Análisis amortizado

- El plan:
  - – Conceptos básicos:
    - Método agregado
    - Método contable
    - Método potencial
  - – Primer ejemplo: análisis de tablas *hash* dinámicas
  - – Montículos agregables (binomiales y de Fibonacci)
  - – **Estructuras de conjuntos disjuntos**
  - – Listas lineales auto-organizativas
  - – Árboles auto-organizativos ("splay trees")

Estructuras de conjuntos disjuntos

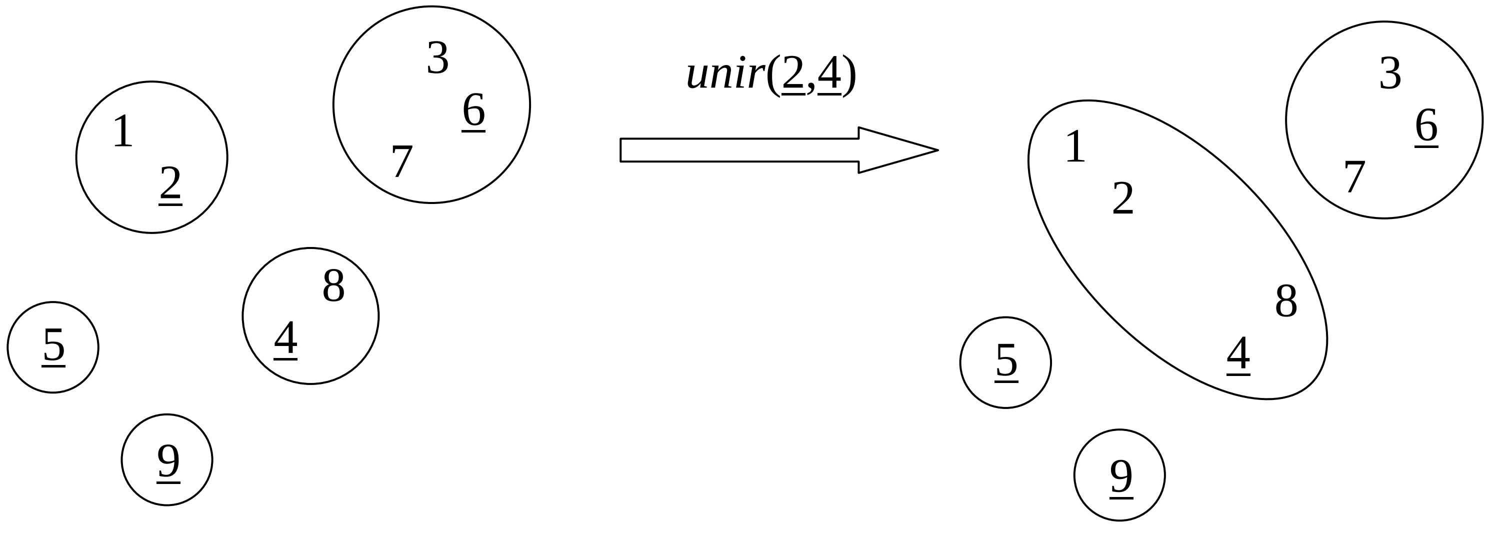
- Motivación:
  - Algoritmo de Kruskal para el cálculo del árbol de recubrimiento de peso mínimo de un grafo

```
var g:grafo no dirig. conexo etiquetado no negat.
    T:montículo de aristas; gsol:grafo; u,v:vért; x:etiq;
    C:estructura de conjuntos disjuntos; ucomp,vcomp:nat
creaECD(C); {cada vért. forma un conjunto}
creaVacío(gsol); creaVacía(T);
para todo v en vért, para todo <u,x> en adyac(g,v) hacer
  inserta(T,v,u,x)
fpara;
mq numConjuntos(C)>1 hacer
  <u,v,x>:=primero(T); borra(T);
  ucomp:=indConjunto(C,u); vcomp:= indConjunto(C,v);
  si ucomp≠vcomp entonces
    fusiona(C,ucomp,vcomp);  añade(gsol,u,v,x)
  fsi
fmq;
devuelve gsol
```

(extraído de “Esquemas Algorítmicos”)

