Assignment #D：May月考

Updated 1654 GMT+8 May 8, 2024

2024 spring, Complied by 王申睿——物理学院

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 typorahttps://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：

代码

```
#

```

代码运行截图 （至少包含有"Accepted"）

20449: 是否被5整除

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

思路:

代码

```
#

```

代码运行截图 （至少包含有"Accepted"）

01258: Agri-Net

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

思路:

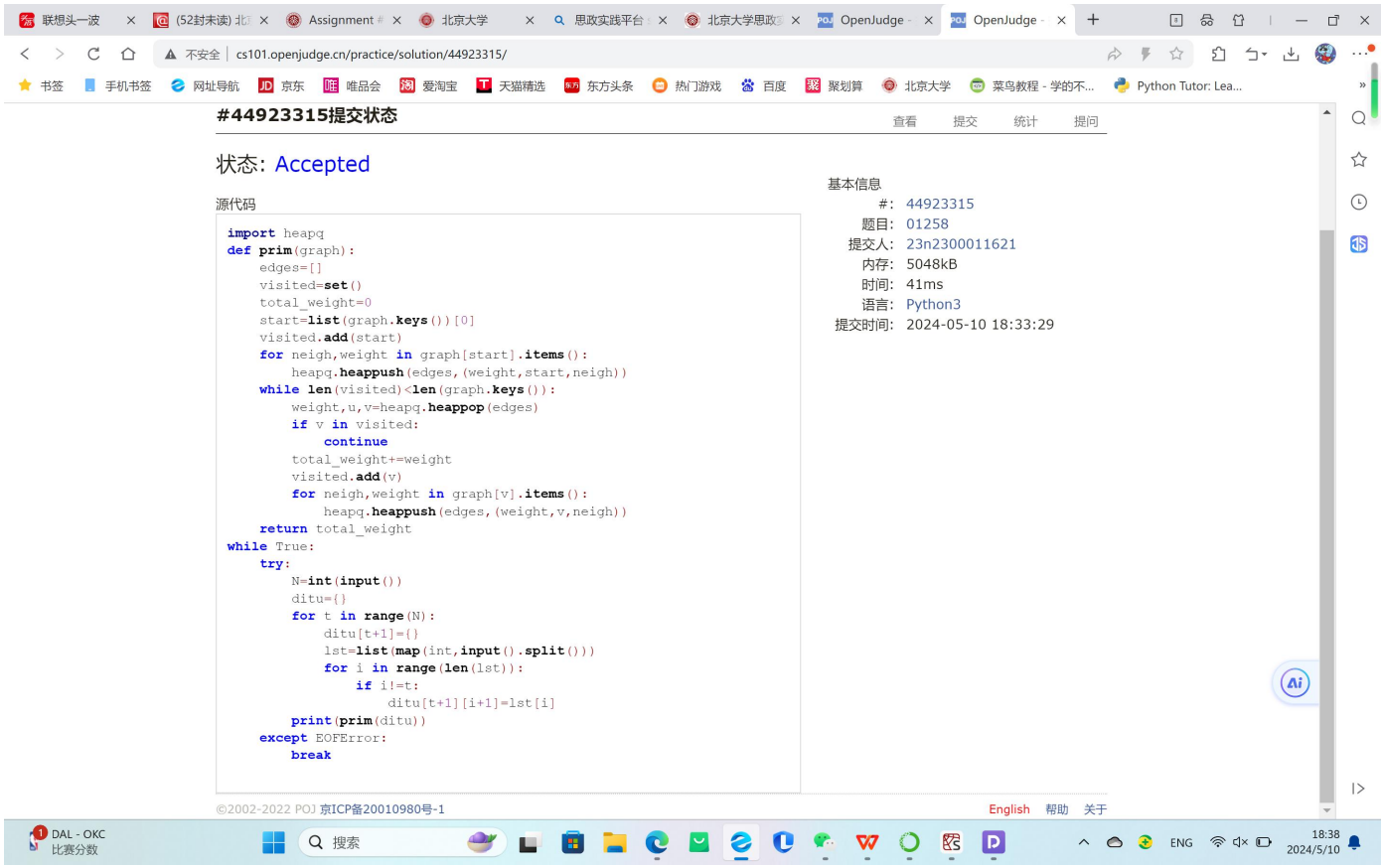
代码

```
import heapq

def prim(graph):
    edges=[]
    visited=set()
    total_weight=0
```

```
    start=list(graph.keys())[0]
    visited.add(start)
    for neigh,weight in graph[start].items():
        heapq.heappush(edges,(weight,start,neigh))
    while len(visited)<len(graph.keys()):
        weight,u,v=heapq.heappop(edges)
        if v in visited:
            continue
        total_weight+=weight
        visited.add(v)
        for neigh,weight in graph[v].items():
            heapq.heappush(edges,(weight,v,neigh))
    return total_weight
while True:
```

```
try:
    N=int(input())
    ditu={}
    for t in range(N):
        ditu[t+1]={}
        lst=list(map(int,input().split()))
        for i in range(len(lst)):
            if i!=t:
                ditu[t+1][i+1]=lst[i]
    print(prim(ditu))
except EOFError:
    break
```



