ESERCIZIO DI ASD DEL 24 NOVEMBRE 2008

Liste Cicliche

Lista con Elementi Positivi. Ogni elemento x della lista L ha due campi $key[x]$ e $next[x]$. Per accedere al primo elemento della lista L si utilizza il campo $head[L]$ di L.

Scorriamo gli elementi della lista L. Trasformiamo la chiave di ogni elemento x che troviamo in $-key[x]$. Se raggiungiamo un elemento con chiave negativa, allora la lista è ciclica.

Algorithm 1 ListaPosCiclica(L)

```
1: if head[L] = NIL then
2:    ciclica ← False
3: else
4:    x ← head[L]
5:    while next[x] ≠ NIL and key[x] > 0 do
6:       key[x] ← −key[x]
7:       x ← next[x]
8:    end while
9:    if next[x] ≠ NIL then
10:      ciclica ← True
11:   else
12:      ciclica ← False
13:   end if
14:   x ← head[L]
15:   while next[x] ≠ NIL and key[x] < 0 do
16:      key[x] ← −key[x]
17:      x ← next[x]
18:   end while
19: end if
20: return ciclica
```

Lista con Elementi Interi. Una possibilità sarebbe quella di scandire la lista alla ricerca del massimo max della lista, per poi modificare gli elementi della lista trasformandoli tutti in $max + 1$. Se si raggiunge un elemento con chiave $max + 1$, allora la lista è ciclica. Purtroppo se la lista è ciclica la ricerca del massimo non termina. Non abbiamo quindi modo di modificare le chiavi degli elementi della lista in modo da essere sicuri di riconoscere gli elementi su cui si è già passati. Proviamo dunque a modificare i campi $next$ degli elementi che scandiamo, in modo

Date: 24 Novembre 2008.

da riconoscere gli elementi su cui siamo già passati. Per esempio possiamo modificare $next[x]$ ed assegnargli come valore x stesso. Se raggiungiamo un elemento il cui $next$ è se stesso, allora o abbiamo raggiunto un elemento già visto oppure la lista ha un ciclo che contiene un solo elemento. In ogni caso la lista è ciclica. Nell'implementare questa procedura dobbiamo fare attenzione a due cose:

- visto che modificheremo $next[x]$, dobbiamo salvare $next[x]$ in una variabile temporanea y, altrimenti non riusciremo a procedere con la scansione
- l'esercizio richiede che al termine la lista non sia stata modificata, quindi procediamo anche a creare una copia della lista.

Usiamo il costrutto *new* per creare gli elementi della lista "copia" che creiamo

Algorithm 2 ListaCiclica(L)

```
1: if head[L] = NIL then
2:    ciclica ← False
3: else
4:    x ← head[L]
5:    new[z]
6:    key[z] ← key[x]
7:    head[L] ← z
8:    while next[x] ≠ NIL and next[x] ≠ x do
9:       y ← next[x]
10:      next[x] ← x
11:      new[w]
12:      key[w] ← key[y]
13:      next[z] ← w
14:      x ← y
15:      z ← w
16:   end while
17:   if next[x] ≠ NIL then
18:      ciclica ← True
19:   else
20:      ciclica ← False
21:   end if
22:   next[z] ← NIL
23: end if
24: return ciclica
```

Correttezza. Dimostriamo la correttezza della procedura ListaPosCiclica(L). Dimostriamo prima gli invarianti relativi ai cicli while.

Invariante 1. *Se L contiene solo elementi aventi chiavi maggiori di 0 ed L ha n elementi, allora per ogni $i \leq n$ all'inizio della i-esima iterazione del ciclo while x è l'i-esimo elemento di L, tutti gli elementi che precedono x hanno chiave negativa e tutti gli elementi che non sono ancora stati raggiunti, x compreso, hanno chiave positiva.*

Dimostrazione. Procediamo per induzione su i.
BASE $i = 1$. x è head L, tutti gli elementi hanno chiave positiva e non ci sono elementi che precedono x.
PASSO.
HPInd) Vale la tesi per $i < k$.
TS) Vale la tesi per $i = k$.

Dobbiamo analizzare cosa avviene durante la $k-1$-esima iterazione del ciclo while. All'inizio dell'iterazione x è il $k-1$-esimo elemento. Per ipotesi induttiva x ha chiave positiva. Durante il ciclo la chiave di x viene modificata e diventa negativa. Inoltre x viene spostato in avanti di una posizione nella lista L. Quindi vale la tesi. □

Quindi possiamo dimostrare che *ciclica* assume valore $True$ se e soltanto se la lista L è ciclica.

Lemma 1. *ciclica assume valore True sse L è ciclica.*

Dimostrazione. Se *ciclica* assume valore $True$, allora abbiamo raggiunto un elemento con chiave negativa. Dall'invariante abbiamo che gli elementi con chiave negativa sono tutti e soli gli elementi che sono già stati analizzati. Quindi c'è un elemento di L che è stato analizzato due volte ed L è ciclica.

Se L è ciclica, allora nel ciclo while torneremo due volte sullo stesso elemento di L. Dall'invariante abbiamo che questo elemento avrà chiave negativa, quindi *ciclica* assume valore $True$. □

Da queste considerazioni segue la correttezza della procedura. Per motivi di tempo non riesco ad entrare maggiormente nei dettagli.

Complessità. È immediato osservare che entrambe le procedure descritte hanno complessità $\Theta(n)$, dove n è la lunghezza della lista L.