

算法竞赛笔记

Chunyin Chan

2025 年 4 月 3 日

目录

1	基础数据结构	1
1.1	Link List in Python:	1
1.2	Queue in Python	4
1.2.1	双端队列	4
1.2.2	单调队列	5
1.2.3	单调队列与动态规划	6
1.2.4	优先队列	8
1.3	Stack in Python	8
1.3.1	单调栈	9
1.4	Binary Tree in Python	10
1.4.1	DFS 遍历	10
1.4.2	哈夫曼编码	12
1.5	Heap in Python	13
2	基本算法	15
2.1	尺取法 (双指针)	15
2.1.1	反向扫描	15
2.1.2	同向扫描	16
2.2	二分法	19
2.2.1	Python 二分搜索库 bisect	19
2.2.2	整数二分	19
2.2.3	整数二分 最大值最小化	22
2.2.4	整数二分 最小值最大化	23
2.2.5	实数二分	23
2.3	三分法	24
2.4	排序与排列	27

1 基础数据结构

1.1 Link List in Python:

```
1 class nodes:
2     def __init__(self, val=None, pre=None, next=None):
3         self.val = val
4         self.next = next
```

EX 1

洛谷 P1996 约瑟夫问题

题目描述

n 个人围成一圈，从第一个人开始报数，数到 m 的人出列，再由下一个人重新从 1 开始报数，数到 m 的人再出圈，依次类推，直到所有的人都出圈，请输出依次出圈人的编号。

注意：本题和《深入浅出-基础篇》上例题的表述稍有不同。书上表述是给出淘汰 n-1 名小朋友，而该题是全部出圈。

输入格式

输入两个整数 n,m。

输出格式

输出一行 n 个整数，按顺序输出每个出圈人的编号。

输入输出样例

输入 #1

10 3

输出 #1

3 6 9 2 7 1 8 5 10 4

说明/提示

1 m,n 100

```
1 # 动态链表
2 class node:
3     data = None
4     next = None
5
6 n, m = map(int, input().split())
7 # init link list
8 head = node()
9 head.data = 1
10 now = head
11 for i in range(2, n+1):
12     p = node()
13     p.data = i
14     now.next = p
15     now = p
16 now.next = head
17 prev = now
18 now = head
19 while n > 0:
20     n -= 1
21     for i in range(m-1):
22         prev = now
23         now = now.next
24     print(now.data, end=' ')
25     prev.next = now.next
26     now = now.next
```

```
1 # 列表和索引计算
2 n, m = map(int, input().split())
3 people = list(range(1, n+1))
4 rst = []
5 current = 0
6 index = 0
7 while people:
8     index = (current + m - 1) % len(people)
9     rst.append(people.pop(index))
10    current = index
11 print(' '.join(map(str, rst)))
```

EX 2

洛谷 P1160 队列安排

题目描述

一个学校里老师要将班上 N 个同学排成一列，同学被编号为 $1 \sim N$ ，他采取如下的方法：

1. 先将 1 号同学安排进队列，这时队列中只有他一个人；
2. $2 \sim N$ 号同学依次入列，编号为 i 的同学入列方式为：老师指定编号为 i 的同学站在编号为 $1 \sim (i - 1)$ 中某位同学（即之前已经入列的同学）的左边或右边；
3. 从队列中去掉 M 个同学，其他同学位置顺序不变。

在所有同学按照上述方法队列排列完毕后，老师想知道从左到右所有同学的编号。

输入格式

第一行一个整数 N ，表示了有 N 个同学。

第 $2 \sim N$ 行，第 i 行包含两个整数 k, p ，其中 k 为小于 i 的正整数， p 为 0 或者 1。若 p 为 0，则表示将 i 号同学插入到 k 号同学的左边， p 为 1 则表示插入到右边。

第 $N + 1$ 行为一个整数 M ，表示去掉的同学数目。

接下来 M 行，每行一个正整数 x ，表示将 x 号同学从队列中移去，如果 x 号同学已经不在队列中则忽略这一条指令。

输出格式

一行，包含最多 N 个空格隔开的整数，表示了队列从左到右所有同学的编号。

输入输出样例 #1

输入 #1

4 1 0 2 1 1 0 2 3 3

输出 #1

2 4 1

说明/提示

** 【样例解释】 **

将同学 2 插入至同学 1 左边，此时队列为：

2 1

将同学 3 插入至同学 2 右边，此时队列为：

2 3 1

将同学 4 插入至同学 1 左边，此时队列为：

2 3 4 1

将同学 3 从队列中移出，此时队列为：

2 4 1

同学 3 已经不在队列中，忽略最后一条指令

最终队列：

2 4 1

【数据范围】

对于 20% 的数据, $1 \leq N \leq 10$ 。

对于 40% 的数据, $1 \leq N \leq 1000$ 。

对于 100% 的数据, $1 < M \leq N \leq 10^5$ 。

```
1  # 链表
2  class nodes:
3      def __init__(self, val=None, next=None, prev=None):
4          self.val = val
5          self.next = next
6          self.prev = prev
7
8  N = int(input())
9  people = nodes(1)
10 head = people
11 idx = {1: people}
12 for i in range(2, N+1):
13     node = nodes(i)
14     k, p = map(int, input().split())
15     k = idx[k]
16     if p == 0:
17         if k.prev:
18             k.prev.next = node
19             node.prev = k.prev
20             node.next = k
21             k.prev = node
22         if k == head:
23             head = node
24     elif p == 1:
25         if k.next:
26             k.next.prev = node
27             node.next = k.next
28             node.prev = k
29             k.next = node
30     idx[i] = node
31
32 M = int(input())
33 for i in range(M):
34     x = int(input())
35     if x in idx.keys():
36         k = idx[x]
37         if k.prev:
38             k.prev.next = k.next
39             if k.next:
40                 k.next.prev = k.prev
41         if k.next:
42             k.next.prev = k.prev
43             if k.prev:
44                 k.prev.next = k.next
45         if k == head:
46             head = k.next
47     del idx[x]
48
49 while head:
50     print(head.val, end=' ')
51     head = head.next
```

