [1, 2, 3, 4, ['a', 'b']]
```

栈代替递归

增加递归深度

```python
import sys
sys.setrecursionlimit(150000000)
print(sys.getrecursionlimit())
```

迭代加深搜索

```python
s = input()
l = len(s)
s = "0" + s  # 1~l
ans = set()
st = {}

def dfs(curlen, last):
    global ans, st
    stack = [(curlen, last)]
    while stack:
        curlen, last = stack.pop()
        if curlen - 2 > 4 and last != s[curlen - 1:curlen + 1]:
            if (curlen - 1, curlen) not in st:
                st[(curlen - 1, curlen)] = 1
                ans.add(s[curlen - 1:curlen + 1])
                stack.append((curlen - 2, s[curlen - 1:curlen + 1]))

        if curlen - 3 > 4 and last != s[curlen - 2:curlen + 1]:
            if (curlen - 2, curlen) not in st:
                st[(curlen - 2, curlen)] = 1
                ans.add(s[curlen - 2:curlen + 1])
                stack.append((curlen - 3, s[curlen - 2:curlen + 1]))

```

```
dfs(1, "")
ans = sorted(ans)  # 将集合转换为列表并排序
print(len(ans))
for si in ans:
    print(si)
```

加速读入

```
import sys
print('Plase input your name: ')
name = sys.stdin.readline()
print(name)
```

队列

Queue中有FIFO（先入先出）队列Queue，LIFO（后入先出）栈LifoQueue，和优先级队列PriorityQueue，但速度较慢，且不能不出栈地访问头部元素，想要访问头部元素，只能用get方法出栈首部获取方法返回值的来进行访问，非常不方便。

可以用deque()模拟

```
import collections

q=collections.deque()

m=int(input())
for i in range(m):
    s = input().split()
    if s[0]=='push':
        q.append(s[1])
    elif s[0]=='pop':
        q.popleft()
    elif s[0]=='empty':
        if len(q)==0:
            print('YES')
        else:
            print('NO')
    else:
        print(q[0])
```

栈

列表模拟

```
m=int(input())

stk=[]
for i in range(m):
    s = input().split()
    if s[0]=='push':
        stk.append(int(s[1]))
    elif s[0]=='pop':
        stk.pop()
    elif s[0]=='empty':
        if len(stk)==0:
            print('YES')
        else:
            print('NO')
    else:
        print(stk[-1])
```

deque模拟

```python
import collections

stk = collections.deque()

m=int(input())
for i in range(m):
    s = input().split()
    if s[0]=='push':
        stk.appendleft( int(s[1]) )
    elif s[0]=='pop':
        stk.popleft()
    elif s[0]=='empty':
        if len(stk)==0:
            print('YES')
        else:
            print('NO')
    else:
        print(stk[0])
```

Python 常用内置库

`array`	**定长数组**
`argparse`	命令行参数处理
`bisect`	二分查找
`collections`	有序字典、双端队列等数据结构
`fractions`	有理数
`heapq`	基于堆的优先级队列
`io`	文件流、内存流
`itertools`	迭代器
`math`	数学函数
`os.path`	系统路径等
`random`	随机数
`re`	正则表达式
`struct`	转换结构体和二进制数据
`sys`	系统信息

defaultdict()

```python
from collections import defaultdict

# 创建一个 defaultdict，指定默认值为 int 类型的 0
d = defaultdict()

# 修改默认值为 100
d.default_factory = lambda: (1,2)
```

Counter()

```python
from collections import Counter

# 定义一个列表
lst = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
s = 'abcdgsaa'

# 使用 Counter 统计列表中元素的出现次数
c1 = Counter(lst)
c2 = Counter(s)

print(c1,c2, sep='\n')

# 使用 most_common() 方法按照元素的出现次数进行排序
sorted_items = c1.most_common()

for x,y in enumerate(sorted_items):
    print(y[0])
```

heapq

建堆 (小根堆)

```python
a = [1, 5, 20, 18, 10, 200]
heapq.heapify(a)
print(a)
```

建大根堆

```python
a = []
for i in [1, 5, 20, 18, 10, 200]:
    heapq.heappush(a,-i)
print( list( map(lambda x:-x,a) ) )
```

heap_sort(heappush)

```python
import heapq
def heap_sort(arr):
    if not arr:
        return []
    h = []  #建立空堆
    for i in arr:
        heapq.heappush(h,i) #heappush自动建立小根堆
    return [heapq.heappop(h) for i in range(len(h))]  #heappop每次删除并返回列表中最小的值
```

