[실습2]

배열의 응용 : 희소행렬

The Index is like this.

INDEX

1. 희소행렬의 표현

- #01.version, #02.version

2. 희소행렬 덧셈 프로그램

- 업로드 된 프로그램 내 함수 설명

희소행렬의 표현

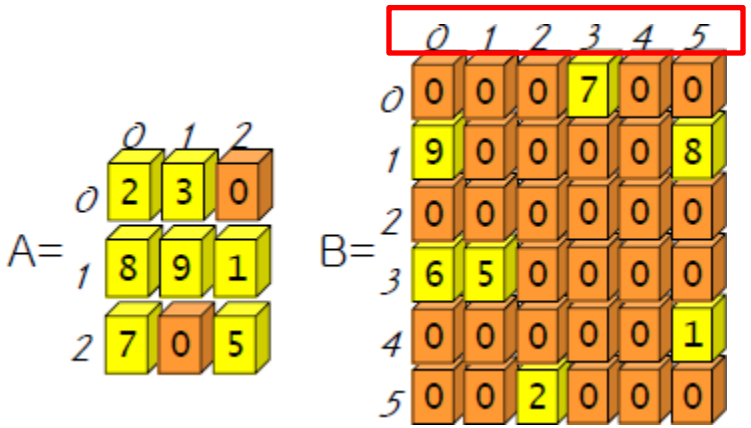
희소행렬의 표현 #01

- **희소행렬** : 대부분의 항들이 0인 2차원 행렬을 의미

- 희소행렬을 나타내기 위한 자료구조 #01 - **2차원 배열**
  : 2차원 배열로 저장하여, 배열의 index에 맞추어 각 항목을 저장한다.
  (행렬의 연산들을 간단하게 구현할 수 있으나, 대부분의 항이 0인 희소행렬의 경우 많은 메모리 공간들이 낭비된다.)

배열의 index = 행렬 위치

$$A = \begin{bmatrix} 2 & 3 & 0 \\ 8 & 9 & 1 \\ 7 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 & 7 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