1.2 Queue in Python

1.2.1 双端队列

```
1 from collections import deque # 双端队列
2
3 queue = deque(maxlen = 10) # 最大长度为10, None为无限制
4 queue = deque(iterable) # init queue by iterable obj.
5
6 queue.append(x) # add x to right side
7 queue.appendleft(x) # add x to left side
8
9 queue.extend(iterable) # extend right side by iterable obj.
10 queue.extendleft(iterable) # extend the left side by iterable obj.
11
12 queue.pop() # pop element from right side
13 queue.popleft() # pop element from left side
14
15 queue.remove(x) # remove x from Queue, raise error if not found
16 queue.clear() # clear Queue
17
18 queue.reverse() # reverse Queue
19 queue.insert(i, x) # insert x into queue at position i
20 queue.count(x) # count the number of queue elements = x
21 queue.index(x) # return first match position, raise error if not found
```

EX 1

洛谷 P1540 [NOIP 2010 提高组] 机器翻译

题目背景

NOIP2010 提高组 T1

题目描述

小晨的电脑上安装了一个机器翻译软件，他经常用这个软件来翻译英语文章。

这个翻译软件的原理很简单，它只是从头到尾，依次将每个英文单词用对应的中文含义来替换。对于每个英文单词，软件会先在内存中查找这个单词的中文含义，如果内存中有，软件就会用它进行翻译；如果内存中没有，软件就会在外存中的词典内查找，查出单词的中文含义然后翻译，并将这个单词和译义放入内存，以备后续的查找和翻译。

假设内存中有 M 个单元，每单元能存放一个单词和译义。每当软件将一个新单词存入内存前，如果当前内存中已存入的单词数不超过 $M - 1$ ，软件会将新单词存入一个未使用的内存单元；若内存中已存入 M 个单词，软件会清空最早进入内存的那个单词，腾出单元来，存放新单词。

假设一篇英语文章的长度为 N 个单词。给定这篇待译文章，翻译软件需要去外存查找多少次词典？假设在翻译开始前，内存中没有任何单词。

输入格式

共 2 行。每行中两个数之间用一个空格隔开。

第一行为两个正整数 M, N ，代表内存容量和文章的长度。

第二行为 N 个非负整数，按照文章的顺序，每个数（大小不超过 1000）代表一个英文单词。文章中两个单词是同一个单词，当且仅当它们对应的非负整数相同。

输出格式

一个整数，为软件需要查词典的次数。

输入输出样例 #1

输入 #1

3 7 1 2 1 5 4 4 1

输出 #1

5

说明/提示

样例解释

整个查字典过程如下：每行表示一个单词的翻译，冒号前为本次翻译后的内存状况：

1. 1: 查找单词 1 并调入内存。2. 1 2: 查找单词 2 并调入内存。3. 1 2: 在内存中找到单词 1。4. 1 2 5: 查找单词 5 并调入内存。5. 2 5 4: 查找单词 4 并调入内存替代单词 1。6. 2 5 4: 在内存中找到单词 4。7. 5 4 1: 查找单词 1 并调入内存替代单词 2。

共计查了 5 次词典。

数据范围

- 对于 10% 的数据有 $M = 1, N \leq 5$ ；- 对于 100% 的数据有 $1 \leq M \leq 100, 1 \leq N \leq 1000$ 。

```
1 # 队列 collections.deque
2 from collections import deque
3 from array import array
4
5 M, N = map(int, input().split())
6 q = deque(maxlen=M) # 双向队列
7 qs = set() # 用作哈希表
8 count = 0 # 计数
9 inp_ary = array('i', map(int, input().split())) # 输入数据
10 l = len(inp_ary)
11 for i in range(l):
12     if inp_ary[i] not in qs:
13         if len(q) == M:
14             qs.remove(q.popleft())
15             qs.add(inp_ary[i])
16             q.append(inp_ary[i])
17         count += 1
18 print(count)
```

1.2.2 单调队列

洛谷 P1886 滑动窗口 / 【模板】单调队列

题目描述

有一个长为 n 的序列 a ，以及一个大小为 k 的窗口。现在这个从左边开始向右滑动，每次滑动一个单位，求出每次滑动后窗口中的最大值和最小值。

例如，对于序列 $[1, 3, -1, -3, 5, 3, 6, 7]$ 以及 $k = 3$ ，有如下过程：

窗口位置								最小值	最大值
[1	3	-1]	-3	5	3	6	7	-1	3
1	[3	-1	-3]	5	3	6	7	-3	3
1	3	[-1	-3	5]	3	6	7	-3	5
1	3	-1	[-3	5	3]	6	7	-3	5
1	3	-1	-3	[5	3	6]	7	3	6
1	3	-1	-3	5	[3	6	7]	3	7

