Ayudantía 8: Heap

Carlos Paredes : cparedesr@uc.cl
Alonso Carrasco : cristian.carrasco@uc.cl

Contenidos

- Repaso de Heap
- Problemas de Heap
- Union-find

¿Qué es un Heap?

Heap binario

- Un heap es un árbol binario que nos permite mantener un orden de prioridad los elemento de un conjunto

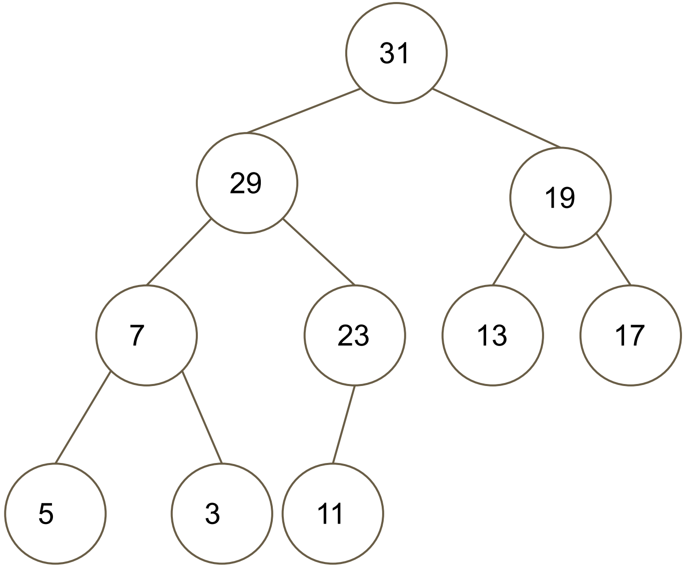
- Puede tener un orden creciente o decreciente

- Es monótono (a medida que se avanza los solo crecen o se achican)

- En caso de estar balanceado la inseración y extracción es log(n)

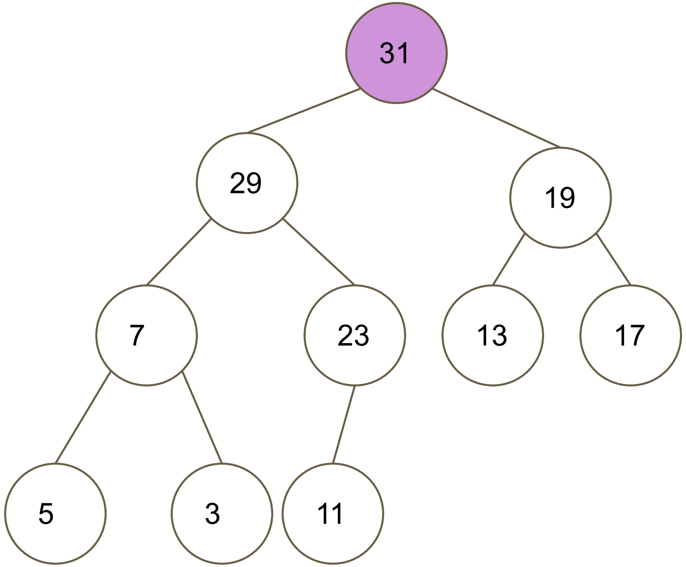
- Puede ser representado con un array 😀😀

Ejemplo de extracción



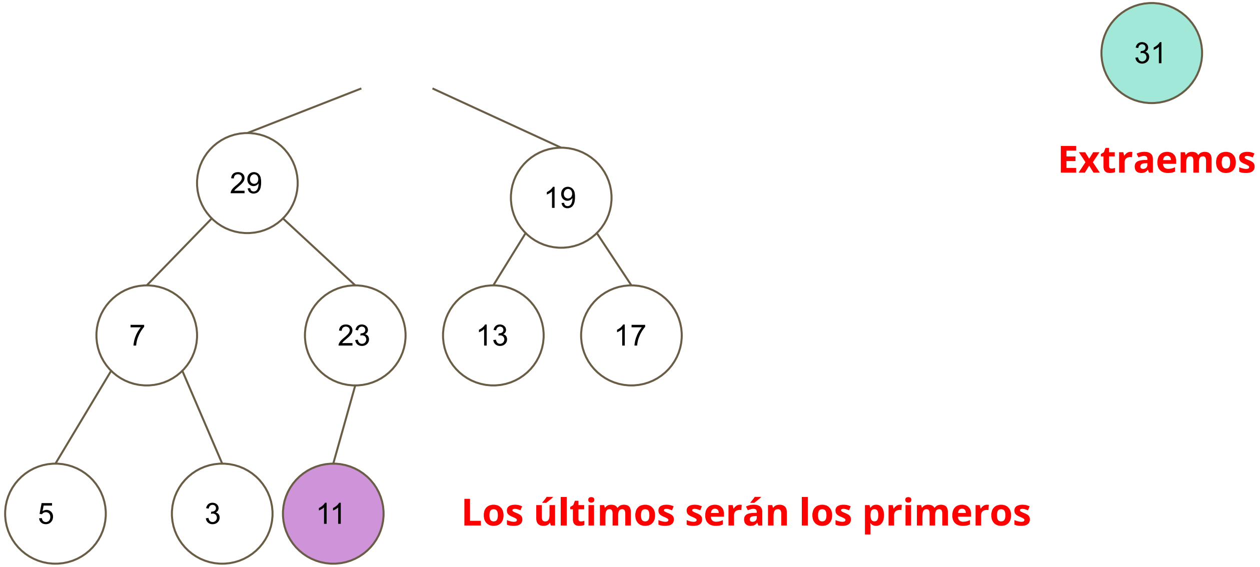
31	29	19	7	23	13	17	5	3	11						
1	2	3	4	5	6	7	8	9	10						

