Assignment #D: Mock Exam

Updated 1955 GMT+8 Dec 5, 2025

2025 fall, Complied by 杨浩、化院

> **说明：**
>
> 1. Dec月考： AC3（请改为同学的通过数）。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. 1. 题目

1.1 E02734:十进制到八进制

http://cs101.openjudge.cn/practice/02734

思路：

- 略

代码

```
n = int(input())
print(oct(n)[2:])
```

Fence 1

代码运行截图（至少包含有"Accepted"）

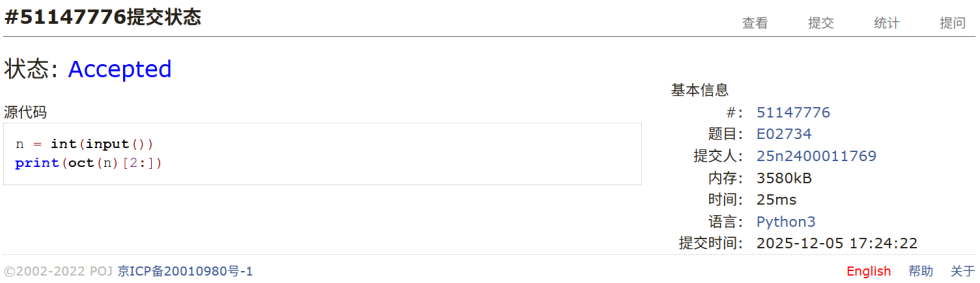

Figure 1

1.2 M21509:序列的中位数

heap, http://cs101.openjudge.cn/practice/21509

思路:

- 大小堆

代码

```
import heapq
def balance():
    while len(up_heap) > len(down_heap)+1:
        t = heapq.heappop(up_heap)
        heapq.heappush(down_heap, -t)

    while len(up_heap) < len(down_heap):
        t = heapq.heappop(down_heap)
        heapq.heappush(up_heap, -t)

N = int(input())
num_list = list(map(int, input().split()))
up_heap = []
down_heap = []
for i in range(N):
    num = num_list[i]
    if i == 0:
        heapq.heappush(up_heap, num)
    else:
        if i % 2 == 1:
            mid = up_heap[0]
        else:
            mid = (up_heap[0]-down_heap[0])/2
        if num < mid:
            heapq.heappush(down_heap, -num)
        else:
            heapq.heappush(up_heap, num)
        balance()
    if i % 2 == 0:
        print(up_heap[0])
```

```
31
```

Fence 2

代码运行截图 （至少包含有"Accepted"）

Figure 2

1.3 M27306: 植物观察

disjoint set, bfs, http://cs101.openjudge.cn/practice/27306/

思路:

- 并查集

代码

```
class DisjointSet:
    def __init__(self,n):
        self.parents = [i for i in range(2*n)]
        self.rank = [0]*2*n

    def find(self,x):
        if self.parents[x] != x:
            self.parents[x] = self.find(self.parents[x])
        return self.parents[x]

    def union(self,x,y):
        x_parent = self.find(x)
        y_parent = self.find(y)
        if x_parent == y_parent:
            return
        elif self.rank[x_parent] > self.rank[y_parent]:
            self.parents[y_parent] = x_parent

        elif self.rank[x_parent] < self.rank[y_parent]:
            self.parents[x_parent] = y_parent

        else:
            self.parents[y_parent] = x_parent
            self.rank[x_parent] += 1
```

```
25
26  n,m = map(int,input().split())
27  d = DisjointSet(n)
28  res = True
29  for i in range(m):
30      x,y,judge = map(int,input().split())
31      if res :
32          if judge == 0:
33              if d.find(x) == d.find(y+n) or d.find(x+n) ==
    d.find(y):
34                  res = False
35                  continue
36              d.union(x,y)
37              d.union(x+n,y+n)
38          else:
39              if d.find(x) == d.find(y) or d.find(x+n) ==
    d.find(y+n):
40                  res = False
41                  continue
42              d.union(x+n,y)
43              d.union(x,y+n)
44
45  if res :
46      print('YES')
47  else:
48      print('NO')
```

