

*assignment*D 1

*Assignment #D : Mock Exam*下元节

Updated 1729 GMT+8 Dec 4, 2025

2025 fall, Comptled by 顾桂榕 基础医学院



顾桂榕 医学预科办

说明:

1. Dec月考: AC2。考试题目都在“题库（包括计概、数算题目）”里面，按照数字题号能找到，可以重新提交。作业中提交自己最满意版本的代码和截图。
2. 解题与记录: 对于每一个题目，请提供其解题思路（可选），并附上使用 *Python*或*C++*编写的源代码（确保已在*OpenJudge*, *Codeforces*, *LeetCode*等平台上获得*Accepted*）。请将这些信息连同显示“*Accepted*”的截图一起填写到下方的作业模板中。（推荐使用*Typora* <https://typoraio.cn> 进行编辑，当然你也可以选择*Word*。）无论题目是否已通过，请标明每个题目大致花费的时间。
3. 提交安排: 提交时，请首先上传*PDF*格式的文件，并将*.md*或*.doc*格式的文件作为附件上传至右侧的“作业评论”区。确保你的*Canvas*账户有一个清晰可见的本人头像，提交的文件为*PDF*格式，并且“作业评论”区包含上传的*.md*或*.doc*附件。
4. 延迟提交: 如果你预计无法在截止日期前提交作业，请提前告知具体原因。这有助于我们了解情况并可能为你提供适当的延期或其他帮助。

请按照上述指导认真准备和提交作业，以保证顺利完成课程要求。

1. 题目

E29945:神秘数字的宇宙旅行

implementation, <http://cs101.openjudge.cn/practice/29945>

思路：

代码

```
n = int(input())
while True:
    if n==1:
        print('End')
        break
    else:
        if n % 2 == 0:
            n_1 = n // 2
            print(f'{n}/2={n_1}')
            n = n_1
        else:
            n_1 = n * 3 + 1
            print(f'{n}*3+1={n_1}')
            n = n_1
```

代码运行截图 (至少包含有 "Accepted")

#51132005提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
n = int(input())
while True:
    if n==1:
        print('End')
        break
    else:
        if n % 2 == 0:
            n_1 = n // 2
            print(f'{n}/2={n_1}')
            n = n_1
        else:
            n_1 = n * 3 + 1
            print(f'{n}*3+1={n_1}')
            n = n_1
```

基本信息

#: 51132005
题目: E29945
提交人: R.
内存: 3632kB
时间: 24ms
语言: Python3
提交时间: 2025-12-04 15:19:05

E29946:删数问题

monotonic stack, greedy, <http://cs101.openjudge.cn/practice/29946>

思路：

代码

```
n = input().strip() # 读取字符串并去除可能的空格
k = int(input())
nums = list(map(int, str(n)))

ans = []
remain = len(nums) - k # 需要保留的数字个数

start = 0
# 每次在可选范围内选择最小的数字
for i in range(remain):
    end = k + i + 1 # 保证后面有足够的数字可以选择
    min_val = nums[start]
    min_index = start

    # 在可选范围内找到最小值及其索引
    for j in range(start + 1, end):
        if nums[j] < min_val:
            min_val = nums[j]
            min_index = j

    ans.append(min_val)
    start = min_index + 1 # 下一次从最小值的下一位开始

# 将结果转换为整数（去除前导零）
result = 0
for num in ans:
    result = result * 10 + num

print(result)
```

代码运行截图 (至少包含有 "Accepted")

#51134648提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
n = input().strip() # 读取字符串并去除可能的空格
k = int(input())
nums = list(map(int, str(n)))

if k >= len(nums): # 如果删除的数字个数大于等于总长度
    print(0)
else:
    ans = []
    remain = len(nums) - k # 需要保留的数字个数

    start = 0
    # 每次在可选范围内选择最小的数字
    for i in range(remain):
        end = k + i + 1 # 保证后面有足够的数字可以选择
        min_val = nums[start]
        min_index = start

        # 在可选范围内找到最小值及其索引
        for j in range(start + 1, end):
            if nums[j] < min_val:
                min_val = nums[j]
                min_index = j

        ans.append(min_val)
        start = min_index + 1 # 下一次从最小值的下一位开始

    # 将结果转换为整数 (去除前导零)
    result = 0
    for num in ans:
        result = result * 10 + num

    print(result)
```

