圖論 II

Graph Theory II

2023 南 11 校聯合寒訓 - 演算法課程

2023.02.02

陳俊安 Colten

主辦單位：

SC ST

NHDK Ten Point Round

協辦單位：

iTSA 教育部 智慧創新跨域 人才培育計畫

成功大學資訊工程學系暨研究所 Department of Computer Science and Information Engineering

贊助單位：

ytp 少年圖靈計畫 Young Turing Program

奧義智慧科技 Powered by CyCraft

TEAMT5

DEVCORE

這一個單元的重點

- Disjoint Set
- 二分圖

Disjoint Set

- 非交集資料結構
- 簡單來說， Disjoint Set 這個資料結構能幫我們維護很多個沒有交集的集合
- Disjoint Set 要實作的主要的操作為
- 合併集合 與 查詢元素 屬於的集合

Disjoint Set 例題

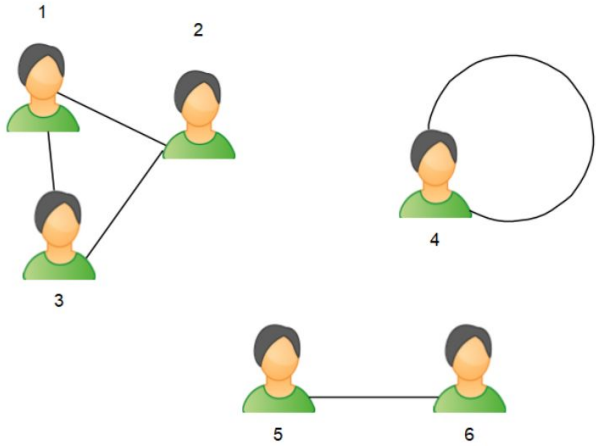
假設 a 與 b 是好朋友，c 與 b 也是好朋友

那麼 a 與 c 也會是好朋友

現在給你 M 段關係，Q 筆查詢

每一筆查詢將詢問 X 跟 Y 是否為好朋友

這是一個經典的 Disjoint Set 題目



Disjoint Set 實作：我們為每一個朋友圈設立一個老大

假設現在每一個朋友圈都有一個老大

那如果我們現在有一個工具可以查詢每一個人的老大是誰

當我要查詢兩個人是否是同一個朋友圈的時候我們就只要確定

這兩個人的老大是不是同一個人，因此我們就可以用並查集來維護狀態

Disjoint Set：Init（初始化）

一開始並須先將 DSU 初始化，在還沒有做任何操作之前

所有人的老大就是自己（第 i 個人的老大就是 i）

因此如果你有 N 個人的話，你當前就會有 N 個群體（Group）

```
for(int i=1;i≤n;i++) dsu[i] = i; // dsu[i] 表示 i 的老大是誰
```

Disjoint Set：Find（尋找老大）

如果我們想要找到某一個人的老大是誰，我們應該怎麼做呢？

我們先試著想想，假設有一段關係是 1 -> 2 -> 3

換句話說：1 的老大是 2，2 的老大是 3，所以 1 的老大會是 3

這時，3 的老大也會是 3（自己），因為目前 3 就是那一個群體的老大

如此一來我們可以發現，尋找某個人的老大的方式就出現了！

我們只需要一直跟隨關係下去，找到某個人的老大是自己就表示找到了

Disjoint Set：Find（尋找老大）

如何跟隨關係？

當我們發現 i 的老大不是自己的時候，我們就去找 i 的老大的老大是誰

這樣一直不斷的下去，直到我們找到，因此我們可以使用 遞迴 來實作

舉例：

如果 i 的老大是 k，k 的老大是 m，m 的老大是 t

那麼因為我們發現 i 的老大不是他自己，因此我們要知道 k 的老大是誰

k 的老大也不是他自己，因此我們必須找 m 的老大是誰

m 的老大也不是他自己，因此我們要找 t 的老大是誰

t 的老大是自己！因此，i 的老大就是 t

Disjoint Set：Find（尋找老大）

```
 7 int find(int n)
 8 {
 9     if( dsu[n] == n ) return n; // 找到老大了
10
11     else
12     {
13         return find(dsu[n]); // n 的老大不是他自己，因此我們需要知道 n 的老大 (dsu[n]) 的老大是誰
14     }
15 }
```

