Algoritmi e Strutture Dati

Esercitazione 7

Domenico Fabio Savo

Esercitazione: heap

Abbiamo visto come utilizzare i MAX-HEAP nell'algoritmo di ordinamento **heapSort** che permette di ordinare un array di dimensione **n** in **O(n log n).**

Per utilizzare il MAX-HEAP nell'algoritmo heapSort è stata introdotta la funzione **`getMax(heap H)` → `elem`** la quale dato il MAX-HEAP H restituisce il massimo valore contenuto in H e lo rimuove da H.

La funzione **`getMax`** fa uso della funzione:

```
fixHeap(nodo v, heap H)
```

che "ripara" un MAX-HEAP in cui tutti i nodi tranne **v** soddisfano la proprietà di ordinamento.

La procedura fixHeap

```
Algoritmo fixHeap(nodo v, heap H)
    if (v è una foglia) then return
    else
        sia u il figlio di v con chiave massima
        if ( chiave(v) < chiave(u) ) then
            scambia chiave(v) e chiave(u)
            fixHeap(u,H)
```

Tempo di esecuzione: **O(log n)**

La funzione getMax

getMax(heap H) → elem **e**

1. elem e ← chiave(radice di H);
2. copio nella radice la chiave contenuta nella la foglia più a destra dell'ultimo livello;
3. rimuovo la foglia;
4. ripristino la proprietà di ordinamento dell'heap richiamando **fixHeap** sulla radice;
5. **return e**;

L'algoritmo sfrutta la caratteristica dei MAX-HEAP per cui il valore massimo è sempre contenuto nella radice dell'albero!

Tempo di esecuzione: **O(log n)**

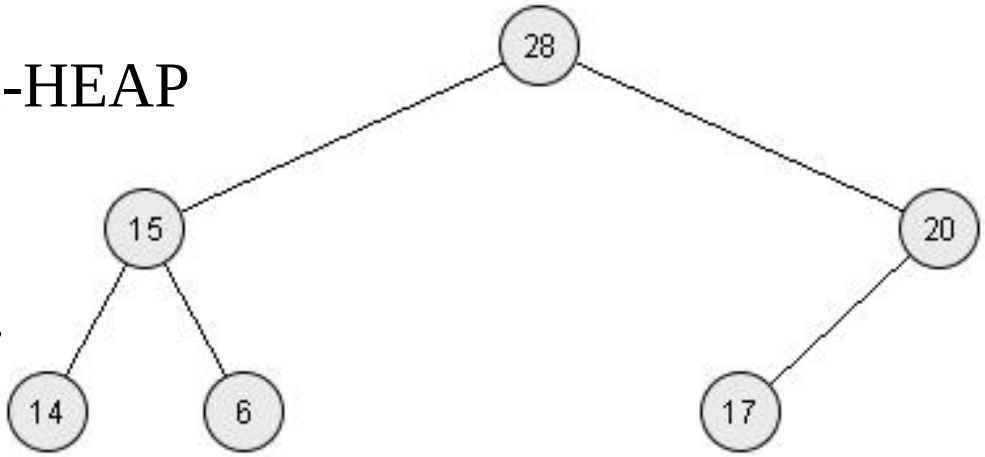
Esercitazione: heap

Esercizio:

Considerare il seguente MAX-HEAP:

Mostrare la struttura del MAX-HEAP dopo aver estratto il massimo.

Ripetere l'operazione tre volte.

