Report 1

학 번	32202546
성 명	안지성
담당교수	황두성교수님
제 출 일	2021.9.26

1. 문제 기술

Problem 1

덧셈, 뺄셈, 곱셈, element-wise 나눗셈을 수행하는 함수를 설계하고 구현하시오. Element-wise 나눗셈은 동일한 행열 위치의 값과 나눗셈을 수행하는 연산입니다. 덧셈, 뺄셈, element-wise 나눗셈 연산은 동일한 크기의 행열에서만 사용합니다.

Problem 2

Problem 1의 테스트를 다양한 행열의 크기에 따라 실행하여 시간의 복잡도를 분석하시요. 테스트는 10개 이상입니다.

덧셈, 뺄셈, 나눗셈 입력이 m과 n이면 x-축은 m × n으로 함
곱셈 입력이 m, k과 k, n이면 x-축은 n × k × m으로 함

Problem 3

주어진 2차 행열의 희소 행열과 전치 행열을 구성하는 함수를 작성하고 테스트하시오.

Problem 4

Problem 3의 테스트를 Problem 2와 동일한 행열에 대해 실행하여 시간의 복잡도를 분석하시오.

Problem 5

Problem 2 & 4의 결과를 데이터 크기와 실행시간을 비교하고 설명하시오.

2. 분석과 설계 방법

우선 main.c에 main함수를 만들어서 전반적인 코드를 실행시키고 공통적인 특성을 가진 함수들을 묶어서 각각 새로운 소스코드인 .c파일로 만들고 이를 main함수에서 사용할 수 있게 각 소스코드 파일에 해당하는 헤더파일인 .h파일들을 만들어 각 함수들을 정의한다. main함수에서 행렬의 크기를 입력받아 문제 조건에 맞는 연산들을 만든 연산함수를 불러와 수행시키고, 희소행렬과 전치행렬로 변환하는 함수들을 불러와 각 행렬들을 변환시킨다. 이 때, 함수들을 사용하기 이전에는 start로, 사용한 이후에서는 end로 실행 시간을 측정하고 end-start로 실행하는데 걸린 시간을 측정하여 기록한다. 그리고 이 모든 결과들을 출력으로 보여서 확인하며 체크한다. 마지막으로 총 10번 이상의 테스트를 통해 각 테스트마다 걸린 수행 시간들을 그래프로 그려 비교해본다.

3. 프로그램 리스트(Make file 포함)

<<사용하는 파일 리스트>>

- makefile
- main.c
- arithmetic.h, arithmetic.c
- common.h, common.c
- matrix.h, matrix.c

- makefile

```
#
# makefile
#
CC=gcc -g
LIBS= -lstdc++

#
# all: hw
#
hw: main.o common.o arithmetic.o matrix.o
	gcc -o hw main.o common.o arithmetic.o matrix.o

#
main.o: main.c
	gcc -c main.c -o main.o

#
common.o: common.c common.h
	gcc -c common.c -o common.o

#
arithmetic.o: arithmetic.c arithmetic.h
	gcc -c arithmetic.c -o arithmetic.o

#
matrix.o: matrix.c matrix.h
	gcc -c matrix.c -o matrix.o

#
clean:
	rm hw.exe *.o
```