基本信息

#: 51134648
题目: E29946
提交人: R.
内存: 3688kB
时间: 24ms
语言: Python3
提交时间: 2025-12-04 16:58:49

E30091:缺德的图书馆管理员

greedy, <http://cs101.openjudge.cn/practice/30091>

思路:

代码

```
L = int(input())
N = int(input())
index = list(map(int, input().split()))
min_ans = 0
max_ans = 0
for i in range(N):
    min_ans=max(min(index[i],L+1-index[i]),min_ans)
    max_ans=max(max(index[i],L+1-index[i]),max_ans)
print(min_ans,max_ans)
```

代码运行截图 (至少包含有 "Accepted")

#51169607 提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
L = int(input())
N = int(input())
index = list(map(int, input().split()))
min_ans = 0
max_ans = 0
for i in range(N):
    min_ans=max(min(index[i],L+1-index[i]),min_ans)
    max_ans=max(max(index[i],L+1-index[i]),max_ans)
print(min_ans,max_ans)
```

基本信息

#: 51169607
题目: 30091
提交人: R.
内存: 4040kB
时间: 28ms
语言: Python3
提交时间: 2025-12-07 09:14:12

©2002-2022 POJ 京ICP备20010980号-1

English 帮助 关于

M27371:Playfair密码

simulation, string, matrix, <http://cs101.openjudge.cn/practice/27371>

思路:

代码

```
key = input()
alphabet = 'abcdefghijklmnopqrstuvwxyz' # 25个字母, 没有j

# 初始化矩阵
matrix = [[''] * 5 for _ in range(5)]
index_list = {}
used = set()

# 填充矩阵
x, y = 0, 0
for char in key:
    if char == 'j': # 'j' 与 'i' 合并
        char = 'i'
    if char not in used:
        matrix[x][y] = char
        index_list[char] = (x, y)
        used.add(char)
        if y == 4:
            x += 1
            y = 0
    else:
```

```
y += 1

for char in alphabet:
    if char not in used:
        matrix[x][y] = char
        index_list[char] = (x, y)
        used.add(char)
    if y == 4:
        x += 1
        y = 0
    else:
        y += 1

n = int(input())
for _ in range(n):
    mingwen = list(input().strip())

    # 预处理: 将所有j转换为i
    for i in range(len(mingwen)):
        if mingwen[i] == 'j':
            mingwen[i] = 'i'

    # 分组处理
    pairs = []
    i = 0
    while i < len(mingwen):
        if i == len(mingwen) - 1:  # 最后一个字符
            if mingwen[i] != 'x':
                pairs.append((mingwen[i], 'x'))
            else:
                pairs.append((mingwen[i], 'q'))
            break

        # 检查相邻字符是否相同
        if mingwen[i] == mingwen[i + 1]:
            # 相同, 需要在中间插入填充字符
            if mingwen[i] != 'x':
                pairs.append((mingwen[i], 'x'))
            else:
                pairs.append((mingwen[i], 'q'))
            i += 1  # 只前进一个位置, 因为插入的字符占用了位置
```

```
else:  
    # 不同，正常分组  
    pairs.append((mingwen[i], mingwen[i + 1]))  
    i += 2  
  
