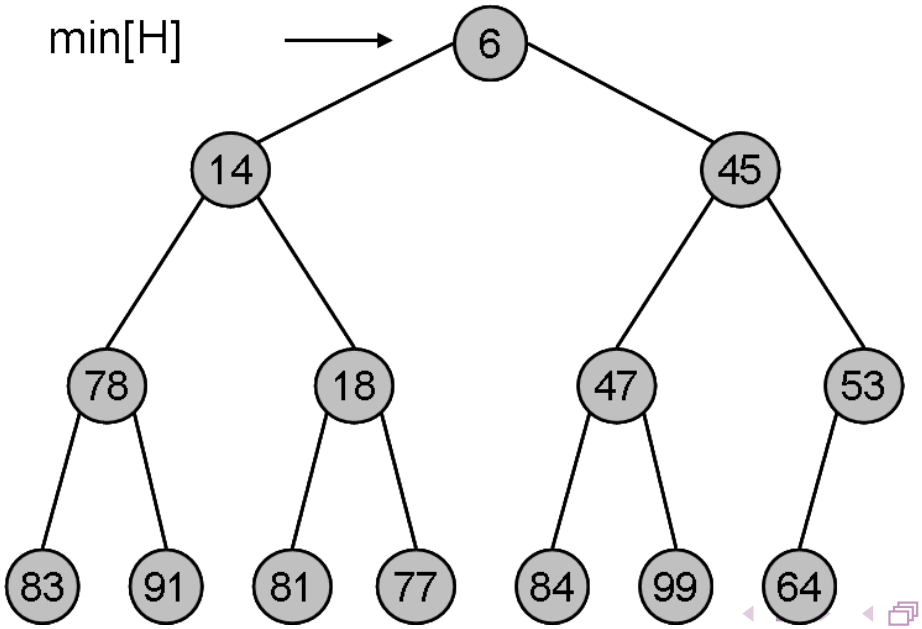
Fibonacci Heaps

Fabiola Di Bartolo

Universidad Simón Bolívar

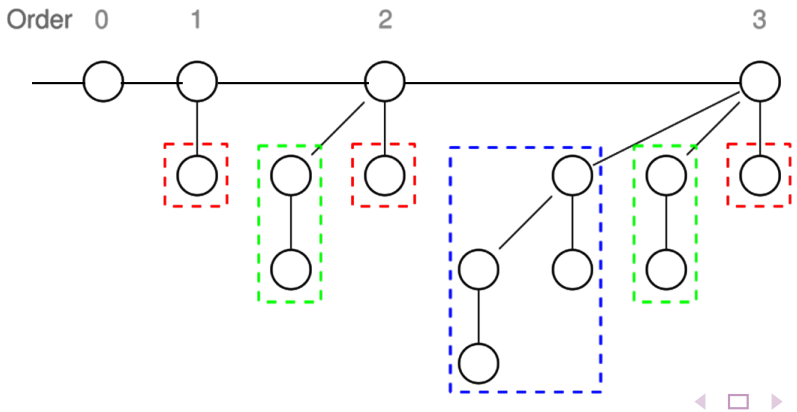
Binary Heaps

- Arbol binario casi completo:
  - Todos los niveles deben estar llenos, a excepción del último que se llena de izquierda a derecha.
- *Min-heap-ordered*:
  - Cada nodo es menor o igual que sus hijos.



Binomial Heaps

- Colección de árboles binomiales.
- Arbol binomial B_k: es un árbol ordenado, que consiste en dos arboles binomiales B_{k-1} enlazados (la raíz de uno es el hijo más izquierdo de la raíz del otro).
- Propiedades del heap binomial:
  - Propiedad *min-heap*: Cada nodo es menor o igual que sus hijos.
  - Para cualquier k entero no negativo, existe a lo más un árbol binomial en H cuya raíz tenga grado k (número de sucesores).



Fibonacci Heaps

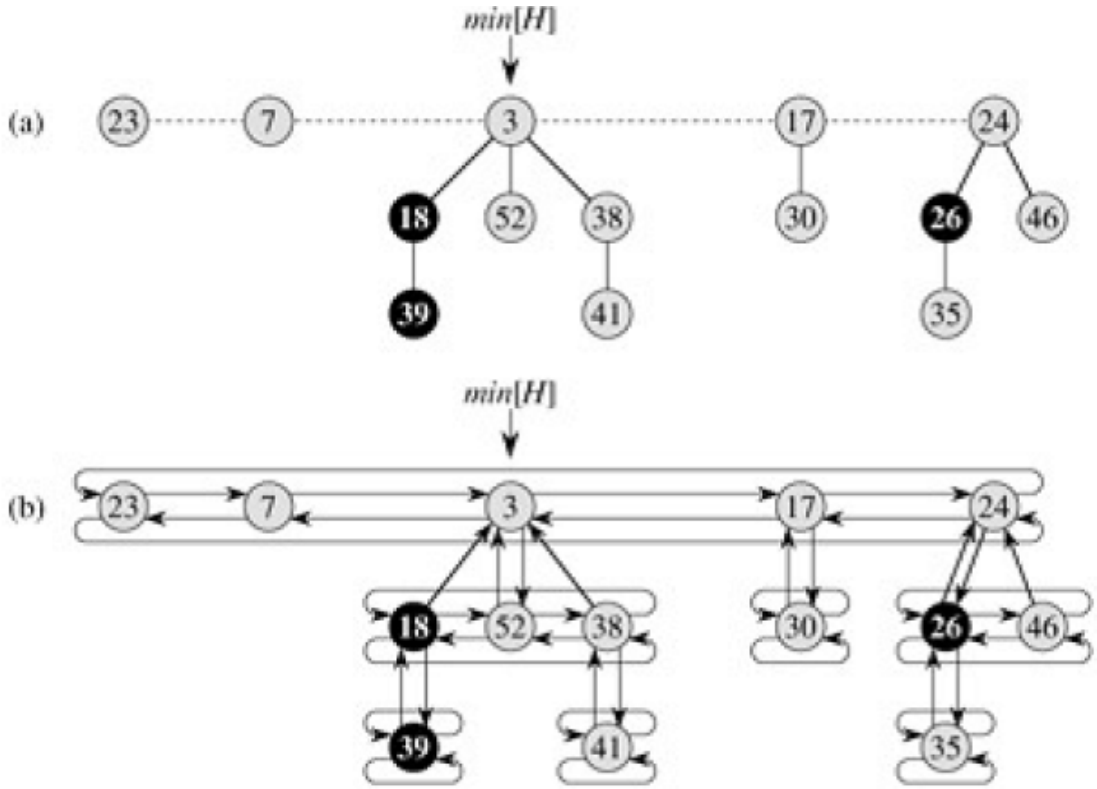
- Colección de árboles *min-heap-ordered*. Similar a los heaps binomiales (si no se realizan operaciones de disminuir la clave, o eliminar)
- Introducido por Michael L. Fredman y Robert E. Tarjan en 1984.
- Usado en colas de prioridades para mejorar el tiempo de los algoritmos Dijkstra (encontrar caminos mínimos) y Prim (calcular el mínimo árbol cobertor de un grafo).
- El nombre proviene de los números Fibonacci utilizados para el análisis del tiempo de ejecución.
- Tiene un costo amortizado mejor que los heaps binomiales y binarios.

