# UFDS

# Union Find Disjoint Set

# DSU

# Disjoint Sets Union

# UFDS – Union Find Disjoint Set



Es una estructura de datos que maneja conjuntos disjuntos, conjuntos que no comparten elementos

Permite realizar estas dos operaciones en forma eficiente $\approx O(1)$:

- Búsqueda: a qué conjunto pertenece un elemento dado (o testear si dos elementos pertenecen al mismo conjunto)
- Unión: unir dos conjuntos en uno

# UFDS – Union Find Disjoint Set



- El UFDS puede ser usado para resolver problemas como el de encontrar las componentes conexas de un grafo no dirigido, o para saber si un vértice está en la misma componente que otro, etc.

- En la implementación del algoritmo de Kruskal

# UFDS: Implementación



- La estrategia que sugiere la estructura es la de escoger un elemento representativo “padre” de cada conjunto, como representante del mismo. Este representante puede ser usado como identificador del mismo.

- Para esto, el UFDS crea un árbol para cada conjunto disjunto, formando éstos un bosque, donde la raíz de cada árbol, simboliza el elemento identificador de un conjunto.

# UFDS: Implementación



- Dado un ítem, para obtener el identificador del conjunto, simplemente hay que seguir la cadena de padres hasta llegar a la raíz del árbol.

- De esta manera, determinar si dos elementos pertenecen al mismo conjunto, es comparar la raíz del árbol que contiene a cada uno, verificando si son iguales (pertenecen al mismo conjunto) o diferentes (pertenecen a diferentes conjuntos).

# UFDS: Implementación



- Para realizar esto en forma eficiente, se almacena en arreglos, el índice del ítem padre y (un límite superior de) la altura del árbol de cada conjunto.

  Vectores: **p** y **rank**

  **p[i]** almacena el padre inmediato del ítem i

  Si i es el ítem representativo de cierto conjunto, entonces **p[i]=i**

  **rank[i]** almacena (el límite superior de) la altura del árbol con raíz i

- Inicialmente cada elemento apunta a sí mismo.

# UFDS: Operación Unión *(UnionSet)*



- Con esta estructura, la unión de conjuntos se limita a cambiar el ítem representativo (raíz) de uno de los conjuntos, para que sea el nuevo padre del ítem representativo del otro.
- De esta manera, unionSet(i, j), hará que los ítems i y j tengan el mismo ítem representativo (directa o indirectamente).

# UFDS: Operación Unión *(UnionByRank)*



- Para mejorar la eficiencia, se puede usar la información contenida en el vector **rank**.
- Se setea el ítem representativo del conjunto con *mayor rank*, como nuevo padre del conjunto con *menor rank*, de esta manera se **minimiza** el rank del árbol resultante.
- Si ambos ranks son iguales, se puede usar cualquiera como nuevo padre y se incrementa el valor.

Estrategia: "union by rank"

# UFDS: Operación Búsqueda *(findSet)*



- Para optimizar la operación de búsqueda findSet(i), existen varias técnicas.
  - Una muy conocida es Path compression
  - Existen otras como Path splitting y Path halving (Tarjan & Van Leeuwen)

# UFDS: Path compression



- Cada vez que encontramos el ítem representativo (raíz) de un conjunto, siguiendo la cadena de links al padre, a partir de un ítem dado, es posible setear el padre de *todos los ítems* atravesados para que apunten directamente a la raíz.

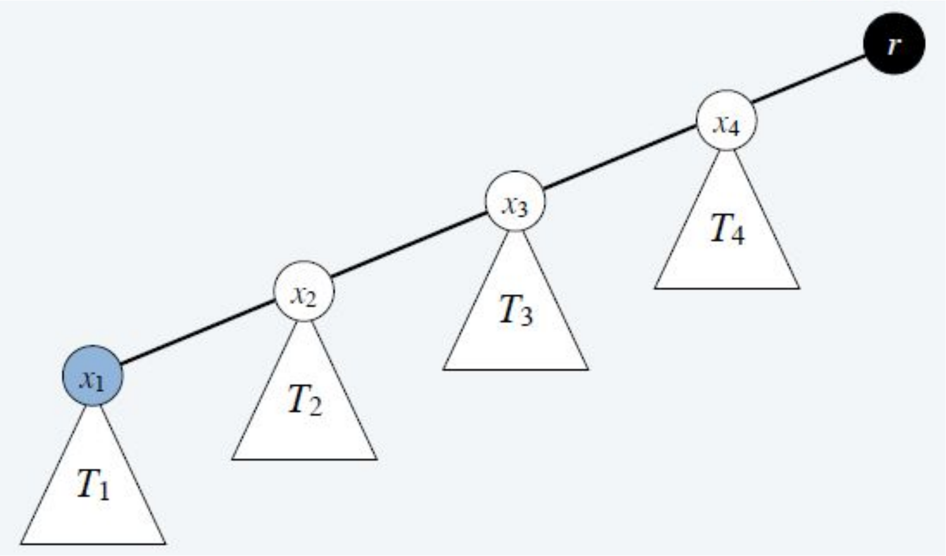
- En todas las llamadas siguientes a findSet(i), sobre los ítems modificados, se atravesará un solo link.

