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")

Montículos de Fibonacci

- ¿Qué se pretende?

operación	mont. binario (caso peor)	mont. binomial (caso peor)	mont. Fibonacci (**amortizado**)
crear vacío	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
insertar	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
mínimo	$\Theta(1)$	$\Theta(\log n)$	$\Theta(1)$
borrar mínimo	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$
unión	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$
reducir clave	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
borrar	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$

Montículos de Fibonacci

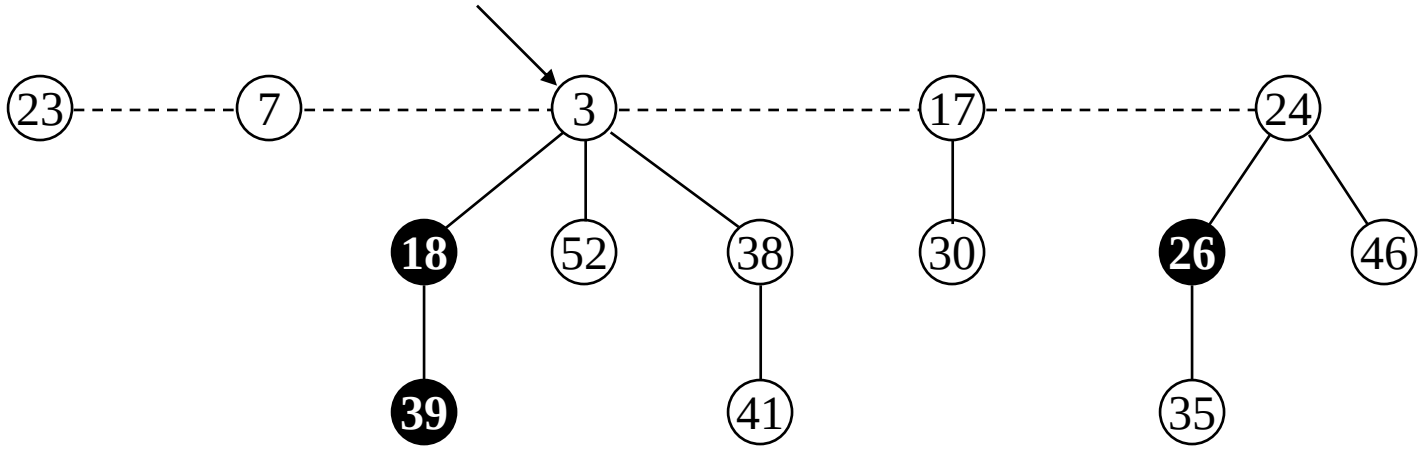
- Situaciones de interés:
  - Problemas en los que las operaciones de borrado del mínimo y de borrado de cualquier elemento son poco frecuentes en proporción con el resto.
  - Ejemplo: muchos algoritmos de grafos en los que se precisan colas con prioridades y con la ejecución frecuente de la operación de reducción de clave.
    - Algoritmo de Dijkstra para el cálculo de caminos mínimos (ver transparencias de *Esquemas Algorítmicos: Algoritmos voraces*, pp. 27-37).
    - Algoritmo de Prim para el cálculo de árboles de recubrimiento de coste mínimo (ver transparencias de *Esquemas Algorítmicos: Algoritmos voraces*, pp. 38-46).

Montículos de Fibonacci

- Desde un punto de vista práctico:
  - Las constantes multiplicativas en el coste y la complejidad de su programación los hacen menos aconsejables que los montículos “ordinarios” en muchas aplicaciones.
  - Por tanto, salvo que se manejen MUCHAS claves, tienen un interés eminentemente teórico.
- ¿Qué son?
  - Podría decirse que son una *versión perezosa* de los montículos binomiales.
    - Si no se ejecutan operaciones de borrado ni de reducción de claves en un montículo de Fibonacci, entonces cada uno de sus árboles es un árbol binomial.
    - Tienen una estructura “más relajada”, permitiendo retrasar la reorganización de la estructura hasta el momento más conveniente para reducir el coste amortizado.