# 加密  
ans = []  
for element1, element2 in pairs:  
    row1, col1 = index_list[element1]  
    row2, col2 = index_list[element2]  
  
    if row1 == row2: # 同一行  
        new_col1 = (col1 + 1) % 5  
        new_col2 = (col2 + 1) % 5  
        ans.append(matrix[row1][new_col1])  
        ans.append(matrix[row2][new_col2])  
    elif col1 == col2: # 同一列  
        new_row1 = (row1 + 1) % 5  
        new_row2 = (row2 + 1) % 5  
        ans.append(matrix[new_row1][col1])  
        ans.append(matrix[new_row2][col2])  
    else: # 不同行不同列  
        ans.append(matrix[row1][col2])  
        ans.append(matrix[row2][col1])  
  
print(''.join(ans))
```

代码运行截图 (至少包含有 "Accepted")

状态: Accepted

源代码

```
key = input()
alphabet = 'abcdefghijklmnopqrstuvwxyz' # 25个字母, 没有j

# 初始化矩阵
matrix = [[''] * 5 for _ in range(5)]
index_list = {}
used = set()

# 填充矩阵
x, y = 0, 0
for char in key:
    if char == 'j': # 'j' 与 'i' 合并
        char = 'i'
    if char not in used:
        matrix[x][y] = char
        index_list[char] = (x, y)
        used.add(char)
    if y == 4:
        x += 1
        y = 0
    else:
        y += 1

for char in alphabet:
    if char not in used:
        matrix[x][y] = char
        index_list[char] = (x, y)
        used.add(char)
    if y == 4:
        x += 1
        y = 0
    else:
        y += 1

n = int(input())
for _ in range(n):
    mingwen = list(input().strip())

    # 预处理: 将所有j转换为i
    for i in range(len(mingwen)):
        if mingwen[i] == 'j':
            mingwen[i] = 'i'

    # 分组处理
    pairs = []
    i = 0
    while i < len(mingwen):
        if i == len(mingwen) - 1: # 最后一个字符
            if mingwen[i] != 'x':
                pairs.append((mingwen[i], 'x'))
            else:
                pairs.append((mingwen[i], 'q'))
            break

        # 检查相邻字符是否相同
        if mingwen[i] == mingwen[i + 1]:
            # 相同, 需要在中间插入填充字符
            if mingwen[i] != 'x':
                pairs.append((mingwen[i], 'x'))
            else:
                pairs.append((mingwen[i], 'q'))
        else:
            i += 1 # 只前进一个位置, 因为插入的字符占用了位置
        else:
            # 不同, 正常分组
            pairs.append((mingwen[i], mingwen[i + 1]))
            i += 2

    # 加密
    ans = []
    for element1, element2 in pairs:
        row1, col1 = index_list[element1]
        row2, col2 = index_list[element2]
```

基本信息

#: 51210242
题目: 27371
提交人: R.
内存: 3812kB
时间: 33ms
语言: Python3
提交时间: 2025-12-09 17:56:37

```

if row1 == row2: # 同一行
    new_col1 = (col1 + 1) % 5
    new_col2 = (col2 + 1) % 5
    ans.append(matrix[row1][new_col1])
    ans.append(matrix[row2][new_col2])
elif col1 == col2: # 同一列
    new_row1 = (row1 + 1) % 5
    new_row2 = (row2 + 1) % 5
    ans.append(matrix[new_row1][col1])
    ans.append(matrix[new_row2][col2])
else: # 不同行不同列
    ans.append(matrix[row1][col2])
    ans.append(matrix[row2][col1])

print('\n'.join(ans))

```

©2002-2022 POJ 京ICP备20010980号-1

English 帮助 关于

T30201:旅行售货商问题

dp,dfs, <http://cs101.openjudge.cn/practice/30201>

思路：

#51197366提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```

import sys
from functools import lru_cache

def main():
    n = int(sys.stdin.readline().strip())
    cost = []
    for _ in range(n):
        row = list(map(int,sys.stdin.readline().split()))
        cost.append(row)

@lru_cache(maxsize=None)
def dfs(mask, current):
    if mask==(1<<n)-1:
        return cost[current][0]

    res = float('inf')
    for next_city in range(n):
        if mask & (1<<next_city):
            continue
        new_mask = mask | (1<<next_city)
        res = min(res,cost[current][next_city]+dfs(new_mask,next_ci
    return res

ans = dfs(1,0)
print(ans)

if __name__ == '__main__':
    main()

