

Assignment #D: Mock Exam

Updated 1955 GMT+8 Dec 5, 2025

2025 fall, Compiled 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

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

Fence 1

代码运行截图 (至少包含有“Accepted”)

#51147776提交状态

[查看](#) [提交](#) [统计](#) [提问](#)

状态: Accepted

源代码

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

基本信息

#: 51147776
 题目: E02734
 提交人: 25n2400011769
 内存: 3580kB
 时间: 25ms
 语言: Python3
 提交时间: 2025-12-05 17:24:22

©2002-2022 POJ 京ICP备20010980号-1

[English](#) [帮助](#) [关于](#)

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

```
1  import heapq
2  def balance():
3      while len(up_heap) > len(down_heap)+1:
4          t = heapq.heappop(up_heap)
5          heapq.heappush(down_heap, -t)
6
7      while len(up_heap) < len(down_heap):
8          t = heapq.heappop(down_heap)
9          heapq.heappush(up_heap, -t)
10
11  N = int(input())
12  num_list = list(map(int, input().split()))
13  up_heap = []
14  down_heap = []
15  for i in range(N):
16      num = num_list[i]
17      if i == 0:
18          heapq.heappush(up_heap, num)
19      else:
20          if i % 2 == 1:
21              mid = up_heap[0]
22          else:
23              mid = (up_heap[0]-down_heap[0])/2
24          if num < mid:
25              heapq.heappush(down_heap, -num)
26          else:
27              heapq.heappush(up_heap, num)
28      balance()
29  if i % 2 == 0:
30      print(up_heap[0])
```

代码运行截图 (至少包含有"Accepted")

#51148492提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
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)
```

基本信息

#: 51148492
 题目: M21509
 提交人: 25n2400011769
 内存: 14860kB
 时间: 200ms
 语言: Python3
 提交时间: 2025-12-05 17:51:41

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

```
1 class DisjointSet:
2     def __init__(self,n):
3         self.parents = [i for i in range(2*n)]
4         self.rank = [0]*2*n
5
6     def find(self,x):
7         if self.parents[x] != x:
8             self.parents[x] = self.find(self.parents[x])
9         return self.parents[x]
10
11     def union(self,x,y):
12         x_parent = self.find(x)
13         y_parent = self.find(y)
14         if x_parent == y_parent:
15             return
16         elif self.rank[x_parent] > self.rank[y_parent]:
17             self.parents[y_parent] = x_parent
18
19         elif self.rank[x_parent] < self.rank[y_parent]:
20             self.parents[x_parent] = y_parent
21
22         else:
23             self.parents[y_parent] = x_parent
24             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")

#51148034提交状态

[查看](#) [提交](#) [统计](#) [提问](#)

状态: Accepted

源代码

```

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:

```

基本信息

#: 51148034
 题目: M27306
 提交人: 25n2400011769
 内存: 3732kB
 时间: 37ms
 语言: Python3
 提交时间: 2025-12-05 17:32:34

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

```

1  from collections import defaultdict
2  from collections import deque
3  def mian():
4
5
6      def begin_check(que, in_degree):
7          check_que = que.copy()
8          degree = in_degree.copy()
9
10         while check_que:
11             v = check_que.popleft()
12
13             for u in graph[v]:
14                 degree[u] -= 1
15                 if degree[u] == 0:
16                     check_que.append(u)
17         for i in degree:
18             if degree[i] != 0:
19                 return False
20         return True
21
22     n, p = map(int, input().split())
23     c_list = []
24     u_list = []
25     for i in range(n):
26         c, u = map(int, input().split())
27         c_list.append(c)
28         u_list.append(u)
29
30     graph = [defaultdict(int) for _ in range(n)]
31     in_degree = [0] * n
32     out_degree = [0] * n
33     for _ in range(p):
34         i, j, w = map(int, input().split())
35         i = i - 1
36         j = j - 1
37         if j not in graph[i]:
38             in_degree[j] += 1
39             out_degree[i] += 1
40
41         graph[i][j] += w
42     que = deque()
43     res = []
44     for i in range(n):
45         if in_degree[i] == 0:
46             que.append(i)
47             u_list[i] = 0
48         if out_degree[i] == 0:
49             res.append(i)
50
51     if not que or not begin_check(que, in_degree):

```

```

52         print('NULL')
53     else:
54
55         while que:
56             i = que.popleft()
57             for u in graph[i]:
58                 if c_list[i] - u_list[i] > 0:
59                     c_list[u] += graph[i][u] * (c_list[i] -
u_list[i])
60                 in_degree[u] -= 1
61                 if in_degree[u] == 0:
62                     que.append(u)
63             ans = True
64             for i in res:
65                 if c_list[i] - u_list[i] > 0:
66                     print(i + 1, c_list[i] - u_list[i])
67                     ans = False
68             if ans:
69                 print('NULL')
70
71 if __name__ == '__main__':
72     mian()

```

Fence 4

代码运行截图 (至少包含有"Accepted")

#51156780提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```

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

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

```

基本信息

#: 51156780
 题目: 29740
 提交人: 25n2400011769
 内存: 3788kB
 时间: 29ms
 语言: Python3
 提交时间: 2025-12-06 10:35:39

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

