Assignment #D: May月考

Updated 1654 GMT+8 May 8, 2024

2024 spring, Complied by 胡登科

说明：

1）请把每个题目解题思路（可选），源码Python, 或者C++（已经在Codeforces/Openjudge上AC），截图（包含Accepted），填写到下面作业模版中（推荐使用 typora https://typoraio.cn ，或者用word）。AC 或者没有AC，都请标上每个题目大致花费时间。

2）提交时候先提交pdf文件，再把md或者doc文件上传到右侧“作业评论”。Canvas需要有同学清晰头像、提交文件有pdf、"作业评论"区有上传的md或者doc附件。

3）如果不能在截止前提交作业，请写明原因。

编程环境

（请改为同学的操作系统、编程环境等）

操作系统：macOS Ventura 13.4.1 (c)

Python编程环境：Spyder IDE 5.2.2, PyCharm 2023.1.4 (Professional Edition)

C/C++编程环境：Mac terminal vi (version 9.0.1424), g++/gcc (Apple clang version 14.0.3, clang-1403.0.22.14.1)

1. 题目

02808: 校门外的树

http://cs101.openjudge.cn/practice/02808/

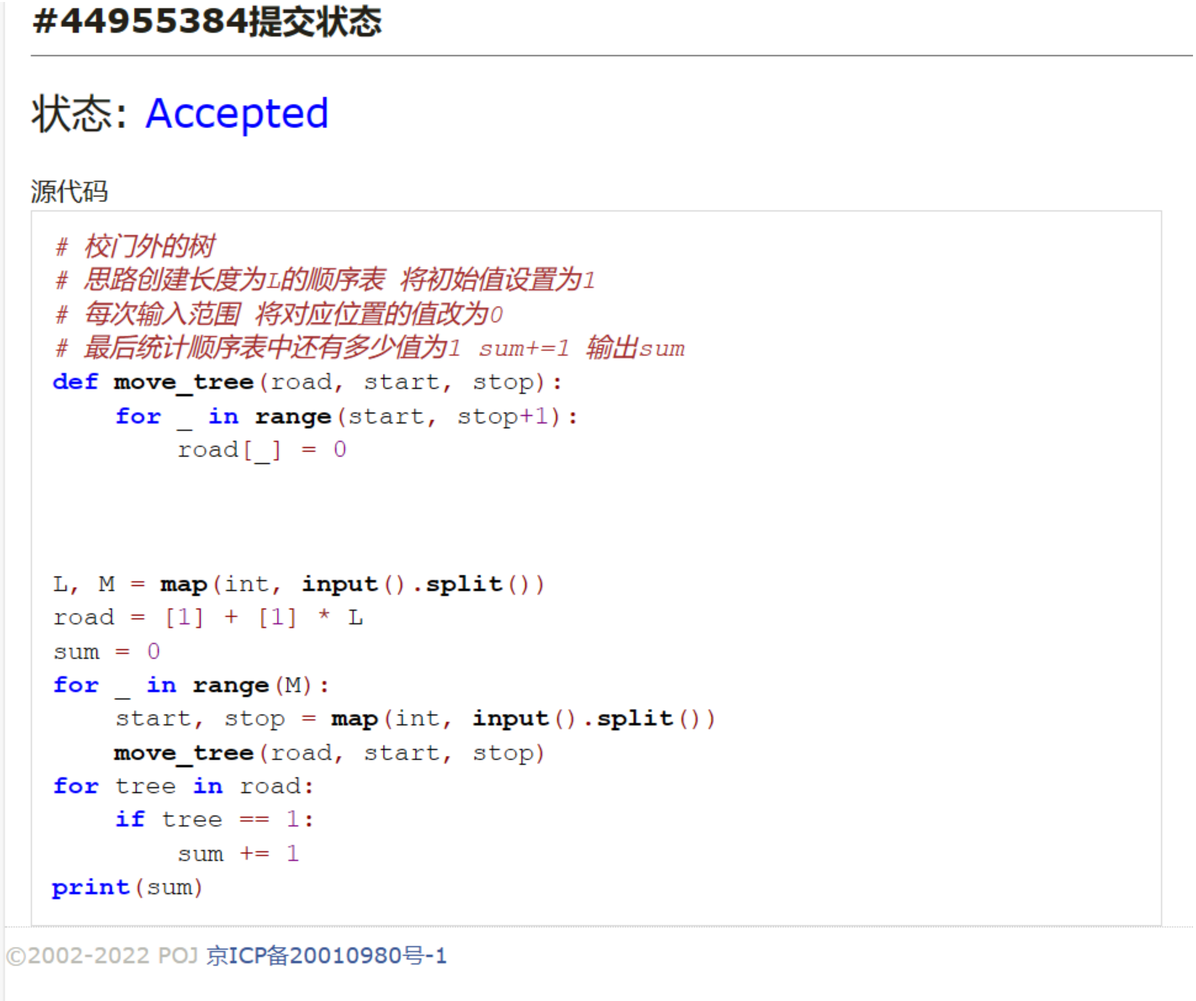
思路：简单

代码

```
#
# 校门外的树
# 思路创建长度为L的顺序表  将初始值设置为1
# 每次输入范围  将对应位置的值改为0
# 最后统计顺序表中还有多少值为1 sum+=1 输出sum
def move_tree(road, start, stop):
    for _ in range(start, stop+1):