Estructuras de conjuntos disjuntos

- Mantener una partición de S={1,2,…,n} con las operaciones de



encontrar(3)=6

Estructuras de conjuntos disjuntos

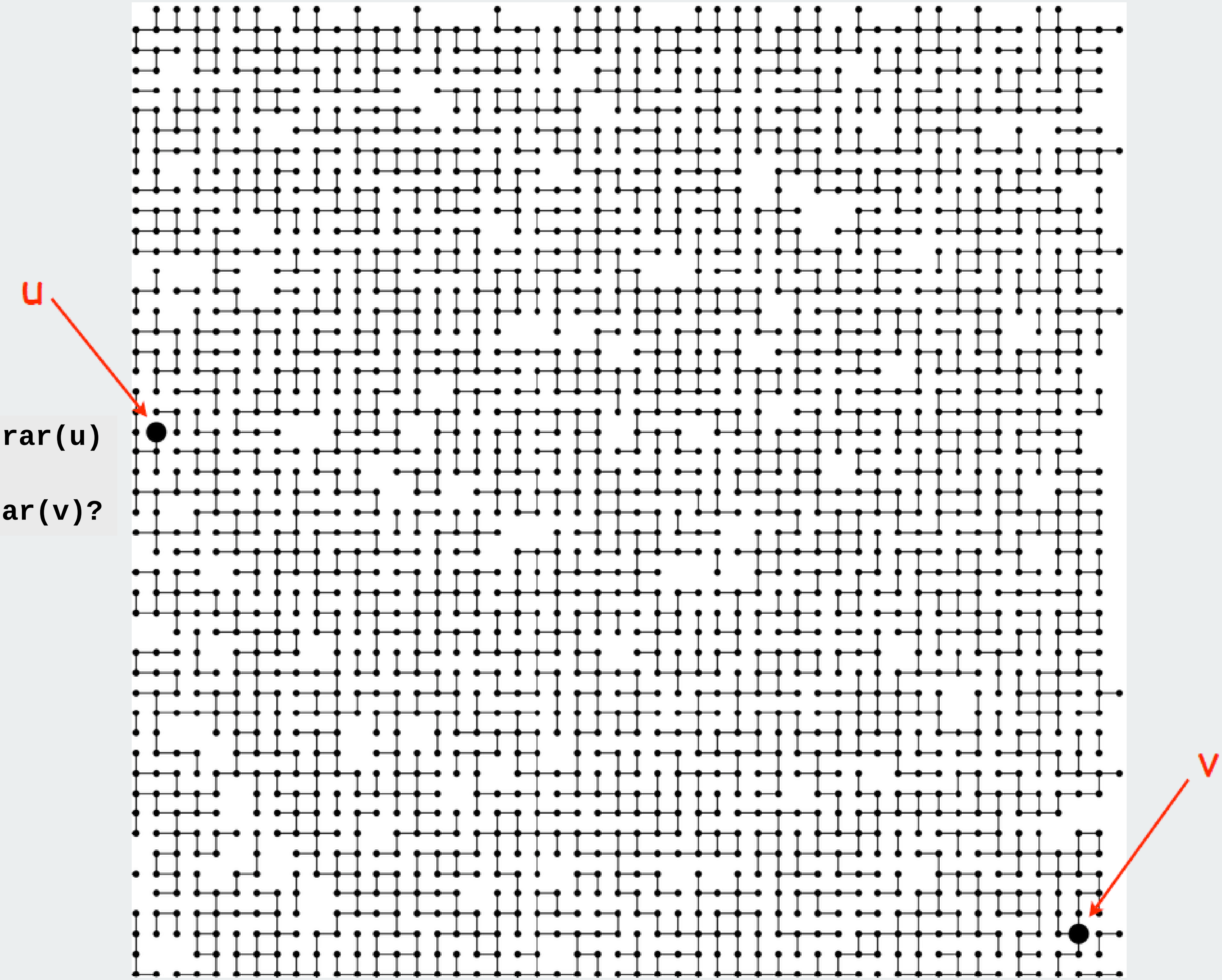
- Definición:
  - Una estructura de conjuntos disjuntos (ECD) (en muchos libros en inglés, "union-find structure") mantiene una colección de conjuntos disjuntos

$$S = \{S_1, S_2, \ldots, S_k\}$$

  - Cada conjunto se identifica por un representante (un miembro del conjunto)
  - Se precisan (como mínimo) las operaciones de:
    - *crear*(x): crea un nuevo conjunto con un solo elemento (x, que no debe pertenecer a ningún otro conjunto)
    - *unir*(x,y): une los conjuntos que contienen a x y a y en un nuevo conjunto, y destruye los viejos conjuntos de x e y
    - *encontrar*(x): devuelve el representante del (único) conjunto que contiene a x

Estructuras de conjuntos disjuntos

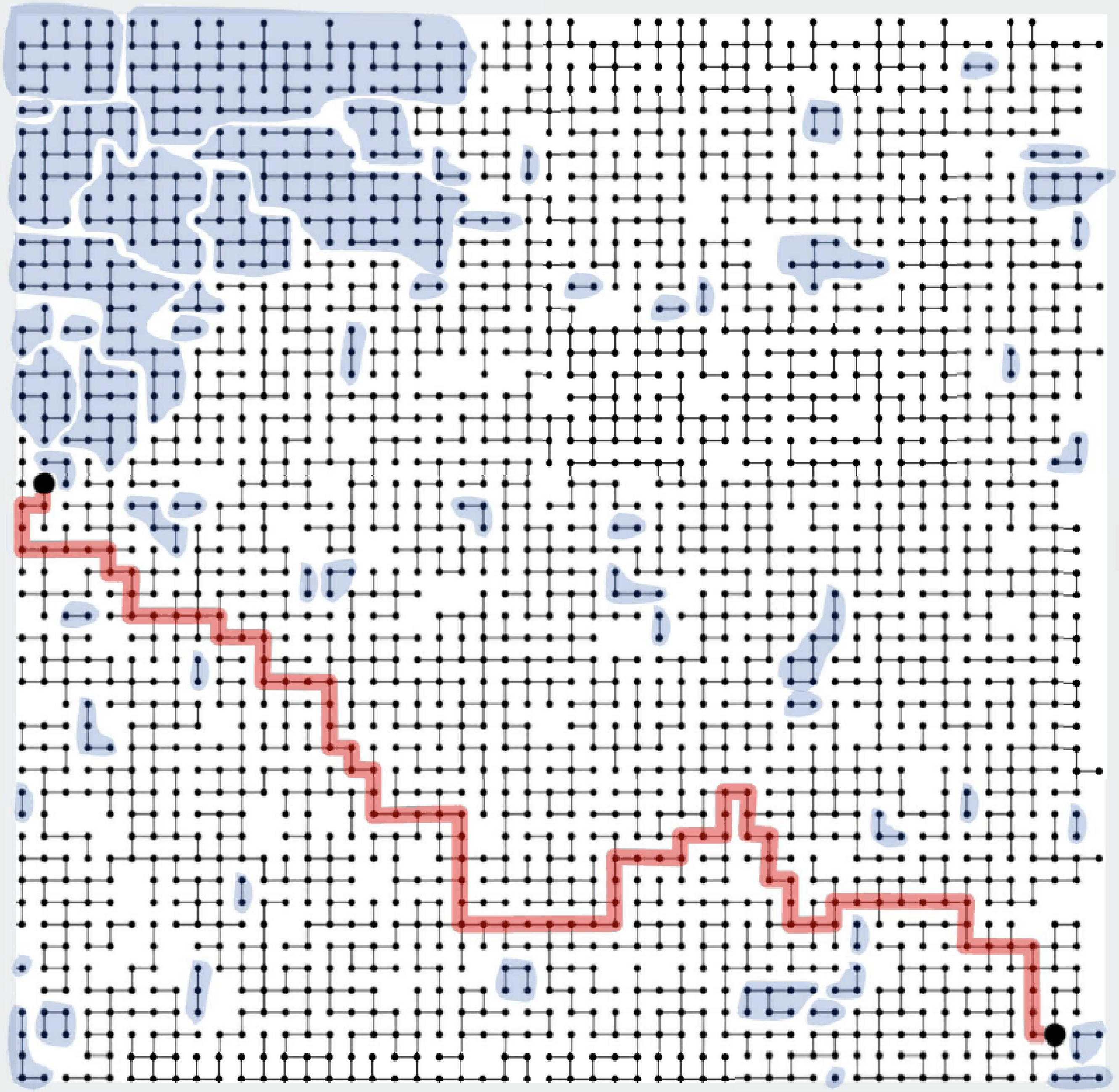
- Conectividad en redes.
- Percolación (flujo de un líquido a través de un medio poroso).
- Procesamiento de imágenes.
- Antecesor común más próximo (en un árbol).
- Equivalencia de autómatas de estados finitos.
- Inferencia de tipos polimórficos o tipado implícito (algoritmo de Hinley-Milner).
- Árbol de recubrimiento mínimo (algoritmo de Kruskal).
- Juego (*Go*, *Hex*).
- Compilación de la instrucción EQUIVALENCE en Fortran.
- Mantenimiento de listas de copias duplicadas de páginas web.
- Diseño de VLSI.