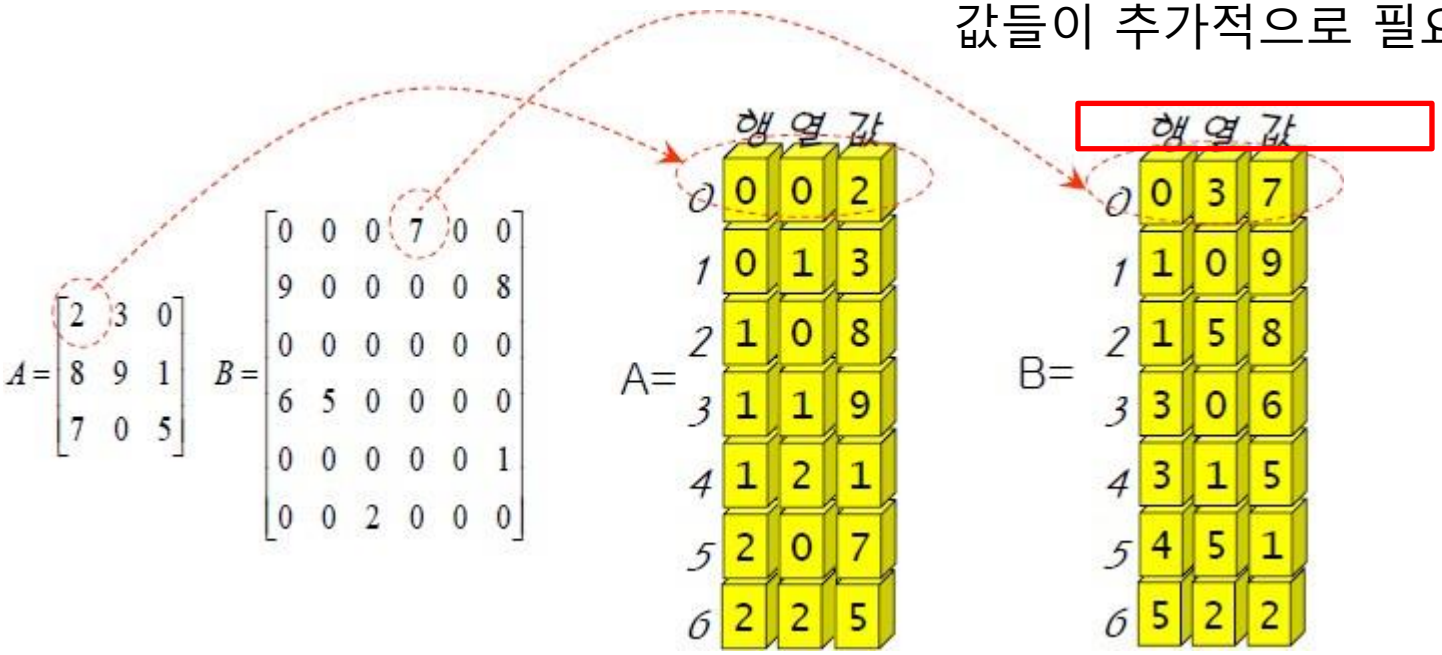


희소행렬의 표현 #02

- **희소행렬** : 대부분의 항들이 0인 2차원 행렬을 의미

- 희소행렬을 나타내기 위한 자료구조 #02 – **배열 / 구조체**
  : 0이 아닌 요소들만 맞추어 저장한다.
  (0이 아닌 요소들만 저장하기 때문에, 해당 값이 분포하는 위치도 함께 저장해주어야 한다. 메모리 공간을 절약할 수 있지만 행렬 연산들의 구현이 복잡해진다.)

행렬의 위치를 나타내기 위한 값들이 추가적으로 필요



배열/구조체 모두로 구현 가능

희소행렬 덧셈 프로그램

희소행렬 덧셈 프로그램

Question.
희소행렬을 저장할 수 있는 자료구조를 정의하고, 정의된 자료구조를 활용하여 희소 행렬 2개의 덧셈 연산을 완성하고 화면에 출력하시오.

- **Constraints**
 : 가장 쉽게 구현할 수 있는 방법은 행렬의 크기에 맞는 2차원 배열을 선언하는 방식, 그러나 이 방식은 대부분이 0인 희소행렬의 경우 공간의 낭비가 심하다. 따라서 0이 아닌 요소들만 맞추어 저장할 수 있는 자료구조를 정의하고 이를 활용하여 희소 행렬의 덧셈 연산을 완성하고 화면에 출력하시오.

희소행렬 덧셈 프로그램

- **구조체(Structure)를 이용한 희소행렬의 표현**

: 희소행렬의 크기 대로 2차원 배열을 선언하여 인덱스에 맞춰 값을 입력할 수도 있지만 공간의 효율이 좋지 않다. 본 예제에서는 구조체를 이용하여 0이 아닌 요소의 위치와 값을 element 변수로 구성하고, 희소행렬을 element구조체와 희소행렬의 행, 열의 크기 그리고 0이 아닌 항의 개수의 값이 들어 있는 SparseMatrix 구조체로 자료구조를 정의한다.

```
typedef struct {
        int row; // 0이 아닌 행렬 값이 존재하는 행의 위치
        int col;  // 0이 아닌 행렬 값이 존재하는 열의 위치
        int value; // 0이 아닌 행렬 값
} element;

typedef struct SparseMatrix {
        element data[MAX_TERMS];   // 0이 아닌 행렬 값과 위치

        int rows;           // 희소행렬의 전체 행의 크기
        int cols;           // 희소행렬의 전체 열의 크기
        int terms;          // 희소행렬 내의 0이 아닌 항의 개수
} SparseMatrix;
```

희소행렬 덧셈 프로그램

- **구조체(Structure)를 이용한 희소행렬의 표현**

: 앞의 구조체를 활용하여 main문 안에서 행렬 요소를 아래와 같이 입력해 볼 수 있다. 정의된 희소행렬 A과 B는 각각 표의 요소들로 구성되어 있으며, 해당되지 않는 위치들은 모두 0이 된다.

```
int main ()
{
  SparseMatrix A = {{ { 1, 0, 8 },{ 3, 1, 2 },{ 4, 3, 9 },{ 4, 5, 2 },{ 5, 2, 5 } },6,6,5};
  SparseMatrix  B = {{ { 0, 3, 7 },{ 1, 0, 9 },{ 1, 5, 8 },{ 3, 0, 6 },{ 3, 1, 5 },{ 4, 5, 1 },{ 5, 2, 2 } },6,6,7};
}
```

