







5° lezione algoritmi

	Ⓛ Luigi Domenico Castano
 Due Date	@October 15, 2024
 Materia	Algoritmi e Laboratorio
 Status	In progress

Ripasso lezione precedente

- Minheap: priorità numeri piccoli
- Maxheap: priorità numeri grandi

Ogni nodo ha sempre proprietà più al/ta, uguale rispetto ai suoi figli. La maggior parte dell'operazioni impiega $O(\log_n)$ poiché è un albero bilanciato, tranne l'ultimo livello in cui i nodi si ammassano su un solo lato.

HEAP struttura dati efficiente

La sua rappresentazione fisica è un array.

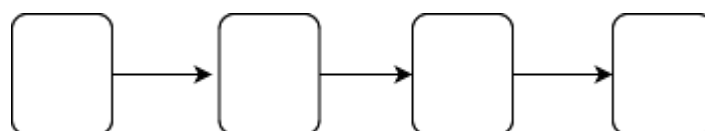
Il livello fisico è un array, mentre il livello astratto è un albero.

Caso diverso per l'array dove rappresentazione fisica e astratta corrispondono.

array

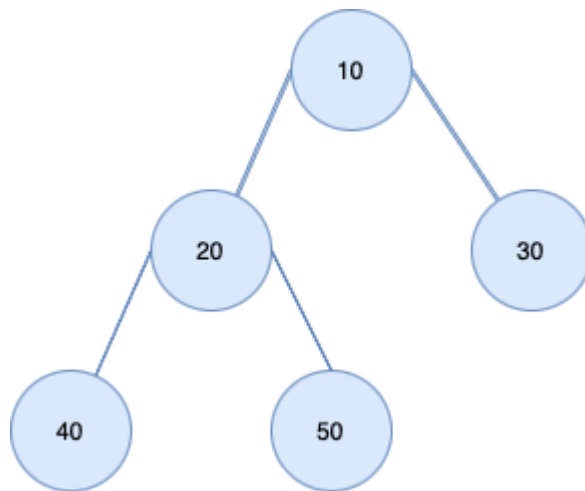


Lista (rappresentazione astratta)



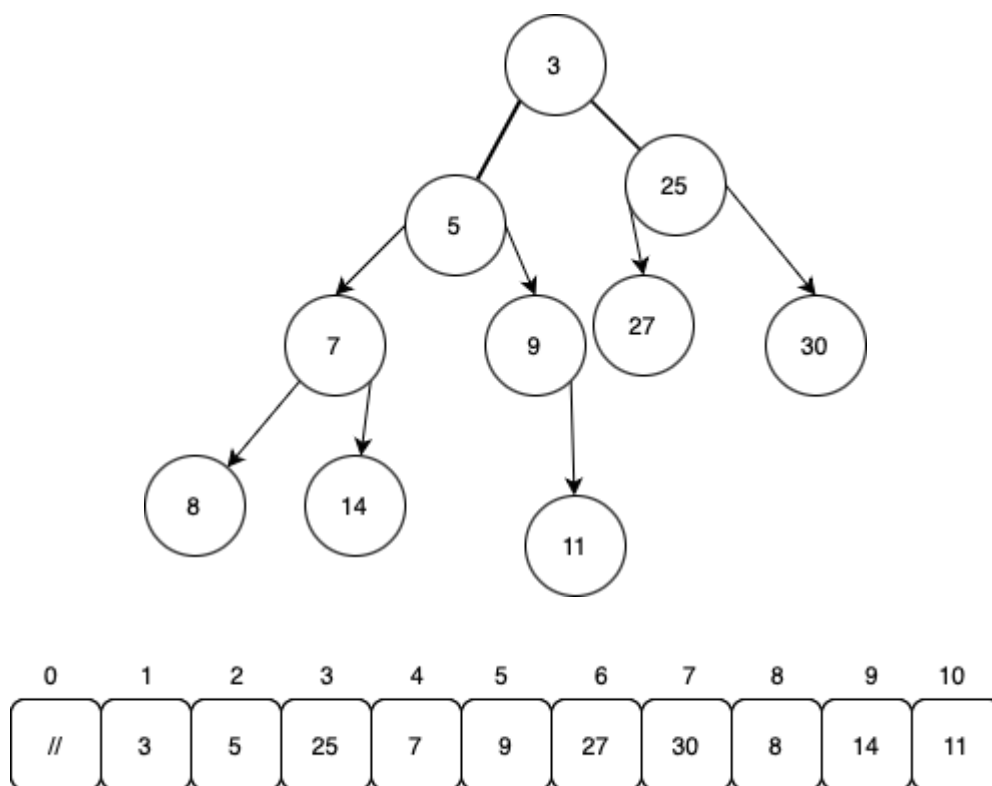
a livello fisico gli elementi non sono contigui ma sparsi