¿encontrar(u) = encontrar(v)?

¿encontrar(u) = encontrar(v)?

verdad



63 componentes

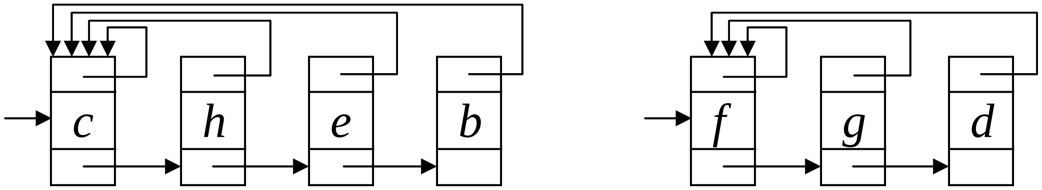
Estructuras de conjuntos disjuntos

- Ejemplo sencillo de aplicación: calcular las componentes conexas de un grafo no dirigido

```
algoritmo componentes_conexas(g)
principio
  para todo v vértice de g hacer
    crear(v)
  fpara;
  para toda (u,v) arista de g hacer
    si encontrar(u)≠encontrar(v) entonces
      unir(u,v)
    fsi
  fpara
fin
```

Estructuras de conjuntos disjuntos

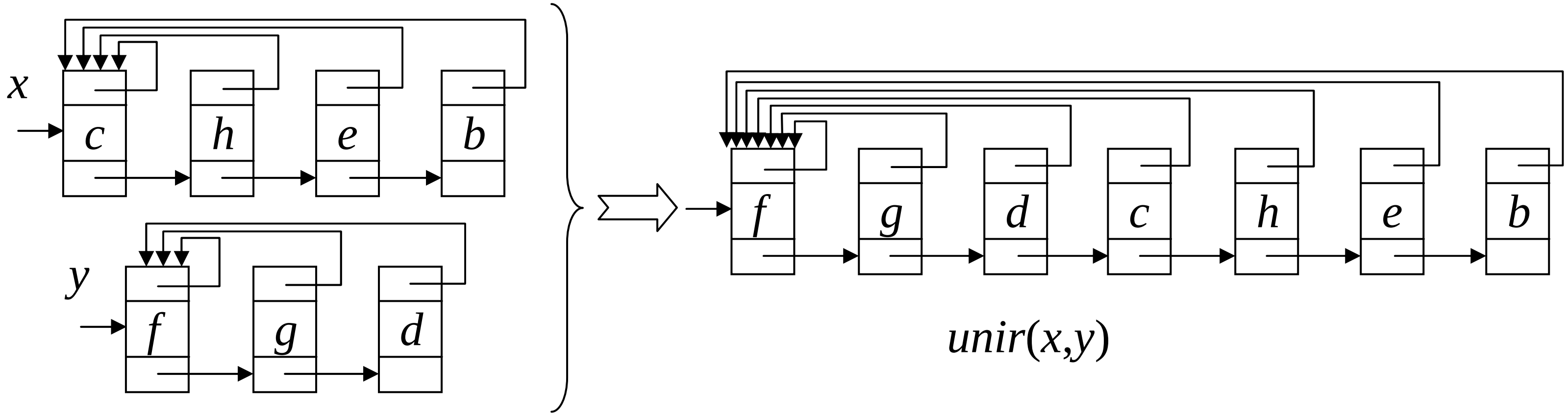
- Primera implementación: listas encadenadas
  - Cada conjunto se almacena en una lista encadenada
    - Cada registro de la lista contiene:
      - Un elemento del conjunto
      - Un puntero al siguiente registro de la lista
      - Un puntero al primer registro de la lista
  - El primer elemento de cada lista es el representante



  - Las implementaciones de *crear* y *encontrar* son triviales y requieren $O(1)$ en tiempo

Estructuras de conjuntos disjuntos

– Implementación de *unir*:



- Concatenar la primera lista tras la segunda y modificar los punteros al primero en la primera lista (coste en tiempo lineal en la longitud de esa lista)
- El coste amortizado **con esta implementación** no puede reducirse (a algo menor que lineal)…

Estructuras de conjuntos disjuntos

- Ejemplo en el que el coste amortizado de *unir* es lineal:

operación	nº de objetos modificados
$crear(x_1)$	1
$crear(x_2)$	1
…	…
$crear(x_n)$	1
$unir(x_1, x_2)$	1
$unir(x_2, x_3)$	2
$unir(x_3, x_4)$	3
…	…
$unir(x_{q-1}, x_q)$	$q-1$

  – Es una secuencia de $m = n + q - 1$ operaciones y requiere un tiempo $\Theta(n + q^2) = \Theta(m^2)$ (porque $n = \Theta(m)$ y $q = \Theta(m)$).