- Esto, si bien cambia la estructura del árbol (para hacer la operación más eficiente), preserva el concepto de conjunto disjunto.

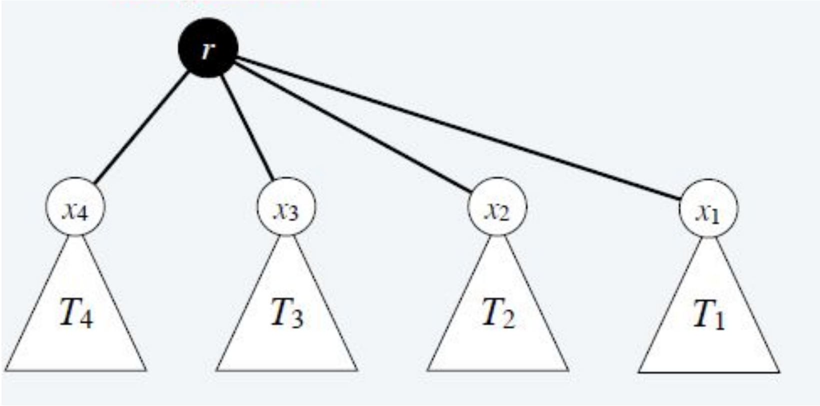
# UFDS: Path compression



Antes de path compression




Después de path compression



# UFDS: Ejemplo (0)



N=5, siendo N la cantidad de nodos

int p[5] = {0, 1, 2, 3, 4};

int rank[5] = {0, 0, 0, 0, 0};

# UFDS: Ejemplo (1)



Operación: 'unión de 0 y 1'.

int p[5] = {0, 0, 2, 3, 4};

int rank[5] = {1, 0, 0, 0, 0};

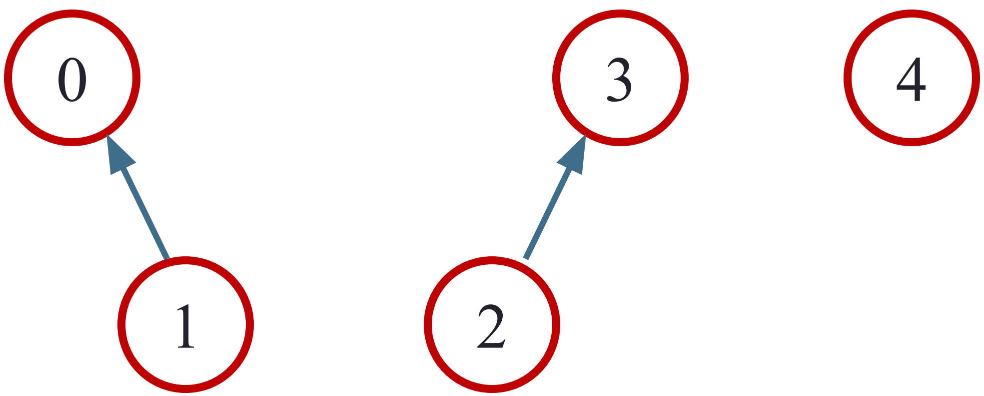
# UFDS: Ejemplo (2)



Operación: 'unión de 3 y 2'.

int p[5] = {0, 0, 3, 3, 4};

int rank[5] = {1, 0, 0, 1, 0};



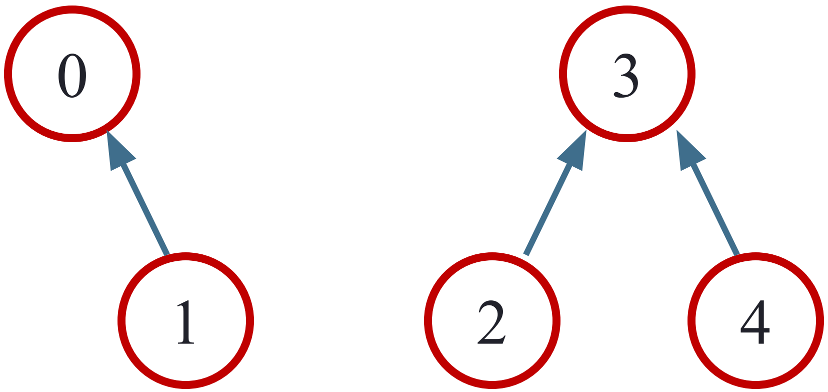
# UFDS: Ejemplo (3)



Operación: 'unión de 4 y 3'.

int p[5] = {0, 0, 3, 3, 3};

int rank[5] = {1, 0, 0, 1, 0};



