heap binario

venerdì 9 agosto 2024 16:59

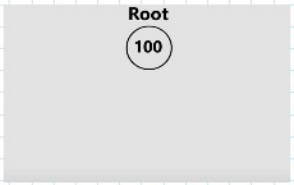
INSERT

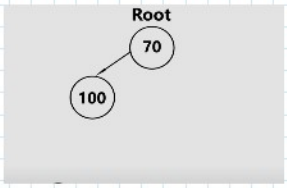
min-heap

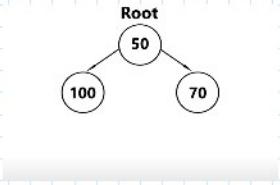
Quando inseriamo un elemento in un heap di tipo min, compariamo l'elemento appena inserito con il padre: se è minore del padre, swappiamo. Continuiamo il procedimento fino alla radice.

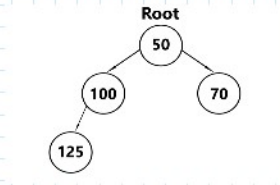
esempio:

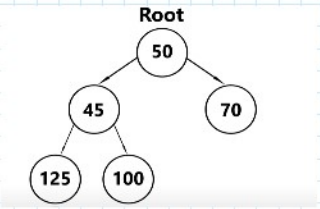
elementi da inserire: 100 70 50 125 45 60 10

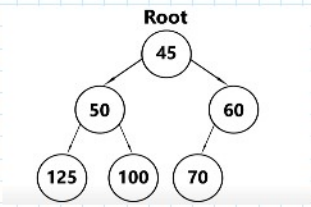
1) 

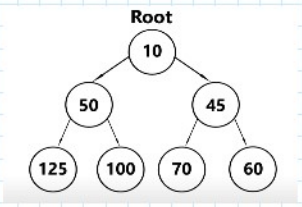
2) 

3) 

4) 

5) 

6) 

7) 

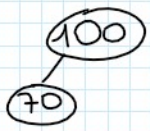
max-heap

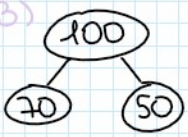
Quando inseriamo un elemento in un heap di tipo max, compariamo l'elemento appena inserito con il padre: se è maggiore del padre, swappiamo. Continuiamo il procedimento fino alla radice.

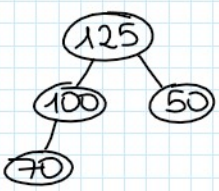
esempio:

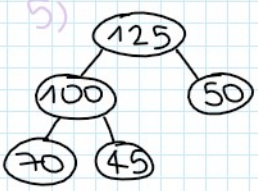
elementi da inserire: 100 70 50 125 45 60 10

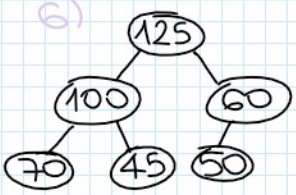
1) 

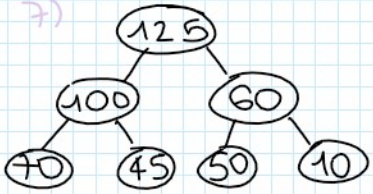
2) 

3) 

4) 

5) 

6) 

7) 

DELETE

Per eseguire la delete dobbiamo seguire questi passaggi:

1) salvare l'elemento nel nodo della radice
2) swappare il nodo della radice con l'ultimo nodo disponibile
3) Se l'heap è di tipo

min:
- se uno dei due figli è più piccolo del padre, swappiamo
- se entrambi i figli sono più piccoli del padre, swappiamo con il più piccolo dei due

max:
- se uno dei due figli è più grande del padre, swappiamo
- se entrambi i figli sono più grandi del padre, swappiamo con il più grande dei due

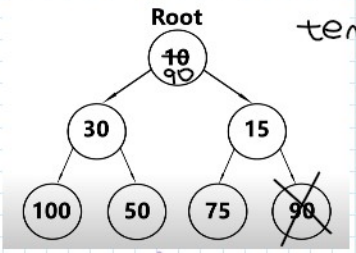
4) ritorniamo l'elemento salvato

- se entrambi i figli sono più grandi del padre, swappiamo con il più grande dei due