  – Por tanto, cada operación requiere, en media (coste amortizado), $\Theta(m)$.

Estructuras de conjuntos disjuntos

– Mejora de la implementación de *unir*:
  - *Heurística de la unión*: si cada lista almacena explícitamente su longitud, optar por añadir siempre la lista más corta al final de la más larga.
  - Con esta heurística sencilla el coste de una única operación sigue siendo el mismo, pero el coste amortizado se reduce:

    Una secuencia de m operaciones de *crear*, *unir* y *encontrar*, n de las cuales sean de *crear*, o lo que es lo mismo,
    una secuencia de m operaciones con una ECD con n elementos distintos cuesta $O(m + n\log n)$ en tiempo.

Estructuras de conjuntos disjuntos

- Demostración:
  - 1º) Calcular para cada elemento x de un conjunto de tamaño n una cota superior del nº de veces que se cambia su puntero al representante:

    La 1ª vez que se cambia, el conjunto resultante tiene al menos 2 elementos.

    La 2ª vez, tiene el menos 4 elementos (x siempre debe estar en el conjunto más pequeño). Con igual argumento, para todo $k \leq n$, después de cambiar $\lceil \log k \rceil$ veces el puntero al representante de x, el conjunto resultante debe tener como mínimo k elementos.

    Como el conjunto más grande tiene no más de n elementos, el puntero al representante de cada elemento se ha cambiado no más de $\lceil \log n \rceil$ veces en todas las operaciones de *unir*.

    Por tanto, el coste total debido a los cambios del puntero al representante es $O(n \log n)$.
  - 2º) Cada operación *crear* y *encontrar* está en $O(1)$, y hay $O(m)$ de esas operaciones.

    Por tanto, el coste total de toda la secuencia es $O(m + n\log n)$.

Estructuras de conjuntos disjuntos

- Ejercicio:

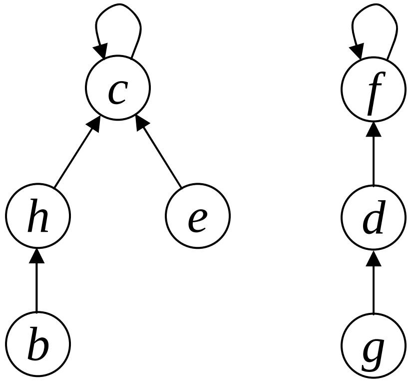
  Teniendo en cuenta que el coste de una secuencia de m operaciones con una ECD con n elementos distintos es $O(m + n\log n)$, se puede demostrar que:

  – El coste amortizado de una operación de *crear* o de *encontrar* es $O(1)$.

  – El coste amortizado de una operación de *unir* es $O(\log n)$.

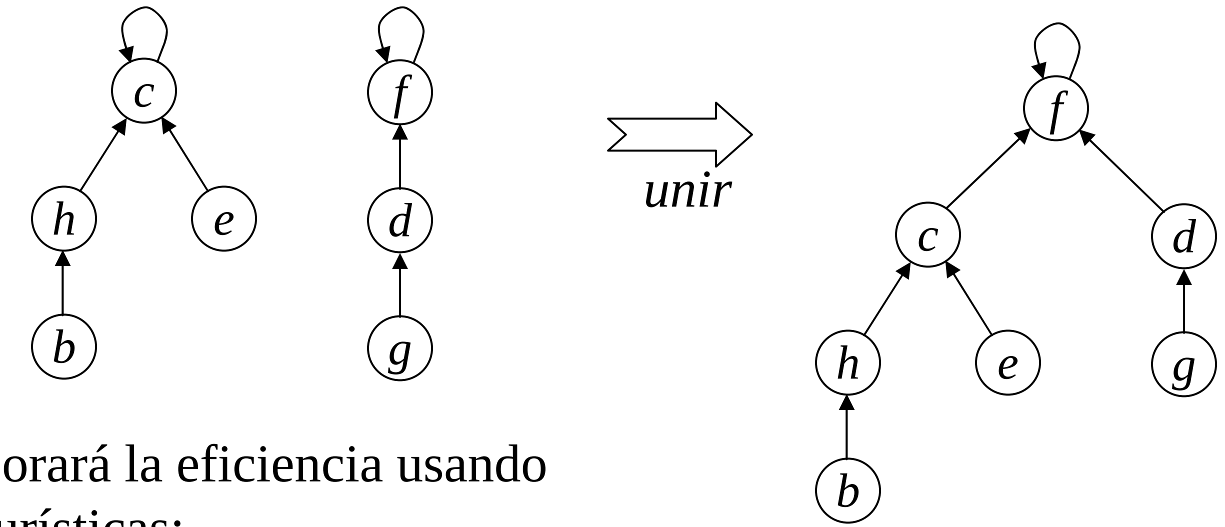
Estructuras de conjuntos disjuntos

- Segunda implementación: bosque
  - Conjunto de árboles, cada uno representando un conjunto disjunto de elementos.
  - Representación con puntero al padre.
  - La raíz de cada árbol es el representante y apunta a si misma.



Estructuras de conjuntos disjuntos

– La implementación “trivial” de las operaciones no mejora la eficiencia de la implementación con listas:
  - *Crear* crea un árbol con un solo nodo.
  - *Encontrar* sigue el puntero al padre hasta llegar a la raíz del árbol (los nodos recorridos se llaman *camino de búsqueda*).
  - *Unir* hace que la raíz de un árbol apunte a la raíz del otro.



