**Report**

**For Data Structures # 3**

**학과 :**

**학번 :**

**이름 :**

컴퓨터공학과

20184071

김도현

**문제**

* 문제
  + 행렬 A, B에 대한 C 선언문(초기화 포함)을 작성하라. (행렬 타입은 SparseMatrix, 행렬의 각 요소는 (row, col, val)로 표현)
  + 위의 희소 행렬 A, B를 전달받아서 A+B를 수행하여 그 결과 희소 행렬을 C를 구하여 반환하는 add()를 작성하라.
    - A, B로부터 원소 한 개씩 가져와서 그 요소의 원래 행렬의 위치를 계산하라. (요소가 (i, j, val)이면, 그 위치는 i\*열\_크기 + 열로 계산된다)
    - 두 요소의 위치가 같으면 더하여 C로 이동하고,
    - 두 요소의 위치가 다르면 위치가 앞선 요소를 C로 이동
  + 주어진 희소 행렬 A를 원래의 행렬로 출력하는 print()를 작성하라.

1. 문제의 요구사항

- 희소행렬 A 와 B 를 구조체를 사용하여 정확하게 표현한다.

- 희소행렬 A 와 B 를 더하는 함수를 작성을 한다.

- 더하는 함수를 작성해서 C 라는 변수에 할당하고, C 라는 희소행렬을 출력한다. > add 함수 구현

- 출력할때는 row \* colSize + col 을 사용하여 출력한다. > print 함수 구현

1. 문제분석

|  |  |
| --- | --- |
| Input | Int rowSize ,int colSize, int terms, 구조체 배열을 담는 구조체 A 와 B  구조체 배열은 row, col, value 로 되어있다. |
| Output | 구조체 A B 를 더한 C 구조체를, 원래 배열로 출력한다. Process 의 희소행렬의 출력 알고리즘을 사용하면 된다. |
| Process | 희소행렬의 표현 방법  - 희소행렬의 구조체로 표현 방법은 배열의 값이 0 이 아니면 아닌 값을 구조체 배열에 저장을 하면 된다. 데이터가 3행 4열에 0 이 있으면 저장을 하지 않고, 데이터가 3행 6열에 7이 있으면 {3,6,7} 로 저장을 하면 된다.  - terms 은 구조체 배열이 가지고 있는 index 수와 같다. 즉 데이터가 {3,3,1},{4,2,4} 이렇게 두개가 있으면, terms 는 2가 된다.  희소행렬 A + B 의 덧셈  - 희소행렬 A + B 의 덧셈은 구조체 배열을 가져와서 그 두가지를 더하면 된다.  예를들어 A 는 { 3, 3 ,1},{ 3 , 5 , 4 } 고 B 는 { 3, 3, 4 }, { 5, 4, 7 } 이라고 가정을 하자생성될 배열을 C 라고 하면, A 의 첫번째인 3 3 1 과 B 의 첫번째인 3 3 4 를 비교 해야 한다.  - 이 케이스는 A 의 row col 은 3 3 , B 의 rowl col 은 3 3 이므로 A B value 를 더한 5를 C 에 저장하면 된다. { 3, 3 ,5} 이런식으로 말이다.    - A 의 3 3 1 B의 3 3 4 는 더해졌으므로 이제 A 의 3 5 4 와 B 의 5 4 7 을 비교하면 된다.  - row 값이 같을 경우 col 값도 비교해야되는데 이경우 row 값이 다르기 때문에 더 작은 row 를 가진 A 의 값이 그대로 C 배열로 들어가면 된다. C 배열은 곧 {3, 3, 5},{3, 5, 4} 가 된다. 그럼 A 의 요소들은 모두 사용됬기 때문에 이제, 남은 B 배열이 C 배열로 그냥 내려오면 된다. 이 알고리즘을 적용해서 구하면, C 는 결국 {3, 3, 5},{3, 5, 4},{5, 4, 7} 이 남게 된다.  위 row 와 col 을 비교하는 알고리즘은 작성하기 매우 까다롭다. 그렇기 떄문에 다음과 같이 배열의 순서를 확인 할 수 있다.  Row \* 배열의열크기 + col  이 변수로 비교 연산을 하면 쉽게 수행 가능하다.  저 값을 각각 Pos1 Pos2 라 하면 pos1 == pos2 일 경우 데이터가 같은 위치 이다. Pos1>Pos2 일 경우 Pos2 가 먼저 들어가야되는 데이터, Pos1 <Pos2 경우 Pos1 이 먼저 들어가야되는 데이터 이다.  - 그러나 3 \* 4 , 5 \* 2 배열의 덧셈도 고려해야 한다. 이 경우 row 와 col 값이 다르므로 더할수 없다. 그러면, 프로그램을 종료시켜야한다.  - 이는 곧 add 함수의 알고리즘이다.  3. 희소 행렬의 출력  희소행렬에는 terms 와 row 와 col, data 가 필드로 이뤄져있다.  그렇기 때문에 2중 for 문을 사용하여 첫번쨰 for 문은 rowSize 전까지, 두번쨰 for 문은 colSize 전까지 반복을 한다.  만약 data 배열의 n 번쨰 인덱스의 row 값이 I 와 일치하고, col 값이 일치하면 value 를 출력하면 된다. 그렇지 않을 경우 0 을 출력하면 된다.  두번쨰 for 문이 끝나면 개행을 해줘, 원본 배열이 출력되게 한다. |
| Example | 1. rowSize colSize 값이 일치하는 희소행렬 A B 를 더하고 그 더한배열을 출력한다.  2. 희소행렬 A 의 rowSize 가 B 의 rowSize 보다 더 큰 희소행렬 A B 덧셈은 프로그램을 종료한다.  3. 희소행렬 A 의 colSize 가 B 의 colSize 보다 더 큰 희소행렬 A B 덧셈은 프로그램을 종료한다.  해당 출력은 검증 및 테스트 부에서 확인이 가능하다. |