输入格式

输入一共有两行，第一行有两个正整数 n, k 。第二行 n 个整数，表示序列 a

输出格式

输出共两行，第一行为每次窗口滑动的最小值第二行为每次窗口滑动的最大值

输入输出样例 #1

输入 #1

8 3 1 3 -1 -3 5 3 6 7

输出 #1

-1 -3 -3 -3 3 3 3 3 5 5 6 7

说明/提示

【数据范围】对于 50% 的数据， $1 \leq n \leq 10^5$ ；对于 100% 的数据， $1 \leq k \leq n \leq 10^6$ ， $a_i \in [-2^{31}, 2^{31})$ 。

```

1 from collections import deque
2 from array import array
3 import sys
4
5 # input
6 n, k = map(int, input().split())
7 a = list(map(int, input().split()))
8
9 min_q = deque()
10 max_q = deque()
11 min_rst = []
12 max_rst = []
13
14 for i in range(n):
15     # minimum queue
16     while min_q and a[min_q[-1]] > a[i]:
17         min_q.pop()
18     min_q.append(i)
19     # maximum queue
20     while max_q and a[max_q[-1]] < a[i]:
21         max_q.pop()
22     max_q.append(i)
23
24     # 开始记录
25     if i >= k - 1:
26         min_rst.append(a[min_q[0]])
27         max_rst.append(a[max_q[0]])
28
29     # 弹出窗口外的元素
30     if min_q and min_q[0] <= i - k + 1:
31         min_q.popleft()
32     if max_q and max_q[0] <= i - k + 1:
33         max_q.popleft()
34
35 # output
36 print(' '.join(map(str, min_rst)))
37 print(' '.join(map(str, max_rst)))

```

1.2.3 单调队列与动态规划

通过前缀和 + 单调队列 + 动态规划求解子序和问题

洛谷 P1714

Problem Description

给定一个序列，给定一个最大长度 m ，求一段长度不超过 m 的连续子序列，使其子序和最大

Input

第 1 行输入 n, m 。分别为序列长度和最大长度

第 2 行输入 N 个数

Output

第 1 个数是最大子序和，第 2 和第 3 个数是开始和终止位置

Case 1:

Input

5 2

1 2 3 4 5

Output

9

Case 2:

Input

6 3

1 -2 3 -4 5 -6

Output

5

Case 3

Input

5 5

1 2 3 4 5

Output

15

```
1 # 洛谷 P1714
2
3 from collections import deque
4 n, m = map(int, input().split())
5 s = list(map(int, input().split())) # 前缀和
6 dp = deque()
7 ans = s[0]
8
9 # 计算前缀和
10 for i in range(1, n):
11     s[i] = s[i-1] + s[i]
12 # 单调队列
13 dp.append(0)
14 for i in range(1, n):
15     # 去头
16     while len(dp) > 0 and dp[0] < i - m:
17         dp.popleft()
18     # 去尾, 去除>=s[i]的元素, 使得s[j] - s[i]更大
19     while len(dp) > 0 and s[dp[-1]] >= s[i]:
20         dp.pop()
21     dp.append(i)
22     if len(dp) == n:
23         ans = max(ans, s[i])
24     elif len(dp) > 1:
25         ans = max(ans, s[i] - s[dp[0]])
26     elif len(dp) == 1:
27         ans = max(ans, s[dp[0]])
28     else:
29         ans = max(ans, s[i])
30
31 print(ans)
```

1.2.4 优先队列

```
1 from queue import PriorityQueue
2 MAXSIZE = 0
3
4 q = PriorityQueue(maxsize=MAXSIZE) # 队列大小，默认为0，<=0的队列大小为无穷
5 q1 = PriorityQueue()
6
7 q.empty() # if empty
8 q.full() # if full
9 q.qsize() # get current size
10
11 # add elem
12 # 两种添加方式不能混用
13 # 直接按照元素大小存入，元素大小越小，优先级越高
14 q.put(1)
15 q.put(2)
16 # q.put((priority number, data))
17 # priority number越小，优先级越高
18 q1.put((2, 'ele'))
19 q1.put((-3, 'ele2'))
20
21 # get element, = pop()
22 q.get()
```

1.3 Stack in Python

使用 deque 作为栈.

hdu 1062

翻转字符串

input:

olleh !dlrow

output:

hello world!

```
1 from collections import deque
2 s = deque()
3 inp = list(input().split())
4 for item in inp:
5     # enter stack
6     for i in item:
7         s.append(i)
8     while len(s) > 0:
9         print(s.pop(), end=' ')
10    print(' ', end=' ')
```

1.3.1 单调栈

洛谷 P2947 [USACO09MAR] Look Up S

题目描述

Farmer John's N ($1 \leq N \leq 100,000$) cows, conveniently numbered $1..N$, are once again standing in a row. Cow i has height H_i ($1 \leq H_i \leq 1,000,000$).

Each cow is looking to her left toward those with higher index numbers. We say that cow i 'looks up' to cow j if $i < j$ and $H_i < H_j$. For each cow i , FJ would like to know the index of the first cow in line looked up to by cow i .