Ejemplo de extracción



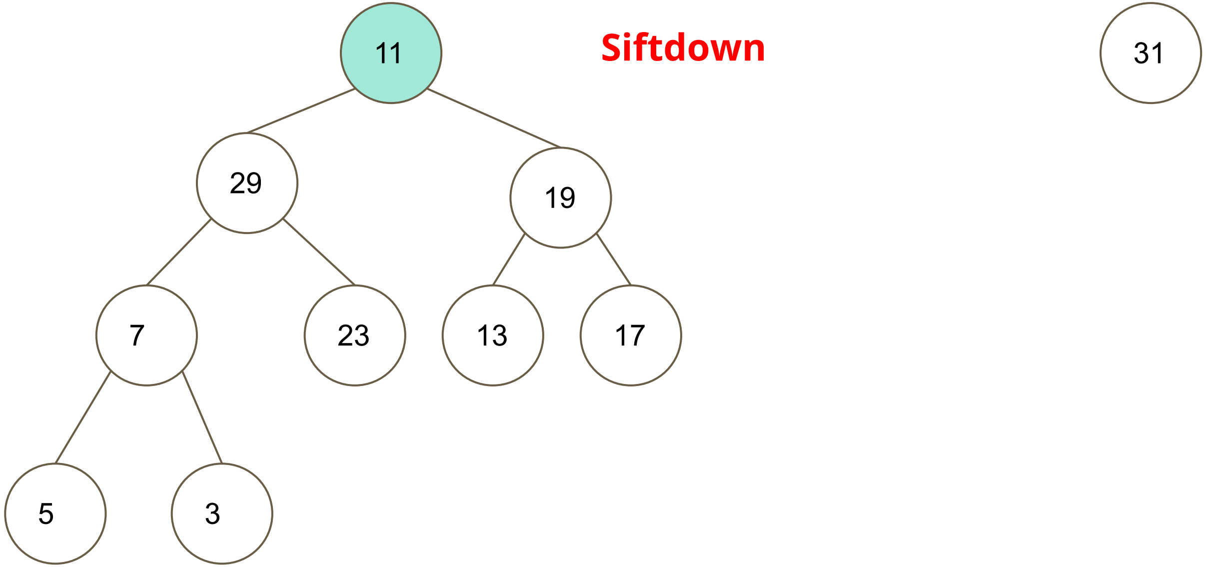
31	29	19	7	23	13	17	5	3	11						
1	2	3	4	5	6	7	8	9	10						

Ejemplo de extracción



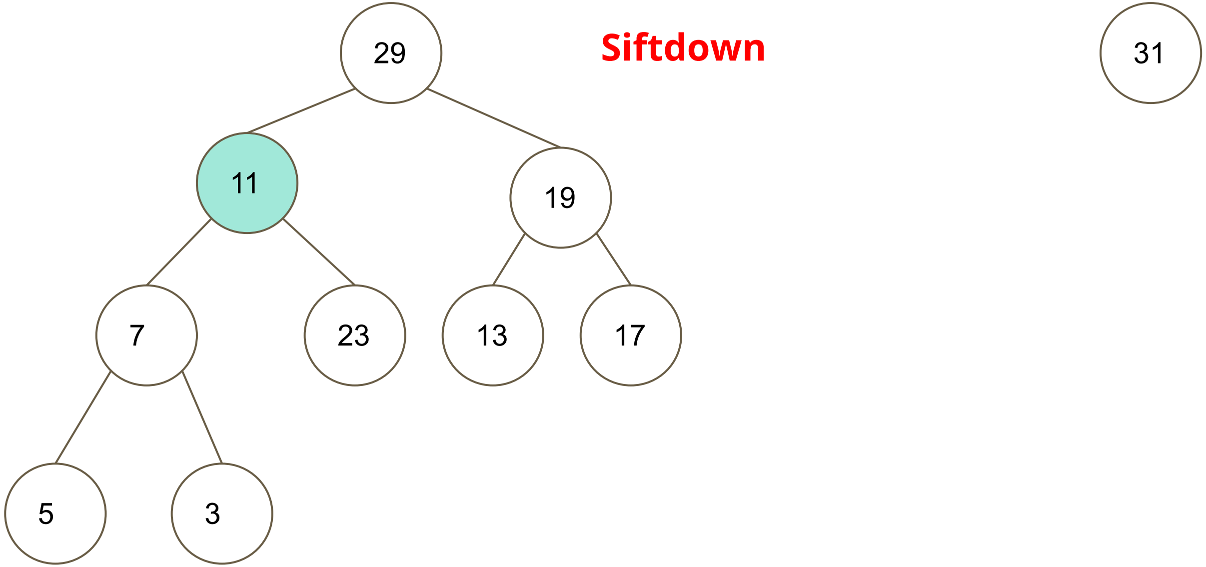
	29	19	7	23	13	17	5	3	11						
1	2	3	4	5	6	7	8	9	10						

Ejemplo de extracción



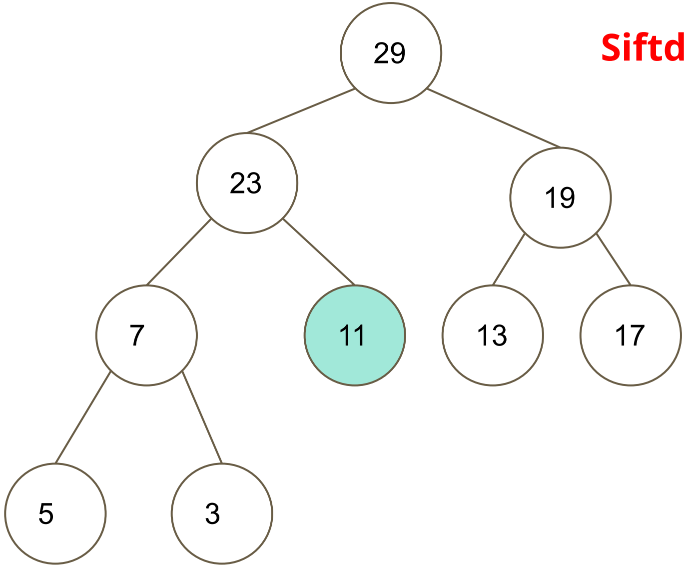
11	29	19	7	23	13	17	5	3							
1	2	3	4	5	6	7	8	9	10						

Ejemplo de extracción

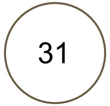


29	11	19	7	23	13	17	5	3							
1	2	3	4	5	6	7	8	9	10						

Ejemplo de extracción



Siftdown



29	23	19	7	11	13	17	5	3							
1	2	3	4	5	6	7	8	9	10						

Heap Sort

¿Qué es heapsort?

¿Cómo funciona?

¿Cuál es su complejidad?

