Code con priorità

Moreno Marzolla
Dip. di Scienze dell'Informazione
Università di Bologna

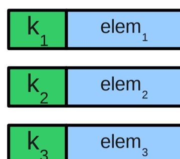
marzolla@cs.unibo.it
http://www.moreno.marzolla.name/

Copyright © 2009, 2010 Moreno Marzolla, Università di Bologna
(http://www.moreno.marzolla.name/teaching/ASD2010/)

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/3.0/ *or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.*

Coda con priorità

- Struttura dati che mantiene il minimo (o il massimo) in un insieme dinamico di chiavi su cui è definita una relazione d'ordine totale
  - Estensione naturale del min- (o max-)heap che abbiamo già trattato
- Una coda di priorità è un insieme di *n* elementi di tipo *elem* cui sono associate chiavi

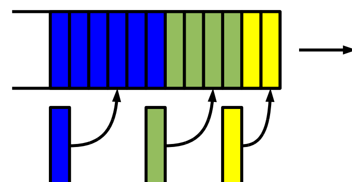


Operazioni

- findMin() → elem
  - Restituisce un elemento associato alla chiave minima
- insert(elem e, chiave k)
  - Inserisce un nuovo elemento e con associata la chiave k
- delete(elem e)
  - Rimuove un elemento dalla coda
- deleteMin()
  - Rimuove un elemento associato alla chiave minima
- increaseKey(elem e, chiave d)
  - Incrementa la chiave dell'elemento e della quantità d
- decreaseKey(elem e, chiave d)
  - Decrementa la chiave dell'elemento e della quantità d
- merge(CodaPri A, CodaPri B) → CodaPri
  - Fonde due code di priorità A e B, restituendo la loro unione

Applicazioni / 1

- Gestione della banda di trasmissione
  - Nel routing di pacchetti in reti di comunicazione è importante processare per primi i pacchetti con priorità più alta (ad esempio, quelli associati ad applicazioni con vincoli di real-time: VoIP, videoconferenza...)
  - I pacchetti in ingresso possono essere mantenuti in una coda di priorità per processare per primi quelli più importanti



Applicazioni / 2

- Simulazione discreta orientata ad eventi
  - Il sistema da simulare è composto da una serie di “eventi”
  - Un “evento” modifica lo stato del sistema, e può definire o cancellare altri eventi previsti per il futuro
  - Ciascun evento è associato ad un timestamp, che rappresenta il tempo (simulato) in cui l'evento viene eseguito
  - Il simulatore processa tutti gli eventi in ordine non decrescente di timestamp

Domanda

- La struttura dati “coda di priorità” può essere utilizzata per implementare uno stack “convenzionale” (politica LIFO), e una coda “convenzionale” (politica FIFO). Come?

Due possibili implementazioni

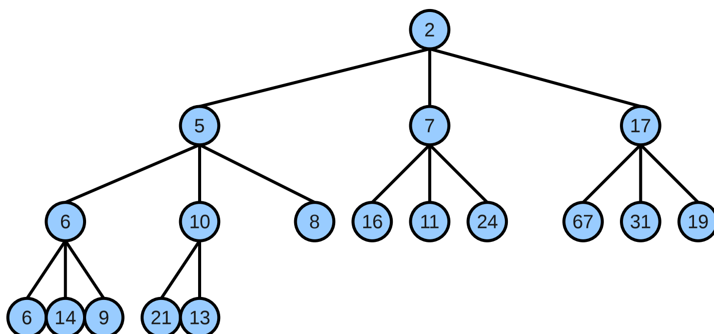
- d-heap
- Heap binomiali

d-heap

d-heap

- Estendono “naturalmente” il concetto di min/max-heap binario già visto
  - Uno heap binario era modellato su un albero binario
  - Un d-heap è modellato su un albero d-ario
- **Definizione**: un d-heap è un albero d-ario con le seguenti proprietà
  - un d-heap di altezza h è completo almeno fino alla profondità h-1
  - ciascun nodo *v* contiene una chiave *chiave(v)* e un elemento *elem(v)*. Le chiavi appartengono ad un dominio totalmente ordinato
  - ogni nodo diverso dalla radice ha chiave non inferiore (≥) a quella del padre

