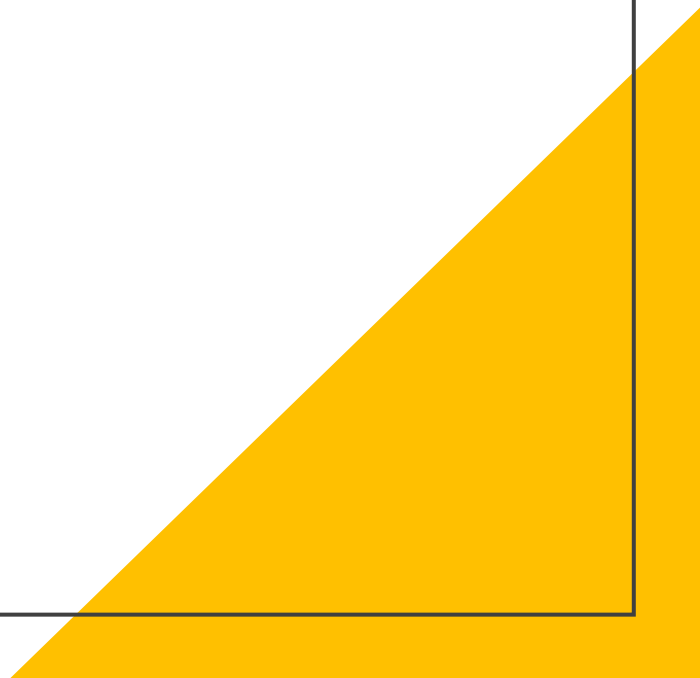


Disjoint Set

日月卦長

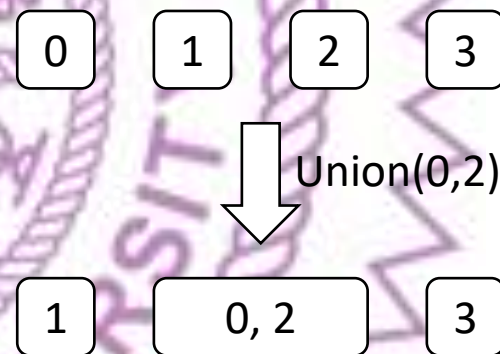


Disjoint Set 互斥集

- n 個元素，編號為 $0 \sim n - 1$ ，每個元素初始位於它自己的集合中。

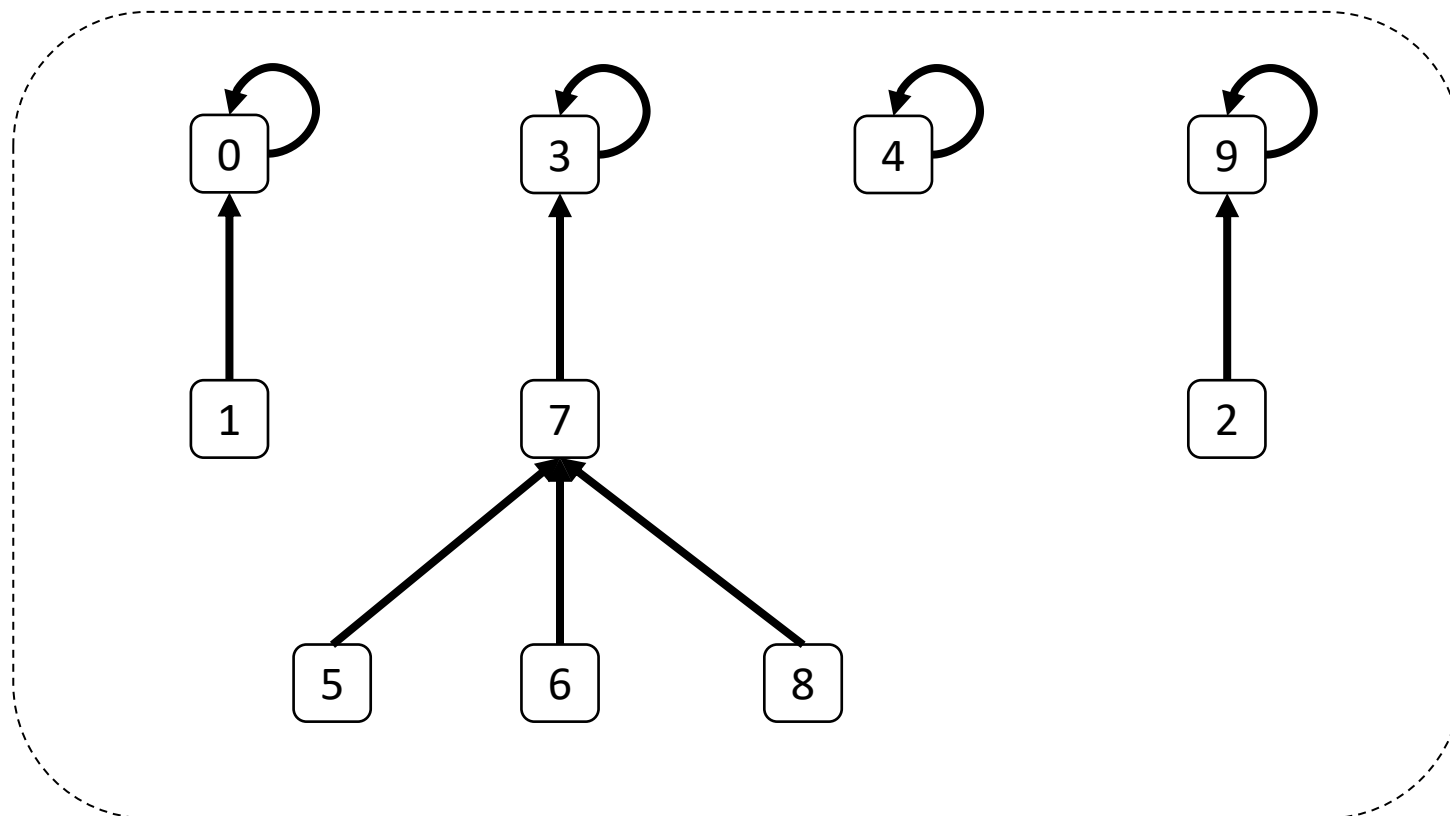
兩種經典操作：

- **Union(a, b):**
合併 a, b 元素所在的集合
- **Same(a, b):**
查詢 a, b 是否在同一個集合中



領袖選舉

- 今天如果想知道兩個員工是不是在同一個公司，我們可以去問他們的上司是誰
- 如果上司相同，則兩人同公司，否則不同公司。



Disjoint Set: 初始化

```
int parent[MAXN];  
void init(int n) {  
    for (int i = 0; i < n; ++i) {  
