Relazione progetto C++

Aprile 2020

Nome: Alessandro

Cognome: Gallizzi

Matricola: 830104

E-mail: a.gallizzi@campus.unimib.it

Tempistiche: Dal 20/03/2020 al 10/04/2020

Introduzione

Dopo aver valutato il problema proposto, ho deciso di rappresentare un singolo valore dell'albero binario di ricerca come una struct *nodo* contente come dati: un valore generico T, un puntatore al nodo sinistro, un puntatore al nodo destro e un puntatore al nodo padre. In aggiunta ho scelto di inserire un puntatore al nodo successivo.

Tipi di dati

Il tipo di dato scelto è un nodo che, come specificato nella parte introduttiva, è una struct contenente cinque dati:

- **Value** di tipo generico T. Contiene il valore che è stato assegnato ad un nodo dell'albero binario di ricerca.
- **left** di tipo puntatore node. È il puntatore al nodo sinistro;
- **right** di tipo puntatore node. È il puntatore al nodo destro;
- **p** di tipo puntatore node. È il puntatore al nodo parent;
- **next** di tipo puntatore node. È il puntatore al nodo successivo (si intende per ordine di inserimento).

La struct *node* è dichiara nel file bstree.h. Il campo next non sarebbe logico implementarlo nel contesto degli alberi binari di ricerca, ma ho deciso per questa soluzione per facilitare l'implementazione dell'iteratore forward data l'impossibilità di usare una struttura che permettesse la memorizzazione del cammino.

L'unico costruttore inizializzato è quello in cui viene passato per parametro un valore generico di tipo T. È stata scelta questa implementazione perché sarebbe inutile la creazione di un nodo senza valore alcuno.

Implementazione

La classe bstree contiene come attributi:

- **_root** di tipo puntatore node. È il puntatore alla radice dell'albero
- **_size** di tipo unsigned int. È la dimensione dell'albero, quindi il numero di nodi inseriti.

Oltre i fondamentali costruttore di default, costruttore di copia, distruttore e operatore d'assegnamento; ho deciso di implementare un costruttore privato che, dato un puntatore a node come parametro, costruisce un albero da esso.

Il costruttore di default inizializza l'albero con la dimensione 0, quindi con radice nulla.

Il costruttore di copia e l'operatore di assegnamento (come anche il costruttore privato) si servono di una funzione privata **copy_helper** per copiare la dimensione della lista come la struttura albero con i puntatori memorizzati in modo ricorsivo a partire dalla radice e quindi per i figli sinistri e destri.

Il distruttore libera ogni allocazione di memoria.

Metodi implementati

I metodi implementati sono i seguenti:

- **insert** che permette l'inserimento di un nodo in un albero binario di ricerca. Il parametro che viene passato è un dato generico di tipo T. L'inserimento viene fatto solamente nel caso, come da richiesta, in cui nell'albero non sia già presente un nodo col valore specificato. Il metodo inoltre chiama un metodo privato insert_next per la creazione di una lista di nodi che sono gli stessi inseriti nell'albero, ma di più facile reperibilità dall'iteratore;
- **clear** metodo che svuota l'albero dai suoi nodi, quindi setta la dimensione dell'albero a 0. Viene chiamato quando invocato dal distruttore oppure quando viene generata un'eccezione di allocazione di memoria. Si serve del metodo privato **clear_helper** per svuotare l'albero a partire dalla radice;
- **size** metodo unsigned int che permette di ritornare la dimensione dell'albero, quindi ritorna semplicemente l'attributo privato _size;
- **search** metodo booleano che permette di cercare un determinato nodo in base al valore inserito. Si serve di un metodo privato **search_helper** per la ricorsione dei puntatori left e right. Ritorna true se il nodo è stato trovato;
- **print_inorder** metodo che stampa i valori dei nodi in un albero secondo l'attraversamento inorder. Si serve di un metodo privato **print_inorder_helper** per la ricorsione;
- **print_preorder** metodo che stampa i valori dei nodi in un albero secondo l'attraversamento preorder. Si serve di un metodo privato **print_preorder_helper** per la ricorsione;
- **print_postorder** metodo che stampa i valori dei nodi in un albero secondo l'attraversamento postorder. Si serve di un metodo privato **print_postorder_helper** per la ricorsione;

- **getMax** metodo di tipo generico T che permette di ritornare il valore massimo del nodo inserito nell'albero binario di ricerca. Si serve di un metodo privato **getMax_helper** per la ricorsione. Data la struttura usata, questo si trova inevitabilmente nel nodo più a destra dell'ultimo livello;
- **getMin** metodo di tipo generico T che permette di ritornare il valore minimo del nodo inserito nell'albero binario di ricerca. Si serve di un metodo privato **getMin_helper** per la ricorsione. Data la struttura usata, questo si trova inevitabilmente nel nodo più a sinistra dell'ultimo livello;
- **successor** metodo di tipo generico T che permette di ritornare il successore di un nodo in base al valore passato come parametro. Utilizza un metodo **successor_helper** per l'iterazione. Nel caso si cerchi il successore del valore massimo dell'albero, viene generata un'eccezione *limit_value*;
- **predecessor** metodo di tipo generico T che permette di ritornare il predecessore di un nodo in base al valore passato come parametro. Utilizza un metodo **predecessor_helper** per l'iterazione. Nel caso si cerchi il successore del valore minimo dell'albero, viene generata un'eccezione *limit_value***;**
- **subtree** metodo di tipo bstree che permette di ritornare un sottoalbero a partire dal valore, passato come parametro, di un nodo presente in un albero principale. Viene lanciata un'eccezione *element_not_found* nel caso il nodo da cui partire la generazione del sottoalbero non sia presente nell'albero principale;
- **printif** funzione globale che permette di stampare i valori dei nodi di un albero e un predicato passati come parametri. I nodi che verranno stampati saranno quelli che soddisferanno la condizione del predicato P.

Iteratori

Come da richiesta, è stato implementato un iteratore a sola lettura di tipo forward. Per l'operator++ ho deciso di creare per semplicità il puntatore a node *next,* aiutato anche dal fatto che, da richiesta, non fosse chiesta l'iterazione dei nodi in un ordine specifico.

Main

Nel file main.cpp vengono testati tutti i metodi della classe bstree. È stato fatto uso sia di assert sia chiamate a ogni metodo implementato. È stato deciso di implementare vari metodi all'interno della classe per dividere opportunamente il testing della libreria bstree.h.