Comparación de los distintos heaps

Tiempo de ejecución de las operaciones

Procedure	Binary heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$O(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$O(\lg n)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$O(\lg n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$

Estructura de los Fibonacci Heaps



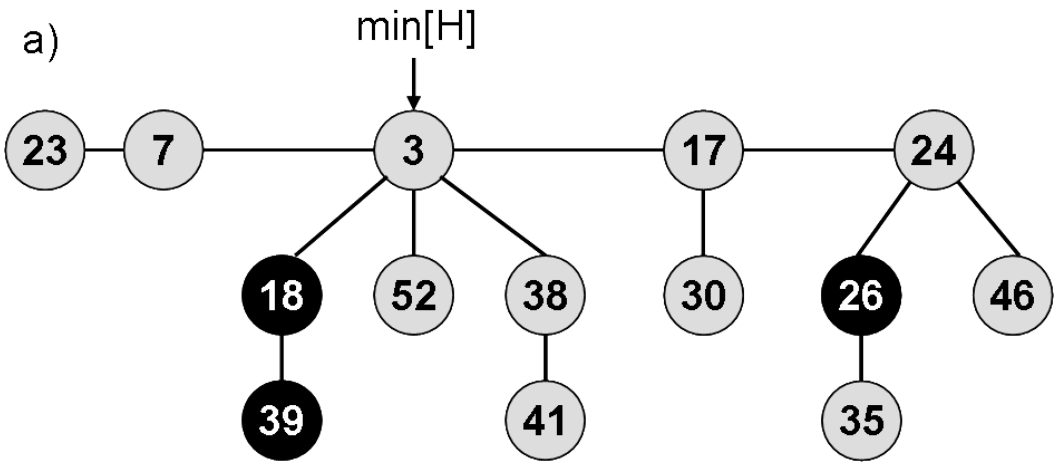
- **Apuntadores de un nodo x:**
  p[x]: padre
  child[x]:cualquiera de sus hijos
  left[x]: hermano izquierdo
  right[x]: hermano derecho
- **Campos de x**:
  degree[x]: ♯ de hijos de x
  mark[x]: indica si x perdió un hijo desde la última vez que x fue hecho hijo de otro nodo.
- **Heap H:**
  min[H]: mínimo nodo
  n[H]: ♯ de nodos en H

Propiedades

- Si sólo se realizan las operaciones MAKE-HEAP, INSERT, MINIMUM, EXTRACT-MIN y UNION, entonces cada **Fibonacci heap es una colección de árboles binomiales no ordenados.**
- Un **árbol binomial no ordenado** U_k consiste en dos árboles binomiales no ordenados U_{k-1}, donde la raíz de uno se coloca como cualquier hijo del otro. U_0 consiste en un solo nodo.
- Propiedades de árbol binomial no ordenado U_k:
  1. Tiene 2^k nodos
  2. La altura del árbol es k
  3. Existen exactamente $\binom{k}{i}$ nodos a la profundidad i para $i = 0, 1, ..., k$
  4. La raíz tiene grado k, el cual es mayor que el de cualquier otro nodo. Los hijos de la raíz son raíces de los subárboles $U_0, U_1, ..., U_{k-1}$ en algún orden.

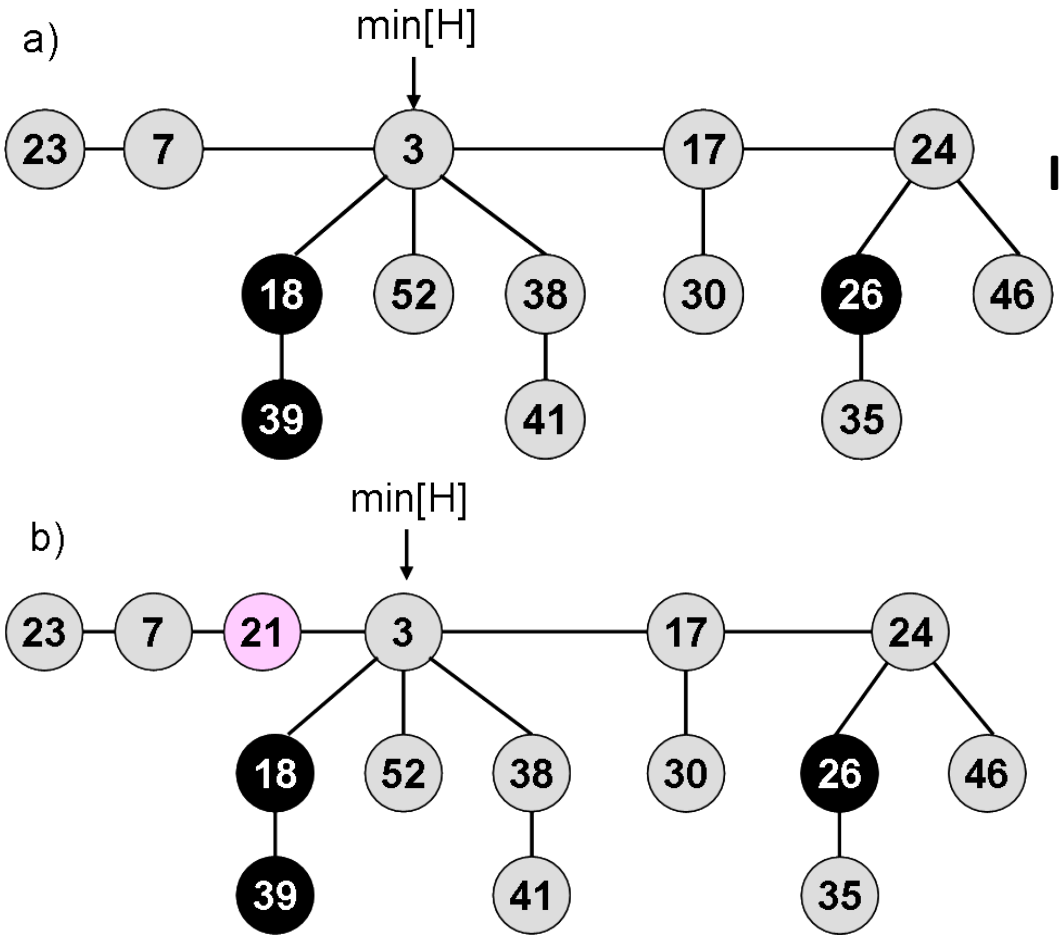
Función potencial

- Dado un heap H:
  - $t(H)$ el número de árboles en la lista de raíces
  - $m(H)$ el número de nodos marcados.
- La función potencial de un Fibonacci heap es: $\phi(H) = t(H) + 2m(H)$
- Idea clave: retrasar el trabajo lo más que se pueda.