Note: about 50

约翰的 $N(1 \leq N \leq 10^5)$ 头奶牛站成一排，奶牛 i 的身高是 $H_i(1 \leq H_i \leq 10^6)$ 。现在，每只奶牛都在向右看齐。对于奶牛 i ，如果奶牛 j 满足 $i < j$ 且 $H_i < H_j$ ，我们可以说奶牛 i 可以仰望奶牛 j 。求出每只奶牛离她最近的仰望对象。

Input

输入格式

1. Line 1: A single integer: N

Lines 2.. $N+1$: Line $i+1$ contains the single integer: H_i

第 1 行输入 N ，之后每行输入一个身高 H_i 。

输出格式

Lines 1.. N : Line i contains a single integer representing the smallest index of a cow up to which cow i looks.

If no such cow exists, print 0.

共 N 行，按顺序每行输出一只奶牛的最近仰望对象，如果没有仰望对象，输出 0。

输入输出样例 # 1

输入 # 1

6

3

2

6

1

1

2

输出 # 1

3

3

0

6

6

0

说明/提示

FJ has six cows of heights 3, 2, 6, 1, 1, and 2.

Cows 1 and 2 both look up to cow 3; cows 4 and 5 both look up to cow 6; and cows 3 and 6 do not look up to any cow.

【输入说明】6 头奶牛的身高分别为 3,2,6,1,1,2。

【输出说明】奶牛 #1,#2 仰望奶牛 #3，奶牛 #4,#5 仰望奶牛 #6，奶牛 #3 和 #6 没有仰望对象。

【数据规模】

对于 20% 的数据： $1 \leq N \leq 10$;

对于 50% 的数据： $1 \leq N \leq 10^3$;

对于 100% 的数据: $1 \leq N \leq 10^5, 1 \leq H_i \leq 10^6$ 。

```

1 from collections import deque
2 from array import array
3 n = int(input())
4 cows = []
5 for i in range(n):
6     cows.append(int(input()))
7 s = deque()
8 rst = array('i')
9 for i in range(n-1, -1, -1):
10     # 由于反向遍历, 栈内元素必然在当前遍历元素的右边
11     # 不比当前高的元素出栈, 保证栈底到栈顶的高度递减
12     # 目的是使得随着逐渐出栈, 栈顶元素逐渐变高, 以找到能比当前遍历元素高的元素
13     while len(s) > 0 and cows[s[-1]] <= cows[i]:
14         s.pop()
15     if len(s) == 0:
16         rst.append(0)
17     else:
18         rst.append(s[-1] + 1) # relationship between index & NO.
19     s.append(i)
20 for i in range(len(rst)-1, -1, -1):
21     print(rst[i])

```

1.4 Binary Tree in Python

```

1 class nodes:
2     def __init__(self, val=None, l=None, r=None):
3         self.val = val
4         self.l = l
5         self.r = r

```

1.4.1 DFS 遍历

```

1 # pre
2 def preorder(node):
3     if not node:
4         return
5     print(node.val, end=' ')
6     preorder(node.l)
7     preorder(node.r)
8
9 # mid
10 def inorder(node):
11     if not node:
12         return
13     inorder(node.l)
14     print(node.val, end=' ')
15     inorder(node.r)
16
17 # post
18 def postorder(node):
19     if not node:
20         return
21     postorder(node.l)
22     postorder(node.r)
23     print(node.val, end=' ')

```

```
1 # pre
2 def preorder(node):
3     if not node:
4         return
5     print(node.val, end=' ')
6     preorder(node.l)
7     preorder(node.r)
8
9 # mid
10 def inorder(node):
11     if not node:
12         return
13     inorder(node.l)
14     print(node.val, end=' ')
15     inorder(node.r)
16
17 # post
18 def postorder(node):
19     if not node:
20         return
21     postorder(node.l)
22     postorder(node.r)
23     print(node.val, end=' ')

```

```
1 # test code
2 ab = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
3 n = list(nodes(i) for i in ab)
4 root = n[4]
5 root.l = n[1]
6 root.r = n[6]
7 root.l.l = n[0]
8 root.l.r = n[3]
9 root.l.r.l = n[2]
10 root.r.l = n[5]
11 root.r.r = n[-1]
12 root.r.r.l = n[-2]

```

```
1 // output
2 E B A D C G F I H
3 A B C D E F G H I
4 A C D B F H I G E

```

根据前序和中序遍历输出后序遍历结果

```
1 n = int(input())
2 global pre_tra
3 pre_tra = list(map(int, input().split()))
4 in_tra = list(map(int, input().split()))
5
6 root = nodes(pre_tra[0])
7 pre_tra = pre_tra[1:]
8 # split left and right subtree
9 mid = in_tra.index(root.val)
10 lp = in_tra[0:mid]
11 rp = in_tra[mid+1:]
12
13 def establish(root, lpart, rpart):
14     global pre_tra
15     # left part
16     if lpart and pre_tra:
17         # connect subtree
18         pivot = nodes(pre_tra[0])
19         pre_tra = pre_tra[1:]
20         root.l = pivot
21         # split left and right subtree
22         mid = lpart.index(pivot.val)
23         lp = lpart[0:mid]
24         rp = lpart[mid+1:]
25         establish(pivot, lp, rp)
26     # right part
27     if rpart and pre_tra:
28         # connect subtree
29         pivot = nodes(pre_tra[0])
30         pre_tra = pre_tra[1:]
31         root.r = pivot
32         # split left and right subtree
33         mid = rpart.index(pivot.val)
34         lp = rpart[0:mid]
35         rp = rpart[mid+1:]
36         establish(pivot, lp, rp)
37
38 establish(root, lp, rp)
39 postorder(root)
```

