Assignment #D: May月考

Updated 1654 GMT+8 May 8, 2024

2024 spring, Complied by 武昱达 23工院

编程环境

Pycharm Windows 11

1. 题目

02808: 校门外的树

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

思路:

简单模拟

代码

```
L,M=map(int,input().split())
arr=[1 for _ in range(L+1)]
for _ in range(M):
    start,end=map(int,input().split())
    for i in range(start,end+1):
        if arr[i]==1:
            arr[i]=0

cnt=0
for i in range(L+1):
    if arr[i]==1:
        cnt+=1
print(cnt)
```

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:

简单问题。

代码

```
def judge(x:str):
    num=int(x,2)
    return num%5==0

raw=input()
n=len(raw)
arr=['0' for _ in range(n)]
for i in range(n):
    if judge(raw[:i+1]):
        arr[i]='1'

print(''.join(arr))
```

代码运行截图 （至少包含有"Accepted"）

01258: Agri-Net

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

思路:

prim

代码

```
import sys
import heapq
class Vertex:
    def __init__(self, key):
        self.id = key
        self.connectedTo = {}
        self.distance = sys.maxsize
```

```
        self.pred = None

    def __str__(self):
        return "*"+str(self.id)

    def addNeighbor(self, nbr, weight=0):
        self.connectedTo[nbr] = weight

    def getConnections(self):
        return self.connectedTo.keys()

    def getWeight(self, nbr):
        return self.connectedTo[nbr]

    def __lt__(self, other):
        return self.distance < other.distance

class Graph:
    def __init__(self):
        self.vertList = {}
        self.numVertices = 0

    def addVertex(self, key):
        newVertex = Vertex(key)
        self.vertList[key] = newVertex
        self.numVertices += 1
        return newVertex

    def getVertex(self, n):
        return self.vertList.get(n)

    def addEdge(self, f, t, cost=0):
        if f in self.vertList and t in self.vertList:
            vert_f=self.vertList[f]
            vert_t=self.vertList[t]
            if vert_t in vert_f.connectedTo:return

        if f not in self.vertList:
            self.addVertex(f)
        if t not in self.vertList:
            self.addVertex(t)
        self.vertList[f].addNeighbor(self.vertList[t], cost)
        self.vertList[t].addNeighbor(self.vertList[f], cost)

def prim(graph, start):
    pq = []
    start.distance = 0
    heapq.heappush(pq, (0, start))
    visited = set()

    while pq:
        currentDist, currentVert = heapq.heappop(pq)
        if currentVert in visited:
            continue
        visited.add(currentVert)

```

```
        for nextVert in currentVert.getConnections():
            weight = currentVert.getWeight(nextVert)
            if nextVert not in visited and weight < nextVert.distance:
                nextVert.distance = weight
                nextVert.pred = currentVert
                heapq.heappush(pq, (weight, nextVert))

while True:
    try:
        N,g=int(input()),Graph()
        matrix=[[i for i in map(int,input().split())] for _ in range(N)]
        for i in range(N):
            for j in range(i,N):
                if i==j:continue
                g.addEdge(i,j,matrix[i][j])
        prim(g,g.getVertex(0))

        res=0
        for vert in g.vertList.values():
            if vert.pred:
                res+=vert.connectedTo[vert.pred]
        print(res)
    except:break
```

```
# 另外提供一个kruskal算法，可以清晰地看到kruskal简洁很多。
class DisJointSet:
    def __init__(self,num_vertices):
        self.parent=list(range(num_vertices))
        self.rank=[0 for _ in range(num_vertices)]

    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):
        root_x=self.find(x)
        root_y=self.find(y)
        if root_x!=root_y:
            if self.rank[root_x]<self.rank[root_y]:
                self.parent[root_x]=root_y
            elif self.rank[root_x]>self.rank[root_y]:
                self.parent[root_y]=root_x
            else:
                self.parent[root_x]=root_y
                self.rank[root_y]+=1

# graph是邻接表
def kruskal(graph:list):
    res,edges,dsj=[],[],DisJointSet(len(graph))
    for i in range(len(graph)):
        for j in range(i+1,len(graph)):
            if graph[i][j]!=0:
```

```
                edges.append((i,j,graph[i][j]))

    for i in sorted(edges,key=lambda x:x[2]):
        u,v,weight=i
        if dsj.find(u)!=dsj.find(v):
            dsj.union(u,v)
            res.append((u,v,weight))
    return res

while True:
    try:
        n=int(input())
        graph=[list(map(int,input().split())) for _ in range(n)]
        res=kruskal(graph)
        print(sum(i[2] for i in res))
    except EOFError:break
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

27635: 判断无向图是否连通有无回路(同23163)

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

思路：

is_connected函数很好理解，从任意一个点出发（0），如果最终BFS结束后的visited长度和顶数相等，则连通，否则不连通。

is_loop函数的BFS部分稍微难理解一点，首先local_visited是字典，其value为走过的步长。

如果无环，那么对于任意的vert，其next_vert不应当出现在local_visited中，除非next_vert是vert的前驱节点，该情况时steps[next_vert]==steps[vert]-1。

如果有环，则必然有next_vert(且非vert的前驱)出现在local_visited中，其步长不大于vert的步长（与vert同时或者在vert之前被访问）。

即：

