Corso di Informatica Generale
Canale M–Z

Esercizi di Programmazione

Roma, 7 giugno 2011

Ovunque ritenuto opportuno definire e avvalersi di funzioni ausiliarie.

Vettori, Matrici e Stringhe

Esercizio 1 Scrivere una funzione C:

```
int eliminaDoppie(char** x)
```

che elimina da `x` tutti i caratteri ripetuti consecutivamente (una o più volte), lasciando una sola occorrenza. L'intero restituito deve essere la lunghezza della nuova stringa.
Esempio: Avendo in input la stringa 'pppppolllooooo' la funzione deve modificarla in 'polo' e restituire il valore 4 al chiamante.

Esercizio 2 Scrivere una funzione C:

```
int rotateK(int A[], int n, int k)
```

che sposta tuti gli elementi di un array `A` di lunghezza `n` di `k` posizioni, interpretando il vettore come circolare, cioè considerando l'elemento di indice 0 successivo a quello di indice `n-1`. L'intero restituito dalla funzione al chiamante dovrà essere il numero di elementi che sono uguali a quello di cui prendono il posto.

Esempi: Se `k=0`, il vettore non viene modificato, e il valore restituito è `n` (ogni elemento "sostituisce" se stesso). Supponendo `k=2`, la funzione dovrà ad esempio modificare il vettore `A={1,4,2,3,2}` in {`3,2,1,4,2`}, restituendo come risultato `1` (un 2 viene messo al posto di un altro 2).

Esercizio 3 [**Quadrato magico**]. Una matrice quadrata di lato n è un quadrato magico se contiene tutti i numeri da 1 a n^2 e la somma di tutte le righe, tutte le colonne e delle due diagonali principali è costante. Ad esempio, ecco due quadrati magici di ordine 3 e 5:

```
8  1  6          17 24  1  8 15
3  5  7          23  5  7 14 16
4  9  2           4  6 13 20 22
                 10 12 19 21  3
                 11 18 25  2  9
```

Scrivere un programma che verifica se una matrice quadrata é un quadrato magico.
Input dimensione della matrice (un intero); poi $n \times n$ valori interi.
Output 1 se la matrice quadrata letta è un quadrato magico, 0 altrimenti.

Esercizio 4 Scrivere una funzione:

```
int maxLoc(int r, int c, int M[][c], int massimi[])
```

che carica il vettore `massimi` con i valori che sono massimi locali nella matrice `M`. Un elemento è *massimo locale*, se è maggiore strettamente di tutti i suoi vicini in tutte le direzioni (attenzione ai bordi!).

La funzione inoltre restituisce il numero dei massimi locali trovati.

Liste

Esercizio 1 Scrivere una funzione **ricorsiva** C:

```
lista sommaListaK(lista L, int k, int* lun)
```

che verifica se in nella lista di interi `L` esiste una sottolista di elementi consecutivi di somma `k`.

Se tale sottolista non esiste, la funzione restituirà il valore `NULL`, altrimenti restituirà il puntatore all'elemento in cui comincia la sottolista di somma `k`, e caricherà nella variabile `lun` la lunghezza di tale sottolista.

Esercizio 2 Un numero intero positivo viene rappresentato con una lista di cifre decimali, dove il primo nodo della lista contiene la cifra meno significativa. Ad esempio l'intero 244, sarà rappresentato dalla lista: $4 \to 4 \to 2 \to NULL$. Scrivere una funzione C:

`lista prod(lista N, int k)`

che crei una *nuova* lista contenente la rappresentazione del numero ottenuto moltiplicando il numero rappresentetato da `N` per l'intero `k` (`k` compreso tra 1 e 9) e restituisca un puntatore alla testa della nuova lista (ricordarsi di propagare i riporti!).

Esercizio 3 Consideriamo liste come nell'esercizio precedente. Scrivere una funzione C:

`lista somma(lista N, lista M)`

che crei una *nuova* lista contenente la rappresentazione del numero ottenuto sommando il numero rappresentetato da `N` con il numero rappresentato da `M` e restituisca un puntatore alla testa della nuova lista (ricordarsi di propagare i riporti!).

Esercizio 4 In una lista di interi, un *punto di equilibrio* è un nodo per cui la somma degli elementi che lo precedono (esso compreso) è uguale alla somma degli elementi che lo seguono (esso escluso). Scrivere una funzione C:

```
int puntiDEquilibrio(lista L)
```

che data la lista `L` conta il numero dei suoi punti di equilibrio.

Esempio: Osservate che se ci sono numeri negativi, i punti d'equilibrio possono essere più d'uno (ovviamente potrebbe anche non esserci nessun punto di equilibrio). A titolo di esempio nella lista $1 \to \mathbf{2} \to -4 \to \mathbf{4} \to 3 \to$ `NULL` sono indicati in grassetto i punti di equilibrio.

Facoltativo: Scrivere sia la versione iterativa che ricorsiva dell'esercizio 3.

Alberi

Esercizio 1 Il *peso* di un albero è la somma delle informazioni contenute nei suoi nodi. Scrivere una funzione C:

`tree pesoMax(tree T)`

che restituisca un puntatore alla radice del sottoalbero di `T` di peso massimo. Similmente con il peso minimo.

Esercizio 2 Un *cammino* in un albero è una sequenza di nodi dalla radice a una foglia. Il *peso* di un cammino è la somma delle informazioni contenute nei suoi nodi. Scrivere una funzione C:

```
lista pesoMaxPath(tree T)
```

che restituisca una lista che contiene il cammino di peso massimo di `T`.

Esercizio 3 Scrivere una funzione C:

```
tree peso(tree T, int k)
```

che restituisca un sottoalbero di `T` di peso `k` se esiste, e `NULL` altrimenti. Scrivere la stessa procedura sotto le ipotesi che nel campo informazione di `T` ci siano solo numeri positivi.

Esercizio 4 Scrivere una funzione C:

```
lista cerca(tree T, int x)
```

che restituisca in una lista il cammino che lega la radice a un nodo che contiene `x` se esiste nell'albero un nodo con campo informazione `x`, e `NULL` altrimenti.