Heap Sort (Montículo)

2 etapas

1) Armado del Heap(montículo)

2) Re ordenamiento de los datos para dejar el orden final

Heap: una secuencia de datos tal que cumple con:

a[i]>a[2\*i]

a[i]>a[2\*i+1]

Ejemplo:

a[3]>a[6] y a[3]>a[7]

Armado de la heat

1

2 3

4 5 6 7

8 9 10 11 12 13 14 15

Con n datos, insertamos uno a uno los datos sobre la secuencia en la primera posición disponible y se corrobora que cumpla con la condición de la heap, sino cumple, hay que hacer los swap correspondientes hasta que valga la heap.

5-8-1- 12-4-9-3-10-6-7

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| 5 |  |  |  |  |  |  |  |  |  |
| 8 | 5 |  |  |  |  |  |  |  |  |
| 8 | 5 | 1 |  |  |  |  |  |  |  |
| 12 | 8 | 1 | 5 |  |  |  |  |  |  |
| 12 | 8 | 1 | 5 | 4 |  |  |  |  |  |
| 12 | 8 | 9 | 5 | 4 | 1 |  |  |  |  |
| 12 | 8 | 9 | 5 | 4 | 1 | 3 |  |  |  |
| 12 | 10 | 9 | 8 | 4 | 1 | 3 | 5 |  |  |
| 12 | 10 | 9 | 8 | 4 | 1 | 3 | 5 | 6 |  |
| 12 | 10 | 9 | 8 | 7 | 1 | 3 | 5 | 6 | 4 |

Reordenamiento de los datos

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **suelto** |
| 12 | 10 | 9 | 8 | 7 | 1 | 3 | 5 | 6 | 4 |  |
| 10 | 8 | 9 | 6 | 7 | 1 | 3 | 5 | 4 | 12 | 4 |
| 9 | 8 | 4 | 6 | 7 | 1 | 3 | 5 | 10 | 12 | 4 |
| 8 | 7 | 4 | 6 | 5 | 1 | 3 | 9 | 10 | 12 | 5 |
| 7 | 6 | 4 | 3 | 5 | 1 | 8 | 9 | 10 | 12 | 3 |
| 6 | 5 | 4 | 3 | 1 | 7 | 8 | 9 | 10 | 12 | 1 |
| 5 | 3 | 4 | 1 | 6 | 7 | 8 | 9 | 10 | 12 | 1 |
| 4 | 3 | 1 | 5 | 6 | 7 | 8 | 9 | 10 | 12 | 1 |
| 3 | 1 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 | 1 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 |  |

Orden del argoritmo

n pasos con lg2 comparaciones a lo sumo en cada paso

O(nlgn)

El final de la complejidad del algoritmo es 2nlgn → O(nlgn)