SparseMatrix A	SparseMatrix B
: 행렬의 크기는 6 X 6 행렬로 총 5개의 0이 아닌 값이 존재하는 희소행렬 (1, 0) = 8 (3, 1) = 2 (4, 3) = 9 (4, 5) = 2 (5, 2) = 5	: 행렬의 크기는 6 X 6 행렬로 총 7개의 0이 아닌 값이 존재하는 희소행렬 (0, 3) = 7 (1, 0) = 9 (1, 5) = 8 (3, 0) = 6 (3, 1) = 5 (4, 5) = 1 (5, 2) = 2

희소행렬 덧셈 프로그램

- **Implementation(덧셈, 출력 함수 구현) (1/3)**
  : Sparse_matrix_add 함수소개

main function

Sparse_matrix_add | printMatrix | matrix_print

Input : A, B (희소행렬 A, B)

1) 새로운 구조체 c를 선언
2) a행렬과 b행렬의 크기비교
   - 다르면 : "희소행렬크기에러"출력
   - 같으면 : c의 크기 = a나 b크기
3) 희소행렬 value값 위치비교
   - inda < indb : c행렬에 a값 입력
   - inda = indb : c행렬에 두 원소를 더한 값 입력
   - inda > indb : c행렬에 b값 입력
4) 나머지 항 옮기기

Output : C (덧셈연산을 통해 얻은 희소행렬 C)

희소행렬 덧셈 프로그램

- **Implementation(덧셈, 출력 함수 구현) (2/3)**
  : printMatrix 함수소개

main function

Sparse_matrix_add | **printMatrix** | matrix_print

Input : A, B, C (희소행렬 A, B, C)

희소 행렬을 2차원 배열로 출력하는 함수
1)receivedMatrix의 rows와 cols까지 범위를 두어 반복문을 사용하여, data element의 위치가 (i, j)의 위치와 일치할 때, data value 출력, 없다면 0 출력

2)row부터 확인하는데, if문을 두어 입력받은 data element의 열이 i열 Finding작업 시에 없으면, 해당 열에는 value값이 존재하지 않으므로 "0 0 0 0 0 0" 출력

Output : X (A, B, C 행렬을 화면에 출력)

희소행렬 덧셈 프로그램

- **Implementation(덧셈, 출력 함수 구현) (3/3)**
  : Add_printMatrix 함수소개

main function

Sparse_matrix_add | printMatrix | **matrix_print**

Input : A, B, C (희소행렬 A, B, C)

희소 행렬을 (행, 열, 값) 으로 출력하는 함수

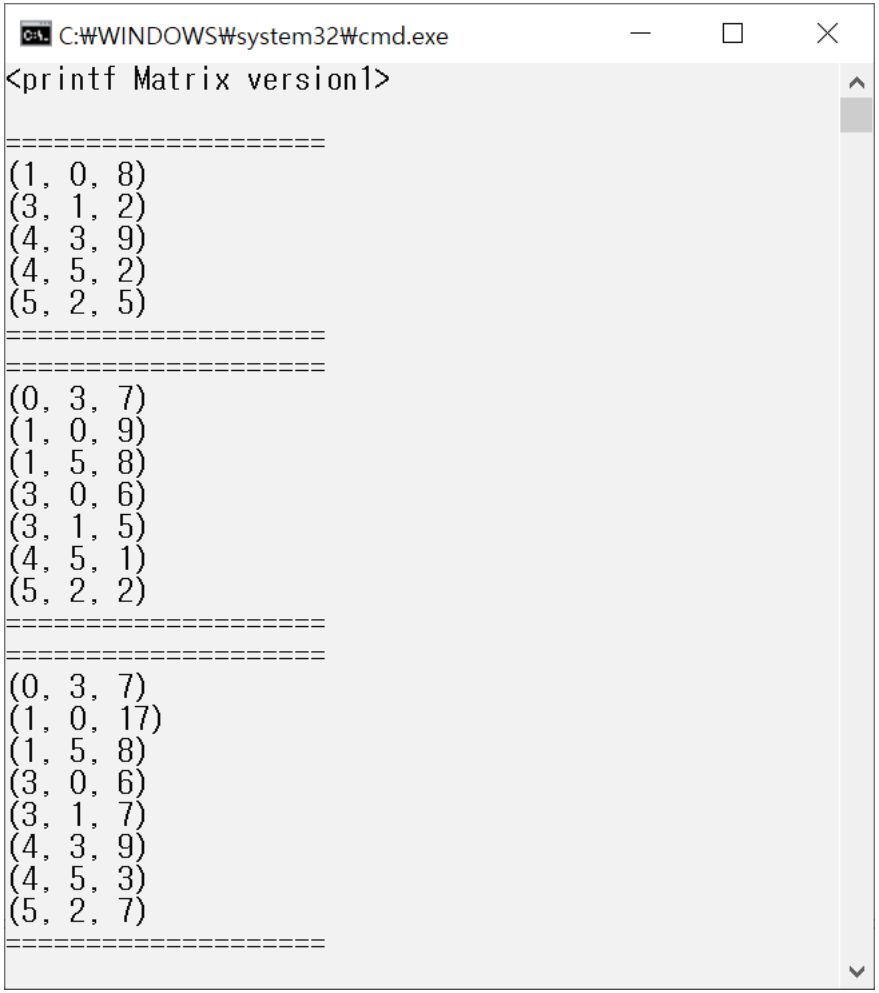
```
for (int i = 0; i < a.terms; i++) {
        printf("(%d, %d, %d) \n", a.data[i].row, a.data[i].col, a.data[i].value);
}
```