```

基本信息

#: 51197366
 题目: 30201
 提交人: R.
 内存: 186464kB
 时间: 6768ms
 语言: Python3
 提交时间: 2025-12-08 20:12:17

代码

```

import sys
from functools import lru_cache

def main():
    n = int(sys.stdin.readline().strip())

```

```

cost = []
for _ in range(n):
    row = list(map(int,sys.stdin.readline().split()))
    cost.append(row)

@lru_cache(maxsize=None)
def dfs(mask,current):
    if mask==(1<<n)-1:
        return cost[current][0]

    res = float('inf')
    for next_city in range(n):
        if mask & (1<<next_city):
            continue
            new_mask = mask | (1<<next_city)
            res = min(res,cost[current]
[next_city]+dfs(new_mask,next_city)))
    return res
ans = dfs(1,0)
print(ans)
if __name__ == '__main__':
    main()

```

代码运行截图 (至少包含有 "Accepted")

T30204: 小P的LLM推理加速

greedy, <http://cs101.openjudge.cn/practice/30204>

思路：

正确的策略：我们将 x_i 有序排列，假设 m 是足够大的（不足够大的话策略显然），现在我们用 bisectleft 插入 $\frac{\min(x_i+y_i)}{2}$ ，它变成了第 k 个数。那么最优解一定是取 x_i 的前 n 个数，其中 n 只可能等于 $k-2, k-1, k$ 。

证明： $x_k + x_{k+1} \geq \min(x_i + y_i) > x_{k-1} + x_{k-2}$ ，所以 $n > k$ 和 $n < k-2$ 都不可能更优。

代码

```

n,m = map(int,input().split())
cold_list = [];res = 0
min_circle = float('inf')
for i in range(n):

```

```

cold,hot = map(int,input().split())
cold_list.append(cold)
min_circle = min(cold+hot,min_circle)
sorted_cold_list=sorted(cold_list)
prefix = [0]*(n+1)
for j in range(1,n+1):
    prefix[j]=prefix[j-1]+sorted_cold_list[j-1]
ans = 0
for d in range(n+1):
    if prefix[d]>m:
        break    remain = m-prefix[d]
circle = remain//min_circle
res=d+circle*2
if res>ans:
    ans=res
print(ans)

```

代码运行截图 (至少包含有 "Accepted")

#51197014 提交状态

[查看](#) [提交](#) [统计](#) [提问](#)

状态: Accepted

源代码

```

n,m = map(int,input().split())
cold_list = [];res = 0
min_circle = float('inf')
for i in range(n):
    cold,hot = map(int,input().split())
    cold_list.append(cold)
    min_circle = min(cold+hot,min_circle)
sorted_cold_list=sorted(cold_list)
prefix = [0]*(n+1)
for j in range(1,n+1):
    prefix[j]=prefix[j-1]+sorted_cold_list[j-1]
ans = 0
for d in range(n+1):
    if prefix[d]>m:
        break
    remain = m-prefix[d]
    circle = remain//min_circle
    res=d+circle*2
    if res>ans:
        ans=res
print(ans)

```

基本信息

#: 51197014
 题目: 30204
 提交人: R.
 内存: 15820kB
 时间: 271ms
 语言: Python3
 提交时间: 2025-12-08 19:57:07

