Experiment 5
Sparse Matrix

Date: 21-09-2020

Aim: To receive two sparse matrices and print their transpose and their sum

Data Structure Used: Arrays

Operation Used: Comparisons, Addition

Algorithm for reading a Sparse matrix and obtaining the tuple representation:

Input: A Sparse matrix A, containing m rows and n columns
Output: Tuple Representation of the array

```
Step 1  : Start
Step 2  : Receive the sparse matrix A
Step 3  : Initialize the array sp which will contain the tuple representation of A
Step 4  : i ← 0
Step 5  : j ← 0
Step 6 : count ← 0
Step 7  : while i<m
        Step 1 : j ← 0
        Step 2 : while j<n
                Step 3 : if A[i][j]!=0
                        Step 1  : count ++
                        Step 2  : sp[count][0] ← i
                        Step 3  : sp[count][1] ←  j
                        Step 4  : sp[count][2] ← A[i][j]
                Step 4 : Endif
                Step 5: j++
        Step 3 : EndWhile
        Step 4 : i++
Step 8  : EndWhile
Step 9  : sp[0][0] ← m
Step 10 : sp[0][1] ← n
Step 11 : sp[0][2] ←  k
Step 12 : Stop
```

Description of the algorithm

The elements of the sparse Matrix A is iterated one by one and the ones which are non-zero is pushed into the tuple representation array, sp.

Algorithm For Transpose:

Input: Sparse matrix, A in sequential tuple representation
Output: Sparse matrix, A_T in tuple representation of the transpose of the input Sparse matrix

```
Step 1  : Start
Step 2  : Receive Sparse Matrix in tuple format
Step 3  : Initialize the array A_T
Step 4  : m ← A[0][0]           //The number of rows in A
Step 5  : n ← A[0][1]           //The number of columns in A
Step 6  : t  ← A[0][2]          //The number of non-zero elements in A
Step 7  : i ← 0                 //Control Variable to iterate through the columns in A
Step 8  : while i<n
        Step 1  : j ← 1         //Control Variable to iterate through the nonzero elements in A
        Step 2  : while j<=t
```

```
                Step 1  :  if A[j][1] = i:
                        Step 1 : k++
                        Step 2: A[j][0]  ←  A_T[k][1]
                        Step 3 : A[j][1] ←  A_T[k][0]
                        Step 4 : A[j][2] ← A_T[k][2]
                Step 2: End if
        Step 9  : EndWhile
        Step 10 : A_T [0][0]← n
        Step 11: A_T [0][1] ← m
        Step 12: A_T [0][2] ← t
        Step 13: Stop
```

Description of the algorithm:

For finding the transpose we require two loops one to keep track of the columns and another loop inside the first to iterate through the non-zero elements of the sparse array A. When the outer loop i is 0 then the inner loop checks for non-zero elements in the first column and puts them in the array A_T. Since the elements in the array A are stored row-wise the transpose matrix will have the elements in order.

<u>Algorithm for Adding two Sparse Array:</u>

Input: Two sparse array (A,B) in tuple representation
Output: Sum of the two arrays in tuple representation, C

```
Step 1  : Start
Step 2  : Receive the two sparse array in tuple representation
Step 4  : Initialize the array C to store the sum of A and B
Step 3  : i← 1                          //Pointer to the non-zero elements of array A
Step 4  : j← 1                          //Pointer to the non-zero elements of array B
Step 5  : if (A[0][0] = B[0][0] and A[0][1] = B[0][1])  //Arrays can only be added if the rows and columns are equal
          Step 1  :  k← 0                       //Variable to count the number of elements in C
          Step 2  : while i<=A[0][2] or j<=B[0][2]
                Step 1 : if A[i][0] < B[i][0]
                        Step 1  : k++
                        Step 2  : C[k][0] ← A[i][0]
                        Step 3  : C[k][1] ← A[i][1]
                        Step 4  : C[k][2] ←  A[i][2]
                        Step 5  : i++
                Step 2  : else if A[i][0] > B[j][0]
                        Step 1  : k++
                        Step 2  : C[k][0] ← B[j][0]
                        Step 3  : C[k][1] ← B[j][1]
                        Step 4  : C[k][2] ←  B[j][2]
                        Step 5  : j++
                Step 3  : else if A[i][1] < B[j][1]
                        Step 1  : k++
                        Step 2  : C[k][0] ← A[i][0]
                        Step 3  : C[k][1] ← A[i][1]
                        Step 4  : C[k][2] ←  A[i][2]
                        Step 5  : i++
                Step 4  : else if A[i][1] > B[j][1]
                        Step 1  : k++
                        Step 2  : C[k][0] ← B[j][0]
                        Step 3  : C[k][1] ← B[j][1]
                        Step 4  : C[k][2] ←  B[j][2]
                        Step 5  : j++
                Step 5: else
                        Step 1  : k++
                        Step 2  : C[k][0] ← A[i][0]
                        Step 3  : C[k][1] ← A[i][1]
```