$$\phi(H) = 5 + 2 \cdot 3 = 11$$

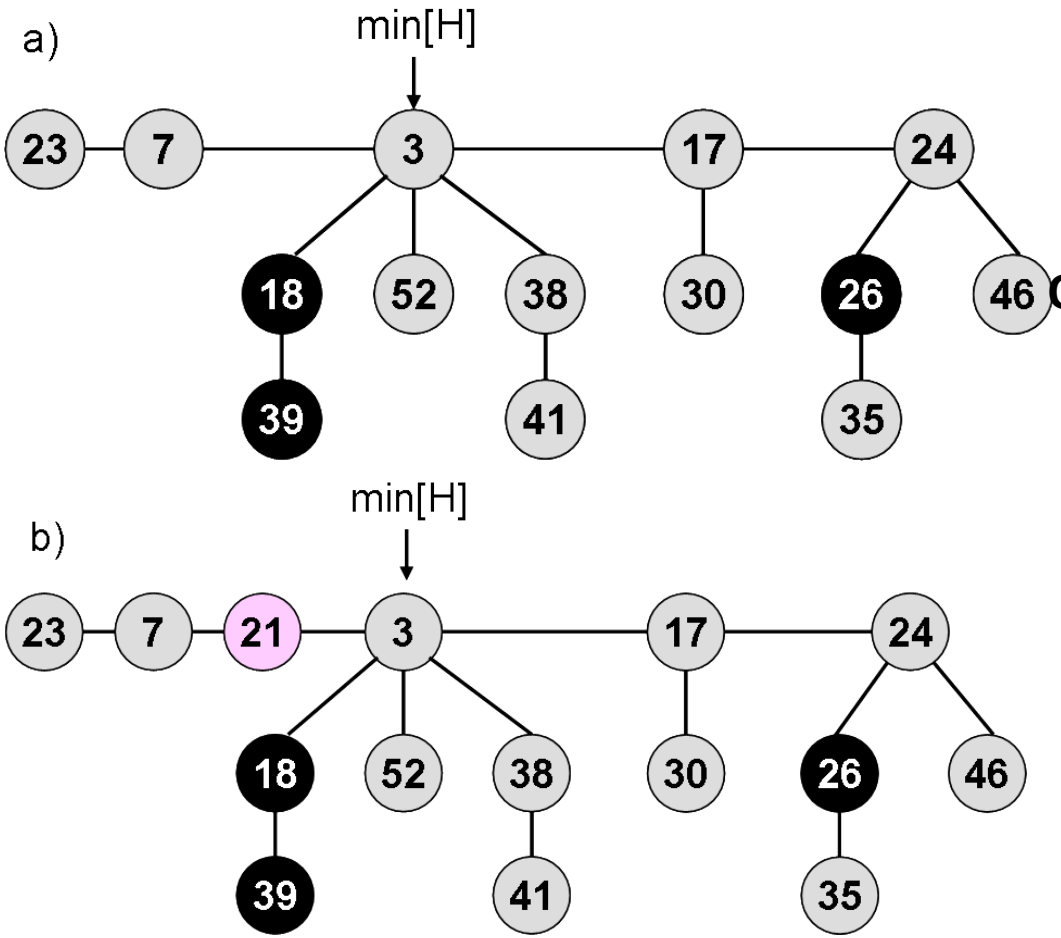
Insertando un nodo



INSERT(H,x)

1. Crea un nuevo árbol sólo con el nodo x
2. Agrega el árbol en la lista de raíces, a la izquierda del mínimo.
3. Actualiza min[H] si es necesario
4. Actualiza n[H] en n[H] + 1.

Insertando un nodo



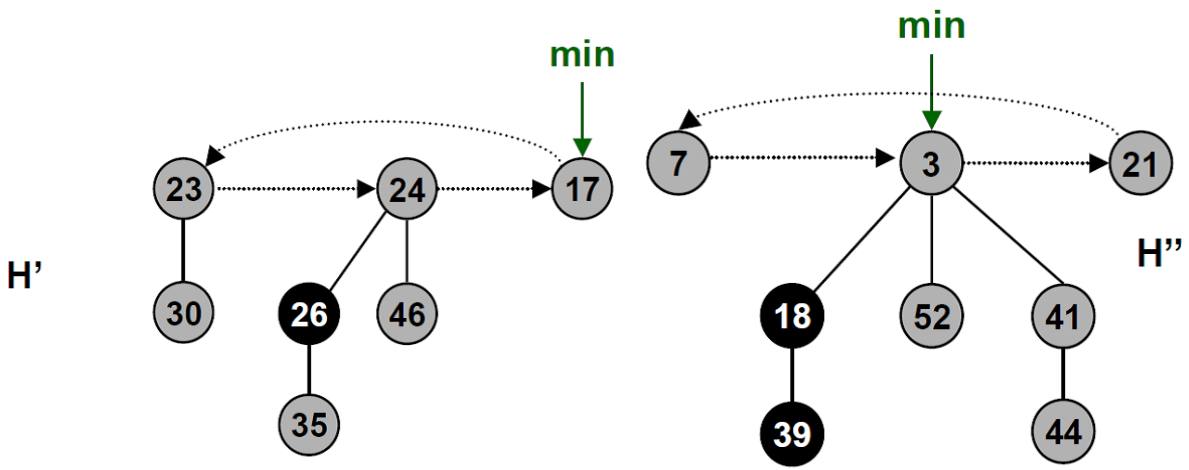
Costo Amortizado

- $\phi(H) = t(H) + 2m(H)$
- Costo de la operación$= O(1)$
- Diferencia de potencial$= 1$
- Costo amortizado$= 1 + O(1)$

Uniendo dos Fibonacci Heaps

UNION(H', H'')