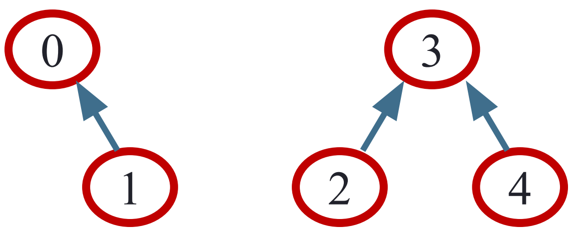
# UFDS: Ejemplo (4)

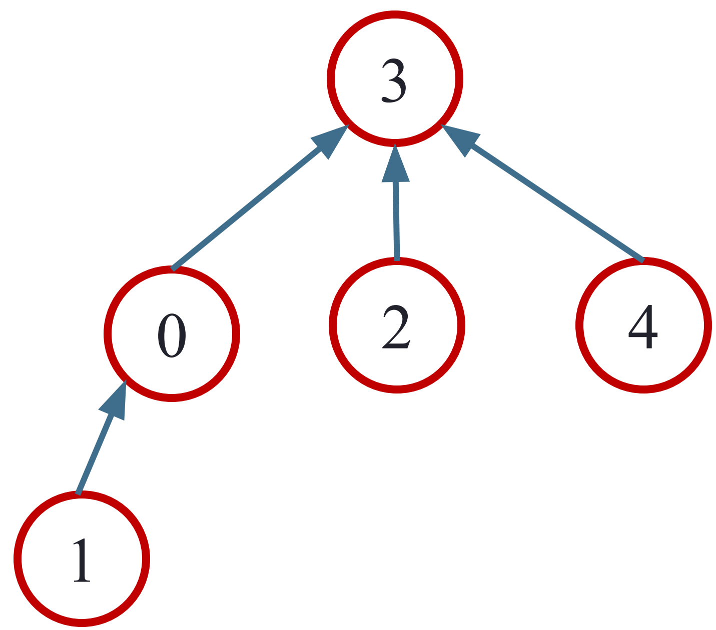


Operación: ‘unión de 3 y 0’.

int p[5] = {3, 0, 3, 3, 3};

int rank[5] = {1, 0, 0, 2, 0};



# UFDS: Ejemplo (5)



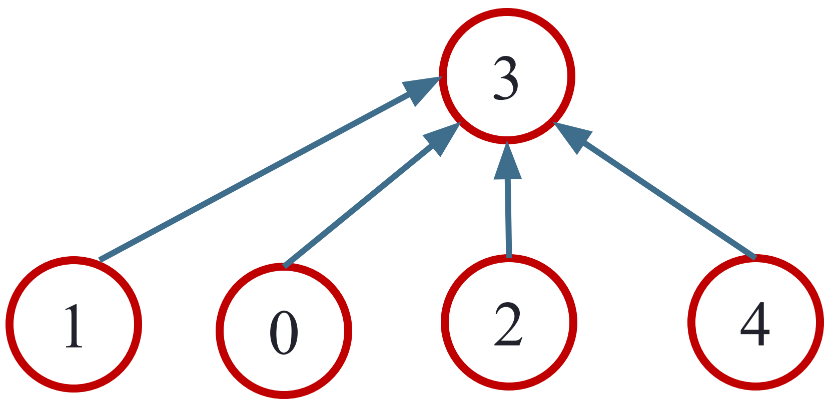
Operación:

¿el elemento 1 pertenece al mismo conjunto que el elemento 4?

¿findSet(1) = findSet(4) ?

int p[5] = {3, 3, 3, 3, 3};

int rank[5] = {1, 0, 0, 2, 0};

