Matrices Esparzas

Prof. Grover Enrique Castro Guzman

Ciencia de la Computación - UNSA

November 3, 2021

Motivación

Una matrix $m \times n$ es una **matriz esparsa** si muchos de sus elementos son ceros. Más precisamente, cuando el número de 0 es mayor que el número de valored diferente de ceros. Ejem: Matriz diagonal, tridiagonal, pues cada uno tiene $\mathcal{O}(n)$ elementos diferentes de cero y $\mathcal{O}(n^2)$ de elementos igual a cero.

Representaciones

- Arreglo de triplas (*triplet form*).
- Listas enlazadas
- *Compressed column*
- *Compressed row*

Arreglo de triplas

Esta forma de representar las matrices es la más simple de crear pero no es muy útil para la mayoría de aplicaciones.
Cada elemento no zero de la matriz es guardado en una tripla que contiene: *row*:

- **row:** la fila en la que se encuentra el elemento
- **col:** la columna en la que se encuentra el elemento
- **value:** el valor en la posición (row, col)

Arreglo de triplas: Ejemplo

4.5	0	3.2	0
3.1	2.9	0	0.9
0	1.7	3.0	0
3.5	0.4	0	1.0

Arreglo de triplas: Ejemplo

row	**2**	**1**	**3**	**0**	**1**	**3**	**3**	**1**	**0**	**2**
col	**2**	**0**	**3**	**2**	**1**	**0**	**1**	**3**	**0**	**1**
val	**3.0**	**3.1**	**1.0**	**3.2**	**2.9**	**3.5**	**0.4**	**0.9**	**4.5**	**1.7**

Arreglo de triplas: Ejemplo

row	**0**	**0**	**1**	**1**	**1**	**2**	**2**	**3**	**3**	**3**
col	**0**	**2**	**0**	**1**	**3**	**1**	**2**	**0**	**1**	**3**
val	**4.5**	**3.2**	**3.1**	**2.9**	**0.9**	**1.7**	**3.0**	**3.5**	**0.4**	**1.0**

Arreglo de triplas: Estructura

```
struct sparseMatrix{
    int m;  // número de filas
    int n; // número de columnas
    int nz; // número de entradas
    int *row; // indices de las filas
    int *col; // indices de las columnas
    double *val; // valores almacenados
};
```

Arreglo de triplas: Operaciones

- **get(i,j):** Obtener el elemento de la posición (i, j). **Complejidad:** $\mathcal{O}(nz)$
- **set(i,j,x):** Cambiar el valor del elemento en la posición (i, j) por x. **Complejidad:** $\mathcal{O}(nz)$
- **transpose:** Obtener la transpuesta de la matriz. **Complejidad:** $\mathcal{O}(nz)$
- **sum(B):** sumar con la matriz B. **Complejidad:** $\mathcal{O}(nz)$ o $\mathcal{O}(nz + nz_B)$.
- **multiply(x):** multiplicar con la matriz columna x (Ax). **Complejidad:** $\mathcal{O}(nz)$.

Compressed column: Multiply

Multiplica la matriz A con el vector columna x ($y = Ax$).

```
void multiply(sparseMatrix A, double *x,
              double *y){
    int nz = A.nz;
    for(int i = 0;i < nz;i++){
        int row_idx = A.row[i];
        int col_idx = A.col[i];
        double val = A.val[i];
        y[row] = y[row] + x[col_idx]*val;
    }
}
```

Compressed column: Transpose

Obtiene la tranpuesta de la matriz A.

```
void transpose(sparseMatrix *A){
     int nz = A->nz;
     swap(A->n,A->m);
     for(int i = 0;i < nz;i++){
          swap(A->row[i],A->col[i]);
     }
}
```

Listas enlazadas

La idea es enlazar las entradas diferentes de cero en cada fila para formar una lista enlazada (*row chain*).

Listas enlazadas

La idea es enlazar las entradas diferentes de cero en cada fila para formar una lista enlazada (*row chain*).