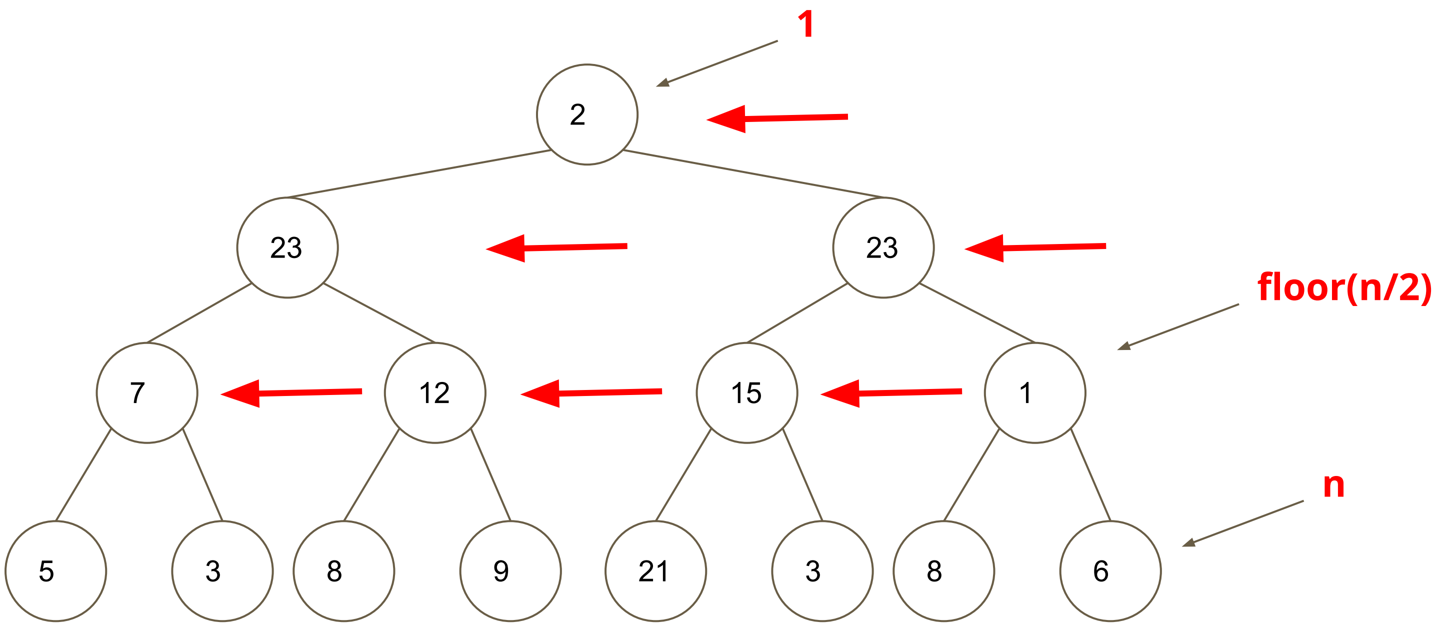
¿Que propiedades aprovecha?

Problema 1

Demuestre que el siguiente algoritmo transforma un array en un heap y además funciona en O(n) (n es la cantidad de elementos del array A)

```
Transform_into_heap(A, n):
    for i in range(floor(n/2), 1):
        Siftdown(A, i)
```

Entendiendo el algoritmo



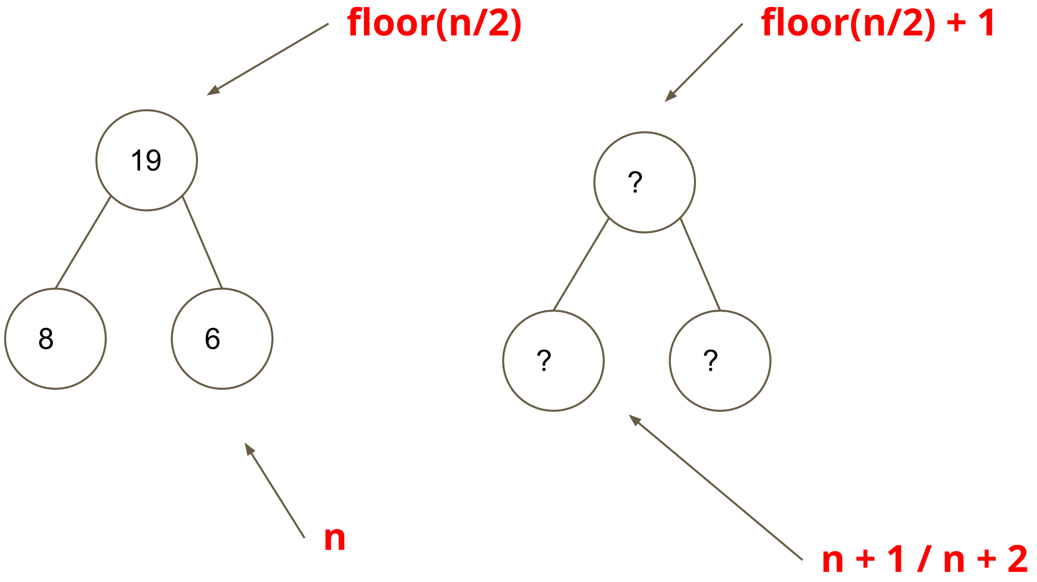
Inducción

Vamos a probar para todo k tal que k esté en [1, n] y además k > i donde i es donde estamos iterando se cumple que el elemento k en la posición k es la raíz de un heap.

Caso base

nótese que una hoja siempre será la raíz de un heap, pues un puro elemento forma un heap. [a] es un array que es un heap. al principio i = floor(n/2), notemos que todos los elementos k entre n y floor(n/2) (sin incluir a floor(n/2)) son hojas.

Caso base



floor(n/2): Va a el último elemento que tenga hijos. Este será padre del elemento n. por ende floor(n/2) + 1 sería padre del elemento n+1 o n+2, pero estos elementos no existen.

No existen!

