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

```python
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)
    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
```

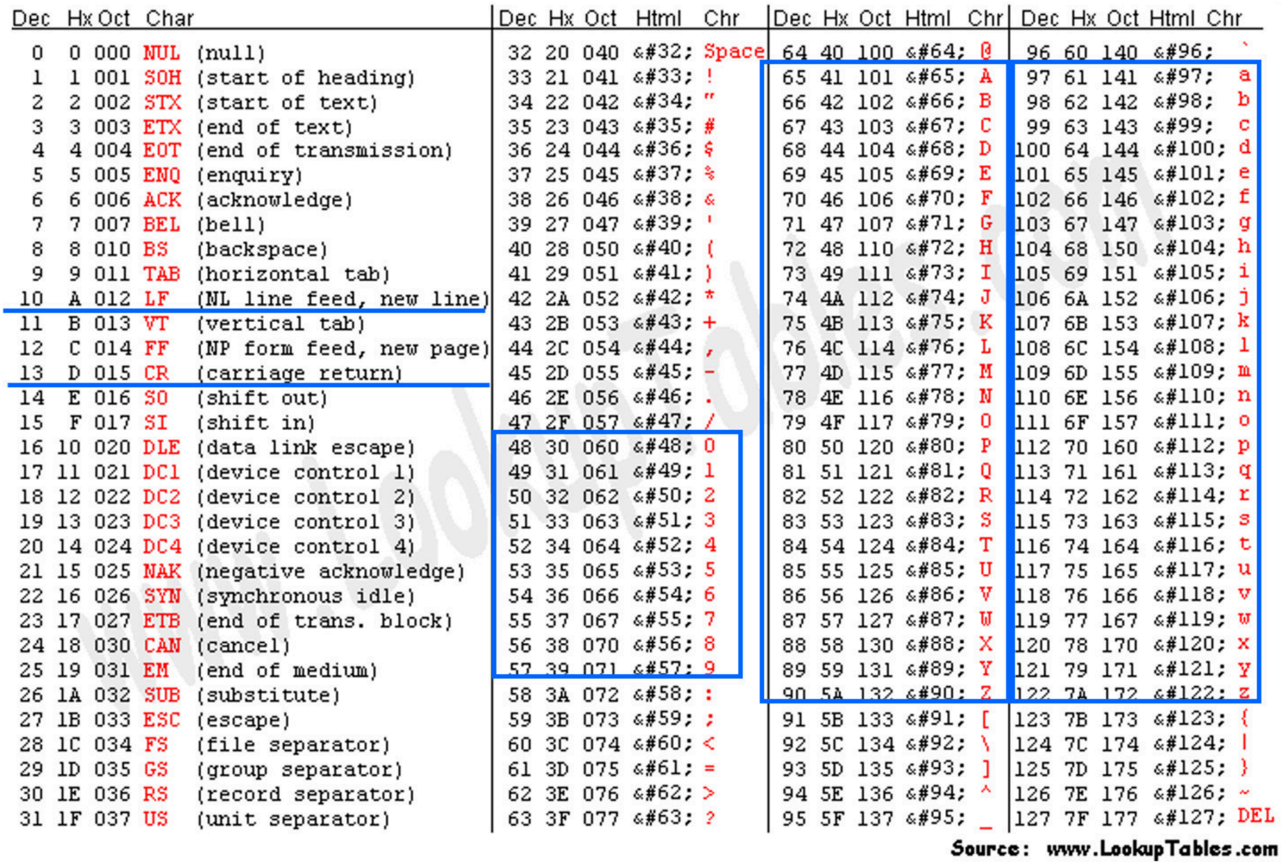
`bisect_left`的差别在于，循环部分的代码为

```python
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)`）
5. `filter(func, iter)`
6. ASCII `ord()` `chr()`



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

7. `sys.setrecursionlimit(1 << 30)`

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）

```python
n = int(input())
```

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

```
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。
- 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。

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

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]] + 1)
```

- 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()))