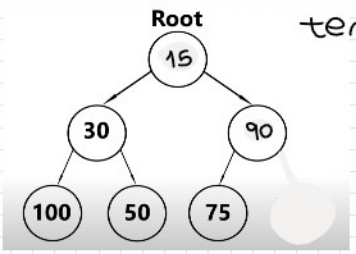
4) ritorniamo l'elemento salvato

esempio con min

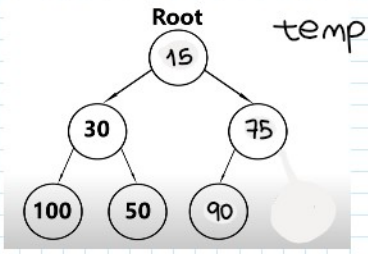
1)



2)

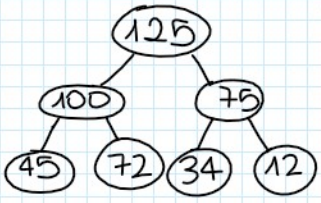


3)

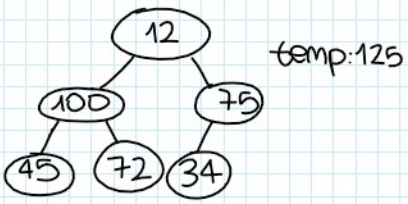


esempio con max

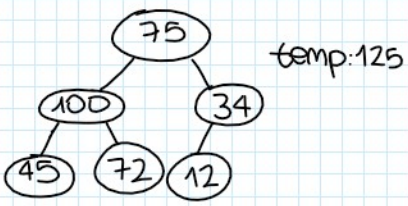
1)



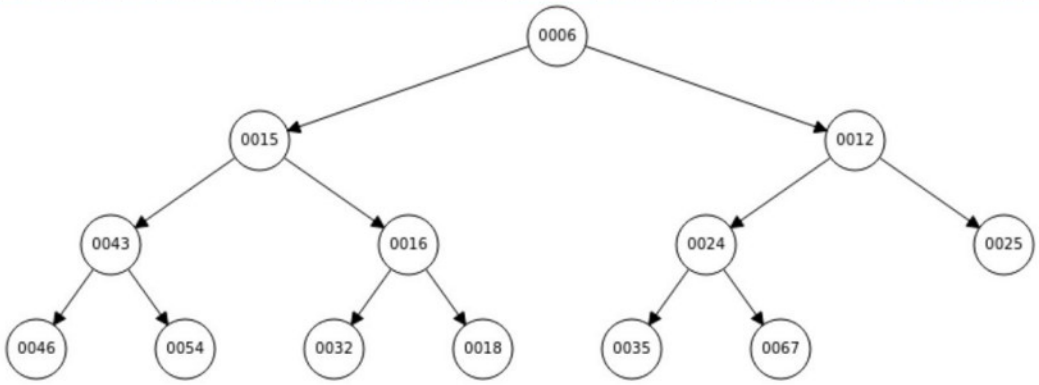
2)



3)



esercizio su aulaweb



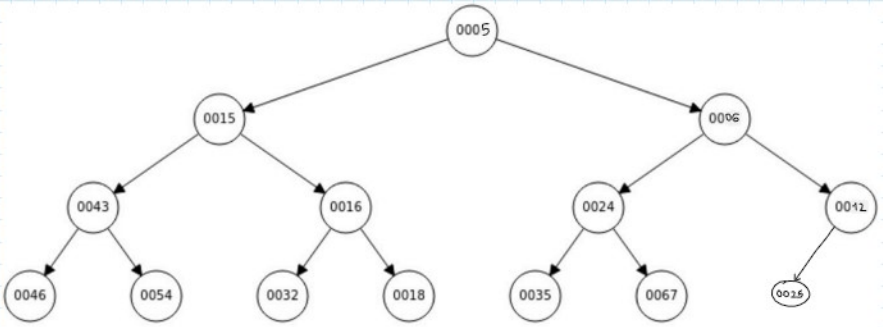
In uno Heap Binario di tipo "min" l**a radice ha chiave minima** e ogni nodo discendente di un nodo N deve avere **etichetta maggiore** di quella di N: le operazioni sono praticamente identiche a quelle di uno Heap Binario di tipo max come visto a lezione, invertendo "<" e ">".

