

1. colaPaquetes

Interfaz

se explica con: COLA DE PRIORIDAD(PAQUETE).

géneros: colaPaquetes.

Operaciones básicas de colaPaquetes

VACÍA() $\rightarrow res : colaPaquetes$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{obs} vacía\}$

Complejidad: $\Theta(1)$

Descripción: genera un colaPaquetes vacía.

ENCOLAR(**in/out** $c : colaPaquetes(paquete)$, **in** $a : paquete$)

Pre $\equiv \{c =_{obs} c_0\}$

Post $\equiv \{c =_{obs} encolar(c_0, a)\}$

Complejidad: $O(\log(n))$

Descripción: pone en la posición adecuada al orden al paquete a en la colaPaquetes c.

VACÍA?(**in** $c : colaPaquetes(paquete)$) $\rightarrow res : bool$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{obs} vacía?(c)\}$

Complejidad: $\Theta(1)$

Descripción: devuelve true si y sólo la colaPaquetes está vacía o, lo que es lo mismo, que no posee paquetes.

PRÓXIMO(**in** $c : colaPaquetes(paquete)$) $\rightarrow res : paquete$

Pre $\equiv \{\neg vacía?(c)\}$

Post $\equiv \{alias(res =_{obs} próximo(c))\}$

Complejidad: $\Theta(1)$

Descripción: devuelve el próximo paquete de la colaPaquetes. Este es el de mayor prioridad

Aliasing: res es modificable si y sólo si d es modificable.

DESENCOLAR(**in/out** $c : colaPaquetes(paquete)$)

Pre $\equiv \{c =_{obs} c_0 \wedge \neg vacía?(c)\}$

Post $\equiv \{c =_{obs} desencolar(c_0)\}$

Complejidad: $O(\log(n))$

Descripción: modifica la colaPaquetes quitando el próximo paquete y después reordenandola.

Representación

Representación de la colaPaquetes

colaPaquetes(paquete) se representa con heap

donde heap es $tupla(raiz : puntero(paquete), izq : puntero(heap), der : puntero(heap), cantidadElementos : nat)$

Rep : heap $\rightarrow bool$

Rep(c) $\equiv true \iff (c.cantidadElementos = 0 \iff (c.raiz = NULL \wedge c.izq = NULL \wedge c.der = NULL)) \wedge (c.cantidadElementos = 1 \iff (c.raiz \neq NULL \wedge c.izq = NULL \wedge c.der = NULL)) \wedge (c.cantidadElementos = 2 \iff (c.raiz \neq NULL \wedge c.izq \neq NULL \wedge c.der = NULL)) \wedge (c.cantidadElementos \geq 3 \iff (c.raiz \neq NULL \wedge c.izq \neq NULL \wedge c.der \neq NULL)) \wedge_L (c.cantidadElementos \geq 3 \Rightarrow c.cantidadElementos = *c.izq.cantidadElementos + *c.der.cantidadElementos + 1) \wedge (c.izq \neq NULL \Rightarrow (*c.raiz.prioridad \leq *c.izq.raiz.prioridad \wedge Rep(*c.izq))) \wedge (c.der \neq NULL \Rightarrow (*c.raiz.prioridad \leq *c.der.raiz.prioridad \wedge Rep(*c.der)))$

-Como se utilizan punteros tenemos que decir que no se permite que dos heap en un árbol tengan como punteros izq o der a otros iguales. En otras palabras que no se formen bucles en el árbol

Abs : heap $c \rightarrow colaPaquetes(paquete)$ $\{Rep(c)\}$

$\text{Abs}(c) \equiv e$

$\text{vacía?}(e) \Leftrightarrow c.\text{raiz} = \text{NULL} \wedge_{\text{L}} (\text{vacía?}(e) \Rightarrow_{\text{L}} \text{próximo}(e) = *c.\text{raiz} \wedge \text{desencolar}(e) = \text{funcionParaDesencolar}(c))$
-funcionParaDesencolar consiste en tomar el valor de el último elemento del heap y colocarlo en la posición del primero (esto es suponiendo que exista alguno a parte del primero, en caso contrario la función terminaría aquí). Con esto nos desasemos del elemento que teníamos que desencolar. Ahora para que restablezca el invariante de heap tenemos que tomar el primer elemnto, que es el que agregamos, e ir bajandolo por izquieda o derecha hasta que encontremos un lugar donde no tenga hijos o los que tenga sean menores en prioridad a él.