        parent[i] = i;  
    }  
}
```



Disjoint Set: find_root

- 輔助函數 `find_root(x)`:
找出 `x` 最上面那個人是誰

```
int find_root(int x) {  
    if (x == parent[x])  
        return x;  
    return find_root(parent[x]);  
}
```



`find_root(6) = 3`

Disjoint Set: Same

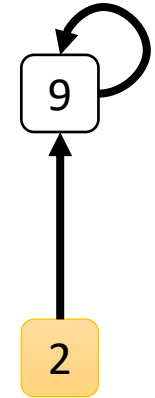
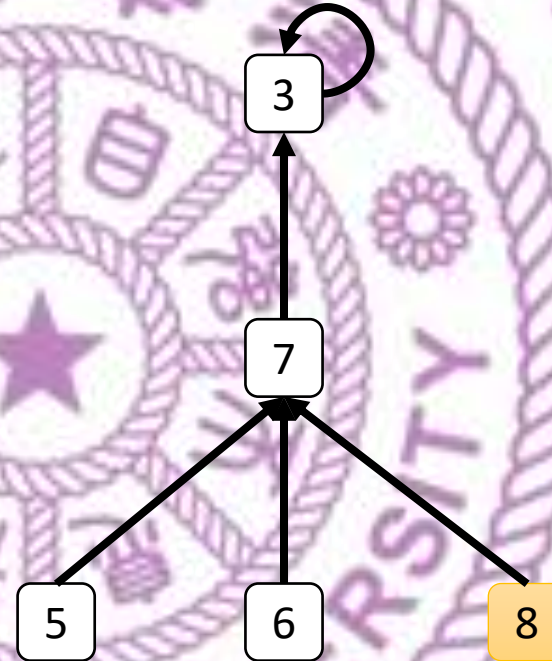
- 簡單到只剩下一行

```
bool Same(int a, int b) {  
    return find_root(a) == find_root(b);  
}
```