        road[_] = 0
```

```
L, M = map(int, input().split())
road = [1] + [1] * L
sum = 0
for _ in range(M):
    start, stop = map(int, input().split())
    move_tree(road, start, stop)
for tree in road:
    if tree == 1:
        sum += 1
print(sum)
```

代码运行截图（至少包含有"Accepted"）



#44955384提交状态

状态: Accepted

源代码

```
# 校门外的树
# 思路创建长度为L的顺序表 将初始值设置为1
# 每次输入范围 将对应位置的值改为0
# 最后统计顺序表中还有多少值为1 sum+=1 输出sum
def move_tree(road, start, stop):
    for _ in range(start, stop+1):
        road[_] = 0



L, M = map(int, input().split())
road = [1] + [1] * L
sum = 0
for _ in range(M):
    start, stop = map(int, input().split())
    move_tree(road, start, stop)
for tree in road:
    if tree == 1:
        sum += 1
print(sum)
```

©2002-2022 POJ 京ICP备20010980号-1

20449: 是否被5整除

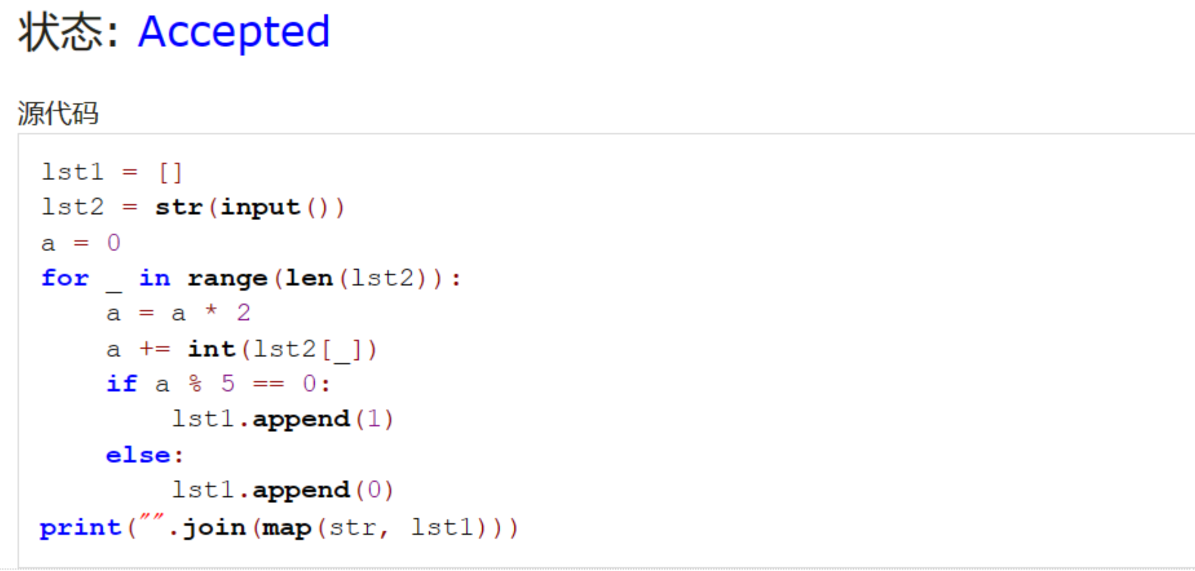
http://cs101.openjudge.cn/practice/20449/