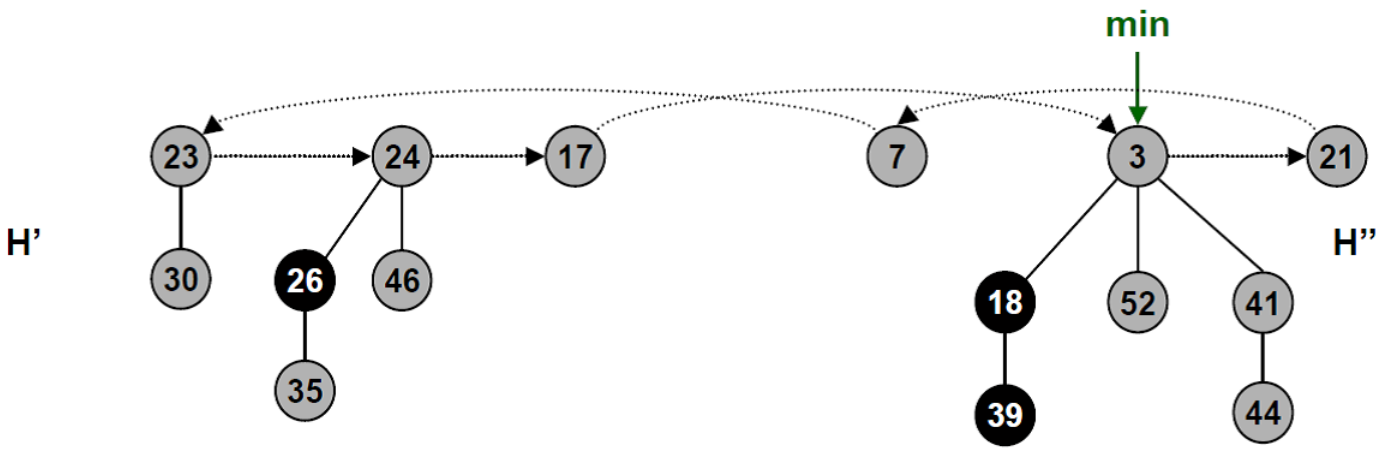
1. Concatena los heaps H' y H''
2. $min[H]$ el mínimo de los dos mínimos.
3. $n[H] = n[H'] + n[H'']$.



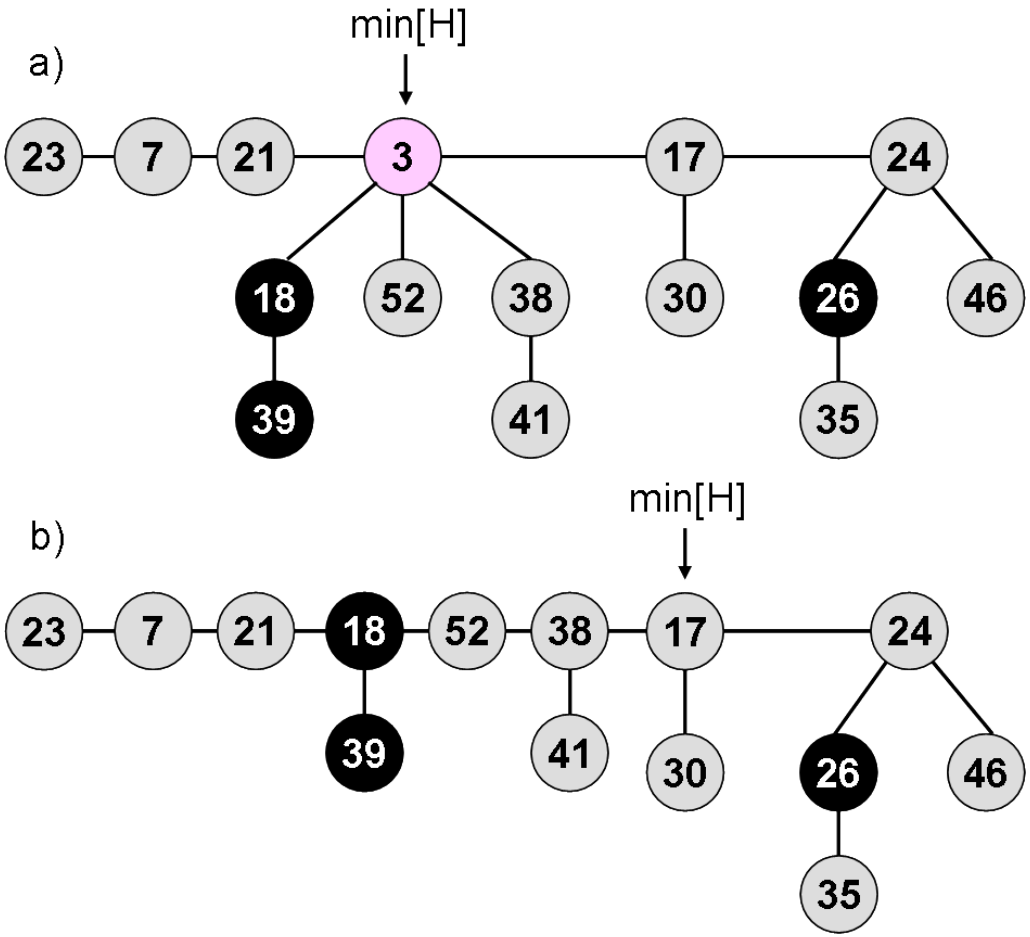
Uniendo dos Fibonacci Heaps - Análisis

Costo Amortizado

- $\phi(H) = t(H) + 2m(H)$
- $t(H) = t(H') + t(H'')$, $m(H) = m(H') + m(H'')$
- Costo de la operación $= O(1)$
- Diferencia de potencial $= \phi(H) - (\phi(H') + \phi(H'')) = 0$
- Costo amortizado $= O(1)$