1.4.2 哈夫曼编码

poj 1521

Question: 输入一个字符串, 分别输出 ASCII(8bit / character) 和哈夫曼编码的长度, 以及压缩比

Sample Input

AAAAABCD

THE_CAT_IN_THE_HAT

END

Sample Output

64 13 4.9

144 51 2.8

```

1 from queue import PriorityQueue
2 s = input()
3 while s != 'END':
4     chaSet = set(item for item in s)
5     q = PriorityQueue()
6
7     # 只有一个字符的情况，建哈夫曼树至少需要两个节点
8     if len(s) == 1:
9         print('8 1 8')
10    else:
11        for item in chaSet:
12            q.put(s.count(item))
13        rst = 0
14        while q.qsize() > 1:
15            a = q.get()
16            b = q.get()
17            q.put(a + b)
18            # 每增加一层，编码长度加1
19            rst += a + b
20
21        # clear queue
22        q.get()
23        print('{ } { } '.format(str(8 * len(s)),
24                                str(rst),
25                                str(8 * len(s) / rst)))
26
27    s = input()

```

1.5 Heap in Python

```

1 import heapq
2
3 # heapq创建的是小根堆，通过对元素取负可以转换为大根堆
4
5 # create a heap
6 # convert a list to heap
7 lst = [2, 8, 1, 63, 8, 1, 0, 4]
8 heapq.heapify(lst)
9 # 此后堆heap的任何操作都要通过库函数操作
10 print(lst)
11
12 # heapq.heappush(heap, item)
13 heapq.heappush(lst, 5)
14 print(lst)
15
16 # heapq.heappop(heap)
17 # -> pop and return the min element of heap
18 print(heapq.heappop(lst))
19 print(lst)
20
21 # heapq.heappushpop(heap, item)
22 # -> push item into heap, and return the min element
23 print(heapq.heappushpop(lst, 4))
24 print(lst)

```

P3378 【模板】堆

题目描述

给定一个数列，初始为空，请支持下面三种操作：

1. 给定一个整数 x ，请将 x 加入到数列中。
2. 输出数列中最小的数。

3. 删除数列中最小的数（如果有多个数最小，只删除 1 个）。

输入格式

第一行是一个整数，表示操作的次数 n 。

接下来 n 行，每行表示一次操作。每行首先有一个整数 op 表示操作类型。

- 若 $op = 1$ ，则后面有一个整数 x ，表示要将 x 加入数列。
- 若 $op = 2$ ，则表示要求输出数列中的最小数。
- 若 $op = 3$ ，则表示删除数列中的最小数。如果有多个数最小，只删除 1 个。

输出格式

对于每个操作 2，输出一行一个整数表示答案。

输入输出样例 1

输入 1

5

1 2

1 5

2

3

2

输出 1

2

5

说明/提示

【数据规模与约定】

- 对于 30% 的数据，保证 $n \leq 15$ 。
- 对于 70% 的数据，保证 $n \leq 10^4$ 。
- 对于 100% 的数据，保证 $1 \leq n \leq 10^6$ ， $1 \leq x < 2^{31}$ ， $op \in \{1, 2, 3\}$ 。

```
1 import heapq
2 lt = list()
3 heapq.heapify(lt)
4 n = int(input())
5 for i in range(n):
6     op = list(map(int, input().split()))
7     if len(op) == 2:
8         heapq.heappush(lt, op[1])
9     elif len(op) == 1:
10         if op[0] == 2:
11             rst = heapq.nsmallest(1, lt)
12             print(rst[0])
13         elif op[0] == 3:
14             heapq.heappop(lt)
```

2 基本算法

2.1 尺取法 (双指针)

用以解决序列的区间问题, 一般有两个要求:

1. 序列是有序的, 需要先对序列进行排序
2. 问题与序列的区间有关, 操作两个或多个指针 i, j 表示区间

在 Python 中, 用 while 实现较为方便

扫描方向:

反向扫描, 左右指针: i, j 的方向相反;

同向扫描, 快慢指针: i, j 的方向相同, 但是扫描速度一般不同, 可以形成一个大小可变的滑动窗口

2.1.1 反向扫描

找指定和的整数对

问题: 输入 n ($n \leq 100000$) 个整数, 放在数组 $a[]$ 中. 找出其中的两个数, 它们之和等于整数 m . (假定肯定有解).

输入:

第 1 行是数组 $a[]$, 第 2 行是 m

Sample input

21 4 5 6 13 65 32 9 23

28

Sample output

5 23

```

1 # 哈希 复杂度为 $O(n)$ , 但是需要较大的哈希空间
2 a = list(map(int, input().split()))
3 m = int(input())
4 s = set(a)
5 outed = set()
6 for item in s:
7     if m - item in s and item not in outed:
8         print(item, m - item)
9         outed.add(m-item)

```

```

1 # 尺取法 复杂度为 $O(n \log_2 n)$ , 其中, 排序的复杂度为 $O(\log_2 n)$ , 检查的复杂度为 $O(n)$ 
2 a = list(map(int, input().split()))
3 a.sort()
4 m = int(input())
5
6 # 双指针
7 i, j = 0, len(a) - 1
8 while (i < j):
9     s = a[i] + a[j]
10    #  $s < m$ :  $i$ 增加1, 之后的 $s \geq$ 当前 $s$ 
11    if s < m:
12        i += 1
13    elif s > m:
14        j -= 1
15    else:
16        print("{} {}".format(a[i], a[j]))
17        i += 1