思路：简单

代码

```
#
lst1 = []
lst2 = str(input())
a = 0
for _ in range(len(lst2)):
    a = a * 2
    a += int(lst2[_])
    if a % 5 == 0:
        lst1.append(1)
    else:
        lst1.append(0)
print("".join(map(str, lst1)))
```

代码运行截图 （至少包含有"Accepted"）



状态: Accepted

源代码

```
lst1 = []
lst2 = str(input())
a = 0
for _ in range(len(lst2)):
    a = a * 2
    a += int(lst2[_])
    if a % 5 == 0:
        lst1.append(1)
    else:
        lst1.append(0)
print("".join(map(str, lst1)))
```

01258: Agri-Net

http://cs101.openjudge.cn/practice/01258/

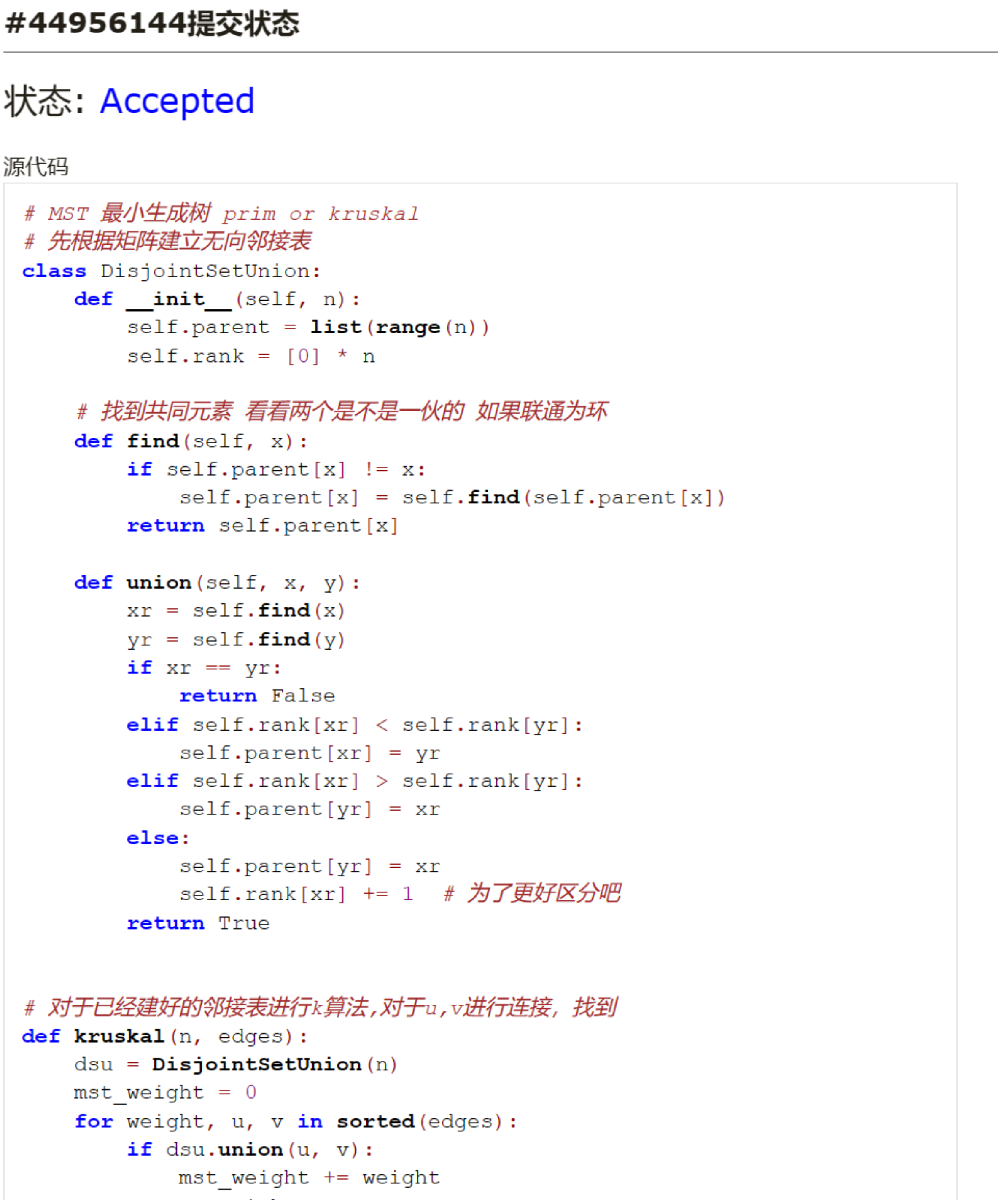
思路：kruskal思路

代码

```
# MST 最小生成树 prim or kruskal
# 先根据矩阵建立无向邻接表
class DisjointSetUnion:
    def __init__(self, n):
        self.parent = list(range(n))
        self.rank = [0] * n

    # 找到共同元素 看看两个是不是一伙的 如果联通为环
```

```python
    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        xr = self.find(x)
        yr = self.find(y)
        if xr == yr:
            return False
        elif self.rank[xr] < self.rank[yr]:
            self.parent[xr] = yr
        elif self.rank[xr] > self.rank[yr]:
            self.parent[yr] = xr
        else:
            self.parent[yr] = xr
            self.rank[xr] += 1  # 为了更好区分吧
        return True


# 对于已经建好的邻接表进行k算法,对于u,v进行连接，找到
def kruskal(n, edges):
    dsu = DisjointSetUnion(n)
    mst_weight = 0
    for weight, u, v in sorted(edges):
        if dsu.union(u, v):