**Unidad 4 Arboles**

Estructura de datos de la lista y pila NO es lineal

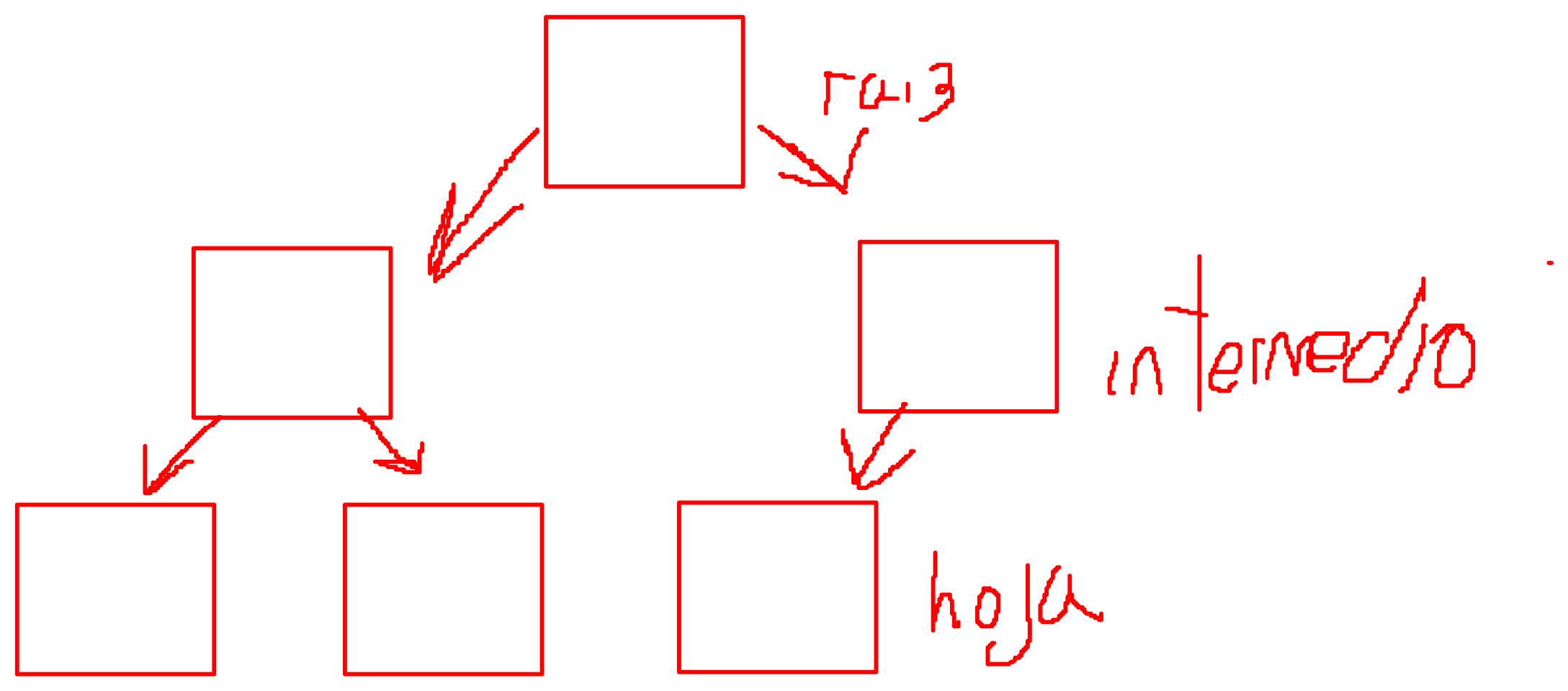
Esta basado en nodos, pero puede tener más de 1 nodo relacionado con cada nodo. El número de nodos puede relacionar con cada nodo es finito, y definido como parámetro del árbol.

Si “m” es el número máximo de nodos que pueden relacionarse con cada nodo, el árbol se llama m-ario.

Si m=2 → Árbol binario

Si m=3 → Árbol ternario

En general hablamos de árboles binarios, salvo que se especifique lo contrario.



Nodo raíz= comienzo del árbol

nodo hijo= son los nodos relacionados con el nodo

nodo padre= es el nodo desde el cual viene la relación

nodo hoja= que no tiene hijos

es estructura recursiva

En un árbol binario, los dos hijos se llamas hijo izquierdo e hijo derecho

Especificación algebraica del tipo árbol de @(alpha) usa tipo Nat, Boolean, Lista de @

un arbol binario de tipo @ es:

1) Un nodo vació

2) Un nodo con un dato de tipo @ y un link con 2 árboles

Un arbol vacio es un nodo que no tiene dato(nulo) y los links a sus 2 nodos también son nulos

operaciones

constructores

crear: → arbol

plantar @ x árbol x árbol → árbol

nodo raiz: plantar(5,crearArb(),crearArb())

otras operaciones

hi: árbol→ árbol

hd: árbol→ árbol

dato: Arbol→ @

nulo: arbol→ bool

altura:Arbol → Nat devuelve la máxima cantidad de nodos entre la raíz y el nodo hoja más alejado de la raiz

nulo(crearArb())=true

nulo(plantar(a,t1,t2))=false a:@ t1,t2:arbol

hi(plantar(a,t1,t2))=t1 hi(crearArb())=error

hd(plantar(a,t1,t2))=t2 hd(crearArb())=error

dato(plantar(a,t1,t2))=a dato(crearArb())=error

altura(crearArb())= 0

altura(plantar(a,t1,t2)= 1+ max(altura(t1),altura(t2))