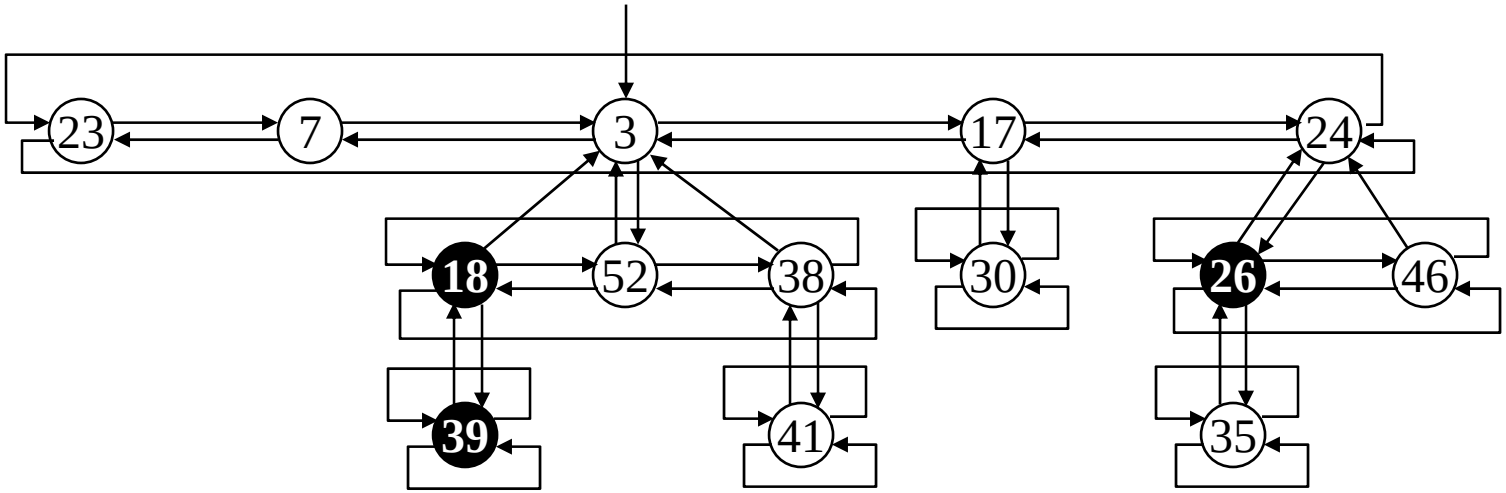
Montículos de Fibonacci

- Estructura de un montículo de Fibonacci:
  - Es un conjunto de *árboles parcialmente ordenados*, es decir, la clave de todo nodo es mayor o igual que la de su padre.
  - Los árboles no precisan ser binomiales.
  - Los árboles no están ordenados.
  - Se accede por un puntero a la raíz de clave mínima.



Montículos de Fibonacci

- Representación en memoria:



  - Cada nodo tiene un puntero al padre y un puntero a alguno de sus hijos.
  - Los hijos de un nodo se enlazan con una lista circular doblemente enlazada (cada nodo tiene un puntero a su hermano izquierdo y al derecho). El orden en esa lista es arbitrario.
    - Ventaja: se puede eliminar un elemento en $O(1)$ y se pueden juntar dos listas en $O(1)$.

Montículos de Fibonacci

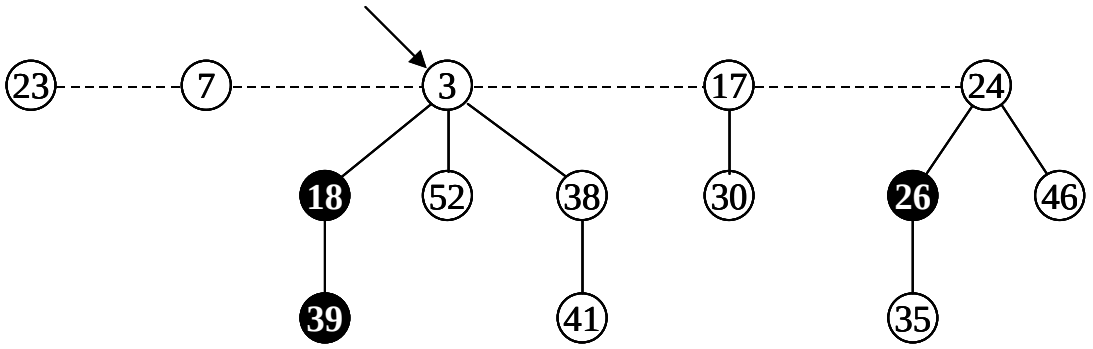
- Representación en memoria (continuación):
  - Además, cada nodo *x* contiene:
    - El número de hijos de *x*: llamado *grado*.
    - Un booleano, *marca*, que indica si el nodo *x* ha perdido un hijo desde la última vez que *x* se puso como hijo de otro nodo.
      - Los nodos recién creados tienen la marca a falso.
      - También se pone la marca a falso cuando el nodo se convierte en hijo de otro nodo.
      - Veremos para qué sirve la marca más adelante…
  - Al montículo se accede mediante un puntero que apunta al elemento mínimo de la lista circular doblemente encadenada de raíces de árboles.
  - El orden de los árboles en la lista de raíces es arbitrario.