Disjoint Set: Union

- Union(a, b)
- 把 a 的根接到 b 的根底下

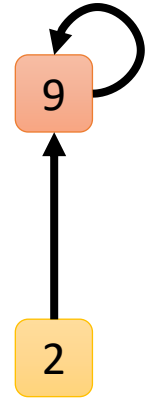
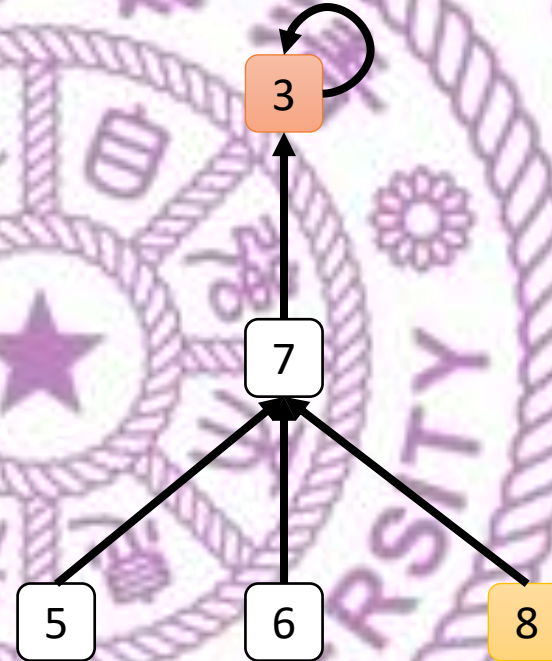
Union(8,2)



Disjoint Set: Union

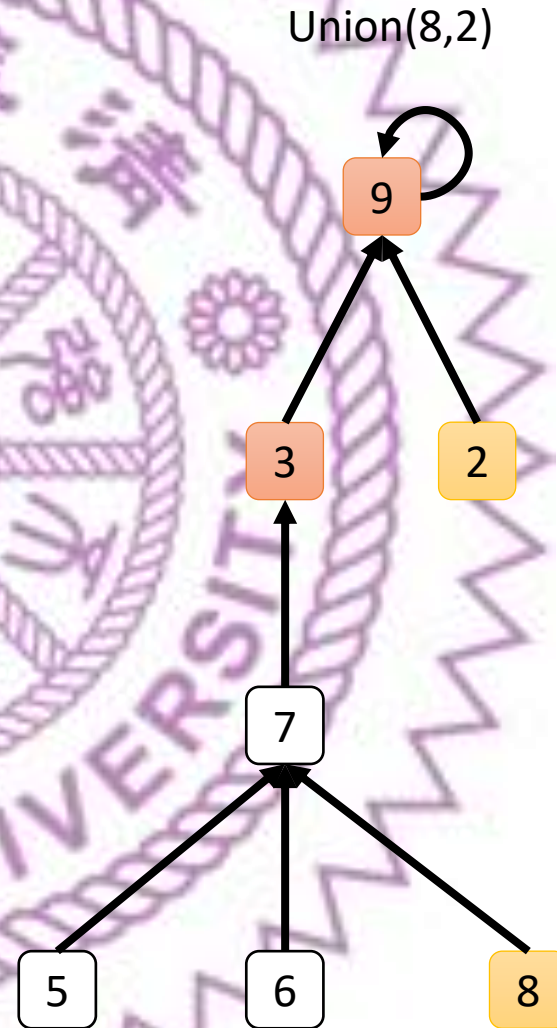
- Union(a, b)
- 把 a 的根接到 b 的根底下

Union(8,2)



