Heap

venerdì 12 febbraio 2021 14:02

CODE DI PRIORITÁ

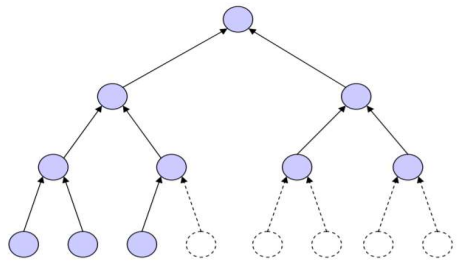
DEF: UNA CODA DI PRIORITÁ È UNA COLLEZIONE DI ELEMENTI A CUI SONO ASSOCIATI DEI VALORI DI PRIORITÁ. QUEST'ULTIMI DEFINISCONO UN ORDINAMENTO.

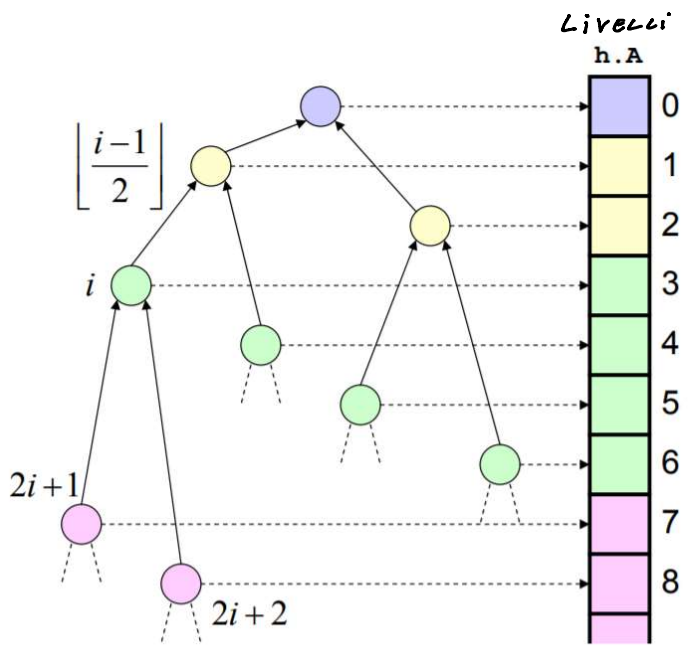
REALIZZAZIONI NON EFFICENTI:

1. CODA DI PRIORITÁ CON LISTA ORDINATA:
   - INSERIMENTO IN LISTA $\Theta(n)$ (CASO PEGGIORE SCORRO LA LISTA)
   - CANCELLAZIONE ELEMENTO CON PIÚ PRIORITÁ $\Theta(1)$ (CASO PEGGIORE → ELEMENTO IN TESTA = MAX PRIORITÁ)
2. CODA DI PRIORITÁ CON LISTA NON ORDINATA:
   - INSERIMENTO IN TESTA $\Theta(1)$
   - CANCELLAZIONE E RICERCA $\Theta(n)$ DOVENDO SCORRERE IN TUTTA LA LISTA

LA STRUTTURA DATI HEAP

DEF: È UN ARRAY IN CUI I VALORI SONO IN RAPPORTO CON LA LORO POSIZIONE NELL'ARRAY. GLI HEAP RAPPRESENTANO ALBERI QUASI COMPLETI, CIOÈ SE L'ULTIMO LIVELLO DELLE FOGLIE A DESTRA RIMANE INCOMPLETO



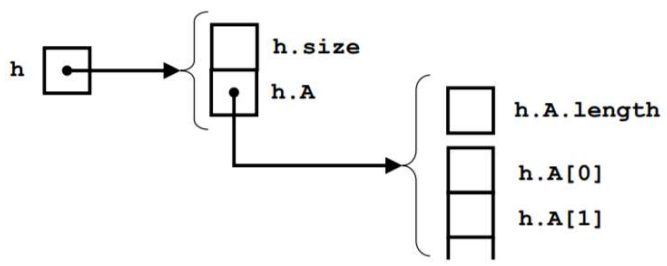


→ LA FORMULA $\left[\frac{i-1}{2}\right]$ SERVE PER TROVARE IL PADRE DEL NODO PRESO IN CAUSA (ARROTONDO IN BASSO)