            mst_weight += weight
    return mst_weight


# 开始将矩阵转化为邻接表

def main():
    while True:
        try:
            n = int(input().strip())
            edges = []
            for i in range(n):
                # Since the input lines may continue onto others, we read them all at once
                row = list(map(int, input().split()))
                for j in range(i + 1, n):
                    if row[j] != 0:  # No need to add edges with 0 weight
                        edges.append((row[j], i, j))
            print(kruskal(n, edges))
        except EOFError:  # Exit the loop when all test cases are processed
            break


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

代码运行截图 （AC代码截图，至少包含有"Accepted"）



#44956144提交状态

状态: Accepted

源代码

```
# MST 最小生成树 prim or kruskal
# 先根据矩阵建立无向邻接表
class DisjointSetUnion:
    def __init__(self, n):
        self.parent = list(range(n))
        self.rank = [0] * n

    # 找到共同元素 看看两个是不是一伙的 如果联通为环
    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        xr = self.find(x)
        yr = self.find(y)
        if xr == yr:
            return False
        elif self.rank[xr] < self.rank[yr]:
            self.parent[xr] = yr
        elif self.rank[xr] > self.rank[yr]:
            self.parent[yr] = xr
        else:
            self.parent[yr] = xr
            self.rank[xr] += 1   # 为了更好区分吧
        return True


