Esercitazione 0
Matrici sparse

Corso di Fondamenti di Informatica II

BIAR2 (Ing. Informatica e Automatica) e BSIR2 (Ing. dei Sistemi Informatici)

A.A. 2012/2013

4 Ottobre 2013

Sommario

Scopo di questa esercitazione è realizzare delle classi Java per il supporto di matrici sparse.

1 Matrici sparse

Una matrice sparsa $m \times n$ è una matrice in cui il numero di elementi non nulli (k) è molto minore del numero complessivo mn di elementi della matrice. Un vettore sparso è una matrice sparsa con $m = 1$ o $n = 1$. L'uso di vettori e matrici sparsi è di grande importanza nel calcolo scientifico [2].

Si intendono realizzare due classi Java, `VettoreSparso` e `MatriceSparsa`. L'implementazione sarà basata sulla classe generica `HashMap<K, V>` del Java Collection Framework, che memorizza coppie $\langle \text{chiave}, \text{valore} \rangle$, in particolare sfruttandone i seguenti metodi:

```
public HashMap(); // crea una HashMap vuota
public V get(Object key);
        // restituisce il valore associato
        // alla chiave, o null se assente
public V put(K key, V value);
        // associa il valore value
        // alla chiave key
public boolean containsKey(Object key);
        // restituisce true se alla chiave
        // è associato un qualche valore
public Set<K> keySet();
        // restituisce l'insieme delle
        // chiavi inserite
```

Per realizzare le classi richieste, si consiglia di utilizzare `HashMap<Integer, Double>`.

2 La classe VettoreSparso

Si vuole realizzare una classe che implementi un vettore sparso ad elementi in virgola mobile (`double`). Si intende rappresentare solo gli elementi diversi da zero, attraverso una struttura di tipo `HashMap` che memorizzi per ciascun elemento non nullo la coppia $\langle \text{Indice di Riga}, \text{Valore} \rangle$ corrispondente. In tal modo l'occupazione di memoria del vettore è $O(k)$, dove k è il numero di elementi non nulli.

Interfaccia. L'interfaccia della classe `VettoreSparso` deve essere la seguente. Si consiglia di implementare i metodi nell'ordine in cui sono elencati.

```
public VettoreSparso(int m)
```

Crea un vettore di m zeri. La complessità temporale del metodo deve essere $O(1)$.

```
public int size()
```

Restituisce la dimensione del vettore.

```
public void put(int i, double x)
```

Setta ad x l'i-esima componente del vettore (con $1 \leq i \leq m$).

```
public double get(int i)
```

Restituisce l'i-esima componente del vettore.

```
public Iterable<Integer> indices()
```

Restituisce la collezione degli indici degli elementi non nulli del vettore.

```
public double dot(VettoreSparso that)
```

Calcola il prodotto scalare tra i vettori `this` e `that`. La complessità temporale del metodo deve essere $O(k)$, dove k è il numero di componenti non nulle del vettore `this`.

```
public VettoreSparso plus(VettoreSparso that)
```

Crea e restituisce un vettore sparso pari alla somma di `this` e `that`. La complessità temporale del metodo deve essere $O(k_1 + k_2)$, dove k_1 e k_2 sono il numero di componenti non nulle dei due vettori `this` e `that`.

```
public String toString()
```

Restituisce una stringa rappresentante il vettore per esteso (incluse le componenti nulle), quale ad esempio `0.0 4.0 0.0 -3.0` per un vettore di 4 elementi.

Si proceda a testare `VettoreSparso` attraverso il driver `Main.java`.

3 La classe MatriceSparsa

Si intende realizzare una classe Java `MatriceSparsa` che implementi una matrice sparsa ad elementi `double`. Si vuole usare la rappresentazione *a colonne* della matrice: ciascuna colonna è rappresentata attraverso un `VettoreSparso` relativo a quella colonna. In questo modo l'occupazione totale di memoria è $O(k+n)$, che in generale è molto minore di mn.

Interfaccia. I metodi della classe `MatriceSparsa` da implementare sono i seguenti. Si suggerisce di implementarli nell'ordine in cui sono elencati.

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

Crea una matrice sparsa di m righe ed n colonne in cui tutti gli elementi hanno valore 0. Il metodo deve avere complessità $O(n)$.

```
public int nRows()
public int nCols()
```

Restituisce il numero di righe e colonne della matrice, rispettivamente.

```
public double get(int i, int j)
```

Restituisce l'elemento memorizzato alla riga i e colonna j della matrice ($1 \leq i \leq m$, $1 \leq j \leq n$).

```
public void put(int i, int j, double x)
```

Setta ad x l'elemento della riga i e colonna j della matrice.

```
public VettoreSparso getCol(int j)
```

Crea e restituisce un vettore corrispondente alla colonna j della matrice ($1 \leq j \leq n$). La complessità temporale del metodo deve essere $O(k_j)$, dove k_j è il numero di componenti non nulle della colonna j.

```
public MatriceSparsa plus(MatriceSparsa that)
```

Crea e restituisce una matrice sparsa pari alla somma di `this` e `that`. La complessità temporale del metodo deve essere $O(k_1 + k_2 + n)$, dove k_1 e k_2 sono il numero di componenti non nulle delle due matrici `this` e `that`.

```
public String toString()
```

Restituisce una stringa rappresentante la matrice per esteso (incluse le componenti nulle), quale ad esempio

$$\begin{matrix} 1.0 & 0.0 & 3.0 \\ 0.0 & 5.0 & 0.0 \\ 7.0 & 0.0 & 9.0 \end{matrix}$$

per una matrice di 3 righe e 3 colonne. Si utilizzi il carattere `\n` come terminatore di riga.

Si proceda a testare `MatriceSparsa` attraverso il driver `Main.java`.

4 Funzionalità ulteriori

Si completi la classe `MatriceSparsa` con i seguenti metodi.

```
public VettoreSparso getRow(int i)
```

Crea e restituisce un vettore corrispondente alla riga i della matrice ($1 \leq i \leq m$).

```
public MatriceSparsa transpose()
```

Crea e restituisce la matrice trasposta della matrice `this`.

```
public static VettoreSparso
        multiply(MatriceSparsa A, VettoreSparso v)
```

Crea e restituisce un vettore pari al prodotto di A e v.

Riferimenti bibliografici

[1] M. T. Goodrich and R. Tamassia. *Strutture dati e algoritmi in Java.* Zanichelli, 2007.

[2] J. R. Gilbert, C. Moler and R. Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM J. Matrix Anal. Appl.*, 13(1):333–356, 1992.