Algoritmos

iVacía() $\rightarrow res : \text{heap}$

$res \leftarrow \langle \text{NULL}, \text{NULL}, \text{NULL}, 0 \rangle$

$\triangleright O(1)$

Complejidad: $\Theta(1)$

```

iEncolar(in/out  $c$  : heap, in  $p$  : paquete)
  *aPoner : puntero(paquete)  $\leftarrow \delta p$  ▷ O(1)
  if  $c.raiz = \text{NULL}$  then ▷ O(3)
     $c.raiz \leftarrow aPoner$  ▷ O(1)
     $c.cantidadElementos \leftarrow 1$  ▷ O(1)
  else ▷ O(2 + \log(n)) = O(\log(n))
    *recorrido : puntero(heap)  $\leftarrow \delta c$  ▷ O(1)
     $llegoAPosicion : \text{bool} \leftarrow \text{false}$  ▷ O(1)
    while  $\neg \text{llegoAPosicion}$  do ▷ O(12 \times \log(n)) = O(\log(n))
      *recorrido.cantidadElementos  $\leftarrow$  *recorrido.cantidadElementos + 1
      if  $\text{recorrido.izq} = \text{NULL}$  then ▷ O(3)
         $llegoAPosicion \leftarrow \text{true}$  ▷ O(1)
        *nuevaRaiz : puntero(paquete)  $\leftarrow aPoner$  ▷ O(1)
        if **recorrido.raiz.prioridad > *aPoner.prioridad then ▷ O(5)
          *aux : puntero(paquete)  $\leftarrow$  *recorrido.raiz ▷ O(1)
          *nuevoHeap : puntero(heap)  $\leftarrow \delta \langle \text{aux}, \text{NULL}, \text{NULL}, 1 \rangle$  ▷ O(1)
          *recorrido.izq  $\leftarrow$  nuevoHeap ▷ O(1)
          *recorrido.raiz  $\leftarrow$  nuevaRaiz ▷ O(1)
        else ▷ O(3)
          *nuevoHeap : puntero(heap)  $\leftarrow \delta \langle \text{nuevaRaiz}, \text{NULL}, \text{NULL}, 1 \rangle$  ▷ O(1)
          *recorrido.izq  $\leftarrow$  nuevoHeap ▷ O(1)
        end if
      else if  $\text{recorrido.der} = \text{NULL}$  then ▷ O(10)
         $llegoAPosicion \leftarrow \text{true}$  ▷ O(1)
        *nuevaRaiz : puntero(paquete)  $\leftarrow aPoner$  ▷ O(1)
        if **recorrido.raiz.prioridad > *aPoner.prioridad then ▷ O(8)
          *aux1 : puntero(paquete)  $\leftarrow$  *recorrido.raiz ▷ O(1)
          *aux2 : puntero(paquete)  $\leftarrow$  *recorrido.izq.raiz ▷ O(1)
          *nuevoHeap1 : puntero(heap)  $\leftarrow \delta \langle \text{aux1}, \text{NULL}, \text{NULL}, 1 \rangle$  ▷ O(1)
          *nuevoHeap2 : puntero(heap)  $\leftarrow \delta \langle \text{aux2}, \text{NULL}, \text{NULL}, 1 \rangle$  ▷ O(1)
          *recorrido.izq  $\leftarrow$  nuevoHeap1 ▷ O(1)
          *recorrido.der  $\leftarrow$  nuevoHeap2 ▷ O(1)
          *recorrido.raiz  $\leftarrow$  nuevaRaiz ▷ O(1)
        else ▷ O(2)
          *nuevoHeap : puntero(heap)  $\leftarrow \delta \langle \text{nuevaRaiz}, \text{NULL}, \text{NULL}, 1 \rangle$  ▷ O(1)
          *recorrido.der  $\leftarrow$  nuevoHeap ▷ O(1)
        end if
      else ▷ O(12)
        nivelCompleto : bool  $\leftarrow (2 \times \text{recorrido.der.cantidadElementos}) + 1 \Rightarrow \text{recorrido.izq.cantidadElementos}$ 
        ▷ O(1)
        if **recorrido.raiz.prioridad > *aPoner.prioridad then ▷ O(11)
          *nuevaRaiz : puntero(paquete)  $\leftarrow aPoner$  ▷ O(1)
          *aux1 : puntero(paquete)  $\leftarrow$  *recorrido.raiz ▷ O(1)
          *aux2 : puntero(paquete)  $\leftarrow$  *recorrido.izq.raiz ▷ O(1)
          *aux3 : puntero(paquete)  $\leftarrow$  *recorrido.der.raiz ▷ O(1)
          *recorrido.raiz  $\leftarrow$  nuevaRaiz ▷ O(1)
          *recorrido.izq.raiz  $\leftarrow$  aux1 ▷ O(1)
          *recorrido.der.raiz  $\leftarrow$  aux2 ▷ O(1)
          aPoner  $\leftarrow$  aux3 ▷ O(1)
        end if
        if nivelCompleto then ▷ O(2)
          recorrido  $\leftarrow$  *recorrido.izq ▷ O(1)
        else ▷ O(2)
          recorrido  $\leftarrow$  *recorrido.der ▷ O(1)
        end if
      end if
    end while
  end if

```

Complejidad: $O(\log(n))$

Justificación: $O(\log(n) + 1) = O(\log(n))$. Tomamos en consideración el camino más costoso del if.