Montículos de Fibonacci

- Definición de la función de potencial:
  - Notación: dado un montículo de Fibonacci, M, sea $a(M)$ el número de árboles en la lista de raíces y $m(M)$ el número de nodos marcados (i.e., con el valor de la marca igual a verdad).
  - Definimos el potencial de M como:

$$P(M) = a(M) + 2m(M)$$

- Ejemplo: el de la figura tiene potencial 5 + 2*3 = 11



Montículos de Fibonacci

- Hipótesis de trabajo:
  - Supondremos que el grado máximo de cualquier nodo en un montículo de Fibonacci de *n* nodos está acotado superiormente por $D(n)$.
    - Luego veremos que $D(n) = O(\log n)$.

- Operaciones de montículo agregable:
  - Crear, insertar, ver el mínimo, borrar el mínimo y unión.
  - Con estas operaciones un montículo de Fibonacci será un conjunto de *árboles binomiales desordenados* (a diferencia del montículo binomial).

Montículos de Fibonacci

- Definición: árbol binomial desordenado, U_k:
  - U_0 es un solo nodo
  - U_k consiste en dos árboles binomiales desordenados U_{k-1} *enlazados* de la siguiente forma: la raíz de uno es *cualquier hijo* de la raíz del otro.
- Las propiedades de los árboles binomiales se mantienen con la siguiente variación de la cuarta propiedad:
  - la raíz tiene *grado k* y es el nodo de máximo grado; más aún, los hijos de la raíz son raíces de subárboles U_0, U_1, …, U_{k-1} en algún orden.
- Por tanto: si un montículo de Fibonacci de n nodos es un conjunto de árboles binomiales desordenados, entonces $D(n) = \log n$.

Montículos de Fibonacci

- Creación de un montículo de Fibonacci vacío:
  - Consta de dos campos,
    - el puntero a la raíz mínima: M.min:=nil, y
    - el número de nodos: M.n:=0.
  - Como $a(M) = m(M) = 0$, el potencial es también 0.
  - El coste amortizado de la operación es igual a su coste real, $O(1)$.

Montículos de Fibonacci

- Operación de inserción de un nuevo nodo:

```
algoritmo insertar(M,x)
principio
  x↑.padre:=nil;
  x↑.hijo:=nil;
  x↑.izq:=nil;
  x↑.dch:=nil;
  x↑.grado:=0;
  x↑.marca:=falso;
  añadir x a la lista de raíces de M;
  si M.min=nil or x↑.clave<M.min↑.clave ent
    M.min:=x
  fsi;
  M.n:=M.n+1
fin
```

  - El coste real de la operación es $O(1)$.

Montículos de Fibonacci

- Nótese que si se insertan *k* nodos consecutivos, la lista de raíces se incrementa con *k* árboles de un solo nodo (a diferencia de los montículos binomiales, en los que se hace un esfuerzo de *compactación*).
- Coste amortizado:
  - Si *M* es el montículo antes de la inserción y *M'* es el resultante tras la inserción, $a(M') = a(M) + 1$ y $m(M') = m(M)$ por lo que el incremento del potencial es 1.
  - Por tanto, el coste amortizado de la operación es:

    $A(M') = C(M') + P(M') - P(M) = O(1) + 1 = O(1)$.

