**자료구조 과제 – Sparse Matrix 프로그래밍 프로젝트**

정보대학 컴퓨터학과 2017320108

고재영

목차

1. 구현환경
2. 프로그램 소개 및 주의사항
3. 프로그램 사용방법, 설명 (콘솔 스크린샷 첨부)
4. 소스코드
5. **구현환경**

OS: Window 10

Language: C language

Tool: Dev-C++

1. **프로그램 소개 및 주의사항**

본 프로그램은 희소행렬에 관한 작업을 하는 용도입니다. 콘솔 창에서 메뉴를 선택하는 방식입니다. 메뉴에서 선택할 수 있는 작업은 총 9가지로 구성되어 있으며, 이는 크게 행렬 생성, 행렬 삭제, 행렬 덧셈, 행렬 곱셈, 행렬 전치, 프로그램 종료로 나눠볼 수 있습니다.

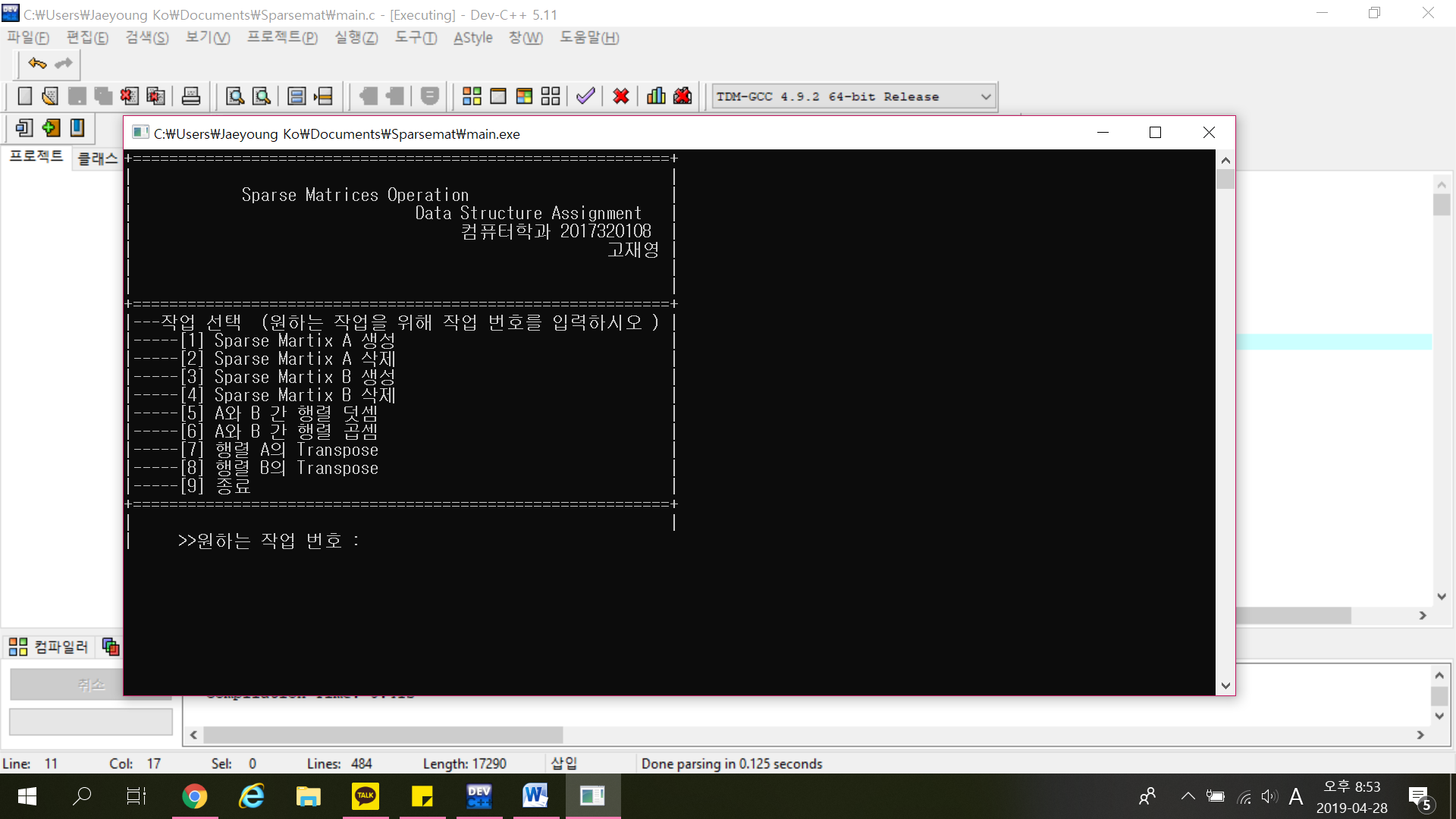
작업에 있어서, 능동적인 작업으로 볼 수 있는 행렬 생성, 행렬 삭제, 행렬 덧셈, 행렬 곱셈, 행렬 전치 이후에 곧바로 해당 결과를 mwrite로 출력합니다. 따라서 mwrite는 독립적으로 작업을 지시할 수 없다는 점 설명드립니다.

본 프로그램을 설명하기에 앞서 주의사항을 알려드립니다. 본 프로그램은 엄밀하고 세심한 예외 처리를 포함하지 않기에, 입력에 있어서 사용자는 콘솔에서 안내하는 바로 반드시 형식에 맞추어야 합니다. 또한 사용하는 데에 있어서 상식적인 순서에 반드시 맞추어야 합니다. (행렬을 생성한 이후에 전치를 해야한다던지, 두 행렬을 모두 생성한 이후 덧셈과 곱셈을 한다던지)

그리고 덧셈, 곱셈, 전치 작업에 관하여 이야기를 하자면, 작업 이전 저장된 A, B의 값은 변하지 않습니다. 예를 들면, 전치 작업으로 A행렬이 전치되면, 전치된 결과를 바로 표현할 뿐 A에 전치된 결과를 덮어씌우지 않습니다.

1. **프로그램 설명, 사용법**

본 프로그램에서 mwrite, mread, merase는 전적으로 수업에 기반한 표현에 근거하고 있으며, madd, mmult, mtranspose는 mread가 함수를 생성하는 데에 있어 코드를 응용하여 구현했습니다.

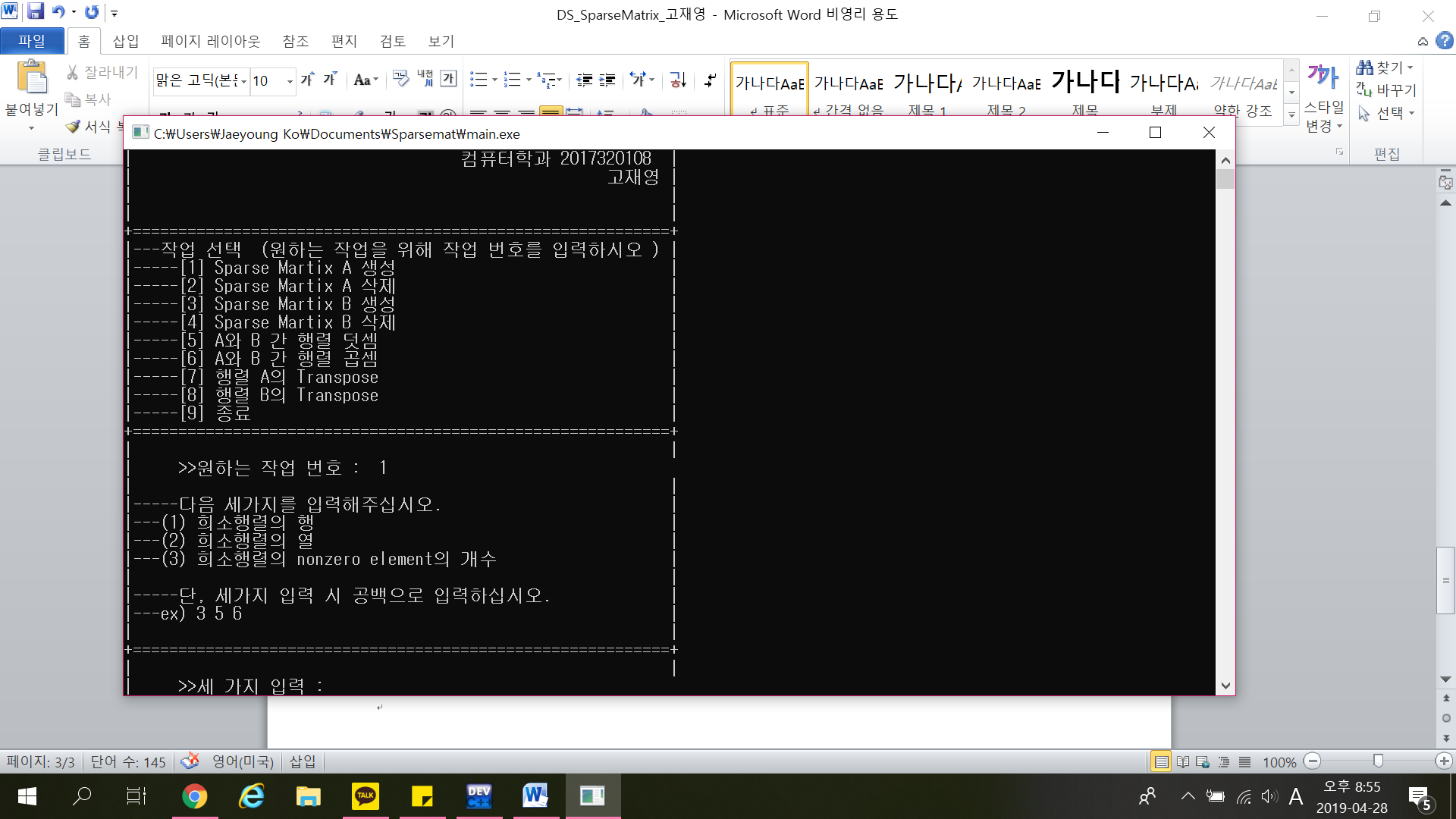


최초 입력창입니다. 작업 번호를 입력함으로 원하는 작업을 실행시킬 수 있습니다.

