Strutture Dati, Algoritmi e Complessità

Esercitazione del 11 Marzo 2025

Scopo di questa esercitazione è realizzare una rappresentazione efficiente di matrici sparse. Si ricorda che una matrice viene definita sparsa quando i valori in essa contenuti sono per la grande maggioranza solitamente oltre il 90%) valori di default (nel nostro caso, 0).

Attenzione: L'esercitazione può essere svolta sia in linguaggio C che in Java. Entrambe le cartelle sono fornite dello scheletro delle classi da completare e di un programma `ran_mat.c` che, compilato ed eseguito con `$./ran_mat m n` genera un file `mat.dat` contenente una matrice sparsa con m righe ed n colonne, con valori randomici. Si noti che è possibile modificare la costante `PERCENTAGE` nel sorgente del programma, per variare la percentuale di valori non nulli presenti nella matrice (default 10%).

Non occorre creare nuovi file, ma soltanto completare le classi/moduli C forniti. Si consiglia agli studenti di implementare le funzioni C/metodi Java nell'ordine in cui sono proposti. In particolare, il metodo/funzione C `set` è quello che in pratica consente di creare una matrice.

Task 1. Rappresentazione con lista collegata lineare

Si vuole rappresentare una matrice sparsa come una lista lineare collegata i cui elementi sono i valori non nulli contenuti nella matrice. Ogni nodo della lista lineare dovrà essere composto da una terna di valori $<i, j, val>$, in cui i è il numero di riga, j è il numero di colonna e val rappresenta il valore contenuto nella cella (i, j) della matrice.

Suggerimento: Lo schema di base per rappresentare una matrice sparsa mediante una lista collegata lineare è riportato di seguito.

Per il linguaggio C:

```c
// Rappresenta un elemento della lista
typedef struct elem {
    int i;
    int j;
    int x;
    struct elem *next;
} elem;

// Rappresenta una matrice
struct matrice_sparsa {
    int m;
    int n;
    elem* head;
};
```

Per il linguaggio Java:

```
public class MatriceSparsa {

    private class Elem{
        int i;
        int j;
        int x;
        Elem next;
    }

    private int m;
    private int n;
    Elem head;

    ....

}
```

Specifiche. Scrivere un modulo C con intestazione mat sparsa lista.h (rispettivamente una classe Java MatriceSparsa.java) con l'interfaccia indicata di seguito:

Per il linguaggio C:

```
//Il tipo matrice sparsa
typedef struct matrice_sparsa matrice_sparsa;

//Costruttore di una matrice ad m righe ed n colonne
matrice_sparsa* matrice_sparsa_new(int m, int n);

//Restituisce il numero di colonne di mat
int get_num_col(matrice_sparsa* mat);

//Restituisce il numero di righe di mat
int get_num_row(matrice_sparsa* mat);

//Distruttore
void matrice_sparsa_delete(matrice_sparsa* mat);

//Applicata a mat, imposta il valore della cella <i,j> ad x
void mat_set(matrice_sparsa* mat, int i, int j, int x);

//Funzione che ritorna il valore in <i,j> di mat
int mat_get(matrice_sparsa* mat, int i, int j);

//Stampa a video la matrice mat
void mat_print(matrice_sparsa* mat);
```

Per il linguaggio Java:

```
// Costruttore
public MatriceSparsa(int m, int n)

// Restituisce il numero di righe
getNumRow()

// Restituisce il numero di colonne
getNumCol()

// Imposta il valore della cella (i, j) della matrice a x
set(int i, int j, int x)

// Restituisce il valore della cella (i, j) della matrice
get(int i, int j)

// Overriding del metodo Object.toString()
toString()
```

Task 2. Operazioni tra matrici

Si vogliono implementare delle operazioni comuni con le matrici come addizioni, calcolo della trasposta, moltiplicazioni. Di seguito viene riportata un'interfaccia che può servire da spunto, qualsiasi integrazione di queste operazioni basilari è da considerarsi un buon esercizio. Si possono testare le funzioni implementate utilizzando il main fornito con il materiale dell'esercitazione.

Nota: Le funzioni che implementano le operazioni tra matrici devono verificare correttamente che le premesse per poter applicare le varie operazioni siano soddisfatte (p.e., somma possibile solo se le due matrici hanno le stesse dimensioni).

Suggerimento: si consiglia di implementare dei metodi ausiliari che, data una matrice e delle coordinate (i, j), permettano di accedere velocemente al valore contenuto nella cella corrispondente alle coordinate $(i–1, j)$, $(i, j –1)$, $(i, j +1)$, $(i+1, j)$.

Interfaccia per il linguaggio C:

```
//Restituisce mat1+mat2 come una nuova matrice
matrice_sparsa* mat_add(matrice_sparsa* mat1, matrice_sparsa* mat2);

//Restituisce la matrice trasposta come una nuova matrice
matrice_sparsa* mat_tra(matrice_sparsa* mat);

//Restituisce mat1*mat2 come una nuova matrice
matrice_sparsa* mat_mul(matrice_sparsa* mat1, matrice_sparsa* mat2);
```

Interfaccia per il linguaggio Java:

```java
//Restituisce mat1+mat2 come una nuova matrice
public MatriceSparsa add(MatriceSparsa mat1, MatriceSparsa mat2);

//Restituisce la matrice trasposta come una nuova matrice
public MatriceSparsa tra(MatriceSparsa mat1, MatriceSparsa mat2);

//Restituisce mat1*mat2 come una nuova matrice
public MatriceSparsa mul(MatriceSparsa mat1, MatriceSparsa mat2);
```

Task 3. Test di prestazione

Eseguire test per verificare l'efficienza della struttura dati realizzata in questa esercitazione rispetto alla struttura tipicamente adottata (un array di array). Di seguito viene indicata una strategia applicabile:

1. Generare una matrice sparsa con valori casuali (ma solo con il 10% delle celle occupate). A questo scopo, il programma fornito `ran_mat` prende in ingresso una coppia di numeri `m` ed `n` e restituisce una matrice sparsa di dimensioni `m*n` con valori casuali.
2. Rappresentare tale matrice usando la struttura dati definita nei **Task** 1 e 2.
3. Eseguire il programma calcolandone il tempo di esecuzione. Si può usare il comando `time` della `bash` shell. Ad esempio `$time ./a.out` esegue il file a.out e ne riporta i tempi. In Java si può eseguire `time java Main`.
4. Ripetere i punti 2 e 3 rappresentando la stessa matrice generata al punto 1 come un array di array (metodo classico di rappresentazioni delle matrici su entrambi i linguaggi C e Java).
5. Riportare i risultati ottenuti.
6. Ripetere i punti 1-5 variando le dimensioni della matrice generata al punto 1.