Disjoint Set: Union

- Union(a, b)
- 把 a 的根接到 b 的根底下



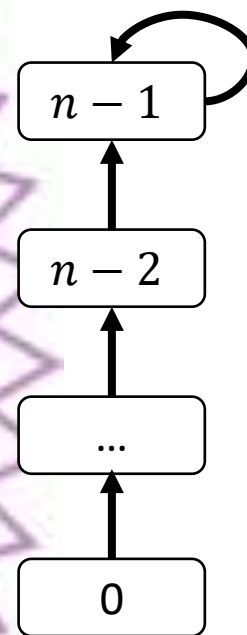
Disjoint Set: Union

- 也是只剩下一行

```
void Union(int a, int b) {  
    parent[find_root(a)] = find_root(b);  
}
```

複雜度

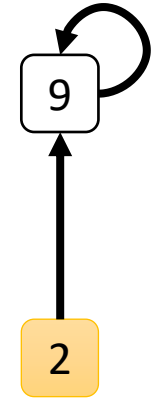
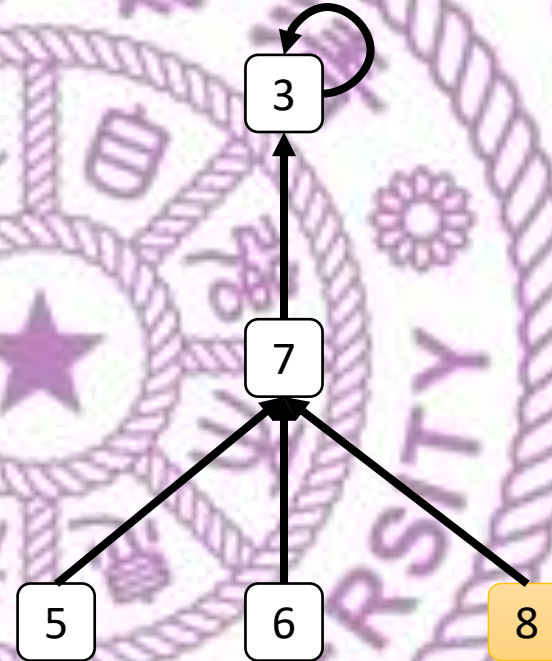
- Same 和 Union 都依賴於 $\text{find_root}(x)$
- 顯然在 $\text{find_root}(x)$ 最糟糕的情況下是 $O(n)$
- 所以每個操作都是 $O(n)$