Esempio
d-heap con d=3



Altezza di un d-heap

- Un d-heap con n nodi ha altezza $O(\log_d n)$
  - Sia h l'altezza di un d-heap con n nodi
  - Il d-heap è completo fino al livello h-1
  - Un albero d-ario completo di altezza h-1 ha

$$\sum_{i=0}^{h-1} d^i = \frac{d^h - 1}{d-1}$$

    nodi
  - Quindi $\frac{d^h - 1}{d-1} < n$

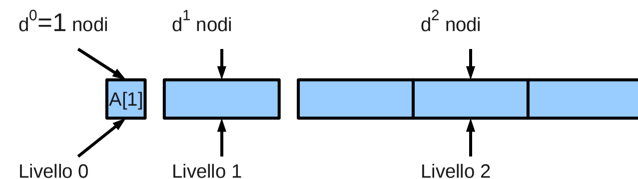
$$d^h < n(d-1)+1$$

$$h < \log_d(n(d-1)+1) = O(\log_d n)$$

Domande

- Supponiamo che un d-heap di n elementi sia memorizzato per livelli in un vettore A[1]...A[n] (esattamente come gli heap binari visti in precedenza).
  - La radice (che si trova a livello zero nell'albero d-ario) è memorizzata in A[1]
  - In quale posizione è memorizzato il primo elemento del livello h?
  - In quale posizione è memorizzato l'ultimo elemento del livello h? (si supponga che il livello h sia completo)

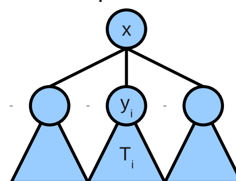
Risposta



- Il livello h inizia da $1+\sum_{k=0}^{h-1} d^k = 1+\frac{d^h-1}{d-1}$
- Il livello h termina in $d^h+\sum_{k=0}^{h-1} d^k = d^h+\frac{d^h-1}{d-1}$
- Nota: dato un elemento in posizione i, il padre si trova in posizione $\lceil (i-1)/d \rceil$

Proprietà fondamentale dei d-heap

- La radice contiene un elemento con chiave minima
- Dimostrazione: per induzione sul numero di nodi
  - Per n=0 (heap vuoto) oppure n=1 la proprietà vale
  - Supponiamo sia valida per ogni d-heap con al più n-1 nodi
  - Consideriamo un d-heap con n nodi. I sottoalberi radicati nei figli della radice sono a loro volta d-heap, con al più n-1 nodi
  - La radice di T_i contiene il minimo di T_i
  - La chiave radice x è ≤ della chiave in ciascun figlio
  - Quindi la chiave in x è il minimo dell'intero heap



Operazioni ausiliarie

Costo: O(h)

```
procedura muoviAlto(v)
   while( v != root(T) and
         chiave(v)<chiave(padre(v)) ) do
      scambia di posto v e padre(v) in T;
      v := padre(v);
   endwhile
```

Costo: O(dh)

Costo: O(d)

```
procedura muoviBasso(v)
   repeat forever
      if ( v non ha figli ) then
         return;
      else
         sia u il figlio di v con la minima chiave(u)
         if ( chiave(u) < chiave(v) ) then
            scambia di posto u e v;
            v := u;
         else
            return;
         endif
      endif
```

findMin() → elem

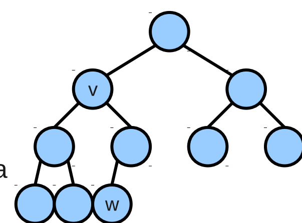
- Restituisce l'elemento associato alla radice dello heap
  - In base alla proprietà fondamentale dei d-heap, la radice è un elemento che ha chiave minima
- Costo complessivo: O(1)

insert(elem e, chiave k)

- Crea un nuovo nodo v con chiave *k* e valore *e*
- Aggiungi il nodo come ultima foglia a destra dell'ultimo livello
  - La proprietà di struttura è soddisfatta
- Per mantenere la proprietà di ordine, esegui muoviAlto(v) (che costa $O(\log_d n)$ nel caso peggiore)
- Costo complessivo: $O(\log_d n)$

