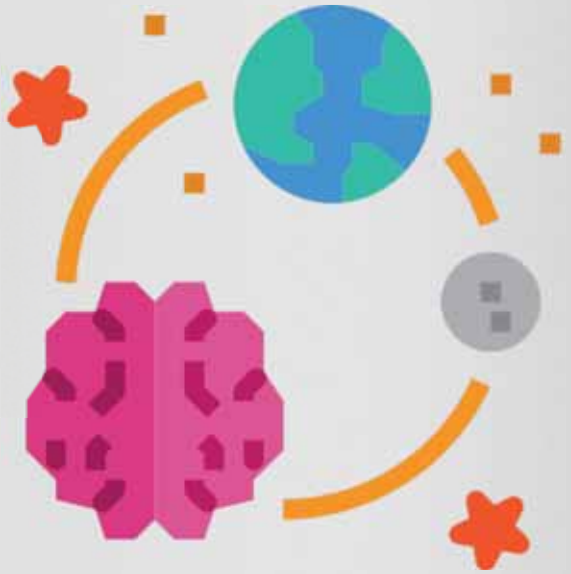
TOI推廣計畫

解題-關鍵星



Icon made by [https://www.flaticon.com/authors/itim2101] from www.flaticon.com

題目

在宇宙中，若沒有星圖，便很容易在虛空中迷失。因此現有的航空模式，大多是藉由星圖往返兩顆星球。假如有人想往返A、B兩顆星球但是沒有星圖指引路線，此時即使A、B兩顆星球直線距離很近，人們大多也會為了保險起見選擇$A-C-B$這樣的路線，其中$A-C$與$C-B$是有星圖指引的。因此某些具有大量星圖的星球，便很容易成為熱門中轉星。

而中轉星裡面還有更特別的是：關鍵星。關鍵星與中轉星一樣，會是人們星際旅行時會經過的星球，但是不一樣的地方是：若是沒有這顆關鍵星，有些星球可能就會再也無法抵達（除非探索出新的航行路線）。亦即：**若有A、B兩顆星球藉由$A-C-B$的路線進行往返，此時，若C星球不復存在，使得A、B星球再也找不到另一條路徑進行往返，則我們稱C星球為關鍵星**。

有一個青年創業家，他想要對星際航空方面進行投資。因此他對進行了研究，而後發現：關鍵星也許是他投資的最佳目標！可以請你藉由現有星圖，找出適合投資的關鍵星嗎？

輸入格式

每筆測試資料第一列有**兩個整數*V*、*E*，*V*表示星球數，*E* (*V* - 1 ≤*E* ≤*V*(*V* - 1)/2) 為已知的星圖數**。星球分別編號為0至V - 1。接下來有E列，每列有兩個數字V_1及V_2，表示編號V_1及V_2的兩顆星球具有星圖指引，可以直接抵達。

註：任意兩星球V_1及V_2，必定具有至少一條路徑進行往返。

輸出格式

每筆測資共輸出一列，為所有關鍵星編號由大至小排序。星球編號間由空格區隔。如果沒有關鍵星，輸出**FAIL**。**最後須有換行字元**。

輸入範例	輸出範例
6 5 0 1 1 2 2 3 3 4 4 5	4 3 2 1

解題重點:

關節點

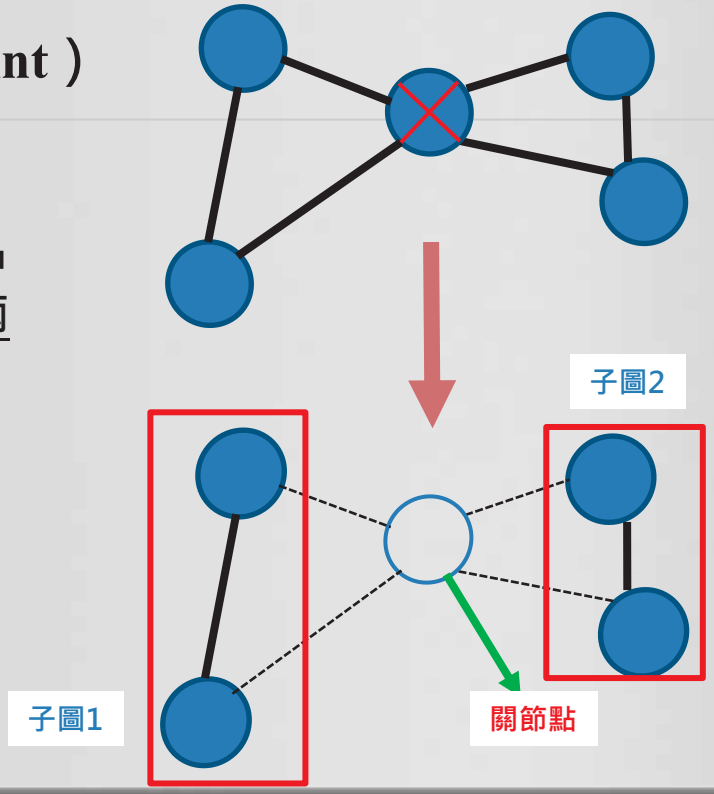
Icon made by [https://www.flaticon.com/authors/photo3idea-studio] from www.flaticon.com

