UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

Welcome to Algorithms and Data Structures! - CS2100

Disjoint Sets

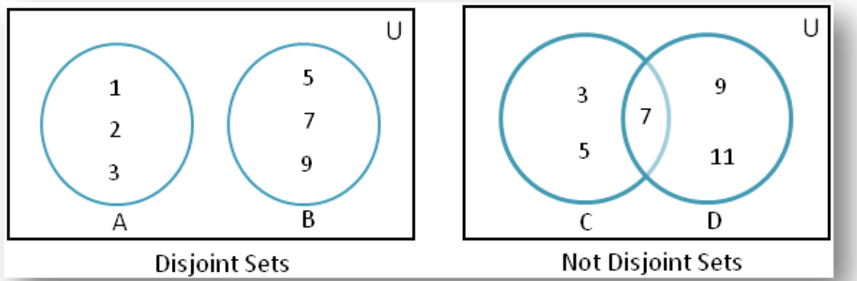
Union-Find

Conjuntos disjuntos (disjoint sets)

Es una estructura de datos que realiza un seguimiento de todos los elementos que están separados por conjuntos no conectados

Por ejemplo, en el lado derecho tenemos dos disjoint sets:

{1,2,3} y {5,7,9}



Los Disjoint sets nos pueden ayudar a encontrar componentes conectados, ciclos en un grafo no direccionado, o en la implementación de Kruskal

Los disjoint sets proveen operaciones que se ejecutan en un tiempo casi constante.

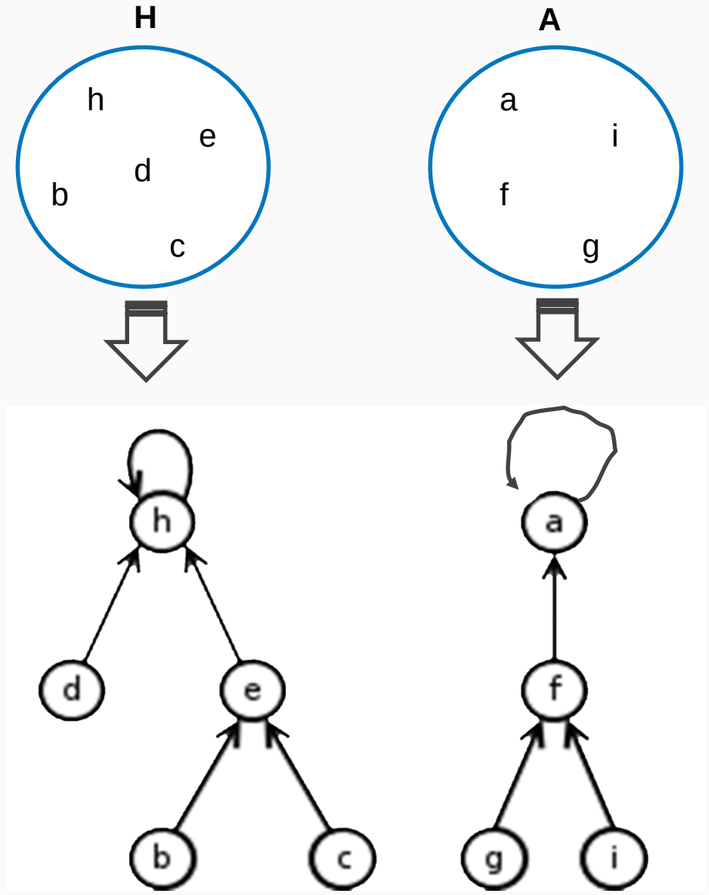
Conjuntos disjuntos (operaciones)

Se elige un representante de cada disjoint set, por ejemplo de los casos de la derecha sería h y a.