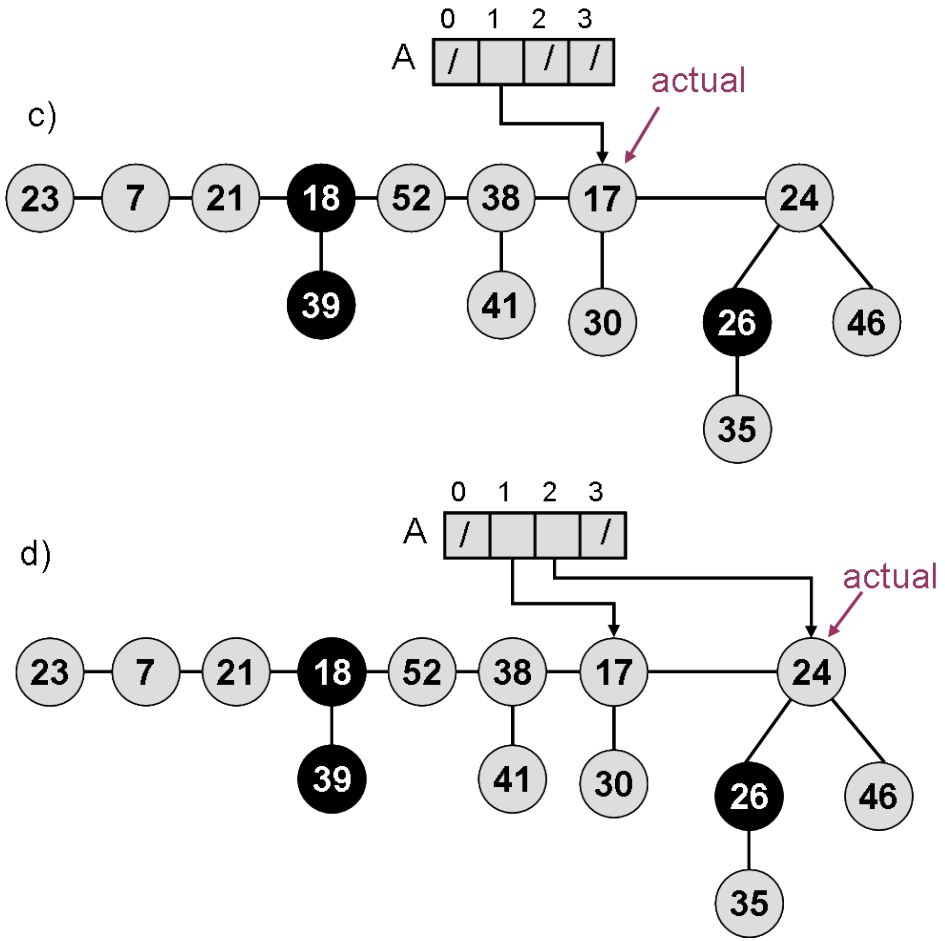
Eliminando el mínimo



EXTRACT-MIN(H)

1. Agrega todos los hijos del mínimo en la lista de raíces.
2. Elimina el mínimo
3. Temporalmente min[H] será el vecino derecho.

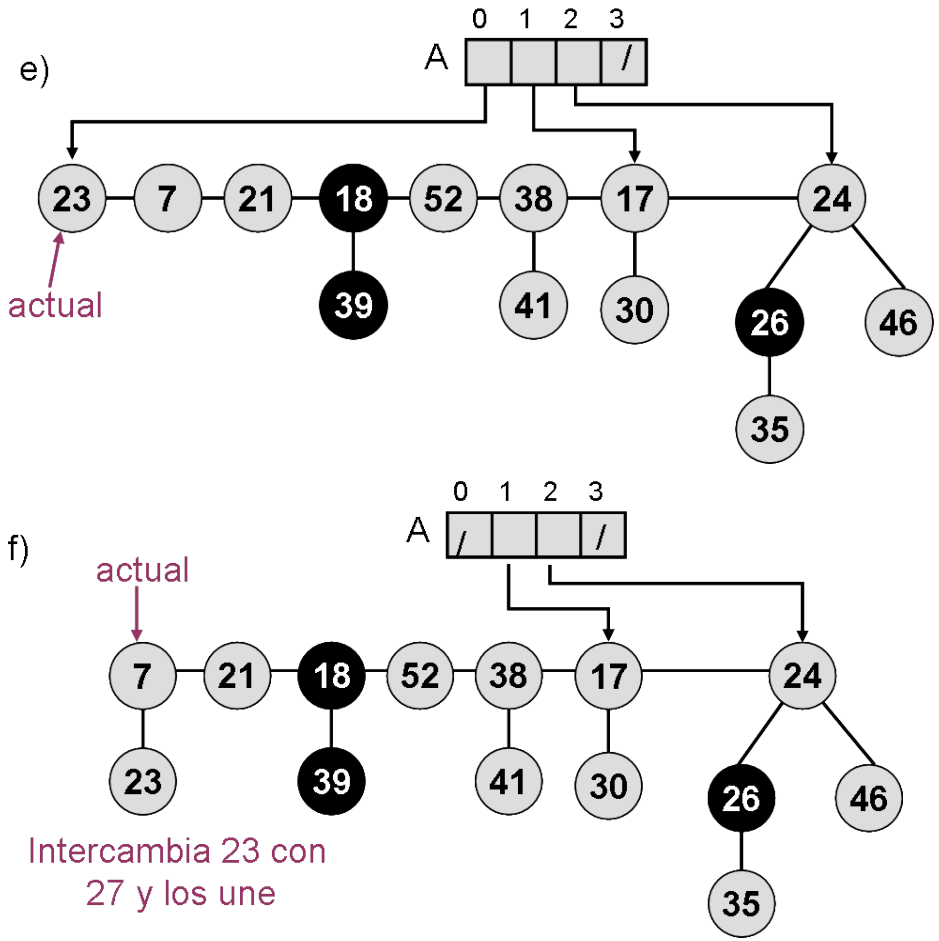
Eliminando el mínimo



EXTRACT-MIN(H)

1. A es un arreglo de tamaño $D(n) + 1$ $(lg(14) \approx 3,8)$
2. Consolida la lista de raíces, enlazando raíces de igual grado,
3. Termina cuando todas las raíces tengan grados distintos
4. Encuentra un nuevo mínimo.

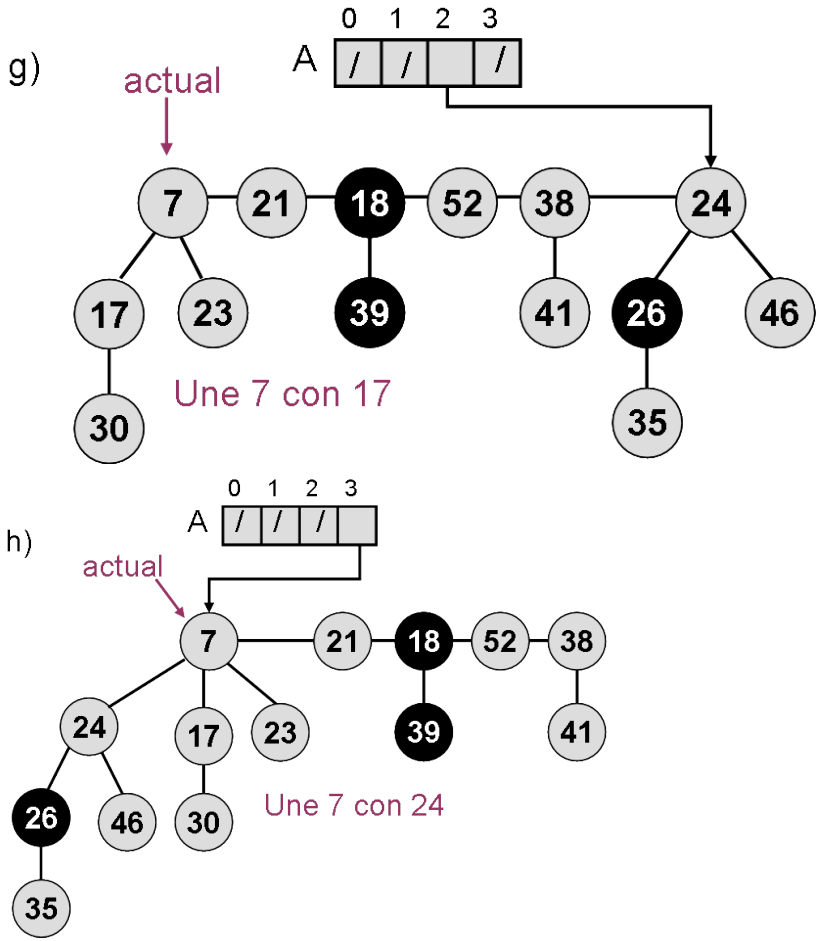
Eliminando el mínimo



EXTRACT-MIN(H)

1. A es un arreglo de tamaño $D(n) + 1$ $(lg(14) \approx 3,8)$
2. Consolida la lista de raíces, enlazando raíces de igual grado,
3. Termina cuando todas las raíces tengan grados distintos
4. Encuentra un nuevo mínimo.

