A library for operations with sparse matrices
Technical Report

Ionescu Iulian Ştefan

December 1, 2016

Student Name: Ionescu Iulian Ştefan
Group: 10105B
Year: I
Section: Computers and Information Technology - English

1 Problem Statement

The task of the project is to realise a library for operations with sparse matrices.

The library will provide the addition and the product of sparse matrices, storing the matrix in a compressed form ,create a matrix from the compressed form , display and reading matrices stored in both forms.The project will also provide the random generation of two sparse matrices which will be used for the previous operations.

2 Pseudocode

The most important algorithms used for sparse matrices operations will be described here in pseudocode.

2.1 Conversion from normal matrix to a sparse form

SparseMatrix_Conversion(mat,m,n)
1. $nrElements \leftarrow 0$
2. $MatRIndex \leftarrow 0$
3. $\triangleright$ The number of nenull elements it is stored in $nrElements$
4. $\triangleright$ Allocate memory for structure arrays $matR.Line$,$matR.Column$,$matR.Value$
5. $matR.nrElements \leftarrow nrElements$
6. $matR.nrLines \leftarrow m$
7. $matR.nrColumns \leftarrow n$
8. **for** $i \leftarrow 0$,$m-1$ **do**
9. **for** $j \leftarrow 0$,$n-1$ **do**
10. **if** $mat[i][j] \neq 0$ **then**
11. $matR.Line[MatRIndex] \leftarrow i$
12. $matR.Column[MatRIndex] \leftarrow j$
13. $matR.Value[MatRIndex] \leftarrow mat[i][j]$
14. $MatRIndex \leftarrow MatRIndex + 1$
15. **return** $matR$

2.2 Addition of two sparse matrices

SparseMatrix_Addition($M1$,$M2$)
1. ▷ Allocate memory for structure variable R and its fields and initialize the fields
2. $i \leftarrow 0$
3. $j \leftarrow 0$
4. $k \leftarrow 0$
5. **if** $M1.nrLines = M2.nrLines$ and $M1.nrColumns = M2.nrColumns$ **then**
6. **while** go through both matrices **do**
7. **if** found nenull elements in the same position **then**
8. **if** summ of elements is null **then**
9. $nr \leftarrow nr + 2$
10. **else**
11. $nr \leftarrow nr + 1$
12. ▷ substract the number of opposite sign elements (elements with null summ)
13. $nr \leftarrow M1.nrElements + M2.nrElements - nr$
14. $R.nrElements \leftarrow nr$
15. $R.nrLines \leftarrow M1.nrLines$
16. $R.nrColumns \leftarrow M1.nrColumns$
17. $i \leftarrow 0$
18. $j \leftarrow 0$
19. $k \leftarrow 0$
20. **while** $k < R.nrElements$ **do**
21. **if** found a nenull element in one of the matrices **then**
22. copy in R the value
23. copy in R position
24. advance in matrix
25. $k \leftarrow k + 1$
26. **else**
27. **if** found nenull elements in same position and summ is null **then**
28. advance in both matrices
29. **else**
30. copy in R the summ
31. copy in R the position
31. advance in both matrices
32. $k \leftarrow k + 1$
33. **return** R
34. Deallocate all fields of structure
35. **else**
36. ▷The condition of matrices summ is not satisfied so it returns the structure with initialized fields
37. **return** R