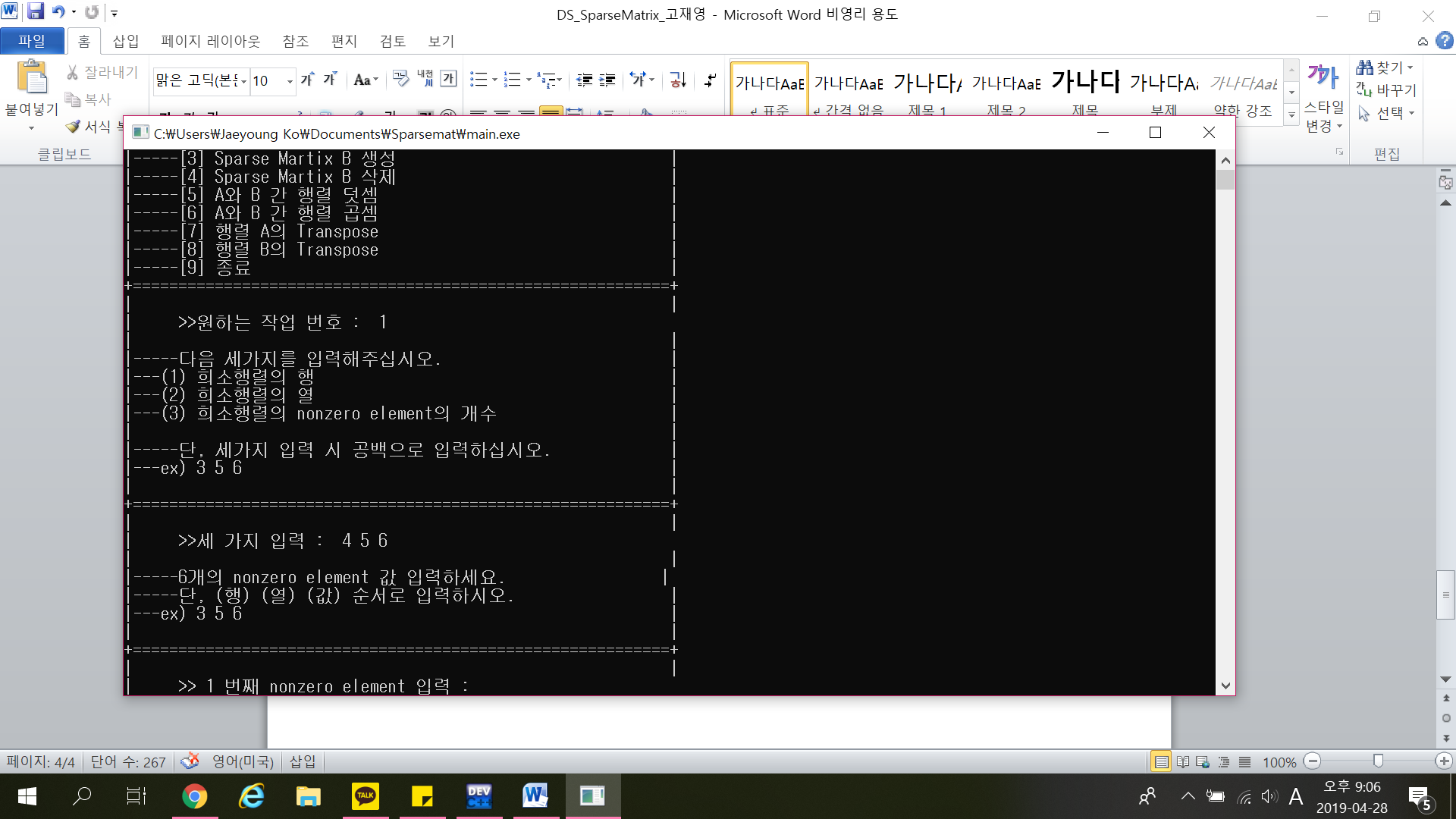
프로그램을 설명하기 위해 저희가 입력할 것을 정리해보겠습니다.

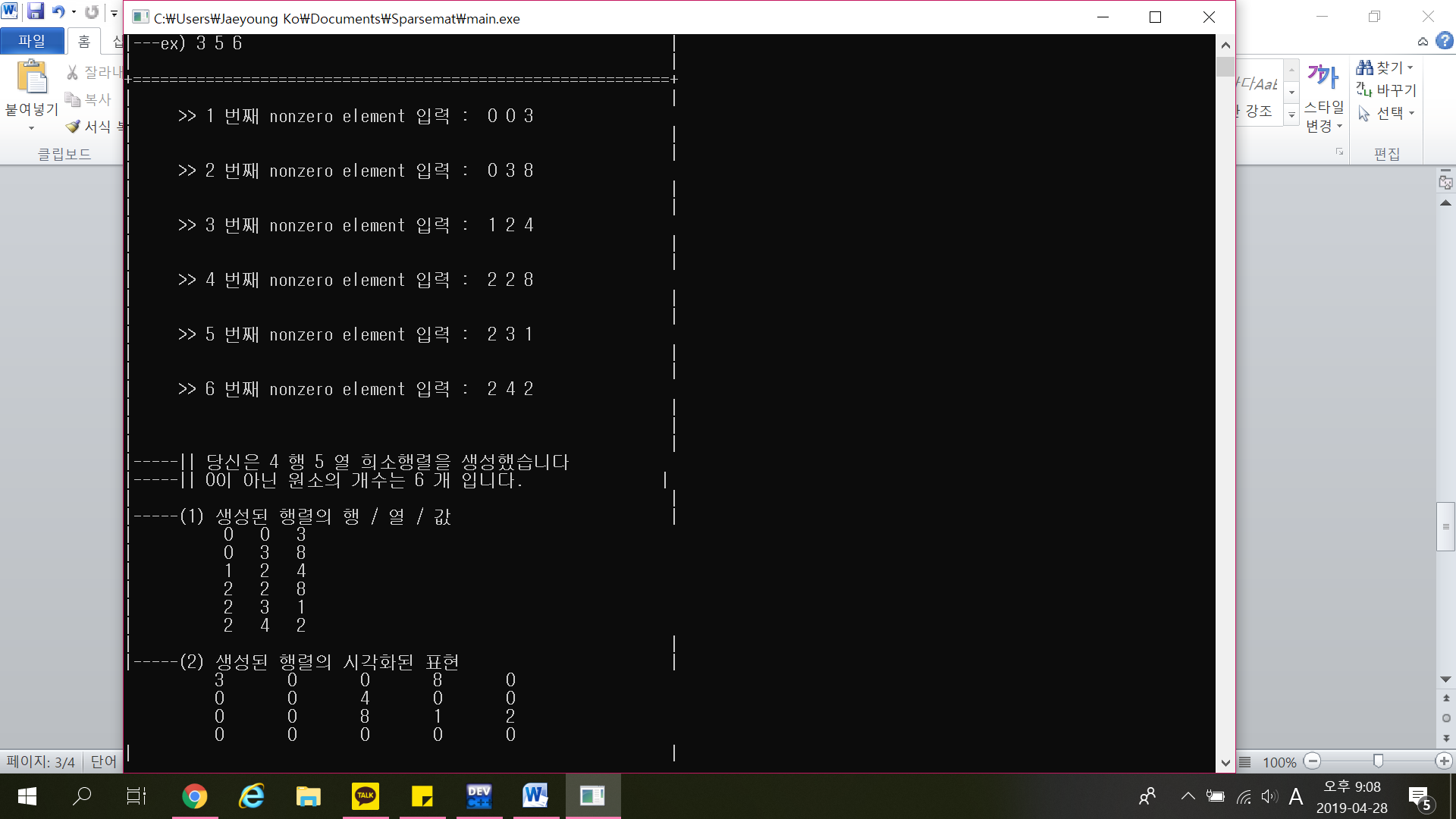
|  |  |  |  |
| --- | --- | --- | --- |
| 설명 | 행렬 A 생성 | 행렬 B 생성 | 행렬 B 삭제 이후 재 생성 |
| 행렬 정보 | **4 5 6** | **4 5 4** | **5 3 5** |
| 인자 | 0 0 3 | 0 0 -3 | 0 1 6 |
|  | 0 3 8 | 0 1 5 | 1 1 8 |
|  | 1 2 4 | 0 3 7 | 1 2 3 |
|  | 2 2 8 | 1 4 1 | 3 0 5 |
|  | 2 3 1 |  | 4 1 1 |
|  | 2 4 2 |  |  |

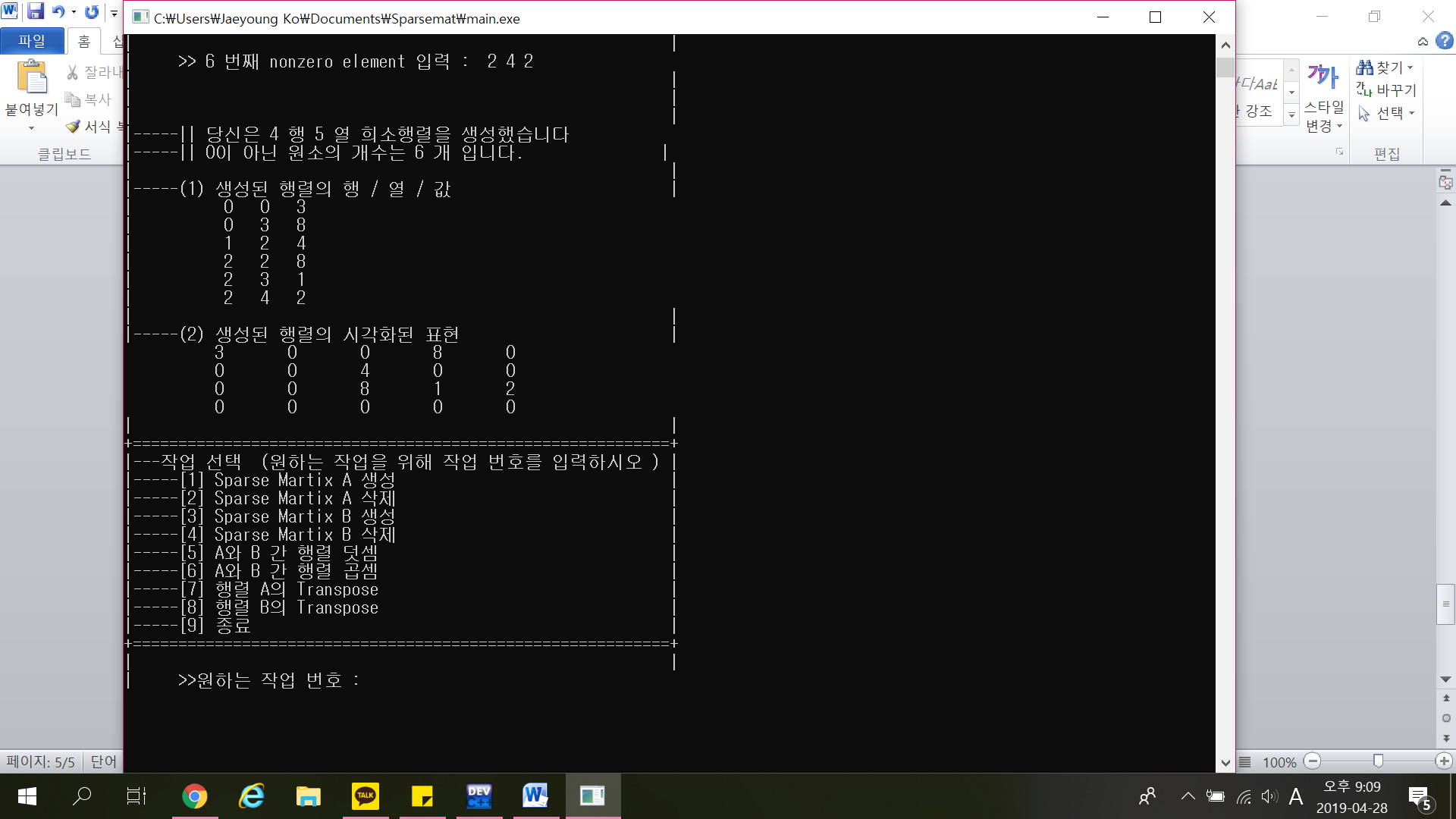
[>>행렬 A 생성을 위해 [1]을 입력합니다.]