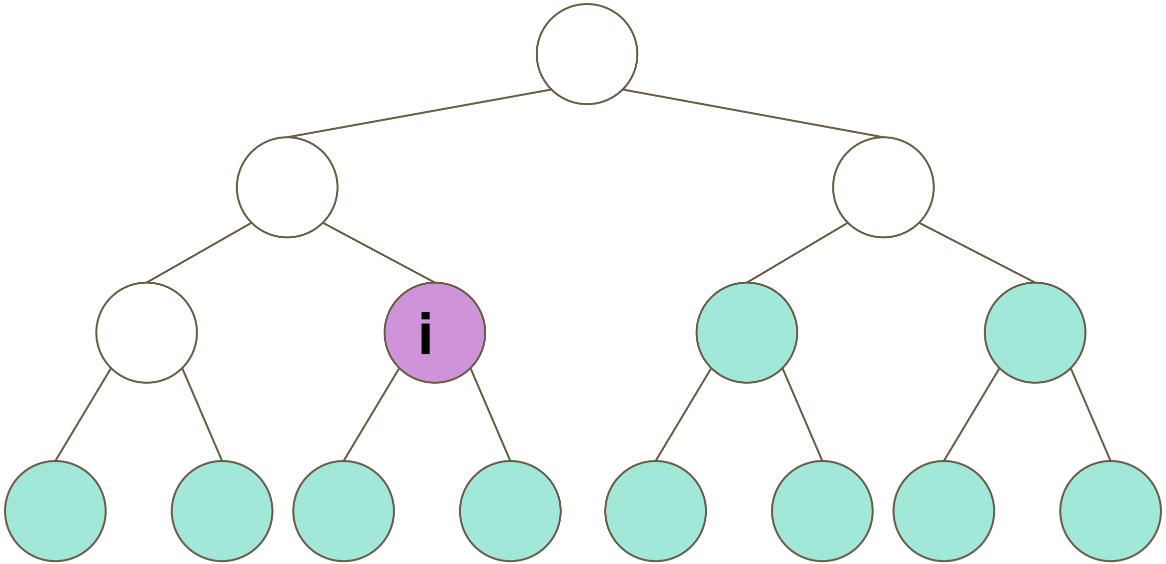
Caso base

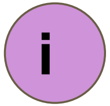
n = 2k
floor(n/2) = k
floor(n/2) + 1 = k + 1
los hijos de k + 1 serían
2(k+1)   y   2(k+1) + 1
2k + 2   y   2k + 3
n + 2   y   n + 3

n = 2k + 1
floor(n/2) = k
floor(n/2) + 1 = k + 1
los hijos de k + 1 serían
2(k+1)   y   2(k+1) + 1
2k + 2   y   2k + 3
n + 1   y   n + 2

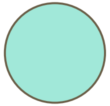
Solo tenemos elementos hasta la posición n por ende no existen elementos ni en n+1, n+2 ni en n+3.

Paso inductivo



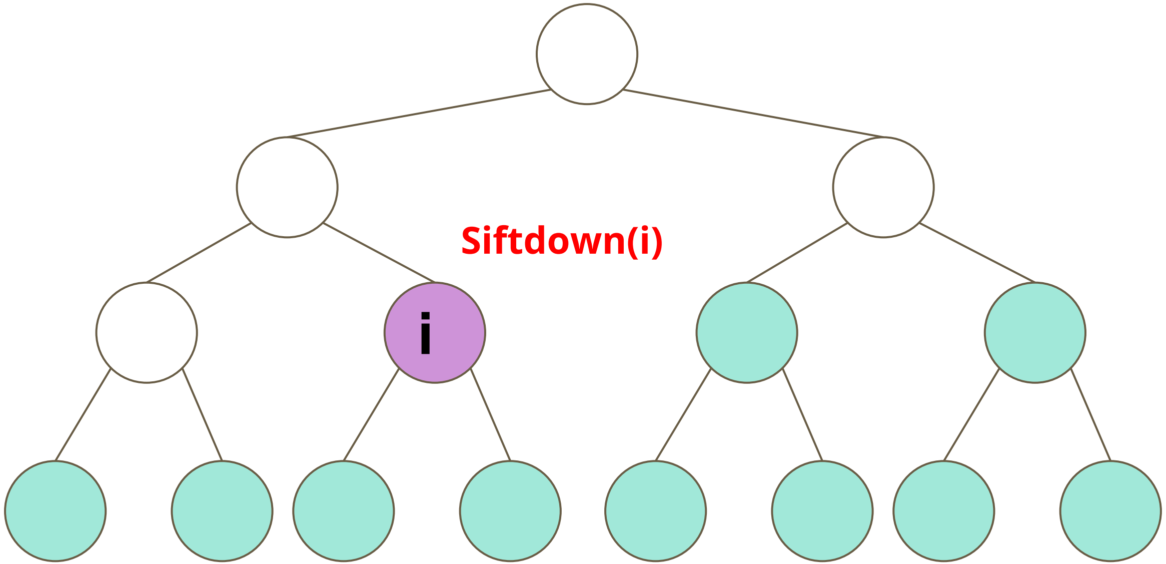


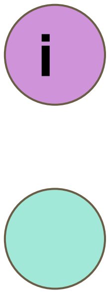
Elemento de la iteración i



Elementos en posiciones mayores a i que son raíces de heaps

Paso inductivo





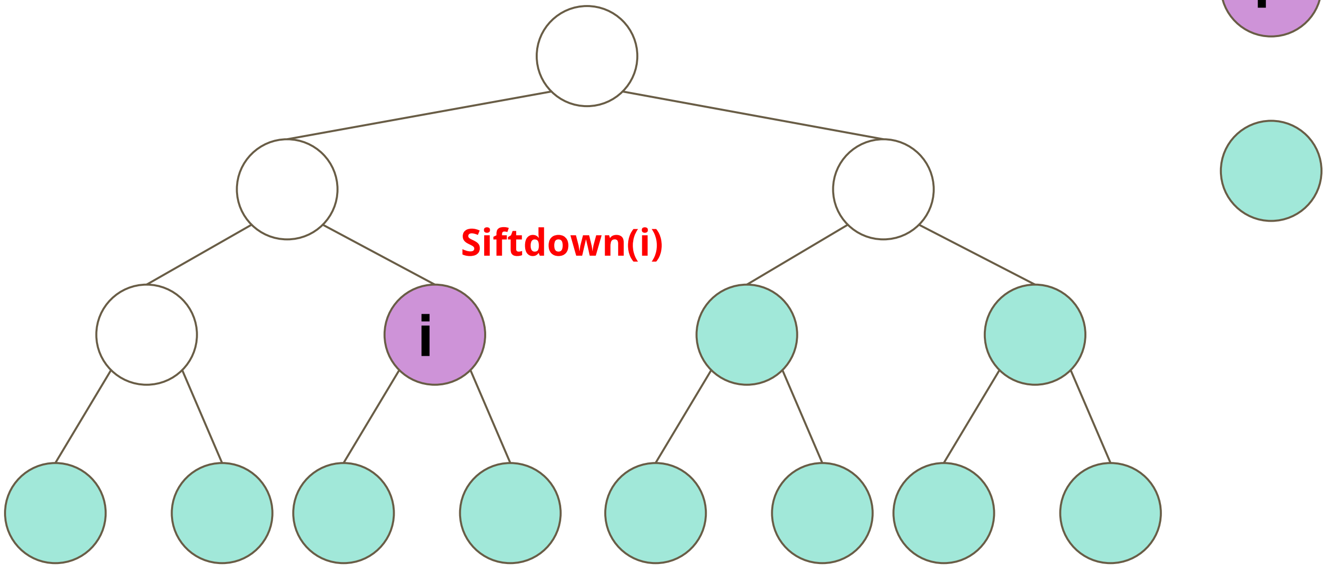
Elemento de la iteración i

Elementos en posiciones mayores a i que son raíces de heaps

Paso inductivo

si hacemos Siftdown de un elemento cuyos dos hijos ya son raíces de heaps, el resultado final será un heap!