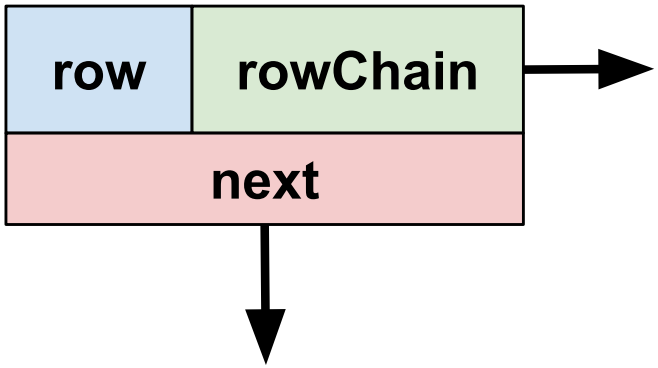


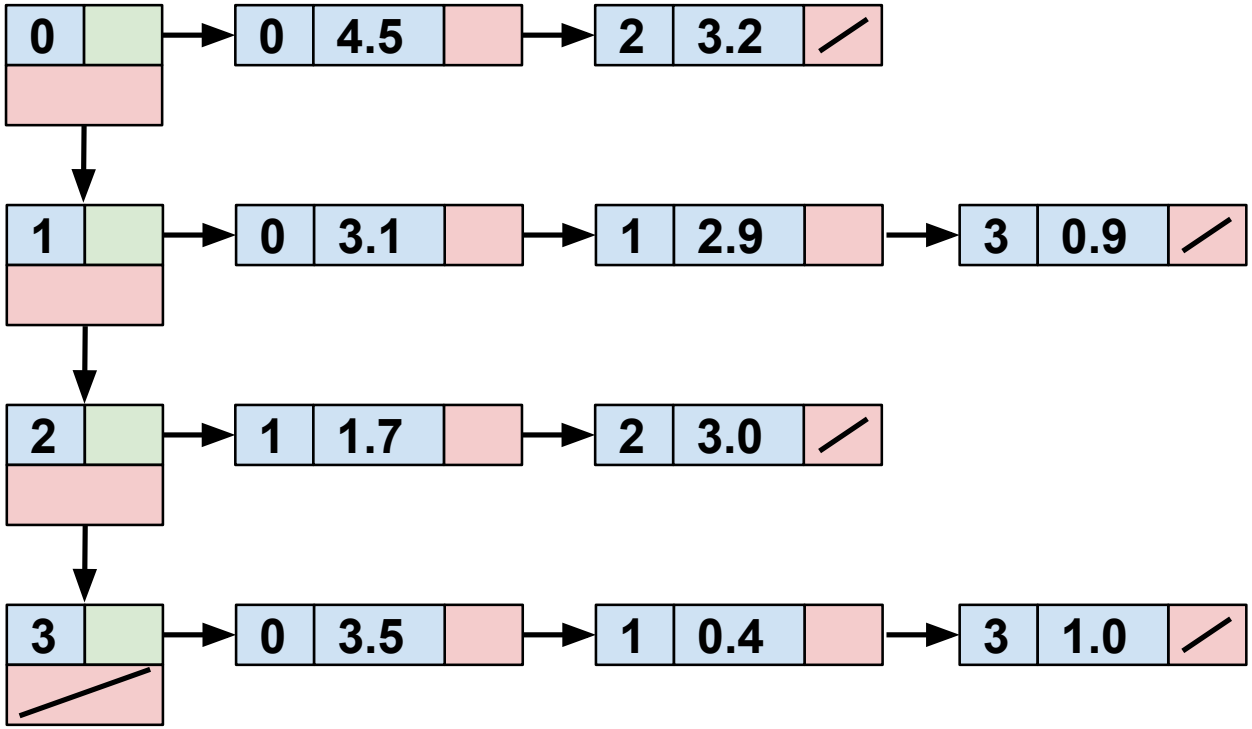
Figure 1: Row Header

Figure 2: rowChain

Listas enlazadas: Ejemplo

4.5	0	3.2	0
3.1	2.9	0	0.9
0	1.7	3.0	0
3.5	0.4	0	1.0

Listas enlazadas: Ejemplo



Listas enlazadas: Estructura

```
struct rowChain{
    int col; // indice de la columna
    double val; // valor almacenado
    rowChain* next; // siguiente columna
};

struct rowHeader{
    int row; // indice de la fila
    rowChain* rowElements; // elementos de la fil
    rowHeader* next; // siguiente fila
};
```

Listas enlazadas: Estructura

```
struct sparseMatrix{
    int m;  // número de filas
    int n; // número de columnas
    int nz; // número de entradas
    rowHeader* firstRow; // puntero a la primera
};
```

Listas Enlazadas: Operaciones

- **get(i,j):** Obtener el elemento de la posición (i, j). **Complejidad:** $\mathcal{O}(nz)$
- **set(i,j,x):** Cambiar el valor del elemento en la posición (i, j) por x. **Complejidad:** $\mathcal{O}(nz)$
- **transpose:** Obtener la transpuesta de la matriz. **Complejidad:** $\mathcal{O}(nz + n)$
- **sum(B):** sumar con la matriz B. **Complejidad:** $\mathcal{O}(nz)$ o $\mathcal{O}(nz + nz_B)$.
- **multiply(x):** multiplicar con la matriz columna x (Ax). **Complejidad:** $\mathcal{O}(nz)$.

