Relazione Progetto Sparse Matrix

Fiorenza Gioele 851631

4/01/2022

1 Introduzione

Il progetto prevede l'implementazione di una Sparse Matrix (o matrice sparsa), dove solo gli elementi effettivamente inseriti, tramite l'apposita chiamata **set**, sono effettivamente memorizzati all'interno della matrice.
Per l'implementazione di questo dettaglio ho fatto uso di una lista concatenata di elementi che va a gestire la matrice logica.
Questa è stata realizzata attraverso 2 struct, la prima, chiamata "**element**" contiene il dato effettivamente memorizzato e la posizione dell'elemento nella matrice, mentre la seconda, chiamata "**nodo**" mantiene la struttura della lista concatenata, avendo un attributo che memorizza l'element e uno che mantiene il puntatore al nodo successivo.
Ovviamente il tipo di element rispetta i vincoli imposti dalla consegna e mantiene come tipo di dato memorizzato il tipo templato T, passato alla classe sparse matrix.

2 Descrizione della struct nodo

Questa struct è fondamentale, in quando permette di nascondere i dettagli implementati della matrice all'utilizzatore, infatti quest'ultimo penserà di star interagendo con una matrice, quando in realtà viene gestita la lista concatenata spiegata precedentemente e le interazioni avvengono sulla singola cella della matrice e attraverso la struct element.

2.1 Prestazioni

Attraverso questa struct è possibile andare a memorizzare solo ke celle che vengono effettivamente inserite, infatti quando viene chiamata la funzione set e non sono presenti elementi nella posizione specificata verrà allocato un nuovo nodo sullo heap (con l'uso della new) e conseguentemente questo sarà accodato nella lista concatenata.

Questa implementazione permette di avere il minimo spreco di memoria per la memorizzazione dei dati, in quanto solo quelli inseriti saranno realmente scritti (a meno di avere una matrice completamente piena), però ha una limitazione nella velocità di lettura dei dati in quando sarà necessario scorrere tutta la lista di elementi per trovare un elemento.

Una possibile soluzione a questo difetto potrebbe essere una struttura ad albero, implementabile tramite due puntatori e i dati sono organizzati secondo una logica che mantiene l'albero compatto, ma organizzato sulla posizione degli elementi.

2.2 Copy constructor del nodo

Anche se è un dettaglio minore in quanto la struct del nodo è privata e utilizzata dalla classe matrice, nel suo copy constructor ho deciso di non copiare il puntatore al next, in quanto non mi sembrava sensato che una copia dovesse condividere l'intero seguito della matrice di cui si sta facendo la copia, quindi nella mia implementazione ho deciso di impostare il next a nullptr, in questo modo sarà poi la matrice che deciderà come gestire la logica del next.

2.3 Eccezioni

Per quanto riguarda le eccezioni ho deciso di creare una classe custom di eccezioni per gestire l'errore di out of range generato quando vengono sforati gli indici della matrice logica attraverso le operazioni di set o lettura e nella sua implementazione ho deciso di estendere la classe **std::runtime_error** rispetto alla classe std::overflow_error perché di fatto non è un limite hardware l'accesso fuori dalla matrice logica, ma solamente una decisione dell'implementazione.