CHAPTER

02

實用的資料結構

2.1 併查集

若部落過於龐大，則部落成員見面也有可能不認識。已知某個部落的親戚關係圖，任意給出其中兩個人，判斷他們是否是親戚。規定：①若 x 和 y 是親戚，y 和 z 是親戚，則 x 和 z 也是親戚；②若 x 和 y 是親戚，則 x 的親戚也是 y 的親戚，y 的親戚也是 x 的親戚。

怎樣快速判斷兩個人是否是親戚呢？因為以上規定中的第①條是傳遞關係，第②條相當於兩個集合的合併，因此對該問題可以用併查集輕鬆解決。併查集是一種樹狀資料結構，用於處理集合的合併及查詢問題。

1．演算法步驟

（1）初始化。將每個節點的集合識別符都初始化成自己。

（2）搜尋。搜尋兩個節點所在的集合，簡稱「找祖先」。搜尋時，首先用遞迴演算法找其祖先，找到祖先（集合識別符為自己）時停止；迴歸時，將從祖先到當前節點路徑上的所有節點的集合識別符都統一為祖先的集合識別符。

（3）合併。若兩個節點的集合識別符不同，則將兩個節點合併為一個集合，合併時只需將一個節點的祖先的集合識別符修改為另一個節點的祖先的集合識別符。擒賊先擒王，只改祖先即可！

碁峰 www.gotop.com.tw

```
    scanf("%d%d",&x,&y);
    Union(x,y);// 合併集合
}
for(i=1;i<=n;i++){// 統計集合數量
    if(i==fa[i])
        ans++;
}
printf("%d\n",ans-1);// 輸出答案
```

訓練 2 方塊棧板

題目描述（POJ1988）：貝西正在玩方塊遊戲，方塊編號為 1 ～ N（$1 \le N \le 30{,}000$），開始時每個方塊都相當於一個棧板。貝西執行了 P 次（$1 \le P \le 100{,}000$）操作，操作類型有兩種：M X Y，將包含 X 的棧板全都移到包含 Y 的棧板的頂部；C X，查詢 X 方塊下方的方塊數量。請統計貝西每次操作的結果。

輸入：第 1 行為單個整數 P，表示操作的次數。第 2 ～ P+1 行，每行都描述一次操作（注意：N 的值不會出現在輸入檔案中，移動操作不會將棧板移動到自己所在的位置）。

輸出：對於每次查詢操作，都輸出統計結果。

輸入範例

```
6
M 1 6
C 1
M 2 4
M 2 6
C 3
C 4
```

輸出範例

```
1
0
2
```

題解：本題包括移動和查詢兩種操作，可以用二維陣列實作方塊的整體移動，但是操作量很大，若一個一個地移動，則會超時。整體移動相當於集合的合併，因此可以藉助併查集快速、高效率地實作，在集合中進行搜尋和合併操作時更新根下方的方塊數量。

1．演算法設計

（1）初始化。每個方塊的集合識別符都初始化成自己。

（2）查詢或者合併。

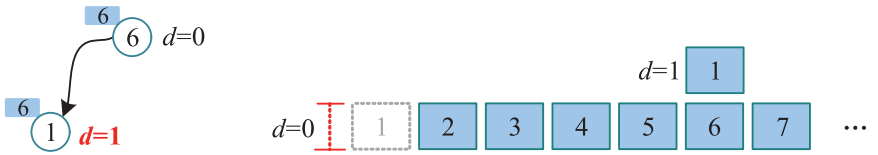
碁峰 www.gotop.com.tw

- C X：查詢方塊 X 的集合識別符，並輸出方塊 X 下方的方塊數量。d[i] 表示方塊 i 下方的方塊數量。查詢方塊 X 的集合識別符，迴歸時將所經過路徑上節點的集合識別符都統一為祖先的集合識別符，將當前節點的 d 值加上其雙親的 d 值。
- M X Y：合併方塊 X、方塊 Y 所在的集合。cnt[i] 表示第 i 個棧板的方塊數量。首先找到方塊 X、方塊 Y 的集合識別符 a、b，然後將方塊 a 的集合識別符修改為 b，fa[a]=b，更新 d[a]=cnt[b]，cnt[b]+=cnt[a]。