Come viene rappresentato l'HEAP a livello astratto?



Come viene rappresentato a livello fisico?

Viene rappresentato a livello fisico mediante un array



Ordinandolo in questa maniera, si perdono le informazioni che si aveva nel modo possibile per mantenerle sarebbe allocare un altro vettore dove si vanno a memorizzare l'informazione, dove si trova il figlio sinistro. Il figlio

destro segue il figlio sinistro, quindi per ottenere il figlio destro devo andare a $i + 1$. I figli dello stesso padre sono continui.

Trovare i figli con unico vettore

Con i puntatori

```
LEFT(X)
    return x->getLeft()
RIGHT(X)
    return x->getRight()
```

Con gli indici

```
LEFT(i)
    return 2*i
RIGHT(i)
    return 2*i+1
```

In questo modo, la lavorazione è veloce e gli elementi della rete sono continui. Conviene per questa operazioni fa iniziare l'indice da uno anziché da zero.

Trovare il parent

```
PARENT(i)
    return floor(i/2) [i/2]
```

Trovare i figli con la moltiplicazione BIT a BIT

```
LEFT(i)
    return (i << 1)

RIGHT(i)
    return (i << 1 | 1)
```

```
PARENT(i)
    return (i >> 1)
```

L' HEAP È quello che conosciamo delle astratto. Fisicamente, l'HEAP è un vettore in è possibile ricostruire la struttura astratta, grazie agli indici (con elementi ordinati da sinistra e a destra)

A questo punto possiamo dare una nuova definizione di HEAP

Un heap è un albero binario posizionale completo a meno dell'ultimo livello, a patto che tutti i suoi elementi sono allineati a sinistra. Fisicamente un HEAP è un vettore in cui tutti i nodi sono disposti in ordine da sinistra verso destra. Da questo vettore è possibile costruire una struttura ad albero grazie agli indici degli elementi all'interno del vettore.

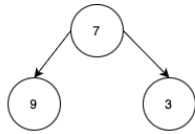
Differenza tra struttura fisica ed astratta di un HEAP

La struttura astratta non lineare è quella di un albero, ma quella fisica è di un array.

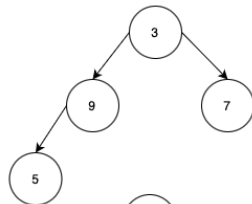
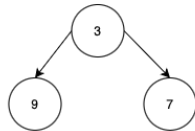
Come costruire un HEAP?

$S = \langle 7, 9, 3, 5, 4, 11, 17, 13, 2, 6 \rangle$

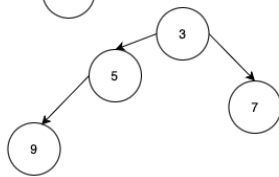
Partendo da un HEAP vuoto si vogliono inserire gli elementi di S.