Step 4 : C[k][2] ← A[i][2]+B[j][2]
Step 5 : i++
Step 6 : j++
Step 6 : Endif
Step 3: EndWhile
Step 4 : C[0][0] ← A[0][0]
Step 5 : C[0][1] ← A[0][1]
Step 6 : C[0][2] ← k
Step 6 : else
Step 1 : Print “The arrays cannot be added”
Step 7 : End if
Step 8 : Stop

Description of the algorithm:

Two pointers to the sparse arrays is taken and, the smallest row value pointed by the any of the two pointers is added to the resulting array C, If the row values are the same then the column values are compared and the one with the smallest column value is add to the array, if the column values are also same then the non-zero element is added and the resulting value is add to the resultant matrix C.

Result: the Program is successfully compiled and the desired output is obtained.

Program/ Source Code:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_SIZE 10

void printSparse(int **sp){
int i;
    printf("The sparse array is \n");
    for(i=0;i<=sp[0][2];i++){
        printf("[%d] %d  %d  %d \n",i,sp[i][0],sp[i][1],sp[i][2]);fflush(stdout);
    }
}

int** readSparse(){
    int** sp = (int**) malloc(MAX_SIZE*sizeof(int*));
    int m,n;
    int i,j,k = 0;
    int A[10][10];

    for (i = 0;i<MAX_SIZE;i++)
        sp[i] = (int*)malloc(3*sizeof(int));

    //get the values for rows and columns in A
    printf("Enter the number of rows and columns-> ");
    scanf("%d%*c%d%*c",&m,&n);

    //Receive values for the sparse matrix A
    printf("Enter the values of the sparse matrix A\n");
    for(i=0;i<m;i++){
        printf("Values for row %d -> ",i+1);
        for(j=0;j<n;j++){
            scanf("%d%*c",&A[i][j]);

            //find the triplet represent of the sparse matrix
            if(A[i][j]!=0){
                k++;
                sp[k][0] = i;
                sp[k][1] = j;
                sp[k][2] = A[i][j];
            }

        }
    }

    sp[0][0] = m;
    sp[0][1] = n;
    sp[0][2] = k;
    return sp;
}

void transpose(int** sp1){
    int m = sp1[0][0];
    int n = sp1[0][1];
    int count = sp1[0][2];
```

```
    int **sp_t = (int**)malloc(MAX_SIZE*sizeof(int*));

    for(int i = 0;i<MAX_SIZE;i++){
       sp_t[i] = (int*) malloc(MAX_SIZE*sizeof(int));
    }

    if(count == 0){
        printf("The array is empty ");
        free(sp_t);
        return;
    }
    else{
        sp_t[0][0]=sp1[0][1];
        sp_t[0][1]=sp1[0][0];
        sp_t[0][2]=sp1[0][2];

        int i,j,k=1;

        for(i = 0;i<n;i++){
            for(j = 1;j<=count;j++){
                if(sp1[j][1]==i){
                    sp_t[k][0] = sp1[j][1];
                    sp_t[k][1] = sp1[j][0];
                    sp_t[k][2] = sp1[j][2];
                    k++;
                }
            }
        }
        printSparse(sp_t);
    }
}

int** add(int** A, int** B){
    int **c;
    int size = A[0][2]+B[0][2]+2;
    int i,j;
    int count = 0;

    if(A[0][0]==B[0][0]&&A[0][1]==B[0][1]){
        printf("Arrays can be added\n");fflush(stdout);

        c  = (int**) malloc(size*sizeof(int*));
        for(i=0;i<size;i++){
            c[i] = (int*) malloc(3*sizeof(int));
        }

        c[0][0] = A[0][0];c[0][1] = A[0][1];

        i=1;j=1;
        while(i<=A[0][2]&&j<=B[0][2]){
            if(A[i][0]<B[j][0]){
                count++;
                c[count][0] = A[i][0];
                c[count][1] = A[i][1];
                c[count][2] = A[i][2];
                i++;
```