2．完美圖解

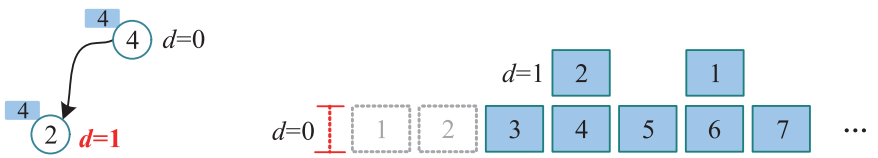
（1）初始化。根據輸入範例，初始時每個方塊的集合識別符都成自己，fa[i]=i；在每個方塊下方都有 0 個方塊，d[i]=0；每個棧都只有 1 個方塊，cnt[i]=1。

（2）合併。M 1 6：將包含方塊 1 的棧板整體移動到包含方塊 6 的棧板的頂部。首先查詢到方塊 1 和方塊 6 的集合識別符為 1、6，然後將方塊 1 的集合識別符修改為 6，fa[1]=6，更新 d[1]=cnt[6]=1，cnt[6]+=cnt[1]=2。



（3）查詢。C 1：查詢在方塊 1 下方有多少個方塊。首先查詢方塊 1 的集合識別符，迴歸時修改集合識別符並將當前節點的 d 值加上其雙親的 d 值，d[1]+=d[6]=1。

（4）合併。M 2 4：將包含方塊 2 的棧板整體移動到包含方塊 4 的棧板的頂部。首先找到方塊 2 和方塊 4 的集合識別符為 2、4，然後將方塊 2 的集合識別符修改為 4，fa[2]=4，更新 d[2]=cnt[4]=1，cnt[4]+=cnt[2]=2。

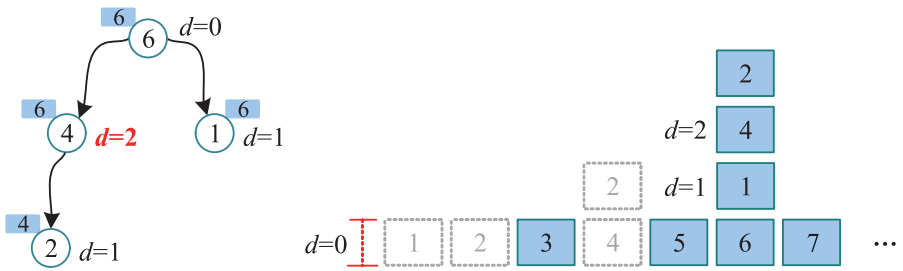


（5）合併。M 2 6：將包含方塊 2 的棧板全都移到包含方塊 6 的棧板的頂部。首先找到方塊 2 和方塊 6 的集合識別符為 4、6，然後將方塊 4 的集合識別符修改為 6，fa[4]=6，更新 d[4]=cnt[6]=2，cnt[6]+=cnt[4]=4。

碁峰 www.gotop.com.tw

注意

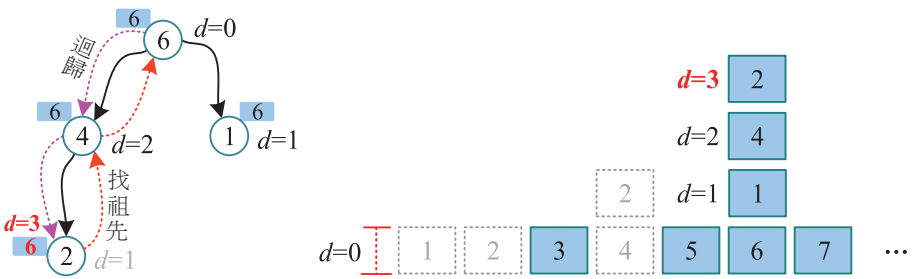
本次只修改祖先的集合識別符，未修改方塊 2 的集合識別符及 *d* 值，下次查詢其集合識別符時才會更新。這正是併查集的妙處。



（6）查詢。C 3：查詢在方塊 3 下方有多少個方塊。查詢到方塊 3 的集合識別符為 3，d[3]=0。

（7）查詢。C 4：查詢在方塊 4 下方有多少個方塊。查詢方塊 4 的集合識別符，迴歸時修改集合識別符並將當前節點的 *d* 值加上其雙親的 *d* 值，d[4]+=d[6]=2。