Montículos de Fibonacci

- Operación de busqueda del elemento de clave mínima:
  - El nodo de clave mínima es precisamente el apuntado por *M*.min, por tanto el coste real de la operación es *O*(1).
  - Como el potencial no cambia, el coste amortizado es también *O*(1).

Montículos de Fibonacci

- Operación de unión de montículos de Fibonacci:

```
algoritmo unión(M1,M2,M)
principio
  crearVacío(M);
  M.min:=M1.min;
  concatenar la lista de raíces de M2 con la de M;
  si M.min=nil or (M2.min≠nil and M2.min<M1.min) ent
    M.min:=M2.min
  fsi;
  M.n:=M1.n+M2.n
fin
```

  - Tampoco hay *compactación*.
  - Coste real: $O(1)$.

Montículos de Fibonacci

- Cambio del potencial con la operación de unión:

$$P(M) - (P(M_1) + P(M_2))$$
$$= (a(M) + 2m(M)) - ((a(M_1) + 2m(M_1)) + (a(M_2) + 2m(M_2)))$$
$$= 0$$

porque $a(M) = a(M_1) + a(M_2)$ y $m(M) = m(M_1) + m(M_2)$

- Coste amortizado:

$$A(M) = C(M) + P(M) - (P(M_1) + P(M_2)) = O(1)$$

Montículos de Fibonacci

- Operación de borrado del elemento con clave mínima (es en la que se *compactan* los árboles):

```
algoritmo borrarMínimo(M)
principio
  z:=M.min;
  si z≠nil ent
    para todo x hijo de z hacer
      añadir x a la lista de raíces de M;
      x↑.padre:=nil;
    fpara;
    borrar z de la lista de raíces de M;
    si z=z↑.dch ent M.min:=nil
    sino
      M.min:=z↑.dch; compactar(M)
    fsi;
    M.n:=M.n-1
  fsi
fin
```

Montículos de Fibonacci

- El algoritmo “compactar” debe juntar las raíces de igual grado hasta conseguir que haya como mucho una raíz de cada grado (así se reduce el número de árboles en el montículo), y ordena las raíces por grado.
- Para compactar, repetimos los siguientes pasos hasta que todas las raíces de la lista de raíces del montículo tengan distinto grado:
  - Buscar dos raíces *x* e *y* con igual grado y con la clave de *x* menor o igual que la clave de *y*.
  - Enlazar *y* a *x*:
    - borrar *y* de la lista de raíces y hacer que *y* sea hijo de *x* (algoritmo “enlazar”);
    - se incrementa el grado de *x* y la *marca* de *y* se pone a falso.

Montículos de Fibonacci

– Veamos primero el algoritmo “enlazar”:

```
algoritmo enlazar(M,y,x)
{borrar y de la lista de raíces
 y hacer que sea hijo de x}
principio
  borrar y de la lista de raíces de M;
  poner y como hijo de x;
  x↑.grado:=x↑.grado+1;
  y↑.marca:=falso
fin
```

Montículos de Fibonacci

– Y ahora el algoritmo “compactar”:

• Se utiliza un vector auxiliar $A[0..D(n)]$, donde n es el número de datos del montículo; $A[i] = y$ significa que y es una raíz con grado i.

```
algoritmo compactar(M)
principio
  para i:=0 hasta D(M.n) hacer
    A[i]:=nil
  fpara;
  para todo w en la lista de raíces de M hacer
    x:=w;
    d:=x↑.grado;
    ...
    A[d]:=x;
  fpara;
  ...
```

inicializar *A*

se “procesa” toda raíz *w* (copiada en *x*); al final, $A[x\uparrow.\text{grado}] = x$

Montículos de Fibonacci

- Veamos el procesamiento de cada raíz *w*:

```
...
para todo w en la lista de raíces de M hacer
  x:=w; d:=x↑.grado;
  mq A[d]≠nil hacer
    y:=A[d];
    si x↑.clave>y↑.clave ent
      intercambiar(x,y)
    fsi;
    enlazar(M,y,x);
    A[d]:=nil;
    d:=d+1
  fmq;
  A[d]:=x;
fpara;
...
```

inicialmente era nil, si no lo es hay otras raíces de grado *d*

hay que enlazar *x* e *y*

el nodo *y* ya no es una raíz

el nodo *x* tiene un hijo más (*y*)

ahora seguro que no hay otra raíz con igual grado que *x*

Montículos de Fibonacci

- Y ya sólo queda reconstruir la lista de raíces a partir de la información del vector *A*:

```
  ...
  M.min:=nil;
  para i:=0 hasta D(M.n) hacer
    si A[i]≠nil ent
      añadir A[i] a la lista de raíces de M;
      si M.min=nil
         or A[i]↑.clave<M.min↑.clave ent
        M.min:=A[i]
      fsi
    fsi
  fpara
fin
```