# 对于已经建好的邻接表进行k算法,对于u,v进行连接, 找到
def kruskal(n, edges):
    dsu = DisjointSetUnion(n)
    mst_weight = 0
    for weight, u, v in sorted(edges):
        if dsu.union(u, v):
            mst_weight += weight
```

27635: 判断无向图是否连通有无回路(同23163)

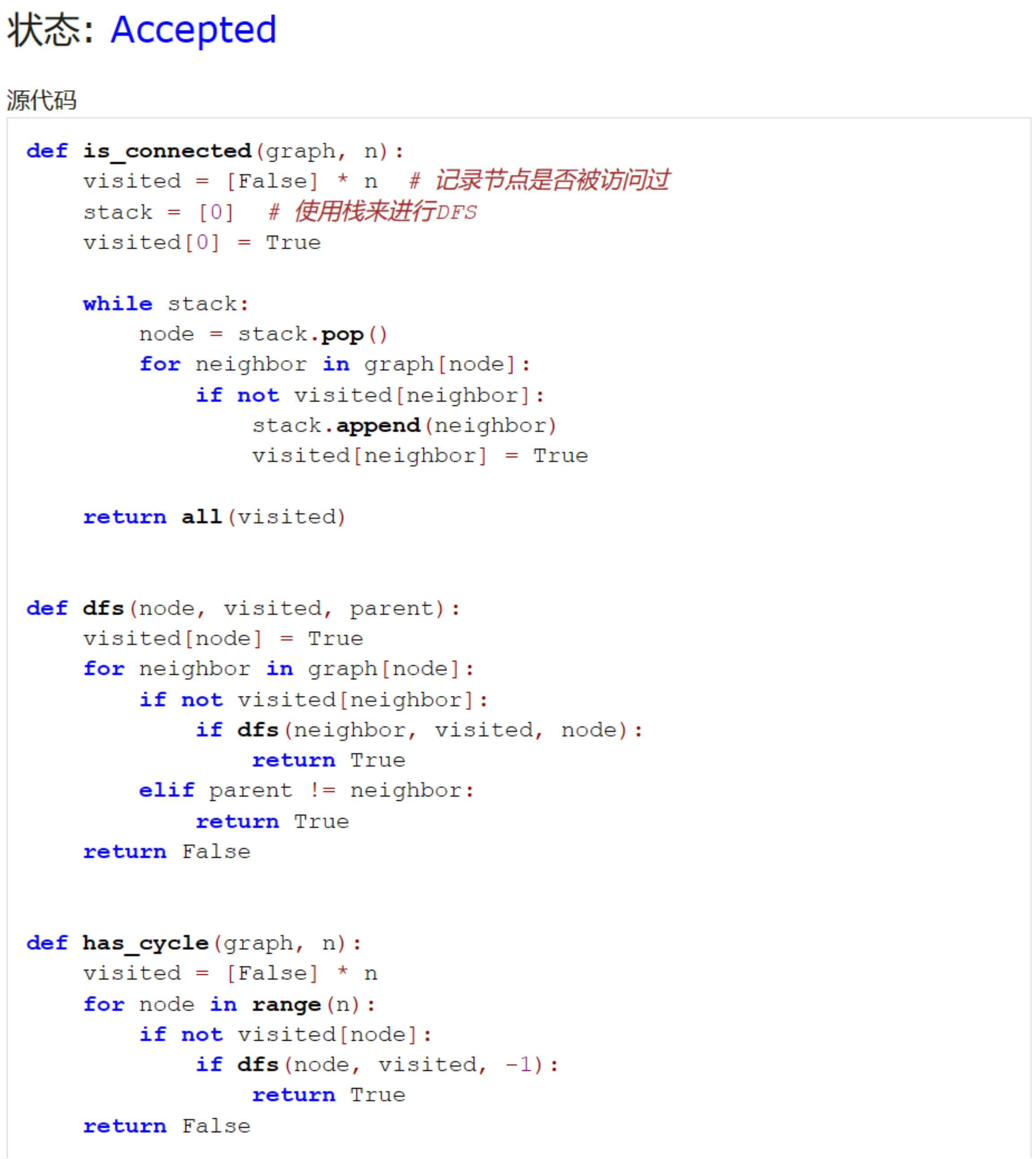
http://cs101.openjudge.cn/practice/27635/

思路：建图，然后dfs搜索是否联通。然后利用从不同点出发

代码

```
#
def is_connected(graph, n):
    visited = [False] * n  # 记录节点是否被访问过
    stack = [0]  # 使用栈来进行DFS
    visited[0] = True

    while stack:
        node = stack.pop()
        for neighbor in graph[node]:
            if not visited[neighbor]:
                stack.append(neighbor)
                visited[neighbor] = True

    return all(visited)

def dfs(node, visited, parent):
    visited[node] = True
    for neighbor in graph[node]:
        if not visited[neighbor]:
            if dfs(neighbor, visited, node):
                return True
        elif parent != neighbor:
            return True
    return False

def has_cycle(graph, n):
    visited = [False] * n
    for node in range(n):
        if not visited[node]:
            if dfs(node, visited, -1):
                return True
    return False


# 读取输入
n, m = map(int, input().split())
graph = [[] for _ in range(n)]
for _ in range(m):
    u, v = map(int, input().split())
    graph[u].append(v)
    graph[v].append(u)

# 判断连通性和回路
connected = is_connected(graph, n)
has_loop = has_cycle(graph, n)
print("connected:yes" if connected else "connected:no")
print("loop:yes" if has_loop else "loop:no")
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）



