1. 문제 해결 및 논의

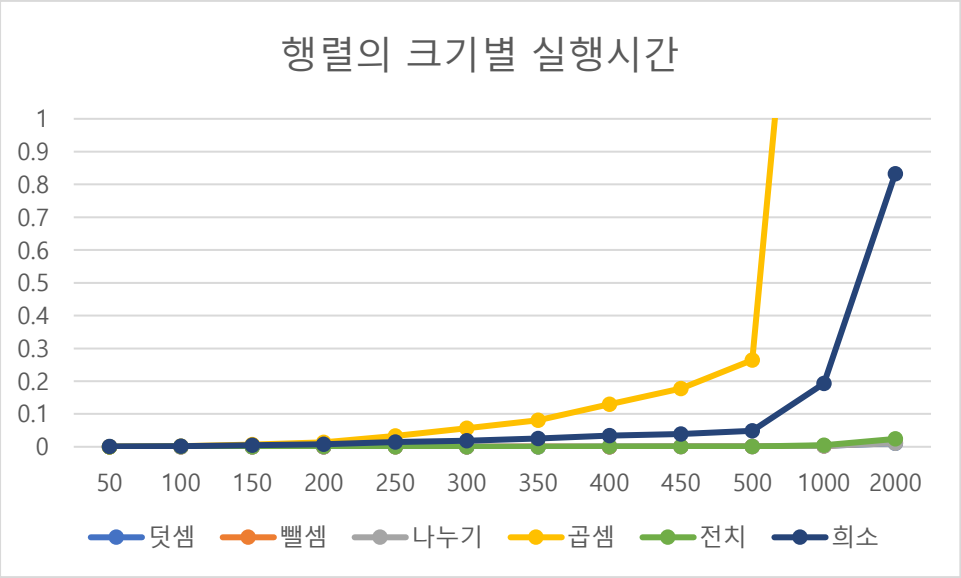
먼저 사용자가 편하게 행렬의 사칙연산을 하는 함수를 호출하고 사용할 수 있게 하기 위해서 행렬의 사칙연산을 구현하는 함수 및 배열의 동적생성 및 반환, 배열 출력 등의 기능을 하는 함수들을 모아 하나의 calculate_array.c 파일로 만들고 행렬의 크기를 입력 받고 원하는 기능을 호출할 수 있는 main.c 파일을 구분하여 작성했다. 추후에 problem4를 해결하기 위해 생성한 행렬의 전치,희소 행렬을 구하는 함수도 calculate_array.c에 추가했다.

각 함수의 시간 복잡도를 Big-O 표기법에 따라 표현해 보았더니

덧셈,뺄셈,나눗셈,전치,희소 함수는 O(mn)이였고, 곱셈의 경우, O(mnk)였다.

또한 행렬의 크기에 따른 실행시간을 측정하기 위해서 n x n 정방행렬을 만들고 n의 크기를

50,100,150,200,250,300,350,400,450,500,1000,2000으로 설정하여 실행시간의 변화를 측정해 보았다.



일단 어지간한 크기의 행렬로는 실행시간을 잘 관찰할 수 없었다. 컴퓨터의 성능이 좋아진 덕분이라고 생각했다. 행렬의 크기를 250x250(250x250x250)쯤에서야 실행시간의 변화가 눈에 보였다. 더 확연한 변화를 보기 위해 크기를 1000,2000 으로 늘려 보았는데 2000이상 부터는 곱셈의 계산이 더뎌 진행이 힘들었다. 하지만 전체적으로 봤을 때 덧셈,뺄셈,나누기,전치 함수는 데이터가 매우 커져도 크게 변화가 없었다. 곱셈의 경우 2000일 때는 40초 가까이 나왔고 희소행렬도 배열의 크기를 1000단위씩 증가시키니 변화가 크게 눈에 띄었다. 곱셈의 경우 시간의 복잡도도 O(mnk)가 나왔기 때문에 이와 같은 그래프를 예상할 수 있었고 희소 행렬의 경우 예상하지 못한 그림인데, 내가 구현한 희소행렬을 만드는 함수에서 함수 내부에 희소행렬 결과를 저장할 행렬을

생성하는 코드가 있어 시간이 더뎌진 것이 아닐까 싶었다. 또한 희소 행렬을 만드는 함수의 경우 연산을 진행하는 것이 아니라 0이 아닌 값을 새로운 배열에 넣어주는 작업을 하다보니 실행시간이 긴 것이 아닌가 싶다.