Montículos de Fibonacci

- Hay que verificar que la operación de borrado mantiene los árboles del montículo en la clase de los *binomiales desordenados*:
  - En la operación “borrarMínimo” todo hijo *x* de *z* se convierte en raíz de un nuevo árbol, pero esos hijos son a su vez árboles binomiales desordenados (por la propiedad cuarta).
  - En la operación de compactación se enlazan parejas de árboles binomiales desordenados de igual grado, el resultado es un nuevo árbol binomial desordenado de grado una unidad mayor.

Montículos de Fibonacci

- Coste amortizado del borrado del elemento mínimo
  - Primero hay que calcular el coste real:

    Algoritmo “borrarMínimo”: $O(D(n))$

    Algoritmo “compactar”:
    - los bucles primero y tercero tienen un coste $O(D(n))$
    - el bucle intermedio:
      - » Antes de ejecutar “compactar” el tamaño de la lista de raíces es como mucho $D(n) + a(M) – 1$, es decir, el tamaño antes de empezar la operación, $a(M)$, menos el nodo raíz extraído, más los hijos del nodo extraído, que son como mucho $D(n)$.
      - » En cada ejecución del “mq” interno una raíz se enlaza con otra, por tanto el coste total es como mucho proporcional a $D(n) + a(M)$.

    Luego el coste real es $O(D(n) + a(M))$

Montículos de Fibonacci

- Coste amortizado del borrado del mínimo (cont.)
  - Después hay que calcular el potencial
    - Antes de extraer el mínimo el potencial es $a(M) + 2m(M)$.
    - Después, es como mucho: $(D(n) + 1) + 2m(M)$, porque quedan como mucho $D(n) + 1$ raíces y no se marcan nodos durante la operación.
  - El coste amortizado es, por tanto:

$$A(M') = C(M') + P(M') - P(M)$$
$$= O(D(n) + a(M)) + ((D(n) + 1) + 2m(M) - a(M) + 2m(M))$$
$$= O(D(n))$$

  - La demostración de que $D(n) = O(\log n)$ vendrá más tarde…

Montículos de Fibonacci

- Operación de reducción del valor de una clave:
  - Vemos antes un subalgoritmo auxiliar:

    - Corta el enlace entre un nodo y su padre, convirtiendo al hijo en raíz.

```
algoritmo cortar(M,x,y)
principio
  borrar x de la lista de hijos de y,
         reduciendo y↑.grado;
  añadir x a la lista de raíces de M;
  x↑.padre:=nil;
  x↑.marca:=falso
fin
```

Montículos de Fibonacci

– El algoritmo de reducción de una clave:

```
algoritmo reducir(M,x,c)
{pre:c<x↑.clave}{post:la clave de x pasa a ser c}
principio
  x↑.clave:=c;
  y:=x↑.padre;
  si y≠nil and x↑.clave<y↑.clave ent
    cortar(M,x,y);
    cortar_arriba(M,y)
  fsi;
  si x↑.clave<M.min↑.clave ent
    M.min:=x
  fsi
fin
```

hay que re-estructurar el árbol

x pasa a ser una raíz de *M*

el nodo *y* acaba de perder un hijo, entonces… lo vemos a continuación

Montículos de Fibonacci

– "Cortar_arriba", una versión recursiva de "cortar":

- Si *y* es una raíz entonces el algoritmo no hace nada.
- Si *y* no está marcado, el algoritmo lo marca y deja de subir.
- Si *y* está marcado, se corta, y se llama a si mismo subiendo hacia el padre.
- En definitiva: un nodo está marcado cuando ha perdido el 1er hijo pero no ha perdido el 2º.

```
algoritmo cortar_arriba(M,y)
principio
  z:=y↑.padre;
  si z≠nil ent
    si y↑.marca=falso ent
      y↑.marca:=verdad
    sino
      cortar(M,y,z);
      cortar_arriba(M,z)
    fsi
  fsi
fin
```