（8）繼續查詢。C 2：查詢在方塊 2 下方有多少個方塊。查詢方塊 2 的集合識別符，迴歸時修改集合識別符並將當前節點的 *d* 值加上其雙親的 *d* 值，d[2]+=d[4]=3。



3．演算法實作

（1）初始化。

```
void Init(){
    for(int i=1;i<N;i++){
        fa[i]=i;// 每個方塊的集合識別符都為自己
        d[i]=0;// 方塊 i 下方的方塊數量為 0
```

碁峰 www.gotop.com.tw

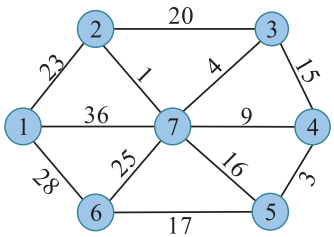
CHAPTER

06

圖論演算法

6.1 最小生成樹

校園網是為學校師生提供資源分享、資訊交流和協同工作的電腦網路。現在需要設計校園網電纜佈線，將各個學校連通起來，如何設計才能使佈線費用最少呢？可以用無向連通圖 $G=(V,E)$ 表示通訊網路，其中 V 表示節點集合，E 表示邊的集合。把各個學校都抽象成圖中的節點，把學校之間的通訊網路抽象為節點與節點之間的邊，邊的權值表示佈線費用，簡稱「邊權」。若在兩個節點之間沒有連線，則代表在這兩個學校之間不能佈線，費用為無窮大，如下圖所示。



對於有 n 個節點的連通圖，只需 $n-1$ 條邊就可以使其連通，要想透過 $n-1$ 條邊保證圖連通，必須不包含回路，所以只需找出 $n-1$ 條權值最小且無回路的邊即可。需要說明以下幾個概念。

- **子圖：**由在原圖中選中的一些節點和邊組成的圖。
- **生成子圖：**由選中的一些邊和所有節點組成的圖，稱為「原圖的生成子圖」。

碁峰 www.gotop.com.tw

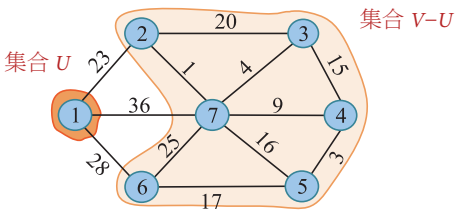
- **生成樹：**若生成的子圖恰好是一棵樹，則稱之為「生成樹」。
- **最小生成樹：**權值之和最小的生成樹。

校園網佈線問題屬於最小生成樹問題，可透過 Prim 演算法或 Kruskal 演算法解決。

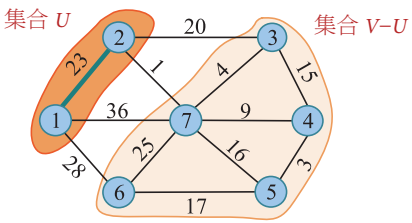
6.1.1 Prim 演算法

找出 $n-1$ 條權值最小的邊很容易，但是怎麼保證無回路呢？若在一個圖中透過深度搜尋或廣度搜尋方式判斷有沒有循環，則工作繁重。這時可以透過**集合避圈法**解決問題。在生成樹的過程中，把已經在生成樹中的節點看作一個集合，把剩下的節點看作另一個集合，從連接兩個集合的邊中選擇一條權值最小的邊即可。

首先任選一個節點，例如選擇節點 1，把它放在集合 U 中，集合 U={1}，剩下的節點為 {2,3,4,5,6,7}，集合 V 是圖中所有節點的集合，如下圖所示。



然後只需看看在連接集合 U 和集合 $V-U$ 的邊中，哪條邊的權值最小，把權值最小的邊關聯的節點加入集合 U。從上圖可以看出，在連接兩個集合的 3 條邊中，1-2 邊權最小，選中這條邊，把節點 2 加入集合 U，此時集合 U={1,2}，集合 $V-U$={3,4,5,6,7}，如下圖所示。