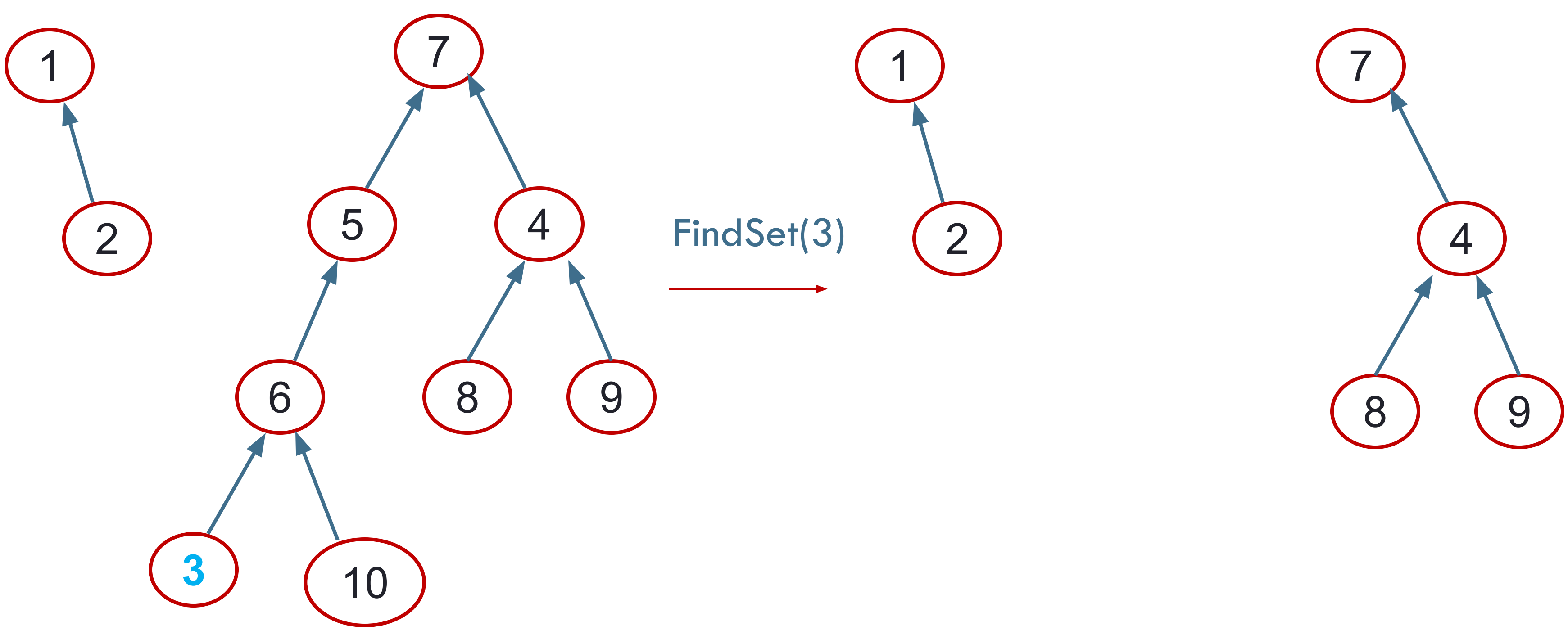



Usando Path compression

# UFDS: Ejemplo de Path compression



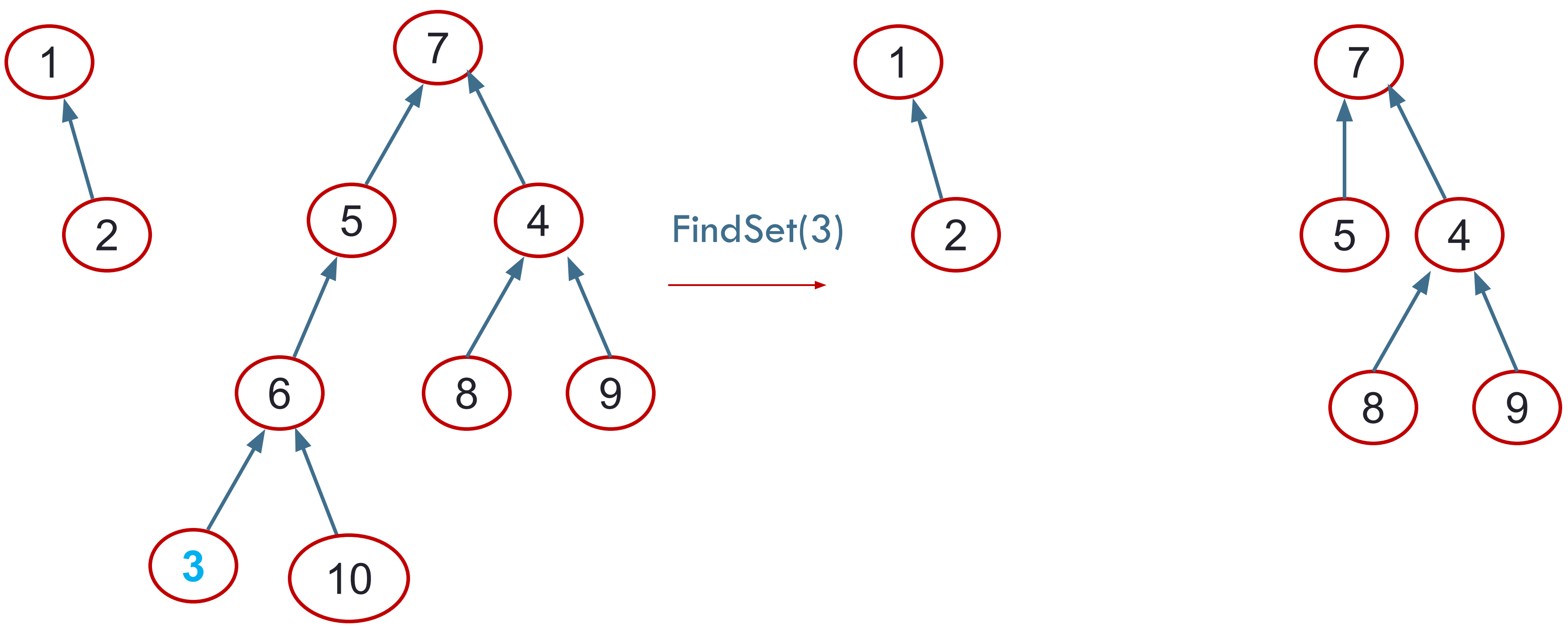
En la operación findSet(i), todos los nodos en el camino de búsqueda apuntan directamente a la raíz.



