

# 计算摄影学Lab3

姓名：葛帅琦

学号：3150102193

## 稀疏矩阵定义

对于那些零元素数目远远多于非零元素数目，并且非零元素的分布没有规律的矩阵称为稀疏矩阵（sparse）。人们无法给出稀疏矩阵的确切定义，一般都只是凭个人的直觉来理解这个概念，即矩阵中非零元素的个数远远小于矩阵元素的总数，并且非零元素没有分布规律。

由于稀疏矩阵中非零元素较少，零元素较多，因此可以采用只存储非零元素的方法来进行压缩存储。

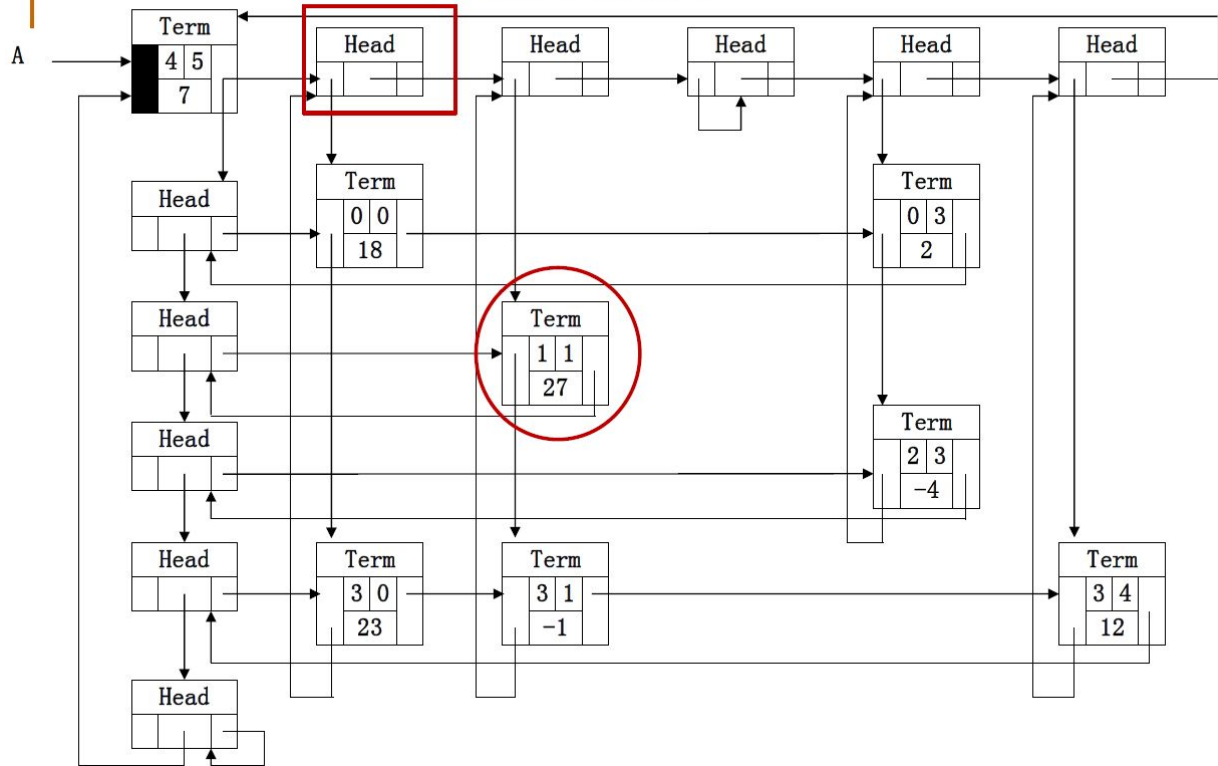
由于非零元素分布没有任何规律，所以在进行压缩存储的时候需要存储非零元素值的同时还要存储非零元素在矩阵中的位置，即非零元素所在的行号和列号，也就是在存储某个元素比如 $a_{ij}$ 的值的同时，还需要存储该元素所在的行号 $i$ 和它的列号 $j$ ，这样就构成了一个三元组 $(i, j, a_{ij})$ 的线性表。

三元组可以采用顺序表示方法，也可以采用链式表示方法，这样就产生了对稀疏矩阵的不同压缩存储方式。

## 稀疏矩阵的实现

主要采用十字链表的结构

## ❖ 矩阵A的多重链表图



Node的结构:

```

template<class T>
class Node
{
public:
    int i; // row id
    int j; // col id
    T v; // element value
    Node* right;
    Node* down;
    // construction for struct node
    Node(int i, int j, T v)
    {
        this->i = i;
        this->j = j;
        this->v = v;
        this->right = nullptr;
        this->down = nullptr;
    }
    Node()
    {
        this->i = 0;
        this->j = 0;
        this->v = 0;
        this->right = nullptr;
        this->down = nullptr;
    }
};
    
```