```

判断回文串

输入: 第 1 行输入测试实例个数, 之后每行输入一个字符串

输出: 是回文串输出 yes, 不是输出 no

```
1 n = int(input())
2 for i in range(n):
3     s = str(input())
4     i, j = 0, len(s) - 1
5     while i < j:
6         flag = False
7         if s[i] == s[j]:
8             flag = True
9         else:
10            flag = False
11            break
12        i += 1
13        j -= 1
14    if (flag):
15        print('yes')
16    else:
17        print('no')
```

2.1.2 同向扫描

使用尺取法产生滑动窗口

寻找区间和

给定一个长度为 n 的正整数数组 $a[]$ 和一个数 s , 在数组中找一个区间, 使得该区间的数组元素之和等于 s .

输出区间的起点和终点位置

第 1 行输入数组长度 n , 第 2 行输入数组, 第 3 行为 s

Sample input

15

6 1 2 3 4 6 4 2 8 9 10 11 12 13 14

6

Sample output

0 0

1 3

5 5

6 7

初始值 $i = j = 0$

如果 $sum = s$: 输出一个解, sum 减去 $a[i]$, $i++$

如果 $sum < s$: $j++$, $sum + a[j]$

如果 $sum > s$: $sum - a[i]$, $i++$

```
1 n = int(input())
2 a = list(map(int, input().split()))
3 s = int(input())
4
5 sum = a[0]
6 i, j = 0, 0
7 while i < n and j < n:
8     if sum == s:
9         print("{} {}".format(i, j))
10        sum -= a[i]
11        i += 1
12        j += 1
13        sum += a[j]
14    elif sum < s:
15        j += 1
16        sum += a[j]
17    elif sum > s:
18        sum -= a[i]
19        i += 1
```

数组去重

给出一个数组, 输出去除重复元素之后的数组

```
1 # 哈希, 数据多或者数值过大时需要占用大量的空间
2 a = list(map(int, input().split()))
3 s = set(a)
4 unique_a = list(s)
5 print(unique_a)
```

```
1 # 尺取法
2 a = list(map(int, input().split()))
3 # a排序, 使得相同元素排列在一起
4 a.sort()
5 n = len(a)
6 # 双指针均从0开始
7 i, j = 0, 0
8 # j始终指向无重复元素部分的最后一个元素
9 while i < n and j < n:
10     # 若i和j指向的元素不同, j++, 将i指向的元素复制到j上
11     # EX: 1 2(j) 3(i) 3
12     # -> 1 2 3(j, i) 3
13     # EX: 1 2 3(j) 3 4(i)
14     # -> 1 2 3 3(j) 4(i)
15     # -> 1 2 3 4(j) 4(i)
16     if a[i] != a[j]:
17         j += 1
18         a[j] = a[i]
19     i += 1
20 unique_a = a[0:j+1]
21 print(unique_a)
```

找相同数对

洛谷 P1102

给出一串数字和一个数字 C , 要求计算出所有 $A - B = C$ 的数对的个数 (不同位置的数字一样的数对算不同的数对)

输入: 2 行, 第 1 行输入整数 n 和 C , 第 2 行输入 n 个整数

输出: 满足 $A - B = C$ 的数对的个数

Sample Input

6 3

8 4 5 7 7 4

Sample Output

5

```
1 n, c = map(int, input().split())
2 a = list(map(int, input().split()))
3 a.sort()
4
5 i, j, k = 0, 0, 0
6 ans = 0
7
8 for i in range(n):
9     # j, k指向相同元素区间的起点和终点后1个元素
10    # 寻找的对象是区间内的元素 - a[i] = C
11    while j < n - 1 and a[j] - a[i] < c:
12        j += 1
13    while k < n and a[k] - a[i] <= c:
14        k += 1
15    if a[j] - a[i] == c and a[k-1] - a[i] == c and k - 1 >= 0:
16        ans += k - j
17 print(ans)
```

2.2 二分法

2.2.1 Python 二分搜索库 bisect

```
1 from bisect import *
2 def fun(find, x, bias=0):
3     global a
4     index = find(a, x) + bias
5     print("index: {}, element: {}".format(index, a[index]))
6
7 global a
8 a = [1, 2, 4, 4, 4, 5]
9 # target: x
10 x = 4
11
12 # first > x
13 fun(bisect_right, x)
14 # first >= x
15 fun(bisect_left, x)
16 # first = x
17 fun(bisect_left, x)
18 # last = x
19 fun(bisect_right, x, bias=-1)
20 # last <= x
21 fun(bisect_right, 3, bias=-1)
22 # last < x
23 fun(bisect_left, x, bias=-1)
24 # count x in a monotonic array
25 # slow
26 print(a.count(x))
27 # fast, using binary search
28 print(bisect_right(a, x) - bisect_left(a, x))
```

output

```
1 index: 5, element: 5
2 index: 2, element: 4
3 index: 2, element: 4
4 index: 4, element: 4
5 index: 1, element: 2
6 index: 1, element: 2
7 3
8 3
```

2.2.2 整数二分

需要注意终止边界和左右区间问题, 避免漏解和死循环

mid 的计算

```
1 # 适用单调递增序列的后继问题
2 mid = l + (r - l) // 2 # 相当于计算出的mid向下取整, 计算的是左中位数
3 # 适用单调递增序列的前驱问题
4 mid = l + (r - l + 1) // 2 # 相当于计算出的mid向上取整, 计算的是右中位数
```