1을 입력하면 희소행렬 A를 생성하게 됩니다.

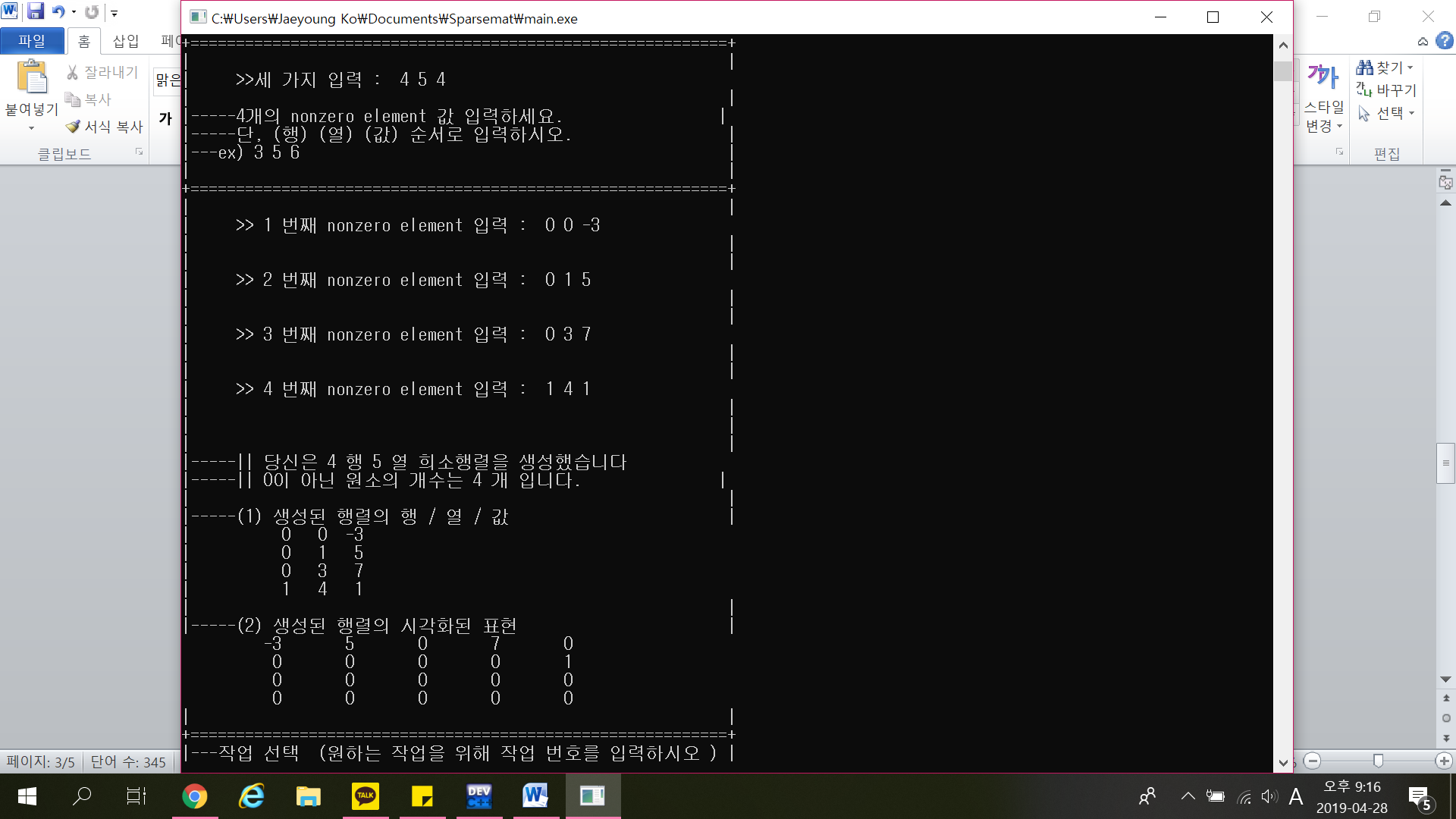




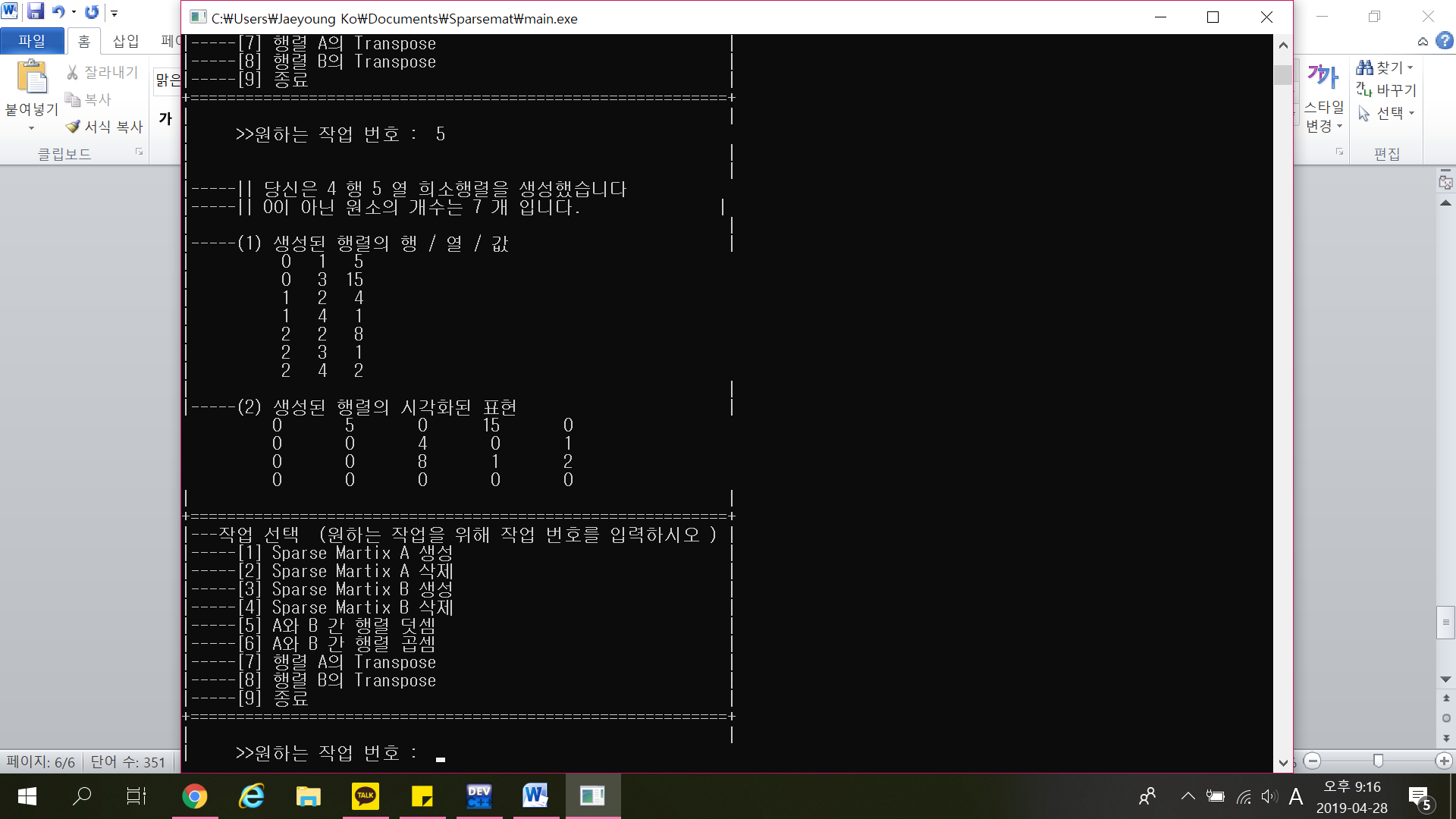


A를 성공적으로 생성한 이후에 곧바로 listed representation으로 행렬을 출력하고, 또 우리가 익숙한 행렬의 시각화된 표현으로도 출력합니다. 결과를 출력하고 메뉴는 다시 작업을 선택할 수 있도록 합니다.