Sus principales operaciones son:

1. **MakeSet(x):** Crea un nuevo conjunto
2. **Union(x, y):** Une dos conjuntos con representantes “x” y “y”, suponiendo que x != y.
3. **Find(x):** Obtiene el elemento representativo del conjunto que contiene a x

   Podemos unir los conjuntos de la derecha y formar. {h, d, e, b, c, a, f, g, i}



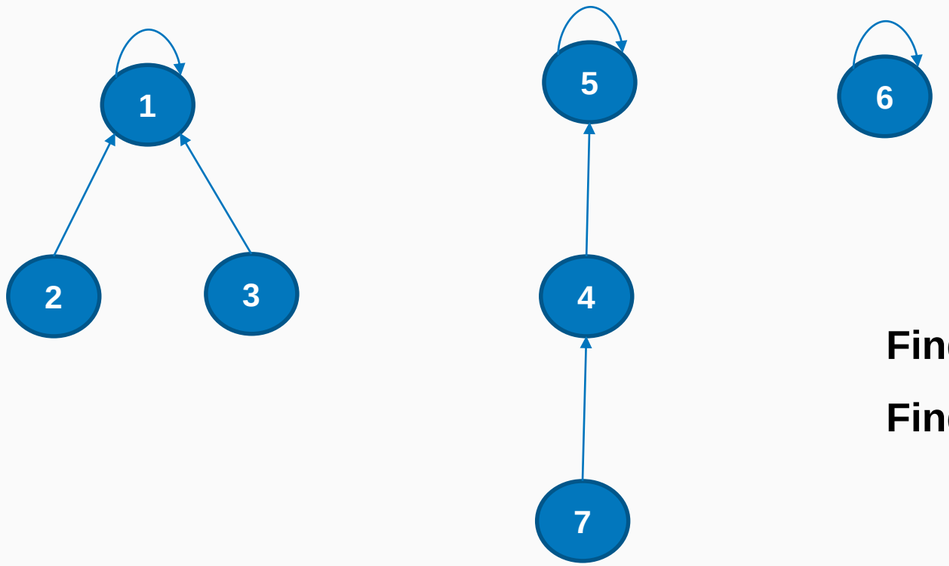
Conjuntos disjuntos (operaciones)

MakeSet se ejecuta en tiempo O(1) para crear un nuevo conjunto.

```
function MakeSet(x)
    x.parent = x
```

Conjuntos disjuntos (operaciones)

Find va a seguir el camino del padre hasta encontrar un bucle y llegar a la raíz.




```
function Find(x)
    if x.parent == x
        return x
    else
        return Find(x.parent)
```

Find(2) :: 1

Find(7) :: 5

Conjuntos disjuntos (operaciones)

Union verifica las raíces de dos disjoint sets, si son diferentes une los dos sets.

Union(1,2)

Union(3,4)

Union(6,7)

Union(4,1)

Union(5,7)

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

Conjuntos disjuntos (operaciones)

Union verifica las raíces de dos disjoint sets, si son diferentes une los dos sets.

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

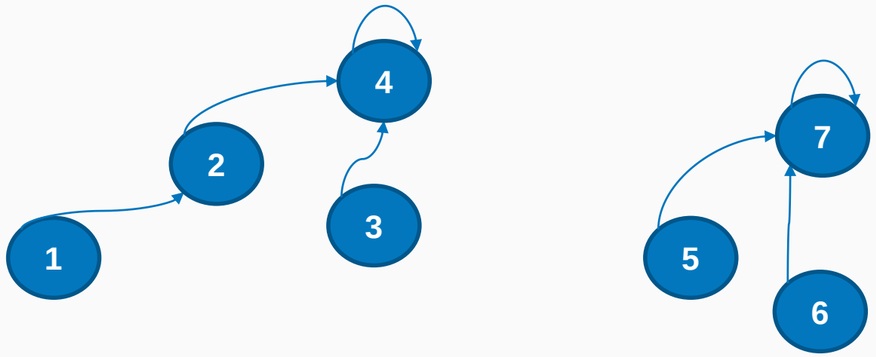
Union(1,2)

Union(3,4)

Union(6,7)

Union(4,1)

Union(5,7)



Conjuntos disjuntos (operaciones)

Union verifica las raíces de dos disjoint sets, si son diferentes une los dos sets.


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

Union(1,2)

Union(3,6)

Union(1,4)

Union(2,5)

Union(1,7)

Union(6,7)

Conjuntos disjuntos (operaciones)

Union verifica las raíces de dos disjoint sets, si son diferentes une los dos sets.