→ LA FORMULA $2i+1$ SERVE INVECE PER TROVARE L'INDICE DEL FIGLIO SINISTRO

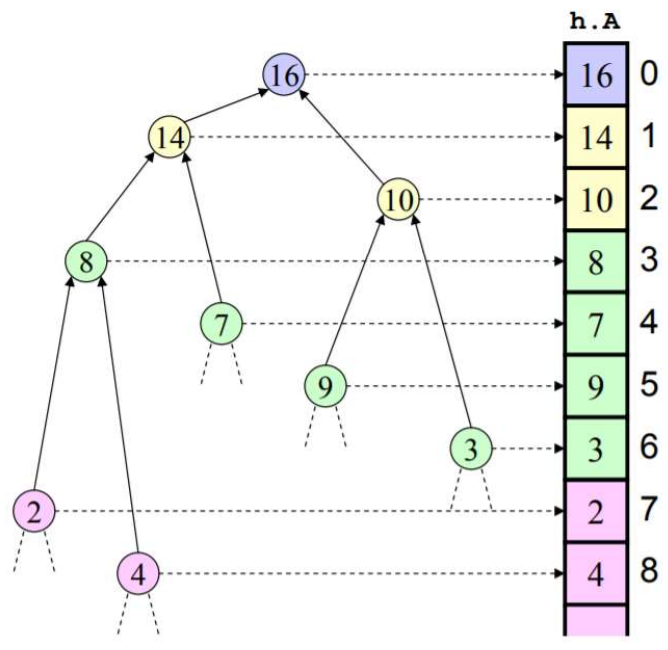
→ LA FORMULA $2i+2$ INVECE PER IL FIGLIO DESTRO

DETTAGLI STRUTTURALI



VALORI CONTENUTI IN UN MAX-HEAP

- L'ELEMENTO MEMORIZZATO NEL NODO i DEVE AVERE VALORE $\geq$ DEI SUOI FIGLI



} VALORE CON PRIORITÁ MASSIMA

→ ARRAY NON NECESSARIAMENTE ORDINATO

→ AVENDO IL NUMERO DI ELEMENTI DELL'ARRAY, GLI ELEMENTI DA 0 A $\left[\frac{n}{2}-1\right]$ SONO NODI INTERNI

(15 NODI? $\frac{15}{2}-1 = 6$ NODI INTERNI SENZA RADICE)

PROCEDURA MAX_HEAPIFY

- ABBIAMO 2 SOTTOALBERI (LEFT) E (RIGHT) CHE SONO DEI MAX-HEAP.

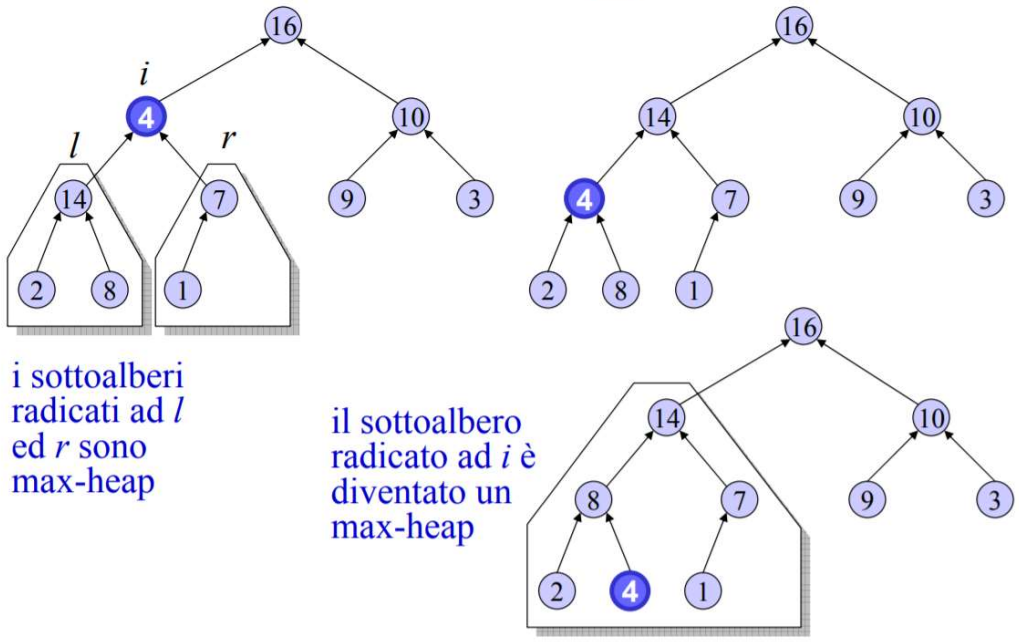
COME POSSIAMO UNIRE I DUE SOTTOALBERI PER AVERNE UNO SOLO CON LE STESSE REGOLE DEL MAX_HEAP

```
MAX_HEAPIFY(h,i)   COSTO Θ(1) E COSTO Θ(1·log n) = Θ(log n)
1.  l = LEFT(i)          ▷ indice del figlio sinistro
2.  r = RIGHT(i)         ▷ indice del figlio destro
3.  if (l ≤ h.size-1 and h.A[l] > h.A[i])    massimo = l
4.  else                                     massimo = i
5.  if (r ≤ h.size-1 and h.A[r] > h.A[massimo]  massimo = r
6.  /* ora massimo è il massimo tra h.A[l], h.A[r] ed h.A[i]
7.  if massimo ≠ i   → SE IL MAX NON ERA i, DEVO SCAMBIARE
8.      SCAMBIA_CASELLE(h.A,i,massimo)   SCAMBIO i CON MASSIMO
9.      MAX_HEAPIFY(h,massimo)   CONTROLLO NUOVAMENTE L'HEAP
```

→ SE IL VALORE DEL FIGLIO SINISTRO È MAGGIORE DEL VALORE IN i ALLORA METTI IL MASSIMO PARI AL FIGLIO SINISTRO, ALTRIMENTI RIMANE i.

