Laboratorio di Algoritmi e Strutture Dati I

Esercitazione 3

Lezione 3: Matrici

- **Problema:** Vogliamo monitorare le vincite di **10** persone che giocano al lotto per 15 giorni (10x15 = 150 elementi).
- Nell'arco di questo periodo vengono registrate solo 7 vincite.
- Utilizziamo una matrice.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
71	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	**99**
0	0	0	0	0	0	0	0	0	0	0	**53**	0	0	0
0	0	0	0	0	0	0	**95**	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	**39**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	**27**	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	**14**

Lezione 3: Matrici Sparse

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
71	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	**99**
0	0	0	0	0	0	0	0	0	0	0	**53**	0	0	0
0	0	0	0	0	0	0	**95**	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	**39**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	**27**	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	**14**

- Ricaviamo una matrice 10x15 ma…
- Quanti sono i valori “utili” (diversi da zero) ?
- Si può pensare di non memorizzare gli zeri e risparmiare spazio: esempio di complessità computazionale spaziale.

Lezione 3: Matrici Sparse

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
71	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	99
0	0	0	0	0	0	0	0	0	0	0	53	0	0	0
0	0	0	0	0	0	0	95	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	39
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	27	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	14

Num. righe

Num. colonne

Num. elem non nulli

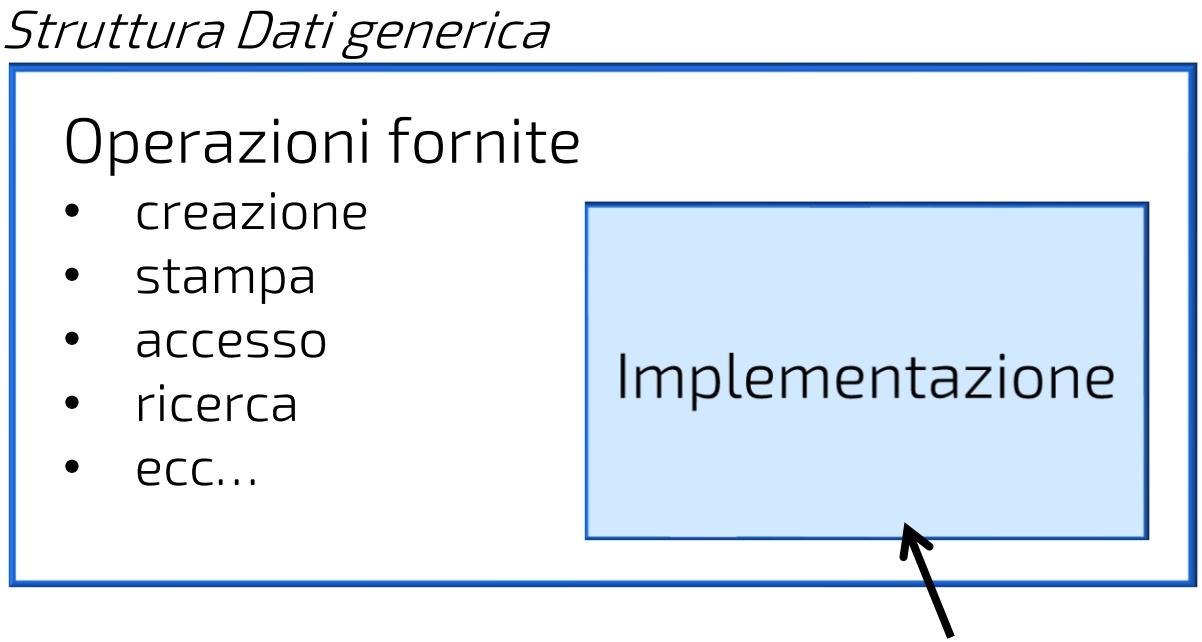
10	15	7
1	0	71
2	14	99
3	11	53
4	7	95
6	14	39
8	1	27
9	14	14

<riga, colonna, valore>

- Le triple (dalla pos 1) sono ordinate per num. riga
- A parità di num. riga sono ordinate per num. colonna
- **150** elementi VS **24** elementi → stesse informazioni

Lezione 3: Strutture Dati Astratte

- L'implementazione della Struttura dati deve essere trasparente per l'utente.



