

Progetto di programmazione c++

Codice scritto da: Alessandro Biagiotti

Numero di matricola: 869014

e-mail: a.biagiotti2@campus.unimib.it

1. Richiesta del progetto

La richiesta del progetto per la parte di programmazione in c++ era quella di andare a costruire un programma che emulasse il comportamento di una matrice sparsa ovvero una matrice molto più grande rispetto alla quantità di dati in essa contenuti.

Una matrice del genere può essere facilmente trovata nei problemi di teoria dei grafi quando consideriamo il caso dei grafi sparsi.

2. Struttura dati utilizzata

La principale richiesta del progetto era dunque quella di realizzare la struttura dati che potesse stare dietro a una Sparse Matrix.

Sicuramente il candidato più adatto era la lista dinamica, le alternative sicuramente c'erano ma l'unica altra struttura dati che mi è venuta in mente è l'array dinamico ridimensionabile a cui si aggiunge la meccanica del fattore di carico (preso in prestito dalle hashtables) per renderlo più efficiente in termini di memoria.

I motivi per cui ho deciso di ignorare tale alternativa sono i seguenti:

- 1) Non si sposava con la richiesta di uso di memoria minimale
- 2) La gestione della memoria e della struttura in sè diventava rognoso perché era importante controllare e riaggiustare la dimensione della struttura in modo da avere sempre abbastanza spazio e guadagnare un po' in termini di complessità temporale (anche se so che l'efficienza non era uno dei goal del progetto)
- 3) Utilizzare memoria minimale su una struttura del genere significava andare ad allargare l'array con ogni inserimento, quindi prendere il contenuto, copiarlo e incollarlo da un'altra parte. Il risultato? Perdo tempo per avere l'accesso randomico e costante ai dati, che non mi e' così utile

Dunque la scelta è chiaramente andata a favore delle liste dinamiche non ordinate, questo perchè, dopo un paio di tentativi, non mi è venuto in mente un ordinamento dei dati che fosse sensato. Due delle alternative che stavo inizialmente considerando erano:

- Ordinamento in base alla distanza dall'origine
- Ordinamento in base alla somma delle coordinate

Alla fine però ho capito di starmi complicando la vita senza riuscire a ottenere nulla di così utile da poterlo includere all'interno del codice, visto che l'ordinamento non era richiesto e non rendeva nessuna operazione più semplice.

Per richiesta del progetto era necessario fare sì che l'iteratore lavorasse su oggetti di tipo element, contenenti: un valore templato, il numero di colonna e il numero di riga. Era dunque evidente che andava inserita una struttura element all'interno del codice.

Prima di cominciare a programmare però mi sono fermato e mi sono chiesto quale fosse il miglior modo di costruire la lista, da principio pensavo di inserire direttamente il puntatore all'elemento successivo dentro la struct element (probabilmente c'era un modo di far funzionare tutto), ma la soluzione che avevo trovato non mi ha mai convinto e dunque mi sono mosso verso quello che è stato, l'approccio definitivo: wrappare un element in una struct di tipo nodo, la struct nodo poi si sarebbe occupata di puntare al nodo successivo. Ad oggi, ritengo ancora che sia la soluzione più sensata ed elegante, visto che c'è separazione degli interessi: element resta element e nodo resta nodo.

3. Tipi di dato

Per quanto riguarda i tipi di dato che ho definito, ho deciso di ridefinire solamente dim_type, visto che la struct nodo è privata e la struct element, anch'essa privata, compare solo nella sezione di definizione dell'iteratore, che ne nasconde il nome tramite i suoi typedef.

L'unica cosa interessante da dire riguardo ai tipi di dato è che, per dim_type, ho avuto non poca indecisione.

Inizialmente, guardando i codici scritti insieme a lezione, avevo definito la dimensione come unsigned int. Il che ha perfettamente senso, ed è in linea con la logica implementativa del progetto.