[ >> 행렬 B를 생성]



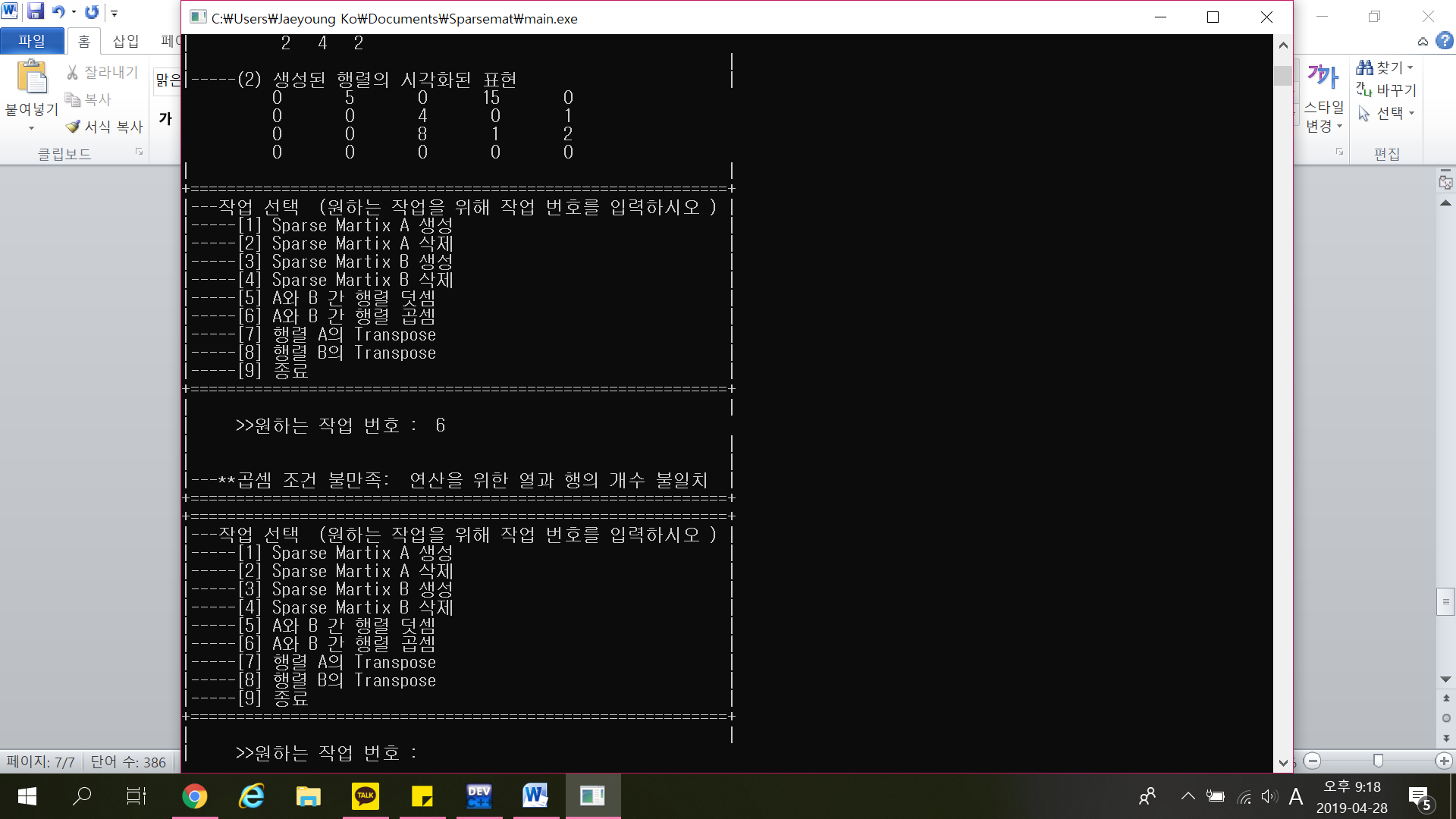
[ >> 행렬 A와 B의 덧셈]



행렬A와 B는 둘다 4X5 행렬로 크기가 같기 때문에 덧셈조건에 만족하여 올바르게 실행됩니다.

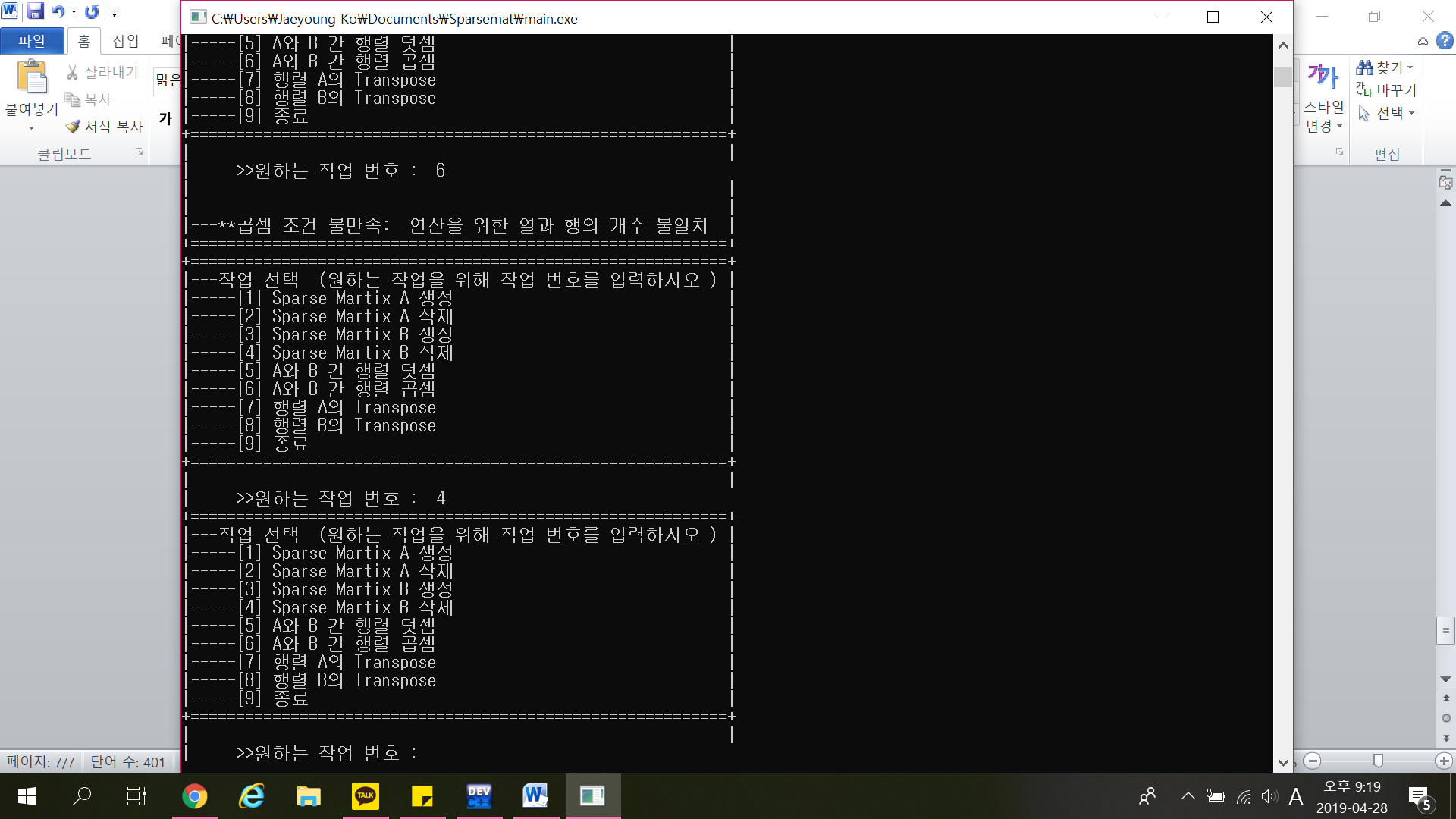
특이한 점을 설명드리자면, 위치 (0, 0)에서 덧셈이 이루어져 3 + (-3) = 0이 되어, 값이 0이 된 노드는 삭제되었습니다.

[>>행렬 A와 B 곱셈]



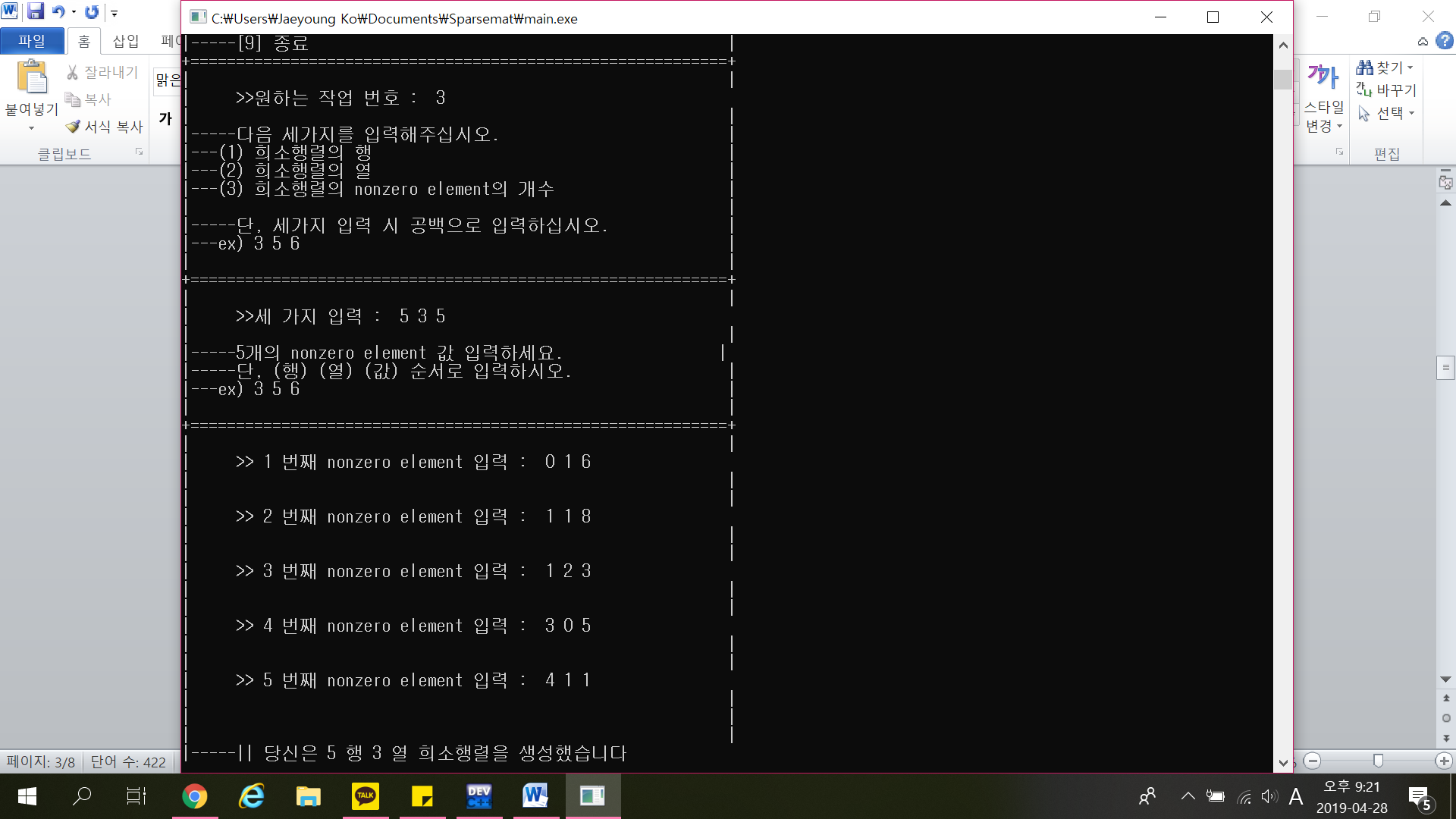
둘다 4X5행렬이기 때문에 열과 행의 개수가 일치하지 않아 곱셈작업이 이루어지지 않는 모습입니다.