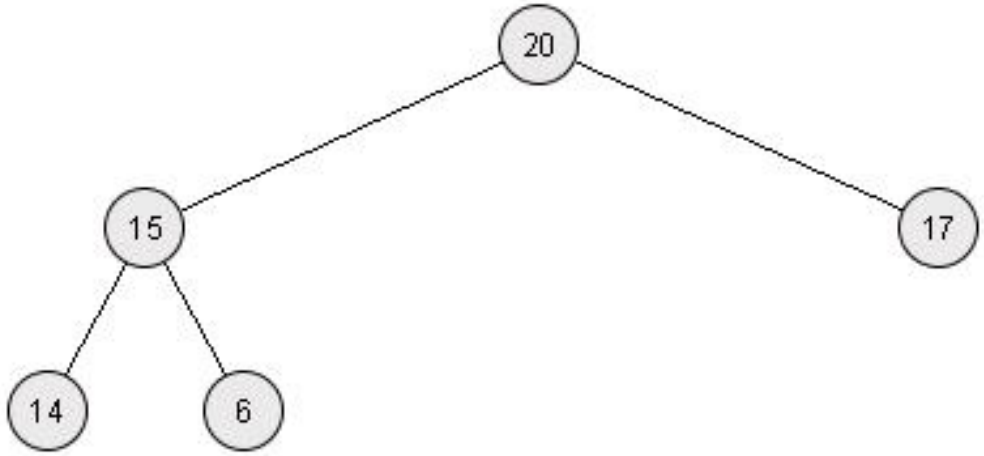


Esercitazione: heap

Soluzione:

Prima estrazione:

getMax(heap H) → 28

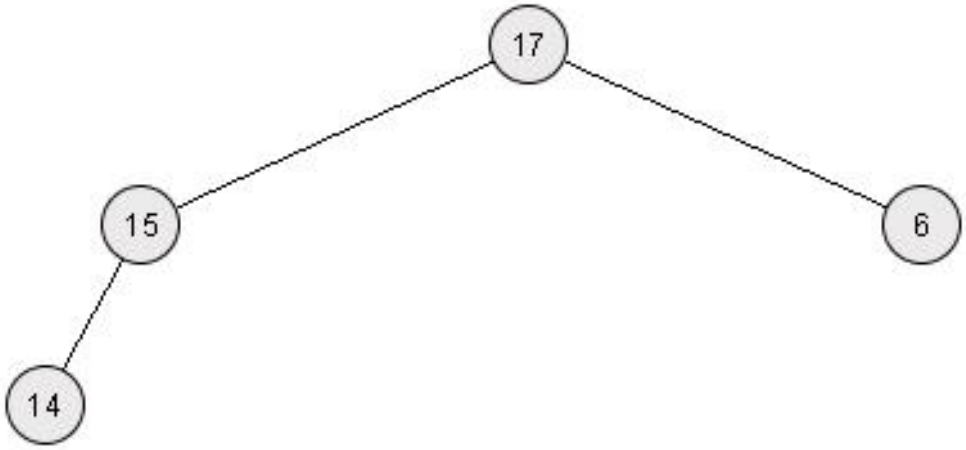


Esercitazione: heap

Soluzione:

Seconda estrazione:

getMax(heap H) → 20

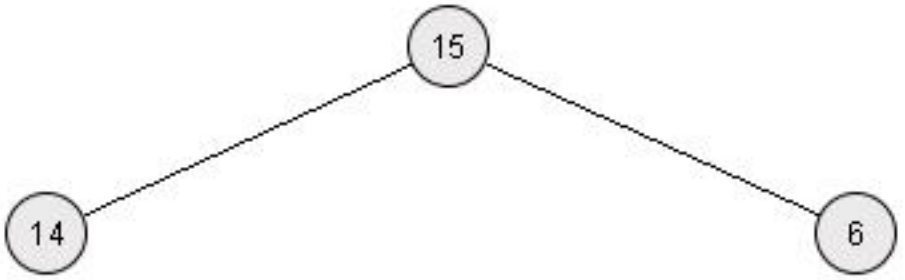


Esercitazione: heap

Soluzione:

Terza estrazione:

getMax(heap H) → 17



L'algoritmo HeapSort

```
Algoritmo heapSort(array S di dimensione n)
if (S è vuoto) then return S;
else {
    - costruisci un albero binario H con gli elementi di S;
    - ordinalo usando heapyfy(H);//costa O(n)
    - crea un coda C vuota.
    - while(H non è vuoto){          //cicla n volte
        C.enqueue(getMax(H));        //getMax ha costo O(log n)
    - copia gli elementi di C in S nel giusto ordine
    - return S
    }
```

Nella versione dell'algoritmo HeapSort visto a lezione viene utilizzato un MAX-HEAP.