Fence 3

代码运行截图 （至少包含有"Accepted"）

Figure 3

1.4 M29740:神经网络

Topological order, http://cs101.openjudge.cn/practice/29740/

思路:

- 题目本身是简单的bfs可以解决的问题，难点在于存在很多的细节和要点（重复的边，输入不用-u），一次写对很有难度，没有错误数据的帮助下也很难debug

代码

```python
from collections import defaultdict
from collections import deque
def mian():


    def begin_check(que,in_degree):
        check_que = que.copy()
        degree = in_degree.copy()

        while check_que:
            v = check_que.popleft()

            for u in graph[v]:
                degree[u] -= 1
                if degree[u] == 0:
                    check_que.append(u)
        for i in degree:
            if degree[i] != 0:
                return False
        return True

    n, p = map(int, input().split())
    c_list = []
    u_list = []
    for i in range(n):
        c, u = map(int, input().split())
        c_list.append(c)
        u_list.append(u)

    graph = [defaultdict(int) for _ in range(n)]
    in_degree = [0] * n
    out_degree = [0] * n
    for _ in range(p):
        i, j, w = map(int, input().split())
        i = i - 1
        j = j - 1
        if j not in graph[i]:
            in_degree[j] += 1
            out_degree[i] += 1

        graph[i][j] += w
    que = deque()
    res = []
    for i in range(n):
        if in_degree[i] == 0:
            que.append(i)
            u_list[i] = 0
        if out_degree[i] == 0:
            res.append(i)

    if not que or not begin_check(que,in_degree):
```

```
            print('NULL')
        else:

            while que:
                i = que.popleft()
                for u in graph[i]:
                    if c_list[i] - u_list[i] > 0:
                        c_list[u] += graph[i][u] * (c_list[i] -
    u_list[i])
                    in_degree[u] -= 1
                    if in_degree[u] == 0:
                        que.append(u)
            ans = True
            for i in res:
                if c_list[i] - u_list[i] > 0:
                    print(i + 1, c_list[i] - u_list[i])
                    ans = False
            if ans:
                print('NULL')

    if __name__ == '__main__':
        mian()
```

Fence 4

代码运行截图 （至少包含有"Accepted"）

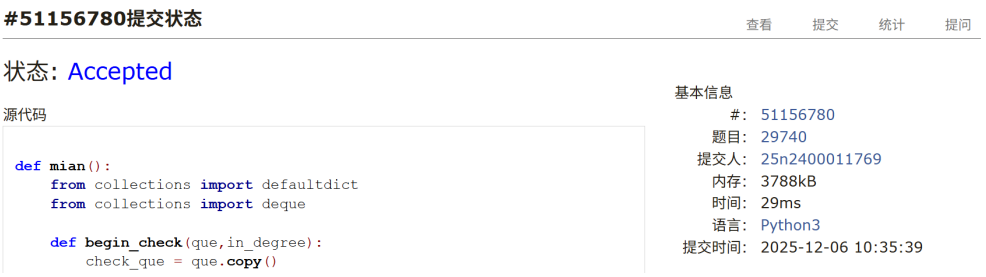

Figure 4

1.5 T27351:01最小生成树

mst, http://cs101.openjudge.cn/practice/27351/

思路:

- 此题难度在与连接边权为0的点，100000的顶点采用`O(n**2)`的暴力遍历构建会TLE，使用集合记录未被访问的顶点来加速连接过程勉强可以AC。
- 找AI写了个更高效的代码，0权边的连接方式依然采取集合链接，但不再使用Kruskal算法进行后续连接，直接对前面0权边连接的顶点岛的经行计数，返回总数-1。
- 此题边权有0和1的差别，且存在大量的顶点，使用Prim和Kruska算法都是相当不合适的。