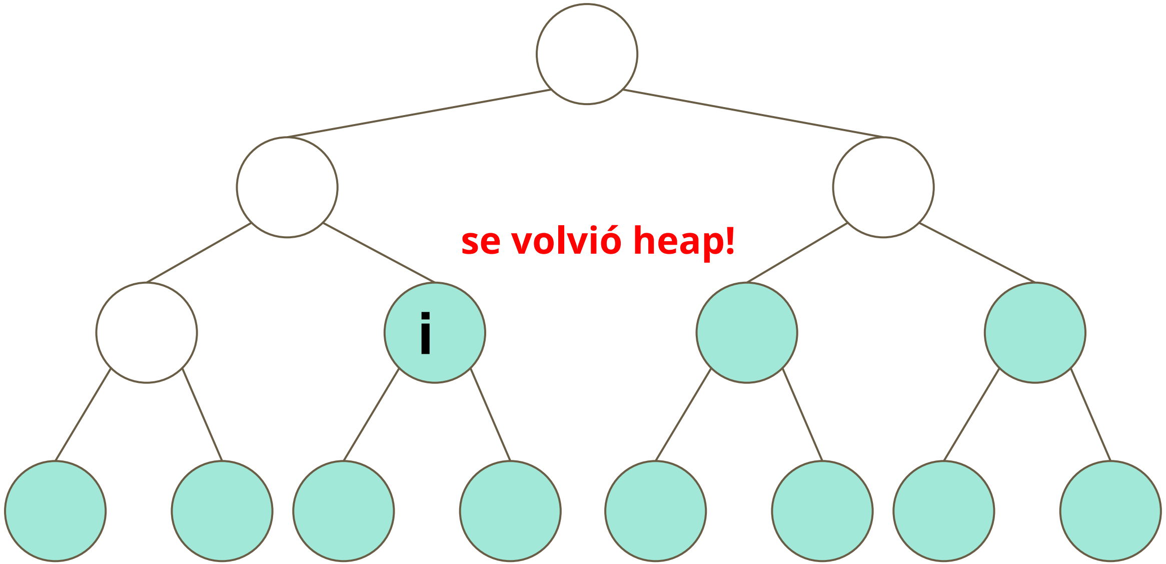
Paso inductivo



Elemento de la iteración i

Elementos en posiciones mayores a i que son raíces de heaps

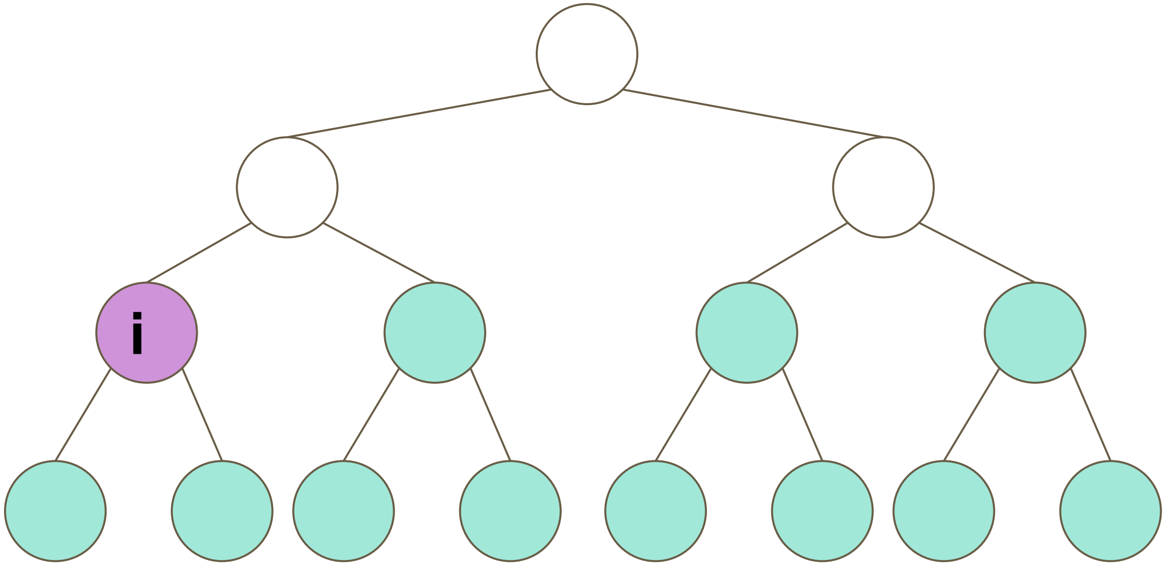
Paso inductivo

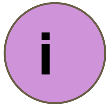


Elemento de la iteración i

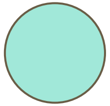
Elementos en posiciones mayores a i que son raíces de heaps