27635: 判断无向图是否连通有无回路(同23163)

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

思路:

代码

```
#

```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

27947: 动态中位数

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

思路:

代码

```
from collections import deque

import heapq

import math

T=int(input())

for _ in range(T):

    lst=deque(map(int,input().split()))

    N=int(math.ceil(len(lst)/2))
```

```
cur=lst.popleft()

ans=[cur]

heap_min=[]

heap_max=[]

while len(lst)>1:

    A=lst.popleft()

    B=lst.popleft()

    if (A-cur)*(B-cur)<=0:

        ans.append(cur)

        if A<=B:

            heapq.heappush(heap_min,-A)

            heapq.heappush(heap_max,B)

        else:

            heapq.heappush(heap_min,-B)
```

```
        heapq.heappush(heap_max,A)

else:

    C=None

    if A>cur:

        heapq.heappush(heap_max,A)

        heapq.heappush(heap_max,B)

        C=heapq.heappop(heap_max)

        heapq.heappush(heap_min,-cur)

        cur=C

    else:

        heapq.heappush(heap_min,-A)

        heapq.heappush(heap_min,-B)

        C=-heapq.heappop(heap_min)

        heapq.heappush(heap_max,cur)
```

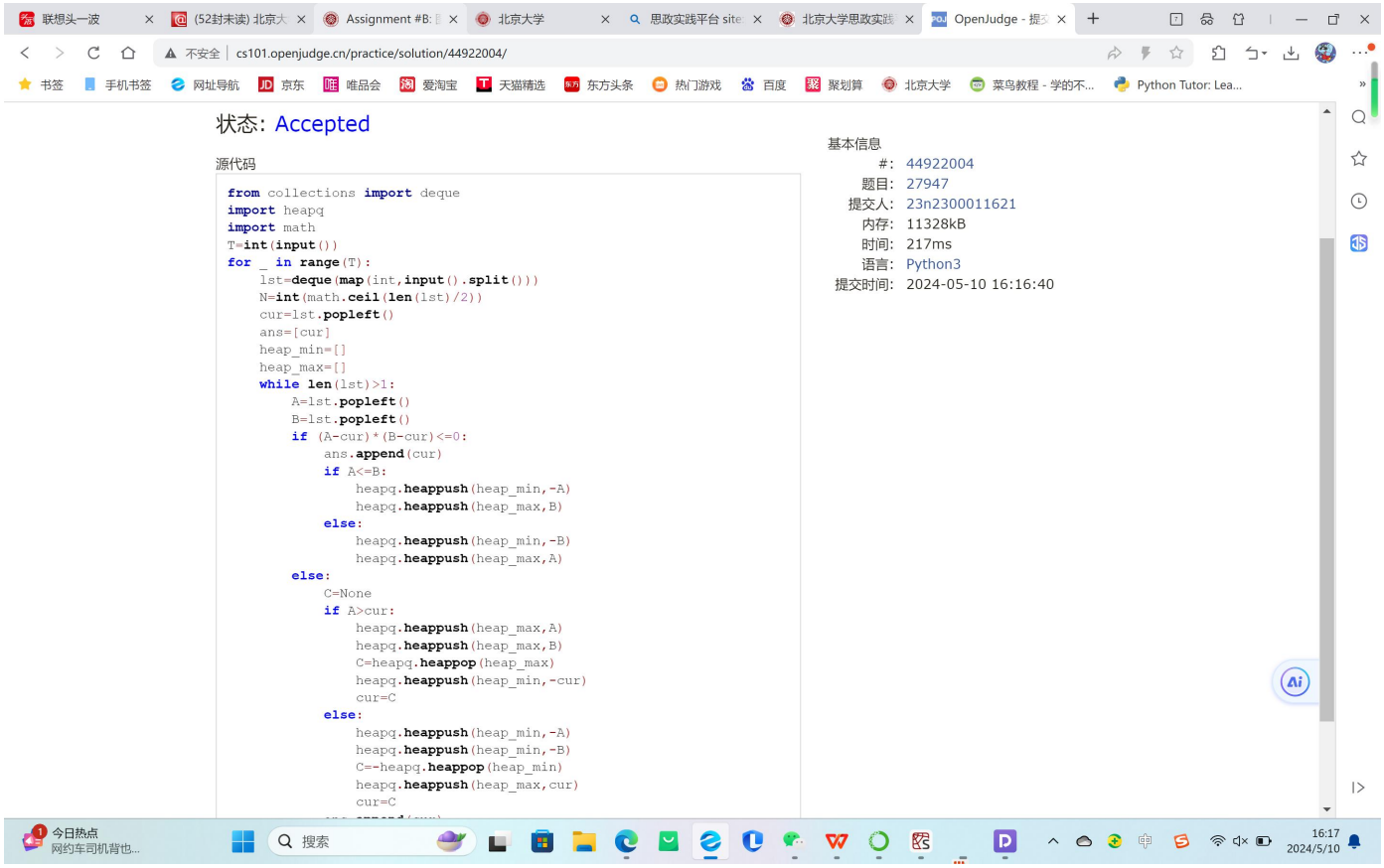
```
            cur=C

        ans.append(cur)

ans=list(map(str,ans))

print(N)

print(' '.join(ans))
```



