ESERCIZIO DI ASD DEL 3 NOVEMBRE 2008

Modifica nella Heap

Versione Ricorsiva. Distinguiamo due casi:

- Se $k \geq 0$ effettuiamo una correzione verso l'alto, sfruttando la procedura Correggi vista a lezione.
- Se $k < 0$ effettuiamo una correzione verso il basso, sfruttando la procedura Heapify vista a lezione.

Algorithm 1 HeapModify(A, i, k)

```
1: A[i] ← A[i] + k
2: if k ≥ 0 then
3:     Correggi(A, i)
4: else
5:     Heapify(A, i)
6: end if
```

Algorithm 2 Correggi(A, i)

```
1: if i > 1 and A[i] > A[parent(i)] then
2:     Scambia(A, i, parent(i))
3:     Correggi(A, parent(i))
4: end if
```

Algorithm 3 Heapify(A, i)

```
1: ℓ ← left(i)
2: r ← right(i)
3: if ℓ ≤ heapsize[A] and A[ℓ] > A[i] then
4:     max ← ℓ
5: else
6:     max ← i
7: end if
8: if r ≤ heapsize[A] and A[r] > A[max] then
9:     max ← r
10: end if
11: if max ≠ i then
12:     Scambia(A, i, max)
13:     Heapify(A, max)
14: end if
```

Date: 3 Novembre 2008.

Versione Ricorsiva. Dobbiamo fornire una versione non ricorsiva delle due procedure CORREGGI ed HEAPIFY. Entrambe le procedure sono ricorsive di coda. La ricorsione si può facilmente rimpiazzare con un ciclo while. Introduciamo una variabile *loop* che indicherà quando il ciclo deve terminare. La variabile *loop* assume valore iniziale $True$. All'interno del ciclo while troviamo tutto il codice che avevamo nella versione ricorsiva. La chiamata ricorsiva viene sostituita con un'opportuna modifica del valore della variabile i. Aggiungiamo un caso else in cui alla variabile *loop* viene assegnato valore $False$. Questo corrisponde al caso in cui nella versione ricorsiva non venivano effettuate chiamate ricorsive.

Algorithm 4 NORICCORREGGI(A, i)

```
1: loop ← True
2: while loop do
3:    if i > 1 and A[i] > A[PARENT(i)] then
4:       SCAMBIA(A, i,PARENT(i))
5:       i ←PARENT(i))
6:    else
7:       loop ← False
8:    end if
9: end while
```

Algorithm 5 NORICHEAPIFY(A, i)

```
1: loop ← True
2: while loop do
3:    ℓ ←LEFT(i)
4:    r ←RIGHT(i)
5:    if ℓ ≤ heapsize[A] and A[ℓ] > A[i] then
6:       max ← ℓ
7:    else
8:       max ← i
9:    end if
10:   if r ≤ heapsize[A] and A[r] > A[max] then
11:      max ← r
12:   end if
13:   if max ≠ i then
14:      SCAMBIA(A, i, max)
15:      i ← max
16:   else
17:      loop ← False
18:   end if
19: end while
```

Nel caso della procedura CORREGGI è facile anche immaginare una versione non ricorsiva semplificata senza la variabile ausiliaria *loop*. Chiamiamo NORICCORREGGI2 la versione semplificata.

La versione non ricorsiva di HEAPMODIFY si ottiene semplicemente sostituendo le chiamate alle procedure CORREGGI e HEAPIFY con chiamate alle versioni

Algorithm 6 NoRicCorreggi2(A, i)

```
1: while i > 1 and A[i] > A[PARENT(i)] do
2:     SCAMBIA(A, i, PARENT(i))
3:     i ← PARENT(i))
4: end while
```

non ricorsive. In particolare nel resto della trattazione facciamo riferimento alla procedura NoRicCorreggi2 come versione non ricorsiva di Correggi.

Correttezza. Dimostriamo la correttezza di NoRicCorreggi2. Indichiamo con $brother(k)$ l'indice in cui si trova il fratello del nodo che ha indice k: se k è pari $brother(k)$ è $k+1$, se k è dispari $brother(k)$ è $k-1$.

Lemma 1 (while in NoRicCorreggi2)**.** *Se $A[k]$ e $A[brother(k)]$ sono radici di heap e $A[2k], A[2k+1], A[brother(k)] \leq A[parent(k)]$, allora dopo una esecuzione del ciclo while della procedura* NoRicCorreggi2 *con indice i uguale a k vale che $A[parent(k)]$ è radice di heap.*

Dimostrazione. Se $A[k] \leq A[parent(k)]$, allora il codice non fa niente ed infatti abbiamo che $A[parent(k)]$ è radice di heap visto che $A[k]$ ed $A[brother(k)]$ sono radici di heap e $A[k], A[brother(k)] \leq A[parent(k)]$.