Disjoint Set：Find（尋找老大）優化 - 路徑壓縮

如果我們現在有一段關係是：

A -> B -> C -> D -> E

那難道我們每次查詢 A 的老大，都要這一條鍊都跑過一次嗎？

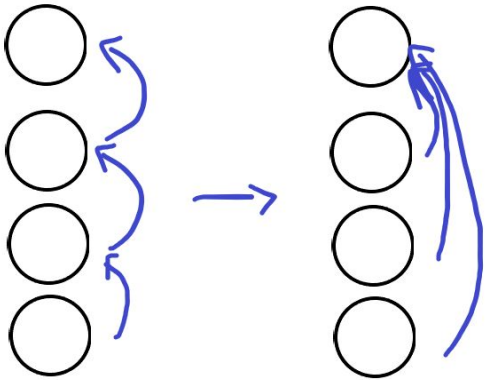
是否可以做一些優化呢？

Disjoint Set：Find（尋找老大）優化 - 路徑壓縮

可能有些學員有發現到，在find函式中，因為只是找出他最大的老大是誰，但卻沒有做任何的更改，這就會導致每次find都需要重新遞迴一次

那如果我們在查詢的過程中，將經過的點都順便把他們的老大變成最大的老大，這樣之後要查詢就可以省去很多多餘的步驟了

```
int find(int i) //查找
{
    if (dsu[i] == i)
        return i;
    else
        return dsu[i] = find(dsu[i]);
}
```



Disjoint Set：Find（尋找老大）優化 - 路徑壓縮

不做這一件事情的話每一次查詢的時間複雜度最恐怖會退化到每一次 O(元素個數)