```

1  from collections import defaultdict
2  from collections import deque
3  class DisjointSet:
4      def __init__(self,n):
5          self.parents = [i for i in range(n)]
6          self.rank = [0]*n
7
8      def find(self,x):
9          if self.parents[x] != x:
10             self.parents[x] = self.find(self.parents[x])
11             return self.parents[x]
12
13     def union(self,x,y):
14         x_parent = self.find(x)
15         y_parent = self.find(y)
16         if x_parent == y_parent:
17             return
18         elif self.rank[x_parent] > self.rank[y_parent]:
19             self.parents[y_parent] = x_parent
20
21         elif self.rank[x_parent] < self.rank[y_parent]:
22             self.parents[x_parent] = y_parent
23
24         else:
25             self.parents[y_parent] = x_parent
26             self.rank[x_parent] += 1
27
28     n,m = map(int,input().split())
29     res = 0
30     disjoint = DisjointSet(n)
31     edges = []
32     graph = defaultdict(set)
33     for i in range(m):
34         u,v = map(int,input().split())
35         u = u-1
36         v = v-1
37         edges.append((u,v))
38         graph[u].add(v)
39         graph[v].add(u)
40
41
42     check_set = set(range(n))
43     for i in range(n):
44
45         if i not in check_set:
46             continue
47
48     que = deque()
49     que.append(i)

```

```

50     check_set.discard(i)
51     while que:
52         u = que.popleft()
53         next_list = []
54         for i in check_set:
55             if i not in graph[u]:
56                 disjoint.union(u,i)
57                 next_list.append(i)
58         for i in next_list:
59             que.append(i)
60             check_set.discard(i)
61
62
63
64     res = 0
65     for u,v in edges:
66         if disjoint.find(u) == disjoint.find(v):
67             continue
68         else:
69             res += 1
70             disjoint.union(u,v)
71     print(res)

```

Fence 5

```

1     import sys
2     from collections import deque
3
4     def main():
5         data = sys.stdin.buffer.read().split()
6         if not data:
7             return
8         it = iter(data)
9         n = int(next(it))
10        m = int(next(it))
11        adj = [set() for _ in range(n + 1)]
12        for _ in range(m):
13            a = int(next(it))
14            b = int(next(it))
15            adj[a].add(b)
16            adj[b].add(a)
17
18        unvisited = set(range(1, n + 1))
19        components = 0
20        while unvisited:
21            u = unvisited.pop()
22            components += 1
23            queue = deque([u])
24            while queue:
25                x = queue.popleft()
26                to_keep = set()
27                for y in adj[x]:

```



```

28         if y in unvisited:
29             to_keep.add(y)
30         candidates = unvisited - to_keep
31         for v in candidates:
32             queue.append(v)
33         unvisited = to_keep
34         print(components - 1)
35
36 if __name__ == "__main__":
37     main()

```

Fence 6

代码运行截图 (至少包含有"Accepted")

#51157636提交状态

[查看](#)
[提交](#)
[统计](#)
[提问](#)

状态: Accepted

源代码

```

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

```

基本信息

#: 51157636
 题目: 27351
 提交人: 25n2400011769
 内存: 76796kB
 时间: 6119ms
 语言: Python3
 提交时间: 2025-12-06 11:08:23

Figure 5

#51157941提交状态

[查看](#)
[提交](#)
[统计](#)
[提问](#)

状态: Accepted

源代码

```

import sys
from collections import deque

def main():
    data = sys.stdin.buffer.read().split()
    if not data:
        return

```

基本信息

#: 51157941
 题目: 27351
 提交人: 25n2400011769
 内存: 82096kB
 时间: 290ms
 语言: Python3
 提交时间: 2025-12-06 11:25:36

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

```

1  from collections import deque
2
3  def manhattan_distance(p1, p2):
4      return abs(p1[0] - p2[0]) + abs(p1[1] - p2[1])
5
6  def distance_check(pr, t):
7      for target_x, target_y, target_t in demand[1:]:
8          if target_t >= t:
9              target = (target_x, target_y)
10             distance = manhattan_distance(pr, target)
11             if visited[target_x][target_y]:
12                 return False
13             if target in bad:
14                 return False
15             if distance > target_t - t:
16                 return False
17         return True
18
19  def connect_check(pr, n, m):
20      que = deque()
21      que.append(pr)
22      reachable = set()
23      while que:
24          x, y = que.popleft()
25          for dx, dy in delta:
26              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:
27                  que.append((x + dx, y + dy))
28                  reachable.add((x + dx, y + dy))
29          if reachable == un_visited:
30              return True
31          else:
32              return False
33
34  def count_neighbors(pr, n, m):
35      cnt = 0
36      for dx, dy in delta:
37          x = dx + pr[0]
38          y = dy + pr[1]
39          if 1 <= x <= n and 1 <= y <= m and not visited[x][y] and
(x, y) not in bad:
40              cnt += 1
41      return cnt
42
43

```

```

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")

#51168013提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
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:]:
```

基本信息

```
#: 51168013
题目: 30193
提交人: 25n2400011769
内存: 3920kB
时间: 274ms
语言: Python3
提交时间: 2025-12-06 22:35:30
```

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。