113 學年度技藝競賽培訓

Ching-Chang Lin
cclin@ba.tpcu.edu.tw

Taipei City University of Science and Technology — November 11, 2024

113 學年度全國高級中等學校技藝競賽官網

為提升全國高級中等學校學生的技藝學習動機，促進學校間技藝的交流與觀摩，並提升學生技藝水準，檢視學生學習成效及技職教育成果，每年 11 月舉辦的全國高級中等學校學生技藝競賽，統計 110 學年度五大類競賽職種，分別有工業類 28 項、商業類 11 項、農業類 9 項、家事類 8 項、海車水產類 7 項。旨在強化學生的學習動機，促進技藝交流與觀摩，並提升技藝水準。

統計 113 學年度商業類競賽職種 (A) 平面設計、(B) 網頁設計、(C) 程式設計、(D) 文書處理、(E) 電腦繪圖、(F) 會計資訊、(G) 商業簡報、(H) 職場英文，八項競賽項目，計有 137 所學校、511 位學生參與競賽。

112 術科重點

詳細說明文件請見：商科技藝競賽線上評分說明文件成績公告

Question 328

Problem A
二數有權重的相加-(單列)

Info:

送分題

```
a,b = map(int,input().split())
print(4*a+6*b)
```

採用 sys.stdin.readline() 或 sys.stdin.readlines() 可以提高效率, 但需要引入 sys 模組

```
import sys

a,b = map(int,sys.stdin.readline().split())
print(4*a+6*b)
```

Question 329

Problem B

剪刀石頭布單行版

Info:

送分題

```
a,b = input().split()

if a=='Y' and b=='X' or a=='X' and b=='O' or a=='O' and b=='Y':
    print(1)
elif a==b:
    print(0)
else:
    print(2)
```

Question 330

Problem C
閏年

Info:
送分題

```
y=int(input())+1911

if  y%400 == 0 or y%4 == 0 and y%100:
    print('True')
else:
    print('False')
```

Question 331

Problem D
ASCII 碼

Info:
簡單題

```
n = int(input())
if 65 <= n <= 90:
    print(1)
elif 97 <= n <=122:
    print(2)
else:
    print(0)
```

Question 332

Problem E
2 的冪

Info:
簡單題

```
n = int(input())
for i in range(100000000):
    la = pow(2,i)
    if la == n:
        print(2**i)
        break
    elif la > n:
        print(2**(i-1))
        break
```

Info: 運用 $log_2(n)$ 解法，效率較高，但是需要引入 math 模組

```
import math
n = int(input())
print(2**int(math.log2(n)))
```

Question 333

Problem F
質因數分解

Info:

基礎數學題，解法 1: 逐一判斷

```
n = int(input())
ans = []
while n > 1:
    for i in range(2,n+1):
        if n % i == 0:
            ans.append(i)
            n //= i
            break

print(*ans)
```

基礎數學題，解法 2:

```
n=int(input())
ans=[]
for i in range(2,n+1):
    while n % i == 0:
        ans.append(i)
        n //= i

print(*ans)
```

Question 334

Problem G

因數加總

Info:

簡單題

```
n = int(input())
ans = 0

for i in range(1,n+1):
    if n%i == 0:
        ans += i

print(ans)
```

Question 335

Problem H

數字運算 (減 1 或除以 10)

Info:

基礎數學題

```
n,k = map(int,input().split())

for i in range(k):
    if n%10 == 0:
        n//=10
    else:
        n -= 1

print(n)
```

Question 336

Problem I

數字相加

Info:

```
for _ in range(int(input())):
n = int(input())
A = list(map(int,input().split()))
q = []
tar = A.pop()
for i in range(n-1):
    if tar - A[i] in q:
        print('YES')
        break
    else:
        q.append(A[i])
else:
    print('NO')
```

Question 337

Problem J

七段顯示器

Info:

```
for _ in range(int(input())):
n = int(input())
ans = '1' * (n//2)
if n%2:
    ans = '7' + ans[1:]
print(ans)
```

Question 338

Problem K
字串

Info:

```
for _ in range(int(input())):
n = int(input())
ans = set()
S = list(input())
for i in range(n-1):
    new = S[:]
    new[i] = ''
    new[i+1] = ''
    s = ''.join(new)
    ans.add(s)

print(len(ans))
```

Question 339

Problem K2
字串

Info: 此題為 Problem K 的進階考題，有時間限制，屬於另一種解法, 但是效率較高

```
for _ in range(int(input())):
n = int(input())
ans = set()
S = list(input())
ans = n - 1
# 1
i = 0
while i < len(S)-1:
    curr = 1
    j = i + 1
    while j < len(S):
        if S[j] == S[i]:
            curr += 1
        else:
            ans -= max(0,curr - 2)
            i = j-1
            break
        j += 1
    else:
        ans -= max(0,curr - 2)
        i = j
    i += 1

# 2
for i in range(2,len(S)):
    if S[i] == S[i-2] and S[i] != S[i-1]:
        ans -= 1

print(ans)
```

