
L'ADT coda con priorit 

Alcune implementazioni

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

L'ADT Coda con Priorit  (1/2)

tipo CodaPriorita:

dati:

un insieme S di n elementi di tipo *elem* a cui sono associate chiavi di tipo *chiave* prese da un universo totalmente ordinato.

operazioni:

findMin() \rightarrow *elem*

restituisce l'elemento in S con la chiave minima.

insert(*elem* e , *chiave* k)

aggiunge a S un nuovo elemento e con chiave k .

delete(*elem* e)

cancella da S l'elemento e .

deleteMin()

cancella da S l'elemento con chiave minima.

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

L'ADT Coda con Priorit  (2/2)

increaseKey(*elem* e , *chiave* d)

incrementa della quantit  d la chiave dell'elemento e in S .

decreaseKey(*elem* e , *chiave* d)

decrementa della quantit  d la chiave dell'elemento e in S .

merge(CodaPriorita c_1 , CodaPriorita c_2) \rightarrow CodaPriorita

restituisce una nuova coda con priorit  $c_3 = c_1 \cup c_2$.

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

Diverse possibili implementazioni

- ⇒ alberi di ricerca bilanciati (ad es. AVL, red-black)
- ⇒ heap binari (o 2-heap), visti per l'ordinamento
- ⇒ d-heap: generalizzazione degli heap binari
- ⇒ heap binomiali
- ⇒ heap di Fibonacci

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

Alberi di ricerca

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

Esercizio (non di laboratorio)

- Aggiungete al codice astratto della classe `AlberoAVL` (presentato in una delle lezioni precedenti) un metodo `searchMin()` che restituisce il minimo elemento contenuto nell'albero e un metodo `deleteMin()` che cancella dall'albero il minimo elemento (sia `AlberoAVL` il nome della classe ottenuta).
- Calcolate la complessità in tempo e spazio delle operazioni `searchMin()` e `deleteMin()`.
- Scrivete il codice astratto di una classe `CodaPrioritàConAlberoAVL` che implementi l'ADT `CodaPriorità` usando la classe `AlberoAVL`.
- Calcolate la complessità in tempo e spazio delle operazioni della classe `CodaPrioritàConAlberoAVL`.

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

Heap binari (o 2-heap)

F. Damiani - Alg. & Lab. 04/05

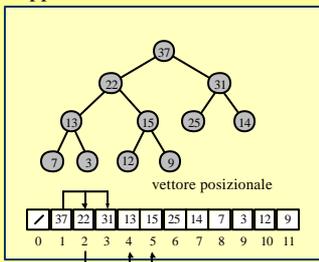
Heap

- **Struttura dati heap** associata ad un insieme $S =$ albero binario radicato con le seguenti proprietà:
 1. completo fino al penultimo livello
 2. gli elementi di S sono memorizzati nei nodi dell'albero (**chiave(v)** denota l'elemento memorizzato nel nodo v)
 3. **chiave(padre(v)) = chiave(v)** per ogni nodo v diverso dalla radice

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

La struttura dati heap

Rappresentazione ad albero e con vettore posizionale



$$\text{sin}(i) = 2i$$
$$\text{des}(i) = 2i+1$$
$$\text{parent}(i) = \lfloor i/2 \rfloor$$

Se le foglie nell'ultimo livello sono compattate a sinistra (**heap a struttura rafforzata**), il vettore posizionale ha esattamente dimensione n

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

Proprietà salienti degli heap

- 1) Il **massimo** è contenuto **nella radice**
- 2) L'albero ha **altezza $O(\log n)$**
- 3) Gli heap con struttura rafforzata possono essere rappresentati in un **array di dimensione pari a n**

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

La procedura `fixHeap`

Se tutti i nodi di H tranne v soddisfano la proprietà di ordinamento a heap, possiamo ripristinarla come segue:

`fixHeap`(nodo v , heap H)

if (v è una foglia) then return

else

 sia u il figlio di v con chiave massima

 if ($\text{chiave}(v) < \text{chiave}(u)$) then

 scambia $\text{chiave}(v)$ e $\text{chiave}(u)$

`fixHeap`(u, H)