Paso inductivo





Elemento de la iteración i



Elementos en posiciones mayores a i que son raíces de heaps

Conclusión

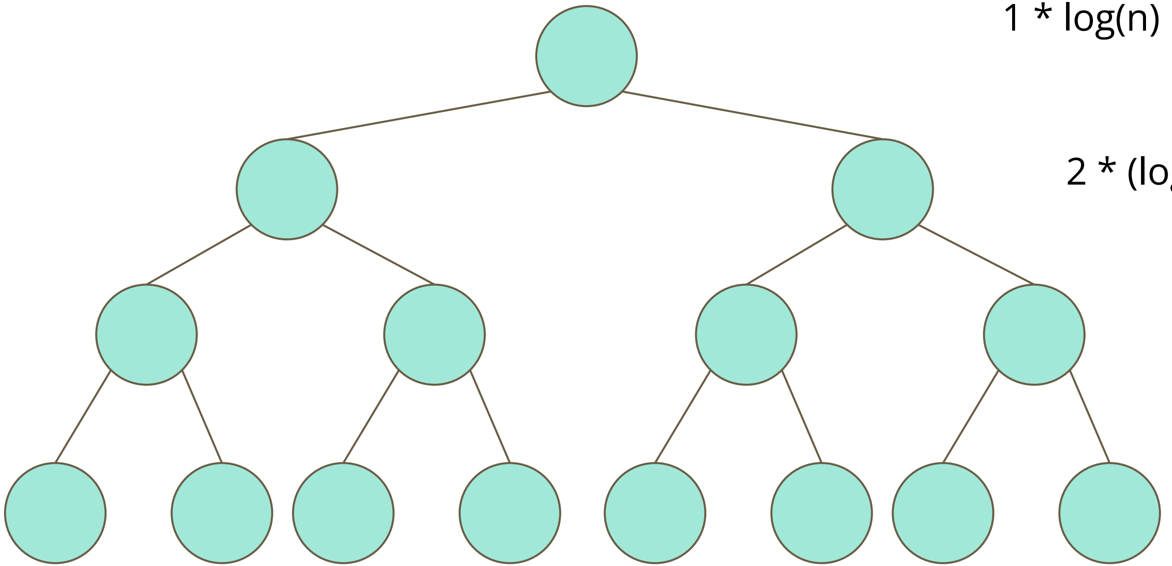
Finalmente, los 2 hijos del elemento de la posición 1 serán heaps y al aplicarle siftdown al elemento de la posición 1 todo se transformará en un heap.

Complejidad

La complejidad de aplicar siftdown depende de la altura. si la altura es h la complejidad de siftdown es O(h)

Complejidad

¿Cuántos elementos tiene cada nivel?
¿cuantas operaciones se hacen por cada elemento?



1 * log(n)

2 * (log(n) - 1)

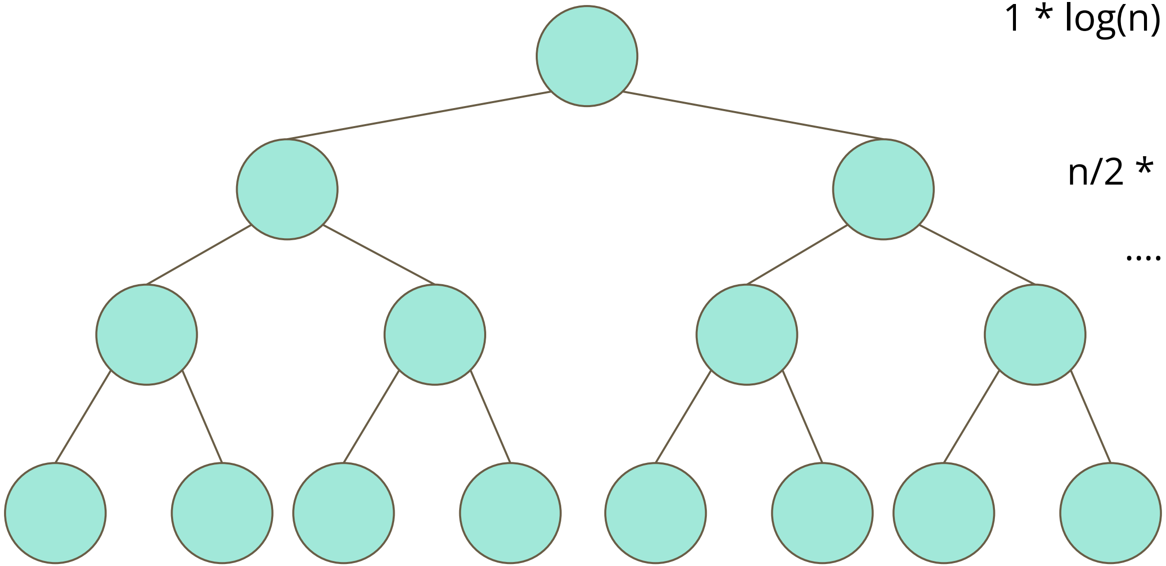
4 * (log(n) - 2)

....

2^(log(n) - 2) * (1)

Complejidad

¿Cuántos elementos tiene cada nivel?
¿cuantas operaciones se hacen por cada elemento?



$1 * \log(n)$

$n/2 * (\log(n) - 1)/2^{(\log(n)-2)}$

....

$n/2 * (2/2^2)$

$n/2 * (1/2)$

Complejidad

$\log(n) = k$

$1\log(n) + 2(\log(n) - 1) + 4(\log(n) + 2) + ... + 2^{(\log(n) - 2)} * (1)$

$= n/2*k/2^{(k)} + (k-1) + n/2*(k - 2)/2^{(k-2)} + ... + n/2 * 1/(2^1)$

$= n/2\sum i/2^{(i)}$ desde $i = 1$ hasta $i = \log(n)$

Complejidad

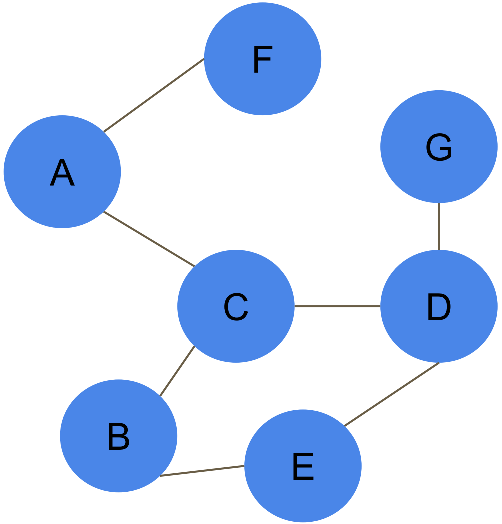
=∑a_i -> converge

lim(a_{i+1} / a_i)<1
lim (i + 1)/2^{i+1} / i/(2^i) = (i+1/i) * 2^{i}/2^{i+1} = 1/2

Complejidad

la sumatoria $\sum i/2^{(i)}$ converge por lo tanto no afecta en la suma total y la complejidad queda determinada por $n/2$ que es $O(n)$

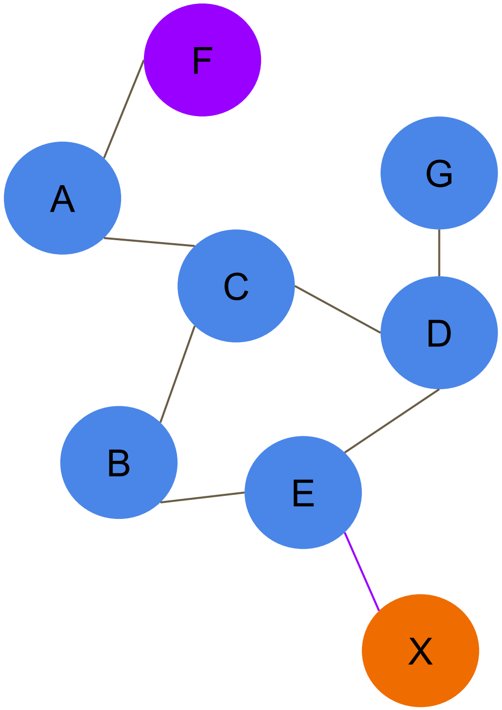
Union-Find



Problema: Dado un grafo no dirigido G, conocido desde antes (y las estructuras relacionadas que queramos), queremos saber en tiempo O(1) si podemos llegar desde un nuevo nodo a otro.