◆ 關節點（Articulation Point）

➢ **定義：**

若將一節點 N 自一張無像圖中移除，會使得此張圖分離成兩個或以上的子圖，則稱之為關節點。

移除之前，圖中任兩點皆連通
移除之後，至少有兩點成為不連通狀態。

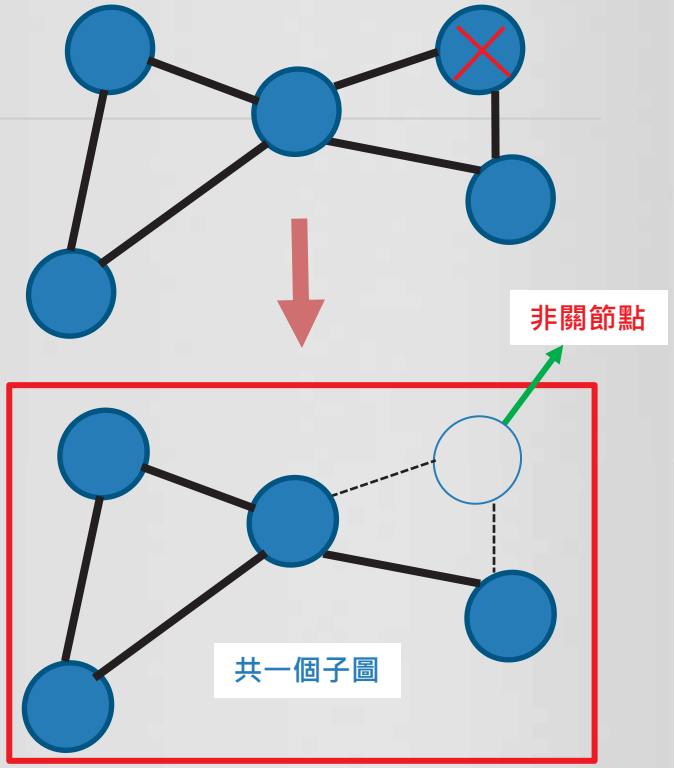


◆ 關節點（Articulation Point）

➢ **定義：**

若將一節點 N 自一張無像圖中移除，會使得此張圖分離成兩個或以上的子圖，則稱之為關節點。

移除之前，圖中任兩點皆連通
移除之後，至少有兩點成為不連通狀態。



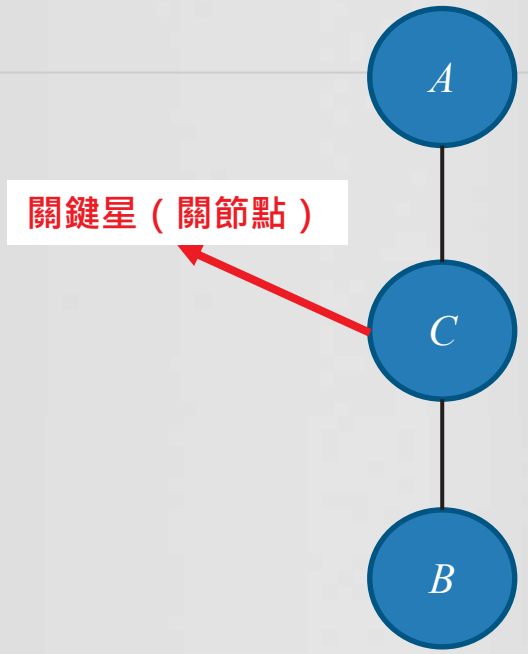
◆ 關節點（Articulation Point）

➢ 題目轉換：

若 *C* 星球不復存在，使得*A*、*B*星球再也找不到另一條路徑進行往返，則我們稱 *C* 星球為關鍵星。

移除 *C* 會使得*A*、*B*不連通

C 為關節點

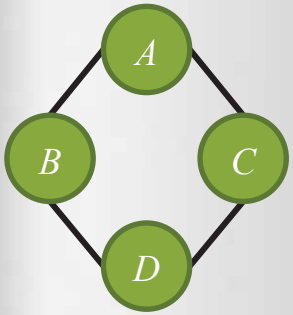


◆ 關節點（Articulation Point）

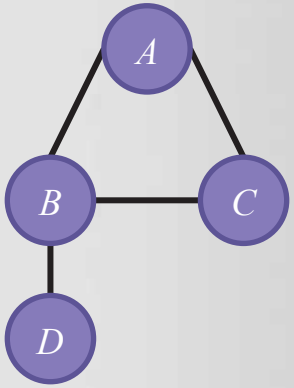
➢ 做法：(1) 找關節點又可以看作是找環

找得到環 → 存在替代路線 → 可以繞過關節點。(因此就不算是關節點)

找不到環 → 不存在替代路線 → 繞不過關節點。



檢查 *A*、*D* 是否在同一個環上：
是 → *A*、*D*必存在至少兩條路徑使其相通（*ADB*、*ACD*）
→ 對*A*、*D*而言關節點不存在。
否 → 至少存在一個點繞不開（*B*）
→ 該點為關節點。



◆ 關節點（Articulation Point）

➢ 做法：

(2) 利用DFS Tree 尋找環
取樹上任一點*N*，當此點的祖先與子孫利用tree edge連通，則顯然經過此點*N*。若*N*非關節點，則其祖先與其子孫之間必然存在back edge。

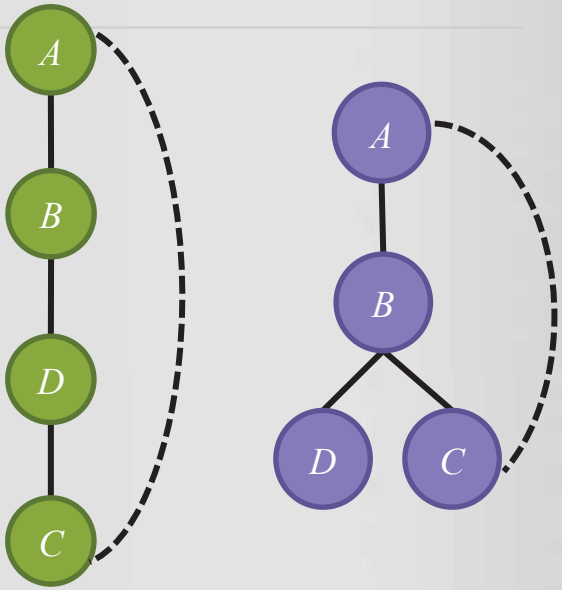
存在一條back edge，僅代表對某一個子樹而言 *N* 並非關節點。若要檢查其是否為整張圖上的關節點，則必須要檢查其所有子樹

