ASD a.a. 2024/25 - Esame del 11 Giugno 2025

Prima di cominciare lo svolgimento leggete attentamente tutto il testo.

Questa prova è organizzata in tre esercizi.
Vi forniamo un file zip che contiene per ogni esercizio: un file per completare la funzione da scrivere e un programma principale per lo svolgimento di test specifici per quella funzione. Ad esempio, per l'esercizio 1, saranno presenti un file es1.cpp e un file es1-test.o. Per compilare dovrete eseguire `g++ -std=c++11 -Wall es1.cpp es1-test.o -o es1-test`. E per eseguire il test, `./es1-test`. Dovete lavorare solo sui file indicati in ciascuno esercizio. Modificare gli altri file è sbagliato (ovviamente a meno di errata corrige indicata dai docenti).
In questi file dovete implementare le funzioni richieste, esattamente con la *segnatura* con cui sono indicate: nome, tipo restituito, tipo degli argomenti nell'ordine in cui sono dati. Non è consentito modificare queste informazioni. Potete invece fare quello che volete all'interno del corpo delle funzioni: in particolare, se contengono già una istruzione `return`, questa è stata inserita provvisoriamente per rendere compilabili i file ancora vuoti, e **dovrete modificarla in modo appropriato**.
Potete inoltre realizzare altre funzioni in tutti i casi in cui lo ritenete appropriato. Potete inserirvi tutti gli `#include` che vi servono oltre a quello relativo allo header con le funzioni da implementare. Attenzione però che **usare una funzione di libreria per evitare di scrivere del codice richiesto viene contato come errore** (esempio: se è richiesto di scrivere una funzione di ordinamento, usare la funzione `std::sort()` dal modulo di libreria standard `algorithm` è un errore).
Per ciascuno esercizio, vi diamo uno programma principale, che esegue i test. Controllate durante l'esecuzione del programma, quanti sono i test che devono essere superati e controllate l'esito (se non ci sono errori deve essere `SI` per tutti).

NB [PATTO]: gli studenti che hanno sottoscritto il patto devono svolgere solo l'esercizio 1 e 2.
NB1: soluzioni particolarmente inefficienti potrebbero non ottenere la valutazione anche se forniscono i risultati attesi. Di contro ci riserviamo di premiare con un bonus soluzioni particolarmente ottimali.
NB2: superare positivamente tutti i test di una funzione non implica soluzione corretta e ottimale (e quindi valutazione massima).

1 Presentazione della struttura dati

Lo scopo di questo laboratorio è di implementare delle funzioni per la struttura dati alberi di ricerca ternari che sono degli alberi dove ciascuno nodo può contenere fino a due elementi (nel nostro caso interi positivi) e puo avere tre figli (`left`,`middle` e `right`). L'albero vuoto sarà rappresentato da `nullptr`. Lo stesso valore non può essere presente in più nodi di un albero. In più, l'albero deve rispettare le seguente regole:

- In ogni nodo del albero, abbiamo due elementi interi `v1` e `v2` che verificano le seguente proprietà:
  - `v1` è superiore o uguale a `0`
  - `v2` è superiore o uguale a `-1`
  - Se `v2` è superiore o uguale a `0` allora abbiamo `v2>v1`.
  - Se `v2` è uguale a `-1` allora il nodo è una foglia (i suoi 3 figli valgono `nullptr`).
- Per ogni nodo, tutti gli elementi diversi da `-1` nell'albero che prende come radice il figlio `left` hanno un valore strettamente inferiore al valore `v1`.
- Per ogni nodo, tutti gli elementi diversi di `-1` nell'albero che prende come radice il figlio `middle` hanno un valore strettamente superiore a `v1` e strettamente inferiore a `v2`.
- Per ogni nodo, tutti gli elementi diversi di `-1` nell'albero che prende come radice il figlio `right` hanno un valore strettamente superiore a `v2`.

Nel file `ternary-tree.h` troverete la descrizione della struttura dati e i prototipi delle tre funzioni da implementare. **Non dovete modificare questo file!**. Questo file contiene:

```
typedef int Elem;

struct terNode{
  Elem v1;
  Elem v2;
  terNode *left;
  terNode *middle;
  terNode *right;
};

typedef terNode *terTree;
```

```
const terTree emptyTerTree=nullptr;

/*************************************************/
/* Funzione da implementare                      */
/*************************************************/

//Es 1
//Ritorna il numero di elementi (positivi) nell'albero
unsigned int nbElem(const terTree&);

//Es 2
//Inserisce un nuovo elemento nell'albero se questo elemento e' positivo
//e non e' gia' presente nell'albero
void insert(Elem, terTree&);

//Es 3
//Ritorna una string con il valore (diverso da -1) degli elementi nel albero
//Separate da spazi e ordinate in ordine crescente
//La string finisce con uno spazio
std::string printElem(const terTree&);
```

Alla fine di questo documento trovate degli esempi di alberi ternari di ricerca. **Per ogni nodo, abbiamo messo a sinistra il valore `v1` e a destra il valore `v2` e non abbiamo rappresentato i figli uguali a `nullptr`.**