```python
# 堆排序取最小的m个数字
import heapq
def heap_sort(arr, k):
    if not arr:
        return []
    h=[]
    for i in arr:
        heapq.heappush(h, i)
    return [heapq.heappop(h) for _ in range(k)]

n,m = map(int, input().split())
arr = list(map(int, input().split()))
ans = heap_sort(arr, m)
```

```
print(' '.join(map(str, ans)))
```

heappushpop

先push再pop

```
[1, 18, 5, 20, 90, 10, 200]
h
[1, 18, 5, 20, 90, 10, 200]
heapq.heappushpop(h, 300)
1
h
[5, 18, 10, 20, 90, 300, 200]
```

`heapreplace`

先pop再push

```
h
[5, 18, 10, 20, 90, 300, 200]
heapq.heapreplace(h, -1)
5
h
[-1, 18, 10, 20, 90, 300, 200]
```

heapq.merge

```
import heapq
h1 = [90, 1, 5, 20, 18, 10, 200]
h2 = [4,2,3,4,1000]
heapq.heapify(h1)
heapq.heapify(h2)
print(list(heapq.merge(h1, h2)))
```

heap.nlargest

```
h1
[1, 18, 5, 20, 90, 10, 200]
heapq.nlargest(2,h1,key=lambda x:-x)
[1, 5]
```

List()

```
del list[1] 删除列表元素

列表比较
    import operator
    operator.eq(a,b)
len(list)
max(list)
min(list)
list(seq) 将元组转换为列表

list.append(obj)
list.count(obj)
list.extend(seq)
list.index(obj)
list.insert(index, obj)
list.pop([index=-1]) 删除列表中一个元素
list.remove(obj) 删除第一个匹配项
```

```
list.reverse()
list.sort(key=None, reverse=False)
list.clear()
list.copy()
```

tuple() 元素组合

类似list

SortedList()

```
from sortedcontainers import SortedList
sl = SortedList()
sl.add(1)
print(sl[-1])
print(sl[0])
sl.update([3,2,1])
print(sl)
sl.update([9,8,7])
print(sl)
##sl.clear()
sl.discard(5)
sl.remove(9)
print(sl)
sl.pop()
print(sl)
sl.pop(-2)
print(sl)
print(sl.bisect_left(12)) #返回需要插入的位置，如有存在则返回左侧的位置
print(sl.bisect_right(2))
print(sl.count(1))
print(sl.index(1))
it = sl.islice(2,4)
print(list(it))
```

dict()

键值必须不可变

```
d = {'1':'a', '2':'b', '99':'xycz'}
print(d)
if '0' in d :del d['0']
del d['1']
print(d)

{'1': 'a', '2': 'b', '99': 'xycz'}
{'2': 'b', '99': 'xycz'}
```

内置方法

```
len str type
dict.clear()
dict.copy()
dict.fromkeys(seq) 将seq作为字典的键值， 字典中val为默认
dict.get(key, default=None)
key in dict
dict.items()
dict.keys()
dict.setdefault(key, default = None)
dict.update(dict2) 把dict2添加到dict中
dict.values() 返回值
pop(key[,default]) 删除字典中key所对应的值并返回
popitem() 返回并删除字典中最后一对键值
```

set()

```
空集合用set()
支持 -, |, &, ^(不同时包含于两个集合)
    difference() 在原集合上修改，无返回值，difference_update() 返回新集合
    union() 并集
    intersection() intersection_update() 返回交集
    isdisjoint() 判断两个集合是否包含相同的元素
    issubset() 判断指定参数的集合是否为该调用方法的集合的子集
    issuperset() 判断该方法是否为指定参数的子集
    symmetric_difference() 返回两个集合中不重复的元素集合
    symmetric_difference_update() 移除相同的元素，并插入没有的元素


s.add(x) 添加元素
s.update(x) 可以添加多个元素，并且可以是列表元组字典
s.remove(x) 将元素从集合中移除， 如果不存在则报错
s.discard(x) 移除元素，但是不报错
s.pop() 设置随机删除结合中的一个元素（无序集合的第一个元素）
len(s) s.clear()
x in s
s.copy()

```

自定义比较参数

```
def test4(things):
    def compare(s1, s2):
        if len(s1) == len(s2):
            for c1, c2 in zip(s1, s2):
                if c1 > c2:
                    return 1
                elif c1 < c2:
                    return -1
            return 0  # 这里的比较其实可以直接使用字符串之间的 >, <, == 实现，之所以手动实现，是为了强调，没有内置比较函数的类也可以手动实现自定义的比较
        else:
            if len(s1) > len(s2):
                return 1
            else:
                return -1
    from functools import cmp_to_key
    things.sort(key=cmp_to_key(compare))
    print(things)

```

```
18
19 test4(words.copy())
20
```