Union(1,2)

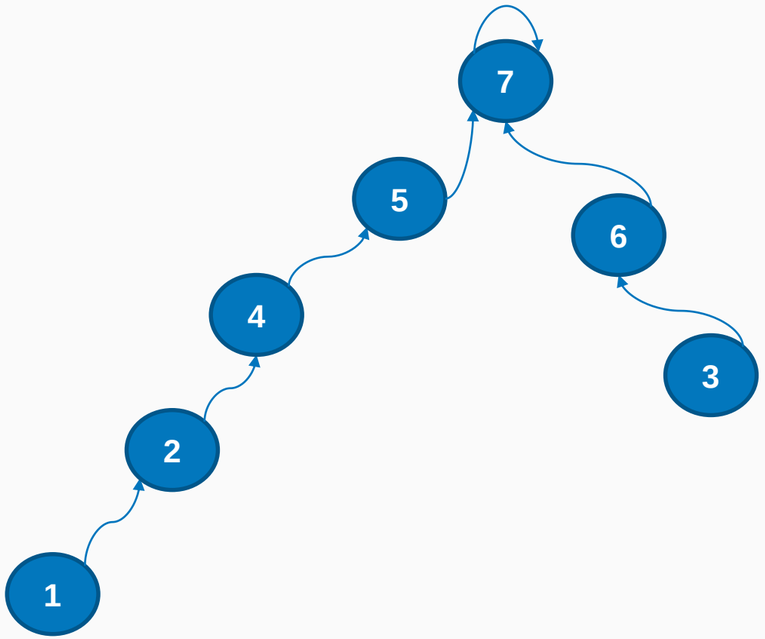
Union(3,6)

Union(1,4)

Union(2,5)

Union(1,7)

Union(6,7)




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

Conjuntos disjuntos (operaciones)

Union verifica las raíces de dos disjoint sets, si son diferentes une los dos sets.

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

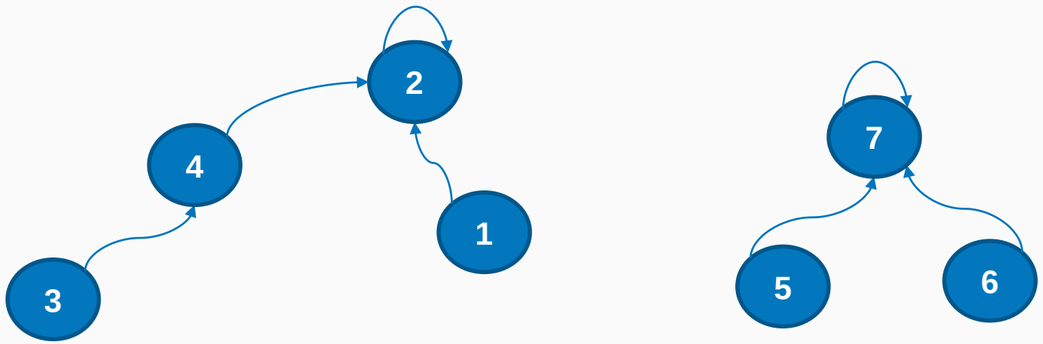
Union(1,2)

Union(3,4)

Union(6,7)

Union(4,1)

Union(5,7)



Conjuntos disjuntos (operaciones)

```
function Find(x)
    if x.parent == x
        return x
    else
        return Find(x.parent)
```

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

¿Cuánto tiempo tarda la función find para encontrar
al representante del conjunto en el peor caso?

O(n)
ya que el árbol puede estar muy desbalanceado

Conjuntos disjuntos (Optimización)

Optimización 1
Unión teniendo en cuenta el tamaño del árbol (union by rank)

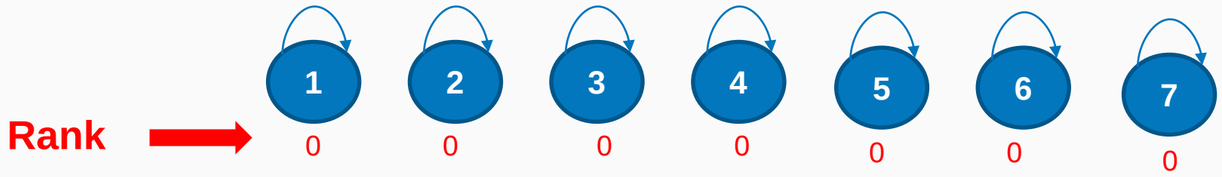
Union

El campo rank se increment en uno, si es igual en ambos disjoint sets, de lo contrario se usa el mayor

```
function MakeSet(x)
    x.parent := x
    x.rank   := 0
```

```
function Union(x, y)
    xRoot := Find(x)
    yRoot := Find(y)
    if xRoot == yRoot
        return
    //Ya que no están en el
    //mismo conjunto, se unen.
    if xRoot.rank < yRoot.rank
        xRoot.parent := yRoot
    else if xRoot.rank > yRoot.rank
        yRoot.parent := xRoot
    else
        xRoot.parent := yRoot
        yRoot.rank := yRoot.rank + 1
```