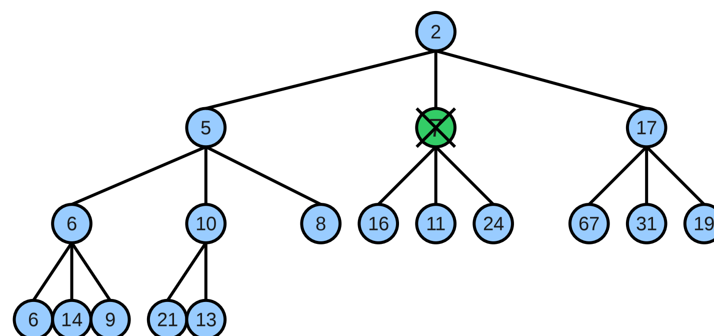
delete(elem e)
(e deleteMin())

- Sia v il nodo che contiene l'elem. e con chiave k
- Sia w l'ultima foglia a destra
  - Setta elem(v) := elem(w);
  - Setta chiave(v) := chiave(w);
  - Stacca e cancella w dallo heap
  - La proprietà di struttura è mantenuta
- Esegui muoviAlto(v)
  - costo $O(\log_d n)$
- Esegui muoviBasso(v)
  - costo $O(d \log_d n)$
- Costo complessivo: $O(d \log_d n)$

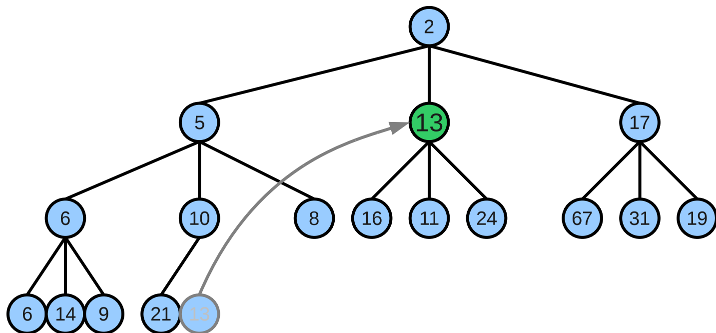


Nota: una sola tra queste operazioni viene eseguita: l'altra termina immediatamente