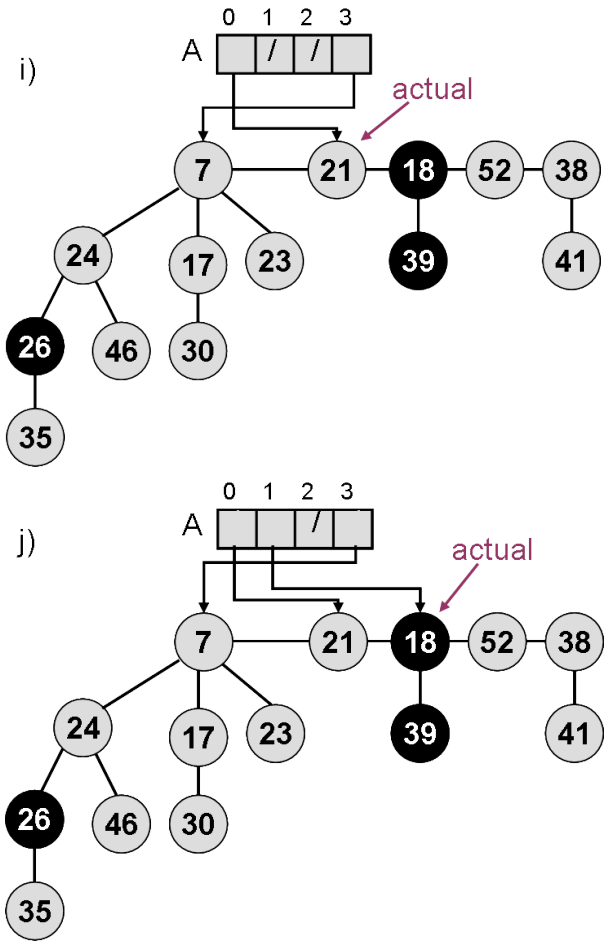
Eliminando el mínimo



EXTRACT-MIN(H)

1. A es un arreglo de tamaño $D(n) + 1$ $(lg(14) \approx 3,8)$
2. Consolida la lista de raíces, enlazando raíces de igual grado,
3. Termina cuando todas las raíces tengan grados distintos
4. Encuentra un nuevo mínimo.

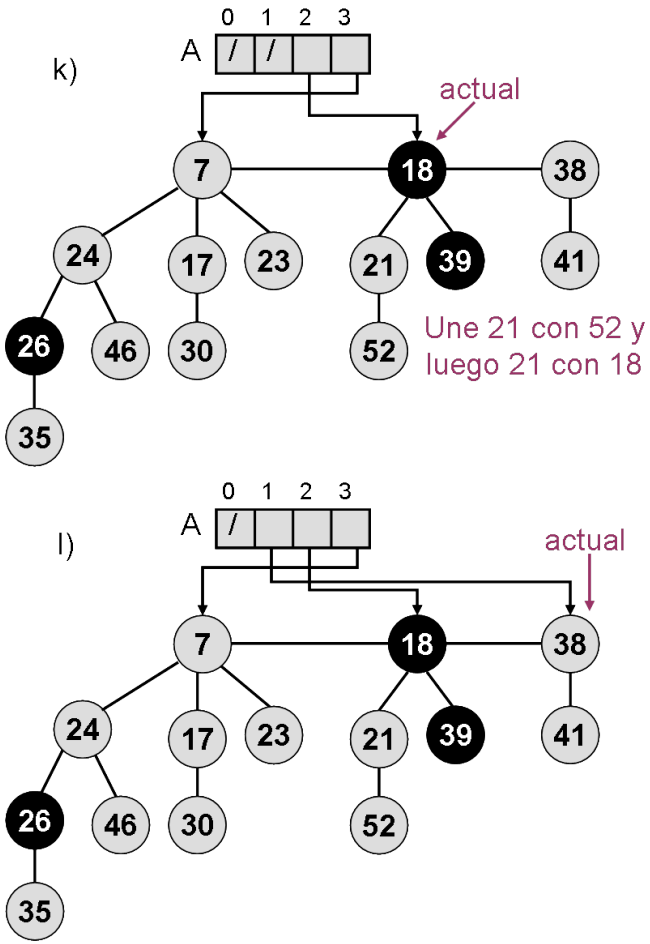
Eliminando el mínimo



EXTRACT-MIN(H)

1. A es un arreglo de tamaño $D(n) + 1$ $(lg(14) \approx 3,8)$
2. Consolida la lista de raíces, enlazando raíces de igual grado,
3. Termina cuando todas las raíces tengan grados distintos
4. Encuentra un nuevo mínimo.

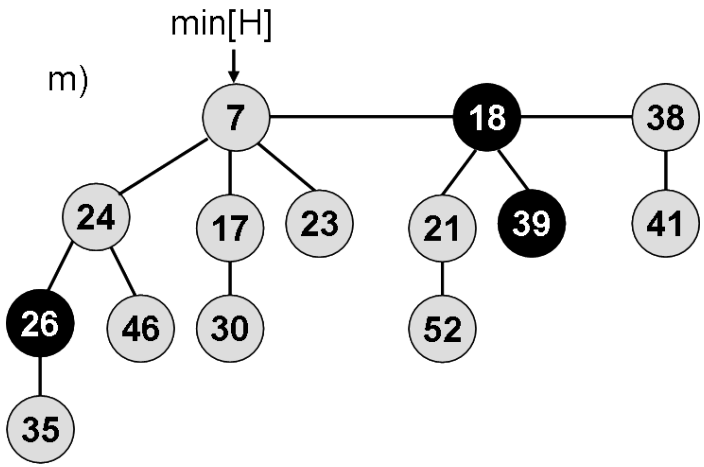
Eliminando el mínimo



EXTRACT-MIN(H)

1. A es un arreglo de tamaño $D(n) + 1$ $(lg(14) \approx 3,8)$
2. Consolida la lista de raíces, enlazando raíces de igual grado,
3. Termina cuando todas las raíces tengan grados distintos
4. Encuentra un nuevo mínimo.