状态: Accepted

源代码

```python
def is_connected(graph, n):
    visited = [False] * n  # 记录节点是否被访问过
    stack = [0]  # 使用栈来进行DFS
    visited[0] = True

    while stack:
        node = stack.pop()
        for neighbor in graph[node]:
            if not visited[neighbor]:
                stack.append(neighbor)
                visited[neighbor] = True

    return all(visited)


def dfs(node, visited, parent):
    visited[node] = True
    for neighbor in graph[node]:
        if not visited[neighbor]:
            if dfs(neighbor, visited, node):
                return True
        elif parent != neighbor:
            return True
    return False


def has_cycle(graph, n):
    visited = [False] * n
    for node in range(n):
        if not visited[node]:
            if dfs(node, visited, -1):
                return True
    return False
```

27947: 动态中位数

http://cs101.openjudge.cn/practice/27947/

思路：

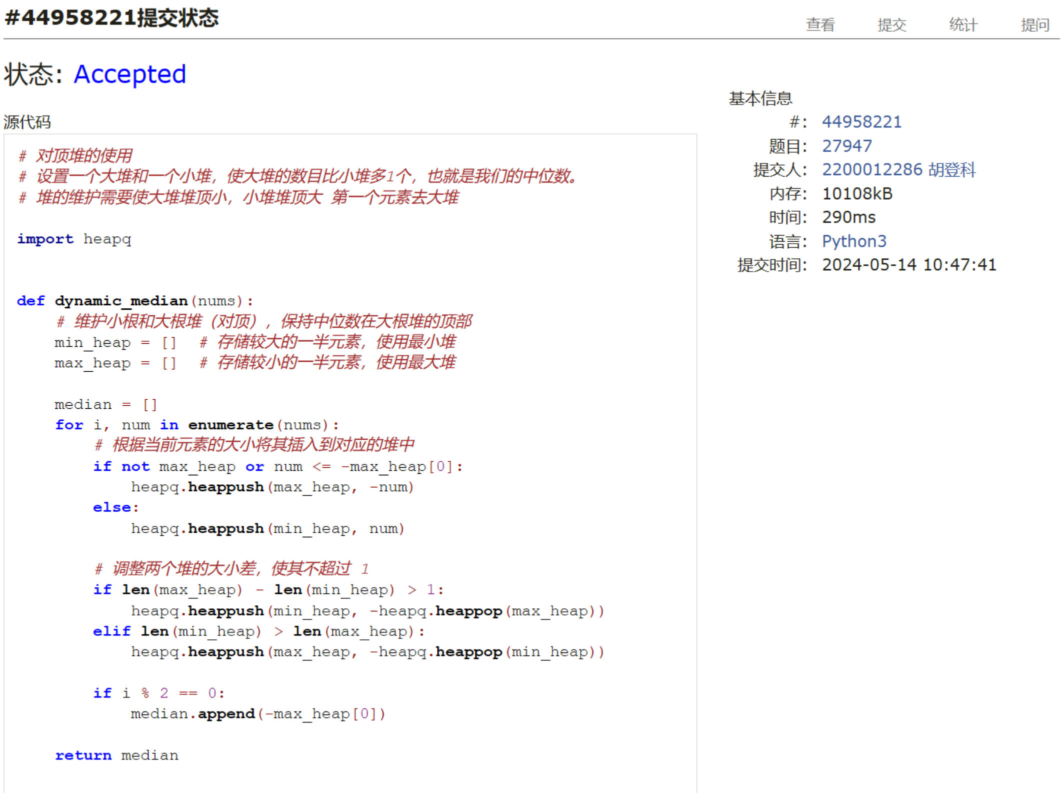
很久没复习堆了差不多快忘了。www

代码

```python
#
# 对顶堆的使用
# 设置一个大堆和一个小堆，使大堆的数目比小堆多1个，也就是我们的中位数。
# 堆的维护需要使大堆堆顶小，小堆堆顶大  第一个元素去大堆