2. 学习总结和收获

如果作业题目简单，有否额外练习题目，比如：OJ“计概2025fall每日选做”、CF、LeetCode、洛谷等网站题目。

状态: Accepted

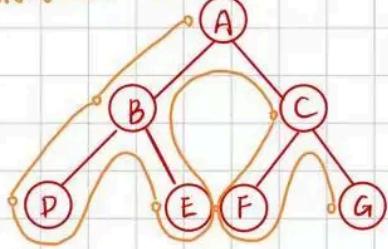
源代码

```
while True:
    try:
        N = int(input())
        battery_list = list(map(int, input().split()))
        max_battery = max(battery_list)
        sum_battery = sum(battery_list)
        if max_battery>sum_battery-max_battery:
            ans = sum_battery-max_battery
            print(f'{ans:.1f}')
        else:
            ans = sum(battery_list)/2
            print(f'{ans:.1f}')
    except EOFError:
        break
```

基本信息

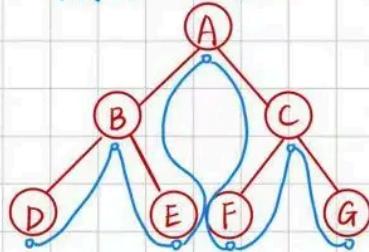
#: 51208674
题目: 03468
提交人: R.
内存: 3644kB
时间: 36ms
语言: Python3
提交时间: 2025-12-09 16:57:55

前序遍历 根左右



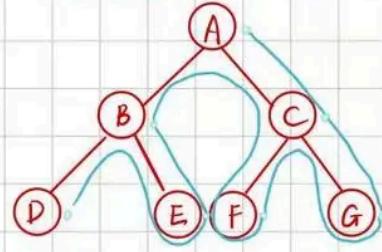
$A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F \rightarrow G$

中序遍历：左根右



$D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$

后序遍历 左右根



$D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$

层序遍历 (类 bfs)

deque



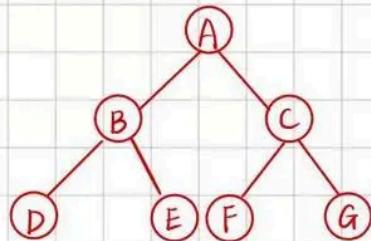
$A \rightarrow \text{left}, \text{right}$



$A \ B \ C \rightarrow \text{left}, \text{right}$

$A \ B \ C \ D \ E \ F \ G$

$\text{left}, \text{right} = \text{None}$



解密 decrypt

解法一：dp (递归)

```

def decrypt(cypher):
    n = len(cypher)
    if n<=1:
        return cypher

    left_side = decrypt(cypher[1:1+(n-1)//2])
    right_side = decrypt(cypher[1+(n-1)//2:])

    return left_side+cypher[0]+right_side

cypher_text = input()
print(decrypt(cypher_text))

```

解法二： *iterative(迭代)-deque*

```

def decrypt_iterative(cypher_text):
    n = len(cypher_text)
    res = [''] * n
    from collections import deque
    queue = deque()
    queue.append((0,n-1,0,n-1))
    while queue:
        cypher_start, cypher_end, plain_start, plain_end =
queue.popleft()
        if cypher_end>cypher_start:
            continue
            plain_mid = plain_start+(plain_end-
plain_start)//2
            res[plain_mid]=cypher_text[cypher_start]
            left_size = plain_mid-plain_start
            if left_size>0:
                queue.append((cypher_start+1,cypher_start+left_size,plain_start,plain_
mid-1))
                    right_size = plain_end-plain_mid
                    if right_size>0:
                        queue.append((cypher_start+left_size+1,cypher_end,plain_mid+1,plain_
end))
    return ''.join(res)

```

```
    cypher_text = input()
print(decrypt_iterative(cypher_text))
```

背包问题：只能放 n 个物体且每个物体只能用一次情况下的最大价值

将二维列表变为一维列表时需倒序处理

设 DP 状态 $f_{i,j}$ 为在只能放前 i 个物品的情况下，容量为 j 的背包所能达到的最大总价值。

考虑转移。假设当前已经处理好了前 $i-1$ 个物品的所有状态，那么对于第 i 个物品，当其不放入背包时，背包的剩余容量不变，背包中物品的总价值也不变，故这种情况的最大价值为 $f_{i-1,j}$ ；当其放入背包时，背包的剩余容量会减小 w_i ，背包中物品的总价值会增大 v_i ，故这种情况的最大价值为 $f_{i-1,j-w_i} + v_i$ 。

