

Cognome: Vismara
Nome: Diego
Matricola: 844796
Mail: d.vismara@campus.unimb.it

Relazione Progetto C++

Implementazione di un MultiSet di elementi generici T

Analisi:

Il MultiSet è un insieme di dati che può contenere duplicati il cui ordine non è rilevante.

Gli elementi possono essere solamente aggiunti e rimossi, non modificabili.

La richiesta è di implementarlo in modo tale da minimizzare l'uso della memoria - senza dover memorizzare i duplicati di un elemento; per conseguire ciò il MultiSet può essere visto come un insieme di nodi <valore, occorrenze>.

Tipi di dati:

L'entità nodo è stata rappresentata tramite una struttura dati eterogenea (struct) per poter rappresentare:

- dato: di tipo coppia<T> contenente <valore, occorrenze>
- nodo successivo: puntatore al nodo successivo

Ogni nodo è rappresentato mediante una classe coppia che definisce:

- elemento: di tipo generico T è l'effettivo elemento da memorizzare nel MultiSet
- occorrenze: di tipo size_type contenente il numero di occorrenze dell'elemento

Implementazione multiset:

Dato che nella traccia si parla di elemento generico T, è stato necessario rendere la classe templata. A sua volta questo ha richiesto l'introduzione di un altro parametro template per il confronto degli elementi per l'uguaglianza. Pertanto l'inizializzazione di MultiSet richiede non solo il tipo T degli elementi contenuti nei nodi, ma anche un altro per poter inizializzare il funtore necessario al confronto.

Per quanto riguarda i metodi fondamentali, sono presenti tre costruttori ed il distruttore:

- costruttore di default: inizializza un MultiSet vuoto
- costruttore copia: inizializza un MultiSet partendo da un altro passato per argomento
- costruttore a partire da sequenza di dati Q - approfondito successivamente
- distruttore

Per gli altri metodi fondamentali si veda il paragrafo successivo.

Implementazione metodi:

`multiset(Q inizio, Q fine)`

Costruttore che prende in input due iteratori di tipo `const_elem_iterator`, `inizio` che punta al primo elemento di un MultiSet e `fine` che punta all'ultimo elemento dello stesso MultiSet. Viene inizializzato un nuovo MultiSet inserendo tutti gli elementi

`nodo *find_helper(const T &elem)`

Metodo helper utile per ritornare il puntatore all'elemento passato come argomento. Si parte dalla `_testa` del MultiSet e viene confrontato l'elemento `elem` con l'elemento di ogni coppia che compone il MultiSet, quando questi sono uguali viene ritornato il puntatore dell'elemento, `nullptr` altrimenti

`nodo *find_previous(const nodo *mionodo)`

Metodo helper utile per ritornare il puntatore del nodo precedente rispetto a quello passato come argomento. Viene memorizzata la `_testa` del MultiSet ed il suo successivo, dopodiché, fintantoché `mionodo` è diverso dal successivo del nodo corrente che viene controllato, si scorre il MultiSet memorizzando sempre nodo attuale ed il suo successivo. Quando `mionodo` coincide con il successivo del corrente, viene ritornato il corrente

`void clear()`

Metodo per svuotare il MultiSet. Viene creato un puntatore d'appoggio `nodo tmp` inizializzato a `nullptr`; fintantoché la `_testa` del MultiSet che si sta svuotando è diversa da `nullptr` si assegna a `tmp` il nodo successivo al nodo in `_testa`, viene rimossa il nodo in `_testa` e viene assegnato a `_testa` il nodo `tmp`

`void add(const T &elem, size_type occ)`

Metodo che inserisce nel MultiSet un nuovo elemento di tipo T con `occ` occorrenze. Si controlla che l'elemento non sia già presente nel MultiSet invocando `contains()`, se non presente si controlla che la `_testa` del MultiSet in cui si vuole inserire l'elemento punti a `nullptr` (MultiSet vuoto), in questo caso viene creato un nuovo `nodo _testa` inserendo l'elemento e il numero di occorrenze. In caso in cui la `_testa` non punti a `nullptr`, si memorizza l'attuale `_testa` in un puntatore `tmp`, si crea una nuova `_testa` inserendo l'elemento e il numero di occorrenze; successivamente viene assegnato come nodo successivo alla `_testa` appena creata la precedente `_testa`. Infine si incrementa la `_dimensione` e il `_numelementi` del MultiSet. Nel caso in cui l'elemento sia già presente, si invoca `find_helper()` sull'elemento, si incrementano le sue occorrenze invocando `set_occorrenze()` ed il `_numelementi` del MultiSet