Montículos de Fibonacci

- ¿Para qué vale la marca de los nodos?
  - Suponer que un nodo *x* “sufre” la siguiente historia:
    - en algún momento, *x* fue raíz;
    - entonces, *x* se enlazó con otro nodo;
    - después, dos hijos de *x* se eliminaron de su lista de hijos mediante cortes.
  - Tan pronto como *x* pierde su 2º hijo, *x* se corta de su padre y pasa a ser raíz.
  - El campo *marca* es verdad si han ocurrido los dos primeros pasos pero no ha perdido aún el 2º hijo.

Montículos de Fibonacci

- Coste amortizado de la reducción de una clave:
  - Primero: calcular el coste real
    - El algoritmo “reducir” tiene un coste del mismo orden que “cortar_arriba”.
    - Suponer que “cortar_arriba” se ejecuta recursivamente *c* veces en una llamada desde “reducir”, como cada llamada a “cortar_arriba” solo cuesta $O(1)$ más la nueva llamada recursiva, el coste de “reducir” es $O(c)$.

Montículos de Fibonacci

- Segundo: calcular el cambio de potencial
  - Sea M el montículo antes de reducir la clave.
  - Cada llamada recursiva a “cortar_arriba”, excepto la última, “corta” un nodo marcado y lo desmarca.
    - » Después de “reducir” quedan $a(M) + c$ árboles en el montículo: los $a(M)$ originales, los $c - 1$ que se han cortado y el de raíz x
    - » Después de “reducir” quedan como mucho $m(M) - c + 2$ nodos marcados: $c - 1$ se desmarcan con la llamada y uno puede marcarse en la última llamada recursiva.
  - Por tanto, el cambio de potencial está acotado por:

    $((a(M) + c) + 2(m(M) - c + 2)) - (a(M) + 2m(M)) = 4 - c$

- Luego el coste amortizado es, como mucho:

  $A(M') = C(M') + P(M') - P(M) = O(c) + 4 - c = O(1)$

  podemos escalar las unidades del potencial para “dominar” la constante escondida en $O(c)$

Montículos de Fibonacci

- Ahora vemos el porqué de esa función de potencial…
  - Cuando un nodo marcado *y* se corta en una operación de "cortar_arriba", su *marca* desaparece, luego el potencial se reduce en 2 unidades ($P(M) = a(M) + 2m(M)$).
  - Una de esas unidades paga por el corte y por quitar la marca y la otra unidad compensa el incremento de potencial en una unidad por el hecho de que *y* pasa a ser raíz de un nuevo árbol.

Montículos de Fibonacci

- Operación de borrado de un elemento:

```
algoritmo borrar(M,x)
principio
  reducir(M,x,-∞);
  borrarMínimo(M)
fin
```

  - Es igual que para los montículos binomiales.
  - El algoritmo hace decrecer el valor de la clave del elemento a borrar hasta el valor mínimo posible con el algoritmo “reducir”, en tiempo (amortizado) $O(1)$.
  - Después, con la operación de borrado del mínimo, de coste (amortizado) $O(D(n))$, se borra esa raíz.
  - El coste total (amortizado) es, por tanto, $O(D(n))$.

Montículos de Fibonacci

- Falta ver que el grado, $D(n)$, de cualquier nodo de un montículo de Fibonacci de n nodos está acotado por $O(\log n)$.
  - Ya vimos que si todos sus árboles son árboles binomiales desordenados, entonces $D(n) = \log n$.
  - Pero los cortes del algoritmo “reducir” pueden hacer que los árboles dejen de ser “binomiales desordenados”.
  - Veremos ahora que, dado que un nodo se corta de su padre tan pronto como pierde dos hijos, se sigue teniendo que $D(n)$ es $O(\log n)$.

Montículos de Fibonacci

– Lema 1: Sea x un nodo cualquiera de grado k de un montículo de Fibonacci, con hijos $y_1, y_2, \ldots, y_k$ (en el orden en que fueron enlazados). Se tiene:

- $y_1\uparrow.\texttt{grado} \geq 0$, y
- $y_i\uparrow.\texttt{grado} \geq i - 2$, para $i = 2, 3, \ldots, k$.