L'algoritmo funzionerebbe ugualmente se volessimo utilizzare un MIN-HEAP?

Ordinamento *in loco*

Diciamo che un algoritmo di ordinamento è in grado di operare ***in loco*** se utilizza solo la memoria necessaria per mantenere gli elementi in ingresso (ES: l'array) e uno spazio addizionale **costante**.

La versione dell'algoritmo HeapSort presentata NON lavora in loco. Infatti utilizza una struttura in cui vengono memorizzati i nodi estratti dall'heap (la coda).

Domanda: come possiamo modificare l'algoritmo in modo che operi *in loco?*

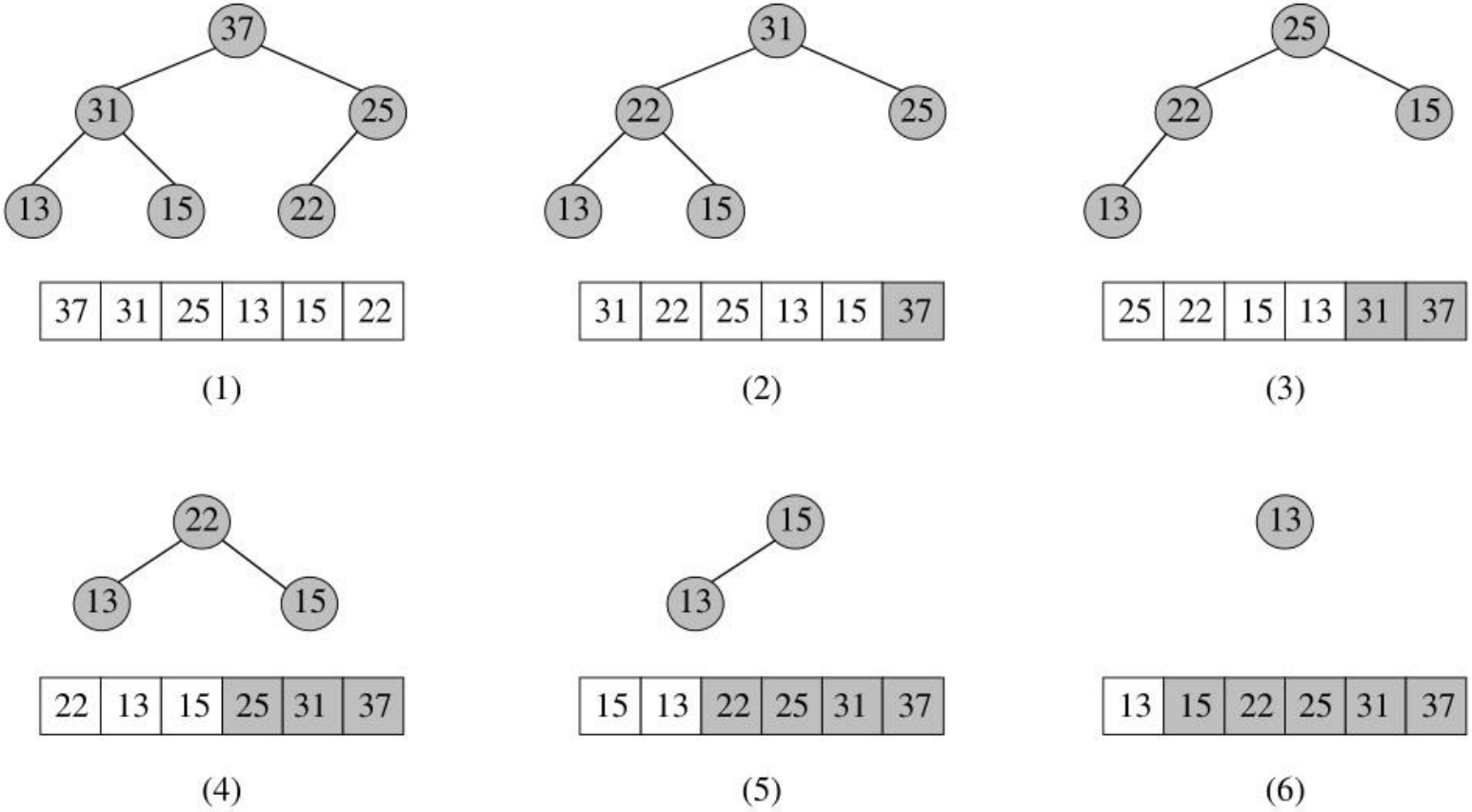
Ordinare *in loco* mediante heap

Domanda: come possiamo modificare l'algoritmo in modo che operi *in loco?*

Idea: ogni volta che il massimo viene estratto dall'heap, la foglia più a destra dell'albero viene spostata. Se rappresentiamo l'heap utilizzando un array, tale foglia coincide con l'ultima casella piena.

Quindi dopo ogni estrazione del massimo liberiamo una posizione in coda all'array che può essere utilizzata per conservare il valore appena estratto.

Ordinare *in loco* mediante heap



Costruzione dell'heap

Per costruire un heap partendo da un array di elementi, è stato presentato l'algoritmo heapify che in grado di trasformare un albero binario in un heap.

```
Algoritmo heapify(albero H)
    if (H è vuoto) then return
    else
        heapify(sottoalbero sinistro di H)
        heapify(sottoalbero destro di H)
        fixHeap(radice di H, H)
```

L'algoritmo è in grado di generare l'heap in tempo **O(n)**

Costruzione dell'heap

Esercizio: Implementare la nuova operazione `insert(H,k)` su MAX-HEAP, che inserisce nell'heap H il nuovo valore k mantenendo le proprietà di ordinamento.

Domanda-1: Quanto costa, nel caso peggiore, costruire un heap inserendo ripetutamente elementi?

Domanda-2: Usare questa costruzione modifica il tempo di esecuzione dell'algoritmo **heapSort** nel caso peggiore?