## 需要实现的目标:

1. `at(row, col)`: 根据row和column的系数来查询矩阵里面的元素的数值
2. `insert(val, row, col)`: 将val替换/插入到 (row, col) 这个位置去
3. `initializeFromVector(rows, cols, vals)`: 根据向量来初始化一个稀疏矩阵。其中 rows, cols, vals皆为等长度的向量。rows里面存的是行系数, cols里面存的是列系数, vals里面存的是数值。
4. 其余的基本功能可以参考Matlab里面的sparse函数, 或者Eigen Library里面的Sparse Matrix的介绍。

## Function AT:

```
/******
 * get the pointer of element by coordinate x,y
 */
T at(int i, int j)
{
    if (i < 0 || i >= this->row || j < 0 || j >= this->col) {
        cout << "out of range" << endl;
        return 0;
    }
    Node<T>* line = &data[i];
    Node<T>* ele = getElementofLine(line, j);
    if (ele == nullptr) {
        return 0;
    }
    return ele->v;
}
```

## Function insert:

```
/******
 * insert a nonzero value into matrix
 */
Node<T>* insert(T val, int i, int j) {
    // warning for saturation or wrong index i,j
    if (nonzero == MAXNonzero || i > row || j > col)
    {
        cout << "The matrix is saturation, fail to insert (" <<
i << ", " << j << ")" << endl;
        return nullptr;
    }
}
```

```

        if (val == 0) {
            //cout << "There is no point to insert a zero." << end
l;

            return nullptr;
        }

Node<T>* p = &data[i]; // find the head of row
while (p) {
    if (p->j == j) {
        if (p->v == 0 && val != 0)
        {
            nonzero++; // counter for nonzero
        }
        p->v = val; // assign the value to element of matri
x

        return p;
    }

    // insert between the nodes
    if(p->j < j && p->right && p->right->j > j )
    {
        Node<T>* newele= new Node<T>(i,j,val);
        nonzero++;
        newele->right = p->right;
        p->right = newele;
        return newele;
    }

    // insert at the end
    if ( p->right == nullptr)
    {
        Node<T>* newele = new Node<T>(i, j, val);
        nonzero++;
        p->right = newele;
        return newele;
    }
    p = p->right;
}

}

```

## InitializewithVector

```

/*****
* Initialize the matrix with 3 vectors

```

```

*/
void initializeFromVector(vector<int> rows, vector<int> cols, vector<T> vals)
{
    auto maxrow = max_element(rows.begin(), rows.end());
    auto maxcol = max_element(cols.begin(), cols.end());
    int size = *maxrow;
    int newrow = *maxrow + 1; // get the maximum row element from the vector
    int newcol = *maxcol + 1; // get the maximum col element from the vector
    Initialize(newrow, newcol); // initialize the matrix
    for (int i = 0; i < rows.size(); i++) {
        int ti = rows[i];
        int tj = cols[i];
        T value = vals[i];
        insert(value, ti, tj);
    }
}

```

其他实现诸如打印矩阵，获取行向量，获取列向量等。

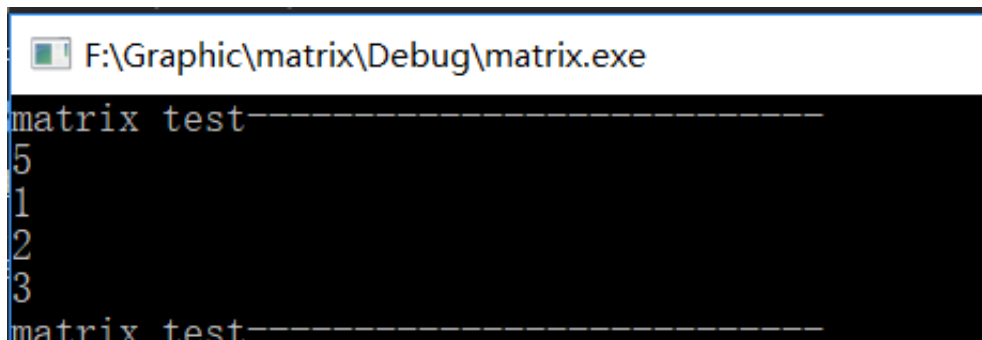
## 实验结果截图1

AT and insert:

```

cout << "matrix test-----" << endl;
Sparsematrix<int> s(4, 2, 2);
s.insert(5, 0, 0);
s.insert(1, 0, 1);
s.insert(2, 1, 0);
s.insert(3, 1, 1);
cout << s.at(0, 0) << endl;
cout << s.at(0, 1) << endl;
cout << s.at(1, 0) << endl;
cout << s.at(1, 1) << endl;

```



```

F:\Graphic\matrix\Debug\matrix.exe
matrix test-----
5
1
2
3
matrix test-----

```

## initializewithvector and printmatrix

```
cout << "matrix test-----" << endl;
vector<int> a = { 0,1,2 };
vector<int> b = { 0,1,2 };
vector<int> c = { 7,8,9 };
s.initializeFromVector(a, b, c);
s.printmatrix();
```

```
matrix test-----
matrix 3*3:
7 0 0
0 8 0
0 0 9
Gauss-Seidel method test-----
```