◆ 關節點（Articulation Point）

➢ 做法：

祖先與每一棵子樹之間都有 back edge，則此點不是關節點。

祖先與其中一棵子樹之間缺少 back edge，則此點就是關節點。



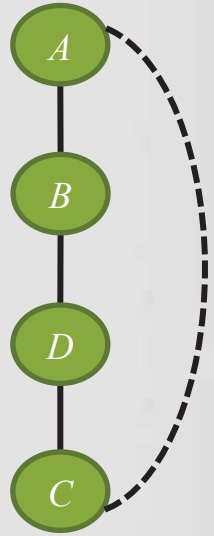
◆ 關節點（Articulation Point）

Root：存在兩棵以上子樹，則為關節點
（因為子樹不可能繞過樹根互通有無）
Leaf：必不為關節點

➢ 做法：

祖先與每一棵子樹之間都有 back edge，則此點不是關節點。

子樹*C* 存在back edge通往*A*(Root)，因此子樹*B*、*D*也存在back edge
→對*B*、*D*而言，其唯一子樹皆存在back edge，因此皆非關節點
A 為Root，只有一棵子樹，非關節點。
C 為Leaf，非關節點。



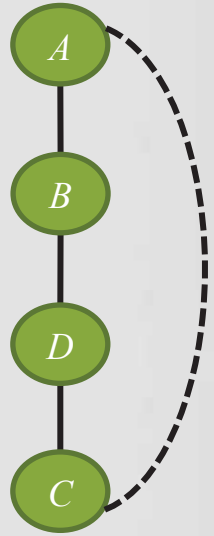
◆ 關節點（Articulation Point）

Root：存在兩棵以上子樹，則為關節點
（因為子樹不可能繞過樹根互通有無）
Leaf：必不為關節點

➢ 做法：

祖先與每一棵子樹之間都有 back edge，則此點不是關節點。

子樹*C* 存在back edge通往*A*(Root)，因此子樹*B*、*D*也存在back edge
→對*B*、*D*而言，其唯一子樹皆存在back edge，因此皆非關節點
A 為Root，只有一棵子樹，非關節點。
C 為Leaf，非關節點。



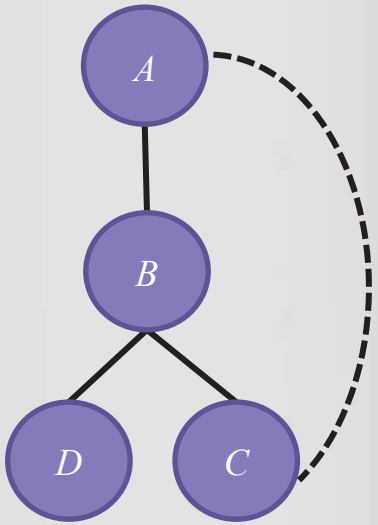
◆ 關節點（Articulation Point）

Root：存在兩棵以上子樹，則為關節點
（因為子樹不可能繞過樹根互通有無）
Leaf：必不為關節點

➢ **做法：**

祖先與其中一棵子樹之間缺少back edge，則此點就是關節點。

對*B*而言，其子樹*D*與其祖先*A*並不存在back edge。因此*B*為關節點。
*A*為Root，只有一棵子樹，非關節點。
C、*D*為Leaf，非關節點。



◆ 關節點（Articulation Point）

➢ 程式碼 (Graph Traversal / DFS)

(1) 宣告

```
std::vector<int> adj[20005];
std::vector<int> ans;
int visit[20005]= {};
int low[20005]= {};
int t = 0;
```

//adjacency list
//儲存答案
//儲存各節點遍歷時刻

//現在的時刻

每個節點不停走tree edge/back edge 所能觸及的遍歷時刻最小的祖先。

◆ 關節點（Articulation Point）

➢ 程式碼(Graph Traversal / DFS)

(2) 遍歷

```
t+=1;
visit[cur] = low[cur] = t;
int Num_child = 0;
bool ap = false;
```

遍歷時刻+=1

子樹數量

是否為關節點

◆ 關節點（Articulation Point）

➢ 程式碼(Graph Traversal / DFS)

