ESAME DI ALGORITMI E STRUTTURE DI DATI (A)
Venerdì 17 Maggio 2002

NOME:
COGNOME:
MATRICOLA:

Per un buon esito dell'esame, si consiglia di:

- scrivere in forma leggibile il proprio nome, cognome e matricola sul testo del compito e su ogni foglio consegnato;
- provare gli esercizi prima in brutta copia. Ricopiarli in bella copia e consegnare solo quest'ultima;
- non fatevi prendere dal panico, e neppure dalla noia. Un giusto livello di tensione è ciò che serve;
- svolgere il compito individualmente. Passare copiando è dannoso per voi, e irrilevante per il docente.

Esercizio 1 (*Punti 15*)

Una lista circolare è una lista in cui il campo next dell'ultimo elemento punta al primo elemento della lista, e il campo prev del primo elemento punta all'ultimo elemento della lista. Sia L una lista circolare. Si scrivano le seguenti procedure e si calcoli la relativa complessità computazionale:

- *$CircularListSearch(L, k)$ che cerca nella lista L l'oggetto con chiave k e ritorna un puntatore a tale oggetto, se esso esiste, e ritorna NIL altrimenti;*
- *$CircularListDelete(L, x)$ che cancella dalla lista L l'oggetto puntato da x .*

Soluzione

Algoritmo 1 CircularListSearch(L,k)

CircularListSearch(L,k)

```
1: if head(L) = NIL then
2:   return NIL
3: end if
4: if key[head(L)] = k then
5:   return head(L)
6: end if
7: x ← next[head(L)]
8: while (x ≠ head(L)) and (key[x] ≠ k) do
9:   x ← next[x]
10: end while
11: if x = head(L) then
12:   return NIL
13: else
14:   return x
15: end if
```

La complessità è $\Theta(n)$, con n la lunghezza della lista.

Algoritmo 2 CircularListDelete(L,x)

CircularListDelete(L,x)

```
1: next[prev[x]] ← next[x]
2: prev[next[x]] ← prev[x]
3: if x = head(L) then
4:   if x = next[x] then
5:     head(L) ← NIL
6:   else
7:     head(L) ← next[x]
8:   end if
9: end if
```

La complessità è $\Theta(1)$.

Esercizio 2 (*Punti 15*)

Si dica, giustificando la risposta, se il seguente algoritmo risolve il problema dell'ordinamento e si calcoli la complessità computazionale ottima e pessima.

Algoritmo 3 InsertionSort(A)

InsertionSort(A)

```
1: for  $j \leftarrow 2$  to  $length(A)$  do
2:    $key \leftarrow A[j]$ 
3:    $i \leftarrow j - 1$ 
4:   while  $(i > 0)$  and  $(key < A[i])$  do
5:      $A[i + 1] \leftarrow A[i]$ 
6:      $i \leftarrow i - 1$ 
7:   end while
8:    $A[i + 1] \leftarrow key$ 
9: end for
```

Soluzione

La procedura ordina il vettore A in senso crescente. Infatti, ogni iterazione del ciclo **for** fa scivolare l'elemento $A[j]$ nella giusta posizione all'interno del sottovettore $A[1, \dots, j]$. Ne deriva che al termine di ogni iterazione del ciclo **for**, il sottovettore $A[1, \dots, j]$ contiene gli elementi che originariamente si trovavano in $A[1, \dots, j]$ ma ordinati in senso crescente. Alla fine dell'ultima iterazione $j = length(A)$, e dunque il vettore $A[1, \dots, length(A)]$ è ordinato in senso crescente.

Sia $n = length(A)$. Il caso ottimo si ha quando A è inizialmente ordinato in senso crescente. In tal caso, il ciclo **while** non viene mai eseguito, e dunque ogni iterazione del ciclo **for** ha complessità costante. Dato che ci sono $n - 1$ iterazioni del ciclo **for**, la complessità ottima risulta $(n - 1)\Theta(1) = \Theta(n - 1) = \Theta(n)$.

Il caso pessimo si ha quando A è inizialmente ordinato in senso decrescente. In tal caso, il ciclo **while** viene sempre eseguito fino all'ultimo, cioè da $i = j - 1$ fino a $i = 0$. Dunque ogni iterazione del ciclo **for** ha complessità $\Theta(j)$. Dato che il ciclo **for** itera da $j = 2$ fino a $j = n$, la complessità pessima risulta $\sum_{j=2}^n \Theta(j) = \Theta(\sum_{j=2}^n j) = \Theta(n(n + 1)/2 - 1) = \Theta(n^2)$.