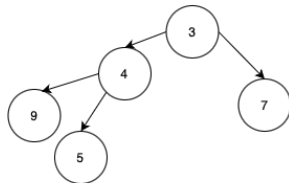
Si applica heapify() siccome si tratta di un min-heap si calcola i min fra 9 e 3 (dx e sx) e il 3 si sposta sopra e il 7 va a posto del 3



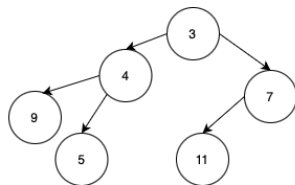
si applica heapify() min tra 5 e 9 quindi il 9 scende di 5 sale



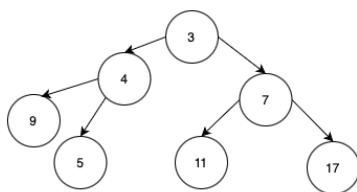
si controlla anche il 5 con il 3 ma rispetta il criterio del min quindi si passa al prossimo inserimento



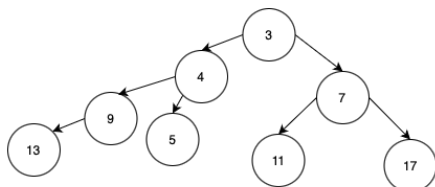
si passa inserire il 4 ma si deve scambiare con il 5 con la solita funzione



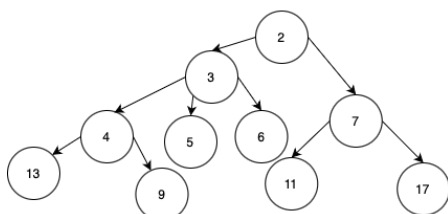
si inserisce 11 si controlla con 7 ma rispetta il criterio quindi si passa ad inserire il prossimo elemento



si inserisce il 17 e si confronta con 11 ma siccome rispetta il criterio si la lascia invariato. si controlla anche 11 con il 7 ma per il medesimo motivo non viene fatto nessuno scambio

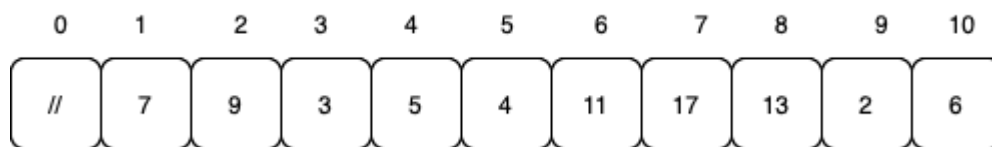


si inserisce il 13 e si confronta con il 9 ma siccome rispetta il criterio si la lascia invariato. si procede in questa maniera per i livelli precedenti



si inserisce il 2 e si confronta con il 9 si scambiano il due su scambia con il 4 il due si scambia con il 3

Come abbiamo detto prima l'heap si può rappresentare anche mediante un array che coincide con la rappresentazione fisica cioè in memoria.



Sicuramente non si ha un HEAP ci saranno sicuramente dei sotto-alberi che lo sono.