- main.c

```
#include <stdio.h>
#include "common.h"
#include "arithmetic.h"
#include "matrix.h"
#include <time.h>

// main함수 작성
int main(void){
    int i,j;
    int m, n, q;
    int r, c;
    int **a, **b, **res1, **res2, **res3, **x, **y, **res4;

    int cnt1=0, cnt2=0, cnt3=0, cnt4=0;
    clock_t start, end;

    //같은 크기의 행렬 행과 열 입력
    printf("i j: ");
    scanf("%d %d", &i, &j);

    //다른 크기의 행렬 행과 열 입력
    printf("m n q: ");
    scanf("%d %d %d", &m, &n, &q);

    //같은 크기의 행렬 - 덧셈, 뺄셈, 나눗셈
    a = array(i,j);
    b = array(i,j);
    res1 = array(i,j);
    res2 = array(i,j);
    res3 = array(i,j);

    for(r=0; r<i; r++){
        for(c=0; c<j; c++){
            a[r][c] = mrand(i);
            b[r][c] = mrand(j);
        }
    }
```

```
    //다른 크기의 행렬 - 곱셈

    x = array(m,n);
    y = array(n,q);
    res4 = array(m,q);

    for(r=0; r<m; r++){
        for(c=0; c<n; c++)
            x[r][c] = mrand(m);
    }

    for(r=0; r<n; r++){
        for(c=0; c<q; c++)
            y[r][c] = mrand(n);
    }

    printf("array: a\n");
    printArray(a, i, j);
    printf("array: b\n");
    printArray(b, i, j);
    printf("array: x\n");
    printArray(x, m, n);
    printf("array: y\n");
    printArray(y, n, q);

    //연산 수행
    //덧셈, 뺄셈, 나눗셈
    start = clock();
    addTwoArrays(res1, a, b, i, j);
    subtractTwoArrays(res2, a, b, i, j);
    ElementWiseDivideTwoArrays(res3, a, b, i, j);
    end = clock();

    printf("<<덧셈, 뺄셈, 나눗셈>>\nstart: %f\nend: %f\n 걸린시간: %f\n\n",
(double)start, (double)end, get_runtime(start, end));

    printf("array: +\n");
    printArray(res1, i, j);
    printf("array: -\n");
    printArray(res2, i, j);
```

```
    printf("array: /\n");
    printArray(res3, i, j);

    //곱셈
    start = clock();
    multiplyTwoArrays(res4, x, y, m, n, q);
    end = clock();

    printf("<<곱셈>>\nstart:   %lf\nend:   %lf\n   걸린시간:   %lf\n\n",
(double)start, (double)end, get_runtime(start, end));

    printf("array: *\n");
    printArray(res4, m, q);

    //같은 크기의 행렬 2개 희소 행렬 변환
    start = clock();
    MATRIX *A = getSparse(a, sizeof(a) / (sizeof(int) * j), i, &cnt1);
    MATRIX *B = getSparse(b, sizeof(b) / (sizeof(int) * j), i, &cnt2);
    end = clock();

    printf("<<같은 크기 행렬들-> 희소 행렬>>\nstart: %lf\nend: %lf\n 걸린시간:
%lf\n\n", (double)start, (double)end, get_runtime(start, end));

    printf("array: a -> sparse: A\n");
    printSparse(cnt1, A);
    printf("array: b -> sparse: B\n");
    printSparse(cnt2, B);

    //다른 크기의 행렬 2개 희소 행렬 변환
    start = clock();
    MATRIX *X = getSparse(x, sizeof(x) / (sizeof(int) * n), i, &cnt3);
    MATRIX *Y = getSparse(y, sizeof(y) / (sizeof(int) * n), i, &cnt4);
    end = clock();

    printf("<<다른 크기 행렬들 -> 희소 행렬>>\nstart: %lf\nend: %lf\n 걸린시
간: %lf\n\n", (double)start, (double)end, get_runtime(start, end));

    printf("array: x -> sparse: X\n");
    printSparse(cnt3, X);
    printf("array: y -> sparse: Y\n");
```

```
    printSparse(cnt4, Y);

    //같은 크기의 행렬-> 전치 행렬 변환
    start = clock();
    MATRIX *At = transpose(A, cnt1);
    MATRIX *Bt = transpose(B, cnt2);
    end = clock();

    printf("<<같은 크기 행렬들 -> 전치 행렬>>\nstart: %lf\nend: %lf\n 걸린시
간: %lf\n\n", (double)start, (double)end, get_runtime(start, end));

    printf("sparse: A -> sparse: At\n");
    printSparse(cnt1, At);
    printf("sparse: B -> sparse: Bt\n");
    printSparse(cnt2, Bt);

    //다른 크기의 행렬-> 전치 행렬 변환
    start = clock();
    MATRIX *Xt = transpose(X, cnt3);
    MATRIX *Yt = transpose(Y, cnt4);
    end = clock();

    printf("<<다른 크기 행렬들 -> 전치 행렬>>\nstart: %lf\nend: %lf\n 걸린시
간: %lf\n\n", (double)start, (double)end, get_runtime(start, end));

    printf("sparse: X -> sparse: Xt\n");
    printSparse(cnt3, Xt);
    printf("sparse: Y -> sparse: Yt\n");
    printSparse(cnt4, Yt);

    freeArray(a, i);
    freeArray(b, i);
    freeArray(res1, i);
    freeArray(res2, i);
    freeArray(res3, i);
    freeArray(x, m);
    freeArray(y, n);
    freeArray(res4, m);

    return 0;
```

```
}
```