Se entregará el nuevo nodo y todas las conexiones que posee

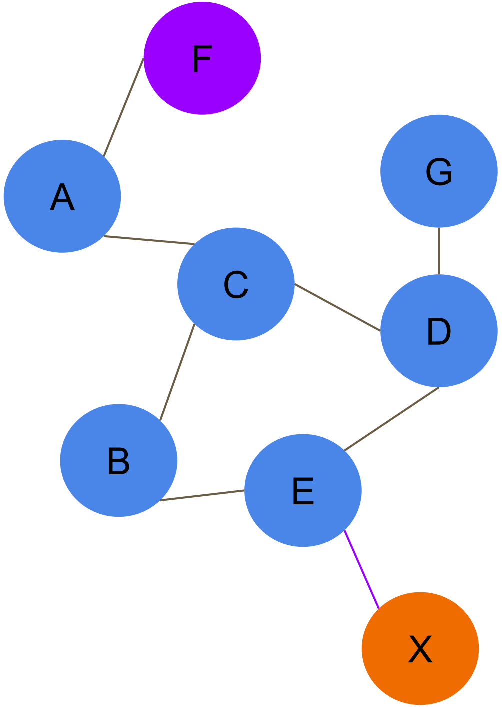
Union-Find



Supongamos que queremos detectar si el nuevo nodo X, con una arista hacia E, puede llegar a F

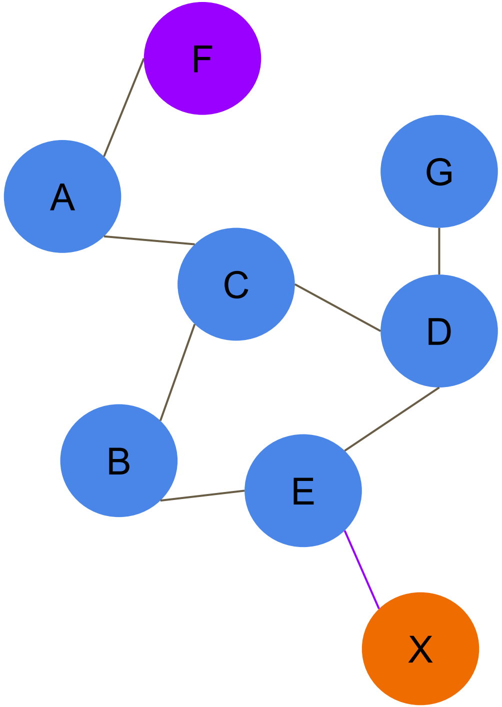
Podría parecer simple, utilizamos DFS desde X y retornamos true si encontramos a F en el camino. Sin embargo queremos hacerlo en O(1)

Union-Find



Como G es conocido a Priori.
Podemos utilizar Union-Find, y el algoritmo sería tan simple como revisar si E posee el mismo *representante* que F

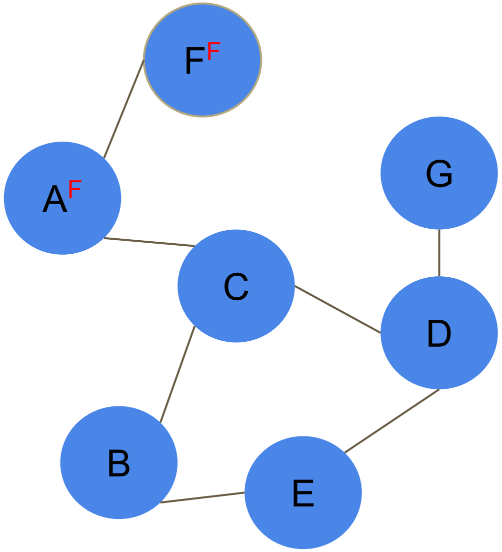
Union-Find



Como G es conocido a Priori.
Podemos utilizar Union-Find, y el algoritmo sería tan simple como revisar si E posee el mismo *representante* que F

Veámoslo en la práctica

Union-Find

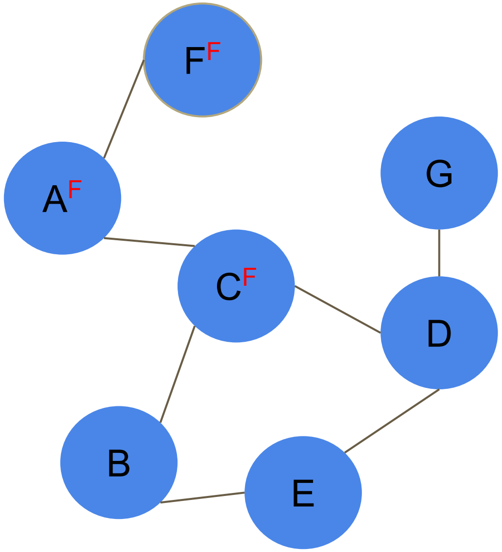


Aplicamos el algoritmo, Recordando que itera por aristas y no por nodo.

Aristas = [(F,A), (A,C), (D,G), (C,B), (C,D), (B,E), (E,D)]

Comenzamos, (F, A). Ni F ni A tienen representantes por lo que a ambos les asignamos F

Union-Find

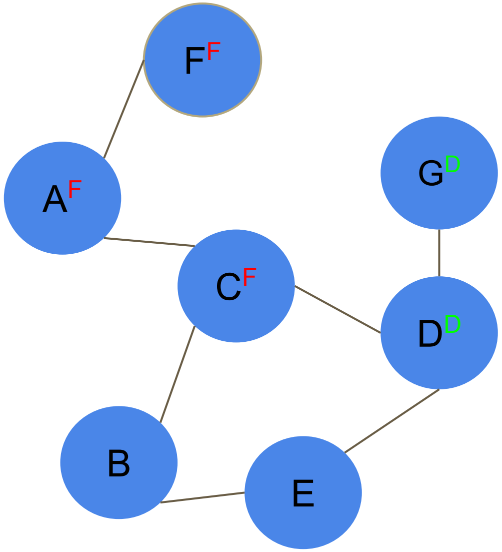


Aristas = [(F,A), (A,C), (D,G), (C,B), (C,D), (B,E), (E,D)]

(A,C). C toma A como representante, pero A tiene representante por lo que se lo propaga.