Conjuntos disjuntos (Optimización)



Union(1,2)

Union(3,6)

Union(1,4)

Union(2,5)

Union(1,7)

Union(6,7)

Conjuntos disjuntos (Optimización)

Rank

Union(1,2)

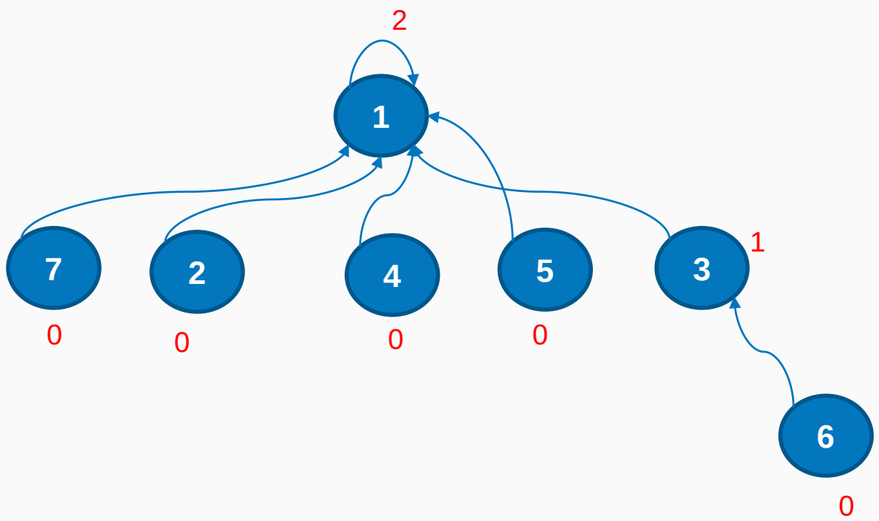
Union(3,6)

Union(1,4)

Union(2,5)

Union(1,7)

Union(6,7)

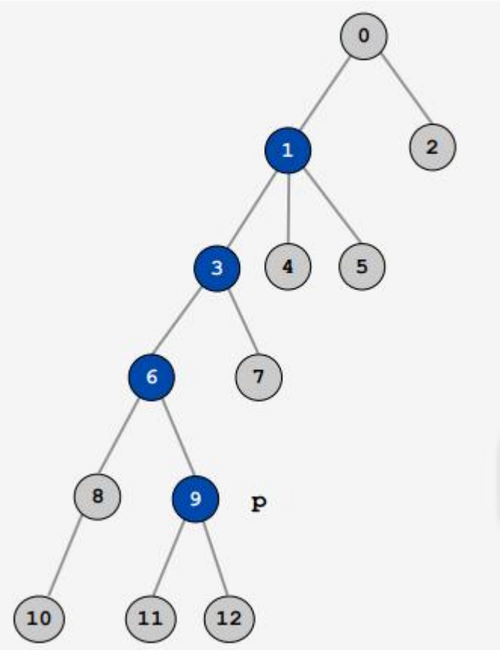


Find O(rank) < O(log n) < O(n)