– Se mejorará la eficiencia usando dos heurísticas:
  - *Unión por rango*
  - *Compresión de caminos*

Estructuras de conjuntos disjuntos

- Implementación en el bosque de la heurística "unión por rango"
  - Similar a la heurística de la unión usada con listas.
  - Hacer que la raíz que va a apuntar a la otra sea la del árbol con menos nodos.
  - En lugar de mantener el tamaño de los árboles, mantenemos siempre el *rango* de los árboles, que es una cota superior de su altura.
    - Al *crear* un árbol, su rango es 0.
    - Con la operación de *encontrar* el rango no cambia.
    - Al *unir* dos árboles se coloca como raíz la del árbol de mayor rango, y éste no cambia; en caso de empate, se elige uno cualquiera y se incrementa su rango en una unidad.

Estructuras de conjuntos disjuntos

```
algoritmo crear(x)
principio
  nuevóArbol(x);
  x.padre:=x;
  x.rango:=0
fin
```

```
algoritmo unir(x,y)
principio
  enlazar(encontrar(x),encontrar(y))
fin
```

```
algoritmo enlazar(x,y)
principio
  si x.rango>y.rango ent
    y.padre:=x
  sino
    x.padre:=y;
    si x.rango=y.rango ent y.rango:=y.rango+1 fsi
  fsi
fin
```

Estructuras de conjuntos disjuntos

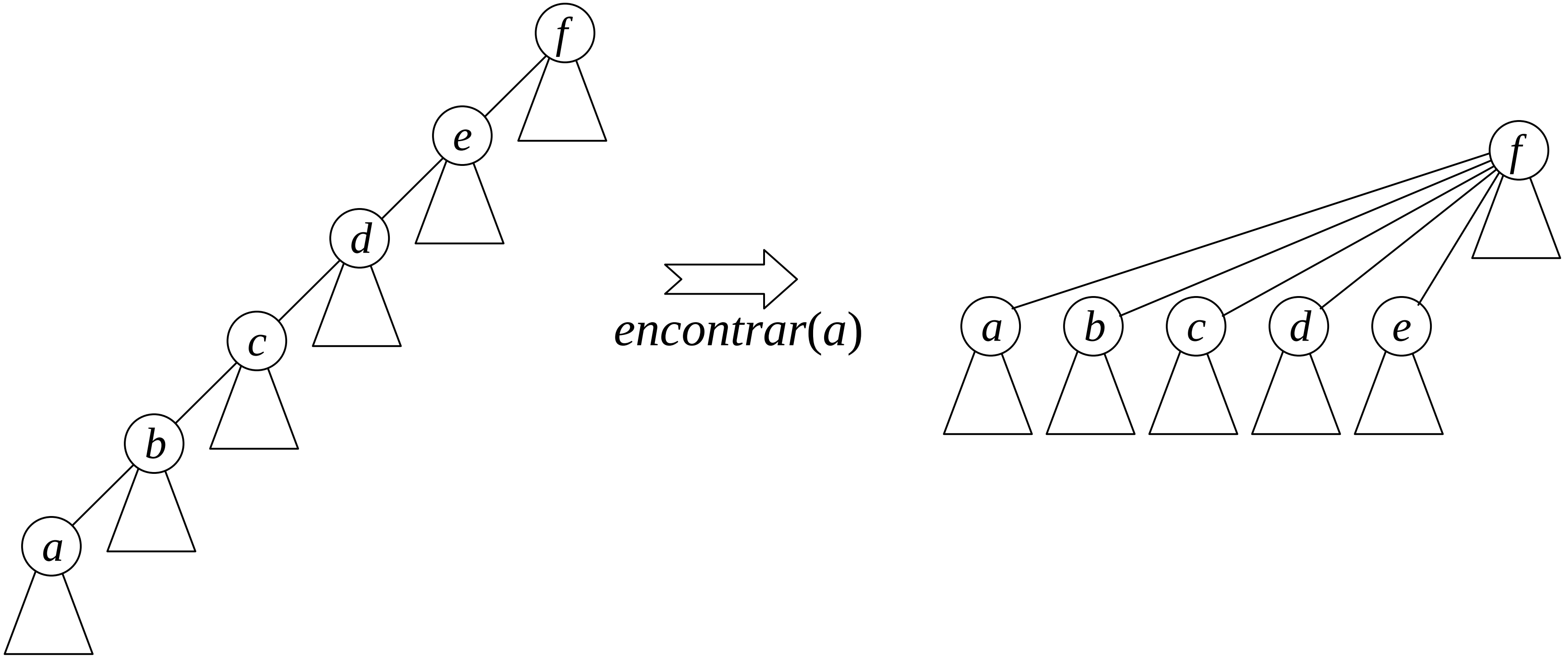
- Implementación en el bosque de la heurística de “compresión de caminos”
  - Muy simple y muy efectiva.
  - Se usa en la operación de *encontrar*.
  - Hace que los nodos recorridos en el *camino de búsqueda* pasen a apuntar directamente a la raíz.
  - No se modifica la información sobre el rango.

```
algoritmo encontrar(x) devuelve puntero a nodo
principio
  si x≠x.padre ent x.padre:=encontrar(x.padre) fsi;
  devuelve x.padre
fin
```

Ojo! Modifica x

Estructuras de conjuntos disjuntos

- Efecto de la compresión de caminos en una operación de *encontrar*



Estructuras de conjuntos disjuntos

- Coste de las operaciones usando las heurísticas:
  - El uso separado de las heurísticas de unión por rango y de compresión de caminos mejora la eficiencia de las operaciones:
    - La unión por rango (sin usar compresión de caminos) da un coste de $O(m \log n)$ para una secuencia de m operaciones con una ECD con n elementos distintos.
    - La compresión de caminos (sin usar unión por rango) da un coste de $\Theta(f \log_{(1+f/n)} n)$ para una secuencia de n operaciones de *crear* (y por tanto hasta un máximo de n–1 de *unir*) y f operaciones de *encontrar*, si $f \geq n$, y $\Theta(n + f \log n)$ si $f < n$.
  - El uso conjunto de las dos heurísticas mejora aún más la eficiencia de las operaciones…

Estructuras de conjuntos disjuntos

- Coste utilizando ambas heurísticas:
  - Si se usan la unión por rango y la compresión de caminos, el coste en el caso peor para una secuencia de m operaciones con una ECD con n elementos distintos es $O(m\ \alpha(m,n))$, donde $\alpha(m,n)$ es una función (parecida a la inversa de la función de Ackerman) que crece **muy** despacio.