Disjoint Set: 優化1

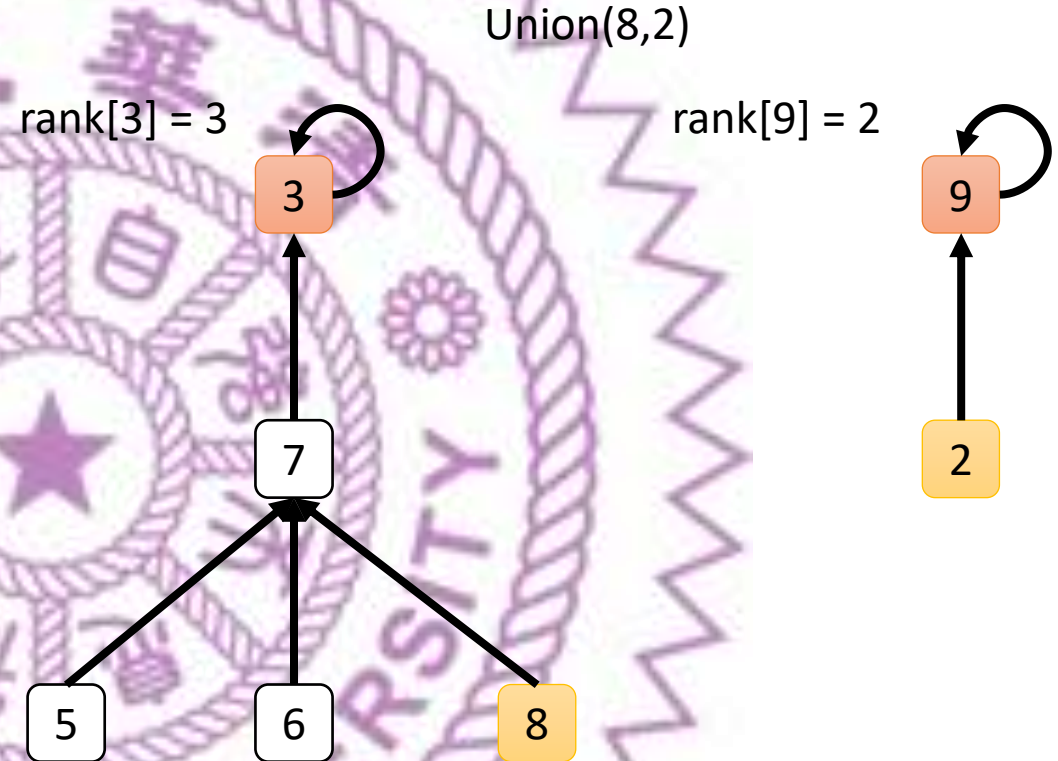
- Union by rank
- 將深度較淺的樹接到深度較深的樹的根底下

Union(8,2)