```

其他例题

转移方程基于最优子结构

数字金字塔

```
N = int(input())
triangle = [[0] + list(map(int, input().split())) + [0] for i in range(N)]
for i in range(1, N):
    for j in range(1, i + 2):
        triangle[i][j] += max(triangle[i - 1][j - 1], triangle[i - 1][j])
print(max(triangle[N - 1]))
```

最长公共子序列

```
for i in A:
    for j in B:
        length[i][j] = length[i - 1][j - 1] + 1 if A[i] == B[j] else max(length[i - 1][j], length[i][j - 1])
print(length[n][m])
```

最长不降子序列

```
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

```
val = list(map(int, input().split(",")))
n = len(val)
dp = [[0 for i in range(n + 1)] for j in range(2)]
dp[0][1] = val[0]
dp[0][2] = max(val[0] + val[1], val[1])
dp[1][2] = val[1]
for i in range(2, 1 + n):
    dp[0][i] = max(dp[0][i - 1] + val[i - 1], val[i - 1])
    dp[1][i] = max(dp[1][i - 1] + val[i - 1], dp[0][i - 2] + val[i - 1])
print(max(map(max, dp)))

```

Boredom

- 给定一个有 n 个元素的序列 $\{a_n\}$。你可以做若干次操作。在一次操作中我们可以取出一个数（假设他为 x）并删除它，同时删除所有的序列中值为 $x+1$ 和 $x-1$ 的数，这一步操作会给玩家加上 x 分，玩家最多能获得多少分

```
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)]
```

```python
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

```python
def dfs(x, y):
    # 标记当前位置为已访问
    field[x][y] = '.'
    '''原地修改，不用回溯'''
    # 遍历8个方向
    for dx, dy in directions:
        nx, ny = x + dx, y + dy
        # 检查新位置是否在地图范围内且未被访问
        if 0 <= nx < n and 0 <= ny < m and field[nx][ny] == 'W':
            dfs(nx, ny)


n, m = map(int, input().split())
field = [list(input()) for _ in range(n)]
directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]
cnt = 0

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

```

广度优先搜索

```python
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。

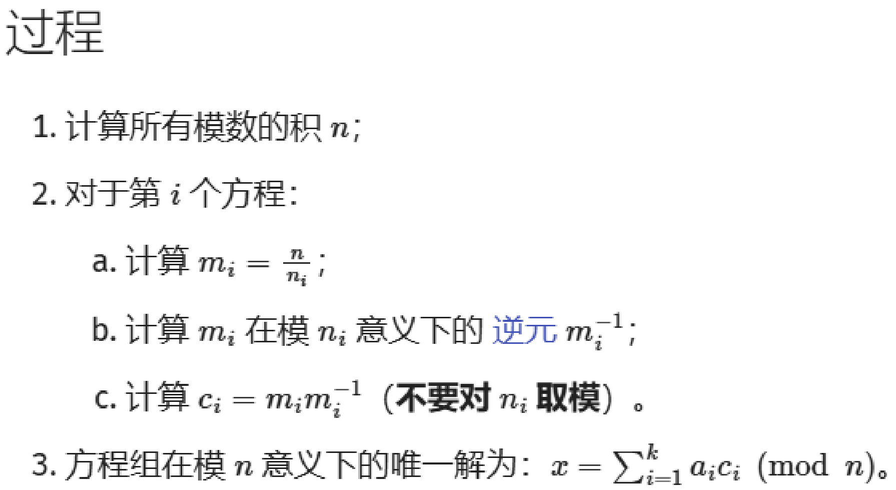
```python
from heapq import heappop, heappush
DIRECTIONS = ((1, 0), (-1, 0), (0, 1), (0, -1))