28190: 奶牛排队

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

思路：单调栈会了，但是再也不想看见奶牛了。

代码

。

```
#

```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

2. 学习总结和收获

在Python中，并不允许直接构建大根堆。若需要大根堆，则应向堆中压入数字的相反数。

本次考试之前我还不熟悉图算法，Prim,Dijkstra还是不够熟悉，在接下来的作业中会加强练习。

关于树，如果你像我一样总是重复建节点，不妨一起来学一学这种覆写方法的方法：

```
class TreeNode:
    _instances = {}

    def __new__(cls, value):
        if value not in cls._instances:
            instance = super().__new__(cls)
            cls._instances[value] = instance
            return instance
        else:
            return cls._instances[value]

    def __init__(self, value):
        if not hasattr(self, 'initialized'):
            self.value = value
            self.left = None
            self.right = None
            self.initialized = True

    def __eq__(self, other):
        return isinstance(other, TreeNode) and self.value == other.value

    def __hash__(self):
        return hash(self.value)
```

首先，我们重写更新的内置方法。通常在使用TreeNode（i）语句后，节点的父节点，子节点的信息会像字典一样丢失。这是__new__方法从中作妖的缘故。要想驯服这个内置方法，我们可以加一条判断：在类的整体中（这里的cls在IDE中是洋红色斜体，表示作为节点类的整体被调用），我们用_instances字典存储节点值，如果该节点值已在代码全域中存在，则直接返回该对象；如果不存在，再调用内置方法。这样就可以避免重新使用这个语句时，系统返回一个“恢复出厂设置”的对象。

然后，我们如何让节点出入边尽头的节点不“走失”呢？__init__方法是我们每个人初学类时必会的语法，但很少有参考书详细地介绍，并强调其作用。它会在实例创建时自动建立起实例的属性。这里的left，right由于和传递的参数无关，会被当作默认值，所以我们应该阻止其进入这个进程，当该节点已经存在时。于是，hasattr()函数应运而生。它本来是用于判断实例是否有某个属性的（推测这个可以用来判断实例属于哪个类），在这里，当“初始化”属性未被定义时，函数返回False，这样新节点就可以建立起来；而已定义后，该if语句可以帮我们跳过__init__方法，从而避免覆盖。__new__方法截断、阻止了节点“恢复出厂设置”后，initialized属性被保留，在__init__这道关卡中，覆盖的进程就会被强行打断。

最后，在绝大多数情境中，不同的节点通常不被允许拥有相同的值。并且，在前中建后，中后建前的过程中，通常在两个表达式中会建两个相同值的节点，如果混淆会导致IDE粗鲁的报错。之前我们通常是通过判断节点的值相等来规避这个问题，现在有了新的方法：通过重写哈希和相等方法，可以让所有节点的值唯一。特别说明，这里的return一个等式或不等式的语句其实是return这些式子的布尔值。

关于二叉堆，构建的规则是在树中从左到右按层级编号，编号和节点在列表中的索引相同。根据这个规则，在看到heapq模块输出的列表时，我们可以根据各个元素的位置，还原出二叉树。