Disjoint Set：Union

如果我想要合併 A 跟 B 這兩個人的 Group 呢？

Disjoint Set：Union

如果現在我想要將 A 所在的 Group 合併到 B 的 Group 去

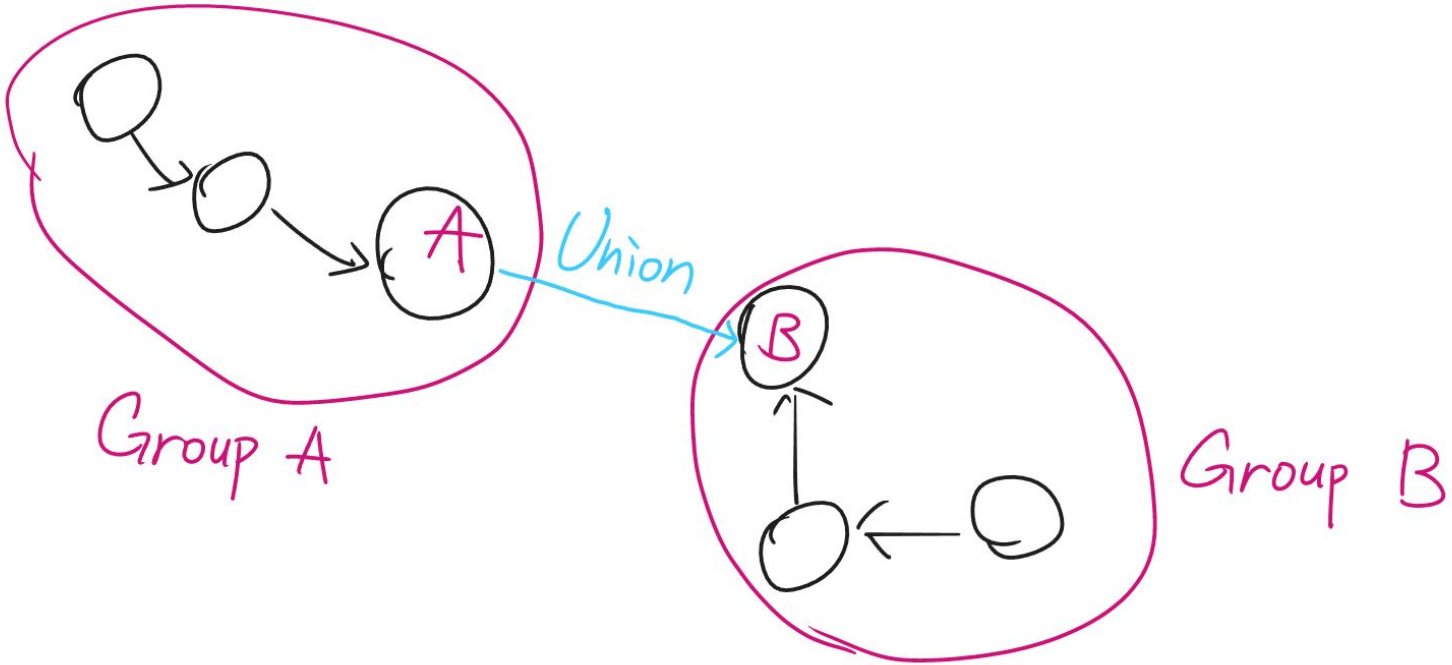
假設 Group A 的老大是 K

那麼如果這 Group A 合併到 Group B 也就意味著：

K 的老大將變成 B 的老大

因此我們 Disjoint Set 的合併過程只需要實作上面這件事就可以完成合併了

Disjoint Set : Union



Disjoint Set：Union 實作

```
 7 void unite(int a,int b)
 8 {
 9     a = find(a); // 找到 a 的老大
10
11     dsu[a] = find(b); // 注意，不能寫 dsu[a] = b，因為當你如果發生邏輯矛盾時會無限遞迴
12
13     // 將 a 的老大的老大設成 b 的老大
14
15     return;
16 }
```

Disjoint Set：Union 優化 - 啟發式合併

通常只做路徑壓縮就已經在絕大多數的情況下已經夠快了

但是啟發式合併這個優化技巧在未來大家更深入圖論的領域時也會碰到

因此在這邊提一下

啟發式合併

- 把每一個集合的深度平衡好
- 好讓我們 find 的時候，時間複雜度可以好一點
  - 因為 find 的時間複雜度跟 集合的深度有關
  - 因此如果我們能控制好深度，就能減少 find 的時間

Disjoint Set：Union 優化 - 啟發式合併

```
 3 void unite(int a,int b)
 4 {
 5     a = find(a);
 6     b = find(b);
 7
 8     if( a == b ) return; // a 跟 b 已經在同一個集合了，不需要合併
 9
10     if( sz[a] > sz[b] ) swap(a,b); // 用小群體去合併大群體
11
12     sz[b] += sz[a]; // 合併後，大群體的人數加上小群體的人數
13
14     dsu[a] = b;
15 }
```

Disjoint Set 的時間複雜度

那麼 Disjoint Set 的時間複雜度會是如何呢？

Disjoint Set 的時間複雜度

Disjoint Set 在兩種優化都有做的情況下 查詢與合併的時間複雜度皆為$O(\alpha(n))$

$\alpha(n)$ 是指 **反**阿克曼函數，（對於任何元素數目 n、$\alpha(n)$ 都小於 5）

因此基本上，Disjoint Set 的查詢與合併的操作基本上可以當常數來看待

不過上述這句話的前題是建立在你已經把所有該優化的東西都優化的時候(路徑壓縮與啟發式合併)

備註：如果只作其中一種優化，那麼時間複雜度會是O(log n)，因此其實基本上做其中一種就夠了

由於 Disjoint Set 的時間複雜度較難以估計

因此會需要大量的數學來輔助證明他的時間複雜度

對證明有興趣的可以參考 *Fredman* 和 *Saks* 在 1989 年提出的證明方式

練習題：ITMO Academy: pilot course » Disjoint Sets Union » Step 1 pA. Disjoint Sets Union

來試試看實作 Disjoint Set 的基本操作吧！

多紀錄一些狀態的 dsu：ITMO Academy: pilot course » Disjoint Sets Union » Step 1 pB. Disjoint Sets Union 2