Tan despacio que en cualquier aplicación práctica que podamos imaginar se tiene que $\alpha(m,n) \leq 4$, por tanto puede interpretarse el coste como lineal en m, en la práctica.

Estructuras de conjuntos disjuntos

- ¿Cómo es la función $\alpha(m,n)$?
  - Funciones previas:
    - Para todo entero $i \geq 0$, abreviamos $\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}i \text{ veces}$ con $g(i)$:

$$g(i)=\begin{cases}2^1, & \text{si } i=0\\ 2^2, & \text{si } i=1\\ 2^{g(i-1)}, & \text{si } i>1\end{cases}$$

    - La función "logaritmo estrella", $\log^* n$, es la inversa de g:

$$\log^* n = \text{mín}\{i \geq 0 \mid \log^{(i)} n \leq 1\}$$

$$\log^{(i)} n = \begin{cases} n, & \text{si } i=0\\ \log\left(\log^{(i-1)} n\right), & \text{si } i>0 \text{ y } \log^{(i-1)} n > 0\\ \text{no definido}, & \text{si } i>0 \text{ y } \log^{(i-1)} n \leq 0 \text{ ó}\\ & \log^{(i-1)} n \text{ no está definido}\end{cases}$$

Es decir, $\log^* g(n) = \log^* \left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}n \text{ veces} = n+1$.

Estructuras de conjuntos disjuntos

- La función de Ackerman, definida para enteros i, $j \geq 1$:

$$A(1,j) = 2^j, \text{ para } j \geq 1$$

$$A(i,1) = A(i-1,2), \text{ para } i \geq 2$$

$$A(i,j) = A(i-1,A(i,j-1)), \text{ para } i, j \geq 2$$

	$j = 1$	$j = 2$	$j = 3$	$j = 4$
$i = 1$	2^1	2^2	2^3	2^4
$i = 2$	2^2	2^{2^2}	$2^{2^{2^2}}$	$2^{2^{2^{2^2}}}$
$i = 3$	2^{2^2}	$\left.2^{2^{\cdot^{\cdot^{2}}}}\right\}16$	$\left.2^{2^{\cdot^{\cdot^{2}}}}\right\}\left.2^{2^{\cdot^{\cdot^{2}}}}\right\}16$	$\left.2^{2^{\cdot^{\cdot^{2}}}}\right\}\left.2^{2^{\cdot^{\cdot^{2}}}}\right\}\left.2^{2^{\cdot^{\cdot^{2}}}}\right\}16$

Estructuras de conjuntos disjuntos

– Por fin, la función $\alpha(m,n)$:

- Es “una especie de inversa” de la función de Ackerman (no en sentido matemático estricto, sino en cuanto a que crece tan despacio como deprisa lo hace la de Ackerman).
- $\alpha(m,n) = \text{mín}\{i \geq 1 \mid A(i,\lfloor m/n \rfloor) > \log n\}$
- ¿Por qué podemos suponer que siempre (en la práctica) $\alpha(m,n) \leq 4$?
  - Nótese que $\lfloor m/n \rfloor \geq 1$, porque $m \geq n$.
  - La función de Ackerman es estrictamente creciente con los dos argumentos, luego $\lfloor m/n \rfloor \geq 1 \Rightarrow A(i,\lfloor m/n \rfloor) \geq A(i,1)$, para $i \geq 1$.
  - En particular, $A(4,\lfloor m/n \rfloor) \geq A(4,1) = A(3,2) = \left. 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \right\} 16$
  - Luego sólo ocurre para n’s “enormes” que $A(4,1) \leq \log n$, y por tanto $\alpha(m,n) \leq 4$ para todos los valores “razonables” de m y n.

Estructuras de conjuntos disjuntos

– La demostración(*) [Tar83, pp.23-31] no es fácil… veremos una cota **ligeramente** peor: $O(m \log^* n)$.

- El "logaritmo estrella" crece **casi tan despacio** como la función $\alpha(m,n)$:

$$\log^* 2 = 1$$

$$\log^* 4 = \log^* 2^2 = 2$$

$$\log^* 16 = \log^* 2^{2^2} = 3$$

$$\log^* 65536 = \log^* 2^{2^{2^2}} = 4$$

$$\log^* (2^{65536}) = \log^* 2^{2^{2^{2^2}}} = 5$$

el número estimado de átomos en el Universo observable es de unos 10^{80}, que es **mucho** menor que 2^{65536} ($\approx 2*10^{19728}$), así que… ¡¡¡ difícilmente nos encontraremos (en este Universo) un n tal que $\log^* n > 5$!!!

(*) de que el coste en el caso peor para una secuencia de m operaciones con una ECD con n elementos distintos es $O(m\ \alpha(m,n))$

- Algunas propiedades del rango de los árboles.
  - $[P_1]$ Para todo nodo x, $rango(x) \leq rango(padre(x))$, y la desigualdad es estricta si $x \neq padre(x)$.
  - $[P_2]$ El valor de $rango(x)$ es inicialmente 0 y crece con el tiempo hasta que $x \neq padre(x)$, desde entonces el valor de $rango(x)$ ya no cambia más.
  - $[P_3]$ El valor de $rango(padre(x))$ es una función monótona creciente con respecto al tiempo.

Todas estas propiedades se deducen fácilmente de la implementación.