Esercizio 3: Progetto base

- Andate su eLearning e scaricate il **progetto base** su cui lavorare.
- Prendete il codice presente sul progetto base e inseritelo in un vostro (nuovo) progetto su CLion.
- Troverete già implementate le funzioni di creazione, stampa e ricerca per una matrice classica (NON modificate queste funzioni).
- Nello stesso progetto inserite le funzioni necessarie per l'esercitazione odierna.
- Utilizzate le variabili globali già dichiarate **numUsedIntegerClassic** e **numUsedIntegerSparse** per contare:
  - Il numero di locazioni intere utilizzate per rappresentare una matrice nella sua forma classica.
  - Il numero di locazioni intere utilizzate per rappresentare la stessa matrice nella sua rappresentazione "sparsa".

Esercizio 3_1: Creazione Matrice Sparsa

Modificare il progetto base:

1. definire una struttura che rappresenti una tripla **<riga, colonna, valore>**
2. aggiungere una funzione che consenta di leggere in input i dati (le triple) e le inserisca in una nuova matrice sparsa (i dati saranno memorizzati ordinati per riga e, a parità di numero di riga, per colonna).
3. utilizzare l'allocazione dinamica per la creazione della matrice sparsa.

Esempio. Inserimento dei dati per creare la matrice dell'esempio precedente:

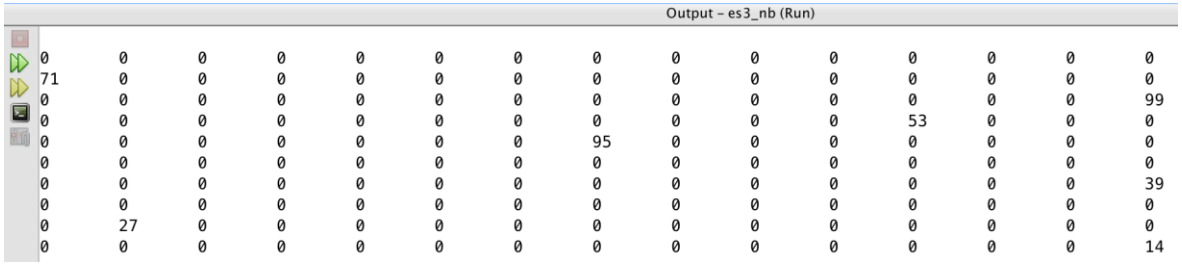
<10 15 7 >
<1 0 71 >
<2 14 99 >
<3 11 53 >
<4 7 95 >
<6 14 39 >
<8 1 27 >
<9 14 14 >

Esercizio 3_2: Stampa Matrice Sparsa

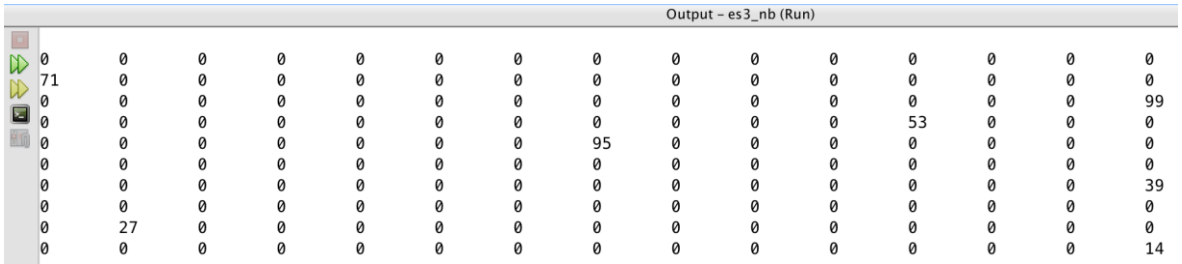
Modificare il progetto base, aggiungendo una funzione che effettui la **visualizzazione** degli elementi della matrice sparsa.

- NB: Il risultato della visualizzazione della matrice sparsa deve essere identico al risultato della visualizzazione di una matrice classica.

Visualizzazione della matrice classica.



Visualizzazione della matrice sparsa.



Hint: ricordate come sono memorizzate (in quale ordine) le triple.

Hint2: NON servono 3 cicli annidati.