## 稀疏矩阵的高斯赛达尔迭代法

定义：

高斯-赛德尔迭代法是解线性方程组的常用迭代法之一，设线性方程组为

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n = b_i \\ (i = 1, 2, \cdots, n),$$

高斯-赛德尔迭代法的迭代公式为

$$x_i^{(k+1)} = \left( b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij}x_j^{(k)} \right) / a_{ii}, \\ (i = 1, 2, \cdots, n; k = 0, 1, 2, \cdots,)$$

当然，此处假定  $a_{ii} \neq 0 (i = 1, 2, \cdots, n)$ ，在很多情况下，它比简单迭代法收敛快，它和简单迭代法的不同点在于计算  $x_i^{(k+1)}$  时，利用了刚刚迭代出的

$x_1^{(k+1)}, x_2^{(k+1)}, \cdots, x_{i-1}^{(k+1)}$  的值，当系数矩阵 A 严格对角占优或对称正定时，高斯-赛德尔迭代法必收敛。<sup>[1]</sup>

代码实现：

```
/******
 * Solving Linear Equation with Gauss_Seidel theory
 */
void Gauss_Seidel(double B[], double X[])
{
    double **A = new double*[row];
    for (int i = 0; i < row; i++)
        A[i] = new double[col];
    int n = this->col;
```

```

        for (int i = 0; i < this->row; i++) {
            for (int j = 0; j < this->col; j++) {
                A[i][j] = at(i, j);
            }
        }
        for (int k = 0; k < 1000; k++)
        {
            for (int i = 0; i < n; i++)
            {
                double sum = 0;
                for (int j = 0; j < n; j++)
                {
                    if (j == i) continue; //跳过aii
                    sum += A[i][j] * X[j];
                }
                X[i] = (B[i] - sum) / A[i][i]; //计算完新的x[i],旧的
                x[i]会被自然冲掉
            }
        }
    }
}

```

## 实验结果截图2

$$A = \begin{pmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{pmatrix}$$

$$b = (6, 25, -11, 15)^T$$

求解  $Ax = b$  得到的结果应该为:

```
x = [1, 2, -1, 1];
```

```

Gauss-Seidel method test-----
matrix 4*4:
10  -1  2  0
-1  11 -1  3
2   -1 10 -1
0   3  -1  8
-----
1       2       -1       1
-----

```

## Bonus 实现共轭梯度法

定义：在数学中，共轭梯度法是特定线性方程组的数值解法的算法，即矩阵是对称

和正定的。共轭梯度法通常作为迭代算法实现，适用于稀疏系统，这些系统太大而无法通过直接实现或其他直接方法（如Cholesky分解）来处理。当数值求解偏微分方程或优化问题时，通常会出现大的稀疏系统。

算法：

```

$$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$$

$$\mathbf{p}_0 := \mathbf{r}_0$$

$$k := 0$$
repeat
$$\alpha_k := \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top \mathbf{A} \mathbf{p}_k}$$

$$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

$$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k$$
if  $r_{k+1}$  is sufficiently small, then exit loop
$$\beta_k := \frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{r}_k^\top \mathbf{r}_k}$$

$$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$$

$$k := k + 1$$
end repeat  
The result is  $\mathbf{x}_{k+1}$ 
```

代码实现：

```
void Conjugate_gradient_method(double B[],double X[]) {  
    vector<double> b;  
    vector<double> xk;  
    for (int i = 0; i < this->row; i++) {  
        b.push_back(B[i]);  
        xk.push_back(X[i]);  
    } // push the value of array into a vecor, which is convenient for computation  
    // the following is consistent with algorithm in wikipedia  
    int k = 0;  
    vector<double> rk = subvector(b, this->multiplyvector(xk));  
    vector<double> Pk = rk;  
    while (1) {  
        double alpha_k = vectormultiply(rk, rk) / vectormultiply(Pk, this->multiplyvector(Pk));  
        vector<double> xk1 = addvector(xk, coefficientvector(alpha_k, Pk));  
        xk = xk1;  
        vector<double> rk1 = subvector(rk, coefficientvector(alpha_k, this->multiplyvector(Pk)));  
        if (sumvector(rk1) < 0.000001 ) {  
            break;  
        }  
    }  
}
```



```

    }
    double betak = vectormultiply(rk1, rk1) / vectormultip
y(rk, rk);
    vector<double> Pk1 = addvector(rk1, coefficientvector(b
etak, Pk));
    Pk = Pk1;
    rk = rk1;
    k++;
}

cout << "solution: ";
for (int i = 0; i < this->row; i++) {
    X[i] = xk[i];
    cout << X[i] << " ";
}
cout << endl;
}
}

```

## 实验结果截图3

测试方程:

$$4x+3y=2$$

$$2x-y=1$$

```

-----
matrix 2*2:
4  3
2  -1
solution: 0.5  3.11798e-08
-----

```

测试方程:

$$7x+y=4$$

$$x-y=-3$$

```

-----
matrix 2*2:
7  1
1  -1
solution: 0.125  3.125
-----

```