Estructuras de conjuntos disjuntos

[P_4] – Si se define el *cardinal*(x) como el nº de nodos del árbol con raíz x (incluido x), entonces

$$cardinal(x) \geq 2^{rango(x)}$$

Demostración: por inducción en el número de operaciones de *enlazar* (la operación *encontrar* no modifica el cardinal ni el rango de la raíz del árbol).

- Base de inducción:

  Antes de la primera operación de *enlazar* los rangos son 0 y cada árbol contiene al menos un nodo, luego

$$cardinal(x) \geq 1 = 2^0 = 2^{rango(x)}$$

Estructuras de conjuntos disjuntos

- Paso de inducción:

  Suponer que es cierto antes de *enlazar* x e y que
  $cardinal(x) \geq 2^{rango(x)}$ y $cardinal(y) \geq 2^{rango(y)}$

  – si $rango(x) \neq rango(y)$, asumir que $rango(x) < rango(y)$, entonces y es la raíz del árbol resultante y se tiene que:

  $$cardinal'(y) = cardinal(x) + cardinal(y) \geq$$
  $$\geq 2^{rango(x)} + 2^{rango(y)} \geq 2^{rango(y)} = 2^{rango'(y)},$$

  y no cambian más rangos ni cardinales

  – si $rango(x) = rango(y)$, el nodo y es de nuevo la raíz del nuevo árbol y se tiene que:

  $$cardinal'(y) = cardinal(x) + cardinal(y) \geq$$
  $$\geq 2^{rango(x)} + 2^{rango(y)} = 2^{rango(y)+1} = 2^{rango'(y)}$$

$[P_5]$ – Para todo entero $r \geq 0$, hay como máximo $n/2^r$ nodos de rango r.

Demostración:

- suponer que cuando se asigna un rango r a un nodo x (en los algoritmos *crear* o *enlazar*) se añade una etiqueta x a cada nodo del subárbol de raíz x
- por la propiedad $[P_4]$, se etiquetan como mínimo 2^r nodos cada vez
- si cambia la raíz del árbol que contiene el nodo x, por la propiedad $[P_1]$, el rango del nuevo árbol será, como mínimo, $r + 1$
- como asignamos etiquetas sólo cuando una raíz recibe el rango r, ningún nodo del nuevo árbol se etiquetará más veces, es decir, cada nodo se etiqueta como mucho una sola vez, cuando su raíz recibe por vez primera el rango r

Estructuras de conjuntos disjuntos

- como hay n nodos, hay como mucho n nodos etiquetados, con como mínimo 2^r etiquetas asignadas para cada nodo de rango r
- si hubiese más de $n/2^r$ nodos de rango r, habría más de $2^r(n/2^r) = n$ nodos etiquetados por un nodo de rango r, lo cual es contradictorio
- por tanto, hay como mucho $n/2^r$ nodos a los que se asigna un rango r

– Corolario: el rango de todo nodo es como máximo $\lfloor \log n \rfloor$

Demostración:

- si $r > \log n$, hay como mucho $n/2^r < 1$ nodos de rango r
- como el rango es un natural, se sigue el corolario

Estructuras de conjuntos disjuntos

- Demostraremos ahora con el método agregado que el coste amortizado para una secuencia de m operaciones con una ECD con n elementos distintos es $O(m \log^* n)$.
  - Primero veamos que se pueden considerar m operaciones de *crear*, ***enlazar*** y *encontrar* en lugar de m de *crear*, ***unir*** y *encontrar*:

    S' = sec. de m oper. de *crear*, *unir* y *encontrar*

    S = sec. obtenida de S' sustituyendo cada oper. de *unión* por dos de *encontrar* y una de *enlazar*

    Si S está en $O(m \log^* n)$, entonces S' también esté en $O(m \log^* n)$.

    Demostración:

    - cada *unión* en S' se convierte en 3 operaciones en S, luego $m' \leq m \leq 3m'$
    - como $m = O(m')$, una cota $O(m \log^* n)$ para la secuencia S implica una cota $O(m' \log^* n)$ para la secuencia original S'.

Estructuras de conjuntos disjuntos

– Veamos ahora el coste amortizado:
  - El coste de cada operación de *crear* y de *enlazar* es claramente $O(1)$.

```
algoritmo crear(x)
principio
  nuevóArbol(x);
  x.padre:=x;
  x.rango:=0
fin
```

```
algoritmo enlazar(x,y)
principio
  si x.rango>y.rango ent
    y.padre:=x
  sino
    x.padre:=y;
    si x.rango=y.rango ent y.rango:=y.rango+1 fsi
  fsi
fin
```

Estructuras de conjuntos disjuntos

- Veamos la operación *encontrar*, antes… algo de notación:
  - Agrupamos los rangos de los nodos en *bloques*:
    el rango r está en el bloque que numeramos con $\log^* r$, para $r = 0,1,\ldots,\lfloor \log n \rfloor$ (recordar que $\lfloor \log n \rfloor$ es el máximo rango).
  - El bloque con mayor número es por tanto el numerado con $\log^* (\log n) = \log^* n - 1$.
  - Notación: para todo entero $j \geq -1$,

$$B(j) = \begin{cases} -1, & \text{si } j = -1 \\ 1, & \text{si } j = 0 \\ 2, & \text{si } j = 1 \\ \left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\} j\text{-}1, & \text{si } j \geq 2 \end{cases}$$

  - Entonces, para $j = 0,1,\ldots,\log^* n - 1$, el bloque j-ésimo consiste en el conjunto de rangos $\{B(j-1)+1, B(j-1)+2, \ldots, B(j)\}$.