由此可以得出状态转移方程：

$$f_{i,j} = \max(f_{i-1,j}, f_{i-1,j-w_i} + v_i)$$

这里如果直接采用二维数组对状态进行记录，会出现 MLE 。可以考虑改用滚动数组的形式来优化。

由于对 f_i 有影响的只有 f_{i-1} ，可以去掉第一维，直接用 f_i 来表示处理到当前物品时背包容量为 i 的最大价值，得出以下方程：

$$f_j = \max(f_{j-w_i} + v_i)$$

务必牢记并理解这个转移方程，因为大部分背包问题的转移方程都是在此基础上推导出来的。

实现

还有一点需要注意的是，很容易写出这样的 **错误核心代码**：

```
for i in range(1, n + 1):
    for l in range(0, w - w[i] + 1):
        f[l + w[i]] = max(f[l] + v[i], f[l + w[i]])
由 f[i][l + w[i]] = max(max(f[i - 1][l + w[i]], f[i - 1][l] +
w[i]),
f[i][l + w[i]]) 简化而来
```

这段代码哪里错了呢？枚举顺序错了。

仔细观察代码可以发现：对于当前处理的物品 i 和当前状态 $f_{\{i,j\}}$ ，在 $j \geq w_i$ 时，是会被 $f_{i,j-w_i}$ 所影响的。这就相当于物品 i 可以多次被放入背包，与题意不符。（事实上，这正是完全背包问题的解法）

为了避免这种情况发生，我们可以改变枚举的顺序，从 \mathcal{W} 枚举到 w_i ，这样就不会出现上述的错误，因为 $f_{i,j}$ 总是在 $f_{i,j-w_i}$ 前被更新。

因此实际核心代码为

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

Bisect

bisect 是 *Python* 内置的二分查找模块，用于在有序列表中快速查找和插入元素。让我详细解释它的用法：

主要函数

1. *bisect_left(a, x, lo=0, hi=len(a))*

在有序列表 a 中查找 x ，返回第一个 大于等于 x 的元素位置

2. *bisect_right(a, x, lo=0, hi=len(a))* 或 *bisect(a, x, lo=0, hi=len(a))*

在有序列表 a 中查找 x ，返回第一个 大于 x 的元素位置

3. *insort_left(a, x, lo=0, hi=len(a))*

将 x 插入到有序列表 a 中，保持有序，如果存在相等元素，插入到左边

4. *insort_right(a, x, lo=0, hi=len(a))* 或 *insort(a, x, lo=0, hi=len(a))*

将 x 插入到有序列表 a 中，保持有序，如果存在相等元素，插入到右边

示例演示

```
import bisect
```

```
# 示例列表
arr = [1, 3, 3, 5, 7, 9, 11]

# 1. bisect_left - 找第一个 >= 目标值的位置
print("bisect_left 示例:")
print(f"bisect_left(arr, 3) = {bisect.bisect_left(arr, 3)}") # 返回
1
print(f"bisect_left(arr, 4) = {bisect.bisect_left(arr, 4)}") # 返回
3 (第一个>=4的位置)
print(f"bisect_left(arr, 0) = {bisect.bisect_left(arr, 0)}") # 返回
0
print(f"bisect_left(arr, 12) = {bisect.bisect_left(arr, 12)}") # 返回7 (超出范围)

# 2. bisect_right - 找第一个 > 目标值的位置
print("\nbisect_right 示例:")
print(f"bisect_right(arr, 3) = {bisect.bisect_right(arr, 3)}") # 返回
3 (第一个>3的位置)
print(f"bisect_right(arr, 5) = {bisect.bisect_right(arr, 5)}") # 返回
4
print(f"bisect_right(arr, 11) = {bisect.bisect_right(arr, 11)}") # 返回7

# 3. insort - 插入元素并保持有序
print("\ninsort 示例:")
arr1 = [1, 3, 5, 7]
bisect.insort(arr1, 4)
print(f"插入4后: {arr1}") # [1, 3, 4, 5, 7]

arr2 = [1, 3, 3, 5, 7]
bisect.insort_left(arr2, 3) # 插入到左边
print(f"左边插入3后: {arr2}") # [1, 3, 3, 3, 5, 7]

bisect.insort_right(arr2, 3) # 插入到右边
print(f"右边插入3后: {arr2}") # [1, 3, 3, 3, 3, 5, 7]
```