2.3 Product of two sparse matrices

```
SparseMatrix_Product(M1,M2)
1. Allocate memory for structure variable R and its fields and initialize the fields
2. nr ← 0
3. index ← 0
4. ▷Test if the the condition of matrices product is satisfied
5. if M1.nrColumns ≠ M2.nrLines then
6.    return R
7. for i ← 0,M1.nrLines − 1 do
8.      for j ← 0,M2.nrLines − 1 do
9.          summ ← 0
10.         for col ← 0,M1.nrColumns − 1 do
11.             for each element of M1 sparse matrix do
12.                 if element is in position i, col then
13.                    for each element of M2 sparse matrix do
14.                        if element is in position col, j then
15.                           sum ← sum + M1.Value ∗ M2.Value
16.         if sum ≠ 0then
17.            nr ← nr + 1
18. R.nrElements ← nr
19. R.nrLines ← M1.nrLines
20. R.nrColumns ← M2.nrColumns
21.for i ← 0,M1.nrLines − 1 do
22.     for j ← 0,M2.nrLines − 1 do
23.         summ ← 0
24.         for col ← 0,M1.nrColumns − 1 do
25.             for each element of M1 sparse matrix do
26.                 if element is in position i, col then
27.                    for each element of M2 sparse matrix do
28.                        if element is in position col, j then
29.                           sum ← sum + M1.Value ∗ M2.Value
30.         if sum ≠ 0then
31.            R.Line[index] ← i
32.            R.Column[index] ← j
33.            R.Value[index] ← summ
34.            index ← index + 1
35. return R
36. Deallocate all fields of structure
```

2.4 Conversion from a sparse matrix to a normal matrix

Sparse_to_Normal($matR$)
1. Allocate memory for matrix
2. Initialize matrix with null elements
3. **for** $i \leftarrow 0$,$matR.nrElements - 1$ **do**
4. $\quad mat[matR.Line[i]][matR.Column[i]] \leftarrow matR.Value[i]$
5. **return** mat

2.5 Printing a matrix in sparse form

print_sparse_matrix($sparse$)
1. **if** $sparse.nrElements \neq 0$ **then**
2. **for** $i \leftarrow 0$,$sparse.nrElements - 1$ **do**
3. $\quad$ **write** $sparse.Line[i]$
4. **for** $i \leftarrow 0$,$sparse.nrElements - 1$ **do**
5. $\quad$ **write** $sparse.Column[i]$
6. **for** $i \leftarrow 0$,$sparse.nrElements - 1$ **do**
7. $\quad$ **write** $sparse.Value[i]$
8. **else**
9. **write** *Matrix has only null elements*

2.6 Void type function which uses the functions described above to make operations with the generated matrices

resolve($NormalMat1$,$NormalMat2$,m,n,p,q,$choice$)
1. Open file for writing
2. $R1 \leftarrow$ SparseMatrix_Conversion($NormalMat1$,m,n)
3. $R2 \leftarrow$ SparseMatrix_Conversion($NormalMat2$,p,q)
4. print_matrix($NormalMat1$,m,n)
5. print_matrix($NormalMat2$,p,q)
6. print_sparse_matrix($R1$)
7. print_sparse_matrix($R2$)
8. $\triangleright$Using the parameter *choice* and the variable *auxiliar*
9. $\quad$ it is randomly chosen which operation will be applied on matrices
10. $auxiliar \leftarrow$ random value between 0 and 1
11. **if** $choice = 1$ or ($auxiliar = 1$ and $choice \neq 2$) **then**
12. $\quad$ **write** Addition of matrices
13. $\quad R\_summ \leftarrow$ Sparse_Addition($R1$,$R2$)
14. $\quad$ **if** condition of adding is satisfied **then**
15. $\quad\quad Summ_Matrix_Normal \leftarrow$Sparse_to_Normal($R\_Summ$)
16. $\quad\quad$ print_matrix($Summ\_Matrix\_Normal$,$R\_summ.nrLines$,$R\_summ.nrColumns$)
17. $\quad\quad$ print_sparse_matrix(R_summ)
18. $\quad$ **else**

19. **write** Error: The matrices doesn't have the same number of lines and columns
20. **write** Matrix1 lines columns: m, n
21. **write** Matrix2 lines columns: p, q

22.**else**

23. **if** choice=2 or auxiliar=0 **then**
24. **write** Multiplication of matrices
25. $R\_product$ ←SparseMatrix_Product($R1$,$R2$)
26. **if** condition of multiplication is satisfied **then**
27. $Product\_Matrix\_Normal$ ←Sparse_to_Normal($R\_product$)
28. print_matrix($Procut\_Matrix\_Normal$,$R\_product.nrLines$,$R\_product.nrColumns$)
29. print_sparse_matrix($R\_product$)
30. **else**
31. **write** Error: The dimensions of matrices doesn't match
32. **write** Matrix1 lines columns: m, n
33. **write** Matrix2 lines columns: p, q