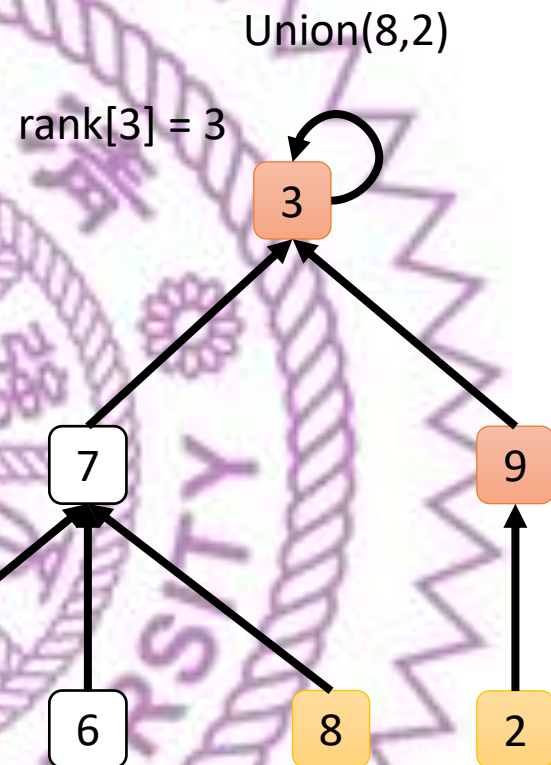
Disjoint Set: 優化1

- Union by rank
- 將深度較淺的樹接到深度較深的樹的根底下



Disjoint Set: 優化1

- Union by rank
- 將深度較淺的樹接到深度較深的樹的根底下

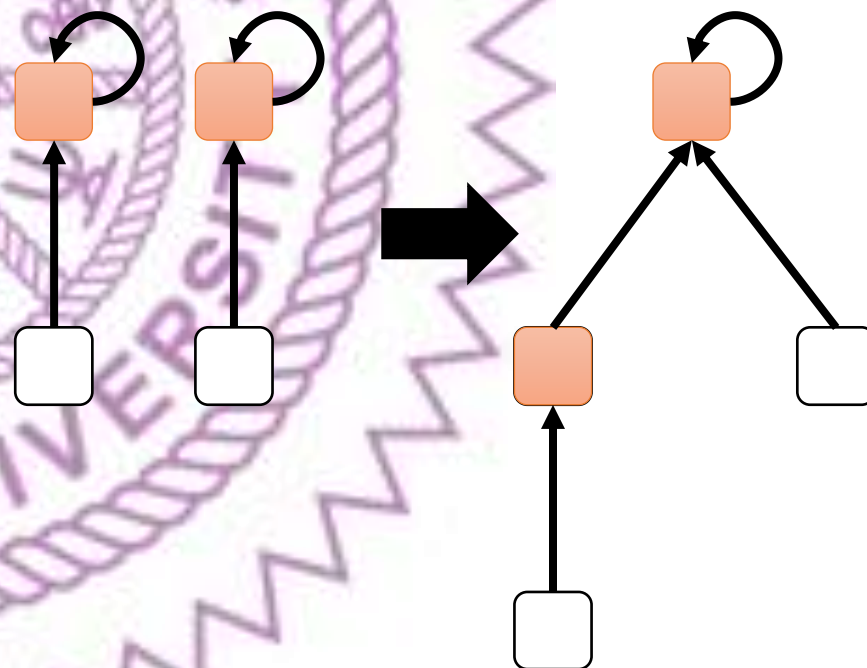


Disjoint Set:優化1

```
void Union(int a, int b) {  
    int pa = find_root(a), pb = find_root(b);  
    if (rank[pa] > rank[pb])  
        swap(pa, pb);  
    parent[pa] = pb;  
    if (rank[pa] == rank[pb])  
        ++rank[pb];  
}
```


複雜度

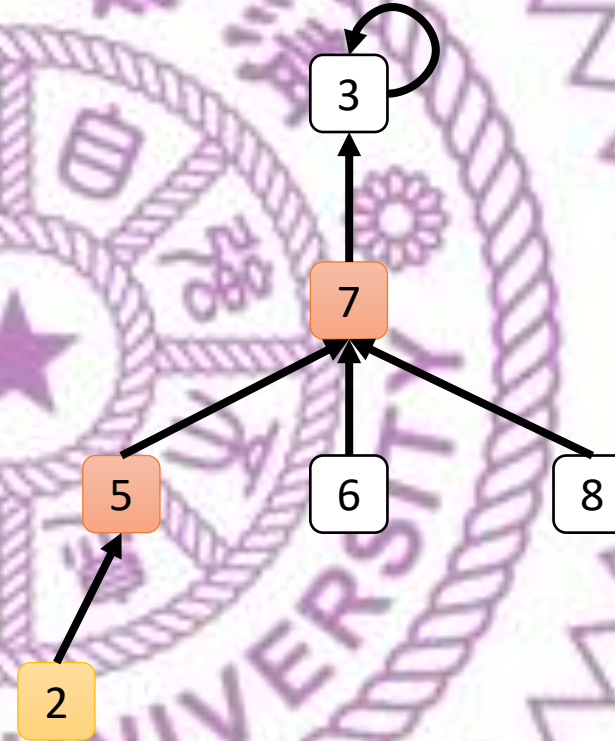
- 只有在合併兩個 **rank** 相同的元素後，**rank** 才會加一
- 也就是說所有元素的 **rank** 最大值為 $\lfloor \log_2 n \rfloor$
- 所以 `find_root(x)` 變成 $O(\log n)$