(2) 遍歷

```
int Num = adj[cur].size();
for (int i=0; i<Num; i+=1)
{
    int dest = adj[cur][i];
    if (dest==pre)
        continue;
    if (visit[dest]!=0)
    {
        low[cur] = std::min(low[cur], visit[dest]);
    }
    else
    {
        Num_child+=1;
        DFS(cur, dest, adj, low, visit, t, ans);
        low[cur] = std::min(low[cur], low[dest]);
        if(low[dest]>=visit[cur]) ap=true;
    }
}
```

如果目標節點已經遍歷，直接修正low的值

子樹數量+=1
繼續往下遍歷，遍歷後修正low值
修正完檢查子樹的low值是否有比當前節點的visit來的大。
是→表示該子樹無法繞過當前節點回到祖先，因此當前節點為關節點。

◆ 關節點（Articulation Point）

➢ 程式碼(Graph Traversal / DFS)

(2) 遍歷

pre==cur為第一次呼叫時，此時當前節點為root，要檢查子樹數量是否大於1。

```
if ((pre==cur&&Num_child>1)||(cur != pre && ap))
    ans.push_back(cur);
```

否則檢查ap是否為真即可

◆ 關節點（Articulation Point）

➢ 程式碼(Graph Traversal / DFS)

```
void DFS(int pre, int cur, std::vector<int> adj[], int low[], int visit[], int &t, std::vector<int> &ans)
{
    t+=1;
    visit[cur] = low[cur] = t;
    int Num_child = 0;
    bool ap = false;
    int Num = adj[cur].size();
    for (int i=0; i<Num; i+=1)
    {
        int dest = adj[cur][i];
        if (dest==pre)
            continue;
        if (visit[dest]!=0)
        {
            low[cur] = std::min(low[cur], visit[dest]);
        }
        else
        {
            Num_child+=1;
            DFS(cur, dest, adj, low, visit, t, ans);
            low[cur] = std::min(low[cur], low[dest]);
            if(low[dest]>=visit[cur]) ap=true;
        }
    }
    if ((pre==cur&&Num_child>1)||(cur != pre && ap))
        ans.push_back(cur);
}
```

◆ 範例程式

答案需要大至小輸出，所以記得對答案進行排序。

```
#include<bits/stdc++.h>
int main()
{
    std::vector<int> adj[20005];
    std::vector<int> ans;
    int visit[20005]= {};
    int low[20005]= {};
    int t = 0;

    int V,E;
    scanf("%d%d", &V, &E);
    for(int i=0; i<E; i+=1)
    {
        int v1,v2;
        scanf("%d%d",&v1, &v2);
        adj[v1].push_back(v2);
        adj[v2].push_back(v1);
    }
    DFS(0,0, adj, low, visit, t, ans);
    int n=ans.size();
    std::sort(ans.begin(),ans.end());
    for(int i=n-1; i>=0; i-=1)
    {
        if(i!=n-1)
            printf(" ");
        printf("%d", ans[i]);
    }
    if (n==0)
        printf("FAIL");
    printf("\n");
}
```

```
void DFS(int pre, int cur, std::vector<int> adj[], int low[], int visit[], int &t, std::vector<int> &ans)
{
    t+=1;
    visit[cur] = low[cur] = t;
    int Num_child = 0;
    bool ap = false;
    int Num = adj[cur].size();
    for (int i=0; i<Num; i+=1)
    {
        int dest = adj[cur][i];
        if (dest==pre)
            continue;
        if (visit[dest]!=0)
        {
            low[cur] = std::min(low[cur], visit[dest]);
        }
        else
        {
            Num_child+=1;
            DFS(cur, dest, adj, low, visit, t, ans);
            low[cur] = std::min(low[cur], low[dest]);
            if(low[dest]>=visit[cur]) ap=true;
        }
    }
    if ((pre==cur&&Num_child>1)||(cur != pre && ap))
        ans.push_back(cur);
}
```