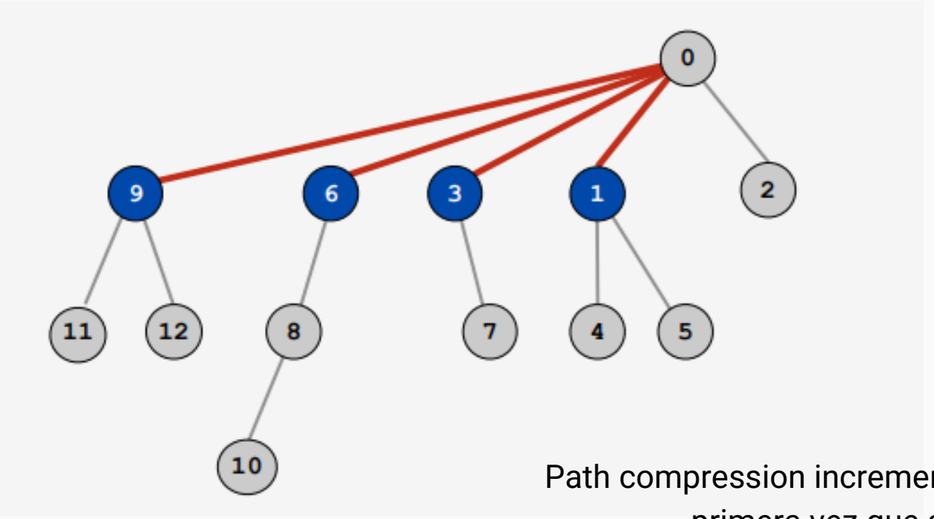
Conjuntos disjuntos (Optimización)

Optimización 2

Path compression! Durante la ejecución de find(x), después de localizar la raíz del árbol que contiene x, se hace que la ruta apunte directamente al representante.



Find(9)

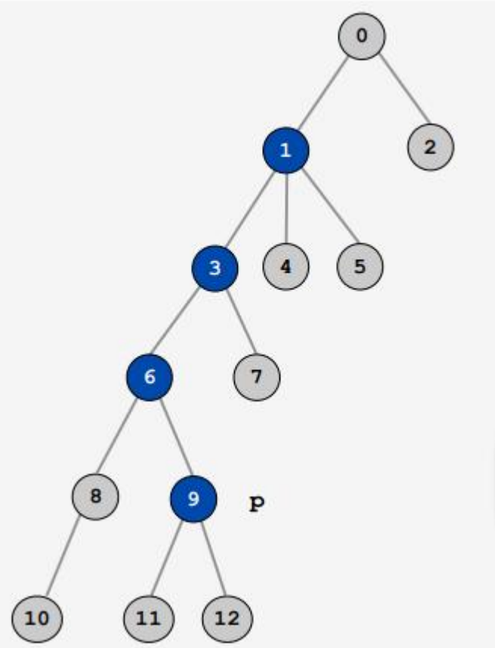


Path compression incrementa el tiempo del find en un factor para la primera vez que se realiza sobre un elemento, pero vuelve constantes los accesos posteriores

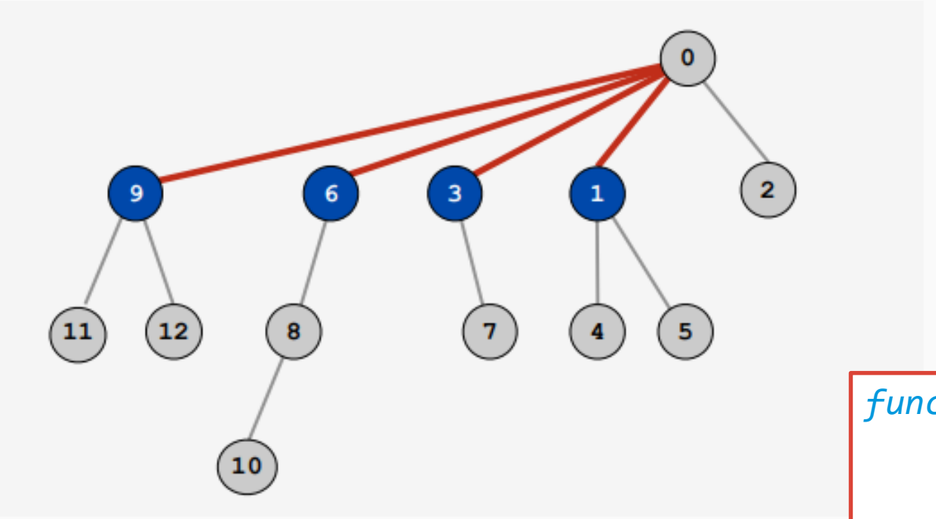
Conjuntos disjuntos (Optimización)

Optimización 2

Path compression! Durante la ejecución de find(x), después de localizar la raíz del árbol que contiene x, se hace que la ruta apunte directamente al representante.