Heap: insert(H,k)

Dato un nodo **u**, indichiamo con **u.padre** il nodo **v** padre di **u**

```
Algoritmo insert(heap H, chiave k)
1. crea un nuovo nodo v e poni v.chiave ← k;
2. aggiungi v come foglia all'albero H;
3. while(v.padre != null && v.padre.chiave < v.chiave){
4.     scambia v con v.padre;
5. }
```

Il tempo richiesto nel caso peggiore dall'algoritmo `insert` è pari alla profondità dell'heap, ovvero **O(log n)**

Heap: insert(H,k)

Il tempo richiesto dall'algoritmo heapSort per ordinare un array di n elementi è di O(n log n).

```
Algoritmo heapSort(array S di dimensione n)
if (S è vuoto) then return S;
else {
    - costruisci un albero binario H con gli elementi di S;
    - ordinalo usando heapyfy(H);//costa O(n)
    - crea un coda C vuota.
    - while(H non è vuoto){        //cicla n volte
        C.enqueue(getMax(H));      //getMax ha costo O(log n)
    - copia gli elementi di C in S nel giusto ordine
    - return S
    }
```

Heap: insert(H,k)

Costruire un heap utilizzando l'operazione `insert` a partire da un array di **n** elementi richiede **O(n log n)** (*applico per **n** volte un operazione con costo **O(log n)***).

Dato che la costruzione deve essere eseguita una sola volta, usare questa costruzione dell'heap tramite inserimenti ripetuti **non** modifica il tempo di esecuzione dell'algoritmo **heapSort** nel caso peggiore.

Heap: insert(H,k)

```
Algoritmo heapSort(array S di dimensione n)
if (S è vuoto) then return S;
else {
    - crea un albero binario H vuoto;
    - while(S non è vuoto)
                estrai elemento v da S ed esegui insert(H,v);
        //creare l'heap attraverso inserimenti ripetuti
          costa O(n log n)
    - crea un coda C vuota.
    - while(H non è vuoto){        //cicla n volte
        C.enqueue(getMax(H));      //getMax ha costo O(log n)
    - copia gli elementi di C in S nel giusto ordine
    - return S
    }
```