Tempo di esecuzione: $O(\log n)$

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

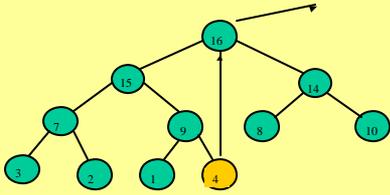
Estrazione del massimo

- Copia nella radice la chiave contenuta nella foglia più a destra dell'ultimo livello
- Rimuovi la foglia
- Ripristina la proprietà di ordinamento a heap richiamando `fixHeap` sulla radice

Tempo di esecuzione: $O(\log n)$

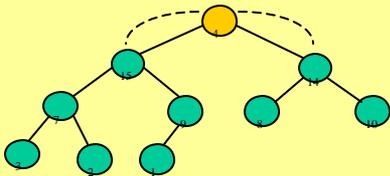
F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

Esempio di estrazione del massimo (1)



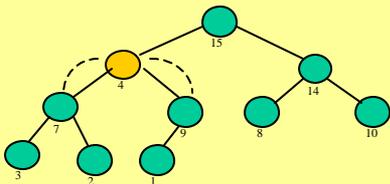
F. Damiani - Alg. & Lab. 04/05

Esempio di estrazione del massimo (2)



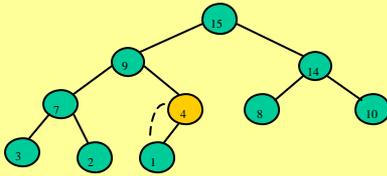
F. Damiani - Alg. & Lab. 04/05

Esempio di estrazione del massimo (3)



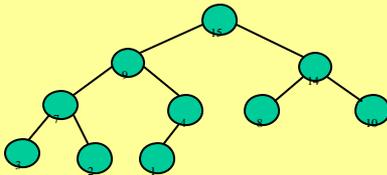
F. Damiani - Alg. & Lab. 04/05

Esempio di estrazione del massimo (4)



F. Damiani - Alg. & Lab. 04/05

Esempio di estrazione del massimo (5)



F. Damiani - Alg. & Lab. 04/05

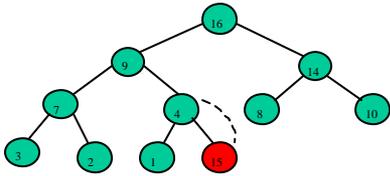
Inserimento di un nuovo elemento

- Aggiungi l'elemento come chiave di una nuova foglia, la foglia più a destra dell'ultimo livello
- Ripristina la proprietà di ordinamento a heap facendo risalire la chiave di tale foglia verso l'alto (fino a quando non raggiunge la radice o un nodo in cui la chiave del padre e' maggiore)

Tempo di esecuzione: $O(\log n)$

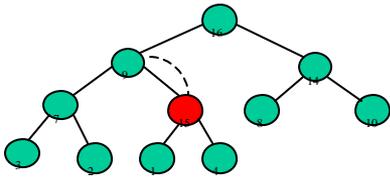
F. Damiani - Alg. & Lab. 04/05

Esempio di inserimento (1)



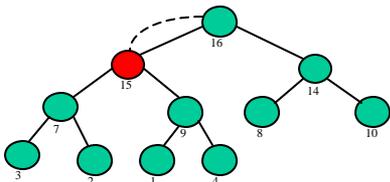
F. Damiani - Alg. & Lab. 04/05

Esempio di inserimento (2)



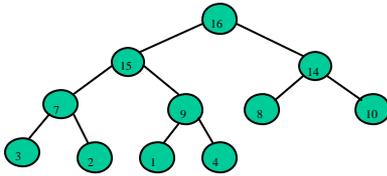
F. Damiani - Alg. & Lab. 04/05

Esempio di inserimento (3)



F. Damiani - Alg. & Lab. 04/05

Esempio di inserimento (4)



F. Damiani - Alg. & Lab. 04/05

Esercizio (non di laboratorio)

- Modificate la struttura dati **Heap** in modo che la radice contenga in minimo elemento (e non il massimo).
- Scrivete il codice astratto di una classe **Heap** che implementi la struttura dati così modificata.
- Scrivete il codice astratto di una classe **CodaPrioritàConHeap** che implementi l'ADT **CodaPriorità** usando la classe **Heap**.
- Calcolate la complessità in tempo e spazio delle operazioni della classe **CodaPrioritàHeap**.