→ SE IL VALORE DEL FIGLIO DESTRO È MAGGIORE DEL VALORE IN i ALLORA METTI IL MASSIMO PARI AL FIGLIO DESTRO, ALTRIMENTI RIMANE i.

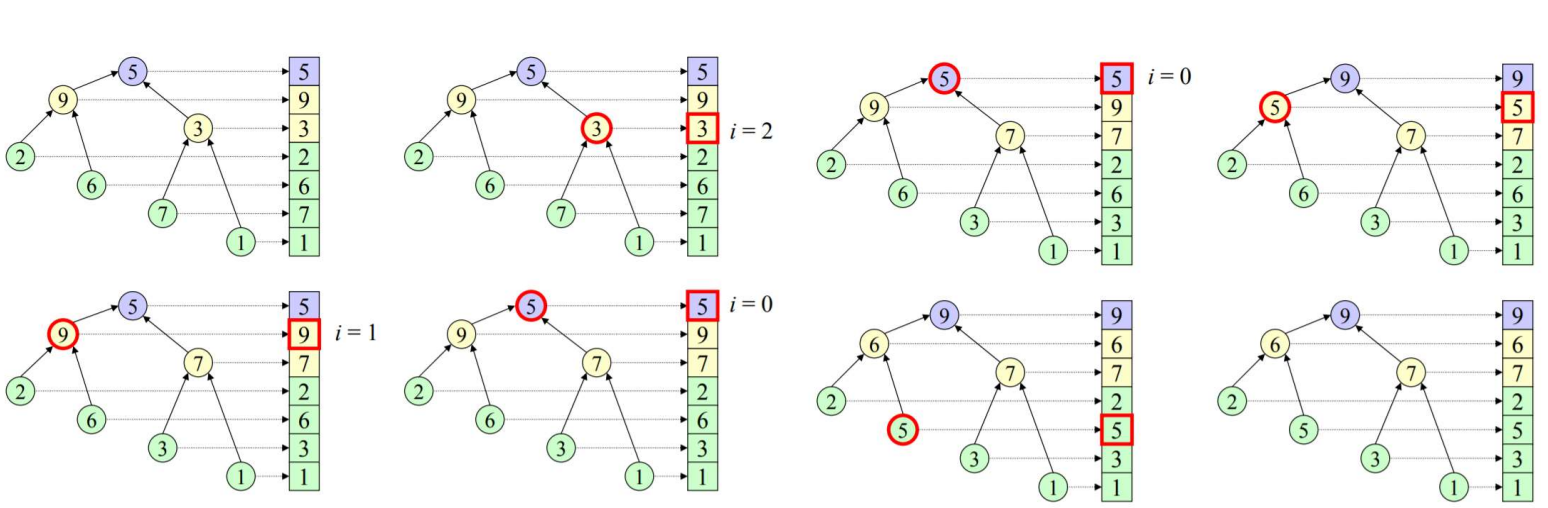
VEDIAMO L'ESECUZIONE



ALGORITMO BUILD MAX_HEAP Θ(n)

- SCOPO: TRASFORMARE UN ARRAY A IN UN HEAP
- LAVORA SOLTANTO SUI NODI INTERNI PERCHÉ MAX HEAP LO SI VA AD UTILIZZARE SOLTANTO SUI GENITORI

PROCEDURA:



PROCEDURA EXTRACT MAX

```
EXTRACT_MAX(h)
1.  if IS_EMPTY(h)
2.      error("heap underflow")
3.  max = h.A[0]
4.  h.A[0] = h.A[h.size - 1]
5.  h.size = h.size - 1
6.  MAX_HEAPIFY(h,0)
7.  return max
```

→ ANDIAMO A PRENDERE L'ULTIMO ELEM

```
INSERT(h,key)
```

```
1.  if h.size == h.length
2.      error("overflow")
3.  h.size = h.size + 1
4.  i = h.size - 1
5.  while i>0 and h.A[PARENT(i)] < key
5.      h.A[i] = h.A[PARENT(i)]
5.      /* il genitore di i è stato spostato in basso */
6.      i = PARENT(i)
9.  h.A[i] = key
```

```
HEAP_SORT(A)
1.  h.A = A    /* h è un nuovo heap */
2.  h.size = A.length
3.  BUILD_MAX_HEAP(h)
4.  for i = h.A.length-1 downto 1
5.      SCAMBIA-CASELLE(A,0,i)
5.      h.size = h.size - 1
7.      MAX-HEAPIFY(h,0)
```