ASD a.a. 2023/24 - Esame del 16 Luglio 2024

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).

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 è di programmare delle funzioni per un tipo di albero estendo i Binary Search Tree (BST) visti a lezione per registrare delle coppie di interi. L'idea è di avere due livelli di BST, il primo livello servirà a registrare il primo valore della coppia e in ogni nodo del primo livello, c'è un puntatore verso un altro BST (di secondo livello dunque) per ricordare il secondo valore della coppia. L'albero vuoto sarà rappresantato da `nullptr`. In un albero, non c'è ripetizione di valore, ed ogni coppia di interi è presente solo una volta. Nel albero di primo livello e nei alberi di secondo livello, vogliamo, come nei BST che se si seleziona un nodo allora tutti gli nodi alla sua sinistra hanno un valore strettamente più piccolo e tutti i nodi alla sua destra hanno un valore strettamente più grande.
La scelta di questa rappresentazione per salvare coppie di interi è dovuta all'ordine che consideriamo su queste coppie. Date due coppie di interi (a, b) e (c, d), supponiamo che (a, b) è più piccola di (c, d) se e solo se $a < b$ oppure $a = b$ e $c < d$, cioé o il primo intero è più piccolo o se i due primi elementi sono uguali, il secondo elemento deve essere più piccolo.
Per esempio, vi diamo la lista degli alberi presenti alla fine di questo documento e l'insieme ordinate di coppie di interi che rappresentano:

- l'insieme vuoto è rappresentato dall'albero `nullptr`
- `pbst1` rappresenta l'insieme `{(4,2)}`
- `pbst2` rappresenta l'insieme `{(4,1),(4,2)}`
- `pbst3` rappresenta l'insieme `{(4,1),(4,2),(4,6)}`
- `pbst4` rappresenta l'insieme `{(4,1),(4,2),(4,4),(4,6)}`
- `pbst5` rappresenta l'insieme `{(2,3),(4,1),(4,2),(4,4),(4,6)}`
- `pbst6` rappresenta l'insieme `{(2,3),(4,1),(4,2),(4,4),(4,6),(6,4)}`
- `pbst7` rappresenta l'insieme `{(3,6),(4,2),(5,7),(5,9),(5,11),(6,2),(6,4),(6,5),(6,9),(8,2),(8,8)}`
- `pbst8` rappresenta l'insieme `{(3,6),(4,2),(5,3),(5,7),(5,9),(5,11),(6,2),(6,4),(6,5),(6,9),(8,2),(8,8)}`
- `pbst9` rappresenta l'insieme `{(3,6),(4,2),(5,3),(5,7),(5,9),(5,11),(6,2),(6,4),(6,5),(6,9),(7,1),(8,2),(8,8)}`

In questi esempi, quando un figlio non è rappresentato, vuol dire che il suo valore è `nullptr`.
Nel file `intpair-bst.h` troverete la descrizione della struttura dati e i prototipi delle tre funzione da implementare. **Non dovete modificare questo file!**. Questo file si presenta cosi:

```
typedef int Elem;

struct intpair{
  Elem v1;
  Elem v2;
};

const intpair noPair={-1000000,-1000000};

struct pairBstNode2{
  Elem el;
  pairBstNode2 *left2;
  pairBstNode2 *right2;
};

struct pairBstNode{
  Elem el;
  pairBstNode2 *bst2;
  pairBstNode *left1;
  pairBstNode *right1;
};

typedef pairBstNode *pairBstTree;

typedef pairBstNode2 *pairBstTree2;

const pairBstTree emptyPairBstTree=nullptr;

const pairBstTree2 emptyPairBstTree2=nullptr;

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

//Es 1
//Ritorna il numero di coppie nel albero
unsigned int nbPairs(const pairBstTree&);

//Es 2
//Inserisce una nuova coppia di interi nell'albero
//Ritorna false, se la coppia e' gia' presente e in questo caso non la inserisce
//Altrimenti ritorna true
bool insertPair(intpair, pairBstTree&);

//Es 3
//Ritorna la coppia piu' piccola dell'albero
//Se l'albero e' vuoto, ritorna noPair
intpair minPair(const pairBstTree&);
```