Il problema che non ho considerato fino a quando non mi è effettivamente venuto in mente è che se io passo un valore interno negativo come unsigned int il problema non mi dava errore in compilazione ma il risultato era certamente catastrofico, perché il valore in ingresso nel metodo diventava randomico (e positivo).

Per questo ho deciso di andare a definire int come dim_type, questo mi consente di fare dei controlli sul valore della dimensione preso in input dal costruttore e questo mi evita gli errori silenti di cui ho parlato sopra.

4. Altri dettagli riguardo alle funzioni del progetto

Sparse Matrix

- Costruttore di default:

Inizializzo l'elemento di default con il suo costruttore di default così da non avere problemi se ho una matrice vuota e viene richiamata la getDefault() o altre funzioni su di essa.

- Ridefinizione dell'operatore di stampa:

Ridefinisco l'operatore direttamente all'interno della classe perchè così evito di riscriverlo in fondo con la dichiarazione del tipo templato.

Per le informazioni principali penso che la documentazione che ho scritto riguardo al codice sia abbastanza buona ed estensiva, io ho scritto delle note anche riguardo ai metodi del main e ai metodi privati (tranne quelli degli iteratori), nel caso in cui non apparissero generati da Doxygen possono essere letti direttamente dal codice (anche se è esattamente l'esperienza ideale).

4. Test

I miei test sono principalmente stampe perchè ho preferito scriverli fin da subito con il debugging in mente (mi torna molto più semplice lavorare con il debugging di stampe che non con le assert et similia).

Ho deciso di sondare la qualità della mia implementazione tramite tre livelli di difficoltà:

- Il test più semplice impiega la classe Sparse Matrix inserendo interi all'interno della matrice
- Il test medio che impiega la classe Sparse Matrix inserendo semplici punti all'interno della matrice
- Un test non trivial che utilizza la classe ClasseNonTrivial definita all'inizio del main e che si propone di modellizzare (seppur in termini estremamente semplicistici e talvolta un po' assurdi) gli studenti che seguono un corso.

Inoltre ho incluso alcuni funtori di test per ciascuna delle difficoltà

La strategia di test è sempre più o meno la stessa:

- 1) Costruisco una SparseMatrix templata vuota che poi riempio con alcuni oggetti, in modo da testare sia l'inserimento che la sovrascrittura
- 2) Testo il copyconstructor e l'inserimento e la sovrascrittura nella nuova matrice (controllando che non ci siano cose strane con la copia della lista e che tutti gli elementi rimangano dove devono rimanere)
- 3) Testo l'operatore di assegnamento a una matrice non vuota e faccio lo stesso test che ho fatto per il copyconstructor

Il tutto condito con un po' di stampe su cout per la SparseMatrix e un po' di stampe per le liste che contengono i valori.

Inoltre per ogni livello di difficoltà del test (anche se non penso proprio fosse necessario) ho aggiunto alcuni piccoli test per controllare che tutti gli assert fossero piazzati nel modo corretto (evitando che l'utente provi ad inserire elementi in luoghi vietati).

Ho aggiunto inoltre un primo test dall'utilità discutibile per mettermi l'animo in pace riguardo al comportamento della struttura vuota.

5. Requisiti di funzionamento

I requisiti affinchè il codice del progetto funzioni con le classi sono i seguenti:

- Devono essere definiti i metodi fondamentali (default constructor, copyconstructor, operatore di assegnamento e distructor).

Tutto il resto è accessorio, a livello di mero funzionamento non serve altro perchè i valori sono incapsulati all'interno dei nodi e degli elementi che contengono le informazioni necessarie per far funzionare tutto.

6. Momento soddisfacente

Per concludere questa relazione con una nota positiva, condivido il risultato di un'esecuzione di valgrind che è particolarmente soddisfacente.

```
==38501== HEAP SUMMARY:
==38501==     in use at exit: 0 bytes in 0 blocks
==38501==   total heap usage: 310 allocs, 310 frees, 102,337 bytes allocated
==38501==
==38501== All heap blocks were freed -- no leaks are possible
==38501==
==38501== For lists of detected and suppressed errors, rerun with: -s
==38501== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```