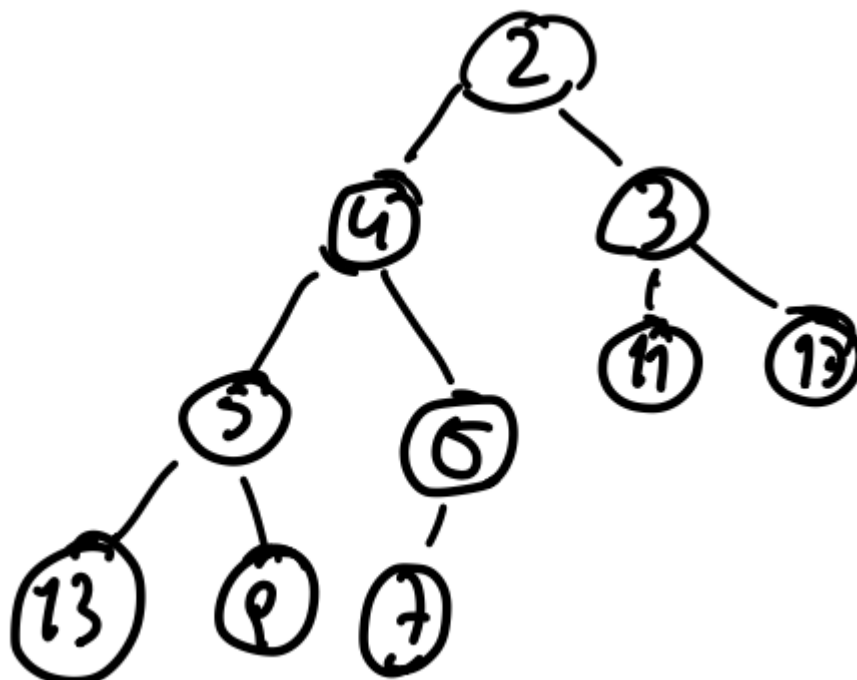
Le foglie sono sicuramente HEAP. Le foglie sono la metà dei nodi.

La procedura `heapfy()` si chiama sui nodi che non sono foglie. Cioè partendo dal livello prima delle foglie.

Inizio a percorrere foglie da sinistra verso destra trovando il modo in cui applicare la procedura `heapfy()`. $\frac{n}{2}$ è Primo elemento, a non essere una foglia nel vettore.

```
BULD_MIN_HEAP(A, N)
  FOR i <- floor(n/2) DOWN TO 1 DO
    HEAPFY(A, i)
```

Questa procedura ha complessità $O(n)$.



Se si costruisce un heap inserendo ogni singolo elemento
 $\rightarrow O(n \log n)$

Se si costruisce un heap da un array $\rightarrow O(n)$

Si può notare come la disposizione degli elementi nell'heap sono disposti diversamente rispetto a quando si inserisce un singolo elemento.

```
HEAPFY(A, i)
  l <- left(i)
  r <- right(i)
  min <- righty(i)
  if(l <= HeapSize && A[l] < A[min]) then min <- l
  if(r <= HeapSize && A[r] < A[min]) then min <- r
  if (min != i) then {swap(A, i, min); Heapfy(A, min);}
```

nel vettore che si sta considerando si hanno due variabili che indicano quanti elementi ci sono nel vettore (`heapSize`) e la lunghezza effettiva del vettore (`N`).

Inserimento

Se con un heap c'era il problema di capire come far andare il nuovo elemento al suo posto con un array è molto più semplice capirlo perché se si dovesse inserire un nuovo elemento questo sarà nella posizione `heapSize + 1`

```
INSERT(A, K)
  A[heapSize + 1] <- K
  heapSize += 1
  i <- heapSize elemento appena inserito
  P <- parent(i) padre dell' nuovo elemento
  WHILE( P != 0 && A[P] > A[i]) ciclo per inserire l'elemento
    SWAP(A, i, P)
    i <- P
    P <- PARENT(i)
```

EXTRACT-MIN

```
EXTRACT-MIN(A)
  SWAP(A, 1, heapSize)
  heapSize -= 1
  HEAPFY(A, 1)
```

Algoritmi di ordinamento con l'HEAP

0	1	2	3	4	5	6	7	8	9	10
//	7	9	3	5	4	11	17	13	2	6

E come il Selection sort ma con complessità $O(n \log n)$

Si sta usando una struttura dati per creare un algoritmo ordinamento efficienti e con soluzioni banali.

```
HEAPSORT(A, N)
  BUILD-MAX-HEAP(A, N)  $O(n)$ 
  FOR i <- 1 to n - 1
    EXTRACT-MAX(A)  $O(\log n)$ 
```

L'HEAP SORT lavora in loco e implementa l'albero direttamente dall' array. Si potrebbe implementare anche in maniera stabile. é migliore del Merge-Sort