Se $A[k] > A[parent(k)]$, allora siano $A[k] = y$, $A[parent(k)] = x$, $A[brother(k)] = z$. Il codice scambia x ed y. A questo punto x si trova in posizione k ed ha come figli $A[2k]$ e $A[2k+1]$ che sono radici di heap per ipotesi. Inoltre per ipotesi $A[2k], A[2k+1] \leq x$, quindi in posizione k abbiamo una heap. $A[brother(k)]$ era radice di heap e tale è rimasto, visto che non si sono toccati nodi che stanno nel suo sottoalbero. y ora si trova in posizione $parent(k)$ ed ha come figli z e x che sono due radici di heap e che soddisfano la proprietà $z \leq x < y$. Quindi in posizione $parent(k)$ abbiamo una heap. □

Invariante 1 (while in NoRicCorreggi2)**.** *Se all'inizio dell'esecuzione della procedura* NoRicCorreggi2(A, i) *A è una heap tranne al più l'errore $A[i] > A[parent(i)]$, allora all'inizio dell'iterazione del ciclo while in cui i vale k abbiamo che $A[k]$ e $A[brother(k)]$ sono radici di heap, $A[2k], A[2k+1], A[brother(k)] \leq A[parent(k)]$ e in A c'è al più l'errore $A[k] > A[parent(k)]$.*

Dimostrazione. Procediamo per induzione su k.

BASE. $k = i$.

Per ipotesi abbiamo che A è una heap tranne al più l'errore $A[i] > A[parent(i)]$, quindi $A[i]$ e $A[brother(i)]$ sono radici di heap e $A[2i], A[2i+1], A[brother(i)] \leq A[parent(i)]$.

PASSO.

HpInd) Supponiamo che all'inizio dell'iterazione in cui i vale h valga che $A[h]$ e $A[brother(h)]$ sono radici di heap, $A[2h], A[2h+1], A[brother(h)] \leq A[parent(h)]$ e in A c'è al più l'errore $A[h] > A[parent(h)]$.

Ts) All'inizio dell'iterazione in cui i vale $h/2$ abbiamo che $A[h/2]$ e $A[brother(h/2)]$ sono radici di heap, $A[2(h/2)], A[2(h/2)+1], A[brother(h/2)] \leq A[parent(h/2)]$, e in A c'è al più l'errore $A[h/2] > A[parent(h/2)]$.

Dobbiamo analizzare cosa avviene durante l'esecuzione del ciclo while con $i = h$. Dal Lemma 1 abbiamo che l'esecuzione del ciclo while rende $A[parent(h)] = A[h/2]$

radice di heap. Per ipotesi in A c'era al più l'errore $A[h] > A[parent(h)]$. Lo scambio tra $A[h]$ e $A[parent(h)]$ ha al più spostato l'errore in $A[h/2] > A[parent(h/2)]$, quindi $A[brother(h/2)]$ è radice di heap e $A[2(h/2)], A[2(h/2)+1], A[brother(h/2)] \leq A[parent(h/2)]$. □

Lemma 2 (Correttezza di NoRicCorreggi2(A, i))**.** NoRicCorreggi2(A, i) *termina sempre. Se A è una heap tranne al più l'errore $A[i] > A[parent(i)]$, allora al termine di* NoRicCorreggi2(A, i) *A è una heap.*

Dimostrazione. Il ciclo while che costituisce NoRicCorreggi2(A, i) termina sempre perché tra le guardie del ciclo abbiamo la condizione $i > 1$ e l'unica istruzione all'interno del ciclo che modifica la variabile i la dimezza ogni volta. Quindi dopo al più un numero finito di esecuzioni del ciclo while la variabile i raggiunge il valore 1.

Dimostriamo che se A è una heap tranne al più l'errore $A[i] > A[parent(i)]$, allora al termine di NoRicCorreggi2(A, i) A è una heap. Ci sono due possibilità:

- se il ciclo while termina perché $i \leq 1$, allora dall'invariante sappiamo che non ci sono errori ed A è una heap (l'errore sarebbe sopra alla radice);
- se il ciclo while termina perché $A[i] \leq A[parent(i)]$, allora sempre dall'invariante sappiamo che A non contiene errori ed è una heap.

□

La correttezza di NoRicHeapify si dimostra in modo simile.

La correttezza di HeapModify segue immediatamente dalla correttezza di NoRicCorreggi2 e NoRicHeapify.

Complessità. Sia $n = length[A] = heapsize[A]$.

La procedura NoRicCorreggi2 esegue un ciclo while. Tutte le operazioni all'interno del ciclo while (compresa l'intestazione del ciclo while) hanno complessità $\Theta(1)$. Quindi per determinare la complessità della procedura dobbiamo determinare quante volte viene eseguito al più il ciclo while. Ad ogni iterazione del ciclo while ci si sposta verso l'alto di un livello sulla heap A. Quindi al più il ciclo while può essere eseguito tante volte quanti sono i livelli della heap A (ovvero tante volte quanta è l'altezza di A). Per quanto visto a lezione l'altezza di A è $\Theta(\log n)$. Quindi la complessità della procedura è $O(\log n)$.

Analogamente, ma scendendo verso il basso, si dimostra che la complessità della procedura NoRicHeapify è $O(\log n)$.

Quindi la complessità della procedura HeapModify è $O(\log n)$.