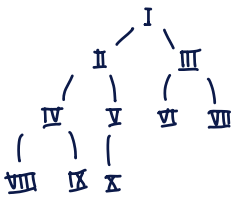
Heap

Definizione L'heap è un array che viene visto come un albero binario quasi completo.

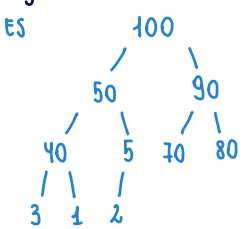


Caratteristiche dello heap

↳ heapsize(A) → dice quante caselle dell'array fanno parte dello heap e quante non ne fanno parte.
OSS heapsize(A) ≤ length(A)

↳ A[parent(i)] ≥ A[i] → ogni elemento dell'array è più grande dei suoi figli.

ES



↳ i → Parent(i) = $\lfloor \frac{i}{2} \rfloor$ (approx. per difetto)
↳ i → left(i) = 2*i
↳ i → right(i) = (2*i) + 1
↳ altezza di heap = log(n)

↳ max = root
Sugli altri elementi non sono certo
min = foglia

↳ n. foglie = $\lceil \frac{n}{2} \rceil$ (approx. per eccesso)

L'heap può essere utile per implementare:

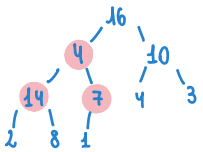
Definizione La coda con priorità è un metodo di implementazione dello heap dove il processo più importante è la radice. Verranno istemati i restanti elementi in modo da ottenere nuovamente il processo più importante alla radice.
L'ordine del resto non è importante.

metodi dello heap

array-heap
↑
heapify(A, h)
↓
posizione in cui voglio mettere l'heap

Condizione Quando lanciamo heapify, dobbiamo essere certi che a dx dell'elemento da sistemare ci sia uno heap e a sx dell'elemento da sistemare ci sia uno heap.

esempio



heapify(A, 2)

come funziona

I Cerco il valore più grande tra il nodo e i figli
II Scambio il valore più grande con il nodo
III Scambiando, può verificarsi che in uno dei due sottoalberi non sia più un heap. Faccio un altro heapify su quello scambiato.

Pseudocodice

```
heapify(A,h)
    largest = h;
    l = left(h);
    r = right(h);
    If A[l] > A[largest] AND l ≤ heapsize(A)
        largest = l;
    If A[r] > A[largest] AND r ≤ heapsize(A)
        largest = r;
    If largest ≠ h {
        APP = A[largest];
        A[largest] = A[h];
        A[h] = APP;
        heapify(A, largest);
    }
```

cerco chi è il più grande

se ho trovato un numero più grande, scambio nodo con il più grande lancio heapify per il nodo scambiato

Tempo di esecuzione

caso migliore : $\Omega(1)$

caso peggiore : lancio heapify sulla radice e devo scambiarlo fino alla foglia e l'albero non è bilanciato

$T(n) = T\left(\frac{2}{3}n\right) + \Theta(1)$ → teorema dell'esperto : $f(n) = \Theta(1)$

$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

$T(n) = \Theta(1 \cdot \log n)$

BuildHeap(A)

```
heapsize(A) = length(A);
For h = ⌊n/2⌋ down to 1
    heapify(A,h);
```

Tempo di esecuzione $O(n)$
↳ ordinamento veloce (con heapify sistemo l'ordine)

Osservazione
$\lceil \frac{n}{2} \rceil$ n. foglie = elementi su cui non devo lanciare heapify
$\lceil \frac{n}{2^2} \rceil$ n. nodi che hanno sotto foglie
⋮
$\lceil \frac{n}{2^{\ell+1}} \rceil$ n. elementi che distano ℓ dalle foglie

$$\Rightarrow T(n) = \frac{n}{2^{0+1}} O(\ell) + \frac{n}{2^{1+1}} O(1) + \frac{n}{2^{2+1}} O(2) + \dots$$

$$= \sum_{\ell=0}^{\log n} \frac{n}{2^{\ell+1}} O(\ell) = n \cdot O\left(\sum_{\ell=0}^{\log n} \frac{\ell}{2^\ell}\right) = n \cdot O\left(\sum_{\ell=1}^{\log n} \ell \cdot \left(\frac{1}{2}\right)^\ell\right) \leq n \cdot O\left(\sum_{\ell=1}^{\infty} \ell \left(\frac{1}{2}\right)^\ell\right)$$

$$= n \cdot O\left(\frac{1}{(1-\frac{1}{2})^2}\right) = n \cdot O(4) = O(n)$$

Heap sort

Definizione L'**heapsort** è un algoritmo di ordinamento che ha un tempo di $O(n \log n)$. Opera in loco ma non stabile. Si basa sulla struttura dati chiamata **heap**.

Come funziona
I Scambio nell'array la radice con l'ultimo elemento
II Stacco l'ultimo elemento, decrementando l'heapsize (rimane nell'array, ma non nel heap)
III lancio heapify sulla radice, risistemando di nuovo l'heap.
IV Scambio nello heap la radice con l'ultimo elemento (penultimo nell'array)
V Stacco l'ultimo elemento, decrementando l'heapsize (rimane nell'array, ma non nel heap)
VI Ripeto finché non arrivo alla radice

Pseudocodice

```
heapSort(A)
O(n)      BuildHeap(A);
          For i = 1 to length(A) {
              app = A[1];
              A[1] = A[heapsize(A)];
              A[heapsize(A)] = app;
              heapsize(A)--;
n·O(log n)    heapify(A,1);
          }
```

Osservazione Un algoritmo di ordinamento basato su i confronti non può impiegarci meno di $n \log n$

Tempo di ordinamento $T(n) = O(n \log n)$

Code con priorità

Definizione La coda con priorità è una struttura che ordina in base alla loro importanza, dove l'elemento più importante sta nella radice.
valore + importante : max → maxHeap
valore + importante : min → minHeap

metodi
Max(H) → dice cosa c'è in 1° posizione ⇝ $\Theta(1)$
ExtractMax(H) → scambio 1° con ultimo + decremento heapsize + faccio heapify su radice + ritorno l'elemento in heapsize + 1 ⇝ $O(\log n)$
Insert(H, k) → inserisco l'elemento come foglia + farlo risalire tramite parent ⇝ $O(\log n)$