2.7 Generate a sparse matrix

generate_matrix(mat,m,n)

1.$elements\_number$ ←0.4 * (m*n)

2.**while** $elements\_numar \neq 0$ **do**

3. $element$ ← random value between 1 and 1000
4. $Line$ ← random value between 1 and m
5. $Column$ ← random value between 1 and n
6. $mat[Line][Column]$ ← $element$
7. $elements\_number$ ← $elements\_number - 1$

2.8 Function used to provide the output

generate_input(*choice*)
1.**if** *choice* = 1 **then**
2. ▷Generate matrices used for summ, so dimensions are the same
3. m ←random value between 1 and 5
4. n ←random value between 1 and 5
5. Allocate a m x n matrix
6. generate_matrix($mat1$,m,n)
7. **write** $mat1$ in file
8. Allocate a m x n matrix
9. generate_matrix($mat2$,m,n)
10. **write** $mat2$ in file
11. **else**
12. **if** *choice* = 2 **then**
13. ▷Generate matrices used for product, so dimensions matches
14. m ←random value between 1 and 5
15. n ←random value between 1 and 5
16. p ←random value between 1 and 5
17. Allocate a m x n matrix
18. generate_matrix($mat1$,m,n)
19. **write** $mat1$ in file
20. Allocate a n x p matrix
21. generate_matrix($mat2$,n,p)
22. **write** $mat2$ in file
23. **else**
24. ▷Generate completely random matrices,so the dimension could not correspond
25. m ←random value between 1 and 5
26. n ←random value between 1 and 5
27. p ←random value between 1 and 5
28. q ←random value between 1 and 5
29. Allocate a m x n matrix
30. generate_matrix($mat1$,m,n)
31. **write** $mat1$ in file
32. Allocate a p x q matrix
33. generate_matrix($mat2$,p,q)
34. **write** $mat2$ in file

3 Application design

3.1 High-level overview of the application

The main goal of the application is to make several operations with sparse matrices, like :

- Store a sparse matrix in a special form, where only the nenull elements are memored, by line position, column position and value
- Create a normal represented matrix, using the described form
- The summ of two sparse matrices
- The product of two sparse matrices
- Print a sparse matrix
- Generate sparse matrices

Using these operations , the program will generate two sparse matrices, print them in normal and compressed form, implement the summ or the product of the matrices and print the resulted matrix in both forms.

3.2 Specification of the input

The input will consists in two matrices and their dimensions. The input will not be generated entirely random, because the condition for matrices summ or product will not be satisfied in most of the cases. Using some maneuver variables, the dimensions of the matrices will be generated to satisfy one of the operations or completely random dimensions, to not exclude the fail case.

3.3 Specification of the output

The output will consists in displaying of the matrices in both forms and the resulted matrix from the executed operations, also in both forms. If the condition of summ or product wasn't satisfied an error message will be displayed, also in the case when the generated matrix has only null elements.

3.4 The modules and their description

- main.c
- functions.c
- generare.c
- functions.h

i) **Main.c**

In the C file **main.c** there are called functions to generate input,allocate and read the matrices from input, resolve the operations on matrices and print the results in an output file. All this process wil be done how many times we want, by choosing a number of testes.

ii) **Functions.c**

In the C file **functions.c** there are provided functions to allocate and read matrices, store them in the described form, create a normal matrix from the compressed form , print them in both forms, implement summ and product of the matrices,deallocate structure fields and a final function which uses the previous functions and provides the output.

iii) **Generare.c**

In the C file **generare.c** there are provided functions used to generate a sparse matrix,to allocate a matrix, to print a matrix in a file and a function which includes the previous functions to create the input such that there will be matrices specially generated for summ or for product or entirely random generated matrices, to not exclude the fail case.

iv) **Functions.h**

In the header file **functions.h** there are the declarations of all the functions and the structure used to store the sparse matrices in a compressed form.