```

```python
import heapq


def dynamic_median(nums):
    # 维护小根和大根堆（对顶），保持中位数在大根堆的顶部
    min_heap = []  # 存储较大的一半元素，使用最小堆
    max_heap = []  # 存储较小的一半元素，使用最大堆

    median = []
    for i, num in enumerate(nums):
        # 根据当前元素的大小将其插入到对应的堆中
        if not max_heap or num <= -max_heap[0]:
            heapq.heappush(max_heap, -num)
        else:
            heapq.heappush(min_heap, num)

        # 调整两个堆的大小差，使其不超过 1
        if len(max_heap) - len(min_heap) > 1:
            heapq.heappush(min_heap, -heapq.heappop(max_heap))
        elif len(min_heap) > len(max_heap):
            heapq.heappush(max_heap, -heapq.heappop(min_heap))

        if i % 2 == 0:
            median.append(-max_heap[0])

    return median


T = int(input())
for _ in range(T):
    # M = int(input())
    nums = list(map(int, input().split()))
    median = dynamic_median(nums)
    print(len(median))
    print(*median)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）



#44958221提交状态

状态: Accepted

源代码

```
# 对顶堆的使用
# 设置一个大堆和一个小堆，使大堆的数目比小堆多1个，也就是我们的中位数。
# 堆的维护需要使大堆堆顶小，小堆堆顶大 第一个元素去大堆

import heapq


def dynamic_median(nums):
    # 维护小根和大根堆（对顶），保持中位数在大根堆的顶部
    min_heap = []  # 存储较大的一半元素，使用最小堆
    max_heap = []  # 存储较小的一半元素，使用最大堆

    median = []
    for i, num in enumerate(nums):
        # 根据当前元素的大小将其插入到对应的堆中
        if not max_heap or num <= -max_heap[0]:
            heapq.heappush(max_heap, -num)
        else:
            heapq.heappush(min_heap, num)

        # 调整两个堆的大小差，使其不超过 1
        if len(max_heap) - len(min_heap) > 1:
            heapq.heappush(min_heap, -heapq.heappop(max_heap))
        elif len(min_heap) > len(max_heap):
            heapq.heappush(max_heap, -heapq.heappop(min_heap))

        if i % 2 == 0:
            median.append(-max_heap[0])

    return median
```

基本信息

#: 44958221
题目: 27947
提交人: 2200012286 胡登科
内存: 10108kB
时间: 290ms
语言: Python3
提交时间: 2024-05-14 10:47:41

28190: 奶牛排队

http://cs101.openjudge.cn/practice/28190/

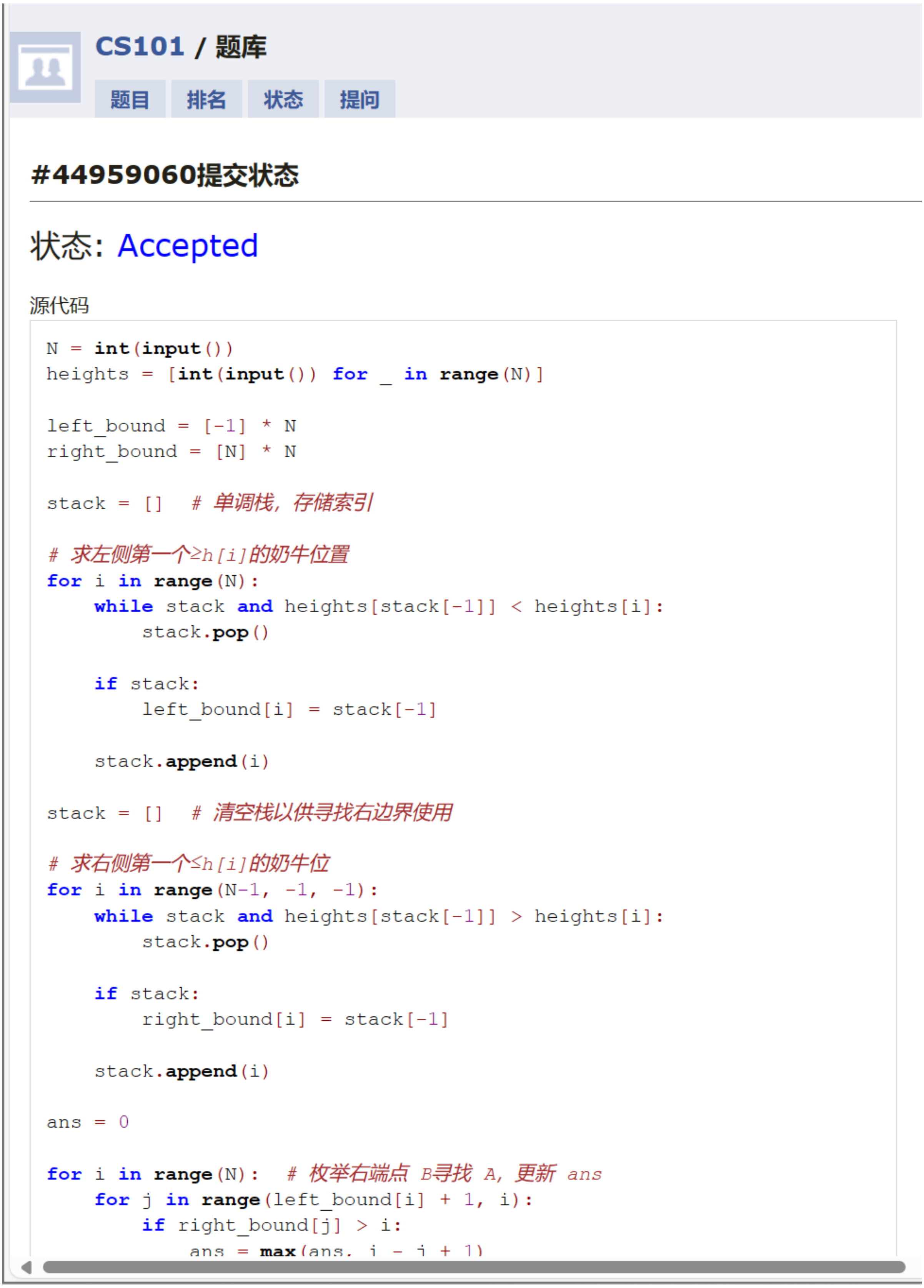
思路：单调栈的引用，也有一点忘了。

代码