实际应用场景

1. 成绩等级划分

```
import bisect

def get_grade(score, breakpoints=[60, 70, 80, 90], grades='FDCBA'):
    """根据分数返回等级"""
    i = bisect.bisect_right(breakpoints, score)
    return grades[i]

scores = [45, 65, 75, 85, 95]
for score in scores:
    print(f"分数 {score}: 等级 {get_grade(score)}")
# 输出: F, D, C, B, A
```

2. 维护有序列表

```
import bisect

# 动态维护有序列表
sorted_list = []
data = [5, 2, 8, 1, 9, 3]

for num in data:
    bisect.insort(sorted_list, num)
    print(f"插入 {num}: {sorted_list}")

# 输出:
# 插入 5: [5]
# 插入 2: [2, 5]
# 插入 8: [2, 5, 8]
# 插入 1: [1, 2, 5, 8]
# 插入 9: [1, 2, 5, 8, 9]
# 插入 3: [1, 2, 3, 5, 8, 9]
```

3. 查找插入位置

```
import bisect

# 在有序数组中查找插入位置
arr = [1, 3, 5, 7, 9]
target = 6
```

```
pos = bisect.bisect_left(arr, target)
print(f"{target} 应该插入到位置 {pos}") # 输出: 6 应该插入到位置 3

# 插入并验证
arr.insert(pos, target)
print(f"插入后数组: {arr}") # [1, 3, 5, 6, 7, 9]
```

参数说明

- a : 有序列表
- x : 要查找或插入的值
- lo : 查找范围的起始索引 (默认 0)
- hi : 查找范围的结束索引 (默认 $\text{len}(a)$)

```
# 指定查找范围
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
# 只在索引2到6之间查找 (元素3,4,5,6)
pos = bisect.bisect_left(arr, 5, lo=2, hi=6)
print(pos) # 输出4
```

性能优势

- 时间复杂度: $O(\log n)$ - 比线性查找 $O(n)$ 快很多
- 空间复杂度: $O(1)$ - 不需要额外空间

```
import time
import bisect

# 性能对比
large_list = list(range(1000000))

# 二分查找
start = time.time()
pos1 = bisect.bisect_left(large_list, 500000)
time1 = time.time() - start

# 线性查找
start = time.time()
pos2 = large_list.index(500000)
```

```
time2 = time.time() - start

print(f"二分查找: {time1:.6f}秒")
print(f"线性查找: {time2:.6f}秒")
```

bisect 模块在处理有序数据时非常高效，特别适合需要频繁查找和插入的场景。

拦截导弹

```
from bisect import bisect_right

def min_testers_needed(scores):
    scores.reverse() # 反转序列以找到最长下降子序列的长度
    lis = [] # 用于存储最长上升子序列

    for score in scores:
        pos = bisect_right(lis, score)
        if pos < len(lis):
            lis[pos] = score
        else:
            lis.append(score)

    return len(lis)

N = int(input())
scores = list(map(int, input().split()))

result = min_testers_needed(scores)
print(result)
```

用*bisect*:

对于一个新数，我可以将它插入到哪里？

如果已在范围内，那么替换，不影响*lis*长度；

不在则添加