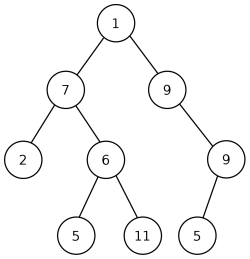
Laboratorio

Fondamenti di Programmazione 2

18 Novembre 2022

Esercizio 1

Un albero può essere rappresentato graficamente in modo testuale. Ad esempio, il seguente albero binario (non differisce molto per alberi generici):



avrà la seguente rappresentazione testuale:

```
1
  7
    2
    6
      5
      11
  9
    9
      5
```

Scrivere una funzione con segnatura:

```
template<typename T>
void stampaAlbero(const AlberoB<T>&)
```

che stampi su standard output la rappresentazione grafica dell'albero in input.

Esercizio 2

Scrivere una funzione con segnatura:

```
template<typename T>
bool isHeap(const AlberoB<T>&)
```

che restituisca `true` se e solo se ogni elemento dell'albero in input ha valore informativo minore o uguale a quello di tutti i propri figli.

Si può supporre il tipo `T` implementi `operator<`.

Esercizio 3

Sia T un albero binario di interi con radice x_0. Ad ogni nodo x_k di T associamo un *costo* $C(x_k)$ così definito:

$$C(x_k) = \sum_{i=0}^{i=k} V(x_i)$$

dove $V(x_i)$ è il valore informativo del nodo x_i e $< x_0, x_1, ..., x_k >$ è il percorso che connette x_k alla radice x_0.

Esempio Consideriamo l'albero:

```
3
  4
    7
    9
  5
    10
```

In questo caso il nodo foglia con valore 10 avrà costo $3 + 5 + 10 = 18$, mentre il nodo con valore 4 avrà costo $3 + 4 = 7$.

Scrivere una funzione con segnatura:

```
bool ogniPercorsoRadiceFoglia(const AlberoB<int>& tree, int k)
```

che restituisce `true` se e solo se nell'albero binario di interi in input non esiste nessun nodo con costo maggiore di k.

Esercizio 4

Scrivere una funzione con segnatura:

```
bool pariEDispari(const AlberoB<int>&);
```

che restituisca `true` se e solo se l'albero binario soddisfa le seguenti condizioni:

- sui livelli dispari compaiono solo numeri dispari;
- sui livelli pari compaiono solo numeri pari.

Il *livello* di un nodo è definito induttivamente come:

- la radice di un albero appartiene al livello 1;
- dato un nodo appartenente al livello L, i suoi figli appartengono al livello $L + 1$.

Esercizio 5

Sia T un albero binario di `char`. Ogni percorso radice-foglia può essere interpretato come una parola, concatenando i valori informativi (caratteri) di un qualsiasi percorso radice-foglia, ottenendo quindi una stringa.

Scrivere una funzione con segnatura:

```
bool vocaliEConsonanti(const AlberoB<char>&);
```

che restituisca `true` se e solo se per ogni parola (percorso radice-foglia) dell'albero in input il numero di consonanti e vocali differisce al più di uno.

Esercizio 6

Sia T un albero binario di interi. Ogni percorso radice-foglia può essere interpretato come l'intero che si ottiene concatenando tutte le cifre nell'ordine in cui si incontrano.

Ad esempio il seguente albero binario:

```
1
  2
  3
    4
```

codifica gli interi 12 (`1 -> 2`) e 134 (`1 -> 3 -> 4`) la cui somma è 146.

Scrivere una funzione con segnatura:

```
int sommaPath(const AlberoB<int>&);
```

che restituisce la somma degli interi rappresentati da ogni percorso radice-foglia di T.

Esercizio 7

Scrivere una funzione con segnatura:

```
bool sommaLivello(const AlberoB<int>&, int k);
```

che restituisca `true` se e solo se la somma dei valori informativi dei nodi in ciascun livello dell'albero binario in input non supera una soglia massima k.

Esercizio 8

Scrivere una funzione con segnatura:

```
bool sommaLivelliAdiacenti(const AlberoB<int>&, int k)
```

che restituisca `true` se e solo se la somma dei valori informativi dei nodi appartenenti a livelli adiacenti dell'albero in input non supera il valore massimo k. Due livelli L, L' si dicono adiacenti se $|L - L'| = 1$.

Esercizio 9

Scrivere una funzione con segnatura:

```
bool fogliePosEqfoglieNeg(const AlberoB<int>&);
```

che restituisca `true` se e solo se il numero di foglie con valore informativo positivo o nullo è uguale al numero di foglie con valore informativo negativo.

Esercizio 10

Per ogni $t \in \mathbb{N}$, una sequenza di $2^t - 1$ elementi codifica un albero binario completo di t livelli.

Ad esempio, la sequenza:

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

codifica un albero binario completo di profondità 3:

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

Scrivere una funzione con segnatura:

```
template<typename T>
AlberoB<T> vecToAlbero(const vector<T>&)
```

che presa in input una sequenza di $2^t - 1$ elementi restituisca il corrispondente albero binario completo.

Suggerimento: Guardando l'esempio, il nodo contenente il valore 3 appare all'indice 2 della sequenza. Il figlio sinistro appare all'indice $5 = 2 \cdot 2 + 1$, mentre il figlio destro appare all'indice $6 = 2 \cdot 2 + 2$. Il nodo contenente il valore 1 appare all'indice 1 della sequenza, il suo figlio sinistro appare all'indice $3 = 2 \cdot 1 + 1$ e il suo figlio destro appare all'indice $4 = 2 \cdot 1 + 2$.