2 Esercizio 1

Nel file `es1.cpp`, dovete implementare la funzione `unsigned int nbElem(const terTree& tr)`. Questa funzione ritorna il numero di elementi nell'albero `tr` che sono diversi da `-1`.
Esempi di esecuzione della funzione sugli alberi dati alla fine di questo documento:

- `nbElem(tr1)` => 1
- `nbElem(tr2)` => 2
- `nbElem(tr3)` => 3
- `nbElem(tr4)` => 4
- `nbElem(tr5)` => 5
- `nbElem(tr12)` => 11

Per testare questa funzione, potete usare il file `es1-test.o` compilando con l'istruzione
`g++ -std=c++11 -Wall es1.cpp es1-test.o -o es1-test`.
In questi test, quando stampiamo un albero, stampiamo per ogni nodo, i suoi elementi seguiti dalla stampa dei suoi alberi figli in ordine `left`, `middle` e `right` (se non sono tutti uguali a `nullptr`). Ogni volta che scendiamo nell'albero, aggiungiamo un asterisco `*` prima di stampare i diversi valori. Ad esempio, la stampa dell'albero `tr12` dato alla fine di questo documento è la seguente:

```
 v1:3 v2:10
 *v1:1 v2:2
 **v1:0 v2:-1
 **nullptr
 **nullptr
 *v1:5 v2:8
 **v1:4 v2:-1
 **nullptr
 **nullptr
 *v1:12 v2:20
 **nullptr
 **v1:15 v2:-1
 **nullptr
```

3 Esercizio 2

Nel file `es2.cpp`, dovete implementare la funzione `void insert(Elem el, terTree& tr)`. Questa funzione inserisce nell'albero l'elemento `el` se `el` è superiore o uguale a `0` e se non è presente nel albero, altrimenti la funzione non fa nulla. Ovviamente l'albero prodotto da l'inserimento deve rispettare le regole degli alberi ternari di ricerca e che l'inserimento non sposta il valore già presente in altri nodi (può spostare il valore nel campo `v1` di un nodo al campo `v2` dello stesso nodo)
Esempi di esecuzione della funzione con gli alberi dati alla fine di questo documento:

- `insert(3,tr0)` con `tr0` uguale `nullptr` cambia `tr0` in `tr1`
- `insert(3,tr1)` non cambia `tr1`
- `insert(10,tr1)` cambia `tr1` in `tr2`
- `insert(2,tr2)` cambia `tr2` in `tr3`
- `insert(1,tr3)` cambia `tr3` in `tr4`
- `insert(0,tr4)` cambia `tr4` in `tr5`
- `insert(5,tr5)` cambia `tr5` in `tr6`
- `insert(8,tr6)` cambia `tr6` in `tr7`
- `insert(12,tr6)` cambia `tr6` in `tr8`
- `insert(8,tr8)` cambia `tr8` in `tr9`
- `insert(4,tr9)` cambia `tr9` in `tr10`
- `insert(20,tr10)` cambia `tr10` in `tr11`
- `insert(15,tr11)` cambia `tr11` in `tr12`

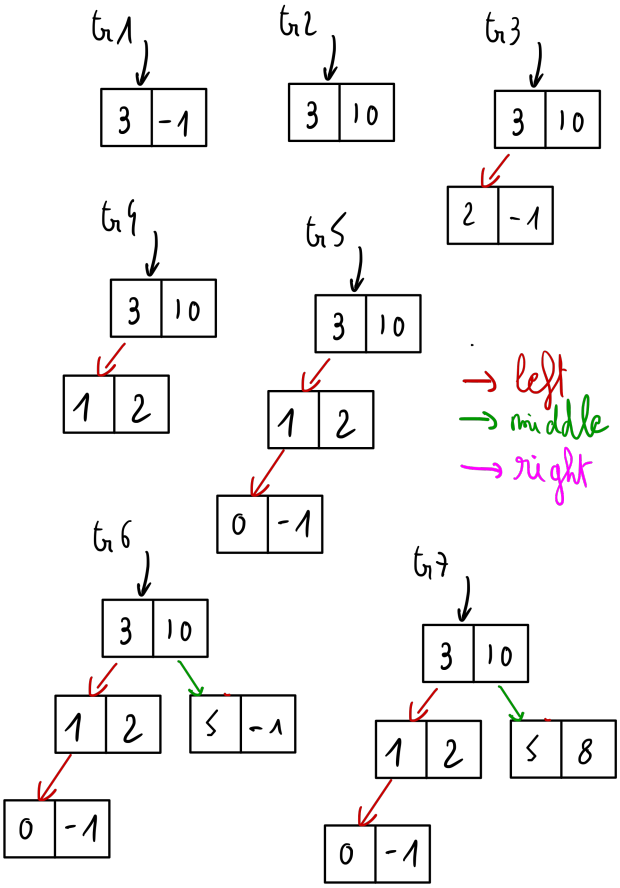
Per testare questa funzione, potete usare il file `es2-test.o` compilando con l'istruzione:
`g++ -std=c++11 -Wall es2.cpp es2-test.o -o es2-test`.

