Cognome: Scurati

Nome: Luca

Matricola: 844711

Mail: l.scurati1@campus.unimib.it

Relazione Progetto C++ - Febbraio 2021

Implementazione di un MultiSet contenente elementi generici di tipo T

Analisi

Un MultiSet è un insieme di dati che può contenere duplicati. L'ordine in cui questi dati appaiono non è rilevante. Gli elementi del MultiSet possono solo essere aggiunti o rimossi e non possono essere modificati.
La richiesta del progetto è quella di implementare questo MultiSet minimizzando l'uso della memoria, quindi di memorizzare gli elementi una sola volta e non di salvare ogni duplicato di un elemento. Il MultiSet è quindi visto come un insieme di nodi che contengono la coppia <valore, numero di occorrenze>.

Implementazione dei tipi di dati

È stata utilizzata una struttura dati eterogenea (struct) per potere rappresentare un nodo che contiene le seguenti informazioni:

- data: di tipo element<T>. Contiene la coppia <valore, numero di occorrenze>
- next: puntatore al nodo successivo

Ogni elemento data è rappresentato da una classe element che contiene a sua volta le seguenti informazioni:

- _elemento: di tipo generico T. Rappresenta l'effettivo elemento memorizzato nel MultiSet
- _occorrenze: di tipo size_type. Rappresenta il numero di occorrenze dell'elemento all'interno del MultiSet

Implementazione della classe Multiset

Trattandosi di tipo generico T, è necessario rendere la classe MultiSet una classe templata. È necessario quindi introdurre anche un altro parametro template, ovvero il parametro funct, utilizzato per il confronto tra gli elementi di tipo T. Quindi l'inizializzazione della classe MultiSet prevede il tipo T degli elementi e il funtore funct, necessario per il confronto degli elementi.
All'interno di questa classe definiamo, oltre alla struct nodo, utilizzata per rappresentare un singolo nodo nel MultiSet, un nuovo tipo size_type, utilizzato per indicare un dato che indica la dimensione o il numero di occorrenze.
Sono stati anche implementati diversi costruttori e il distruttore, elencati di seguito.

Costruttori e Distruttori classe Multiset

Multiset()

Costruttore di default che inizializza un MultiSet vuoto

Multiset(const Multiset &toCopy)

Costruttore di copia. Prende come argomento un MultiSet e ne crea un altro che contiene gli stessi valori e con lo stesso numero di occorrenze.

template<typename IteratorT> Multiset(IteratorT begin, IteratorT end)

Costruttore secondario. Prende come argomenti due iteratori relativi a una sequenza di dati generici Q. begin punta alla testa della sequenza, mentre end alla fine. Questo costruttore inizializza un MultiSet a partire dai valori restituiti dagli iteratori passati come argomento.

~Multiset()

Distuttore. Richiama il metodo clear() elimina il contenuto del MultiSet.

Metodi classe Multiset

nodo *find(const T &e)

Metodo helper che, dato un elemento di tipo T, ne ritorna il puntatore. Il metodo parte dalla testa del multiset (_head), scorre tutti i nodi e confronta l'elemento e con l'elemento di ogni coppia all'interno del MultiSet. Quando questi elementi sono uguali ritorna il puntatore dell'elemento, mentre ritorna nullptr se non trova corrispondenze.

nodo *find_previous(const nodo *curr)

Metodo helper che, dato un nodo, ritorna il puntatore al nodo precedente. Il metodo si salva la testa e il suo nodo successivo e, fintantochè il nodo successivo è diverso da curr continua a scorrere gli elementi del MultiSet. Quando il nodo successivo è uguale a curr termina la ricerca e restituisce l'elemento corrente, che sarà l'elemento precedente rispetto a quello passato come argomento.

int count(const T &e)

Metodo che restituisce il numero di occorrenze dell'elemento passato come parametro. Richiamando il metodo find() e passandogli come parametro l'elemento e, il metodo si memorizza il puntatore al nodo dell'elemento e. Se il puntatore punta a nullptr vuol dire che e non è presente nel MultiSet e quindi ritorna 0, altrimenti viene ritornato il numero di occorrenze dell'elemento e invocando il metodo occorrenze().

void clear()

Metodo che svuota il MultiSet. Partendo dalla testa (_head), fintanto che la testa non punta a nullptr, ovvero finchè il MultiSet non è vuoto, si salva il puntatore all'elemento successivo di _head

in un puntatore tmp, elimina il nodo _head e poi assegna a _head il puntatore che si è salvato in precedenza (tmp)

void add(const T &e, size_type no)