def bfs(x1, y1, x2, y2):
    min_heap = []
    min_seg = 1e9
    min_heap.append((0, x1, y1, {(x1, y1)}, (0, 0)))
    while min_heap:
        seg, x, y, visited, last_dir = heappop(min_heap)
        for dx, dy in DIRECTIONS:
            nx, ny = x + dx, y + dy
            if (nx, ny) not in visited:
                if (nx, ny) == (x2, y2):
                    min_seg = min(min_seg, seg + (1 if (dx, dy) != last_dir else 0))
                    break
                if board[ny][nx] == " ":
                    if seg + (1 if (dx, dy) != last_dir else 0) < min_seg:
                        heappush(min_heap, (seg + (1 if (dx, dy) != last_dir else 0), nx, ny, visited | {(nx, ny)}, (dx, dy)))
    return min_seg

for _ in range(1, int(1e9)):
    w, h = map(int, input().split())
    if w == 0:
        break
    print(f"Board #{_}:")
    board = [["X" for i in range(w + 4)]]\
     + [["X"] + [" " for i in range(w + 2)] + ["X"]]\
     + [["X", " "] + list(input()) + [" ", "X"] for i in range(h)]\
     + [["X"] + [" " for i in range(w + 2)] + ["X"]]\
     + [["X" for i in range(w + 4)]]
    for cnt in range(1, int(1e9)):
        x1, y1, x2, y2 = map(lambda t: int(t) + 1, input().split())
        if x1 == 1:
            break
        min_seg = bfs(x1, y1, x2, y2)
        if min_seg == 1e9:
            print(f"Pair {cnt}: impossible.")
        else:
            print(f"Pair {cnt}: {min_seg} segments.")
    print()
```

迪杰斯特拉算法

```python
def dijkstra(e, s):
    """
    输入:
    e:邻接表
    s:起点
    返回:
    dis:从s到每个顶点的最短路长度
    """
    dis = defaultdict(lambda: float("inf"))
    dis[s] = 0
    q = [(0, s)]
    vis = set()
    while q:
        _, u = heapq.heappop(q)
        if u in vis:
            continue
        vis.add(u)
        for v, w in e[u]:
            if dis[v] >= dis[u] + w:
                dis[v] = dis[u] + w
                heapq.heappush(q, (dis[v], v))
    return dis
```

数学问题

中国剩余定理



过程

1. 计算所有模数的积 n；
2. 对于第 i 个方程：

   a. 计算 $m_i = \frac{n}{n_i}$；

   b. 计算 m_i 在模 n_i 意义下的 逆元 m_i^{-1}；

   c. 计算 $c_i = m_i m_i^{-1}$（**不要对** n_i **取模**）。
3. 方程组在模 n 意义下的唯一解为：$x = \sum_{i=1}^{k} a_i c_i \pmod n$。

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

```
primes = []
primesstatus = [True for i in range(int(1e6 + 1))]
primesstatus[0], primesstatus[1] = False, False
for i in range(2, int(1e6 + 1)):
    if primesstatus[i]:
        primes.append(i)
```

```python
    for j in primes:
        if i * j > 1e6:
            break
        primesstatus[i * j] = False
        if i % j == 0 and j != i: # 确保每个数被其最小的约数约掉
            break
```

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

```python
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

河中跳房子

```python
def check(dist):
    t, num = 0, 0
    for i in range(1, N + 2):
        if stone[i] - t < dist:
            num += 1
        else:
            t = stone[i]
    return num > M
L, N, M = map(int, input().split())
stone = [0] + [int(input()) for i in range(N)] + [L]
lo, hi = 0, L
while lo < hi:
    mid = (lo + hi) // 2
    if check(mid):
        hi = mid
    else:
        lo = mid + 1
```

```
print(lo - 1)

```

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

```
inv += len(left) - i
```

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

```python
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。

```python
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))))
```

单调栈

接雨水

```python
class Solution:
    def trap(self, height: List[int]) -> int:
        stack = []
        water = 0
        for i in range(len(height)):
            while stack and height[i] > height[stack[-1]]:
                top = stack.pop()
                if not stack:
                    break
                distance = i - stack[-1] - 1
                bounded_height = min(height[i], height[stack[-1]])-height[top]
                water += distance * bounded_height
            stack.append(i)
        return water
```