Question 340

Problem L
中位數

Info:

```
A = []
for i in range(int(input())):
    A.append(int(input()))
    n = len(A)//2
    A.sort()
    if len(A)%2 ==1:
        print(A[len(A)//2])
    else:
        print((A[n]+A[n-1])//2)
```

Question 341

Problem M

分組

Info:

```
for _ in range(int(input())):
n,x = map(int,input().split())
A = list(map(int,input().split()))
A.sort(reverse=True)
ans = 0
p = 1
i = 0
while i < len(A):
    if A[i] * p >= x:
        ans += 1
        i += p
    else:
        p += 1
        i += 1
print(ans)
```

Question 342

Problem N

函數

Info:

```
n = int(input())

if n%2 == 0:
    print(n//2)
else:
    print(n//2-n)
```

Question 343

Problem O

洗牌

Info:

```
n,m = map(int,input().split())

A = list(map(int,input().split()))
for i in range(m):
    new = []
    a = A[:n//2]
    b = A[n//2:]
    for i in range(len(a)):
        new.append(a[i])
        new.append(b[i])
    A = new[:]

print(*A)
```

Question 344

Problem P
魔術方陣

Info:

```
A = []
total = 0
for i in range(3):
    arr = list(map(int,input().split()))
    total += sum(arr)
    A.append(arr)

total //= 2
A[0][0] = total-A[1][0]-A[2][0]
A[1][1] = total-A[0][1]-A[2][1]
A[2][2] = total-A[0][2]-A[1][2]
for i in A:
    print(*i)

'''
0 1 6
3 0 7
4 9 0
'''
```

程式碼第 13 行:

print(*i) 中的星號 * 是 Python 的「開箱運算子」(unpacking operator)，Unpacking 顧名思義就是將打包好的資料取出來。

Unpacking Operator(開箱運算子) 分為:

- * 符號：可用於任何可疊代的 (Iterable) 物件。
- ** 符號：只能用於字典 (Dictionary) 物件。

主要用途為分解可疊代的 (Iterable) 物件元素，在進行建立或合併時非常的實用。

舉例: i 是一個串列（例如 [1, 2, 3]），使用 *i 可以將串列中的每個元素取出，並分別傳遞給 print 函數。這樣可以實現將串列中的元素逐一列印出，並以空格分隔，而不會有串列的中括號 []，若搭配每次迴圈執行時，print(*i) 會將當前串列的所有元素列印在同一行，並以空格分隔，這樣就實現了將二維串列中的每一行元素依序列印出的效果。

PS: 進階學習: [Python 教學]Python Unpacking 實用技巧分享

Question 345

Problem Q

數字全部相加成本

Info:

```
while 1:
n = int(input())
if not n:
    break
A = list(map(int,input().split()))
ans = 0
while len(A)>1:
    A.sort()
    A[0] += A[1]
    ans += A[0]
    A.pop(1)
print(ans)
```

Question 346

Problem R
樹的直徑

Info:

問題分析:

- 樹的直徑: 樹中任意兩個節點之間最長的路徑長度。
- 輸入格式: 第一行為節點數，接下來的 n-1 行表示邊的連接。
- 輸出格式: 輸出樹的直徑。

解題思路:

- 建立樹的結構：使用鄰接列表或鄰接矩陣來表示樹的結構，並根據輸入的節點和邊，構建每個節點的相鄰節點。
- BFS 函數：使用 BFS 計算從起始節點出發到其他節點的距離，並找到最遠的節點及其距離。
- 第一次 BFS：從節點 1 出發，找到距離最遠的節點。
- 第二次 BFS：從第一次找到的最遠節點出發，再次進行 BFS，計算出樹的直徑。
- 輸出結果：最終輸出樹的直徑。

討論:

為何從最遠節點出發可以找到直徑？

當我們第一次 BFS 從隨便選的一個節點（例如節點 1）開始時，可以確保找到的最遠節點（如 A）已經在樹的某個「邊界」上。這樣，再從 A 開始，因為樹是無環結構，所以 A 到樹的另一端（即 B）的路徑必然是樹的最長路徑，也就是直徑。

結論:

這兩次 BFS 過程等效於從樹的一端找另一端，利用無環圖的特性，保證了找到的路徑是最長的，也就是樹的直徑。

Info:

方法 1:

```python
# 定義一個函數來進行 BFS 並找出最遠距離及節點
def bfs(start):
    distance = [-1] * (n + 1)
    distance[start] = 0
    queue = [start]
    head = 0
    while head < len(queue):
        node = queue[head]
        head += 1
        for neighbor in edges[node]:
            if distance[neighbor] == -1:
                distance[neighbor] = distance[node] + 1
                queue.append(neighbor)
    max_dist = max(distance)
    farthest_node = distance.index(max_dist)
    return max_dist, farthest_node

# 建立樹的鄰接表表示法
edges = {}
n = int(input())
for _ in range(n - 1):
    a, b = map(int, input().split())
    if a not in edges:
        edges[a] = [b]
    else:
        edges[a].append(b)
    if b not in edges:
        edges[b] = [a]
    else:
        edges[b].append(a)

# edges = [(1, 2), (1, 3), (3, 4), (3, 5)]
# 第一次 BFS 從節點 1 開始
_, farthest_from_1 = bfs(1)

# 第二次 BFS 從第一次找到的最遠節點開始
diameter, _ = bfs(farthest_from_1)
# 輸出樹的直徑
print(diameter)
```

Info:

方法 2:

```python
def tree_diameter(edges):
    n = len(edges) + 1
    graph = {}
    for u, v in edges:
        if u not in graph:
            graph[u] = []
        graph[u].append(v)
        if v not in graph:
            graph[v] = []
        graph[v].append(u)

    dis_1to = [-1] * (n+1)
    dis_1to[1] = 0
    que = [1]
    head = 0
    while head<len(que):
        node = que[head]
        head += 1
        for v in graph[node]:
            if dis_1to[v] == -1:
                dis_1to[v] = dis_1to[node]+1
                que.append(v)

    i = dis_1to.index(max(dis_1to))
    dis_ito = [-1] * (n+1)
    dis_ito[i] = 0
    que = [i]
    head = 0
    while head<len(que):
        node = que[head]
        head += 1
        for v in graph[node]:
            if dis_ito[v] == -1:
                dis_ito[v] = dis_ito[node]+1
                que.append(v)
    return max(dis_ito)

# 輸入範例  edges = [(1, 2), (1, 3), (3, 4), (3, 5)]
edges = []
n = int(input())
for _ in range(n-1):
    a,b = map(int,input().split())
    edges.append((a, b))

# 計算並輸出樹的直徑
diameter = tree_diameter(edges)
print(diameter)
```

Info:

方法 3:

```
graph = {}
n = int(input())
for _ in range(n-1):
    a,b = map(int,input().split())
    if a not in graph:
        graph[a] = [b]
    else:
        graph[a].append(b)
    if b not in graph:
        graph[b] = [a]
    else:
        graph[b].append(a)

dis_1to = [-1] * (n+1)
dis_1to[1] = 0
que = [1]
head = 0
while head<len(que):
    node = que[head]
    head += 1
    for v in graph[node]:
        if dis_1to[v] == -1:
            dis_1to[v] = dis_1to[node]+1
            que.append(v)

i = dis_1to.index(max(dis_1to))

dis_ito = [-1] * (n+1)
dis_ito[i] = 0
que = [i]
head = 0
while head<len(que):
    node = que[head]
    head += 1
    for v in graph[node]:
        if dis_ito[v] == -1:
            dis_ito[v] = dis_ito[node]+1
            que.append(v)

print(max(dis_ito))
```

Question 347

Problem S
迴圈

Info:

方法 1 解題思路:

- 我們首先構建圖的鄰接表並計算每個節點的度數。
- 然後，我們遍歷所有節點，對於每個未訪問且度數為 2 的節點，使用 DFS 找到其所在的連通子圖。
- 若該子圖中的所有節點度數都為 2，則該子圖為循環圖。
- 最後，返回所有符合條件的循環圖數量。

在此程式碼中，component 用於儲存 DFS（深度優先搜尋）探索到的同一連通子圖中的所有節點。其主要用途如下：

1. **確認連通子圖是否為循環圖**：
    - 當我們找到一個尚未被訪問且度數為 2 的節點時，從該節點執行 DFS，收集與其相連的所有節點，形成一個連通子圖，並將這些節點存入 component 列表中。
    - DFS 結束後，component 中的節點即為這個連通子圖的所有節點。
2. **檢查條件**：
    - 程式會檢查 component 中每個節點的度數是否均為 2，以確認該連通子圖是否滿足「循環圖」的條件。
    - 若 component 中所有節點的度數都為 2，則此連通子圖為一個循環圖，並將計數器 cycle_count 增加 1。

因此，component 的作用是幫助我們檢查一個連通子圖是否為循環圖。如果不使用 component，我們無法確認 DFS 遍歷的子圖中所有節點的度數是否都為 2。

Info:

方法 1

```python
from collections import defaultdict

def count_cycle_graphs(n, edges):
    # Step 1: Initialize graph and degree count
    graph = defaultdict(list)
    degree = [0] * (n + 1)
    # Step 2: Build the graph and degree count
    for u, v in edges:
        graph[u].append(v)
        graph[v].append(u)
        degree[u] += 1
        degree[v] += 1
    # Step 3: Function to perform DFS and return nodes in a component
    def dfs(node, visited):
        stack = [node]
        component = []
        while stack:
            u = stack.pop()
            if not visited[u]:
                visited[u] = True
                component.append(u)
                for v in graph[u]:
                    if not visited[v]:
                        stack.append(v)
        return component
    # Step 4: Count cycle graphs
    visited = [False] * (n + 1)
    cycle_count = 0

    for i in range(1, n + 1):
        if not visited[i] and degree[i] == 2:
            component = dfs(i, visited)
            # Check if all nodes in the component have degree 2
            if all(degree[node] == 2 for node in component):
                cycle_count += 1

    return cycle_count

# 讀取輸入
n, m = map(int, input().split())
edges = []
for _ in range(m):
    u, v = map(int, input().split())
    edges.append((u, v))

# Output result
print(count_cycle_graphs(n, edges))
```

Info:

方法 2
程式碼解析：尋找無向圖中的循環
程式碼目的
這段 Python 程式碼旨在找出給定無向圖中的循環數量。
核心概念

- **無向圖**：一種圖結構，其中邊沒有方向性。
- **循環（Cycle）**：一條從一個節點出發，經過若干個節點後又回到起點的路徑。
- **深度優先搜索（DFS）**：一種圖的搜索算法，從起始節點開始，深度優先地探索圖中的節點。

程式碼解說

1. `find_cycles` **函數**
    - dfs 遞迴函數
        - visited 字典：用來標記已訪問過的節點，避免重複訪問。
        - path 列表：用來記錄當前搜索路徑上的節點。
        - cycle_count 變數：用來計數找到的循環數量。
        - 遞迴邏輯：
            * 將當前節點標記為已訪問。
            * 將當前節點加入路徑。
            * 對每個未訪問的鄰居節點遞迴調用 dfs 函數。
        - 循環檢測：
            * 如果遇到已經在路徑中的節點，且不是起點節點，則找到一個循環。
            * 檢查循環中的所有節點是否度數都為 2，以確保是簡單循環。
            * 如果是有效循環，則增加 cycle_count。
        - 回溯：從路徑中移除當前節點，以便繼續探索其他分支。
2. **主程式**
    - 讀取圖的節點數 n 和邊數 m。
    - 建立一個默認字典 graph 來表示圖，其中鍵是節點，值是該節點的鄰接節點列表。
    - 對於每條邊，將兩個端點互相添加到鄰接表中。
    - 呼叫 find_cycles 函數來計算循環數量。
    - 打印結果。

關鍵點

- **DFS 算法**：核心算法是深度優先搜索，逐層深入探索圖的節點。
- **循環檢測**：透過檢查當前節點是否已經在路徑中來判斷是否形成循環。
- **簡單循環驗證**：確保循環中的每個節點的度都為 2，以避免重複計數。
- **回溯**：遞迴函數在探索完一個分支後，需要回溯到上一個節點，繼續探索其他分支。

通過以上步驟，該程式可以有效地找到給定無向圖中的循環數量。

Info:

方法 2

```python
from collections import defaultdict

def find_cycles(graph):
    def dfs(node, visited, path):
        nonlocal cycle_count
        visited[node] = True
        path.append(node)

        for neighbor in graph[node]:
            if neighbor not in visited:
                if dfs(neighbor, visited, path):
                    return True
            elif neighbor == path[0] and len(path) > 2:
                # 找到一個循環，且所有節點的度都為2
                if all(len(graph[n]) == 2 for n in path):
                    cycle_count += 1
                return True

        path.pop()
        return False

    cycle_count = 0
    visited = {}
    for node in graph:
        if node not in visited:
            dfs(node, visited, [])
    return cycle_count

n, m = map(int, input().split())
graph = defaultdict(list)
for _ in range(m):
    u, v = map(int, input().split())
    graph[u].append(v)
    graph[v].append(u)

result = find_cycles(graph)
print(result)
```

Info:

方法 3

```
n,m = map(int,input().split())

graph = {i:[] for i in range(1,n+1)}
visited = []
for i in range(m):
    a,b = map(int,input().split())
    graph[a].append(b)
    graph[b].append(a)

ans = 0
for i in range(1,n+1):
    if i not in visited:
        arr = []
        que = [i]
        head = 0
        while head<len(que):
            node = que[head]
            head += 1
            for v in graph[node]:
                if v not in visited:
                    visited.append(v)
                    que.append(v)
                    arr.append(v)
        if arr and all(len(graph[num]) == 2 for num in arr):
            ans += 1
            # print(arr)

print(ans)

'''
5 4
1 2
3 4
5 4
3 5
'''
```