F. Damiani - Alg. & Lab. 04/05

d-heap

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

Definizione

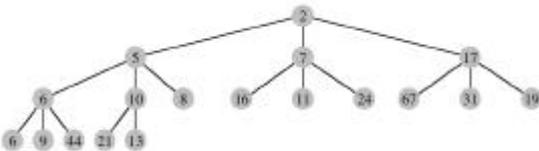
Un d-heap è un albero radicato d-ario con le seguenti proprietà:

1. **Struttura:** è completo almeno fino al penultimo livello
2. **Contenuto informativo:** ogni nodo v contiene un elemento $\text{elem}(v)$ ed una chiave $\text{chiave}(v)$ presa da un dominio totalmente ordinato
3. **Ordinamento a heap:** $\text{chiave}(v) = \text{chiave}(\text{parent}(v))$ per ogni nodo v diverso dalla radice

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

Esempio

Heap d-ario con 18 nodi e $d=3$



F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

Proprietà

1. Un d-heap con n nodi ha altezza $O(\log_d n)$
2. La radice contiene l'elemento con chiave minima (per via della proprietà di ordinamento a heap)
3. Può essere rappresentato implicitamente tramite vettore posizionale grazie alla proprietà di struttura

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

Procedure ausiliarie

Utili per ripristinare la proprietà di ordinamento a heap su un nodo v che non la soddisfi

$T(n) = O(\log_d n)$

```
procedura muoviAlto( $v$ )
  while (  $v \neq radice(T)$  and  $chiave(v) < chiave(padre(v))$  ) do
    scambia di posto  $v$  e  $padre(v)$  in  $T$ 
```

$T(n) = O(d \log_d n)$

```
procedura muoviBasso( $v$ )
  repeat
    sia  $u$  il figlio di  $v$  con la minima  $chiave(u)$ , se esiste
    if (  $v$  non ha figli o  $chiave(v) \leq chiave(u)$  ) break
    scambia di posto  $v$  e  $u$  in  $T$ 
```

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

findMin

$findMin() \rightarrow elem$
restituisce l'elemento nella radice di T .

$T(n) = O(1)$

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

insert(elem e, chiave k)

crea un nuovo nodo v con elemento e e chiave k , in modo che diventi una foglia sull'ultimo livello di T . La proprietà dell'ordinamento a heap viene poi ripristinata spingendo il nodo v verso l'alto tramite ripetuti scambi di nodi.

$T(n) = O(\log_d n)$ per l'esecuzione di $muoviAlto$

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

delete(elem e) e deleteMin

scambia il nodo v contenente l'elemento e con una qualunque foglia u sull'ultimo livello di T , e poi elimina v . Ripristina infine la proprietà dell'ordinamento a heap spingendo il nodo u verso la sua posizione corretta scambiandolo ripetutamente con il proprio padre o con il proprio figlio contenente la chiave più piccola

$T(n) = O(d \log_d n)$ per l'esecuzione di muoviBasso

Può essere usata anche per implementare la cancellazione del minimo

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

decreaseKey(elem e, chiave d)

decrementa il valore della chiave nel nodo v contenente l'elemento e della quantità richiesta d . Ripristina poi la proprietà dell'ordinamento a heap spingendo il nodo v verso l'alto tramite ripetuti scambi di nodi.

$T(n) = O(\log_d n)$ per l'esecuzione di muoviAlto

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

increaseKey(elem e, chiave d)

aumenta il valore della chiave nel nodo contenente l'elemento e della quantità richiesta d . Ripristina poi la proprietà dell'ordinamento a heap spingendo il nodo v verso il basso tramite ripetuti scambi di nodi.

$T(n) = O(d \log_d n)$ per l'esecuzione di muoviBasso

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

d-heap: tabella riassuntiva

Complessità in tempo delle operazioni su DHeap nel caso peggiore:

	DHeap
findMin	$O(1)$
insert	$O(\log n)$
delete	$O(\log n)$
deleteMin	$O(\log n)$
increaseKey	$O(\log n)$
decreaseKey	$O(\log n)$
merge	$O(n)$

Altre strutture dati (si veda il capitolo 8 di [Dem]) permettono di realizzare l'operazione merge con complessità inferiore.

F. Damiani - Alg. & Lab. 04/05 (da C. Demetrescu et al - McGraw-Hill)

Esercizio (non di laboratorio)

- Scrivete il codice astratto dell'operazione heapify (vista per i 2-heap) per un d-heap.
- Scrivete il codice astratto di un metodo che esegua, in tempo lineare, l'operazione merge su un d-heap.
(Suggerimento: usate l'operazione heapify.)

F. Damiani - Alg. & Lab. 04/05