Disjoint Set: 優化2

- 路徑壓縮
- 在 `find_root` 的時候把經過的元素都指向 root

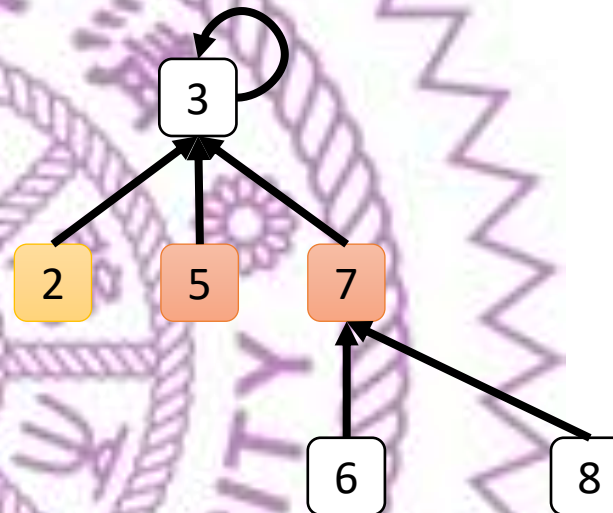
`find_root(2)` with 路徑壓縮



Disjoint Set: 優化2

- 路徑壓縮
- 在 `find_root` 的時候把經過的元素都指向 root

`find_root(2)` with 路徑壓縮



Disjoint Set:優化3

```
int find_root(int x) {  
    if (x == parent[x])  
        return x;  
    int root = find_root(parent[x]);  
    return parent[x] = root; // 路徑壓縮  
}
```


複雜度

- 如果 Union by rank 和 路徑壓縮 一起做的話
所有操作的複雜度均攤是 $O(\alpha(n))$
 - $\alpha(n) = A^{-1}(n, n)$
 - $A(n, n)$ 會隨 n 變大而遞增
 - $A(4, 4) = 2^{2^{2^{2^{2^2}}}} - 3$
 - 所以幾乎 $\alpha(n) < 5$
- 證明：我不會證

如果只作路徑壓縮...

- Union by rank 還須要維護一個 rank 陣列有點難寫啊
- 如果只作路徑壓縮複雜度會怎樣呢？
 - 其實是可以的，複雜度是 $O\left(n + f * \left(1 + \log_{2+f/n} n\right)\right)$
這裡 n 是點數， f 是呼叫 find_root 的次數
 - 大部分時候只需要用路徑壓縮就可以了

路徑壓縮 完整程式

- 只作路徑壓縮完整的程式非常短

```
int parent[MAXN];
void init(int n) {
    for (int i = 0; i < n; ++i) {
        parent[i] = i;
    }
}
int find_root(int x) {
    if (x == parent[x])
        return x;
    int root = find_root(parent[x]);
    return parent[x] = root; // 路徑壓縮
}
bool Same(int a, int b) {
    return find_root(a) == find_root(b);
}
void Union(int a, int b) {
    parent[find_root(a)] = find_root(b);
}
```

用 struct 包起來方便使用

```
class DisjointSet {
    vector<int> parent;
    int find_root(int x) {
        if (x == parent[x]) return x;
        int root = find_root(parent[x]);
        return parent[x] = root; // 路徑壓縮
    }

public:
    DisjointSet(size_t n) : parent(n + 1) { init(); }
    void init() {
        for (size_t i = 0; i < parent.size(); ++i)
            parent[i] = i;
    }
    bool Same(int a, int b) { return find_root(a) == find_root(b); }
    void Union(int a, int b) { parent[find_root(a)] = find_root(b); }
}
```