```
if local_visited[next_vert]>=steps:return True
```

代码

```
from collections import defaultdict,deque
# graph是邻接表{1:[2,3,4]}
def is_connected(graph,n):
    dq=deque()
    dq.append(0)
    visited=set()
    visited.add(0)
    while dq:
        cur_vert=dq.popleft()
        for next_vert in graph[cur_vert]:
            if next_vert not in visited:
                dq.append(next_vert)
                visited.add(next_vert)
    return len(visited)==n

def is_loop(graph):
    global_visited=set()
    for vertex in graph:
        if vertex not in global_visited:
            # 以下是一个BFS函数。
            local_visited={}
            dq=deque()
            dq.append((vertex,0))
            local_visited[vertex]=0
            global_visited.add(vertex)
            while dq:
                cur_vert,steps=dq.popleft()
                for next_vert in graph[cur_vert]:
                    if next_vert in local_visited:
                        # 关键步骤
                        if local_visited[next_vert]>=steps:
                            return True
                    else:
                        dq.append((next_vert,steps+1))
                        local_visited[next_vert]=steps+1
                        global_visited.add(next_vert)
    return False

n,m=map(int,input().split())
graph=defaultdict(list)
for _ in range(m):
```

```
    a,b=map(int,input().split())
    graph[a].append(b)
    graph[b].append(a)
print('connected:yes' if is_connected(graph,n) else 'connected:no')
print('loop:yes' if is_loop(graph) 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

```
from collections import defaultdict,deque
# graph是邻接表{1:[2,3,4]}
def is_connected(graph,n):
    dq=deque()
    dq.append(0)
    visited=set()
    visited.add(0)
    while dq:
        cur_vert=dq.popleft()
        for next_vert in graph[cur_vert]:
            if next_vert not in visited:
                dq.append(next_vert)
                visited.add(next_vert)
    return len(visited)==n

def is_loop(graph):
    global_visited=set()
    for vertex in graph:
        if vertex not in global_visited:
            # 以下是一个BFS函数。
            local_visited={}
            dq=deque()
            dq.append((vertex,0))
            local_visited[vertex]=0
            global_visited.add(vertex)
            while dq:
                cur_vert,steps=dq.popleft()
                for next_vert in graph[cur_vert]:
                    if next_vert in local_visited:
                        if local_visited[next_vert]>=steps:
                            return True
                    else:
                        dq.append((next_vert,steps+1))
                        local_visited[next_vert]=steps+1
                        global_visited.add(next_vert)
    return False

n,m=map(int,input().split())
graph=defaultdict(list)
for _ in range(m):
    a,b=map(int,input().split())
    graph[a].append(b)
    graph[b].append(a)
print('connected:yes' if is_connected(graph,n) else 'connected:no')
print('loop:yes' if is_loop(graph) else 'loop:no')
```

基本信息

#:	44958289
题目:	27635
提交人:	23n2300011119 (武)
内存:	3820kB
时间:	30ms
语言:	Python3
提交时间:	2024-05-14 10:55:33