Metodo che inserisce nel MultiSet un nuovo elemento e di tipo T con no occorrenze. Per prima cosa il metodo controlla che l'elemento e non sia già presente invocando il metodo contains(). Se e non è già presente all'interno del MultiSet allora si controlla che _head punti a nullptr. In questo caso si crea un nuovo nodo _head inserendo l'elemento e e il numero di occorrenze no. Nel caso in cui la testa non punti a nullptr allora il metodo si salva in un nodo temporaneo tmp il riferimento alla testa, crea un nuovo nodo _head inserendo l'elemento e e il numero di occorrenze no e assegna come nodo successivo di _head, il nodo tmp. Infine incrementa _dimensione e aggiunge no al numero di elementi totali (_numeroelementi).
Nel caso in cui l'elemento e sia già presente, viene invocato il metodo find() sull'elemento e per trovare il puntatore al nodo di quell'elemento, si aumentano di no le occorrenze di quell'elemento col metodo setOccorrenze() e si aumenta, sempre di no, il numero di elementi totali (_numeroelementi).

void add(const T &e)

Metodo che inserisce nel MultiSet un nuovo elemento e di tipo T con una sola occorrenza. Viene invocato il metodo add(const T &e, size_type no) descritto in precedenza con parametri e e 1.

void remove(const T &e)

Metodo che rimuove dal MultiSet un'occorrenza dell'elemento e passato come parametro. Per prima cosa il metodo controlla che l'elemento e sia presente all'interno del MultiSet invocando il metodo contains(). In caso negativo viene generata un'eccezione di tipo element_not_found.
In caso positivo invece viene evocato il metodo find() con parametro l'elemento e per salvarsi in tmp il puntatore relativo. Dopodiché, se le occorrenze di e sono maggiori di uno, il numero di occorrenze viene decrementato di 1, invocando il metodo setOccorrenze() e passando come parametro il numero di occorrenze attuali tramite il metodo occorrenze() – 1. Se invece il numero di occorrenze fosse uguale a 1 allora si controlla che l'elemento da eliminare non sia la testa. Nel caso in cui _head fosse uguale a tmp allora _head prende come nuovo valore tmp->next, ovvero il nodo successivo, e si elimina il nodo tmp. Altrimenti viene invocato il metodo find_previous() passando tmp come parametro per memorizzarsi il puntatore dell'elemento precedente in prec. L'elemento successivo di prec diventa quindi l'elemento successivo di tmp, e poi viene eliminato il nodo tmp. Nel caso in cui si vada a rimuovere un intero nodo viene anche decrementata la dimensione del MultiSet (_dimensione). Infine viene decrementato anche _numeroelementi.

void remove(const T &e, size_type no)

Metodo che rimuove dal Multiset no occorrenze dell'elemento e. Viene invocato il metodo remove(const T &e) descritto in precedenza per tante volte quante il numero di occorrenze da rimuovere.

bool contains(const T &e)

Metodo che restituisce true se l'elemento e è presente all'interno del MultiSet, false altrimenti. Per fare ciò invoca la funzione find() passando come parametro e per trovare il puntatore a quell'elemento. Se questo puntatore punta a nullptr vuol dire che l'elemento non è presente e quindi restituisce false. Se invece è diverso da nullptr allora restituisce true.

int getNumeroElementi()

Metodo getter che restituisce il numero di elementi totali del MultiSet, ovvero il valore di _numeroelementi.

int getSize()

Metodo getter che restituisce il numero di nodi totali presenti nel MultiSet, ovvero il valore di _dimensione.

Iteratore

È stato implementato un iteratore di tipo forward di sola lettura, in modo da non poter modificare il contenuto del MultiSet. Il nome della classe è const_iterator.
Questo iteratore permette l'accesso al valore dell'elemento e del relativo nodo. Viene utilizzato dalla ridefinizione dell'operator<< per la stampa dei singoli nodi e anche per la stampa del MultiSet completo.

Ridefinizione operator<<

La ridefinizione dell'operator<< prende come parametro un'ostream e un MultiSet, che viene stampato nella forma <elemento, numero di occorrenze>. Per scorrere da _head alla fine del MultiSet viene usato l'iteratore definito in precedenza.

Main

All'interno del file main.cpp vengono definiti tre funtori utilizzati per la comparazione di due elementi

- compare_int: viene utilizzato per la comparazione tra interi
- compare_string: viene utilizzato per la comparazione tra stringhe

Vengono inoltre definiti due metodi utilizzati per testare le funzionalità implementate e descritte precedentemente. Ogni metodo è relativo a un tipo diverso di dato

- primoTest(): effettua test utilizzando valori interi
- secondoTest(): effettua test utilizzando stringhe

I test eseguiti sono gli stessi per ogni metodo, e sono i seguenti:

- costruttore di default
- metodo add()
- metodo remove()
- operatore di assegnamento (operator=)
- costruttore copia

- operatore di confronto (operator==)
- metodo clear()
- metodo contains()
- costruttore da sequenza dati Q
- output tramite const_iterator
- operatore di stream di output (operator<<)

Tutti gli altri metodi descritti in questa relazione che non vengono esplicitamente testati vengono richiamati all'interno dei vari test eseguiti.