Compressed column: Multiply

Multiplica la matriz A con el vector columna x.

```
void multiply(sparseMatrix A, double *x,
              double *y){
    rowHeader *ptr = A.firstRow;
    while(ptr != NULL){
        int row_idx = ptr->row;
        rowChain *ptr_row = ptr->rowElements;
        while(ptr_row != NULL){
            int col_idx = ptr_row->col;
            y[row_idx] += x[col_idx]*ptr_row->val
            ptr_row = ptr_row->next;
        }
        ptr = ptr->next;
    }
}
```

Compressed column

Esta forma es la más útil para la mayoría de algoritmos en matrices esparsas (es usada en la mayoría de funciones en MATLAB). Esta estructura tiene los siguientes elementos:

- **p**: arreglo de longitud $n + 1$
- **idx**: arreglo de longitud nz
- **val**: arreglo de longitud nz

Los índices de las filas de la columna j están almacenados desde $idx[p[j]]$ hasta $idx[p[j + 1] - 1]$, y sus valores en $val[p[j]]$ hasta $val[p[j + 1] - 1]$.

Compressed column: Ejemplo 1

4.5	**0**	**3.2**	**0**
3.1	**2.9**	**0**	**0.9**
0	**1.7**	**3.0**	**0**
3.5	**0.4**	**0**	**1.0**

Listas enlazadas: Ejemplo 1

p	0	3	6	8	10

	0	1	2	3	4	5	6	7	8	9
idx	0	1	3	1	2	3	0	2	1	3
val	4.5	3.1	3.5	2.9	1.7	0.4	3.2	3.0	0.9	1.0

Compressed column: Ejemplo 2

1.0	0	0	2.0	0
3.0	4.0	0	5.0	0
6.0	0	7.0	8.0	9.0
0	0	10.0	11.0	0
0	0	0	0	12.0

Compressed column: Ejemplo 2

p	0	3	4	6	10	12

	0	1	2	3	4	5	6	7	8	9	10	11
idx	0	1	2	1	2	3	0	1	2	3	2	4
val	1.0	3.0	6.0	4.0	7.0	10	2.0	5.0	8.0	11	9.0	12

Compressed column: Estructura

```
struct sparseMatrix{
    int m;  // número de filas
    int n; // número de columnas
    int nz; // número de entradas
    int *p; // columnas (n+1)
    int *idx; // indices de las filas
    double *val; // valores numéricos
};
```

Compressed column: Operaciones

- **get(i,j):** Obtener el elemento de la posición (i, j). **Complejidad:** $\mathcal{O}(nz)$
- **set(i,j,x):** Cambiar el valor del elemento en la posición (i, j) por x. **Complejidad:** $\mathcal{O}(nz)$
- **transpose:** Obtener la transpuesta de la matriz. **Complejidad:** $\mathcal{O}(nz + n)$
- **sum(B):** sumar con la matriz B. **Complejidad:** $\mathcal{O}(nz)$ o $\mathcal{O}(nz + nz_B)$.
- **multiply(x):** multiplicar con la matriz columna x (Ax). **Complejidad:** $\mathcal{O}(nz)$.

Compressed column: Multiply

Multiplica la matriz A con el vector columna x.

```
void multiply(sparseMatrix A, double *x,
              double *y){
    for(j = 0;j < A.n;j++){
        for(int p = A.p[j];p < A.p[j + 1];p++){
            y[A.idx[p]] += A.val[p]*x[j];
        }
    }
}
```

Tarea

- Proponga la estructura compressed row
- Implemente las operaciones:
  - **init(A)**: Que reciba como entrada una matriz 2D A y la convierta en una matriz esparsa.
  - **init(T)**: Que reciba como entrada un arreglo de triplas (posición fila, posición columna y valor) y que la transforme en una matriz esparsa.
  - **get(i,j)**: recupera el elemento de la posición (i, j)
  - **set(i,j,x)**: coloca el elemento x en la posición (i, j)
  - **transpose**: Transpuesta de una matriz
  - **multiply(x)**: multiplica la matriz con el vector columna x.
  - **multiply(A)**: multiplica la matriz con la matriz B.

Gracias por su atención!