Ej plantar(5,plantar(2,crearArb(),crearArb()), crearArb())=t

hi(t)=plantar(2,crearArb(),crearArb()),

hd(t)=crearArb()

dato(t)=5

dato(hi(t))=dato(plantar(2,crearArb(),crearArb()))=2

recorrido del árbol: Dado un árbol retorna una lista con los valores de los nodos del árbol

preorden: Arbol → lista

inorden: Arbol → lista

posorden: Arbol → lista

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 6 |  |  |  |  |
|  | 3 |  |  | 8 |  |  |  |
|  |  |  | 10 |  | 2 |  |  |
|  |  |  |  |  |  | 6 |  |

preorden(crearArb())= nuevaLista()

inorden(crearArb())= nuevaLista()

posorden(crearArb())= nuevaLista()

preorden(plantar(a,t1,t2)) = [a] ++ preorden(t1) ++ preorden(t2)

= invertir(agregar(a,preordenten(t1),preorden(t2)

en el ejemplo [6,3,8,10,2,6]

inorden(plantar(a,t1,t2))= inorden(t1) ++ [a] ++ inorden (t2)

en el ejemplo=[3,6,10,8,2,6]

posorden(plantar,a,t1,t2)= posorden(t1)++posorden(t2)++[a]

en el ejemplo= [3,10,6,2,8,6]

3+(120\*9-6)(inorden)

3 10 9 6-\*+(posorden)

árbol binario de búsqueda (ABB)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 6 |  | |  |  |  |
|  | 3 |  |  |  | | 10 |  |  |
|  |  |  |  | 8 | |  | 12 |  |
|  |  |  |  |  | 9 |  |  |  |

es un arbol t= plantar(a,t1,t2) tal que raiz(t)= a > raiz(hi(t)) y raiz(t)<raiz(hd(t))

raiz(t1)<a<raiz(t2)

y que t1 y t2 también respeten la condición de abb

Uso de abb: hacer búsquedas

buscar(a,abb) : @ x Arbol→ Bool a:@ abb: arbol que cumple con ser abb

buscar(a,abb)= { a=raiz(abb) → true

esnilo(abb) → false

a<raiz(abb) → buscar(a,hi(abb))

inorden(abb) da una lista ordenada=[3,6,8,10,12]

addabb:arbol x @ →arbol

addabb(crearArb(),a)= plantar (a,crearArb(),crearArb())

add(plantar(a1,t1,t2),x)= { x<a → plantar(a,addabb(t1,x),t2)

x>a → plantar(a,t1,addabb(t2,x))

comparaciones minimas → miniizar la altura del arbol

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 5 |  |  |  |  |  |  |
| 3 |  | 7 |  |  |  |  |  |
|  |  |  | 10 |  |  |  |  |
|  |  |  |  | 12 |  |  |  |
|  |  |  |  |  | 20 |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 10 |  |  |  |
|  | 5 |  |  | 12 |  |  |
| 3 |  | 7 |  |  | 20 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

arbol completo t: altura(hi(t))= altura(hd(t)) y completo(hi(t)) y completo hd(t))

semicompleto t: completo(t) o bien que

altura(hi(t))= altura (hd(t)) pero completo (hi(t)) y semicompleto (hd(t))

altura(hi(t))= 1+altura(hd(t)) y semicompleto(hi(t)) y completo (hd(t))

balanceado t: abs(altura(hi(t))- altura(hd(t))<= 1 y balanceado (hi(t)) y balanceado (hd(t))