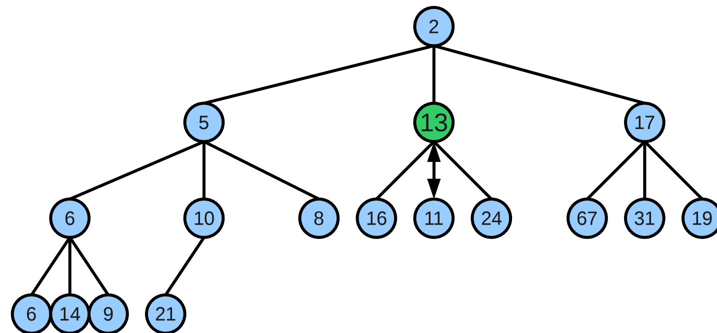
Esempio / 1



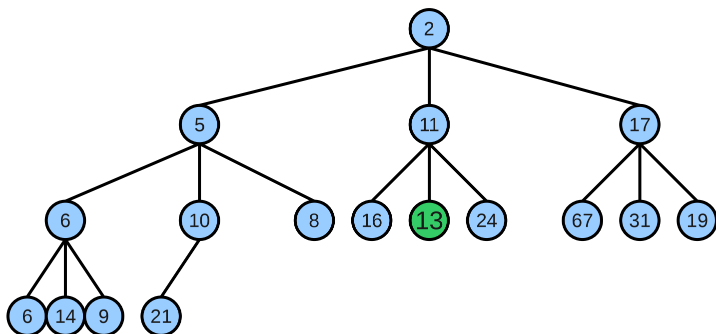
Esempio / 2



Esempio / 3



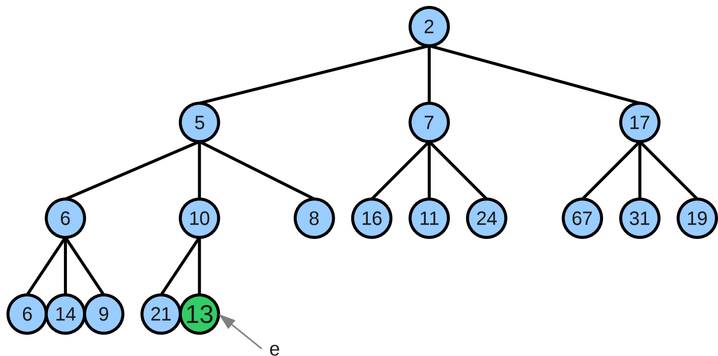
Esempio / 4



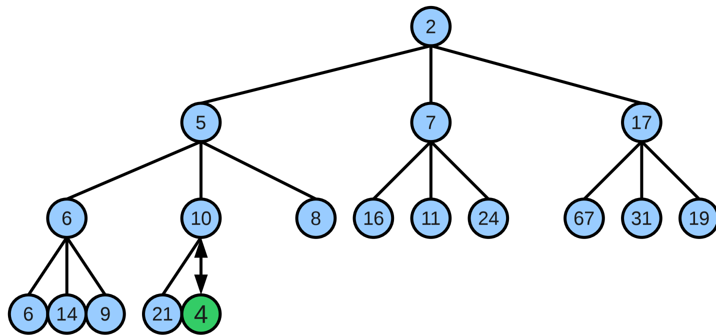
decreaseKey(elem e, chiave d)

- Sia v il nodo contenente e
- Setta chiave(v) := chiave(v) – d;
- Esegui muoviAlto(v)
  - Costo: $O(\log_d n)$
- Costo complessivo: $O(\log_d n)$

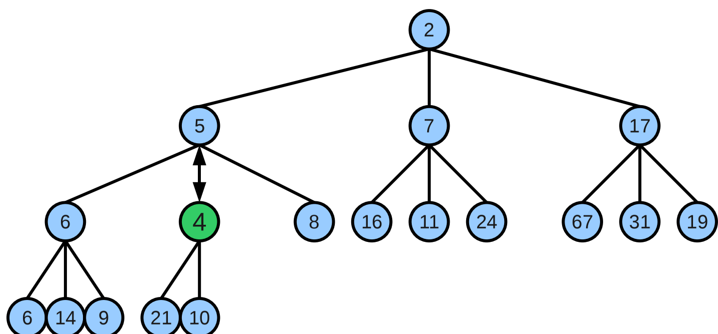
Esempio: decreaseKey(e, 9)



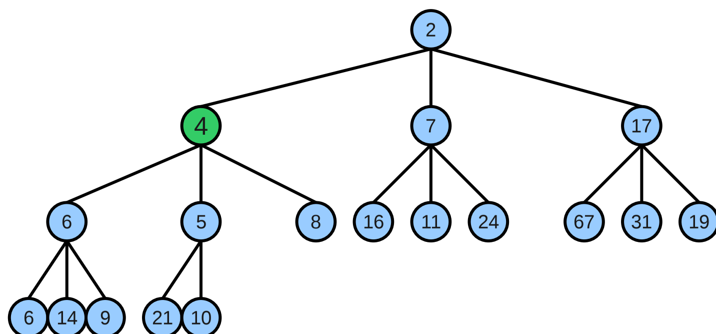
Esempio / 2



Esempio / 3



Esempio / 4



increaseKey(elem e, chiave d)

- Sia v il nodo contenente e
- Setta chiave(v) := chiave(v) + d;
- Esegui muoviBasso(v)
  - Costo: $O(d \log_d n)$
- Costo complessivo: $O(d \log_d n)$

merge(CodaPri L1, CodaPri L2) → CodaPri

- Non realizzabile in modo efficiente
  - **Domanda**: come è possibile realizzarla (anche se in modo relativamente inefficiente)? Quale è il costo computazionale?

Riepilogo costi per d-heap

- findMin() → elem — $O(1)$
- insert(elem e, chiave k) — $O(\log_d n)$
- delete(elem e) — $O(d \log_d n)$
- deleteMin() — $O(d \log_d n)$
- increaseKey(elem e, chiave d) — $O(d \log_d n)$
- decreaseKey(elem e, chiave d) — $O(\log_d n)$
- merge(CodaPri A, CodaPri B) → CodaPri
  - Non supportata