# Tecniche di Programmazione

## Esercitazione 11

- Si consideri alberi binari con valori di tipo float.
- Scrivere dei test nel main per verificare che le funzioni scritte siano corrette.
- Tutti gli esercizi con interfaccia funzionale devono essere risolti usando solo le funzioni dell'implementazione funzionale del tipo astratto Albero Binario disponibili nelle dispense. Tutti gli esercizi con interfaccia con side effect devono essere risolti usando solo le funzioni dell'implementazione con side-effect del tipo astratto Albero Binario pubblicate su classroom (è possibile usare inoltre la funzione free per deallocare memoria)
- Svolgere almeno due esercizi in maniera iterativa utilizzando le strutture Pila/Coda (da implementare)

### Esercizio 11.1

Implementare la funzione

```
void normalizza(TipoAlbero *a);
```

Che dato in ingresso un albero `a` ne modifichi il contenuto come segue:
Il valore di ogni nodo deve essere normalizzato (diviso) sulla somma di tutti i nodi dell'albero.

### Esercizio 11.2

Implementare la funzione

```
TipoAlbero normalizza_funz(TipoAlbero a);
```

che calcola lo stessa funzione di 11.1 ma viene restituito un nuovo albero e quello in input non viene modificato.

### Esercizio 11.3

Implementare la funzione

```
TipoAlbero da_array_ordinato(const float *a, int n);
```

che, dato un array di valori ordinati in modo crescente, crei un albero binario di ricerca bilanciato contenente tutti e soli i valori di n.

Un albero binario è detto bilanciato se per ogni suo nodo, la profondità del suo sottoalbero sinistro differisce di al più una unità, in modulo, rispetto a quella del suo sottoalbero destro.

## Esercizio 11.4

Implementare la funzione

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

Che dato un albero in input e un livello, modifichi l'albero a come segue:

- Tutti i nodi a profondità maggiore di `livello` devono essere eliminati e deallocati e la somma dei loro valori deve essere aggiunta al padre.

## Esercizio 11.5

Implementare la funzione

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

che calcola lo stessa funzione di 11.6 ma viene restituito un nuovo albero e quello in input non viene modificato.

## Esercizio 11.6

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.