<<arithmetic>>

- arithmetic.h

```
#ifndef _ARITHMETIC_

#define _ARITHMETIC_

void addTwoArrays(int **, int **, int **, int, int);
void subtractTwoArrays(int **, int **, int **, int, int);
void ElementWiseDivideTwoArrays(int **, int **, int **, int, int);
void multiplyTwoArrays( int **, int **, int **, int, int, int);

#endif
```

- arithmetic.c

```
#include <stdio.h>
#include "arithmetic.h"

/*
a + b = c

c[i][j] = a[i][j]+b[i][j]
*/

void addTwoArrays(int **res, int **a, int **b, int m, int n){
    int r, c;

    for(r=0; r<m; r++){
        for(c=0; c<n; c++){
            res[r][c] = a[r][c] + b[r][c];
        }
    }
}

/*
a - b = c

c[m][n] = a[m][n] - b[m][n]
*/
```

```
void subtractTwoArrays(int **res, int **a, int **b, int m, int n){
    int r, c;

    for(r=0; r<m; r++){
        for(c=0; c<n; c++){
            res[r][c] = a[r][c] - b[r][c];
        }
    }
}

/*
a / b = c

c[m][n] = a[m][n] / b[m][n]
*/

void ElementWiseDivideTwoArrays(int **res, int **a, int **b, int m, int
n){
    int r, c;

    for(r=0; r<m; r++){
        for(c=0; c<n; c++){
            // 값이 둘 중에 하나라도 0이 있다면 결과값 0 대입
            if(a[r][c] == 0 || b[r][c] == 0)
                res[r][c] = 0;
            // 값이 둘 다 0이 아니라면 나눈 값을 대입
            else
                res[r][c] = a[r][c] / b[r][c];
        }
    }
}

/*
a * b = c

c[i][j] = a[i][0]*b[0][j] + a[i][1]*b[1][j]+ ... +a[i][n-1]*b[n-1][j]
= sum(a[i][k]*b[k][j], k=0,...,n-1)
*/
```

```
void multiplyTwoArrays( int **res, int **a, int **b, int m, int n, int q){
    int r, c, k;
    int sum;

    for(r=0; r<m; r++){
        for(c=0; c<q; c++){
            sum = 0;
            for(k=0; k<n; k++)
                sum += a[r][k] + b[k][c];
            res[r][c] = sum;
        }
    }
}
```

<<common>>

- common.h

```
#ifndef _COMMON_

#define _COMMON_

int **array(int, int);
void freeArray(int **, int);
void freeVector(int *);
void printArray(int **, int, int);

int mrand(int);
double get_runtime(clock_t, clock_t);

#endif
```

