Algoritmi e Strutture Dati 16/17 *Durata: 2 ore*

Primo esonero di laboratorio del 15 Novembre 2016

1 Traccia

In matematica una matrice sparsa è una matrice i cui valori sono quasi tutti uguali a zero. Rappresentare una matrice sparsa con un array bidimensionale corrisponderebbe ad un grosso spreco di memoria. Il principio di base che si segue invece per memorizzare una matrice sparsa è di salvare soltanto i valori diversi da zero. A seconda del numero e della distribuzione dei valori diversi da zero, si possono utilizzare diverse strutture dati ottenendo risparmi considerevoli in termini di memoria.

Alcune delle strutture che permettono modifiche efficienti alla matrice sono *DOK* (Dictionary of keys), *LIL* (List of lists), o *COO* (Coordinate list). LIL memorizza una lista per ogni riga della matrice. Ogni elemento della lista contiene l'indice di colonna e il valore. Tipicamente gli elementi sono ordinati per indice di colonna. COO mememorizza una lista di tuple $< row, column, value >$. Idealmente, gli elementi sono memorizzati per indice di riga prima e per indice di colonna dopo.

Realizzare e completare in C++ **una** delle due classi `sparseLIL` e `sparseCOO`, definite in seguito, per la memorizzazione di matrici sprarse. Prevedere una funzione main che contenga una procedura di inserimento automatico e successivamente il test dei metodi implementati.

1.1 sparseLIL (max 30 punti)

Nota: gli elementi in ogni lista sono ordinati per colonna.

```
template<class value_type>
class LIL_pair{
public:
   int column;
   value_type value;
};

template<class value_type>
class sparseLIL {
   // costruttore: crea una matrice sparsa nr x nc di elementi nulli
   sparseLIL(int nr, int nc);
   // aggiunge un valore non nullo alla matrice
   void insert(int row, int column, value_type value);
   // rimuove un valore non nullo dalla matrice
   void remove(int row, int column);
   // restituisce un valore dalla matrice
   value_type get(int row, int column);
   // rende la matrice sparsa una matrice identit\‘a di dimensione mxm
   void eye(int m);
   // somma alla matrice implicita la matrice A senza usare il metodo get
   void sum(sparseLIL<value_type> A);
private:
   List<List<LIL_pair<value_type> > > M;
   // o alternativamente un array di liste List<LIL_pair<value_type> >
};
```

1.2 sparseCOO (max 27 punti)

Nota: gli elementi nella lista sono ordinati prima per riga e poi per colonna.

```
template<class value_type>
class COO_tuple{
public:
   int row;
   int column;
   value_type value;
};

template<class value_type>
class sparseCOO {
   // costruttore: crea una matrice sparsa nr x nc di elementi nulli
   sparseCOO(int nr, int nc);
   // aggiunge un valore non nullo alla matrice
   void insert(int row, int column, value_type value);
   // rimuove un valore non nullo dalla matrice
   void remove(int row, int column);
   // restituisce un valore dalla matrice
   value_type get(int row, int column);
   // rende la matrice sparsa una matrice identit\'a di dimensione mxm
   void eye(int m);
   // somma alla matrice implicita la matrice A senza usare il metodo get
   void sum(sparseLIL<value_type> A);
private:
   List<COO_tuple<value_type> > M;
};
```

Prevedere una funzione main che contenga una procedura di inserimento automatico e successivamente il test dei metodi implementati.