Aprile 2020

Cognome: Armanini
Nome: Justin
Matricola: 830103
Mail: j.armanini@campus.unimib.it

Relazione Progetto C++

Implementazione di un albero binario di ricerca templato

Analisi:

L'albero, al di là della definizione prettamente formale (grafo non orientato, connesso e aciclico), può essere visto come un insieme di nodi connessi secondo una gerarchia padre-figlio.
Tre sono le caratteristiche fondamentali dell'albero:

- Binario: ogni nodo ha grado massimo di uscita pari a 2;
- Di ricerca: per ogni nodo x, il sottoalbero di sinistra deve essere composto da nodi le cui chiavi sono minori di quella del nodo x, il sottoalbero di destra da nodi le cui chiavi sono maggiori di quella del nodo x, e infine sia il sottoalbero di sinistra che quello di destra devono essere essi stessi alberi binari di ricerca;
- I valori contenuti nei nodi sono elementi generici di tipo T.

Tipi di dati:

Sulla base di quanto concluso in fasi di analisi, l'entità nodo è stata rappresentata tramite una struttura dati eterogenea (struct), per poter rappresentare:

- Valore: l'elemento di tipo T contenuto nel nodo;
- Figlio sinistro: puntatore al nodo che è figlio sinistro del nodo corrente;
- Figlio destro: puntatore al nodo che è figlio destro del nodo corrente;
- Genitore: puntatore al nodo genitore di quello corrente. Quest'ultimo dato è stato reso necessario dalla richiesta di implementare un iteratore associato all'albero, in modo da rendere più facile l'attraversamento dell'albero senza ricorrere alla ricorsione. A meno di questa richiesta non sarebbe stato necessario.

Implementazione binary_search_tree:

Siccome nella traccia si parla di elemento generico T, è stato necessario rendere la classe templata. A sua volta questo ha richiesto l'introduzione altri due parametri template per il confronto degli elementi: uno per l'uguaglianza e uno per l'ordinamento. Pertanto l'inizializzazione di un albero binario di ricerca richiede non solo il tipo T degli elementi contenuti nei nodi, ma anche di altri due per poter inizializzare i due funtori necessari al confronto. In questo modo è stata soddisfatta la richiesta della traccia di permettere all'utente di "scegliere la strategia usata per confrontare due dati T".

Per quanto riguarda i metodi fondamentali, è presente un solo costruttore, quello di default, che inizializza l'albero vuoto, di dimensione pari a zero. Per gli altri metodi fondamentali si veda il paragrafo successivo.

Implementazione metodi:

helper_copy_tree

Metodo helper usato per copiare l'albero radicato in x in *this. La scelta di creare questo metodo è stata dettata dal fatto che fosse utile in due casi: sia per il costruttore di copia, in cui si copia l'intero albero passato come parametro, per cui si passa come argomento la radice di quest'ultimo, sia per il metodo `subtree`, per

creare una copia del sottoalbero radicato nel nodo che contiene l'elemento x, per cui prende come argomento quel particolare nodo.

`remove_helper`
Metodo helper usato solamente dal distruttore per disallocare dalla memoria l'intero albero. È stato necessario introdurlo per poter sfruttare la ricorsione.

`find_helper`
Metodo helper per ottenere il puntatore al nodo che contiene l'elemento x. Anche questo metodo è stato creato perché utile in due casi: per verificare l'esistenza di un elemento nell'albero (vedi `contains`), e per ottenere la radice del sottoalbero da copiare nel metodo `subtree`.

`print_helper`
Metodo helper usato solamente da `print` per poter sfruttare la ricorsione.

`size`
Metodo che ritorna il numero di nodi dell'albero, per soddisfare la richiesta n.1 della traccia. Anziché contare ogni volta i nodi attraversando l'albero, il che potrebbe essere un'operazione onerosa, restituisce l'attributo di classe `_size` che viene aggiornato ad ogni inserimento.

`insert`
Metodo fondamentale perché l'albero rispetti la proprietà di essere di ricerca. Per ogni nuovo elemento esso scende l'albero scegliendo il ramo sinistro o destro a seconda che l'elemento da inserire sia minore o maggiore di quello contenuto nel nodo corrente. Data la specifica che non permette l'inserimento di duplicati esso lancia un'eccezione personalizzata (`duplicated_value`) nel caso l'utilizzatore della classe tenti di farlo. Questa eccezione è derivata dall'eccezione `logic_error` perché si tratta di un errore nella logica del programma. A sua volta `logic_error` deriva dalla classe exception, per cui si tratta di eccezioni standard.

`contains`
Metodo che risponde alla richiesta n.2 della traccia. Come già accennato sfrutta il metodo `find_helper`.