3.5 The functions and their description

The functions from functions.c :

- **void freeEverything(SparseMatrix** $sparse$**)**
- **SparseMatrix SparseMatrix_Conversion(int**$**$ mat**, int**m**, int**n**)**
- **SparseMatrix SparseMatrix_Addition(SparseMatrix** $*M1$**,SparseMatrix** $*M2$**)**
- **SparseMatrix SparseMatrix_Product(SparseMatrix** $*M1$**, SparseMatrix** $*M2$**)**
- **void print_matrix(int** $**$ $matrice$**,int** m**,int** n**,FILE** $*g$**)**
- **void print_sparse_matrix(SparseMatrix** $sparse$**,FILE** $*g$**)**
- **void allocate_and_read(int** $***matrice1$**,int** $***matrice2$**,int**$*m$**,int**$*n$**,int** $*p$**,int** $*q$**)**
- **int ** Sparse_to_Normal(SparseMatrix** $*matR$**)**
- **int ERROR(SparseMatrix** R**)**

- **void resolve(int** $**NormalMat1$**,int** $**NormalMat2$**,int** m**,int** n**,int** p**,int** q**,int** $choice$**)**

The functions from generare.c:

- **int generate_number(int** x**)**
- **void generate_matrix(int** $**mat$**,int** m**,int** n**)**
- **void write_matrix(int** $**mat$**,int** m**,int** n**,FILE** $*f$**)**
- **void allocate(int** $***mat$**,int** m**,int** n**)**
- **void generate_input(int** $choice$**)**

The description of functions:

- **void freeEverything(SparseMatrix** $sparse$**)** Deallocate memory used for structure variable and the structure fields. The given paramater is a structure.
- **SparseMatrix SparseMatrix_Conversion(int**$**mat$**, int** m**, int**n**)**
  Computes the conversion of a matrix to a form where only the nenull elements are stored with their positions, using a strucutre . It returns the resulted structure.
  **mat : Double pointer which stores the elements of matrix
  m,n : The number of lines and columns of matrix
- **SparseMatrix SparseMatrix_Addition(SparseMatrix** $*M1$**,SparseMatrix** $*M2$**)**
  Computes the addition of two sparse matrices. It returns the resulted matrix as a structure.
  *M1,*M2 : Two matrices stored in structure used for addition,transmitted by reference
- **SparseMatrix SparseMatrix_Product(SparseMatrix** $*M1$**, SparseMatrix** $*M2$**)**
  Computes the product of two sparse matrices stored using the structure. It returns the resulted matrix by a structure. *M1,*M2 : Two structures used to store two sparse matrices used for product,transmitted by reference
- **void print_matrix(int** $**matrice$**,int** m**,int** n**,FILE** $*g$**)**
  It prints a matrix in the file specified by *g file pointer
  ***matrice : A double pointer transmitted by reference, used to store the elements of matric
  m,n : Dimensions of the matrix
  *g : The file pointer used to print in the file

- **void print_sparse_matrix(SparseMatrix** $sparse$**,FILE** $*g$**)**
  It prints a sparse matrix ih this form: the line index, the column index and the nenull value.
  sparse : The sparse matrix stored with structure
  *g : The file pointer used to print in the file

- **void allocate_and_read(int** $***matrice1$**,int** $***matrice2$**,int**$*m$**,int**$*n$**,int** $*p$**,int** $*q$**)**
  It allocates and reads two matrices.The elements of the matrices are read from an input file. ***matrice1, ***matrice2 : Two double ponters transmitted by reference, used to to store the elements of matrices
  *m,*n,*p,*q : The dimensions of the matrices

- **int ** Sparse_to_Normal(SparseMatrix** $*matR$**)**
  Create a normal matrix from the compressed form.
  *matR : Matrix stored in structure, transmitted by reference.

- **int ERROR(SparseMatrix** R**)**
  If the fields of structure are not filled, results that the addition or the product operation failed so the function returns 1 or 0. 1 if the operation failed , 0 if didn't/
  R : The compressed matrix used for testing

- **void resolve(int** $**NormalMat1$**,int** $**NormalMat2$**,int** m**,int** n**,int** p**,int** q**,int** $choice$**)**
  Function where the above functions are called. It will print,add or multiply matrices and print thre result in an output file.
  NormalMat1,NormalMat2 : Two double pointers used for the matrices operations
  m,n,p,q : The number of lines and columns of the matrices
  choice : Paremeter used to choose what operation from summ and product will be done.