```
        }
        else if(A[i][0]>B[j][0]){
            count++;
            c[count][0] = B[j][0];
            c[count][1] = B[j][1];
            c[count][2] = B[j][2];
            j++;
        }
        else{
            if(A[i][1]<B[i][1]){
                count++;
                c[count][0] = A[i][0];
                c[count][1] = A[i][1];
                c[count][2] = A[i][2];
                i++;
            }
            else if(A[i][1]>B[j][1]){
                count++;
                c[count][0] = B[j][0];
                c[count][1] = B[j][1];
                c[count][2] = B[j][2];
                j++;
            }
            else{
                count++;
                c[count][0] = B[j][0];
                c[count][1] = B[j][1];
                c[count][2] = B[j][2]+A[i][2];
                i++;j++;
            }
        }
    }

while(i<=A[0][2]){
        count++;
        c[count][0] = A[i][0];
        c[count][1] = A[i][1];
        c[count][2] = A[i][2];
        i++;
    }

while(j<=B[0][2]){
        count++;
        c[count][0] = B[j][0];
        c[count][1] = B[j][1];
        c[count][2] = B[j][2];
        j++;
    }
    c[0][2] = count;
    return c;
}

else{
    printf("Matrices cant be added");
    free(A);
    free(B);
```

```
        exit(0);
        return 0;
    }
}
void main(){
    int i,j,k=0;
    int A[10][10];
    int **sp1;
    int **sp2;
    int m,n;    //m is the number of rows and n is the number of columns

    int** sum;
    char c;
        sp1 = readSparse();
        sp2 = readSparse();
        //terminate the loop if n is entered
        printSparse(sp1);
        printSparse(sp2);

        //Print the transposes
        printf("\nTranspose of the first matrix is \n");
        transpose(sp1);
        printf("\nTranspose of the second matrix is \n");
        transpose(sp2);

        //Find the sum of the matrices
        sum = add(sp1,sp2);

        printf("Sum of both the matrices is \n");
        printSparse(sum);
        free(sp1);
        free(sp2);
        free(sum);
}
```

Sample Input/Output

Sample input 1:

```
4 5
0 1 0 0 0
0 0 2 1 0
0 0 0 0 0
0 3 0 0 1
4 5
1 0 1 0 0
0 0 0 0 2
0 0 3 0 0
1 0 0 1 0
```

Sample output 1:

```
Enter the number of rows and columns-> 4 5
Enter the values of the sparse matrix A
Values for row 1 -> 0 1 0 0 0
```

```
Values for row 2 -> 0 0 2 1 0
Values for row 3 -> 0 0 0 0 0
Values for row 4 -> 0 3 0 0 1
Enter the number of rows and columns-> 4 5
Enter the values of the sparse matrix A
Values for row 1 -> 1 0 1 0 0
Values for row 2 -> 0 0 0 0 2
Values for row 3 -> 0 0 3 0 0
Values for row 4 -> 1 0 0 1 0
The sparse array is
[0] 4  5  5
[1] 0  1  1
[2] 1  2  2
[3] 1  3  1
[4] 3  1  3
[5] 3  4  1
The sparse array is
[0] 4  5  6
[1] 0  0  1
[2] 0  2  1
[3] 1  4  2
[4] 2  2  3
[5] 3  0  1
[6] 3  3  1

Transpose of the first matrix is
The sparse array is
[0] 5  4  5
[1] 1  0  1
[2] 1  3  3
[3] 2  1  2
[4] 3  1  1
[5] 4  3  1

Transpose of the second matrix is
The sparse array is
[0] 5  4  6
[1] 0  0  1
[2] 0  3  1
[3] 2  0  1
[4] 2  2  3
[5] 3  3  1
[6] 4  1  2
Arrays can be added
Sum of both the matrices is
The sparse array is
[0] 4  5  9
[1] 0  0  1
[2] 0  2  2
[3] 1  4  4
[4] 1  3  1
[5] 2  2  3
[6] 3  1  3
[7] 3  0  1
[8] 3  3  1
[9] 3  4  1
```

Sample input 2:

```
3 2
0 0
0 1
-2 0
2 3
1 0 0
0 0 3
```

Sample output 2:

```
Enter the number of rows and columns-> 3 2
Enter the values of the sparse matrix A
Values for row 1 -> 0 0
Values for row 2 -> 0 1
Values for row 3 -> -2 0
Enter the number of rows and columns-> 2 3
Enter the values of the sparse matrix A
Values for row 1 -> 1 0 0
Values for row 2 -> 0 0 3
The sparse array is
[0] 3  2  2
[1] 1  1  1
[2] 2  0  -2
The sparse array is
[0] 2  3  2
[1] 0  0  1
[2] 1  2  3

Transpose of the first matrix is
The sparse array is
[0] 2  3  2
[1] 0  2  -2
[2] 1  1  1

Transpose of the second matrix is
The sparse array is
[0] 3  2  2
[1] 0  0  1
[2] 2  1  3
Matrices cant be added
```