2 Esercizio 1

Nel file `es1.cpp`, dovete dare l'implementazione della funzione `unsigned int nbPairs(const pairBstTree& pbst)`. Questa funzione ritorna il numero di coppie nel albero `pbst`.
Esempi con gli alberi dati alla fine di questo documento:

- `nbOccurrence(pbst1)` => 1
- `nbOccurrence(pbst4)` => 4
- `nbOccurrence(pbst6)` => 6
- `nbOccurrence(pbst8)` => 12

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 tests, quando stampiamo un albero, stampiamo per ogni nodo il suo elemento, poi il suo albero di secondo livello (se è un nodo del albero di primo livello), poi la stampa dei suoi figli. Ogni volta che scendiamo nell' albero, aggiungiamo una stella prima di stampare i diversi valore per l'albero di primo livello e il simbolo + per gli alberi di secondo livello. Quando i due figli valgono `nullptr` non li stampiamo. Ad esempio, la stampa dell'albero `pbst6` dato alla fine di questo documento darebbe:

```
4
+2
++1
++6
+++4
+++nullptr
*2
*+3
*6
*+4
```

3 Esercizio 2

Nel file `es2.cpp`, dovete dare l'implementazione della funzione `bool insertPair(pair p, pairBstTree& pbst)`. Questa funzione inserisce una nuova coppia di interi `p` nell'albero `pbst` se non è già presente, in questo caso modifica l'albero e ritorna `true` altrimenti ritorna `false` (e non modifica l'albero). L'inserimento, si fa inserendo il primo valore nell'albero di primo livello (se non è presente) come si fa dal solito nei BST, poi inserendo il secondo valore nell'albero di secondo livello legato al nodo dove si trova il primo valore, anche li usando lo stesso metodo che per i classici BST.
Esempi con gli alberi dati alla fine di questo documento:

- `insertPair(p,pbst1)` con `p` che codifica la coppia $(4,2)$ non cambia `pbst1` e ritorna `false`
- `insertPair(p,pbst1)` con `p` che codifica la coppia $(4,1)$ cambia `pbst1` in `pbst2` e ritorna `true`
- `insertPair(p,pbst2)` con `p` che codifica la coppia $(4,6)$ cambia `pbst2` in `pbst3` e ritorna `true`
- `insertPair(p,pbst3)` con `p` che codifica la coppia $(4,4)$ cambia `pbst3` in `pbst4` e ritorna `true`
- `insertPair(p,pbst4)` con `p` che codifica la coppia $(2,3)$ cambia `pbst4` in `pbst5` e ritorna `true`
- `insertPair(p,pbst5)` con `p` che codifica la coppia $(6,4)$ cambia `pbst5` in `pbst6` e ritorna `true`
- `insertPair(p,pbst7)` con `p` che codifica la coppia $(8,2)$ non cambia `pbst7` e ritorna `false`
- `insertPair(p,pbst7)` con `p` che codifica la coppia $(5,3)$ cambia `pbst7` in `pbst8` e ritorna `true`
- `insertPair(p,pbst8)` con `p` che codifica la coppia $(7,1)$ cambia `pbst8` in `pbst9` e ritorna `true`

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

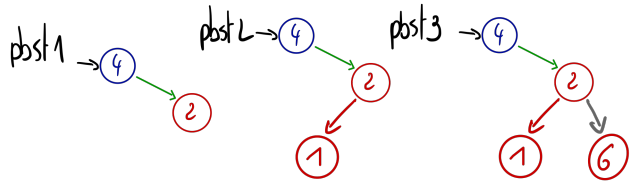
Nel file `es3.cpp`, dovete dare l'implementazione della funzione `pair minPair(const pairBstTree& pbst)`. Questa funzione ritorna la coppia di interi più piccola dell'albero `pbst`. Se l'albero è vuoto (uguale a `nullptr`), la funzione ritorna `noPair`.
Esempi con gli alberi dati alla fine di questo documento:

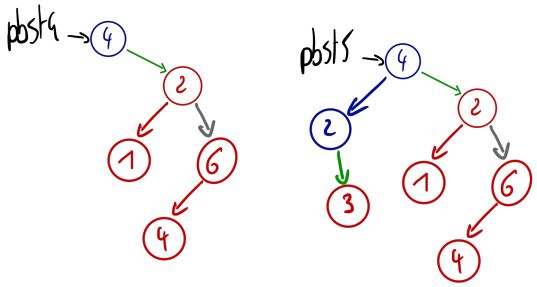
- `minPair(pbst1)` ritorna `{4,2}`
- `minPair(pbst2)` ritorna `{4,1}`
- `minPair(pbst3)` ritorna `{4,1}`
- `minPair(pbst4)` ritorna `{4,1}`
- `minPair(pbst5)` ritorna `{2,3}`
- `minPair(pbst6)` ritorna `{2,3}`
- `minPair(pbst7)` ritorna `{3,6}`

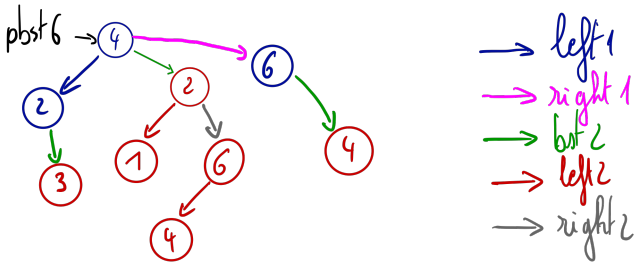
Per testare questa funzione, potete usare il file `es3-test.o` compilando con l'istruzione:
`g++ -std=c++11 -Wall es3.cpp es3-test.o -o es3-test`.

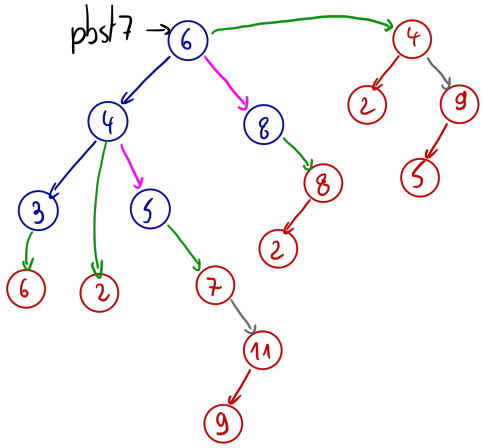
5 Consegna

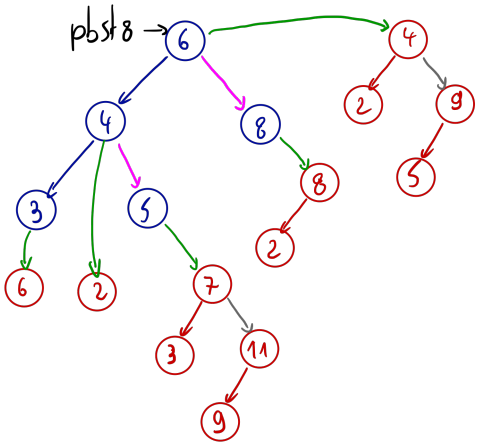
Per la consegna, creare uno zip con tutti i file forniti.

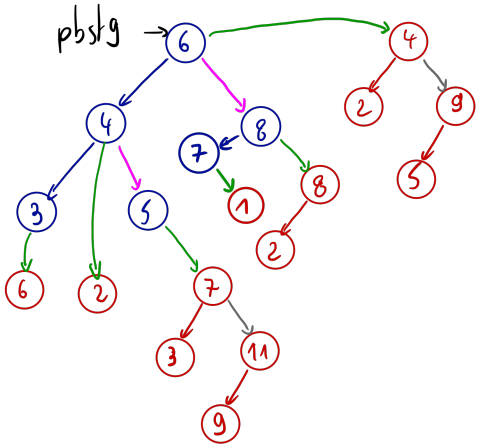

bst 1
bst 2
bst 3


bst 4
bst 5


bst 6
left 1
right 1
bst 2
left 2
right 2


bst 7


bst 8


pbstg
6
4
8
4
7
3
5
8
1
2
6
2
7
3
11
9
4
2
9
5