Eliminando el mínimo



EXTRACT-MIN(H)

1. Heap resultante.

Eliminando el mínimo - Análisis

Notación:

- $D(n) =$ máximo grado de cualquier nodo en un Fibonacci heap de n
- $t(H) =$ número de árboles en el heap H.
- $\phi(H) = t(H) + 2m(H)$

Costo de la operación: $O(D(n) + t(H))$

- $O(D(n))$: proceso de colocar los **hijos como raíces**. A lo más el grado del mínimo será $D(n)$.
- $O(D(n) + t(H))$: proceso de **consolidación**. Proporcional al tamaño de la lista de raíces, el número de raíces decrece en uno luego de cada unión o mezcla. Al inicio de la consolidación son $D(n) + t(H) - 1$ raíces.

Costo amortizado: $O(D(n))$

- $t(H') \leq D(n) + 1$, no habrán dos árboles con el mismo grado.
- Diferencia de potencial $= ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H))$ $= D(n) + 1 - t(H)$
- Función potencial $= O(D(n) + t(H)) + (D(n) + 1) - t(H) = O(D(n))$

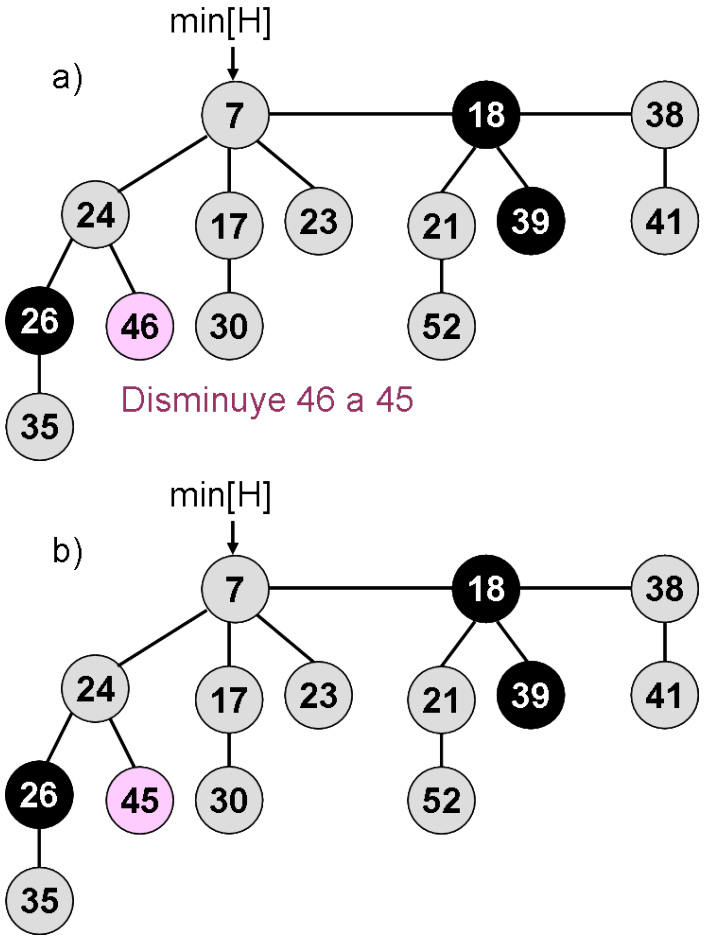
Eliminando el mínimo - Análisis

Cota para D(n)

- Si sólo soporta las operaciones MAKE-HEAP, INSERT, MINIMUM, EXTRACT-MIN y UNION en un Fibonacci heap de n nodos:
  - Tiene sólo árboles binomiales no ordenados ya que sólo se mezclan árboles que tengan raíces de igual grado.
  - $D(n) \leq \lfloor lg\ n \rfloor$
- Cuando soporta las operaciones DECREASE-KEY y DELETE en un Fibonacci heap de n nodos:
  - No preserva la propiedad de que todos los árboles sean binomiales no ordenados.
  - $D(n) = O(lg\ n)$

Disminuyendo una clave

Caso 0: propiedad min-heap no es violada

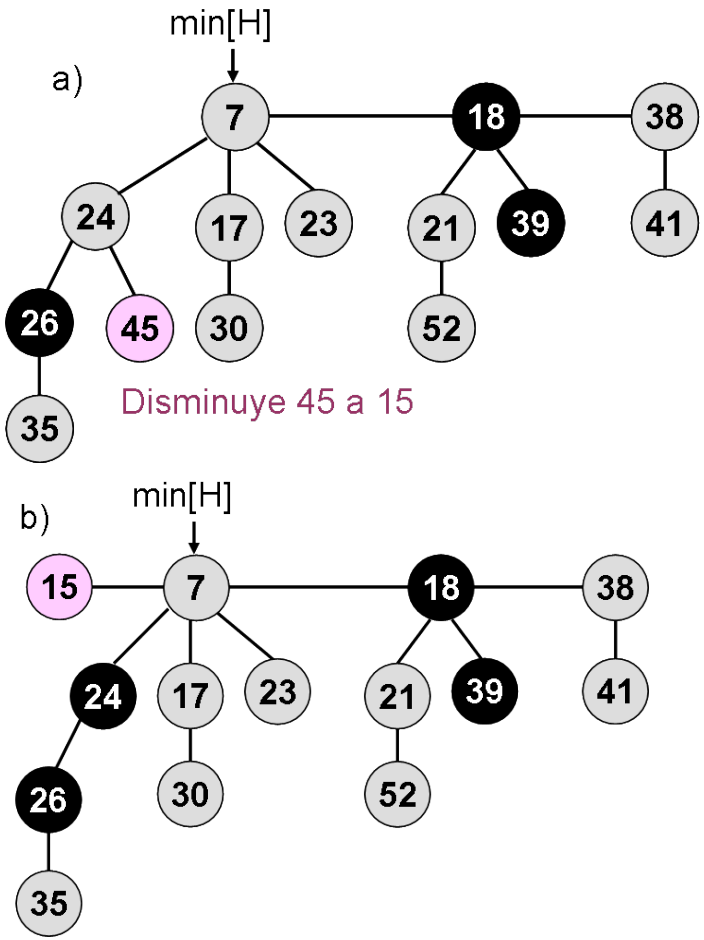


DECREASE-KEY(H,x,k)