代码

```python
from collections import defaultdict
from collections import deque
class DisjointSet:
    def __init__(self,n):
        self.parents = [i for i in range(n)]
        self.rank = [0]*n

    def find(self,x):
        if self.parents[x] != x:
            self.parents[x] = self.find(self.parents[x])
        return self.parents[x]

    def union(self,x,y):
        x_parent = self.find(x)
        y_parent = self.find(y)
        if x_parent == y_parent:
            return
        elif self.rank[x_parent] > self.rank[y_parent]:
            self.parents[y_parent] = x_parent

        elif self.rank[x_parent] < self.rank[y_parent]:
            self.parents[x_parent] = y_parent

        else:
            self.parents[y_parent] = x_parent
            self.rank[x_parent] += 1

n,m = map(int,input().split())
res = 0
disjoint = DisjointSet(n)
edges = []
graph = defaultdict(set)
for i in range(m):
    u,v = map(int,input().split())
    u = u-1
    v = v-1
    edges.append((u,v))
    graph[u].add(v)
    graph[v].add(u)


check_set = set(range(n))
for i in range(n):

    if i not in check_set:
        continue

    que = deque()
    que.append(i)
```

```
    check_set.discard(i)
    while que:
        u = que.popleft()
        next_list = []
        for i in check_set:
            if i not in graph[u]:
                disjoint.union(u,i)
                next_list.append(i)
        for i in next_list:
            que.append(i)
            check_set.discard(i)



res = 0
for u,v in edges:
    if disjoint.find(u) == disjoint.find(v):
        continue
    else:
        res += 1
        disjoint.union(u,v)
print(res)
```

Fence 5

```
import sys
from collections import deque

def main():
    data = sys.stdin.buffer.read().split()
    if not data:
        return
    it = iter(data)
    n = int(next(it))
    m = int(next(it))
    adj = [set() for _ in range(n + 1)]
    for _ in range(m):
        a = int(next(it))
        b = int(next(it))
        adj[a].add(b)
        adj[b].add(a)

    unvisited = set(range(1, n + 1))
    components = 0
    while unvisited:
        u = unvisited.pop()
        components += 1
        queue = deque([u])
        while queue:
            x = queue.popleft()
            to_keep = set()
            for y in adj[x]:
```

```
                if y in unvisited:
                    to_keep.add(y)
            candidates = unvisited - to_keep
            for v in candidates:
                queue.append(v)
            unvisited = to_keep
        print(components - 1)

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

Fence 6

代码运行截图 （至少包含有"Accepted"）

Figure 5

Figure 6

1.6 T30193:哈密顿激活层

DFS+剪枝, http://cs101.openjudge.cn/practice/30193/

思路：

- 时间复杂度优化作用：被锁定的关键神经元剪枝（距离+时间）>>bfs连通性剪枝>>Warnsdorff 算法。
- 不对关键神经元剪枝，必定TLE；bfs连通性剪枝和Warnsdorff 算法二选一可以AC，仅使用前者可优化至1000ms，仅使用后者可优化至5000ms，二者一起使用优化至300ms。
- 被锁定的关键神经元剪枝：
    - 所有时间戳大于当前时间戳的关键神经元曼哈顿距离小于时间差

- 这些关键神经元未被访问（可以等价替换为 当且仅当时间戳相等时，坐标相等）

代码

```python
from collections import deque

def manhattan_distance(p1, p2):
    return abs(p1[0] - p2[0]) + abs(p1[1] - p2[1])

def distance_check(pr,t):
    for target_x,target_y,target_t in demand[1:]:
        if target_t >= t:
            target = (target_x,target_y)
            distance = manhattan_distance(pr,target)
            if visited[target_x][target_y]:
                return False
            if target in bad:
                return False
            if distance > target_t-t:
                return False
    return True

def connect_check(pr,n,m):
    que = deque()
    que.append(pr)
    reachable = set()
    while que:
        x,y = que.popleft()
        for dx,dy in delta:
            if 1<=x+dx<=n and 1<=y+dy<=m and not visited[x+dx][y+dy] and (x+dx,y+dy) not in reachable and (x+dx,y+dy) not in bad:
                que.append((x+dx,y+dy))
                reachable.add((x+dx,y+dy))
    if reachable == un_visited:
        return True
    else:
        return False

def count_neighbors(pr,n,m):
    cnt = 0
    for dx,dy in delta:
        x = dx+pr[0]
        y = dy+pr[1]
        if 1<=x<=n and 1<=y<=m and not visited[x][y] and (x,y) not in bad:
            cnt += 1
    return cnt


```

