Esercitazione 11 (per casa)

Istruzioni

- Scaricare la directory `Esercitazione11` ed estrarne il contenuto.
- Completare la definizione delle funzioni presenti nel file `esercizio.c` secondo le indicazioni di seguito riportate.
- Per compilare il file `testexe.c`, contenente i test di verifica del codice, eseguire il comando `make` dalla directory `Esercitazione11`, quindi eseguire il file `test_esercizio`.

Si ricorda che non è consentito modificare la segnatura delle funzioni (tipo restituito, nome della funzione, tipo e numero dei parametri). È invece consentito definire (ed usare) funzioni ausiliarie. Nei casi in cui si richiede l'implementazione ricorsiva di una funzione, è consentito usare funzioni ausiliarie, purché implementate ricorsivamente. In caso di dubbio chiedere al docente.

1 Esercizio 11.1

Fornire un'implementazione ricorsiva della funzione:

- `void scambiaSCL(TipoLista* l)`

Presa in input una lista `l` rappresentata mediante struttura collegata, `scambiaSCL` deve scansionare `l` dal primo all'ultimo elemento, scambiando un elemento con il successivo ogni volta che il primo sia maggiore del secondo. Per la definizione di `TipoLista` si faccia riferimento ai file `lista.h` e `lista.c` presenti nella directory `Esercitazione11`.

Esempio La lista $\langle 3, 5, 4, 1, 8, 7\rangle$ deve essere modificata dalla funzione come segue: $\langle 3, 4, 1, 5, 7, 8\rangle$. Si osservi che un elemento può essere scambiato più volte: ad esempio, il 5 viene prima scambiato con il 4 e successivamente con l'1.

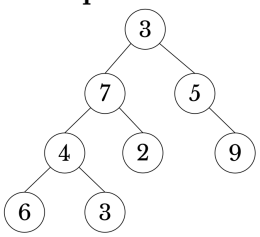
2 Esercizio 11.2

Implementare la funzione:

- `Insieme* nodiNormali(TipoAlbero a, int l)`

Presi in input un albero binario `a` ed un intero `l`, la funzione deve restituire l'insieme degli elementi contenuti nei nodi *normali* che si trovano al livello `l` di `a`. Un nodo di un albero binario è detto *normale* se i suoi sottoalberi sono o entrambi vuoti o entrambi non vuoti. Per la definizione di `TipoAlbero` ed `Insieme` si faccia riferimento ai file `albero_binario.h`, `albero_binario.c`, `insieme.h` e `insieme.c` presenti nella directory `Esercitazione11`.

Esempio Sia `a` il seguente albero di input:



- L'invocazione `nodiNormali(a, 1)` deve restituire l'insieme $\{7\}$
- L'invocazione `nodiNormali(a, 2)` deve restituire l'insieme $\{4, 2, 9\}$

3 Esercizio 11.3

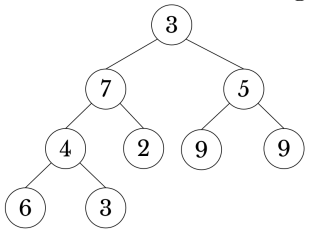
Implementare la funzione:

- `void normalizza(TipoAlbero* a)`

Preso in input un riferimento `a` ad albero binario, la funzione deve modificare l'albero puntato da `a` come segue:

- ad ogni nodo non *normale* (ovvero contenente esattamente un sottoalbero non vuoto) di `a` aggiunge un nodo figlio contenente il valore della radice del sottoalbero presente.

Esempio Preso un riferimento all'albero dell'esempio precedente, `normalizza` deve modificare l'albero come segue:



4 Esercizio 11.4

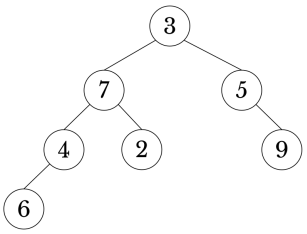
Implementare la funzione:

- `void tagliaFoglie(TipoAlbero* a)`

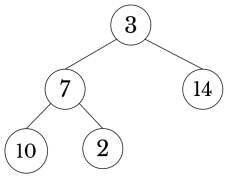
che, preso in input un riferimento `a` ad un albero binario, modifica l'albero come segue:

- per ogni nodo padre di un solo nodo che sia foglia, la funzione aggiunge al valore del nodo padre il valore della foglia, quindi la elimina.

Esempio Con il seguente albero di input:



`tagliaFoglie` deve modificare l'albero come segue:



5 Esercizio 11.5

Implementare la funzione:

- `TipoAlbero normalizzaFunzionale(TipoAlbero a)`

Preso in input un albero binario `a`, la funzione deve restituire un nuovo albero binario ottenuto da `a` modificandolo come segue:

- ad ogni nodo non *normale* (ovvero contenente esattamente un sottoalbero non vuoto) di `a` aggiunge un nodo figlio contenente il valore della radice del sottoalbero presente.

6 Esercizio 11.6

Implementare la funzione:

- `TipoAlbero tagliaFoglieFunzionale(TipoAlbero a)`

che, preso in input un albero binario `a`, restituisce un nuovo albero ottenuto da `a`, modificandolo come segue:

- per ogni nodo padre di un solo nodo che sia foglia, aggiunge al valore del nodo padre il valore della foglia, qunidi la elimina.