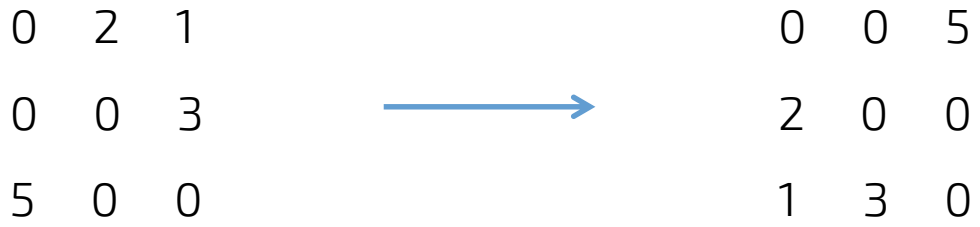
Esercizio 3_3: Ricerca di un valore

Modificare il progetto base, aggiungendo una funzione che effettui la **ricerca** di un valore all'interno della matrice sparsa.

- NB: La funzione deve esclusivamente stampare la posizione dell'elemento o un'informazione che comunichi l'inesistenza dell'elemento (la posizione non deve essere restituita…).

Esercizio 3_4: Matrici Trasposte

- ESEMPIO:



- Generiamo la trasposta di una matrice sparsa utilizzando l'algoritmo di **trasposizione rapida**. Tale algoritmo utilizza due vettori di appoggio, termini_riga e pos_iniziale, così definiti:
  - **termini_riga** indica, per ogni riga (della trasposta), il numero di elementi diversi da zero.
  - **pos_iniziale** indica, per ogni riga (della trasposta), la posizione del primo elemento diverso da zero.
  - Il punto di partenza per la prima riga (riga 0 -> pos_iniziale[0]) della matrice trasposta è 1 (perché?)
  - Il punto di partenza per la riga i-esima della matrice trasposta (i ≥ 1) è dato da **pos_iniziale[i-1] + termini_riga[i-1]**.

Esercizio 3_4: Matrici Trasposte

Esempio: costruzione di **termini_riga** e **pos_iniziale** per la nostra matrice.

10	15	7
1	0	71
2	14	99
3	11	53
4	7	95
6	14	39
8	1	27
9	14	14

termini_riga = numero di valori != 0 in ogni riga della trasposta
pos_iniziale[0] = 1
pos_iniziale[i] = pos_iniziale[i-1] + termini_riga[i-1]

termini_riga	1	1	0	0	0	0	0	1	0	0	0	1	0	0	3
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

pos_iniziale	1	2	3	3	3	3	3	3	4	4	4	4	5	5	5
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Esercizio 3_4: Trasposizione rapida

- Implementare l'algoritmo di **Trasposizione Rapida** per generare la trasposta di una matrice sparsa

```
algoritmo trasp_rapida(array di triple a, array di triple b)

num_col ← a[0].col
num_val ← a[0].valore
b[0].riga ← num_col
b[0].col ← a[0].riga
b[0].valore ← num_val

alloca dinamicamente il vettore termini_riga
alloca dinamicamente il vettore pos_iniziale

if (num_val > 0) then
    // creazione termini_riga
    for i ← 0 to num_col-1 do termini_riga[i] ← 0
    for i ← 1 to num_val do termini_riga[a[i].col] ← termini_riga[a[i].col] + 1

    //creazione pos_iniziale
    pos_iniziale[0] ← 1
    for i ← 1 to num_col-1 do pos_iniziale[i] ← pos_iniziale[i-1] + termini_riga[i-1]

    //trasposizione
    for i ← 1 to num_val do
            cur_pos ← pos_iniziale[a[i].col]
            pos_iniziale[a[i].col] ← pos_iniziale[a[i].col] + 1
            b[cur_pos].riga ← a[i].col
            b[cur_pos].col ← a[i].riga
            b[cur_pos].valore ← a[i].valore
```

Esercizio 3 : matrici a confronto

- Osservare i risultati ottenuti, in termini di memoria occupata
- Quale rappresentazione è più efficiente e quando?
- Esiste una rappresentazione che è SEMPRE più efficiente dell'altra?

Lezione 3 : Ricapitolando

Implementare:

- 3_1: creazione matrice sparsa
- 3_2: visualizzazione (stampa) matrice sparsa
- 3_3: ricerca di un valore nella matrice sparsa
- 3_4: trasposizione matrice sparsa
- NB: contare e confrontare le locazioni intere utilizzare per le due rappresentazioni

end().