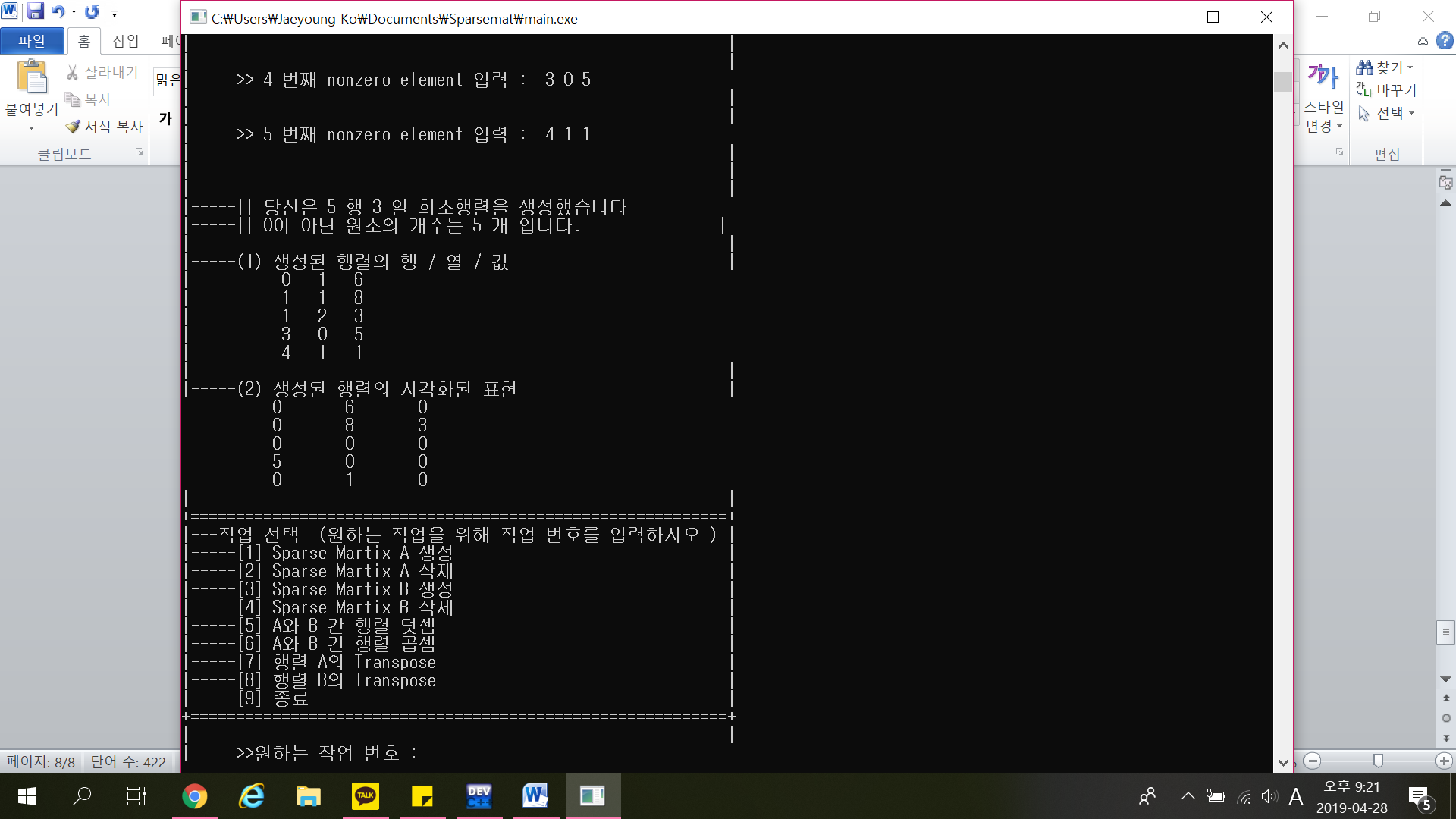
[>>행렬 B 삭제]



행렬 B에 저장된 것을 삭제하고 free 시키는 것입니다. 프로그램 상으로는 알 수 없지만 수업시간에 배운 내용을 기반으로 구현했습니다.

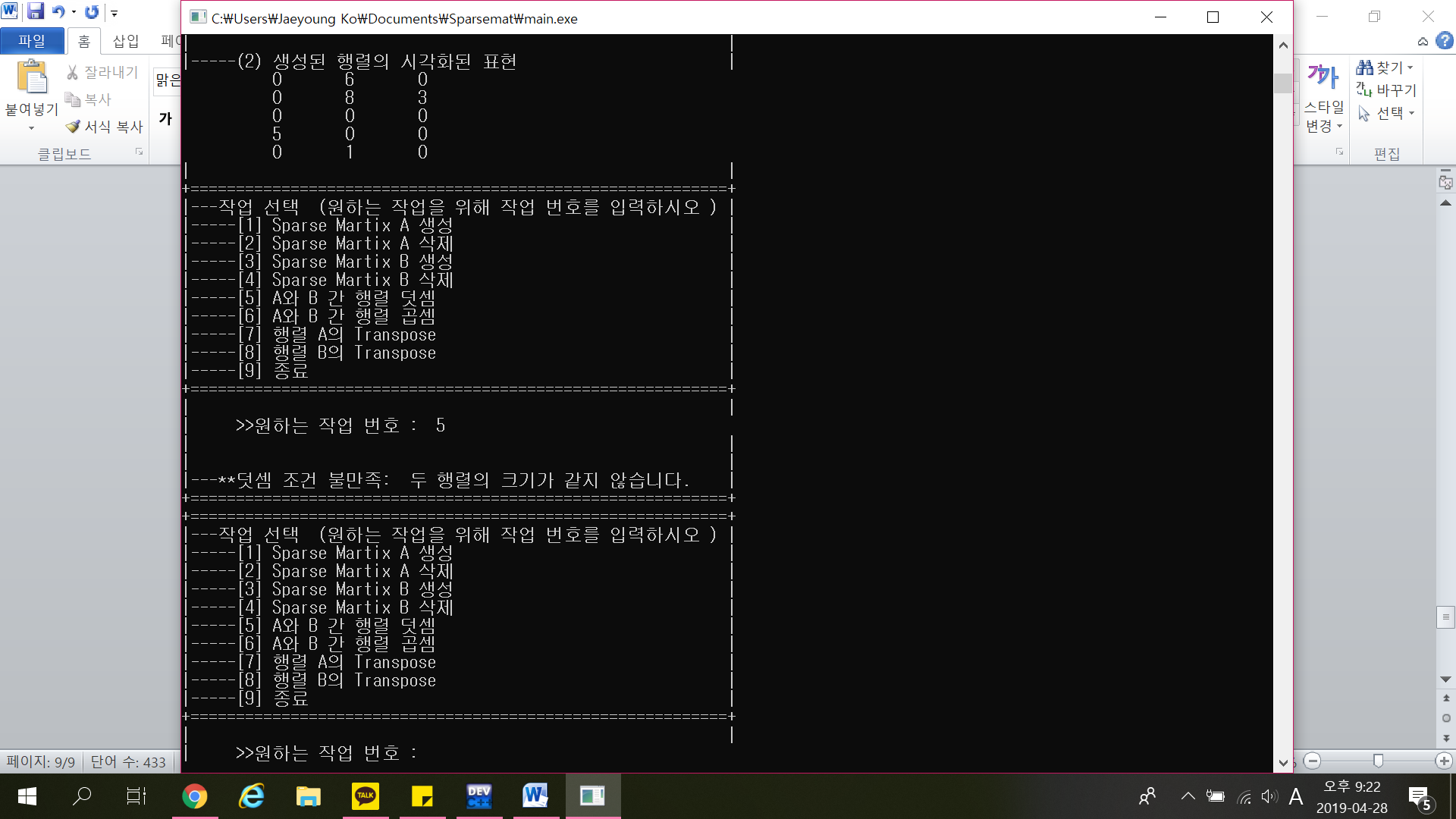
[>>행렬 B 재생성]





