Davide Raccuglia, mat. 807160
d.raccuglia@campus.unimib.it

Sparse Matrix<T>

Corso di Programmazione C++
Relazione sulla progettazione e l'implementazione.

Il progetto d'esame in questione richiede la progettazione e la realizzazione di una classe *template* che implementa una matrice sparsa, contenente dati generici di tipo T. Per definizione matematica, una matrice sparsa è quel tipo di matrice i cui valori sono quasi tutti uguali a zero. Dunque dal punto di vista implementativo è conveniente pensare di memorizzare solo i valori esplicitamente inseriti dall'utente e di scegliere un valore di default per rappresentare gli zeri, o in generale gli elementi *nulli.* Se consideriamo una matrice di dimensione $n \times m$, secondo il criterio descritto, avremo una migliore gestione delle risorse rispetto a memorizzare tutti gli $n \times m$ valori della matrice. Dal punto di vista logico, invece, la matrice si deve comportare come tale e, dunque, alla richiesta di un elemento in posizione (i, j), dove i e j sono rispettivamente l'indice di riga e di colonna, deve essere ritornato l'elemento inserito esplicitamente in quella posizione, oppure il valore di default, che viene definito dall'utente in fase di costruzione della matrice stessa.

Avendo letto attentamente le specifiche di progetto richieste, ho scelto di utilizzare come struttura dati di base l'array, in particolare un array di puntatori, ognuno dei quali referenzia un determinato elemento della matrice.

Il primo passo è stato dunque definire il singolo elemento della matrice. Esso possiede tre caratteristiche: un valore di tipo T, un indice di riga e un indice di colonna. Questi ultimi costituiscono la sua posizione all'interno della matrice. In altre parole identifico un elemento mediante la tripla (v, i, j). Per rappresentare ciò ho definito una struttura dati per l'elemento, `struct element`, che possiede questi tre attributi e un costruttore che li inizializza. Inoltre,

come da specifiche, gli indici di riga e colonna sono stati dichiarati costanti e per tanto non è stato definito il costruttore di default per l'elemento.

Per quanto riguarda la matrice, nello specifico, essa possiede un doppio puntatore agli elementi che contiene, ossia un puntatore ad un array di puntatori, i quali referenziano ognuno un elemento. Tra le varie implementazioni possibili, tra cui ad esempio una linked list, questa mi è sembrata la più intuitiva e semplice da gestire per quanto riguarda l'allocazione delle risorse. Altre informazioni importanti definite all'interno della matrice sono: il valore di default di tipo T e una variabile che tiene traccia del numero di elementi esplicitamente inseriti dall'utente.

Come di consueto nella definizione di una classe, sono stati dichiarati e definiti i costruttori e gli operatori opportuni , in particolare:

- un costruttore primario che inizializza il valore di default della matrice, con il valore passato dall'utente in fase di costruzione della matrice, e inoltre inizializza gli altri parametri a 0. Tale costruttore rappresenta il costruttore di default per la matrice, di fatto non è stato dichiarato il costruttore vuoto, poiché, come da specifiche di progetto, l'utente deve passarlo alla creazione della matrice. Mi è sembrato sensato, quindi, non permettere la creazione di una matrice non definendo tale valore;
- un costruttore secondario template, crea una matrice di tipo T prendendo in input una matrice di tipo Q, sfruttando un iteratore costante per la lettura dei valori e il passaggio di questi alla funzione `add`. In questo caso, come da specifiche, è stato lasciato al compilatore il lavoro di controllare la compatibilità fra i tipi;
- un distruttore che si occupa di liberare le risorse utilizzate dalla matrice durante l'esecuzione del programma e fa ciò sfruttando il metodo `clear`.

A questo punto passiamo alla definizione dei metodi necessari per il funzionamento e l'utilizzo di tale classe. Tra quelli essenziali, sono stati definiti: un metodo che permette di recuperare il valore di default della matrice e un metodo che permette di settarlo, dunque sostituire quello corrente con uno nuovo; dei metodi che permettono di conoscere rispettivamente il numero di elementi inseriti esplicitamente (ovvero la `size` della matrice), il numero massimo di colonne e il numero massimo di righe.

Gli elementi vengono aggiunti all'interno della matrice tramite il metodo `add`, il quale effettua un'aggiunta sequenziale, ossia un valore alla volta, prendendo in input la tripla che identifica un elemento, ossia (v, i, j). L'implementazione di tale metodo permette di avere

una matrice dinamica, in particolare la sua dimensione viene incrementata di uno ogni volta che si aggiunge un elemento. Ho preferito lasciare la scelta all'utente riguardo le dimensioni della matrice, senza permetterne la creazione con dimensioni prefissate. Inoltre il corpo di tale metodo è racchiuso all'interno di un blocco `try/catch` per la gestione delle eccezioni, questo perché vengono effettuate delle istruzioni `new` e qualora si verificasse un fallimento, tale gestione ci permette di liberare le risorse utilizzate e lasciare la classe in uno stato coerente. Infine all'interno del metodo `add` viene chiamato il metodo di supporto `sort`, che è semplicemente una versione adattata dell'algoritmo insertion sort, il quale si occupa dell'ordinamento degli elementi, effettuato considerando il classico ordine logico degli elementi di una matrice.

Fatto ciò, ho definito l'operatore di lettura coordinate, ossia `operator()`, il quale ci permette di accedere tramite indice di riga e di colonna, al valore di tipo T che si trova in quella posizione nella matrice. In caso quella cella dovesse essere vuota, verrà ritornato il valore di default. Ho successivamente sfruttato tale operatore per definire un metodo `show`, che permette la stampa completa della matrice, dunque anche degli elementi di default, e cercando di rispettare quella che è la classica rappresentazione di una matrice.

Inoltre ho ridefinito l'operatore `operator<<`, sfruttando gli iteratori costanti della matrice, utile per la stampa a video dei soli elementi inseriti esplicitamente.

Infine l'ultimo metodo definito è `evaluate`, che come da specifiche, verifica un predicato sugli elementi di una matrice, ritornando il numero di valori che lo soddisfano, compresi anche tutti i valori rappresentati dal valore di default.

Sono state infine dichiarate due *inner class* dentro la classe della matrice, ossia una classe per l'iteratore forward il quale potrà sia leggere che scrivere, ma questo vale solo per il valore di tipo T dell'elemento, poiché riga e colonna essendo costanti non possono essere modificati; l'altra classe invece implementa l'iteratore costante della matrice, che appunto è utilizzabile solo in lettura poiché costante.

Test.

Nel file main, vengono creati quelli che sono i *funtori* di uguaglianza dei vari tipi utilizzati per testare la classe e i *funtori* per poter testare il metodo `evaluate`.

Nella prima parte vengono effettuati dei test con due tipi primitivi e in seguito `std::string`, per testare i metodi fondamentali ed effettuare operazioni che caratterizzano l'uso della classe: aggiunta di elementi in modo disordinato oppure recupero di alcuni

elementi, stampa dei valori, creazione di altre matrici usando i vari costruttori disponibili. Successivamente sono stati testati `std::vector` e due tipi custom, effettuando le medesime operazioni e ottenendo esito positivo ai vari test.