1. 의사 코드를 사용하여 알고리즘 작성

Add 함수

Type element = record

Row = int

Col = int

Values = int

EndType

Type Sparse\_matrix = record

rowSize = int

colSize = int

Terms = int

Data = array [1…Max] of element

End Type

**Procedure** add (Sparse\_matrix matrix1, Sparse\_matrix matrix2)

Sparse\_matrix temp;

Int iT <- 0

Int i1 <- 0

Int i2 <- 0

If matrix1.rowSize != matrix2.rowSize or matrix1.colSize != matrix2.colSize then

Exit

endif

Temp.rowSize -> matrix1.RowSize

Temp.colSize -> matrix1.colSize

Temp.terms = 0

Int Matrix1Pos;

Int Matrix2Pos;

**while** (matrix1.terms > i1 and matrix2.terms > i2) do

Matrix1Pos <- matrix1.data[i1].row \* matrix1.col + matrix1.data[i1].col

Matrix2Pos <- matrix2.data[i2].row \* matrix2.col + matrix2.data[i2].col

**if** (Matrix1Pos = Matrix2Pos ) then

temp.data[it].col <- matrix1.data[i1].col

temp.data[it].row <- matrix1.data[i1].row

temp.data[it].value <- matrix1.data[i1].value + matrix2.data[i2].value;

i1 -> i1 + 1, i2 -> i2 + 1 , it -> it + 1;

else if ((Matrix1Pos > Matrix2Pos) then

temp.data[it].col <- matrix2.data[i2].col;

temp.data[it].row <- matrix2.data[i2].row

temp.data[it].value <- matrix2.data[i2].value;

i2 <- i2 + 1 , it <- it + 1;

else

temp.data[it].col <- matrix1.data[i1].col

temp.data[it].row <- matrix1.data[i1].row

temp.data[it].value <- matrix1.data[i1].value;

i1 <- i1 + 1, it <- it + 1;

**endif**

**repeat**

temp.terms <- it

return temp

**end add**

print 함수

Procedure print(sparse\_matrix matrix) {

int index <- 0

**for** ( int I <- 0 to matrix.rows by I <- I + 1) {

for (int j <- 0 to matrix.cols by j <- J +1)

if (I = matrix.data[index].row and j = matrix.data[index].col) then

print(matrix.data[index].value)

else then print(0 )

endif

print(“\n”)

repeat

**repeat**

end print

1. C 언어를 사용한 프로그램 작성

/\*

\* 문제

\* 행렬 A, B에 대한 C 선언문(초기화 포함)을 작성하라.

\* (행렬 타입은 SparseMatrix, 행렬의 각 요소는 (row, col, val)로 표현)

\* 희소행렬 A B 를 받아서 그 결과를 반환하는 add 함수를 작성하라.

\* 알고리즘

\* 1. 희소행렬의 표현 :

\* 희소행렬의 표현은 value 가 0 이 아닌 값을 구조체 배열에 저장을 하면된다.

\* 그 데이터는 0 이 아닌 value 의 row 값, col 값을 저장을 하게 된다.