```
#
N = int(input())
heights = [int(input()) for _ in range(N)]

left_bound = [-1] * N
right_bound = [N] * N

stack = []  # 单调栈，存储索引

# 求左侧第一个≥h[i]的奶牛位置
for i in range(N):
    while stack and heights[stack[-1]] < heights[i]:
        stack.pop()

    if stack:
        left_bound[i] = stack[-1]

    stack.append(i)

stack = []  # 清空栈以供寻找右边界使用
```

```
# 求右侧第一个≤h[i]的奶牛位
for i in range(N-1, -1, -1):
    while stack and heights[stack[-1]] > heights[i]:
        stack.pop()

    if stack:
        right_bound[i] = stack[-1]

    stack.append(i)

ans = 0

for i in range(N):  # 枚举右端点 B寻找 A，更新 ans
    for j in range(left_bound[i] + 1, i):
        if right_bound[j] > i:
            ans = max(ans, i - j + 1)
            break
print(ans)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）



#44959060提交状态

状态: Accepted

源代码

```
N = int(input())
heights = [int(input()) for _ in range(N)]

left_bound = [-1] * N
right_bound = [N] * N

stack = []  # 单调栈，存储索引

# 求左侧第一个≥h[i]的奶牛位置
for i in range(N):
    while stack and heights[stack[-1]] < heights[i]:
        stack.pop()

    if stack:
        left_bound[i] = stack[-1]

    stack.append(i)

stack = []  # 清空栈以供寻找右边界使用

# 求右侧第一个≤h[i]的奶牛位
for i in range(N-1, -1, -1):
    while stack and heights[stack[-1]] > heights[i]:
        stack.pop()

    if stack:
        right_bound[i] = stack[-1]

    stack.append(i)

ans = 0

for i in range(N):  # 枚举右端点 B寻找 A，更新 ans
    for j in range(left_bound[i] + 1, i):
        if right_bound[j] > i:
            ans = max(ans, i - j + 1)
```

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“2024spring每日选做”、CF、LeetCode、洛谷等网站题目。

本周学习了所有的数据结构的基础定义看书。感觉栈的题目有一点手生了，可以再练习一下。另外：感觉我需要更加关注一些笔试中的计算题，算错或者读题错误丢分严重。