Relazione progetto C++
Aprile 2020

Nome: Farjad
Cognome: Ali
Matricola: 829940
Mail: f.ali1@campus.unimib.it

PROGETTO CONSOLE

Introduzione:

L'analisi del problema ha portato alla conclusione che era necessaria una struttura ad albero binario che avesse le proprieta' BST (dato un nodo, a sinistra tutti valori inferiori, a destra tutti quelli superiori) con particolarità di non supportare valori duplicati. Poichè la classe deve essere resa generica, l'implementazione dovrà fare uso del tipo generico e di conseguenza dei template. Dato che la struttura deve poter garantire l'integrità dei dati salvati, tutti gli elementi inseriti vengono copiati e mantenuti in memoria fino all'eliminazione della struttura stessa.

Dati membro:

Padre: puntatore al padre, necessario per poter interare in maniera corretta senza usare strutture di supporto. La ricorsione non è possibile poichè è necessario iterare sull'albero passo per passo.
Dato: puntatore al dato di tipo generico T che è salvato nel nodo.
Elementi: puntatore ad un *long* che conta gli elementi figli di un nodo includendo se stesso.
Figlio destro: puntatore al figlio destro del nodo.
Figlio sinistro: puntatore al figlio sinistro del nodo.
Comparatore statico: è il metodo di confronto usato dall'albero per poter effettuare operazioni di inserimento e ricerca. Restituisce >0 se A>B, <0 se A<B, = se A==B. Viene passato come parametro del template ed è statico perchè bastava una istanza per tutte le istanze dell'albero.

Implementazione BST:

Dato che era stato richiesto un albero per elementi generici, è stato necessario l'uso dei template per poter rendere generici sia il tipo (tipo T) che il metodo di confronto (Comparator). L'utente puo' anche non specificare il metodo di confronto se il tipo di dato passato ha delle operazioni definite per l'operatore < e >. La classe infatti predispone di un comparatore di default per l'ordine crescente (restituisce 1 se A>B, -1 se A<B, 0 altrimenti).
La classe dispone di un costruttore a zero parametri che inizializza un BST vuoto, un distruttore che dealloca tutti i dati puntati dai membro della classe . Per facilitare le operazioni di inserimento e per rendere leggibile il codice è stato predisposto un costruttore privato che crea un nuovo nodo prendendo in parametro il dato da salvare ed il padre del nuovo nodo. Il costruttore non è pubblico perchè potrebbe essere invocato per compromettere le proprietà della struttura dati (Ad esempio inserendo come figlio sinistro un numero maggiore).

Metodi per le operazioni:

`print:` prende come parametro una std::ostream e restituisce la stream su cui è stato stampato il contenuto iterato della struttura dati. Nel caso non fosse specificato allora il parametro di default è std::cout. E' stata effettuata un'implementazione del genere tenendo conto che l'utente vorrebbe stampare su file.

`insert:` crea un copia dell'elemento fornito come parametro e lo salva sull'albero. Viene sfruttata la ricorsione per l'inserimento: in base al risultato del comparatore il dato finira a destra o a sinistra fino a quando eventualmente non verrà raggiunto il livello dei nodi foglia. Nel caso si provi ad inserire un elemento duplicato, la funzione lancerà un'eccezione BSTDoubleElementException (implementazione troppo banale per mettere in un file separato).

`exists:` navigazione della struttura ad albero uguale a quella della funzione insert, se viene raggiunto il livello dei nodi foglia e l'elemento allora viene restituito false.

subtree: navigazione della struttura uguale a quella della funzione insert, se trova il nodo allora crea una copia del sottoalbero che ha come radice il nodo trovato e ne restiuisce il puntatore. Nel caso non venga trova l'elemento allora viene restituito un nullptr. Aveva poco senso restituire sempre reference, specialmente reference di alberi vuoti per gli elementi non trovati. Usare eccezioni nel caso non si trovasse l'elemento sembrava eccessivo e senza senso.

`printIF:` funzione quasi identica alla print, la peculiarità sta nel fatto che è templata con due variabili: il tipo T della struttura iterabile ed il predicato che l'elemento iterato deve soddifare per essere stampato su schermo. L'implementazione ha cercato di evitare più di due parametri di template per leggibilità e per renderla utilizzabile da altre classi iterabili. Si suppone che il tipo T sia iterabile.

`overloading <<:` overloading al quanto semplice che richiama la funzione print della classe con parametro il parametro dell'operatore.

`size:` figli totali del nodo incluso se stesso.

Implementazione iteratore: *ConstBSTForwardIterator*

L'iteratore è di tipo solo lettura e forward. Sono stati definiti i traits STL. L'iterazione senza struttura di supporto su un albero binario richiede due cose: un puntatore al padre per tornare indietro ed una variabile booleana di supporto.

`next:` e' la funzione che itera in order sugli elementi dell'albero, prima effettua una visita in profondita' verso il figlio leftmost, successivamente sale dal padre ed infine scende dal figlio destro. La variabile boolean serve al metodo per capire se deve scendere in profondità a cercare il figlio left most o passare al prossimo ramo.

Main:

Testa tutti i metodi pubblici e meccanismi di copia e stampa attraverso una serie di test, questi test sono specificati come funzioni booleane. Dentro le funzioni ci sono delle asserzioni che se non vengono rispettate fanno ritornare false alla funzione. Questo si è rivelato il modo più facile per testare separatamente le funzionalità e trovare eventuali falle.

Vengono testati inserimenti (test1); i controlli di presenza di un elemento (test2); l'overloading dell'operatore, la stampa con print e la printIF (test3); subtree (test4) ed un test misto con classi (test5).

PROGETTO QT

Interfaccia:

Il programma è strutturato a tab perchè rendono l'interfaccia user friendly. All'avvio il programma prova a scaricare i due file. Appena finiti i download abilita le 5 tab di visualizzazione, due per le liste dei artisti delle rispettive etichette discografiche. Le liste sono state implementate con l'uso della tabella perchè le più facili da utilizzare. Ovviamente i file sono stati processati per estrarre il nome e ricostruire il link dove possibile. Per ogni etichetta ci sono diagrammi a barre che mostra il numero degli artisti il cui nome inizia con una certa lettera. L'ultima tab di visualizzazione mostra un diagramma a barre dove sono mostrati gli artisti per etichetta. La sesta tab è la tab di download in cui possiamo trovare due barred di stato per i rispettivi file (per connessioni lente e consentire all'utente di seguire il progresso del download), una textbox per messaggi dal programma (inclusi errori di rete ed SSL) ed infine un bottone per riscaricare da capo i file (nel caso si volesse aggiornare).

Scelte implementative:

Per conteggiare il numero di artisti in base all'iniziale del nome sono state usate delle mappe fornite da QT. Per evitare di vedere barre che toccano il limite il valore massimo di tutti i grafici calcolato prendendo il valore più alto della serie di dati ed incrementandolo del 10%.
Nel caso il link per un artista non fosse disponibile, al posto del link nella lista troviamo la frase "Link non disponibile". La dimensione minima del programma è dettata dai limiti di visualizzazione dei grafici, dimensioni troppo piccole infatti non permettono una visualizzazione corretta dei dati nei grafici. Il programma è svincolato per quanto riguarda le dimensioni massime.