- common.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "common.h"

int **array(int rows, int cols){
    int **arr;
    int i;
```

```
    arr = (int **) malloc(sizeof(int)*rows);
    for(i=0; i<rows; i++)
        arr[i] = (int *) malloc(sizeof(int)*cols);

    return arr;
}

void freeArray(int **arr, int rows){
    int r;
    for(r=0; r<rows; r++)
        freeVector(arr[r]);
    free(arr);
}

void freeVector( int *vec ){
    free( vec );
}

void printArray(int **arr, int rows, int cols){
    int r,c;

    for(r=0; r<rows; r++){
        for(c=0; c<cols; c++)
            printf("\t %d", arr[r][c]);
        printf("\n");
    }
    printf("\n");
}

int mrand(int range){
    return rand() % range;
}

double get_runtime(clock_t start, clock_t end)
{
  return (double)(end-start)/(double)CLOCKS_PER_SEC;
}
```

<<matrix>>

- matrix.h

```
#ifndef _MATRIX_

#define _MATRIX_

typedef struct sparse{
    int col;
    int row;
    int value; //0이 아닌 데이터의 수
}MATRIX;

void printSparse(int, MATRIX *);
MATRIX *getSparse(int **, int, int, int *);
MATRIX *transpose(MATRIX *, int);

#endif
```

- matrix.c

```
#include <stdio.h>
#include <stdlib.h>
#include "matrix.h"

void printSparse(int cnt, MATRIX *S){
    for(int i=0; i<cnt; i++){
        printf("%d\t%d\t%d\n", S[i].col, S[i].row, S[i].value);
    }
    printf("\n");
}

//Sparse Matrix
MATRIX *getSparse(int **a, int col, int row, int *cnt){
    int i,j;
    MATRIX *S;
    for(i=0; i<col; i++){
        for(j=0; j<row; j++){
            if(a[i][j])
                (*cnt)++; //0이 아닌 데이터 개수 구하기
        }
    }

    //초기값 때문에 공간이 1개 더 필요하다
```

```
    S = (MATRIX *)malloc(sizeof(MATRIX)*((*cnt)+1));

    //index=0 시작값 행, 열, 데이터 크기 넣기
    S[0].col = col;
    S[0].row = row;
    S[0].value = *cnt;

    //index=1 MATRIX 각 원소에 값 할당하기
    *cnt = 1;
    for(i=0; i<col; i++){
        for(j=0; j<row; j++){
            if(a[i][j]){
                S[*cnt].col = i;
                S[*cnt].row = j;
                S[(*cnt)++].value = a[i][j];
            }
        }
    }
    return S;
}

//Transpose Matrix
MATRIX *transpose(MATRIX *a, int cnt){
    int n, i, j, currentb;
    MATRIX *b = (MATRIX*)malloc(sizeof(MATRIX)*cnt);

    //우선 첫 줄을 바꾸기
    n = a[0].value;
    b[0].row = a[0].col;
    b[0].col = a[0].row;
    b[0].value = n;

    //행을 기준으로
    if(n>0) {
        currentb = 1;
        for(i=0; i<a[0].col; i++){
            for(j=1; j<=n; j++){
                if(a[j].col==i){
                    b[currentb].row = a[j].col;
                    b[currentb].col = a[j].row;
```

```
                    b[currentb].value = a[j].value;
                    currentb++;
                }
            }
        }
    }
    return b;
}
```

4. 결과

makefile 실행 -> make command 입력

```
$ make
gcc -c main.c -o main.o
In file included from main.c:2:
common.h:11:1: warning: parameter names (without types) in function declaration
   11 | double get_runtime(clock_t, clock_t);
      | ^~~~~~
gcc -c common.c -o common.o
gcc -c arithmetic.c -o arithmetic.o
gcc -c matrix.c -o matrix.o
gcc -o hw main.o common.o arithmetic.o matrix.o

user@LAPTOP-SIENDSTS /c/HW1
$ ls
arithmetic.c  arithmetic.o  common.c  common.o  main.c  makefile  matrix.h
arithmetic.h  backup        common.h  hw.exe    main.o  matrix.c  matrix.o
```

hw.exe 실행 (ex -> i = 5, j = 5 / m = 6, n = 7, q = 8)