```
44
45  def dfs(pr,t,n,m,path,cnt,end):
46
47      if cnt == end:
48          ans.append(path[:])
49          return True
50      if not connect_check(pr,n,m):
51          return False
52
53      candidate = []
54      for dx, dy in delta:
55          new_pr = (pr[0] + dx, pr[1] + dy)
56          if 1 <= new_pr[0] <= n and 1 <= new_pr[1] <= m and
    not visited[new_pr[0]][new_pr[1]] and new_pr not in bad:
57              if distance_check(new_pr, t + 1):
58                  count = count_neighbors(new_pr, n, m)
59                  candidate.append((count, new_pr[0],
    new_pr[1]))
60      candidate.sort()
61
62      for cnt_0, x, y in candidate:
63          visited[x][y] = True
64          un_visited.discard((x, y))
65          if dfs((x, y), t + 1, n, m, path + [(x, y)], cnt + 1,
    end):
66              return True
67          visited[x][y] = False
68          un_visited.add((x, y))
69      return False
70
71
72  n,m,k,b = map(int,input().split())
73  demand = [list(map(int,input().split())) for i in range(k)]
74  demand.sort(key=lambda x:x[2])
75  bad = [ tuple(map(int,input().split())) for i in range(b)]
76  bad = set(bad)
77  delta = [(0,1),(0,-1),(1,0),(-1,0)]
78  visited = [[False]*(m+1) for _ in range(n+1)]
79  un_visited = set()
80  for x in range(1,n+1):
81      for y in range(1,m+1):
82          if (x,y) not in bad:
83              un_visited.add((x,y))
84  begin_pr = tuple(demand[0][:2])
85  path = [begin_pr]
86  visited[begin_pr[0]][begin_pr[1]] = True
87  un_visited.discard(begin_pr)
88  ans = []
89  dfs(begin_pr,1,n,m,path,1,n*m-len(bad))
90  if not ans:
91      print(-1)
```

```
92  else:
93      for x,y in ans[0]:
94          print(x,y)
```

Fence 7

代码运行截图 （至少包含有"Accepted"）

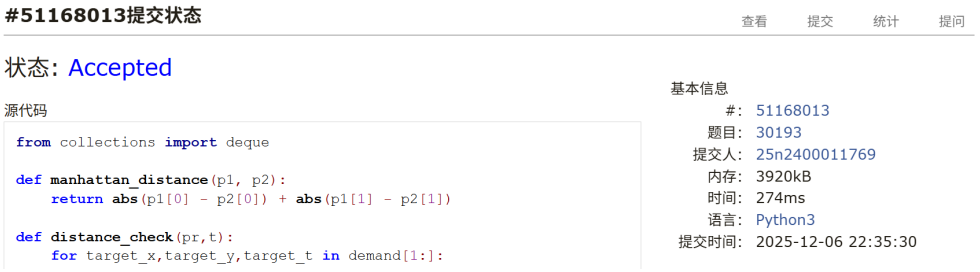

Figure 7

2. 2. 学习总结和收获

这次考试AC3。第4题比较可惜，注意到了提示，大体思路也是对的，但在计算出入度时重复计数了重边，考试中没有发现这个问题。第5，6题难度比较大。第5题被MST题面误导，思考用Kruska算法解决，后面意识到了点的数量很多，边权为1的边很少，尝试优化并查集连接边权为0的边的时间复杂度。考完了尝试用集合优化才AC掉。问了AI，才发现还是被MST题面误导，不该用Kruska算法，直接使用集合数独立单元数量即可。第6题优化策略比较复杂。之前学习过A*算法，这对这个题有一定启发，故一开始能想到对锁定的关键神经元剪枝，但发现还是TLE。想到Warnsdorff 算法，再度优化勉强AC。又找AI老师学习了一下，发现还有bfs连通性剪枝优化、死胡同剪枝优化。bfs连通性剪枝比较好理解，效果也很好。

近期练习的图的题目都是所学算法的基本应用，像第5题这种没有使用所学的基本算法的题目就很难做出来。第4题这类阅读量较大，细节较多的题目平时有训练，但很难一次AC，debug很又依赖测试数据，一般给的测试数据很简单，无法暴露bug，也有一定的困难。