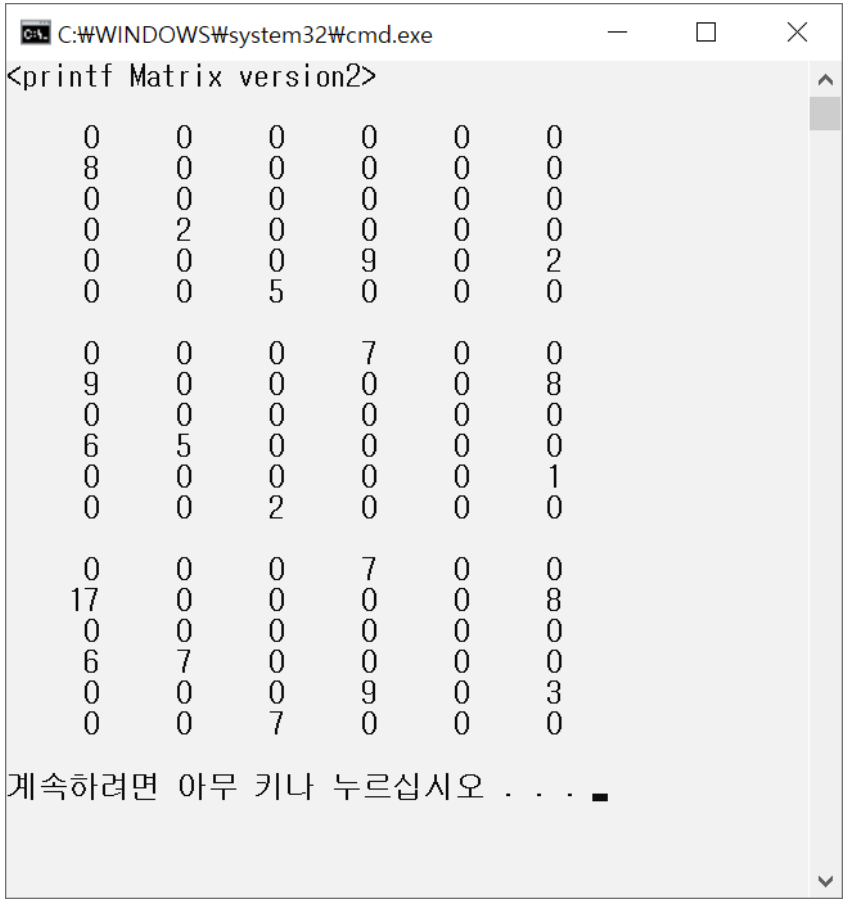
- 반복문을 사용하여, 각 희소행렬의 (행, 열, 값) 을 출력한다.

Output : X (희소행렬 A, B, C (행, 열, 값)을 print)

희소행렬 덧셈 프로그램

- **Results(결과 화면)**





Thank You

Q&A