- **int generate_number(int** x**)**
  Function used to generate a random number between 0 and x, using rand() function. x : the end of the interval

- **void generate_matrix(int** $**mat$**,int** m**,int** n**)**
  Function used to create a sparse matrix by generating a small number of nenull elements and their positions in the matrix.
  ** mat : Double pointer used to store the elements of the matrix.
  m,n : Dimensions of matrix.

- **void write_matrix(int** $**mat$**,int** m**,int** n**,FILE** $*f$**)**
  Function used to write a matrix in the file specified by the file pointer parameter.

** mat : Double pointer used to store the elements of the matrix.
m,n : Dimensions of matrix.
*f : The file pointer which indicates the file that will be written.

- **void allocate(int** $***mat$**,int** m**,int** n**)**
  Function used to allocate a matrix unsing a double pointer transmitted by reference.
  *** mat : The double pointer used to store the elements of the matrix, transmitted by reference.
  m,n : Dimensions of matrix.

- **void generate_input(int** *choice***)**
  Function that uses the above functions to provide the input, such that the conditions for summ or product of matrices will be satisfied in equal measure, also for the fail case, when the dimensions are entirely random generated.
  choice : Parameter used to decide for which case will be generated the input: matrices summ, matrices product or fail case.

4 Conclusions

In this technical report, it was presented basic operations with sparse matrices.

The method used to implement this operations was efficient, because we didn't use the entire form of a matrix,just the nenull elements and their positions, stored using a structure.

In my opinion, the most challenging part was the implementation of matrices product, because we had to go through matrices using just the position of nenull elements. That involed a lot of attention in choosing the correct position indices. The most interesting part was the implementation of sparse matrix generation. We had to choose an algotihm that provide us a matrix with most of the elements beeing null. It was generated a small number of nenull elements and their positions.

It is known that a sparse matrix has 80% of elements zeroes, but we decreased this procentage to 60%, because sometimes the generated position for two elements was the same. The future directions of the project could involve the transpose of a sparse matrix, the product of a scalar with a sparse matrix, the product of a line vector with a sparse matrix.In long term the project will include a GUI menu to choose what operation will be applied on matrices.

5 Experiments and results

5.1 Description of output

The output content begins with "New Test" to know when a new test was executed, then two matrices printed in normal form and in the compressed form, the name of the operation that will be applied , the resulted matrix of the operation represented in both forms.

To verify if the output is correct we can test the algorithm on a trivial input and verify the results or we can create a program that implements these operations, this time using the matrices on their entire form and compare the results on the same input.

5.2 Results

Example of output for matrices product:
NEW TEST

Matrix 1:
0 5 0
0 0 4
0 0 0
0 0 9
Matrix 2:
0 2 0
8 0 0
0 5 0
0 0 2

Print as sparse matrix

Matrix 1:
0 1 3
1 2 2
5 4 9
Matrix 2:
0 1 2 3
1 0 1 2
2 8 5 2

Addition of matrixes

The summ matrix represented as normal matrix
0 7 0
8 0 4
0 5 0
0 0 11

The summ matrix represented as sparse matrix
0 1 1 2 3
1 0 2 1 2
7 8 4 5 11

NEW TEST

Matrix 1:
6 6 0
0 0 9
0 0 0
Matrix 2:
7 0 4 0 0
0 1 0 0 0
0 0 0 9 5

Print as sparse matrix

Matrix 1:
0 0 1
0 1 2
6 6 9
Matrix 2:
0 0 1 2 2
0 2 1 3 4
7 4 1 9 5

Multiplication of matrices

The product matrix represented as normal matrix

42 6 24 0 0
0 0 0 0 0
0 0 0 0 0
The product matrix represented as sparse matrix

0 0 0
0 1 2
42 6 24

References

[1] Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.

[2] SparseMatrix Documentations,https://en.wikipedia.org/wiki/Sparse_matrix/, accessed in May 2016.

[3] LaTeXproject site, http://latex-project.org/, accessed in May 2016.