# UFDS: Ejemplo de Path compression



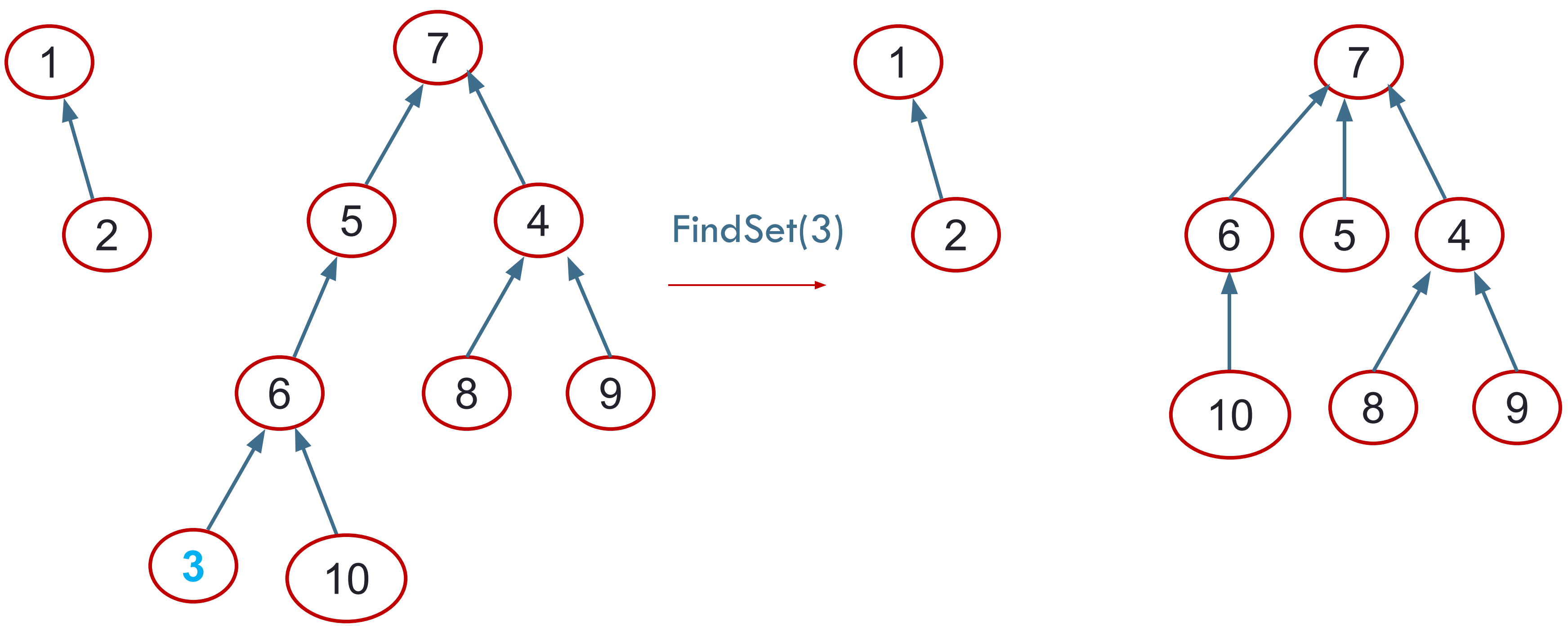
En la operación findSet(i), todos los nodos en el camino de búsqueda apuntan directamente a la raíz.