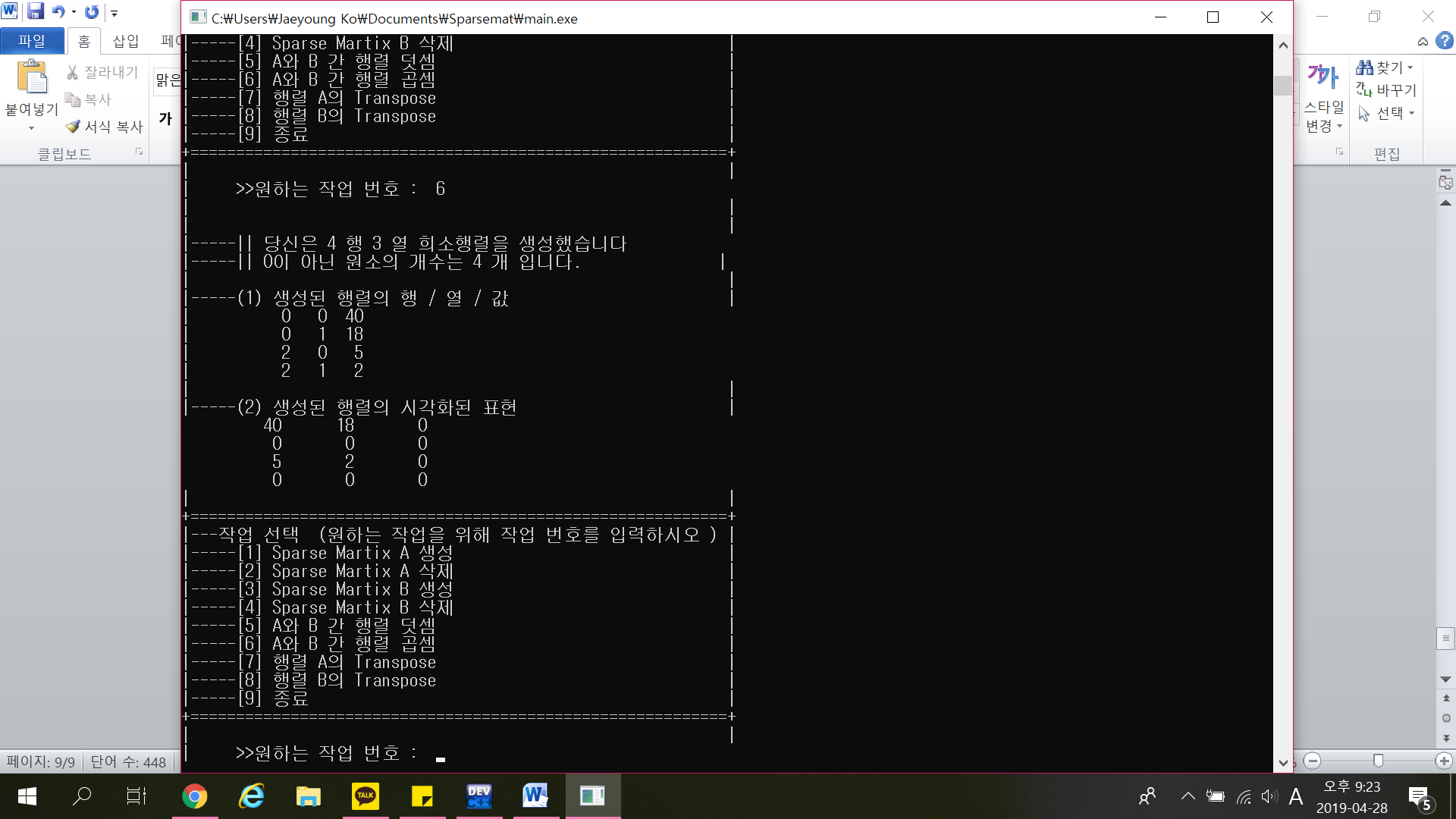
이미 설명드린 작업이니 별다른 설명 첨가하지 않겠습니다.

[>>행렬 A와 B 덧셈]



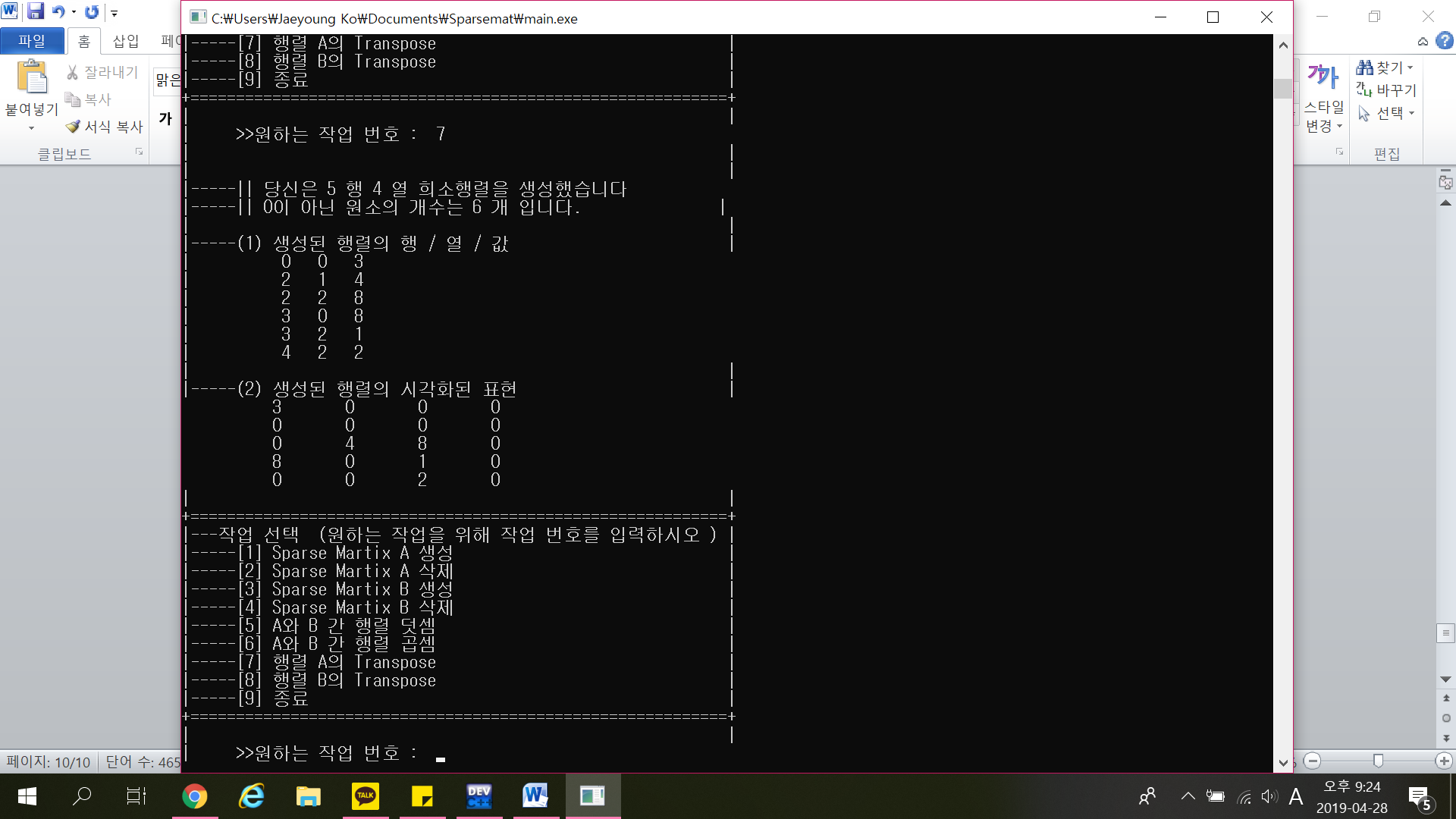
B행렬이 5X3행렬로 재생성되어 4X5행렬인 A행렬과 크기가 달라 덧셈 조건 불일치로 실패하는 모습입니다.

[>>행렬A와 B 곱셈]



A,B 행렬의 열과 행이 5로 일치하기 때문에 올바르게 곱셈이 진행되어 결과물로 4X3 행렬을 만들어낸 모습입니다.

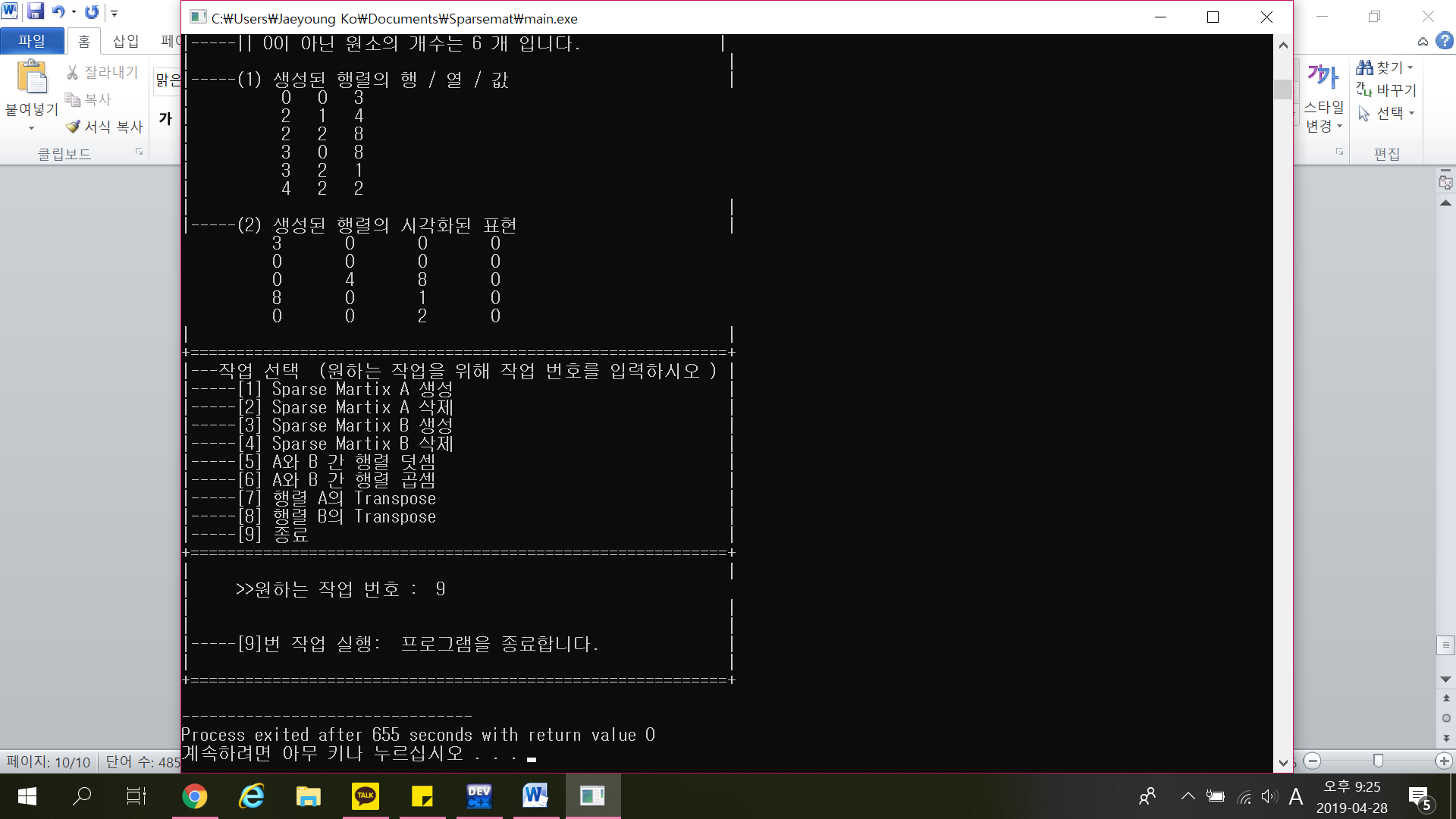
[>>행렬A Transpose]



4X5행렬인 A를 전치시킨 5X4행렬 At의 결과를 출력합니다.