`add(const T &elem)`

Metodo che inserisce nel MultiSet un nuovo elemento di tipo T senza occorrenze passate in input. Viene invocato il metodo `add()` precedentemente descritto passando come parametri `elem` ed `1` occorrenza

`remove(const T &elem)`

Metodo che rimuove dal MultiSet un'occorrenza dell'elemento passato come parametro. Si controlla che l'elemento sia presente nel MultiSet invocando `contains()`, in caso negativo viene generata un'eccezione `el_non_trovato`. Nel caso in cui l'elemento esista viene salvato il suo puntatore in una variabile `nodo n`, dopodiché, se le occorrenze di questo elemento sono maggiori di `1`, le viene decrementato di `1` invocando `set_occorrenze()` e passandogli come parametro le sue `occorrenze() - 1`. Se le occorrenze dell'elemento fossero uguali ad `1` si controlla innanzitutto che l'elemento che voglio eliminare non coincida con la `_testa`, in questo caso, prima di eliminarlo, si assegna come nuova `_testa` il nodo successivo a quello che si vuole eliminare; se non coincide con la `_testa` si memorizza il suo nodo precedente, si assegna come nodo successivo al precedente appena salvato il successivo del nodo che si vuole eliminare, dopodiché viene eliminato il nodo. Nel caso in cui venga rimosso un nodo viene decrementata la `_dimensione` del MultiSet, in ogni caso si decrementa il `_numelementi` del MultiSet

`remove(const T &elem, size_type occ)`

Metodo che rimuove dal MultiSet un numero di occorrenze definito in input passandolo come parametro. Viene invocata la funzione `remove()` precedentemente descritta un numero di volte pari al numero di occorrenze da rimuovere

`contains()`

Metodo utile a verificare l'effettiva esistenza di un elemento all'interno del MultiSet. Salvo il puntatore dell'elemento di cui voglio verificarne l'esistenza: se il puntatore punta a `nullptr` l'elemento non esiste, in caso contrario esiste

`get_numelementi()`

Metodo getter utile a conoscere il numero di elementi del MultiSet

`get_dimensione()`

Metodo getter utile a conoscere il numero di nodi del MultiSet

`get_occorrenze(const T &elem)`

Metodo getter utile a conoscere il numero di occorrenze di un elemento passato come parametro. Viene memorizzato il puntatore del nodo che contiene quell'elemento, se il puntatore punta a `nullptr`, viene ritornato `0`, altrimenti vengono ritornate le occorrenze di quell'elemento invocando `occorrenze()`

Iteratori:

Sono stati implementati due iteratori di tipo forward di sola lettura:

- `const_iterator` --> itera ogni nodo all'interno del MultiSet - utilizzato da `operator<<` per la stampa dei singoli nodi
- `const_elem_iterator` --> itera ogni elemento all'interno del MultiSet - utilizzato per output dei singoli elementi (ogni elemento viene restituito un numero pari alle sue occorrenze) e per il costruttore tramite sequenza di dati Q

Implementazione/Ridefinizione funzioni globali:

`operator<<`

Stampa il MultiSet utilizzando l'iteratore `const_iterator` nella forma `<elemento, occorrenze>` nel caso in cui ci siano elementi da stampare, in caso contrario stampa una stringa personalizzata "`<0,0> - MultiSet vuoto`"

Main:

- Vengono definiti tre funtori di comparazione:

- `intcmp` --> compara interi
- `charcmp` --> compara caratteri
- `strcmp` --> compara stringhe

- Vengono definiti tre metodi per testare le funzionalità del programma, ogni funzione si occupa di una tipologia di dato:

- `test1_int()` --> effettua test con interi
- `test2_str()` --> effettua test con stringhe
- `test3_char()` --> effettua test con caratteri

- Ogni metodo esegue gli stessi test:
 - 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_elem_iterator
 - operatore di stream di output (operator<<)