# UFDS: Ejemplo de Path compression



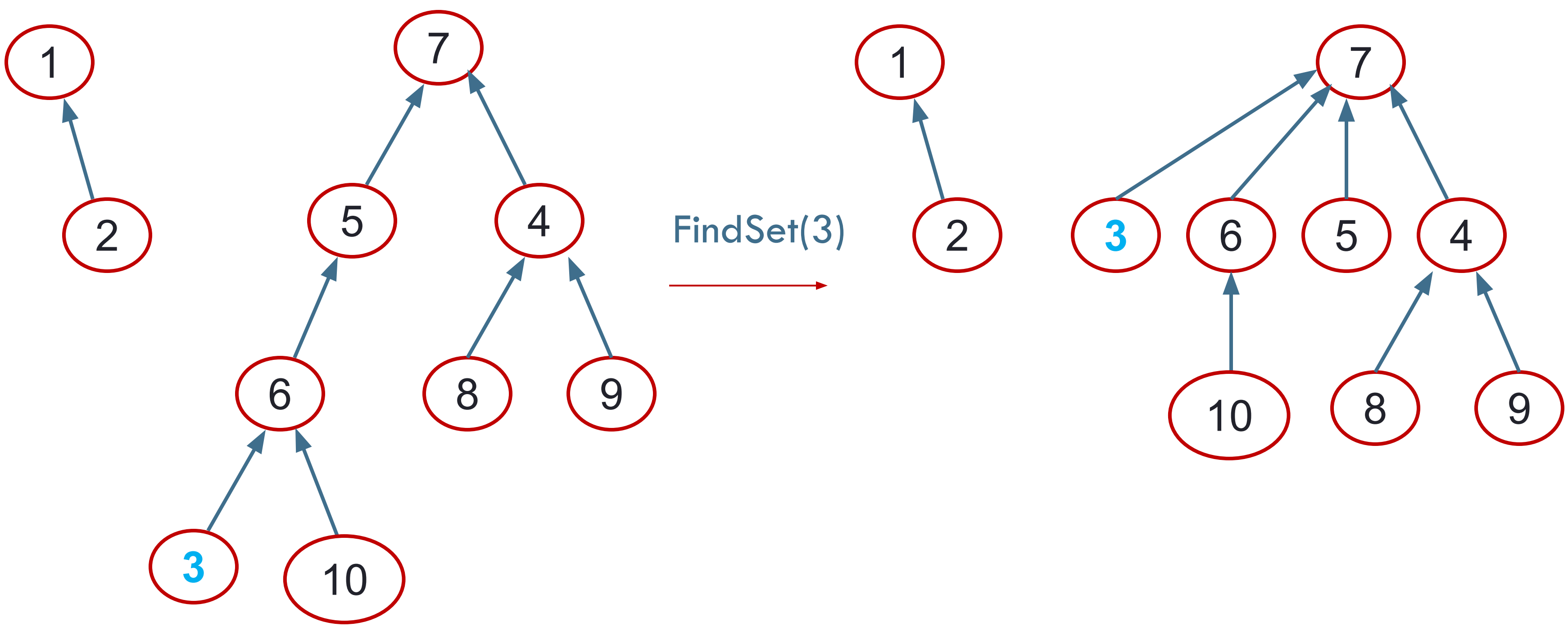
En la operación findSet(i), todos los nodos en el camino de búsqueda apuntan directamente a la raíz.



# UFDS: Ejemplo de Path compression



En la operación findSet(i), todos los nodos en el camino de búsqueda apuntan directamente a la raíz.



# UFDS: Union by rank



Mantiene un rank para cada nodo, inicialmente en 0. Engancha la raíz de menor rank, a la de mayor rank; si son iguales, incrementa el rank de la nueva raíz en 1.

**make-Set**$(x)$

$p(x) \leftarrow x$

$rank(x) \leftarrow 0$

**findSet**$(x)$

while $x \neq p(x)$

  $x \leftarrow p(x)$

return $x$

**unionByRank**$(x, y)$

$r \leftarrow$ findSet$(x)$

$s \leftarrow$ findSet$(y)$

if $(r = s)$ return

else if $rank(r) > rank(s)$

  $p(s) \leftarrow r$

else if $rank(r) < rank(s)$

  $p(r) \leftarrow s$

else

  $p(r) \leftarrow s$

  $rank(s) \leftarrow rank(s) + 1$

# UFDS: Path compression



Luego de encontrar la raíz ***r*** de un árbol que contiene a **x**, se modifica el puntero al padre de todos los nodos atravesados para que apunten directamente a ***r***.

**findSet**$(x)$

$$
\begin{aligned}
&\text{if } x \neq p(x) \\
&\quad p(x) \leftarrow \text{findSet } p(x) \\
&\text{return } p(x)
\end{aligned}
$$

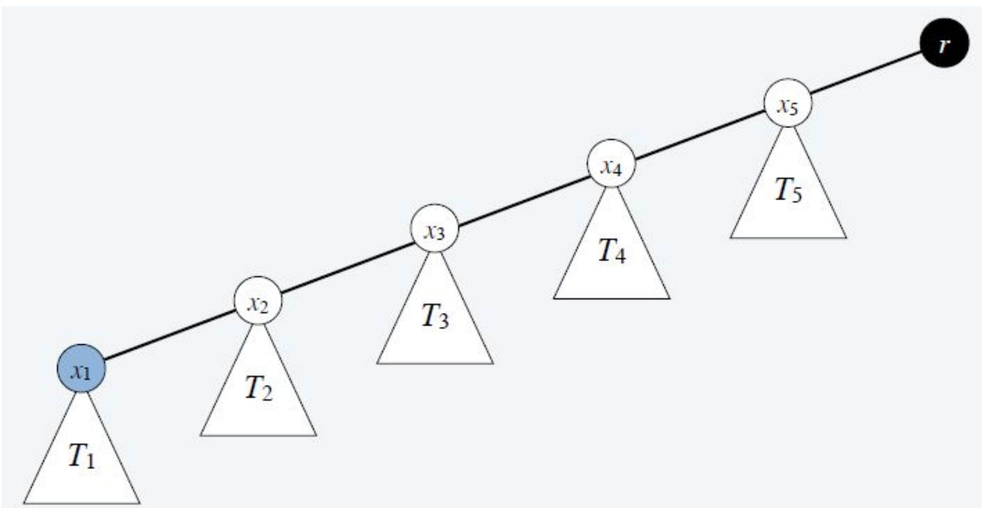
Nota: Path compression no cambia el rank de un nodo; entonces height(x) ≤ rank(x) pero no son necesariamente iguales

