2024 Data Structure - Homework 2: Transpose

A1115530 劉柏均

Introduce

Given a sparse matrix A, transpose it by the following three methods:

- Using traditional 2-dimensional array representation
- Using the “Transpose” method in the textbook
- Using the “FastTranspose” method in the textbook

Implement

```
int n,m ;       You, 1 小時前 • first commit
cin >> n >> m;
vector<MatrixTerm> terms;
vector<vector<int>> A(n,vector<int>(m));
int row,col,val;
while(cin >> row >> col >> val){
    A[row][col] = val;
    MatrixTerm term(row, col, val);
    terms.push_back(term);
};
SparseMatrix A_Matrix(n,m,terms.size());

for (int i = 0; i < terms.size(); i++){
    A_Matrix.addTerm(terms[i].GetRow(), terms[i].GetCol(), terms[i].GetValue(),i);
}
```

將測資傳入A vector 以及 term存入terms vector，並得到terms的大小建立SpareseMatrixy。

再將terms拿出來再放進A_Matrix。

Traditional 2-dimensional

```
vector<vector<int>> B(m,vector<int>(n));
for (int i = 0; i < m; i++){
    for (int j = 0; j < n; j++){
        B[i][j] = A[j][i];
    }
};
```

Transpose

```
SparseMatrix SparseMatrix::Transpose(){
    SparseMatrix b(rows, cols, terms);
    if (terms > 0){
        int currentB = 0;
        for (int c = 0; c < cols; c++)
            for (int i = 0; i < terms; i++)
                if (smArray[i].col == c) {
                    b.smArray[currentB].row = c;
                    b.smArray[currentB].col = smArray[i].row;
                    b.smArray[currentB++].value = smArray[i].value;
                }
    }
    return b;
```

Fast Transpose

```
SparseMatrix SparseMatrix::FastTranspose(){
    SparseMatrix b(cols, rows, terms);
    if (terms>0){
        int *rowSize = new int[cols];
        int *rowStart = new int[cols];
        fill(rowSize,rowSize+cols,0);
        for (int i = 0; i < terms; i++) {
            rowSize[smArray[i].col]++;
        }
        rowStart[0] = 0;
        for (int i = 1 ; i < cols ; i++) rowStart[i] = rowStart[i-1] + rowSize[i-1];
        for (int i = 0 ; i < terms ; i++){
            int j = rowStart[smArray[i].col];
            b.smArray[j].row= smArray[i].col;
            b.smArray[j].col= smArray[i].row;
            b.smArray[j].value = smArray[i].value;
            rowStart[smArray[i].col]++;
        }
        delete [] rowSize;
        delete [] rowStart;
    }
    return b;

}
```

Result

7_9.out

```
124   8 3 183
125   8 5 173
126   8 6 8
127    2-dimensional array use 0.006ms
128   Transpose use 0.001ms
129   FastTranspose use 0ms
130
```

15_12.out

```
289   11 11 116
290    2-dimensional array use 0.004ms
291   Transpose use 0.001ms
292   FastTranspose use 0.001ms
293
```

60_74.out

```
8251   73 54 207
8252   73 55 9
8253   73 59 105
8254    2-dimensional array use 0.042ms
8255   Transpose use 0.164ms
8256   FastTranspose use 0.031ms
8257
```

100_100.out

```
17118   99 96 142
17119   99 97 210
17120   99 98 89
17121   99 99 104
17122    2-dimensional array use 0.085ms
17123   Transpose use 0.394ms
17124   FastTranspose use 0.055ms
17125
```

256_512.out

```
268825   511 252 111
268826   511 254 178
268827   511 255 214
268828    2-dimensional array use 1.177ms
268829   Transpose use 36.234ms
268830   FastTranspose use 1.094ms
268831
```

721_850.out

```
1107315   849 716 91
1107316   849 717 192
1107317   849 718 75
1107318   849 719 173
1107319   849 720 62
1107320    2-dimensional array use 6.017ms
1107321   Transpose use 211.933ms
1107322   FastTranspose use 3.285ms
1107323
```

Discussion

各測資使用這三種方法所花費的時間，單位毫秒(ms)

	7_9	15_12	60_74	100_100	256_512	721_850
Traditional 2-dimensional	0.006	0.004	0.42	0.085	1.177	6.017
Transpose	0.001	0.001	0.164	0.394	36.234	211.933
Fast Transpose	$< 1\mu s$	0.001	0.031	0.055	1.094	3.285

由上表可知大部分時候所耗時間Fast Transpose快於traditional 2-dimensional快於Transpose，但在$7 * 9$ 和 $15 * 12$ 中，Transpose快於traditional 2-dimensional，我執行了很多次結果也是一樣