27947: 动态中位数

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

思路:

维护两个堆，一个大根堆，一个小根堆，保证两个堆的并是当前所有元素，且大根堆的最大元素不大于小根堆的最小元素

也就是把数据分成两半，其中一半严格大于另一半且其两个堆size相等或至多相差1，则易得中位数。

代码

```
import heapq
```

```python
class Medium_finder:
    def __init__(self):
        self.big_heap=[]
        self.small_heap=[]
        self.big_size=0
        self.small_size=0

    # 插入元素
    def insert(self,val):
        if self.big_size-self.small_size==1:
            heapq.heappush(self.small_heap,val)
            self.small_size+=1
        elif self.big_size==self.small_size:
            heapq.heappush(self.big_heap,-val)
            self.big_size+=1
        if not self.big_heap or not self.small_heap:return
        while -self.big_heap[0] > self.small_heap[0]:
            big=-heapq.heappop(self.big_heap)
            small=heapq.heappop(self.small_heap)
            heapq.heappush(self.big_heap,-small)
            heapq.heappush(self.small_heap,big)

    # 查找中位数
    def findMedium(self):
        if self.big_size==self.small_size:
            return (self.small_heap[0]-self.big_heap[0])/2
        if self.big_size>self.small_size:
            return -self.big_heap[0]

for _ in range(n:=int(input())):
    raw=list(map(int,input().split()))
    m_finder=Medium_finder()
    res=[]
    for idx,val in enumerate(raw):
        m_finder.insert(val)
        if (idx+1)%2==1:
            res.append(m_finder.findMedium())
    print(len(res))
    print(*res)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态: Accepted

源代码

```
import heapq
class Medium_finder:
    def __init__(self):
        self.big_heap=[]
        self.small_heap=[]
        self.big_size=0
        self.small_size=0

    def insert(self,val):
        if self.big_size-self.small_size==1:
            heapq.heappush(self.small_heap,val)
            self.small_size+=1
        elif self.big_size==self.small_size:
            heapq.heappush(self.big_heap,-val)
            self.big_size+=1
        if not self.big_heap or not self.small_heap:return
        while -self.big_heap[0] > self.small_heap[0]:
            big=-heapq.heappop(self.big_heap)
            small=heapq.heappop(self.small_heap)
            heapq.heappush(self.big_heap,-small)
            heapq.heappush(self.small_heap,big)

    def findMedium(self):
        if self.big_size==self.small_size:
            return (self.small_heap[0]-self.big_heap[0])/2
        if self.big_size>self.small_size:
            return -self.big_heap[0]

for _ in range(n:=int(input())):
    raw=list(map(int,input().split()))
    m_finder=Medium_finder()
    res=[]
    for idx,val in enumerate(raw):
        m_finder.insert(val)
        if (idx+1)%2==1:
            res.append(m_finder.findMedium())
    print(len(res))
    print(*res)
```

基本信息

#: 44958829
题目: 27947
提交人: 23n2300011119 (武)
内存: 10816kB
时间: 498ms
语言: Python3
提交时间: 2024-05-14 11:46:24

28190: 奶牛排队

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

思路:

再也不想看到这个题了！！！

```
#
N,res=int(input()),0
hi=[int(input()) for _ in range(N)]
# left[i]是i左边第一个不小于他的元素的索引，right[i]是i右边第一个不大于他的元素的索引。
# 容易知道，对于指定的i，如果i作为右端点，left[i]是左端点的一个上界，反之同理。
left,right=[-1 for _ in range(N)],[N for _ in range(N)]
stack1,stack2=[],[]

for i in range(N-1,-1,-1):
    while stack1 and hi[stack1[-1]]>hi[i]:
        stack1.pop()
    if stack1:right[i]=stack1[-1]
    stack1.append(i)

for i in range(N):
    while stack2 and hi[stack2[-1]]<hi[i]:
        stack2.pop()
    if stack2:left[i]=stack2[-1]
    stack2.append(i)
```

```
20
21 for i in range(N):
22     for j in range(right[i]-1,i,-1):
23         if left[j]<i:
24             res=max(j-i+1,res)
25             break
26
27 print(res)
```

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

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。