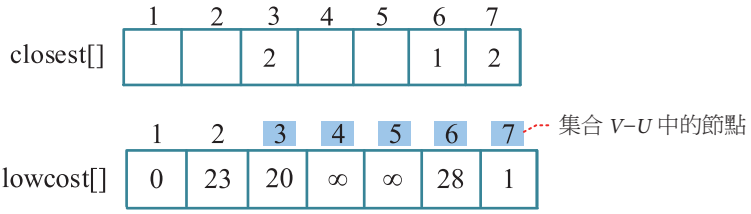


碁峰 www.gotop.com.tw

接著從連接集合 U 和集合 $V-U$ 的邊中選擇一條權值最小的邊。從上圖可以看出，在連接這兩個集合的 4 條邊中，2-7 邊權最小，選中這條邊，把 7 加入集合 U，此時集合 U={1,2,7}，集合 $V-U$={3,4,5,6}。如此進行下去，直到集合 U= 集合 V 時結束，由選中的邊和所有節點組成的圖就是最小生成樹。這就是 Prim 演算法。

直觀地看圖，雖然很容易找出連接集合 U 與集合 $V-U$ 的邊中哪條邊的權值最小，但若在程式中窮舉這些邊，再找最小值，則時間複雜度太高，該怎麼辦呢？可以透過設定兩個陣列巧妙地解決這個問題：closest[j]，表示集合 $V-U$ 中節點 j 到集合 U 的最鄰近點；lowcost[j]，表示集合 $V-U$ 中的節點 j 到集合 U 的最鄰近點的邊權。

例如在上圖中，7 到集合 U 的最鄰近點是 2，記為 closest[7]=2。7 到最鄰近點 2 的邊權為 1，記為 lowcost[7]=1，如下圖所示。



這樣，只需在集合 $V-U$ 中找到 lowcost[] 陣列中最小的節點即可。

1．演算法步驟

（1）令集合 U={u_0}，u_0∈ 集合 V，並初始化 closest[]、lowcost[] 和 s[] 陣列。

（2）在集合 $V-U$ 中搜尋 lowcost[] 陣列中最小的節點 t，即 lowcost[t]= min{lowcost[j]| j∈ 集合 $V-U$}，滿足該公式的節點 t 就是集合 $V-U$ 中連接集合 U 的最鄰近點。

（3）將節點 t 加入集合 U。

（4）若集合 $V-U$ 為空，則演算法結束，否則進行第 5 步。

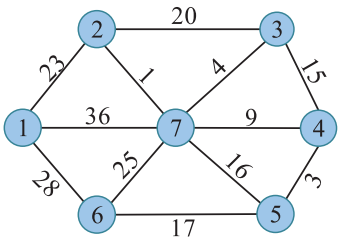
（5）對集合 $V-U$ 中的所有節點 j 都更新 if(C[t][j]<lowcost[j]){lowcost[j]=C[t][j]; closest[j]=t;}，轉向第 2 步。

按照上述步驟，最終可以得到一棵邊權之和最小的生成樹。

碁峰 www.gotop.com.tw

2．完美圖解

圖 G（$G=(V, E)$）是一個無向連通帶權圖，如下圖所示。

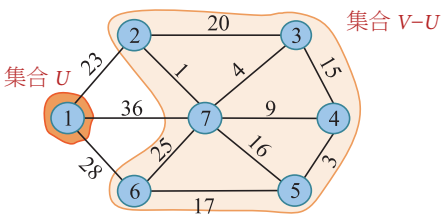


（1）假設 $u_0=1$，令集合 $U=\{1\}$，集合 $V-U=\{2,3,4,5,6,7\}$，TE={}，s[1]=true，初始化 closest[] 陣列：除了節點 1，其餘節點均為 1，表示集合 $V-U$ 中的節點到集合 U 的最鄰近點均為 1。lowcost[] 陣列儲存節點 1 到集合 $V-U$ 中節點的邊權。closest[] 和 lowcost[] 陣列如下圖所示。

	1	2	3	4	5	6	7
closest[]		1	1	1	1	1	1

	1	2	3	4	5	6	7
lowcost[]	0	23	∞	∞	∞	28	36

初始化後如下圖所示。



（2）搜尋 lowcost[] 陣列中最小的節點。在集合 $V-U=\{2,3,4,5,6,7\}$ 中依照貪心策略搜尋 lowcost[] 陣列中最小的節點 t。找到的最小值為 23，對應的節點 $t=2$，如下圖所示。

	1	2	3	4	5	6	7
lowcost[]	0	23	∞	∞	∞	28	36

碁峰 www.gotop.com.tw