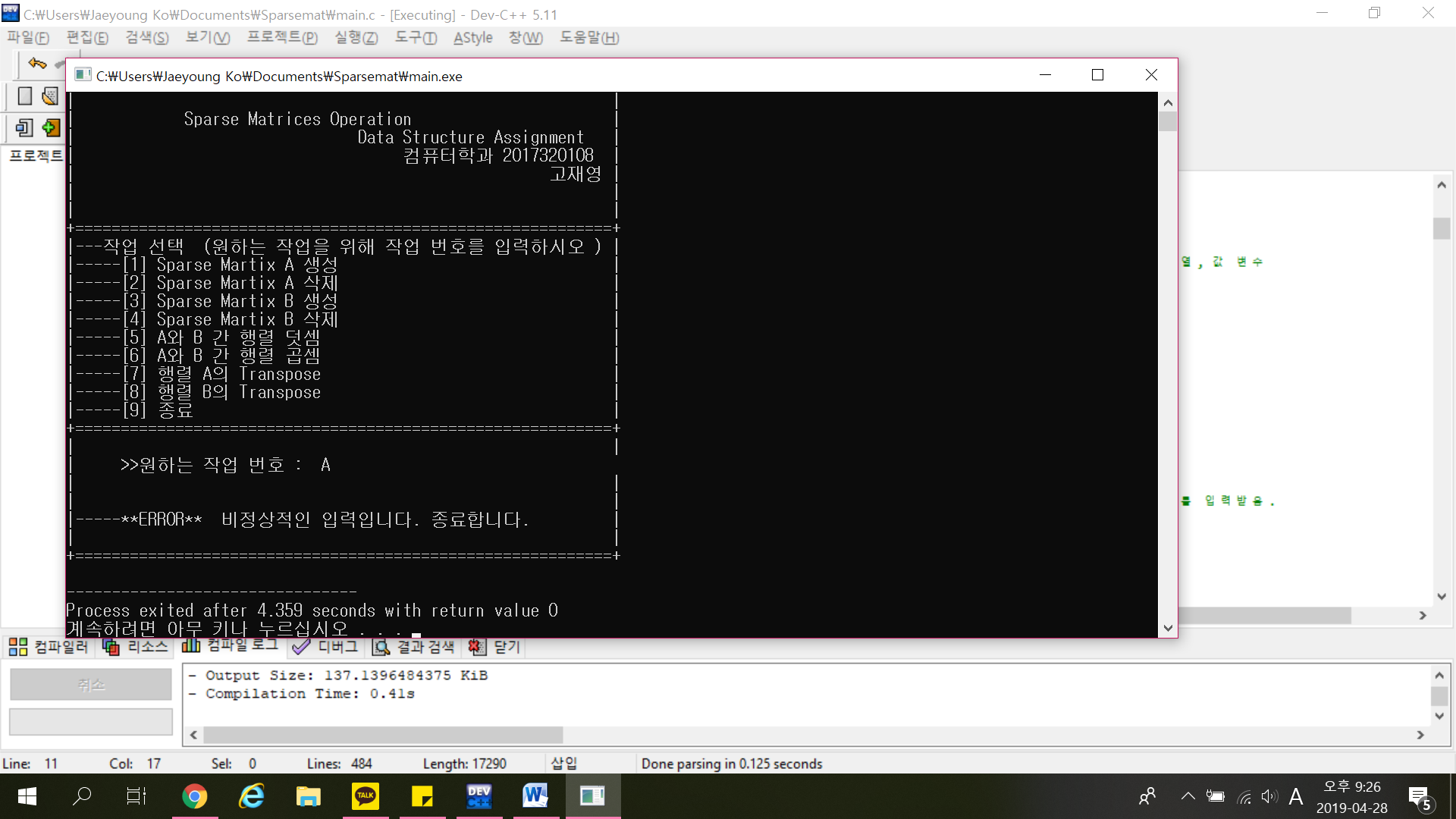
B의 전치도 마찬가지로 작업 선택을 통해 볼 수 있습니다.

[>>종료를 위해 9 입력]



9를 입력하면 프로그램을 종료시킵니다.

추가적으로, 예외 처리를 엄밀하게 하지 않았으나 작업 선택에 있어서는 적시된 내용 이외의 입력 시 오류 발생으로 프로그램을 종료시킵니다.



1. **소스코드**

#include <stdio.h>

#include <stdlib.h>

//이 코드는 Sparse Matrix 과제를 위한 코드입니다. 2017320108 고재영.

#define MAX\_SIZE 50

typedef enum {head, entry} tagfield;

typedef struct matrixNode \*matrixPointer;

typedef struct entryNode{

int row;

int col;

int value;

}entryNode;

typedef struct matrixNode {

matrixPointer down;

matrixPointer right;

tagfield tag;

union{

matrixPointer next;

entryNode entry;

} u;

}matrixNode;

typedef struct { //수업에 기반한 구조 토대로 새로운 구조체 생성

matrixPointer hdnode[MAX\_SIZE];

matrixPointer node;

}SparseMatrix;

SparseMatrix \*first, \*second, \*result; //글로벌로 생성

void mread(SparseMatrix \*sm){ //행렬을 생성하는데에 사용하는 함수

int numRows, numCols, numTerms, numHeads, i; //희소행렬 전체 행, 열, 논제로 엘리먼트 수

int row, col, value, currentRow; //입력받을 논제로 엘리먼트의 값을 받기 위한 행, 열, 값 변수

matrixPointer temp, last;

printf("| |\n");

printf("|-----다음 세가지를 입력해주십시오. |\n");

printf("|---(1) 희소행렬의 행 |\n");

printf("|---(2) 희소행렬의 열 |\n");

printf("|---(3) 희소행렬의 nonzero element의 개수 |\n");

printf("| |\n");

printf("|-----단, 세가지 입력 시 공백으로 입력하십시오. |\n");

printf("|---ex) 3 5 6 |\n");

printf("| |\n");

printf("+===========================================================+\n");

printf("| |\n");

printf("| >>세 가지 입력 : ");

scanf("%d %d %d", &numRows, &numCols, &numTerms); //희소행렬 자체 큰 틀의 정보를 입력받음.

numHeads = (numCols > numRows) ? numCols: numRows;

sm->node = malloc(sizeof(struct matrixNode));

sm->node->tag = entry;

sm->node->u.entry.row = numRows;

sm->node->u.entry.col = numCols;

sm->node->u.entry.value = numTerms;/////////

if(!numHeads) //밸류가 하나도 없어서 자기 자신 가리킴

sm->node->right = sm->node;

else {

//헤더 노드 초기화

for (i = 0; i < numHeads; i++){

temp = malloc(sizeof(struct matrixNode));

sm->hdnode[i] = temp;

sm->hdnode[i]->tag = head;

sm->hdnode[i]->right = temp;

sm->hdnode[i]->u.next = temp;

}

currentRow = 0;

last = sm->hdnode[0];

printf("| |\n");

printf("|-----%d개의 nonzero element 값 입력하세요. |\n", numTerms);

printf("|-----단, (행) (열) (값) 순서로 입력하시오. |\n");

printf("|---ex) 3 5 6 |\n");

printf("| |\n");

printf("+===========================================================+\n");

for(i = 0; i < numTerms; i++){

printf("| |\n");

printf("| >> %d 번째 nonzero element 입력 : ", i+1);

scanf("%d %d %d", &row, &col, &value);

printf("| |\n");

if (row > currentRow) { //다음 행 넘어가기

last->right = sm->hdnode[currentRow];

currentRow = row;

last = sm->hdnode[row];

}

temp = malloc(sizeof(struct matrixNode));

temp->tag = entry;

temp->u.entry.row = row;

temp->u.entry.col = col;

temp->u.entry.value = value;

last->right = temp;

last = temp;

sm->hdnode[col]->u.next->down = temp;

sm->hdnode[col]->u.next = temp;

}

last->right = sm->hdnode[currentRow];

for (i = 0; i < numCols; i++)

sm->hdnode[i]->u.next->down = sm->hdnode[i];

for( i = 0; i < numHeads - 1; i++)

sm->hdnode[i]->u.next = sm->hdnode[i+1];

sm->hdnode[numHeads-1]->u.next = sm->node;

sm->node->right = sm->hdnode[0];

}

}

void mwrite(SparseMatrix \* sm){ //행렬을 출력합니다.

int i, row, col;

matrixPointer temp;

matrixPointer head = sm->node->right;

printf("| |\n");

printf("| |\n");

printf("|-----|| 당신은 %d 행 %d 열 희소행렬을 생성했습니다\n", sm->node->u.entry.row, sm->node->u.entry.col);

printf("|-----|| 0이 아닌 원소의 개수는 %d 개 입니다. |\n", sm->node->u.entry.value);

printf("| |\n");

printf("|-----(1) 생성된 행렬의 행 / 열 / 값 |\n");

for( i = 0; i < sm->node->u.entry.row; i++){

for(temp = head->right; temp != head; temp = temp->right)

printf("| %4d%4d%4d\n", temp->u.entry.row, temp->u.entry.col, temp->u.entry.value);

head = head->u.next;

}

printf("| |\n");

printf("|-----(2) 생성된 행렬의 시각화된 표현 |\n");

for(row = 0; row < sm->node->u.entry.row; row++){

for(col = 0; col < sm->node->u.entry.col; col++){

printf("\t%3d ", readElement(sm, row, col));

}

printf("\n");

}

printf("| |\n");

}

void merase(SparseMatrix \* sm){

matrixPointer x, y;

matrixPointer head = sm->node->right;

int i, numHeads;

for (i = 0; i < sm->node->u.entry.row; i++){

y = head->right;

while (y != head){

x = y;

y = y->right;

free(x);

}

x = head;

head = head->u.next;

free(x);

}

y = head;

while (y != sm->node) {

x = y;

y = y->u.next;

free(x);

}

free(sm->node);

sm->node = NULL;

}

// readElement함수는 연산하는 데에 유용하게 사용할 함수입니다.