Find(9)



¿Como sería el algoritmo?

```
function Find(x)
    if x.parent == x
        return x
    else
        return Find(x.parent)
```

Conjuntos disjuntos (Optimización)

Ejercicio:

Cree un disjoint set con el siguiente arreglo y operaciones

1, 2, 3, 4, 5, 6, 7

makeSet(1)
makeSet(2)
makeSet(3)
makeSet(4)
makeSet(5)
makeSet(6)
makeSet(7)

union(5, 2)
union(2, 3)
union(4, 3)
union(1, 6)
union(7, 1)
union(1, 5)

Conjuntos disjuntos (Implementación)

Con Nodos

```
struct Node {
    Node* parent;
    int rank;
    T data;
}
```

```
function MakeSet(int x)
```

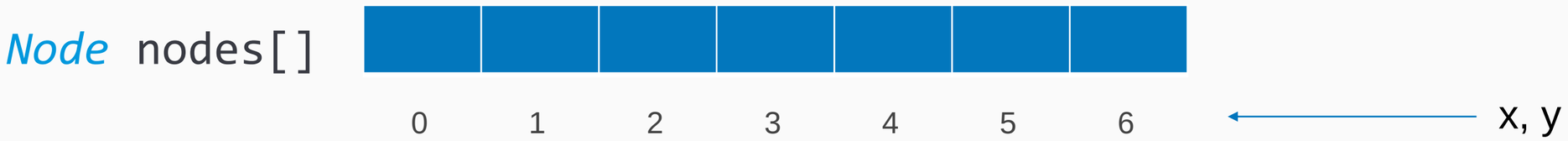
?

```
function Find(int x)
```

?

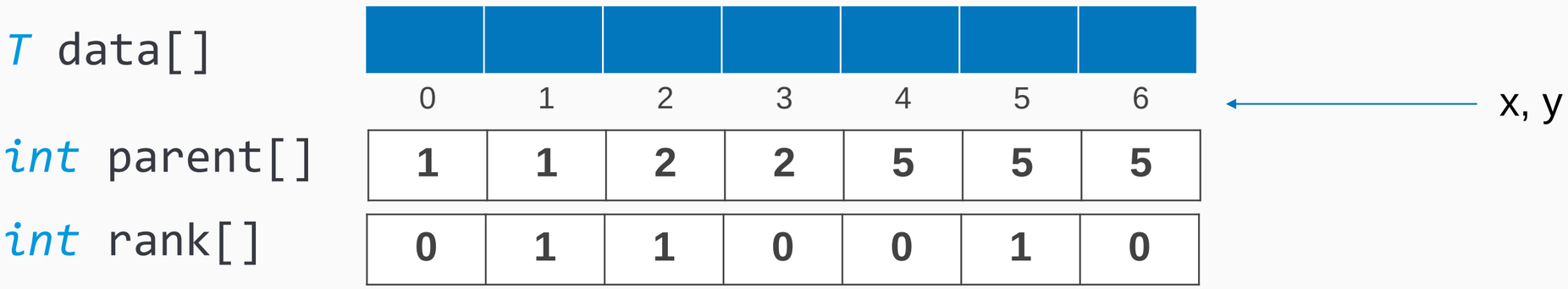
```
function Union(int x,int y)
```

?



Conjuntos disjuntos (Implementación)

Con Arrays




```
function MakeSet(int x)
```

?

```
function Find(int x)
```

?

```
function Union(int x,int y)
```

?

Conjuntos disjuntos (Implementación)

```
class DisjoinSet // interface
{
public:
    virtual ~DisjoinSet();
    virtual void MakeSet(int x) = 0;
    virtual int Find(int x) = 0;
    virtual void Union(int x, int y) = 0;

    //verifica si hay un camino entre x e y
    virtual bool isConnected(int x, int y) = 0;
};
```

```
template <typename T>
class DisjoinSetTree: public DisjoinSet
{
private:
    // define the structures
public:
    DisjoinSetTree(T* data, int n);
    // implement all functions
}
```

```
template <typename T>
class DisjoinSetArray: public DisjoinSet
{
private:
    // define the structures
public:
    DisjoinSetArray(T* data, int n);
    // implement all functions
}
```

UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

Welcome to Algorithms and Data Structures! - CS2100