Demostración: La primera afirmación es obvia. Veamos la segunda:

- Cuando y_i ($i \geq 2$) fue enlazado a x, los otros $y_1, y_2, \ldots, y_{i-1}$ ya eran hijos de x, por tanto $x\uparrow.\texttt{grado} \geq i - 1$.
- El nodo y_i se enlaza a x sólo si $x\uparrow.\texttt{grado} = y_i\uparrow.\texttt{grado}$, por tanto $y_i\uparrow.\texttt{grado} \geq i - 1$ en el momento de enlazarse.
- Desde enlazarse a x, y_i sólo ha podido perder como mucho un hijo, de lo contrario (si hubiese perdido dos hijos) se habría cortado de x, por tanto $y_i\uparrow.\texttt{grado} \geq i - 2$.

Montículos de Fibonacci

Llegamos a la explicación del nombre de estos montículos…

- Lema 2: Sea x un nodo cualquiera de grado k de un montículo de Fibonacci. Sea $card(x)$ el número de nodos del subárbol con raíz x (incluido x). Entonces:

$$card(x) \geq F_{k+2} \geq \phi^k, \text{ donde:}$$

$$F_k = \begin{cases} 0, & \text{si } k = 0 \\ 1, & \text{si } k = 1 \\ F_{k-1} + F_{k-2}, & \text{si } k \geq 2 \end{cases}$$

es el $k-$ésimo nº de Fibonacci, y

$\phi = \left(1 + \sqrt{5}\right)/2$ es la razón aúrea ($=1'61803...$).

Montículos de Fibonacci

- Demostración del Lema:
  - Sea s_k el valor mínimo de $card(z)$ para todos los z con $z\uparrow$`.grado` $= k$.
  - Por tanto: $s_0 = 1$, $s_1 = 2$, $s_2 = 3$, $s_k \leq card(x)$.
  - Sean $y_1, y_2, \ldots, y_k$ los hijos de x (en el orden en que fueron enlazados).
  - Por el Lema 1, se tiene:

$$card(x) \geq s_k \geq 2 + \sum_{i=2}^{k} s_{i-2}$$

(1 por el nodo x + 1 por y_1)

Montículos de Fibonacci

- Demostración del Lema (cont.):
  - Veamos un resultado previo para los F_k:

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i, \text{ para } k \geq 0$$

    - Demostración: por inducción sobre *k*.
      - » $k = 0$: $F_2 = 1 + F_0 = 1 + 0 = 1$
      - » Suponiendo que

$$F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$$

        se tiene:

$$F_{k+2} = F_k + F_{k+1} = F_k + \left(1 + \sum_{i=0}^{k-1} F_i\right) = 1 + \sum_{i=0}^{k} F_i$$

Montículos de Fibonacci

- Demostración del Lema (cont.):
  - Ahora veamos por inducción sobre k que $s_k \geq F_{k+2}$ para todo k no negativo.
    - Para $k = 0$ y $k = 1$ es trivial.
    - Sea $k \geq 2$. Supóngase que $s_i \geq F_{i+2}$ para $i = 0, 1, \ldots, k-1$. Entonces:

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2} \geq 2 + \sum_{i=2}^{k} F_i = 1 + \sum_{i=0}^{k} F_i = F_{k+2}$$

  - Sólo falta ver que $F_{k+2} \geq \phi^k$.
    - Se puede demostrar (ejercicio) a partir del resultado:

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}, \text{ donde } \hat{\phi} = \frac{1-\sqrt{5}}{2}$$

      que, a su vez, puede demostrarse por inducción a partir de la definición.

Montículos de Fibonacci

– Corolario (del Lema 2): El grado máximo, $D(n)$ de cualquier nodo de un montículo de Fibonacci de n nodos es $O(\log n)$.

  Demostración:

  - Sea x un nodo cualquiera y $k = x\uparrow.\texttt{grado}$.
  - Por el Lema 2, $n \geq card(x) \geq \phi^k$. Por tanto: $k \leq \log_\phi n$, y se sigue el resultado.