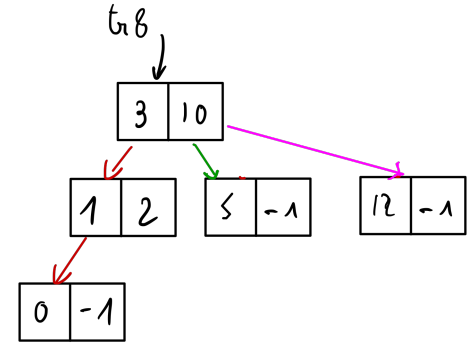
4 Esercizio 3

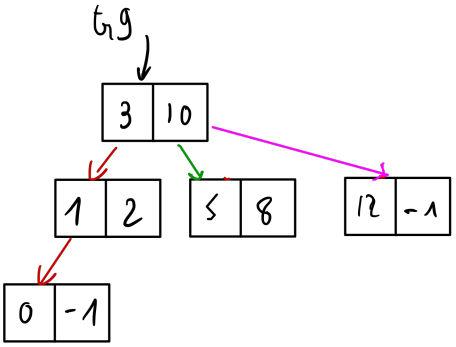
Nel file `es3.cpp`, dovete implementare la funzione `std::string printElem(const terTree& tr);`. Questa funzione ritorna una string con gli elementi diversi da `-1` presenti nell'albero, ordinati in ordine crescente e separati da spazi (c'è anche uno spazio dopo l'ultimo valore).
Esempi di esecuzione della funzione con gli alberi dati alla fine di questo documento:

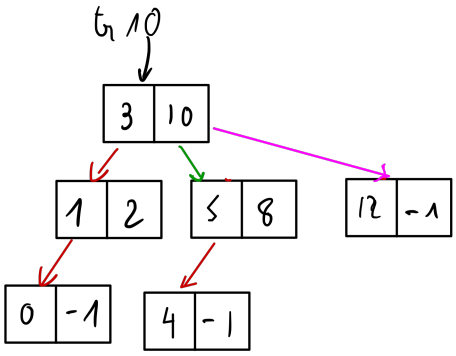
- Se `tr0=nullptr`, abbiamo `printElem(tr0)` => `""`
- `printElem(tr1)` => `"3 "`
- `printElem(tr2)` => `"3 10 "`
- `printElem(tr3)` => `"2 3 10 "`
- `printElem(tr4)` => `"1 2 3 10 "`
- `printElem(tr5)` => `"0 1 2 3 10 "`
- `printElem(tr6)` => `"0 1 2 3 5 10 "`
- `printElem(tr7)` => `"0 1 2 3 5 8 10 "`
- `printElem(tr8)` => `"0 1 2 3 5 10 12 "`
- `printElem(tr9)` => `"0 1 2 3 5 8 10 12 "`
- `printElem(tr10)` => `"0 1 2 3 4 5 8 10 12 "`
- `printElem(tr11)` => `"0 1 2 3 4 5 8 10 12 20 "`
- `printElem(tr12)` => `"0 1 2 3 4 5 8 10 12 15 20 "`

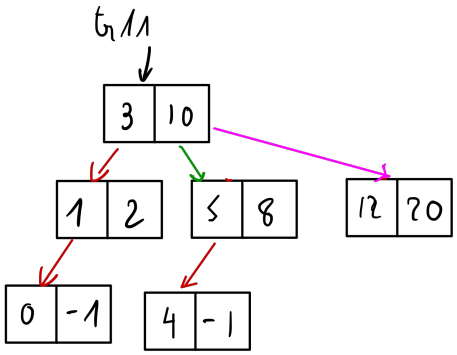
Per testare questa funzione, potete usare il file `es3-test.o` compilando con l'istruzione:
`g++ -std=c++11 -Wall es3.cpp es3-test.o -o es3-test`.
E' possibile usare la funzione `string to_string(int)` per tradurre un intero in string.

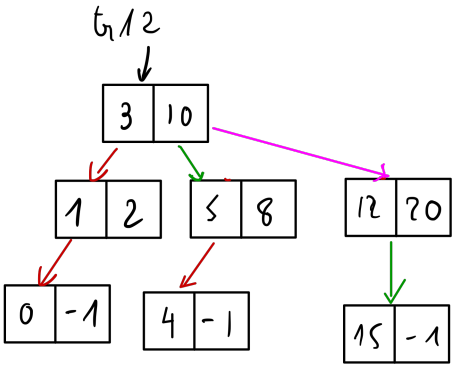

tr1
3 -1
tr2
3 10
tr3
3 10
2 -1
tr4
3 10
1 2
tr5
3 10
1 2
0 -1
→ left
→ middle
→ right
tr6
3 10
1 2
5 -1
0 -1
tr7
3 10
1 2
5 8
0 -1


tr8
3 10
1 2
5 -1
12 -1
0 -1


tr9
3 10
1 2
5 8
12 -1
0 -1


tr10
3 10
1 2
5 8
12 -1
0 -1
4 -1


tr11
3 10
1 2
5 8
12 20
0 -1
4 -1


tr12
3 10
1 2
5 8
12 20
0 -1
4 -1
15 -1