1. Disminuye la clave de x (46 a 45)
2. Actualiza el apuntador al mínimo si es necesario

Disminuyendo una clave

Caso 1: padre de x no está marcado

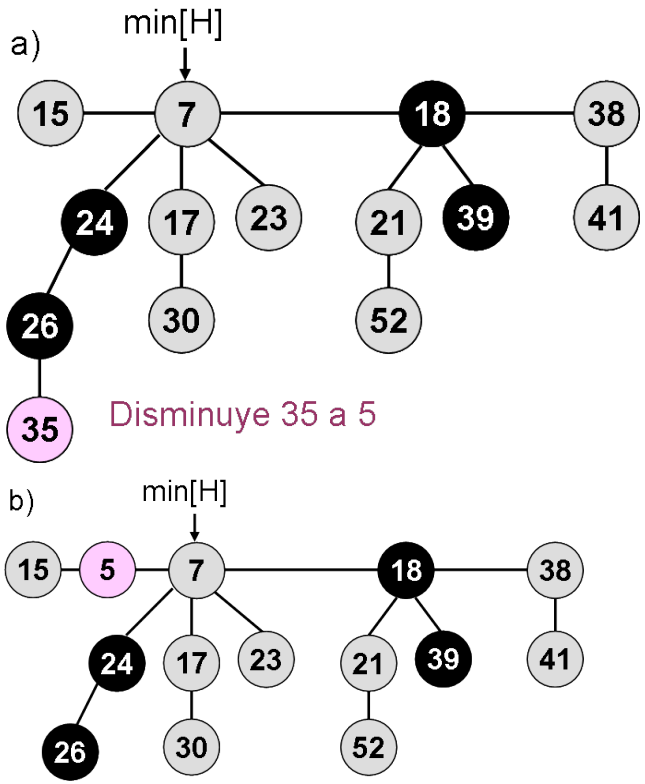


DECREASE-KEY(H,x,k)

1. Disminuye la clave de x (45 a 15)
2. Corta el link entre x y su padre
3. Marca al padre (perdió un hijo)
4. Agrega a x a la lista de raíces.
5. Actualiza el apuntador al mínimo si es necesario

Disminuyendo una clave

Caso 2: padre de x si está marcado

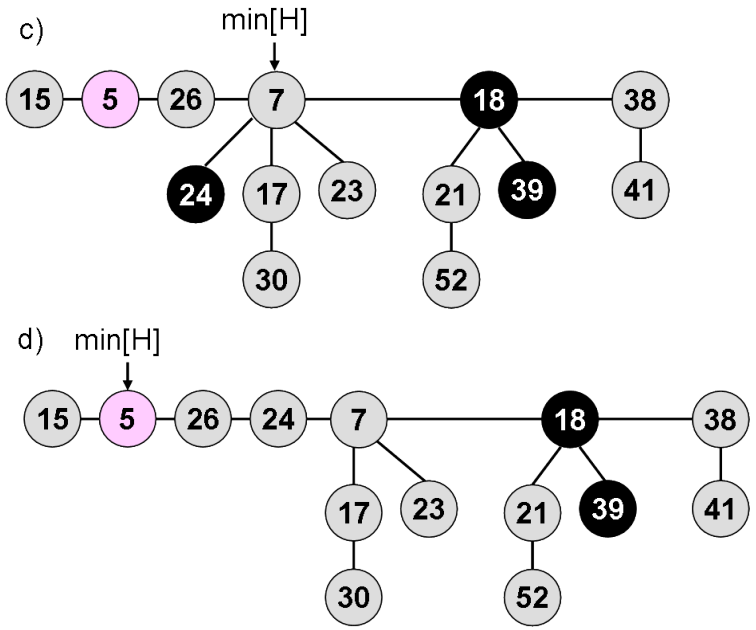


DECREASE-KEY(H,x,k)

1. Disminuye la clave de x a k (35 a 5)
2. Corta el link entre x y su padre p[x]
3. Agrega a x a la lista de raíces.
4. Actualiza el apuntador al mínimo si es necesario

Disminuyendo una clave

Caso 2: padre de x si está marcado



DECREASE-KEY(H,x,k)

1. Corta el link entre p[x] y su padre p[p[x]] (corte en cascada)
2. Desmarca p[x] y lo agrega a la lista de raíces.
   - Si p[p[x]] no está marcado, se marca (si no es raíz) y termina.
   - Si p[p[x]] si está marcado, se repite.

Disminuyendo una clave - Análisis

Notación:

- $t(H) =$ número de árboles en el heap H.
- $m(H) =$ número de nodos marcados en el heap H.
- $\phi(H) = t(H) + 2m(H)$

Costo de la operación: $O(c)$**.** (c es el número de nodos cortados)

- $O(1)$: para **disminuir la clave**.
- $O(1)$: c/u de los **cortes en cascada** y la **inserción en la lista de raíces**.

Costo amortizado: $O(1)$

- $t(H') = t(H) + c$: por el corte en cascada.
- $m(H') \leq m(H) - c + 2$: cada corte desmarca un nodo $(c - 1)$. El último corte en cascada podría marcar a un nodo.
- Diferencia de potencial $=$
  $((t(H) + c) + 2(m(H) - c + 2)) - (t(H) + 2m(H)) = 4 - c$
- Función potencial $= O(c) + 4 - c = O(1)$

Eliminando un nodo

DELETE(H,x)

1. Disminuye la clave de x a $-\infty$. DECREASE-KEY(H,x,$-\infty$)
2. Elimina el mínimo del heap. EXTRACT-MIN(H).

Costo Amortizado: $O(lg\ n)$

- $O(1)$: por DECREASE-KEY.
- $O(D(n))$: EXTRACT-MIN. $(D(n) = O(lgn))$

Acotando el máximo grado D(n)

Cota superior de D(n)

- Para probar que el tiempo amortizado de EXTRACT-MIN y DELETE es $O(lg\ n)$, se demuestra que una cota superior para $D(n)$ en un Fibonacci heap de n nodos es $O(lg\ n)$.
- Cuando todos los árboles son binomiales no ordenados, $D(n) = \lfloor lg\ n \rfloor$.
- Los cortes realizados por DECREASE-KEY, pueden generar árboles que violen la propiedad de binomiales no ordenados.
- Debido a que un nodo es cortado tan pronto pierde dos hijos, se demuestra que $D(n)$ es $O(lg\ n)$, en particular $D(n) \leq \lfloor log_\phi\ n \rfloor$, donde $\phi = (1 + \sqrt{5})/2$

Lema clave

Sea $size(x)$ el número de nodos del subárbol formado por x, incluyendo a x y $k = degree[x]$. Entonces $size(x) \geq F_{k+2} \geq \phi^k$.

Conclusiones

- Ineficiente para buscar un elemento que no sea el mínimo.
- Operaciones MAKE-HEAP, INSERT, MINIMUM, EXTRACT-MIN y UNION son $O(1)$
- Operaciones EXTRACT-MIN y DELETE son $O(lg\ n)$.
- Mejores tiempos que los heaps binomiales.