Vacia?(in $c : \text{heap}$) $\rightarrow res : \text{bool}$

$res \leftarrow c.\text{cantidadElementos} = 0$

$\triangleright O(1)$

Complejidad: $O(1)$

iPróximo(in $c : \text{heap}$) $\rightarrow res : \text{paquete}$

$res \leftarrow *c.\text{raiz}$

$\triangleright O(1)$

Complejidad: $O(1)$

```

iDesencolar(in/out c: heap)
  if c.izq = NULL then                                     ▷ O(3)
    c.raiz ← NULL                                           ▷ O(1)
    c.cantidadElementos ← 0                                 ▷ O(1)
  else                                                       ▷ O(2 + 2 x log(n)) = O(2 x log(n)) = O(log(n))
    *recorrido : puntero(heap) ← δc                         ▷ O(1)
    final : bool ← false                                    ▷ O(1)
    while ¬ final do                                         ▷ O(4 x log(n)) = O(log(n))
      if *recorrido.izq = NULL then                           ▷ O(2)
        final ← true                                         ▷ O(1)
      else if *recorrido.der = NULL then                       ▷ O(4)
        final ← true                                         ▷ O(1)
        *recorrido.cantidadElementos = *recorrido.cantidadElementos - 1 ▷ O(1)
        recorrido ← *recorrido.izq                           ▷ O(1)
      else                                                     ▷ O(4)
        nivelCompleto : bool ← (2 x *recorrido.der.cantidadElementos) + 1 => **recorrido.izq.cantidadElementos
        ▷ O(1)
        *recorrido.cantidadElementos = *recorrido.cantidadElementos - 1 ▷ O(1)
        if nivelCompleto then                                 ▷ O(2)
          recorrido ← *recorrido.der                         ▷ O(1)
        else                                                   ▷ O(2)
          recorrido ← *recorrido.izq                         ▷ O(1)
        end if
      end if
    end while
    c.raiz ← *recorrido.raiz                                 ▷ O(1)
    recorrido ← NULL                                         ▷ O(1)
    *recorrido2 : puntero(heap) ← δc                         ▷ O(1)
    llegoAPosicion : bool ← false                            ▷ O(1)
    while ¬ llegoAPosicion do                                 ▷ O(6 x log(n)) = O(log(n))
      if *recorrido2.izq = NULL then                           ▷ O(2)
        llegoAPosicion ← true                                ▷ O(1)
      else if *recorrido2.der = NULL then                       ▷ O(1)
        llegoAPosicion ← true                                ▷ O(1)
      if ***recorrido2.izq.raiz.prioridad < **recorrido2.raiz.prioridad then ▷ O(4)
        *aux : puntero(paquete) ← *recorrido2.raiz          ▷ O(1)
        *recorrido2.raiz ← **recorrido2.izq.raiz            ▷ O(1)
        **recorrido2.izq.raiz ← aux                           ▷ O(1)
      end if
    else                                                       ▷ O(6)
      nivelCompleto ← (2 x **recorrido2.der.cantidadElementos) + 1 => **recorrido2.izq.cantidadElementos
      ▷ O(1)
      if **recorrido2.raiz.prioridad > ***recorrido2.izq.raiz.prioridad ∧ ¬ nivelCompleto then ▷ O(5)
        *aux : puntero(paquete) ← *recorrido2.raiz          ▷ O(1)
        *recorrido2.raiz ← **recorrido2.izq.raiz            ▷ O(1)
        **recorrido2.izq.raiz ← aux                           ▷ O(1)
        recorrido2 ← *recorrido2.izq                         ▷ O(1)
      else if **recorrido2.raiz.prioridad > ***recorrido2.der.raiz.prioridad ∧ nivelCompleto then ▷ O(5)
        *aux : puntero(paquete) ← *recorrido2.raiz          ▷ O(1)
        *recorrido2.raiz ← **recorrido2.der.raiz            ▷ O(1)
        **recorrido2.der.raiz ← aux                           ▷ O(1)
        recorrido2 ← *recorrido2.der                         ▷ O(1)
      else                                                     ▷ O(2)
        llegoAPosicion ← true                                ▷ O(1)
      end if
    end if
  end while
end if

```

Complejidad: $O(\log(n))$

Justificación: Tomamos en consideración el camino más costoso del if.