不用想的太難，就像是啟發式合併多紀錄群體大小那樣而已 ><

時光倒流 : Atcoder Beginner Contest 229 pE. Graph Destruction

題目跟你說，現在他會從 1 ~ N 依序刪點，只要點 i 不見了

那麼跟點 i 相連的邊就會不見，問你每一次刪完點後會有幾個連通塊

如果這題不是刪邊，是加邊

你會發現，如果這題變成每一個時間點加一個點進來

只要某一條邊的兩個點都已經存在，就把這一條邊加進來

那這題就會變成一個超級單純的 Disjoint Set 基本操作題

那我們把時光逆流，是不是就符合我們要的了？

可以確定的是，圖最後一定是完全沒有東西

但如果我們把時光逆流，那麼我們就變成

我們依序把 N ~ 1 這些點加回來，邊也依序加回來

如此一來就變成一個基本的 Disjoint Set 問題了

也就是說，我們反著做，從 N 開始求答案，做回 1

我們先將圖的所有邊用 vector 存起來

```
43      vector <pair<int,int>> path;
44
45      while(m--)
46      {
47          int a,b;
48
49          cin >> a >> b;
50
51          path.push_back({min(a,b),max(a,b)});
52      }
53
```

將邊排序，讓節點編號大的邊在後面

這裡也可以開 n 個 queue 來維護這件事，看個人喜好

```
54      sort(path.begin(),path.end());
```

開始時光逆流

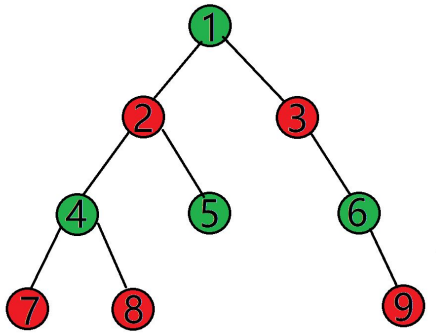
```
58      int group = 0; // 一開始沒有任何一個連通塊
59
60      for(int i=n;i≥1;i--)
61      {
62          // 由於先前我們將 ( i + 1 ) 這個點加回來了，檢查看看有沒有可以加回來的邊
63
64          // 由於一開始 1 號這個點就毀先被刪掉，所以他的邊我們沒有必要加回來
65
66          while( path.size() > 0 && path[path.size()-1].first > i && path[path.size()-1].second > i )
67          {
68              int a = path[path.size()-1].first , b = path[path.size()-1].second;
69
70              if( find(a) ≠ find(b) )
71              {
72                  unite(a,b); // a 跟 b 屬於不同的連通塊，因此將 a 與 b 合併
73
74                  group--; // 如果兩個不同的連通塊合併了，那麼連通塊的數量會減少 1
75              }
76
77              path.pop_back();
78          }
79
80          ans.emplace_back(group);
81
82          group++; // 將點 i 加回來，因此連通塊多了一個
83      }
```

最後輸出答案

```
85      for(int i=ans.size()-1;i≥0;i--) // 由於我們從最後面開始做，所以一開始的答案會在最後面
86      {
87          cout << ans[i] << "\n";
88      }
```

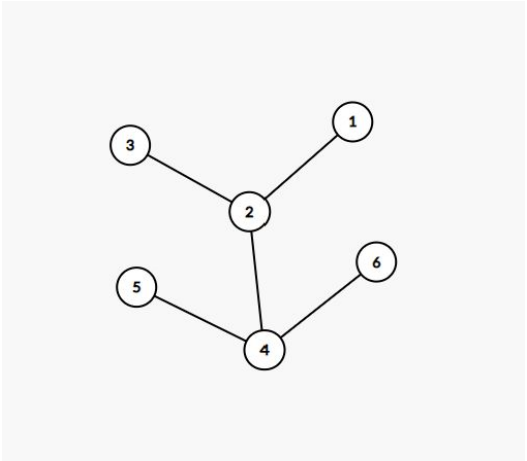
二分圖

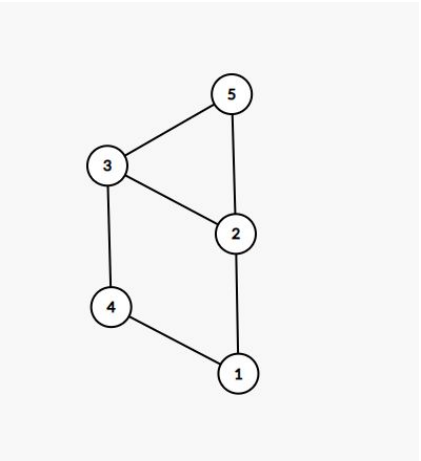
- 如果一張 連通 的圖可以被我們用兩種不同的顏色把點塗色，且塗完色之後，如果兩個點之間有邊，那麼他們兩個不是同個顏色的
- 這樣子的圖我們就叫做 二分圖