Bloque 0:
{0,1}
Bloque 1:
{2}
Bloque 2:
{3,4}
Bloque 3:
{5,…,16}
Bloque 4:
{17,…,65536}
Bloque 5:
{65537,..,BIG}

log*2=1
log*4=2
log*16=3
log*65536=4
log*(2^{65536})=5

Estructuras de conjuntos disjuntos

- Definimos dos tipos de "unidades de coste" asociadas a la operación *encontrar*: unidad de coste de bloque y unidad de coste de camino.
- Supongamos que *encontrar* empieza en el nodo x_0 y que el camino de búsqueda es $x_0, x_1, \ldots, x_l$, con $x_i = padre(x_{i-1})$, $i = 1, 2, \ldots, l$, y $x_l = padre(x_l)$, es decir x_l es una raíz.
- Asignamos una unidad de coste de bloque a:
  - el último(*) nodo del camino de búsqueda cuyo rango está en el bloque j, para todo $j = 0,1,\ldots,\log^* n - 1$; es decir, a todo x_i tal que $\log^* rango(x_i) < \log^* rango(x_{i+1})$; y a
  - el hijo de la raíz, es decir, $x_i \neq x_l$ tal que $padre(x_i) = x_l$.
- Asignamos una unidad de coste de camino a:
  - cada nodo del camino de búsqueda al que no le asignamos unidad de coste de bloque.

(*) Nótese que por la propiedad P_1, los nodos con rango en un bloque dado aparecen consecutivos en el camino de búsqueda.

Estructuras de conjuntos disjuntos

- Una vez que un nodo (distinto de la raíz o sus hijos) ha recibido en alguna operación de *encontrar* una unidad de coste de bloque, ya nunca (en sucesivas operaciones de *encontrar*) recibirá unidades de coste de camino.

  Demostración:

  – en cada compresión de caminos que afecta a un nodo x_i tal que $x_i \neq padre(x_l)$, el rango de ese nodo se mantiene, pero el nuevo padre de x_i tiene un rango estrictamente mayor que el que tenía el anterior padre de x_i;

  – por tanto, la diferencia entre los rangos de x_i y su padre es una función monótona creciente con respecto al tiempo;

  – por tanto, también lo es la diferencia entre $\log^* rango(padre(x_i))$ y $\log^* rango(x_i)$;

  – luego, una vez que x_i y su padre tienen rangos en distintos bloques, siempre tendrán rangos en distintos bloques, y por tanto nunca más se le volverá a asignar a x_i una unidad de coste de camino.

Estructuras de conjuntos disjuntos

- Como hemos asignado una unidad de coste para cada nodo visitado en cada operación de *encontrar*, el nº total de unidades de coste asignadas coincide con el nº total de nodos visitados; queremos demostrar que ese total es $O(m \log^* n)$.
- El nº total de unidades de coste de bloque es fácil de acotar:

  en cada ejecución de *encontrar* se asigna una unidad por cada bloque, más otra al hijo de la raíz;

  como los bloques varían en $0,1,\ldots, \log^* n - 1$, entonces hay como mucho $\log^* n + 1$ unidades de coste de bloque asignadas en cada operación de *encontrar*, y por tanto no más de $m(\log^* n + 1)$ en toda la secuencia de m operaciones.

- Falta acotar el nº total de unidades de coste de camino…

Estructuras de conjuntos disjuntos

- Acotación del nº total de unidades de coste de camino:
  - Observar que si a un nodo x_i se le asigna una unidad de coste de camino entonces $padre(x_i) \neq x_l$ antes de la compresión, luego a x_i se le asignará un nuevo padre en la compresión, y el nuevo padre tendrá un rango mayor que el anterior padre.
  - Supongamos que el rango de x_i está en el bloque j.
  - ¿Cuántas veces se le puede asignar a x_i un nuevo padre (y por tanto asignarle una unidad de coste de camino), antes de asignarle un padre cuyo rango esté en un bloque distinto (después de lo cual ya nunca se le asignarán más unidades de coste de camino)?
    - » Ese nº es máximo si x_i tiene el rango más pequeño de su bloque, $B(j–1)+1$, y el rango de sus padres va tomando los valores $B(j–1)+2$, $B(j–1)+3$, …, $B(j)$.
    - » Es decir, puede ocurrir $B(j) – B(j–1) – 1$ veces.

Estructuras de conjuntos disjuntos

– Siguiente paso: acotar el nº de nodos cuyo rango está en el bloque j, para cada $j \geq 0$ (recordar, propiedad P_2, que el rango de un nodo es fijo una vez que pasa a ser hijo de otro nodo).

» $N(j)$ = nº nodos cuyo rango es del bloque j.

» Por la propiedad P_5:

$$N(j) \leq \sum_{r=B(j-1)+1}^{B(j)} \frac{n}{2^r}$$

» Para $j = 0$: $N(0) = n/2^0 + n/2^1 = 3n/2 = 3n/2B(0)$

» Para $j \geq 1$:

$$N(j) \leq \frac{n}{2^{B(j-1)+1}} \sum_{r=0}^{B(j)-(B(j-1)+1)} \frac{1}{2^r}$$

$$< \frac{n}{2^{B(j-1)+1}} \sum_{r=0}^{\infty} \frac{1}{2^r}$$

$$= \frac{n}{2^{B(j-1)}} = \frac{n}{B(j)}$$

» Por tanto: $N(j) \leq 3n/2B(j)$, para todo entero $j \geq 0$.

Estructuras de conjuntos disjuntos

– Sumando para todos los bloques el producto del máximo nº de nodos con rango en ese bloque por el máximo nº de unidades de coste de camino por nodo de ese bloque, se tiene:

$$P(n) \le \sum_{j=0}^{\log^* n-1} \frac{3n}{2B(j)}\left(B(j) - B(j-1) - 1\right)$$

$$\le \sum_{j=0}^{\log^* n-1} \frac{3n}{2B(j)} B(j) = \frac{3}{2} n \log^* n$$

- Por tanto, el nº máximo de unidades de coste por todas las operaciones de *encontrar* es $O(m(\log^* n + 1) + n \log^* n)$, es decir, $O(m \log^* n)$, porque $m \ge n$.
- Como el número total de operaciones de *crear* y de *enlazar* es $O(n)$, el coste total de las m operaciones con una ECD con n elementos distintos es $O(m \log^* n)$.