Disjoint Set Union (a.k.a. Union-Find)

A disjoint-set data structure (also called a union–find data structure or merge–find set) is a data structure that tracks a set of elements partitioned into a number of disjoint (non-overlapping) subsets.

It provides near-constant-time operations (bounded by the inverse Ackermann function) to

- Create new sets,
- Merge two existing sets, and
- Determine whether two or more elements are in the same set.

To maintain the information, **it uses a rooted tree to represent a set**. The vertices of the tree are the elements in the set. For each vertex (element), the pointer to its parent in the tree is stored. For root vertex, the pointer points to the vertex itself.

Assumptions (Disjoint-Set 運作時的基本假設)

1. Each element belongs to exactly one set.
2. No sets are empty.

Note that, the above implies that the sets are mutually disjoint and form a partition of the elements.

Implementation

The followings are the pseudo-codes for the three operations supported by this data structure:

- **MakeSet(x)** - create a set containing the element x

```
function MakeSet(x)
   if x is not already present:
     x.parent := x
```

- **Find(x)** - find the root vertex of the set containing x

```
function Find(x)
   if x.parent != x
     x.parent := Find(x.parent)

   return x.parent
```

- **Union(x,y)** - union the set containing x and the set containing y

```
function Union(x, y)
   xRoot := Find(x)
   yRoot := Find(y)

   // x and y are already in the same set
   if xRoot == yRoot
       return

   // swap xRoot and yRoot uniformly at random
   if rand() % 2 == 0
     swap xRoot and yRoot

   // merge y into x
   yRoot.parent := xRoot
```

Note that, the swap operation in Union(x,y) is very important in providing the near-constant-time guarantee. Other ways of implementation are possible. See Wiki - Disjoint-set data structure (https://en.wikipedia.org/wiki/Disjoint-set_data_structure).

註：隨機交換 xRoot 與 yRoot 的動作非常重要，少了這個動作，將無法保證此資料結構的效能。
亦有其它的實作方法，可參考 Wiki - Disjoint-set data structure (https://en.wikipedia.org/wiki/Disjoint-set_data_structure).

Hints & Comments

在實際的實作裡，你可以使用以下的方式來簡化上述的實作：

- 假設每個元素(element)皆以整數編號代表，編號從 0 到 $n-1$.
- 使用一個整數陣列 $parent[]$ 來儲存每個元素的 parent

如此可用靜態的方式完成此資料結構的實作。

否則的話，需要使用 struct 宣告所需之欄位，並動態配置記憶體空間