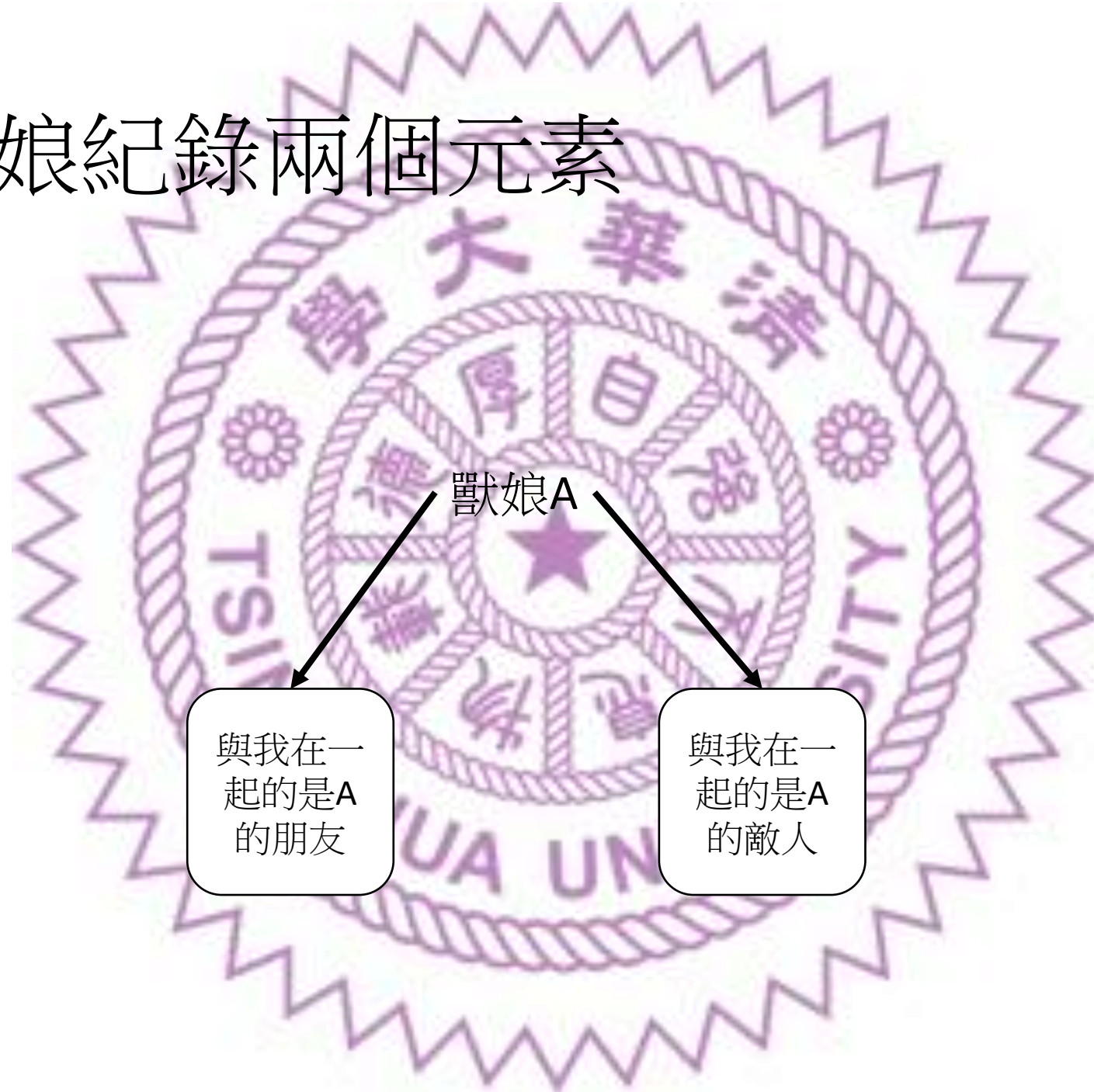
在ジャパリパーク中，獸娘都會有自己喜歡 (朋友) 跟討厭 (敵人) 的人，這些獸娘是很團結的，所以：

- 自己朋友的朋友也是自己的朋友
- 自己朋友的敵人也是自己的敵人
- 自己敵人的朋友也是自己的敵人
- 自己敵人的敵人就是自己的朋友

今天有依序有一些指令列在下方，告訴你哪兩位獸娘彼此間是朋友，哪兩位間是敵人，以及調查兩位獸娘間的關係是朋友或是敵人。如果遇到有衝突的指令的話，請無視這一條指令。

1. `you are friends a b`：指定 a, b 是朋友
2. `you are enemies a b`：指定 a, b 是敵人
3. `are you friends a b`：詢問 a, b 是否為朋友
4. `are you enemies a b`：詢問 a, b 是否為敵人

每個獸娘紀錄兩個元素



可愛的小動物 – A、B是朋友

與我在一起的是A
的朋友

與我在一起的是A
的敵人

與我在一起的是B
的朋友

與我在一起的是B
的敵人

可愛的小動物 – A、B是敵人

與我在一起的是A
的朋友

與我在一起的是A
的敵人

與我在一起的是B
的朋友

與我在一起的是B
的敵人