```
C:\HW1>hw.exe
i j: 5 5
m n q: 6 7 8
array: a
        1       4       4       3       2
        0       1       1       0       2
        1       2       2       1       3
        2       1       4       2       0
        3       2       3       1       3

array: b
        2       0       4       3       4
        0       2       1       2       1
        4       3       2       1       0
        1       3       2       4       4
        1       3       4       1       3

array: x
        5       2       4       3       3       1       5
        3       1       4       4       5       2       0
        0       4       4       2       4       4       2
        3       2       3       4       2       0       3
        3       2       3       5       0       4       0
        2       4       2       5       4       0       3

array: y
        0       4       0       6       4       3       4       0
        3       0       5       6       3       4       5       3
        5       0       2       3       2       4       5       5
        2       5       1       6       4       4       5       2
        4       1       2       6       6       5       1       2
        1       3       1       0       6       0       3       4
        1       2       5       2       0       5       2       0
```

<덧셈, 뺄셈, 나눗셈>

```
<<덧셈, 뺄셈, 나눗셈>>
start: 5515.000000
end: 5515.000000
 걸린시간: 0.000000

array: +
        3       4       8       6       6
        0       3       2       2       3
        5       5       4       2       3
        3       4       6       6       4
        4       5       7       2       6

array: -
        -1      4       0       0       -2
        0       -1      0       -2      1
        -3      -1      0       0       3
        1       -2      2       -2      -4
        2       -1      -1      0       0

array: /
        0       0       1       1       0
        0       0       1       0       2
        0       0       1       1       0
        2       0       2       0       0
        3       0       0       1       1
```

<곱셈>

```
<<곱셈>>
start: 5549.000000
end: 5549.000000
 걸린시간: 0.000000

array: *
        39      38      39      52      48      48      48      39
        35      34      35      48      44      44      44      35
        36      35      36      49      45      45      45      36
        33      32      33      46      42      42      42      33
        33      32      33      46      42      42      42      33
        36      35      36      49      45      45      45      36
```

<희소행렬 전환> -> 구현을 잘못했는지 row값과 value값이 0으로 나온다..

```
<<같은 크기 행렬들-> 희소 행렬>>
start: 5567.000000
end: 5567.000000
 걸린시간: 0.000000

array: a -> sparse: A
0       5       0

array: b -> sparse: B
0       5       0

<<다른 크기 행렬들 -> 희소 행렬>>
start: 5578.000000
end: 5578.000000
 걸린시간: 0.000000

array: x -> sparse: X
0       5       0

array: y -> sparse: Y
0       5       0
```

<전치행렬 전환> -> 희소행렬 구현이 잘못되어서 전치행렬에도 영향을 그대로 받은 것 같다.
그래도 전치행렬 변환은 행과 열이 바뀐 것이 보인다.