C queda con F como representante

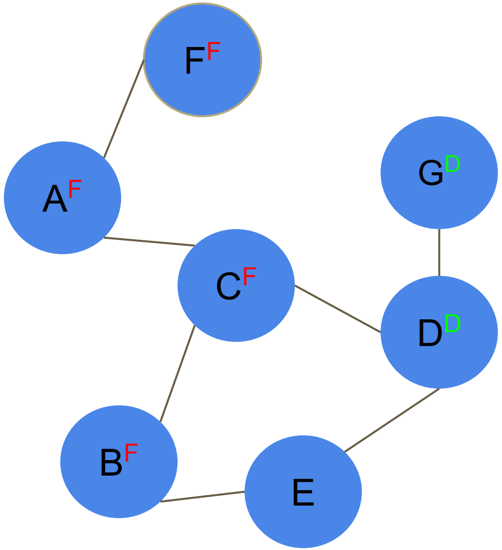
Union-Find



Aristas = [(F,A), (A,C), (D,G), (C,B), (C,D), (B,E), (E,D)]

(D,G), Ninguno tiene representante, Asignamos D
G y D queda con D como representante

Union-Find

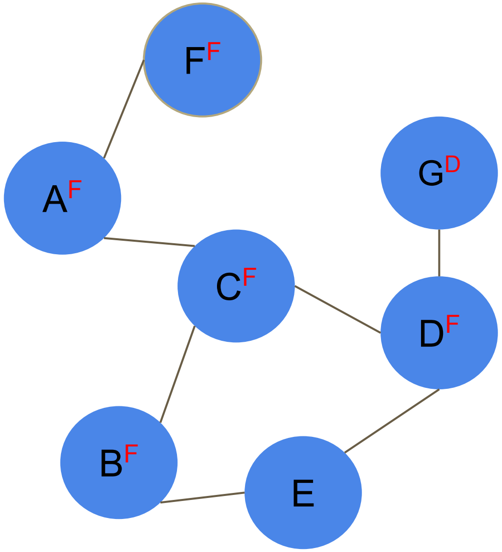


Aristas = [(F,A), (A,C), (D,G), (C,B), (C,D), (B,E), (E,D)]

(D,G), Ninguno tiene representante, Asignamos D G y D queda con D como representante

(C, B), B se asigna C, pero C propaga F

Union-Find



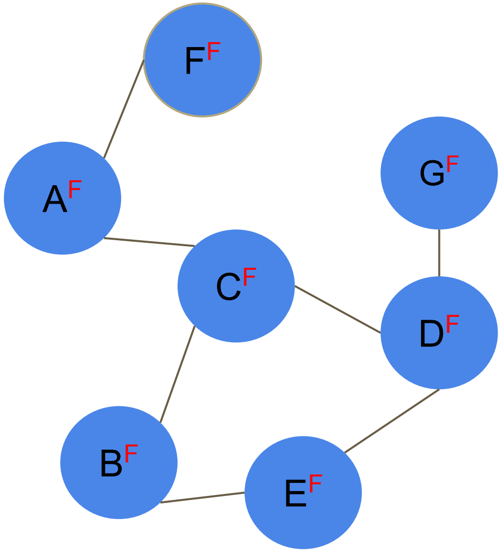
Aristas = [(F,A), (A,C), (D,G), (C,B), (C,D), (B,E), (E,D)]

(D,G), Ninguno tiene representante, Asignamos D
G y D queda con D como representante

(C, B), B se asigna C, pero C propaga F

(C,D), C y D tienen ambos representantes. Sin embargo vemos que el rank del representante de C es mayor (4>2). Por lo que propagamos F al representante D

Union-Find



Aristas = [(F,A), (A,C), (D,G), (C,B), (C,D), (B,E), (E,D)]

(D,G), Ninguno tiene representante, Asignamos D
G y D queda con D como representante

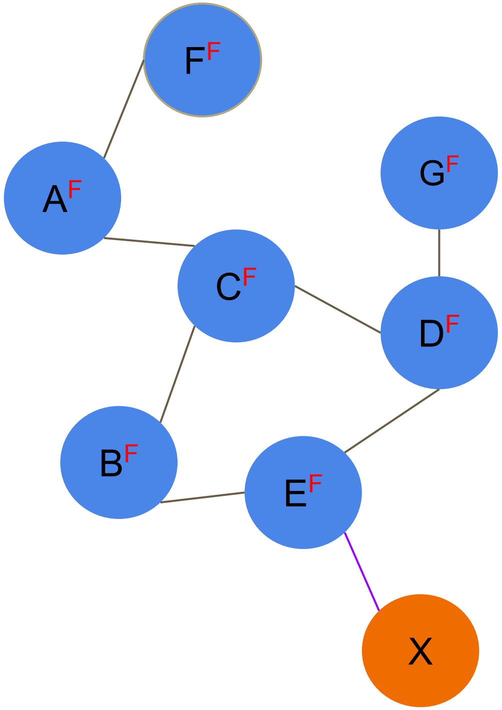
(C, B), B se asigna C, pero C propaga F

(C,D), C y D tienen ambos representantes. Sin embargo vemos que el rank del representante de C es mayor (4>2). Por lo que cambiamos el representante de D y propagamos

(B, E), Lo mismo

(E, D) detectamos que ambos corresponden al mismo representante por lo que no hacemos nada

Union-Find



Ahora nuestro SET esta completo, todos los nodos tienen como representante a F (incluido F que se referencia a si mismo)

Ahora el problema original. Entra (X, E).

Revisamos que E tiene representante F, y F a su vez tiene representante F. Como ambos poseen el mismo representante, pertenecen al mismo conjunto. Lo que nos ASEGURA que existe una forma de llegar de X a F.

La revisión, como se observo fue en O(1)

Ayudantía 8: Heap

Carlos Paredes : cparedesr@uc.cl
Alonso Carrasco : cristian.carrasco@uc.cl