2. 프로그램 리스트

calculate_array.c - 행렬 사칙연산 함수, 전치/희소 행렬 변환 함수, 배열의 동적 할당 및 반환 함수, 결과 출력 함수 작성한 코드 파일

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

int** array(int rows, int cols) {  //2차원 자료구조 동적생성
        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 cols) {
        int r, c;
        for (r = 0; r < rows; r++)
                free(arr[r]);

        free(arr);
}


void printArray(int** arr, int rows, int cols, const char msg[]) {
        int r, c;

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

void multiplyArrays(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;
                }
}

void plusArrays(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];
                }
        }

}

void minusArrays(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];
                }
        }
}

void divisionArrays(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];
                }
        }
}

void transpoArrays(int** a, int m, int n) {
        int i, j;
        int** res;

        res = array(n, m); //결과 저장하는 배열 생성

        for (i = 0; i < n; i++) {
                for (j = 0; j < m; j++) {
                        res[i][j] = a[j][i];
                }
        }
        printArray(res, n, m, "trans"); //전치행렬 구성 결과 출력
}

void sparseArrays(int** a, int rows, int cols) {
        int** res;
```

```
        int count=0; //value(값)의 개수를 세는 변수

        res = array(101, 3); //결과 저장하는 배열 생성
        res[0][0] = rows;
        res[0][1] = cols;

        for (int i = 0; i < rows;i++) {
                for (int j = 0; j < cols; j++) {
                        if (a[i][j] != 0) { //값이 0이 아닌 경우
                                res[count + 1][0] = i; //행 저장
                                res[count + 1][1] = j; //열 저장
                                res[count + 1][2] = a[i][j]; //값 저장
                                count++;
                        }
                }
        }
        res[0][2] = count;
        printArray(res,count+1,3, "sparse");  //희소행렬 구성 결과 출력
}
```

calculate_array.h - calculate_array.c에 사용되는 함수들의 프로토타입을 적어 둔 헤더파일

```
int** array(int, int);
void freeArray(int**, int, int);
void printArray(int** arr, int rows, int cols, const char msg[]);

void multiplyArrays(int** res, int** a, int** b, int m, int n, int q);
void plusArrays(int** res, int** a, int** b, int m, int n);
void minusArrays(int** res, int** a, int** b, int m, int n);
void divisionArrays(int** res, int** a, int** b, int m, int n);
void transpoArrays(int** a, int m, int n);
void sparseArrays(int** a, int rows, int cols);
```

mian.c – 매개변수를 입력 받고 함수를 호출하는 코드를 작성한 파일

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

int main(void) {
        int m, n, k, num, g, h;
        int** a, ** b, ** c, ** d, ** res1, ** res2;
        //double start, end;


        printf("(1):덧셈 (2):뺄셈 (3):나눗셈 (4):곱셈₩n");
        scanf_s("%d", &num);

        printf("₩n배열의 크기 3개(m,n,k)를 입력하세요.₩n m x n 배열 2개, n x k배열 1개가
만들어집니다.₩n");
        scanf_s("%d%d%d", &m, &n, &k); //배열의 크기를 입력받아 int형 변수 3개(m/n/k)에 저장


        a = array(m, n); //배열 생성(나눗셈의 경우 나누는 숫자가 0이면 안 됨)
```

```
b = array(m, n);
res1 = array(m, n);

c = array(n, k);
res2 = array(m, k);

int i, j;

for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
                a[i][j] = (rand() % 10);
                b[i][j] = (rand() % 10) + 1;
        }
}

for (i = 0; i < n; i++) {
        for (j = 0; j < k; j++) {
                c[i][j] = (rand() % 5);
        }
}

//배열 출력
printArray(a, m, n, "a");
printArray(b, m, n, "b");
printArray(c, n, k, "c");


switch(num) {
case 1:
        //start = clock();
        plusArrays(res1, a, b, m, n);
        //end = clock();
        printArray(res1, m, n, "plus");
        break;

case 2:
        minusArrays(res1, a, b, m, n);
        printArray(res1, m, n, "minus");
        break;

case 3:
        divisionArrays(res1, a, b, m, n);
        printArray(res1, m, n, "division");
        break;

case 4:
        multiplyArrays(res2, a, c, m, n, k);
        printArray(res2, m, k, "multiply");
        break;


}

/*전치,희소 행렬 구성 위한 배열을 따로 생성했습니다*/
int g, h,i,j;
```

```
	int** d;

	printf("₩n전치행열, 희소행열을 만듭니다.₩n배열의 크기 2개(g,h)를 입력하세요.₩n");
	scanf_s("%d%d", &g,&h);

	d = array(g, h);

	for (i = 0; i < g; i++) {
		for (j = 0; j < h; j++) {
			d[i][j] = (rand() % 5);
		}
	}

	printArray(d, g, h, "d");

	//start = clock();
	transpoArrays(d, g, h); //전치행렬 구성
	//end = clock();
	//printf("₩n%f", (double)((end - start) / CLOCKS_PER_SEC));

	//start = clock();
	sparseArrays(d, g, h); //희소행렬 구성
	//end = clock();
	//printf("₩n%f", (double)((end - start) / CLOCKS_PER_SEC));



	return 0;
}
```

Makefile – .c 파일을 .o 파일로 컴파일하고 Array_func 수행 파일을 만듬

```
Array_func: calculate_array.o main.o
	gcc -o Array_func calculate_array.o main.o


calculate_array.o: calculate_array.c calculate_array.h
	gcc -c calculate_array.c -o calculate_array.o


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


clean:
```

```
rm -f Array_func *.o
```