Tecniche di Programmazione

Esercitazione 12 (mista)

Questa è un'esercitazione contenente argomenti misti, usata per ripassare e migliorare sugli argomenti fatti finora. Troverete vari tipi al suo interno: strutture matrici, SCL, e tipi astratti. Per ognuno degli esercizi che seguono è necessario utilizzare le definizioni ogni volta appropriate. Tutte le implementazioni necessarie allo svolgimento sono state rilasciate nel corso delle scorse esercitazioni. Inoltre, tutti gli esercizi che manipolano tipi astratti devono essere risolti senza accedere all'implementazione.

Esercizio 12.1

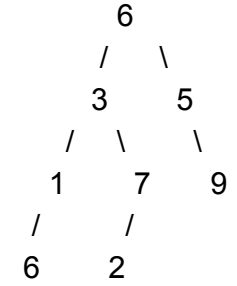
Implementare la funzione

```
int singleChildSum(TipoAlbero a);
```

che, dato in input un albero binario, restituisca la somma dei valori dei nodi che hanno un solo figlio

Esempio

Dato il seguente albero src in ingresso:



`singleChildSum(src)`
dovrà restituire: 1 + 7 + 5 = 13

Esercizio 12.2

Scrivere una funzione

```
int livelli_completi(TipoAlbero a);
```

che restituisca il livello massimo `l` a cui l'albero `a` è completo fino alla profondità `l`. Per un albero vuoto, si restituisca -1.

Esercizio 12.3

Implementare la seguente funzione

```
TipoLista listaNodiFoglia(TipoAlbero a);
```

che, dato un albero binario, restituisca una lista con i valori contenuti nelle foglie (seguendo l'ordine da sinistra a destra).

Esercizio 12.4

Implementare la una funzione

```
int conditionalSum(Mat* src, int flag);
```

che, data in input una matrice src, restituisca un intero ottenuto sommando tutti gli elementi della matrice, rispettando le seguenti regole:

- Se flag == 0, gli elementi di src presenti nelle colonne di indice dispari devono essere premoltiplicati per -1;
- Se flag == 1, gli elementi di src presenti nelle righe di indice dispari devono essere premoltiplicati per -1;

Esempio

Input:

```
1 4 5
9 2 6
8 7 3
```

Output:

- Se flag == 0: (1-4+5+9-2+6+8-7+3) = 19
- Se flag == 1: (1+4+5-9-2-6+8+7+3) = 11

Esercizio 12.5

Implementare la funzione

```
TipoAlbero taglia_livello(TipoAlbero a, int livello);
```

che, dato in input un albero *a* ed un intero `livello`. Restituisca un nuovo albero, costruito a partire da `a`, in cui ogni nodo profondità maggiore di `livello` deve essere eliminato e il suo valore sommato al suo antenato di livello "`livello`".

Esercizio 12.5

Sia data una struttura collegata lineare di interi `TipoSCL`.

Scrivere una funzione C *ricorsiva*

```
int contieneElemento(TipoSCL scl, TipoInfoSCL e);
```

che, data in input una SCL scl e un elemento e, restituisca 1 se la scl contiene l'elemento `e`, 0 altrimenti. La SCL in ingresso alla funzione non deve essere modificata in alcun modo.

Esempio: Data la seguente lista in ingresso:

```
scl = 5 -> 3 -> 8 -> 8 -> 7 -> 0 -> 11

contieneElemento(scl, 3) = 1
contieneElemento(scl, 2) = 0
```

Esercizio 12.6

Scrivere una funzione

```
TipoSCL complemento(TipoSCL scl, TipoSCL elementiDaScartare);
```

che data in input una SCL scl e una seconda SCL `elementiDaScartare` (eventualmente vuote), restituisca una nuova lista contenente tutti gli elementi di scl non contenuti in `elementiDaScartare`.
Se scl contiene duplicati, i duplicati devono essere riportati tutti nella SCL di output se il valore non compare in `elementiDaScartare`, e non essere presenti neanche una volta nella SCL di output se il valore è presente in `elementiDaScartare`.
Le SCL in ingresso alla funzione non devono essere modificate in alcun modo.
Nota: una o entrambe le SCL di input possono essere vuote.

Esempio: Data le seguenti scl in ingresso:

```
scl = 5 -> 3 -> 8 -> 8 -> 2 -> 7 -> 0 -> 11 -> 2
elementiDaScartare = 8 -> 7 -> 11

complemento(scl, elementiDaScartare) = 5 -> 3 ->  2 -> 0 -> 2
```

Esercizio 12.7

Si consideri una matrice che rappresenta la mappa del gioco *snake*, in cui:

- 'X' e' la testa del serpente;
- '#' delimita i bordi della mappa;
- 'o' sono le mele da mangiare;
- lo spazio ' ' rappresenta spazio vuoto nella mappa.

Si implementi la funzione:

```
int next_step(Mat *mat, char direction)
```

che, data la matrice mat e un carattere direction, che rappresenta una direzione secondo la seguente convenzione:

- 'w' in alto
- 'a' a sinistra
- 's' in basso
- 'd' a destra

effettui side effect sulla matrice in modo tale da spostare la testa del serpente ('X') nella casella adiacente a quella occupata, secondo la direzione indicata da direction. Inoltre la funzione dovrà restituire un valore intero secondo le seguenti regole:

- 1 in caso il serpente raggiunga una mela,
- -1 in caso il serpente vada su un bordo della mappa,
- 0 altrimenti.

NOTA: la testa del serpente va spostata anche quando raggiunge un bordo.
NOTA: si assuma che inizialmente il serpente sia dentro i bordi.

Esempio

Data la matrice:

```
# # # # # # # # # #
# o o o   o o     #
#   o   o   o   o #
#       o o       #
#             o   #
# X o       o     #
# o       o o o   #
#             o o #
#     o       o o #
# # # # # # # # # #
```

e la direzione 'a', la funzione deve restituire -1, modificando la matrice come segue:

```
# # # # # # # # # #
# o o o   o o     #
#   o   o   o   o #
#       o o       #
#             o   #
X   o       o     #
# o       o o o   #
#             o o #
#     o       o o #
# # # # # # # # # #
```

Esercizio 12.8

Implementare **ricorsivamente** la funzione:

```
TipoLista inFondo(TipoLista l);
```

che restituisce una nuova lista, costruita a partire da l, in cui il primo nodo di l si trovera' in ultima posizione nella lista di output.
E' obbligatorio usare solo l'interfaccia del tipo astratto.

Esempio

Input:
l = 1,2,3,4,5,6
il risultato e'
2,3,4,5,6,1

Esercizio 12.9

Implementare la funzione:

```
TipoLista invertiInParte(TipoLista l, int inizio);
```

che restituisce una nuova lista, costruita a partire da l, in cui gli elementi in posizioni {0, 1, …, inizio - 1} sono copiati in output, mentre i rimanenti sono copiati in ordine inverso.
E' obbligatorio usare solo l'interfaccia del tipo astratto.

Esempio

input l: 1, 2, 3, 4, 5, 6
input inizio: 3
output: 1, 2, 3, 6, 5, 4