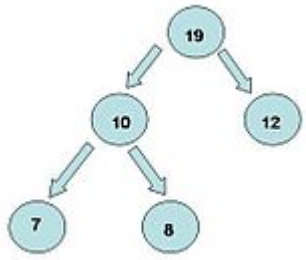
WIKIPEDIA

Montículo (informática)

En computación, un **montículo** (o ***heap*** en inglés) es una estructura de datos del tipo árbol con información perteneciente a un conjunto ordenado. Los montículos máximos tienen la característica de que cada nodo padre tiene un valor mayor que el de cualquiera de sus nodos hijos, mientras que en los montículos mínimos, el valor del nodo padre es siempre menor al de sus nodos hijos.



Ejemplo de montículo de máximos.

Un árbol cumple la condición de montículo si satisface dicha condición y además es un árbol binario casi completo. Un árbol binario es completo cuando todos los niveles están llenos, con la excepción del último, que se llena desde la izquierda hacia la derecha.

En un montículo de prioridad, el mayor elemento (o el menor, dependiendo de la relación de orden escogida) está siempre en el nodo raíz. Por esta razón, los montículos son útiles para implementar colas de prioridad. Una ventaja que poseen los montículos es que, por ser árboles completos, se pueden implementar usando arreglos (arrays), lo cual simplifica su codificación y libera al programador del uso de punteros.

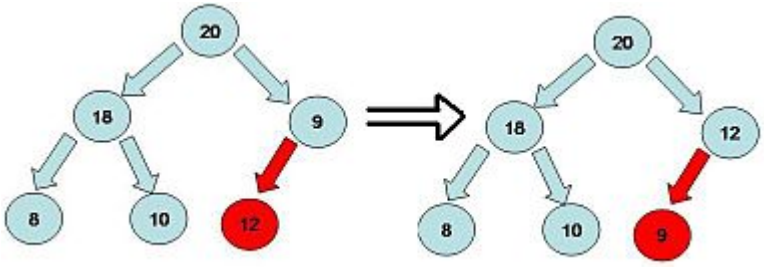
La eficiencia de las operaciones en los montículos es crucial en diversos algoritmos de recorrido

Índice

Operaciones

Insertar

Monticulus: Esta operación parte de un elemento y lo inserta en un montículo aplicando su criterio de ordenación.Si suponemos que el monticulo está estructurado de forma que la raíz es mayor que sus hijos, comparamos el elemento a insertar (incluido en la primera posición libre)



Cómo se inserta un elemento en un montículo de máximos.

con su padre.Si el hijo es menor que el padre,entonces el elemento es insertado correctamente, si ocurre lo contrario sustituimos el hijo por el padre.

¿Y si la nueva raíz sigue siendo más grande que su nuevo padre?. Volvemos a hacer otra vez dicho paso hasta que el montículo quede totalmente ordenado. En la imagen adjunta vemos el ejemplo de cómo realmente se inserta un elemento en un montículo. Aplicamos la condición de que cada padre sea mayor que sus hijos, y siguiendo dicha regla el elemento a insertar es el 12. Es mayor que su padre, siguiendo el método de ordenación, sustituimos el elemento por su padre que es 9 y así quedaría el montículo ordenado.

Ahora veremos la implementación en varios lenguajes de programación del algoritmo de inserción de un elemento en un montículo.

En *Maude* el insertar se realiza a través de un constructor:

```
op insertarHeap : X$Elt Heap{X} -> HeapNV{X}
eq insertarHeap(R, crear) = arbolBin(R, crear, crear) .
eq insertarHeap(R1, arbolBin(R2, I, D)) =
       if ((altura(I) > altura(D)) and not estaLleno?(I))
           or (((altura(I) == altura(D)) and estaLleno?(D))
        then arbolBin(max(R1, R2),insertarHeap(min(R1, R2), I), D)
       else arbolBin(max(R1, R2), I,insertarHeap(min(R1, R2), D))
       fi .
```

En pseudolenguaje quedaría:

```
PROC Flotar ( M, i )
 MIENTRAS (i>1) ^ (M.Vector_montículo[i div 2] < M.Vector montículo[i] HACER
            intercambiar M.Vector montículo[i div 2] ^ M.Vector_montículo[i]
            i = i div 2
 FIN MIENTRAS
FIN PROC
PROC Insertar ( x, M )
 SI M.Tamaño_montículo = Tamaño_máximo ENTONCES
          error Montículo lleno
 SINO M.Tamaño_montículo = M.Tamaño_montículo + 1
          M.Vector_montículo[M.Tamaño montículo] = x
          Flotar ( M, M.Tamaño_montículo )
FIN PROC
```

En Java el código sería el siguiente:

```
public void insertItem(Object k, Object e) throws InvalidKeyException {
   if(!comp.isComparable(k))
      throw new InvalidKeyException("Invalid Key");
   Position z = T.add(new Item(k, e));
   Position u;
   while(!T.isRoot(z)) { // bubbling-up
```

```
        u = T.parent(z);
        if(comp.isLessThanOrEqualTo(key(u),key(z)))
            break;
        T.swapElements(u, z);
        z = u;
    }
}
```

Eliminar

En este caso eliminaremos el elemento máximo de un montículo. La forma más eficiente de realizarlo sería buscar el elemento a borrar, colocarlo en la raíz e intercambiarlo por el máximo valor de sus hijos satisfaciendo así la propiedad de montículos de máximos. En el ejemplo representado vemos como 19 que es el elemento máximo es el sujeto a eliminar.

Se puede observar que ya está colocado en la raíz al ser un montículo de máximos, los pasos a seguir son:

1. Eliminar el elemento máximo (colocado en la raíz).
2. Hemos de subir el elemento que se debe eliminar, para cumplir la condición de montículo a la raíz, que ha quedado vacía.
3. Una vez hecho esto queda el último paso el cual es ver si la raíz tiene hijos mayores que ella si es así, aplicamos la condición y sustituimos el padre por el mayor de sus progenitores.

A continuación veremos la especificación de eliminar en distintos lenguajes de programación.

En Maude el código será el siguiente:

```
eq eliminarHeap(crear) = crear .
eq eliminarHeap(HNV) =
    hundir(arbolBin(ultimo(HNV),
    hijoIzq(eliminarUltimo(HNV)),
    hijoDer(eliminarUltimo(HNV))
)
```

Donde hundir es una operación auxiliar que coloca el nodo en su sitio correspondiente.

En código Java:

```
public Object removeMin() throws PriorityQueueEmptyException {
    if(isEmpty())
        throw new PriorityQueueEmptyException("Priority Queue Empty!");
    Object min = element(T.root());
    if(size() == 1)
        T.remove();
    else {
        T.replaceElement(T.root(), T.remove());
        Position r = T.root();
        while(T.isInternal(T.leftChild(r))) {
            Position s;
            if(T.isExternal(T.rightChild(r)) || comp.isLessThanOrEqualTo(key(T.leftChild(r)),key(T.rightChild(r))))
                s = T.leftChild(r);
            else
                s = T.rightChild(r);
            if(comp.isLessThan(key(s), key(r))) {
                T.swapElements(r, s);
                r = s;
            }
            else
                break;
        }
    }
}
```

Tras haber especificado ambas operaciones y definir lo que es un montículo sólo nos queda por añadir que una de las utilizaciones más usuales del tipo *heap* (montículo) es en el algoritmo de ordenación de montículo (*heapsort*). También puede ser utilizado como montículo de prioridades donde la raíz es la de mayor prioridad.

Véase también

- Montículo binario
- Montículo binómico
- Montículo de Fibonacci
- Montículo suave
- Montículo 2-3