ASD, laboratorio, terzo appello

18 Luglio 2018
14 punti al massimo, sufficienza 8
Tempo a disposizione per lo svolgimento: 3 ore

Questa prova d'esame consente di totalizzare al massimo **14** punti. **Su ciascuna funzione si raggiunge il punteggio massimo indicato se la funzione è corretta e se opera con una complessità "ragionevole" e non presenta errori di stile**. Con errori di stile si intendono ad esempio l'accesso alle strutture dati che non avviene attraverso le funzioni del TDD ma in modo diretto, la mancata fattorizzazione (frammenti di codice ripetuti svariate volte invece che racchiusi in una funzione), codice poco commentato, difficile da seguire, inutilmente complesso.

La soglia per raggiungere la sufficienza della prova di laboratorio è **8** punti. Il tempo a disposizione per lo svolgimento è **3 ore.**

Testo

In questo laboratorio è richiesto di implementare in C++ il tipo di dato **albero binario** (ATTENZIONE: non stiamo parlando di alberi binari di ricerca – BST, ma di semplici alberi binari) con le operazione specificate nell'apposita sezione mostrata sotto. Un'albero binario è un albero con radice dove ogni nodo ha al più due figli (figlio sinistro e figlio destro).

La traccia che vi viene fornita contiene un file *main.cpp* che implementa un semplice *menu* che richiama le operazioni del tipo di dato albero binario. Tra le opzioni offerte dal *menu* è presente la lettura da file di un albero binario.

Formato del file di input

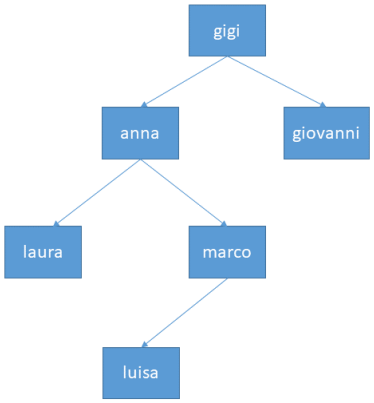
Sulla prima linea abbiamo la radice dell'albero in questo formato:
`nome_radice`
sulle linee successive abbiamo gli archi dell'albero, uno su ciascuna linea, con questo formato:
`S nome_nodo nome_figlio_sx`
oppure
`D nome_nodo nome_figlio_dx`
dove:

- `S` indica il figlio sinistro e `D` quello destro;
- la stringa `nome_nodo` è già comparsa in una linea precedente (identifica un nodo già letto)
- `nome_figlio_sx` e `nome_figlio_dx` sono nuove stringhe mai apparse prima.

Il carattere '0' (zero) segna la fine dell'input.

Assumiamo che gli alberi in input siano coerenti e sintatticamente corretti (non sono richiesti controlli), ovvero che: ogni nodo ha al più un figlio sinistro e un figlio destro.

Esempio



Esempio di albero

```
gigi
S gigi anna
D gigi giovanni
S anna laura
D anna marco
S marco luisa
0
```

Formato di input

Implementazione

Implementare una struttura dati per l'albero e le funzioni necessarie a costruire l'albero a partire dal formato di input fissato (vedere sopra). In particolare è richiesto di implementare tutte le funzioni previste dall'header che viene riportato sotto e presenti nel file *tree.h*, il cui funzionamento è spiegato brevemente nel commento dato sopra al prototipo della funzione stessa. Ogni nodo dell'albero deve essere rappresentato/implementato come una struttura (*struct*) con un membro chiamato label (di tipo stringa) e due link/puntatori al figlio destro e al figlio sinistro. La funzione di lettura da file è già implementata. Viene inoltre fornito il file *main.cpp* che contiene il *menu*, e un file di input (*esempio.txt*). I file *tree.h e main.cpp* non possono essere modificati.

Funzioni da implementare e relativo punteggio attribuito

Il punteggio assegnato a ciascuna funzione è indicato a fianco della funzione stessa. Il punteggio massimo si raggiunge non solo se la funzione è corretta, ma anche se opera con una complessità "ragionevole" e se non presenta problemi di stile.

```
namespace tree {

typedef string Label;

struct Nodo; // definita nel file tree.cpp

typedef Nodo* Tree; // un albero e' identificato dal puntatore alla sua radice;
useremo indistintamente "albero" e "puntatore a nodo"

const Tree alberoVuoto = NULL;

// Costruisce albero vuoto
Tree creaAlberoVuoto(); /* MAX punti = 0.5 */

// Restituisce true se l'albero t e' vuoto, false altrimenti
bool vuoto(const Tree& t); /* MAX punti = 0.5 */

// Aggiunge nodo radice, fallisce se l'albero e' non vuoto
bool inserisciRadice(const Label l, Tree& t); /* MAX punti = 1 */
```

```
// Aggiunge nuovo nodo sinistro di un nodo esistente. Ritorna false se non
esiste il nodo "questo" nell'albero o se la label "nome_figlio_sx" e' gia'
presente nell'albero
bool inserisciFiglioSX(Label questo, Label nome_figlio_sx, Tree& t);
/* MAX punti = 2 */

// Aggiunge nuovo nodo destro di un nodo esistente. Ritorna false se non esiste
il nodo "questo" nell'albero o se la label "nome_figlio_dx" e' gia' presente
nell'albero
bool inserisciFiglioDX(Label questo, Label nome_figlio_dx, Tree& t);
/* MAX punti = 2 */

// Stampa l'albero usando lo stesso formato di input. Se l'albero e' vuoto
stampa "Albero vuoto"
void print(const Tree& t); /* MAX punti = 2 */

// Ritorna il numero di terminali isolati. Denominiamo terminale isolato una
qualunque foglia, esclusa la radice, priva di "fratelli" (cioè il genitore della
foglia non ha altri figli)
int contaNumeroTerminaliIsolati(const Tree& t); /* MAX punti = 2.5 */

// Data una label, stampa il cammino dal nodo con quella label fino ad arrivare
alla radice. Altrimenti stampa che non esiste
void stampaCammino(Label labek, const Tree& t); /* MAX punti = 3.5 */
}
```

Funzioni di Test

Il main include alcune funzioni di testing (opzioni 5, 6, 7 del menu). Il testing che mettiamo a disposizione NON E' IN ALCUN MODO ESAUSTIVO. Il testing non è mai esaustivo e per sua natura non dà certezze sulla correttezza del programma. E' opportuno effettuare altri test in particolar modo sui casi limite.