Esercitazione 11

Prof. Riccardo Zese - riccardo.zese@unife.it

Ing. Alice Bizzarri - alice.bizzarri@unife.it
Ing. Elisabetta Gentili - elisabetta.gentili1@unife.it

Laboratorio Fondamenti di Informatica - Modulo B

Esercitazione su Alberi I

Si vuole creare un albero binario di ricerca (BST) contenente interi in modo da poter eseguire funzioni su esso.

Si astragga il concetto di albero in modo da rendere l'implementazione il più strutturata possibile.

Esercitazione su Alberi II

Il programma deve visualizzare un menù con diverse opzioni. Prende quindi in input un intero che corrisponde a una delle seguenti opzioni, esegue la funzione corrispondente e ritorna al menù.

0. Uscire dal programma
1. Generare un nuovo albero. Il programma deve prendere in input i numeri da inserire all'interno dell'albero usando la funzione `getElement`. Dopo ogni inserimento chiedere se si vuole continuare o tornare al menù
2. Eseguire stampa **preorder**
3. Eseguire stampa **postorder**
4. Eseguire stampa **inorder**
5. Ricerca di un elemento
6. Calcolare l'altezza dell'albero
7. Calcolare la somma degli elementi dell'albero
8. Calcolare il numero di nodi
9. Calcolare il bilanciamento dell'albero

Esercitazione su Alberi III

Per fare ciò, si cominci ad implementare una libreria (`element.h`, `element.c`) contenente la definizione dei tipi di dato e la dichiarazione delle funzioni primitive.
File element.h

```
1  //DEFINIZIONI
2  typedef int element;
3  typedef enum {FALSE, TRUE} boolean;
4
5  //DICHIARAZIONI
6  boolean isLess(element e1, element e2);
7  boolean isEqual(element e1, element e2);
8  element getElement(void);
9  void printElement(element e);
10 boolean isNull(element e);
11 element sumElement(element e1, element e2);
```

Esercitazione su Alberi IV

- `boolean isLess(element e1, element e2)` - Controlla se un elemento è minore dell'altro
- `boolean isEqual(element e1, element e2)` - Controlla l'uguaglianza fra elementi
- `element getElement(void)` - Input da tastiera di un elemento
- `void printElement(element e)` - Stampa a video di un elemento
- `boolean isNull(element e)` - Controlla che l'elemento sia diverso da 0
- `element sumElement(element e1, element e2)` - Esegue la somma di due elementi e ne restituisce il risultato

Esercitazione su Alberi V

Le funzioni di manipolazione di alberi richieste, da inserire nei file `tree.c` e `tree.h` secondo le solite modalità, sono già state introdotte a lezione. Di seguito un rapido ripasso:

```
1  typedef struct nodo {
2      element value;
3      struct nodo *left, *right;
4  } NODO;
5
6  typedef NODO *tree;
```

Esercitazione su Alberi VI

```
1  tree ord_ins(element, tree)
2  void preorder(tree);
3  void postorder(tree);
4  void inorder(tree);
5  boolean member(element, tree);
6  int height(tree);
7  element sum(tree);
8  int nnodes(tree);
9  int balance(tree);
```

NB1: nella funzione `height` è necessario richiamare una funzione aggiuntiva.
NB2: se volete implementare una funzione che restituisca il massimo fra due numeri, NON chiamatela `max` (basta, per esempio, chiamarla `mymax`).

Esercitazione su Alberi VII

Implementare le funzioni con il maggior riutilizzo di codice. In altre parole, se è già stata implementata una funzione `A` che esegue una certa operazione `Op`, se una seconda funzione necessita del risultato di `Op`, richiamare `A`.

Esercitazione su Alberi VIII

FACOLTATIVO: modificare i file `element.h` e `element.c` in modo da avere anche la seguente definizione di element:

```
typedef float element;
```

L'istruzione `typedef` e le funzioni definite nel file `element.c` vanno modificate nel passaggio da intero a float.