// sm이라는 행렬에서 행과 열의 좌표로부터 값을 반환하는 함수입니다.

int readElement(SparseMatrix \*sm, int row, int col){

matrixPointer temp;

matrixPointer head = sm->hdnode[row];

for(temp = head->right; temp != head; temp = temp->right){ //값이 존재한다면 반환

if(temp->u.entry.col == col)

return temp->u.entry.value;

}

return 0;//노드에 값이 존재하지 않는다는 것은 0이기 때문에 0을 반환

}

//희소행렬 d는 연산의 결과로 생성된 함수, 희소행렬 a,b는 피연산될 두 희소행렬입니다

void madd(SparseMatrix \*sm, SparseMatrix \*a, SparseMatrix \*b){

int numRows = a->node->u.entry.row, numCols = a->node->u.entry.col, numTerms, numHeads, i; //희소행렬 전체 행, 열, 논제로 엘리먼트 수

int row, col, value, currentRow; //입력받을 논제로 엘리먼트의 값을 받기 위한 행, 열, 값 변수

matrixPointer temp, last;

numHeads = (numCols > numRows) ? numCols: numRows;

sm->node = malloc(sizeof(struct matrixNode));

sm->node->tag = entry;

sm->node->u.entry.row = numRows;

sm->node->u.entry.col = numCols;

sm->node->u.entry.value = 0; //원소 개수는 0으로 초기화하고 노드 추가할 때마다 올려줍니다

if(!numHeads) //밸류가 하나도 없어서 자기 자신 가리킴

sm->node->right = sm->node;

else {

//헤더 노드 초기화

for (i = 0; i < numHeads; i++){

temp = malloc(sizeof(struct matrixNode));

sm->hdnode[i] = temp;

sm->hdnode[i]->tag = head;

sm->hdnode[i]->right = temp;

sm->hdnode[i]->u.next = temp;

}

currentRow = 0;

last = sm->hdnode[0];

for(row = 0; row < numRows; row++){

for(col = 0; col < numCols; col++){

value = readElement(a, row, col) + readElement(b, row, col);

if(value == 0) //덧셈 결과가 0이면 노드를 생성할 필요없음

continue;

//덧셈 결과 토대로 노드 생성

sm->node->u.entry.value++; //0이 아닌 값이 나오면 덧셈 결과 행렬의 원소개수 늘어나는 거니까 올려줌

if (row > currentRow) { //

last->right = sm->hdnode[currentRow];

currentRow = row;

last = sm->hdnode[row];

}

temp = malloc(sizeof(struct matrixNode));

temp->tag = entry;

temp->u.entry.row = row;

temp->u.entry.col = col;

temp->u.entry.value = value;

last->right = temp;

last = temp;

sm->hdnode[col]->u.next->down = temp;

sm->hdnode[col]->u.next = temp;

}

}

//행 종료 후 닫아줌

last->right = sm->hdnode[currentRow];

for (i = 0; i < numCols; i++)

sm->hdnode[i]->u.next->down = sm->hdnode[i];

for( i = 0; i < numHeads - 1; i++)

sm->hdnode[i]->u.next = sm->hdnode[i+1];

sm->hdnode[numHeads-1]->u.next = sm->node;

sm->node->right = sm->hdnode[0];

}

}

void mmult(SparseMatrix \*sm, SparseMatrix \*a, SparseMatrix \*b){

int numRows = a->node->u.entry.row, numCols = b->node->u.entry.col, numTerms, numHeads, i; //희소행렬 전체 행, 열, 논제로 엘리먼트 수

int row, col, value, currentRow; //입력받을 논제로 엘리먼트의 값을 받기 위한 행, 열, 값 변수

int k;

matrixPointer temp, last;

numHeads = (numCols > numRows) ? numCols: numRows;

sm->node = malloc(sizeof(struct matrixNode));

sm->node->tag = entry;

sm->node->u.entry.row = numRows;

sm->node->u.entry.col = numCols;

sm->node->u.entry.value = 0;

if(!numHeads) //밸류가 하나도 없어서 자기 자신 가리킴

sm->node->right = sm->node;

else {

//헤더 노드 초기화

for (i = 0; i < numHeads; i++){

temp = malloc(sizeof(struct matrixNode));

sm->hdnode[i] = temp;

sm->hdnode[i]->tag = head;

sm->hdnode[i]->right = temp;

sm->hdnode[i]->u.next = temp;

}

currentRow = 0;

last = sm->hdnode[0];

for(row = 0; row < numRows; row++){ //첫 번째 행렬의 row순서로

for(col = 0; col < numCols; col++){ //두 번째 행렬의 col 순서로

value = 0;

for(k = 0; k < a->node->u.entry.col; k++){

value += readElement(a, row, k) \* readElement(b, k, col);

}

if(value == 0) //덧셈 결과가 0이면 노드를 생성할 필요없음

continue;

//곱셈 결과 토대로 노드 생성

sm->node->u.entry.value++;

if (row > currentRow) { //

last->right = sm->hdnode[currentRow];

currentRow = row;

last = sm->hdnode[row];

}

temp = malloc(sizeof(struct matrixNode));

temp->tag = entry;

temp->u.entry.row = row;

temp->u.entry.col = col;

temp->u.entry.value = value;

last->right = temp;

last = temp;

sm->hdnode[col]->u.next->down = temp;

sm->hdnode[col]->u.next = temp;

}

}

//행 종료 후 닫아줌

last->right = sm->hdnode[currentRow];

for (i = 0; i < numCols; i++)

sm->hdnode[i]->u.next->down = sm->hdnode[i];

for( i = 0; i < numHeads - 1; i++)

sm->hdnode[i]->u.next = sm->hdnode[i+1];

sm->hdnode[numHeads-1]->u.next = sm->node;

sm->node->right = sm->hdnode[0];

}

}

void Transpose(SparseMatrix \*sm, SparseMatrix \*a){

int numRows = a->node->u.entry.col, numCols = a->node->u.entry.row, numTerms, numHeads, i; //희소행렬 전체 행, 열, 논제로 엘리먼트 수

int row, col, value, currentRow; //입력받을 논제로 엘리먼트의 값을 받기 위한 행, 열, 값 변수

matrixPointer temp, last;

numHeads = (numCols > numRows) ? numCols: numRows;

sm->node = malloc(sizeof(struct matrixNode));

sm->node->tag = entry;

sm->node->u.entry.row = numRows;

sm->node->u.entry.col = numCols;

sm->node->u.entry.value = a->node->u.entry.value; //전치를 해도 원소의 개수는 변하지 않음

if(!numHeads) //밸류가 하나도 없어서 자기 자신 가리킴

sm->node->right = sm->node;

else {

//헤더 노드 초기화

for (i = 0; i < numHeads; i++){

temp = malloc(sizeof(struct matrixNode));

sm->hdnode[i] = temp;

sm->hdnode[i]->tag = head;

sm->hdnode[i]->right = temp;

sm->hdnode[i]->u.next = temp;

}

currentRow = 0;

last = sm->hdnode[0];

for(row = 0; row < numRows; row++){

for(col = 0; col < numCols; col++){

value = readElement(a, col, row);

if(value == 0) //덧셈 결과가 0이면 노드를 생성할 필요없음

continue;

//덧셈 결과 토대로 노드 생성

if (row > currentRow) { //

last->right = sm->hdnode[currentRow];

currentRow = row;

last = sm->hdnode[row];

}

temp = malloc(sizeof(struct matrixNode));

temp->tag = entry;

temp->u.entry.row = row;

temp->u.entry.col = col;

temp->u.entry.value = value;

last->right = temp;

last = temp;

sm->hdnode[col]->u.next->down = temp;

sm->hdnode[col]->u.next = temp;

}

}

//행 종료 후 닫아줌

last->right = sm->hdnode[currentRow];

for (i = 0; i < numCols; i++)

sm->hdnode[i]->u.next->down = sm->hdnode[i];

for( i = 0; i < numHeads - 1; i++)

sm->hdnode[i]->u.next = sm->hdnode[i+1];

sm->hdnode[numHeads-1]->u.next = sm->node;

sm->node->right = sm->hdnode[0];

}

}

int main(void) {

int a, b, c; //a, b, c는 메뉴 상에서 사용자가 입력을 처리하기 위한 변수입니다.

first = (SparseMatrix\*) malloc(sizeof(SparseMatrix));

second = (SparseMatrix\*) malloc(sizeof(SparseMatrix));

result = (SparseMatrix \*) malloc(sizeof(SparseMatrix));

printf("+===========================================================+\n");

printf("| |\n");

printf("| Sparse Matrices Operation |\n");

printf("| Data Structure Assignment |\n");

printf("| 컴퓨터학과 2017320108 |\n");

printf("| 고재영 |\n");

printf("| |\n");

printf("| |\n");

//초기 시작 메뉴 창이 뜹니다.

do{

printf("+===========================================================+\n");

printf("|---작업 선택 (원하는 작업을 위해 작업 번호를 입력하시오 ) |\n");

printf("|-----[1] Sparse Martix A 생성 |\n");

printf("|-----[2] Sparse Martix A 삭제 |\n");

printf("|-----[3] Sparse Martix B 생성 |\n");

printf("|-----[4] Sparse Martix B 삭제 |\n");

printf("|-----[5] A와 B 간 행렬 덧셈 |\n");

printf("|-----[6] A와 B 간 행렬 곱셈 |\n");

printf("|-----[7] 행렬 A의 Transpose |\n");

printf("|-----[8] 행렬 B의 Transpose |\n");

printf("|-----[9] 종료 |\n");

printf("+===========================================================+\n");

printf("| |\n");

printf("| >>원하는 작업 번호 : ");

scanf("%d", &a); //작업 번호에 따라 분기됩니다.

if(a == 1){

mread(first);

mwrite(first);

} else if (a == 2){

merase(first);

} else if (a == 3){

mread(second);

mwrite(second);

} else if (a == 4){

merase(second);

} else if (a == 5){ //덧셈

if((first->node->u.entry.col == second->node->u.entry.col) &&

(first->node->u.entry.row == second->node->u.entry.row)){ //덧셈 조건:

madd(result, first, second);

mwrite(result);

}else {

printf("| |\n");

printf("| |\n");

printf("|---\*\*덧셈 조건 불만족: 두 행렬의 크기가 같지 않습니다. |\n");

printf("+===========================================================+\n");

}

} else if (a == 6){ //곱셈

if(first->node->u.entry.col == second->node->u.entry.row){ //곱셈 조건: 열과 행의 개수가 일치해야합니다

mmult(result, first, second);

mwrite(result);

}else {

printf("| |\n");

printf("| |\n");

printf("|---\*\*곱셈 조건 불만족: 연산을 위한 열과 행의 개수 불일치 |\n");

printf("+===========================================================+\n");

}

} else if (a == 7){ //A 전치

Transpose(result, first);

mwrite(result);

} else if (a == 8){ //B 전치

Transpose(result, second);

mwrite(result);

} else if (a == 9){ // 종료

printf("| |\n");

printf("| |\n");

printf("|-----[9]번 작업 실행: 프로그램을 종료합니다. |\n");

printf("| |\n");

printf("+===========================================================+\n");

break;

} else {

printf("| |\n");

printf("| |\n");

printf("|-----\*\*ERROR\*\* 비정상적인 입력입니다. 종료합니다. |\n");

printf("| |\n");

printf("+===========================================================+\n");

break;

}

} while(1);

return 0;

}