在单调递增序列中寻找 x 或 x 的后继

在单调递增序列中寻找第一个 x 的位置, 若没有 x , 则寻找比 x 大的第一个数的位置, 即寻找第一个 $\geq x$ 的位置

```
1 # 左闭右开 [0, n)
2 l, r = 0, n
3 while l < r:
4     mid = l + (r - l) // 2
5     if a[mid] >= x:
6         r = mid
7     else:
8         l = mid + 1
9 return l
```

在单调递增序列中寻找 x 或 x 的前驱

在单调递增序列中寻找第一个 x 的位置, 若没有 x , 则寻找比 x 小的第一个数的位置, 即寻找第一个 $\leq x$ 的位置

```
1 # 左开右闭 (-1, n-1]
2 l, r = -1, n - 1
3 while l < r:
4     mid = l + (r - l + 1) // 2
5     if a[mid] <= x:
6         l = mid
7     else:
8         r = mid - 1
9 return l
```

寻找 minimum

```
1 while l < r:
2     mid = l + (l - r) // 2
3     if check(mid):
4         # reduce
5         r = mid
6     else:
7         # enlarge
8         l = mid + 1
9 # 若  $r = mid - 1$ , 则当  $best=mid$  时可能遗漏最优解
```

寻找指定和的整数对

输入 n ($n \leq 100000$) 个整数, 找出其中的两个数, 使它们之和等于整数 m , 假设肯定有解

```
1 from bisect import *
2
3 a = list(map(int, input().split()))
4 m = int(input())
5 n = len(a)
6 a.sort()
7
8 # ver. 1
9 for i in range(n-1):
10     # bisearch, a[k] = m - a[i]
11     l, r = i+1, n
12     x = m - a[i]
13     while l < r:
14         mid = l + (r - l) // 2
15         if a[mid] >= x:
16             r = mid
17         else:
18             l = mid + 1
19     if a[l] == x:
20         print("{} {}".format(a[i], a[l]))
21
22 # ver. 2
23 for i in range(n-1):
24     # bisearch, a[k] = m - a[i]
25     x = m - a[i]
26     # search from a[i+1] to a[end-]
27     p = bisect_left(a, x, lo=i+1, hi=n)
28     if a[p] == x:
29         print("{} {}".format(a[i], a[p]))
```

2.2.3 整数二分 最大值最小化

序列划分: 二分 + 贪心

给定一个序列, 如 $\{2, 2, 3, 4, 5, 1\}$, 将其划分为 m 个连续的子序列 S_1, S_2, S_3 , 每个子序列至少有一个元素, 使得每个子序列的值的最大值最小

EX. $m = 3$

划分为 $(2, 2, 3), (4, 5), (1)$ 子序列和分别为 7, 9, 1, 最大值为 9

划分为 $(2, 2, 3), (4), (5, 1)$ 子序列和为 7, 4, 6, 最大值为 7, 优于前一个

```

1 # Input:
2 # array
3 # m: amount of subarray
4 # Output:
5 # x: minimum of maximum of sum(all possible subarrays)
6 # subarrays
7
8 from bisect import *
9
10 a = list(map(int, input().split()))
11 m = int(input())
12 n = len(a)
13 l, r = max(a), sum(a)
14 subs = []
15
16 while l < r:
17     mid = l + (r - l) // 2
18     # greedy divide subarray
19     # each sum(subarray) <= mid
20     flag = False
21     idx = 0
22     subs = []
23     for i in range(m):
24         sum = 0
25         while idx < n and sum + a[idx] <= mid:
26             sum += a[idx]
27             idx += 1
28         if idx == n or sum + a[idx] > mid:
29             subs.append(idx-1)
30     # judge
31     if subs[-1] < n-1:
32         flag = False
33     else:
34         flag = True
35     # binary control
36     if flag:
37         # reduce
38         r = mid
39     else:
40         # enlarge
41         l = mid + 1
42
43 print('minimum sum: ', l)
44 for i in range(m):
45     if i == 0:
46         print(a[ : subs[0]+1])
47     else:
48         print(a[subs[i-1]+1 : subs[i]+1])

```

2.2.4 整数二分 最小值最大化

洛谷 P1824 进击的奶牛

在一条很长的直线上, 指定 n 个坐标点. 有 c 头牛, 每头牛占据一个坐标点, 求相邻两头牛之间距离的最大值

Input:

第 1 行输入: $n\ c$

第 2 行开始每行输入: 一个整数, 表示每个点的坐标

```

1 n, c = map(int, input().split())
2 x = []
3 for i in range(n):
4     x.append(int(input()))
5 x.sort()
6
7 def check(x, dis, c):
8     i = 1
9     last = 0
10    c -= 1
11    while c > 0 and i < len(x):
12        while i < len(x) and x[i] - x[last] < dis:
13            i += 1
14        if i < len(x) and x[i] - x[last] >= dis:
15            c -= 1
16            last = i
17            i += 1
18    if c == 0:
19        return True
20    else:
21        return False
22
23 l, r = 0, x[-1] - x[0]
24 ans = 0
25 while l < r:
26     mid = l + (r - l) // 2
27     if check(x, mid, c):
28         ans = mid
29         l = mid + 1
30     else:
31         r = mid
32 print(ans)