`subtree`
Metodo che risponde alla richiesta n.5 della traccia. Sfrutta due metodi helper: `find_helper` per trovare la radice del sottoalbero e `helper_copy_tree` per copiarlo in uno nuovo, inizialmente vuoto, che viene poi ritornato dal metodo. La scelta di ritornare una copia del sottoalbero è data dalla frase "ritorna un nuovo albero".

`print`
Metodo che stampa l'albero sfruttando `print_helper`. Questo metodo, apparentemente ridondante vista la ridefinizione dell'operatore <<, è stato implementato per via della frase "anche usando `operator<<`".

Iteratore

Il punto n.3 della traccia richiede l'implementazione di un iteratore di tipo forward a sola lettura, quindi costante. Siccome la scelta dell'ordine con cui ritornare i dati è libera, è stata scelta la visita in order, per risaltare il fatto che si tratti di un albero binario di ricerca.

Per poter implementare l'operatore di incremento (pre e post) è stato creato un metodo `next_helper` che ad ogni invocazione si muove di un nodo attuando la visita in order dell'albero. In questa maniera si è evitato l'uso di una struttura di supporto

Implementazione / Ridefinizione funzioni globali:

`operator<<`
Stampa l'albero utilizzando l'iteratore associato. Vista la scelta della visita in order, gli elementi sono stampati in ordine crescente, il che è utile per vertificare il corretto funzionamento della `insert`.

printIF

Funzione che stampa gli elementi dell'albero che soddisfano il predicato P. Oltre ai parametri template che corrispondono a quelli dell'albero passato come parametro, se ne aggiunge uno che indica il predicato da applicare. Risponde all'ultima specifica della traccia.

Nella traccia si legge che "dato un albero binario di tipo T, e un predicato P, [la funzione] stampa…", facendo venire il dubbio che sia l'albero che il funtore predicato debbano essere passati come parametri della funzione.
Tuttavia questo richiederebbe di istanziare l'oggetto funtore nel main, per poi passarlo come parametro alla funzione. Per questo è stato scelto di usare come parametro formale della funzione solamente l'albero, mentre il tipo del predicato da utilizzare è stato messo come parametro template. In particolare il tipo del predicato P è stato messo come primo parametro template, in quanto è l'unico che non è deducibile automaticamente dal compilatore, mentre gli altri, appunto, sono dedotti automaticamente dal tipo dell'albero passato come parametro.

Main:

Semplicemente si occupa di testare tutti i metodi della classe binary_search_tree, più operator<< ridefinito e printIF, passando come parametro template sia un tipo built-in (int) che uno custom (point). Per entrambi i casi viene fatta la distinzione tra un'istanza di binary_search_tree costante e non.

Relazione Progetto Qt

Applicazione visuale

Scelte implementative

Siccome nel punto n.1 della traccia è richiesto che "vengano scaricati i file" l'applicazione, ancora prima di inizializzare la parte grafica, si occupa di scaricare il contenuto dei due link e memorizzarlo in due file .txt distinti. In seguito il contenuto dei due file è caricato in due vettori di tuple, in modo da avere i dati immediatamente disponibili, limitando la lettura dei file a una volta soltanto durante la vita dell'applicazione. Ogni tupla contiene una stringa per l'url della pagina Wikipedia e una per il nome dell'artista.

Il conteggio degli artisti per lettera è stato fatto con un piccolo accorgimento: nel caso di quelli con l'articolo "The" iniziale la prima lettera utile al conteggio è quella del nome dopo l'articolo.
ES: "The Weeknd" è stato conteggiato sotto la lettera "w", anziché "t".

Ragionamento analogo è stato fatto per l'ordinamento, per cui l'articolo "The" non è stato preso in considerazione.

Scelte di design

L'interfaccia è stata organizzata in 3 schede, una per le informazioni relative agli artisti Universal, una per quelli EMI e l'ultima per la comparazione tra il numero di artisti delle due etichette.

Le prime due sono identiche per quanto riguarda la struttura: l'elenco degli artisti a sinistra, mentre il grafico per il conteggio degli artisti per ogni lettera a destra. L'idea di accorpare elenco e grafico è data dalla volontà di rendere l'interfaccia il più semplice e meno dispersiva possibile.

Il punto n.3 della consegna è stato soddisfatto non inserendo “in chiaro” il link alla pagina web, ma rendendo il nome di ciascun artista inserito in elenco esso stesso un link cliccabile. Questo perché inserire il link in forma visibile (generalmente abbastanza lungo) avrebbe ridotto lo spazio disponibile per visualizzare il grafico. Ma anche perché, immaginando l’utilizzo dell’applicazione da parte di un utente “normale”, è stato ritenuto inutile e ridondante visualizzare il link esplicitamente.

Note

Utilizzando la versione 5.9.9 di Qt si ottiene il messaggio di warning: “QNetworkReplyHttpImplPrivate::_q_startOperation was called more than once QUrl” che cercando su internet risulta essere un bug.