二分圖如何判斷 - DFS

- DFS！隨便挑一個起點塗一個顏色
- 以這一個點為基準看看有沒有辦法塗出滿足條件的狀態



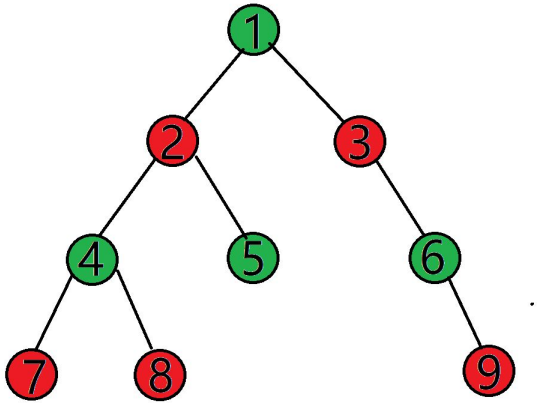


二分圖如何判斷 - DFS

```cpp
int c[200005];

vector<int> e[200005];

bool ok = 1;

void dfs(int idx,int color)
{
    c[idx] = color;
    for( auto i : e[idx] )
    {
        if( c[i] == 0 ) // 這個還沒被塗過顏色
        {
            if( c[idx] == 1 ) dfs(i,2);
            else dfs(i,1);
        }
        else if( c[i] == c[idx] ) ok = 0;
    }
}
```

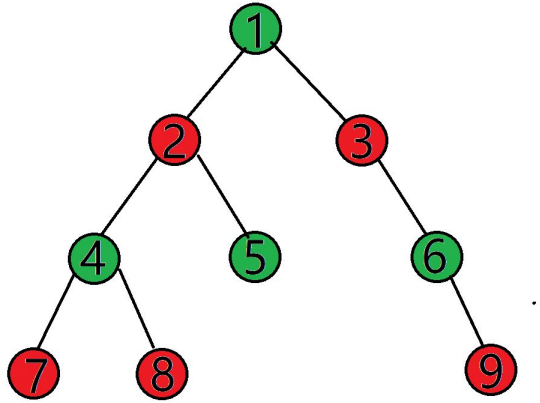
二分圖如何判斷 - Disjoint Set

- 右邊這張圖是一張二分圖
- 第一個點是綠色的
- 那這次改把它塗成紅色的
- 也一定有解



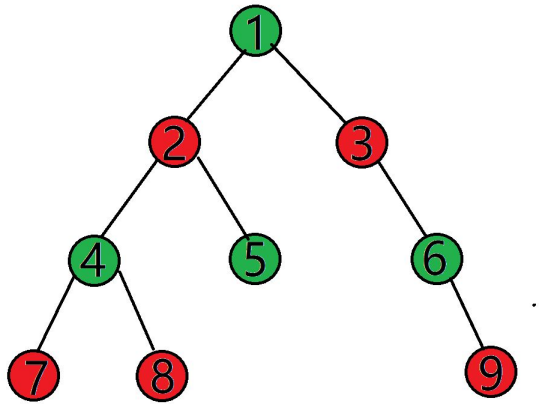
二分圖如何判斷 - Disjoint Set

- 開 2n 長度的陣列
- 1 ~ n 表示紅色
- (n + 1) ~ 2n 表示綠色
- 兩點之間有邊
  - 紅色 接 綠色
  - 綠色 接 紅色



二分圖如何判斷 - Disjoint Set

- 如果一張圖是二分圖
- 那麼同一個點的綠色跟紅色
- 必須是不同的連通塊
- 否則矛盾，該圖不是二分圖



二分圖如何判斷 - Disjoint Set

```
21
22 int main()
23 {
24     int n,m;
25     cin >> n >> m;
26
27     init(2*n);
28     for(int i=0;i<m;i++)
29     {
30         int x,y;
31         cin >> x >> y;
32
33         unite(x+n,y);
34         unite(x,y+n);
35     }
36
37     bool ok = 1;
38     for(int i=1;i≤n;i++) if( find(i) == find(i+n) ) ok = 0;
39 }
```