```
<<같은 크기 행렬들 -> 전치 행렬>>
start: 5585.000000
end: 5585.000000
 걸린시간: 0.000000

sparse: A -> sparse: At
5       0       0

sparse: B -> sparse: Bt
5       0       0

<<다른 크기 행렬들 -> 전치 행렬>>
start: 5603.000000
end: 5603.000000
 걸린시간: 0.000000

sparse: X -> sparse: Xt
5       0       0

sparse: Y -> sparse: Yt
5       0       0
```

'시간복잡도'를 체크하고 싶었으나 실행이 너무 빨라서인지 잘 모를 이유로 계속 start값과 end값이 거의 똑같이 나와 실행시간이 0이 나온다. 이 부분은 구현의 오류인지 잘 모르겠으나 많이 아쉽다.
따라서 그래프는 직접 시간복잡도를 계산하여 구해본다.

<<시간복잡도>>

<연산>

덧셈 -> addTwoArrays() -> O(m*n) -> O(n^2)

뺄셈 -> subtractTwoArrays() -> O(m*n) -> O(n^2)

Element-wise 나눗셈 -> ElementWiseDivideTwoArrays() -> O(m*n) -> O(n^2)

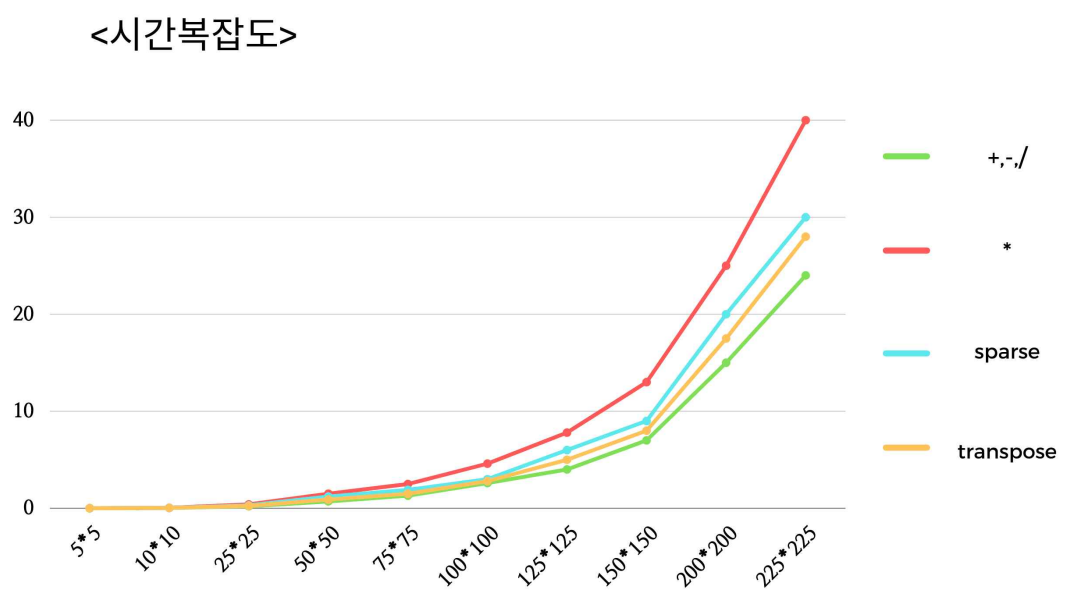
곱셈 -> multiplyTwoArrays() -> O(m*n*q) -> O(n^3)

<행렬>

희소행렬 -> getSparse() -> O(row*col) -> O(n^2)

전치행렬 -> transpose() -> O(a[0].col*n) -> O(n^2)

<<그래프>>



<<결론>>

데이터의 크기가 커질수록, 시간복잡도에 직접적으로 영향을 끼치던 for문의 수행시간이 늘어나고, 따라서 실행 시간이 점점 늘어난다. 즉, 데이터 크기↑ -> 실행 시간↑

5. 토의와 개선점

• 테스트 결과에 대한 비교(ex: 계산의 제한성, 올바른 계산 결과의 여부 등)

-> 우선 희소행렬로 변환하는 함수를 완벽하게 구현하지 못한 거 같아서 테스트 시 잘 확인이 되지 않았다. 시간측정에서도 잘 측정이 되지 않아 정확한 값으로 결과를 도출할 수 없어서 직접 시간복잡도를 구하여 어림잡아 결과를 만들었다. 이후 더 코드를 고쳐서 정확한 결과를 얻을 수 있게끔 해야겠다.

• 데이터 크기에 따른 측정된 실행시간 비교

-> 데이터 크기가 커질수록 같은 함수를 쓰더라도 더 실행시간이 늘어남을 알 수 있다. 비록 정확한 데이터 값을 얻지 못하여 어림잡아 계산하였지만 결론은 데이터 크기가 늘어날수록 실행시간이 더 커진다.

스스로 과제 평가

모듈라 기법 사용	Make 사용	테스트	결과	분석, 토의&개선점
O	O	△	△	O