ESAME DI ALGORITMI E STRUTTURE DI DATI (B)
Venerdì 17 Maggio 2002

NOME:
COGNOME:
MATRICOLA:

Per un buon esito dell'esame, si consiglia di:

- scrivere in forma leggibile il proprio nome, cognome e matricola sul testo del compito e su ogni foglio consegnato;
- provare gli esercizi prima in brutta copia. Ricopiarli in bella copia e consegnare solo quest'ultima;
- non fatevi prendere dal panico, e neppure dalla noia. Un giusto livello di tensione è ciò che serve;
- svolgere il compito individualmente. Passare copiando è dannoso per voi, e irrilevante per il docente.

Esercizio 3 (Punti 15)

Una lista circolare è una lista in cui il campo next dell'ultimo elemento punta al primo elemento della lista, e il campo prev del primo elemento punta all'ultimo elemento della lista. Sia L una lista circolare. Si scrivano le seguenti procedure e si calcoli la relativa complessità computazionale:

- *CircularListLength(L)* ritorna la lunghezza della lista L ;
- *CircularListInsert(L, x)* che inserisce l'oggetto puntato da x in testa alla lista L .

Soluzione

Algoritmo 4 CircularListLength(L)

CircularListLength(L)

```
1: if head( $L$ ) = NIL then  
2:    $len \leftarrow 0$   
3: else  
4:    $len \leftarrow 1$   
5:    $x \leftarrow next[head(L)]$   
6:   while  $x \neq head(L)$  do  
7:      $x \leftarrow next[x]$   
8:      $len \leftarrow len + 1$   
9:   end while  
10: end if  
11: return  $len$ 
```

La complessità è $\Theta(n)$, con n la lunghezza della lista.

Algoritmo 5 ListInsert(L,x)

ListInsert(L,x)

```
1: if head(L) ≠ NIL then
2:   y ← prev[head(L)]
3:   z ← head(L)
4: else
5:   y ← x
6:   z ← x
7: end if
8: prev[x] ← y
9: next[x] ← z
10: if y ≠ x then
11:   next[y] ← x
12: end if
13: if z ≠ x then
14:   prev[z] ← x
15: end if
16: head(L) ← x
```

La complessità è $\Theta(1)$.

Esercizio 4 (Punti 15)

Si dica, giustificando la risposta, se il seguente algoritmo risolve il problema dell'ordinamento e si calcoli la complessità computazionale ottima e pessima.

Algoritmo 6 InsertionSort(A)

InsertionSort(A)

```
1: for j ← 2 to length(A) do
2:   key ← A[j]
3:   i ← j - 1
4:   while (i > 0) and (key > A[i]) do
5:     A[i + 1] ← A[i]
6:     i ← i - 1
7:   end while
8:   A[i + 1] ← key
9: end for
```

Soluzione

La procedura ordina il vettore A in senso decrescente. Infatti, ogni iterazione del ciclo **for** fa scivolare l'elemento $A[j]$ nella giusta posizione all'interno del sottovettore $A[1, \dots, j]$. Ne deriva che al termine di ogni iterazione del ciclo **for**, il sottovettore $A[1, \dots, j]$ contiene gli elementi che originariamente si trovavano in $A[1, \dots, j]$ ma ordinati in senso decrescente. Alla fine dell'ultima iterazione $j = \text{length}(A)$, e dunque il vettore $A[1, \dots, \text{length}(A)]$ è ordinato in senso decrescente.

Sia $n = \text{length}(A)$. Il caso ottimo si ha quando A è inizialmente ordinato in senso decrescente. In tal caso, il ciclo **while** non viene mai eseguito, e dunque ogni iterazione del ciclo **for** ha complessità costante. Dato che ci sono $n - 1$ iterazioni del ciclo **for**, la complessità ottima risulta $(n - 1)\Theta(1) = \Theta(n - 1) = \Theta(n)$.

Il caso pessimo si ha quando A è inizialmente ordinato in senso crescente. In tal caso, il ciclo **while** viene sempre eseguito fino all'ultimo, cioè da $i = j - 1$ fino a $i = 0$. Dunque ogni iterazione del ciclo **for** ha complessità $\Theta(j)$. Dato che il ciclo **for** itera da $j = 2$ fino a $j = n$, la complessità pessima risulta $\sum_{j=2}^n \Theta(j) = \Theta(\sum_{j=2}^n j) = \Theta(n(n+1)/2 - 1) = \Theta(n^2)$.