# UFDS: Variantes de compresión de caminos

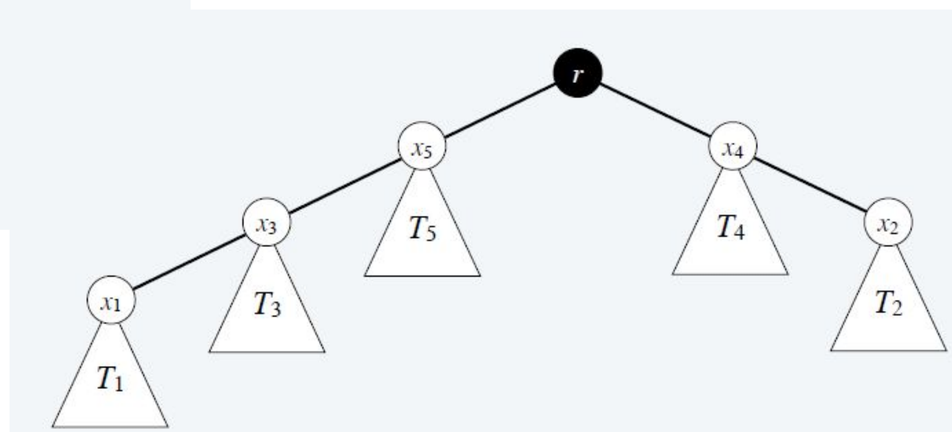


Path splitting: Todos los nodos en el camino recorrido apuntan a su abuelo

Antes de path splitting



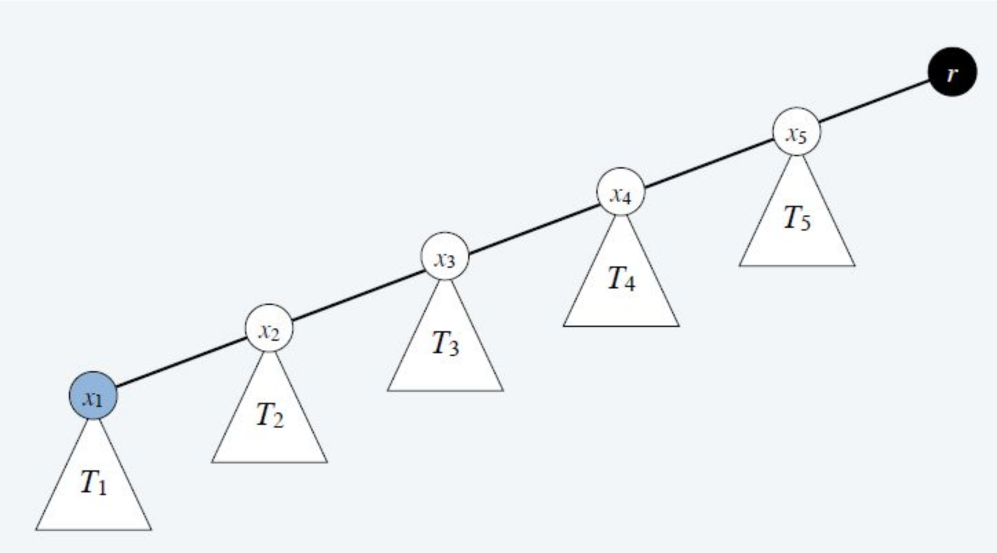

Después de path splitting



# UFDS: Variantes de compresión de caminos

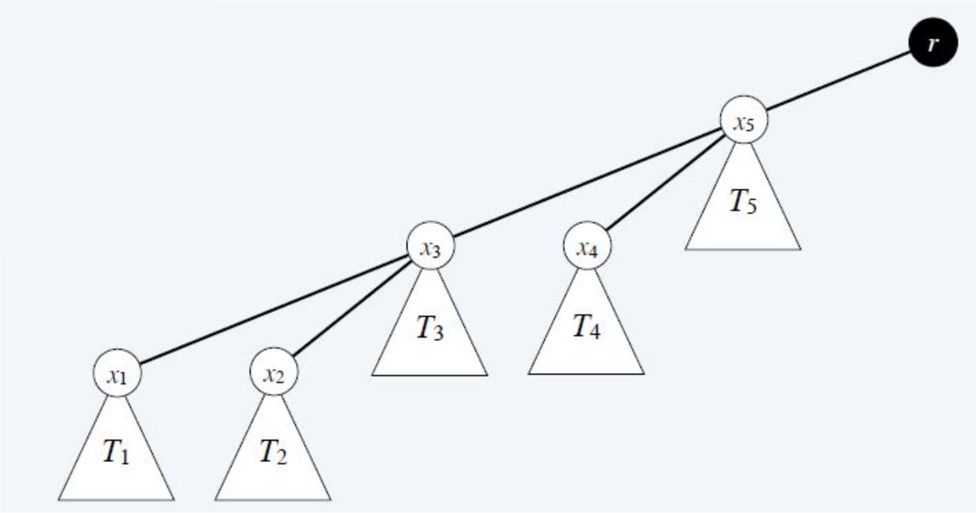


Path halving: Los nodos modificados en el camino apuntan a su abuelo




Antes de path halving

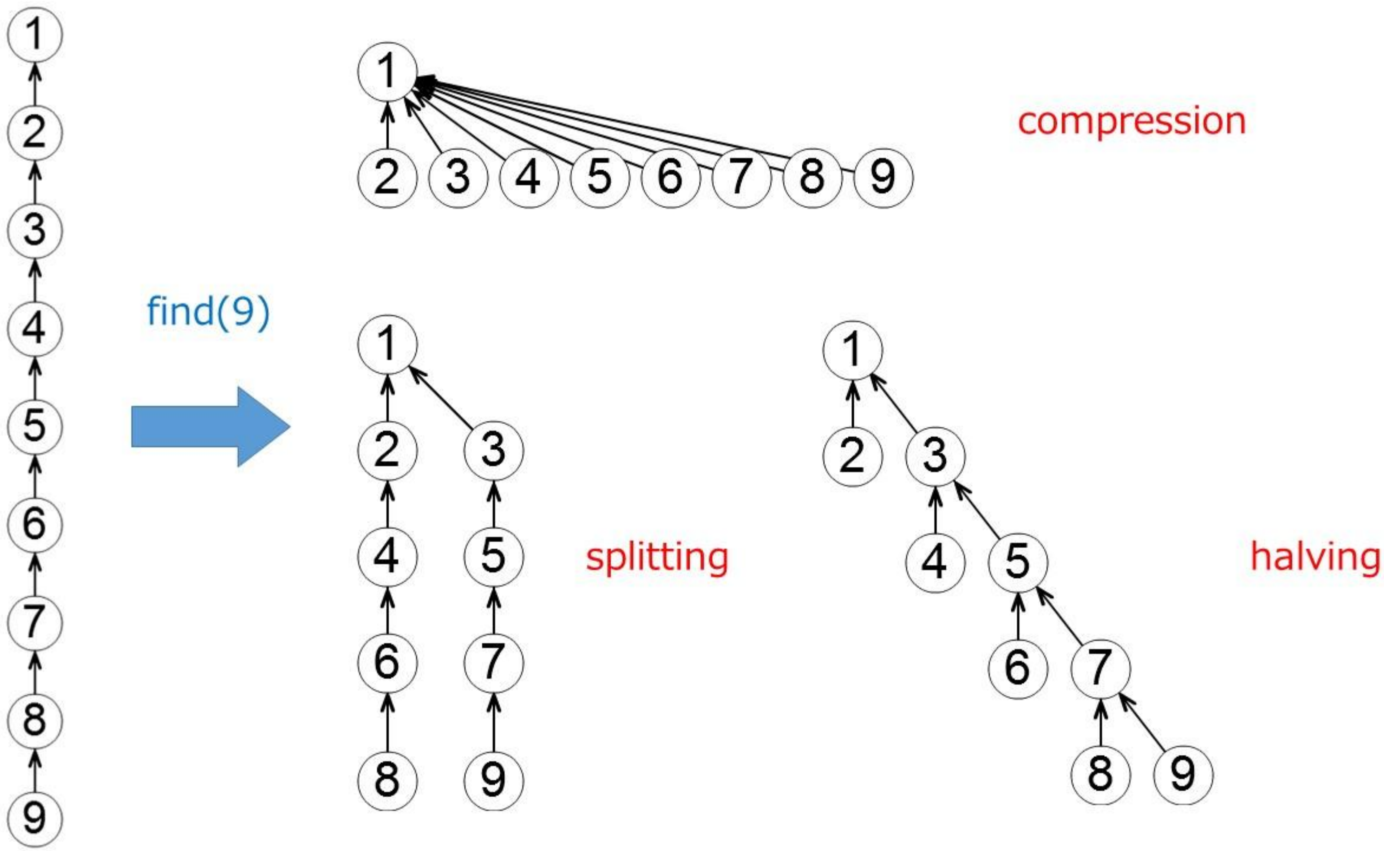
Después de path halving



# UFDS: Variantes de compresión de caminos



Ejemplos de las tres formas de compresión



# Referencias



Steven Halim – Felix Halim. ***Competitive Programming 3***. Handbook for ACM ICPC and IOI Contestants 2013

Jon Kleinverg – Eva Tardos. ***Algorithm Design***. Copyright © 2005 Pearson-Addison Wesley
http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Robert Tarjan – Jan Van Leeuwen. ***Worst Case Analysis of Set Unions Algoritms***. Journal of the Association for Computing Machinery (ACM), Vo.. 31, No. 2, April 1984, pp 245--281.