```

2.2.5 实数二分

for 控制: 过大的 for 次数会超时, 过小的会导致精度不够答案错误. 一般取 100, 但是循环体内计算量大的时候容易超时, 可以缩减到 50

while 控制: while 需要设计精度 eps, 过小的 eps 会超时, 过大的会导致精度不够答案错误

```

1 # 精度, 可以调整
2 eps = 1e-7
3 while r - l > eps:
4     # for ver.
5     # epoch = 100 # 轮次
6     # for i in range(epochs):
7         mid = l + (r - l) / 2.0
8         if check(mid):
9             # reduce range
10            r = mid
11        else:
12            # enlarge range
13            l = mid
14 return l

```

分蛋糕 poj 3122

$m+1$ 个人分 n 个半径不同的蛋糕, 要求每个人分得的蛋糕重量一致, 且必须是可以切出来的一整块, 每个人能分到的最大蛋糕是多少.

Input:

第一行: 1 个整数, 表示测试用例个数

对每个测试, 第一行输入 n, m . 第二行输入 n 个整数, 表示每个蛋糕的半径

Output:

对于每个测试, 输出一个答案, 保留 4 位小数

可以将问题建模为最小值最大化问题, 用面积代替重量

保留小数

```
1 a = 1.13456 # float
2 ans = format(a, '.4f') # 四舍五入保留4位小数
```

```
1 def check(mid, area, f):
2     sum = 0
3     for i in range(len(area)):
4         sum += int(area[i] / mid)
5     if sum >= f:
6         return True
7     else:
8         return False
9
10 eps = 1e-5
11 pi = 3.1415926
12 T = int(input())
13 for t in range(T):
14     n, f = map(int, input().split())
15     cakes = list(map(float, input().split()))
16     maxx = 0
17     area = []
18     for i in range(n):
19         area.append(pi * cakes[i] * cakes[i])
20         if area[i] > maxx:
21             maxx = area[i]
22     l, r = 0, maxx
23     while r - l > eps:
24         mid = l + (r - l) / 2.0
25         if check(mid, area, f):
26             l = mid
27         else:
28             r = mid
29     ans = format(l, '.4f')
30     print(ans)
```

2.3 三分法

用于求取单峰函数的极值. 通过在 $[l, r]$ 内取两个点 **mid1**, **mid2**, 将函数分为三段

```
1 k = (r - l) / 3.0
2 mid1, mid2 = l + k, r - k
```

三分法模板 洛谷 P3382

给出一个 N 次多项式函数, 保证在区间 $[l, r]$ 内存在一点 x , 使得 x 是函数在区间上的极大值, 求出 x

Input:

第一行输入 n : $N \mid r$

第二行输入: $N + 1$ 个实数, 表示从高到低各项的系数

Output:

x , 四舍五入保留 5 位小数

```
1 n, l, r = map(float, input().split())
2 n = int(n)
3 cons = list(map(float, input().split()))
4 def f(x, n, cons):
5     sum = 0
6     for i in range(1, n+2):
7         sum += cons[-i] * pow(x, i-1)
8     return sum
9 eps = 1e-6
10 while r - l > eps:
11     k = (r - l) / 3.0
12     mid1, mid2 = l + k, r - k
13     if f(mid1, n, cons) < f(mid2, n, cons):
14         l = mid1
15     else:
16         r = mid2
17 ans = format(l, '.5f')
18 print(ans)
```

三分法函数洛谷 P1883

给定 n 个二次函数 $f_1(x), f_2(x), \dots, f_n(x)$ (均形如 $ax^2 + bx + c$), 设 $F(x) = \max\{f_1(x), f_2(x), \dots, f_n(x)\}$, 求 $F(x)$ 在区间 $[0, 1000]$ 上的最小值。

输入格式

输入第一行为正整数 T , 表示有 T 组数据。

每组数据第一行一个正整数 n , 接着 n 行, 每行 3 个整数 a, b, c , 用来表示每个二次函数的 3 个系数, 注意二次函数有可能退化成一个。

输出格式

每组数据输出一行, 表示 $F(x)$ 的在区间 $[0, 1000]$ 上的最小值。答案精确到小数点后四位, 四舍五入。

输入输出样例 1

输入 1

2

1

2 0 0

2

2 0 0

2 -4 2

输出 1

0.0000

0.5000

说明/提示

对于 50% 的数据, $n \leq 100$ 。

对于 100% 的数据, $T < 10$, $n \leq 10^4$, $0 \leq a \leq 100$, $|b| \leq 5 \times 10^3$, $|c| \leq 5 \times 10^3$ 。

```

1 def cal(cons, x):
2     ans = -float('inf')
3     for a, b, c in cons:
4         ans = max(ans, a * x ** 2 + b * x + c)
5     return ans
6
7 eps = 1e-9
8 T = int(input())
9 for t in range(T):
10    n = int(input())
11    cons = []
12    for i in range(n):
13        a, b, c = map(float, input().split())
14        cons.append([a, b, c])
15    l = 0
16    r = 1000
17    while r - l > eps:
18        margin = (r - l) / 3.0
19        mid1 = l + margin
20        mid2 = r - margin
21        f1 = cal(cons, mid1)
22        f2 = cal(cons, mid2)
23        if f1 <= f2:
24            r = mid2
25        else:
26            l = mid1
27    ans = cal(cons, l)
28    print(format(ans, '.4f'))

```

2.4 排序与排列