Si consideri lo Heap Binario di tipo min disegnato sopra, che indicheremo con **A**.

➜ **[1/3 del punteggio]** Assumendo che le chiavi siano numeri interi, si disegni come viene modificato lo heap **A** dopo la seguente chiamata (senza fornire alcuna spiegazione dei passaggi: disegnate solo il risultato)

```
insert(5, "elem", A);
```

La chiamata `insert(5, "elem", A)` rappresenta il caso peggiore della operazione insert sullo heap **A**, rispetto alla complessità temporale? Motivare la risposta.



Il nuovo nodo viene aggiunto come foglia nel primo posto disponibile ossia come figlio sinistro di 25. Da lì parte la procedura che sistema i nodi, quindi il

posto disponibile ossia come figlio sinistro di 25. Da lì parte la procedura che sistema i nodi, quindi il nuovo nodo viene fatto "risalire" fino alla radice.

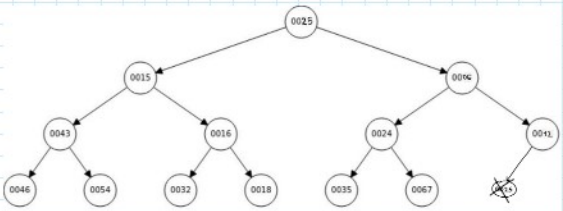
- Dato che dobbiamo risalire l'heap ricadiamo nel caso peggiore per l'inserimento che vale $\vartheta(\log n)$

[2/3 del punteggio] Si spieghino dettagliatamente, mediante disegni chiari e autoesplicativi, i passaggi principali della chiamata

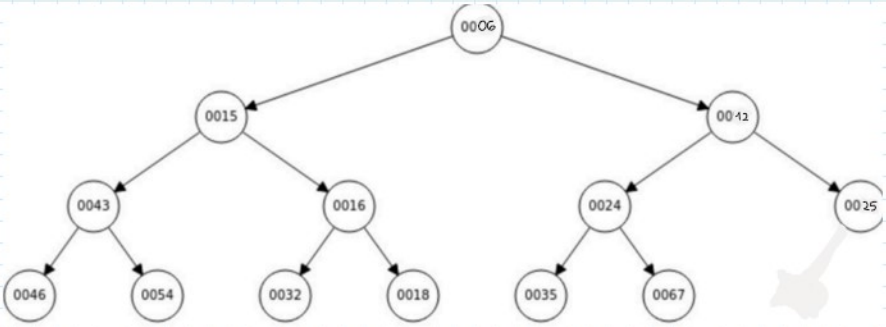
```
deleteMin(A);
```

effettuata sullo heap **A** modificato a seguito dell'inserimento dell'elemento con chiave `5`.

La chiamata `deleteMin(A)` rappresenta il caso peggiore della operazione deleteMin sullo heap A, rispetto alla complessità temporale? Motivare la risposta.



1) elimino nodo 0005 e swappo con l'ultimo (0025)
2) confronto 0025 con 0015 e 0006 e swappo con il minore
3) continuo a confrontare e swappare 0025



Ci troviamo nel caso peggiore in quanto l'elemento da eliminare si trova alla radice

complessità:

If a node is to be inserted at a level of height H:

Complexity of adding a node is: ***O(1)***

Complexity of swapping the nodes(upheapify): ***O(H)***
(swapping will be done H times in the worst case scenario)

Total complexity: ***O(1) + O(H) = O(H)***

For a Complete Binary tree, its height H = O(log N), where N represents total no. of nodes.

Therefore, Overall Complexity of insert operation is ***O(log N)***.

If a node is to be deleted from a heap with height H:

Complexity of swapping parent node and leaf node is: ***O(1)***

Complexity of swapping the nodes(downheapify): ***O(H)***
(swapping will be done H times in the worst case scenario)

Total complexity: ***O(1) + O(H) = O(H)***

For a Complete Binary tree, its height H = O(log N), where N represents total no. of nodes.

Therefore, Overall Complexity of delete operation is ***O(log N)***.

3. **Complexity of getting the Minimum value from min heap**

In order to obtain the minimum value just return the value of the root node (which is the smallest element in Min Heap), So simply return the element at index 0 of the array.

Hence, Complexity of getting minimum value is: ***O(1)***