\* {2,5,4} 로 저장이 되어있으면, 2행 5열의 4 라는 값이 있다는 뜻 이다.

\*

\* 2. 희소행렬 A + B 의 덧셈

\* 희소행렬 A + B 의 덧셈은 구조체 배열을 가져와서 그 두가지를 더하면 된다.

\* 예를들어 A 는 { 3, 3 ,1},{ 3 , 5 , 4 } 고 B 는 { 3, 3, 4 }, { 5, 4, 7 } 이라고 가정을 하자

\* 생성될 배열을 C 라고 하면, A 의 첫번째인 3 3 1 과 B 의 첫번째인 3 3 4 를 비교 해야 한다.

\* 이 케이스는 A 의 row col 은 3 3 , B 의 rowl col 은 3 3 이므로 A B value 를 더한 5를 C 에 저장하면 된다.

\* { 3, 3 ,5} 이런식으로 말이다.

\* A 의 3 3 1 B의 3 3 4 는 더해졌으므로 이제 A 의 3 5 4 와 B 의 5 4 7 을 비교하면 된다.

\* row 값이 같을 경우 col 값도 비교해야되는데 이경우 row 값이 다르기 때문에 더 작은 row 를 가진

\* A 의 값이 그대로 C 배열로 들어가면 된다. C 배열은 곧 {3, 3, 5},{3, 5, 4} 가 된다.

\* 그럼 A 의 요소들은 모두 사용되기 때문에 이제, 남은 B 배열이 C 배열로 그냥 내려오면 된다.

\* 그러나 이 row 와 col 을 비교하는 경우는 너무 비효율적이다.

\* 그렇기 떄문에 matrix.data[index].row \* matrix.colSize + matrix.data[index].col 을 사용해서 비교한다

\* 위 식을 matrix1Pos 와 matrix2Pos 라 생각하면 둘이 같은경우는, 위치가 같은 경우,

\* matrix1Pos > matrix2Pos 일 경우 matrix2Pos 가 먼저 들어가야 하는경우

\* 그 외의 경우는 matrix1Pos 가 들어가야 하는 경우다.

\* 그러나 3 \* 4 , 5 \* 2 배열의 덧셈도 고려해야 한다.

\* 희소행렬의 크기가 다를 경우 더할수 없다 즉, 종료해야한다.

\* 이는 곧 add 함수의 알고리즘이다.

\*

\* 3. 희소 행렬의 출력

\* 희소행렬에는 terms 와 row 와 col, data 가 필드로 이뤄져있다.

\* 희소행렬의 n 번째 데이터는 다음과 같이 표현을 하면 된다.

\* n = col \* data[index].row + data[index].col

\* for 문을 사용하여 i < col \* row 까지 반복을 하고, 만약 i 값이 n 과 일치하면

\* n 번째 데이터를 출력한다. 개행은 i 를 col 로 나누었을때 나누어 떨어지면 개행을 하면 된다.

\*

\* 출력

\* 희소행렬 A B 를 더한 C 의 원래 배열의 형태로 출력을 하면 된다.

\* 3 번 알고리즘을 통해 구현이 가능하다.

\*

\* 함수

\* sparse\_matrix add(sparse\_matrix matrix1, sparse\_matrix matrix2) : matrix1 과 matrix2 의 합한 구조체 반환

\* void print(sparse\_matrix matrix); matrix 배열을 출력하는 함수

\*/

#include <stdio.h>

#define MAX\_TERMS 100 //data 배열에 들어갈 최대 갯수를 기호 상수로 선언

#define ROW 5 //행의 크기를 기호 상수로 선언

#define COL 10 //열의 크기를 기호 상수로 선언

typedef struct {

int row; //데이터의 row 를 저장한다.

int col; //데이터의 col 을 저장한다.

int value; //데이터를 저장한다.

} element; //구조체를 element 로 타입을 지정한다.

typedef struct {

element data[MAX\_TERMS]; //MAX\_TERMS 만큼 data의 최대 배열 길이를 지정해준다.

int rows; //행의 총 크기

int cols; //열의 총 크기

int terms; //data 배열의 들어간 index 수

} sparse\_matrix;

sparse\_matrix add(sparse\_matrix matrix1, sparse\_matrix matrix2); //sparse\_matrix 2개를 받아 더하고 반환하는 add 함수 원형 정의

void print(sparse\_matrix matrix); //sparse\_matrix 를 받아서 출력하는 print 함수 원형 정의

int main(void) {

sparse\_matrix SparseA = { { { 0, 2, 1 },

{ 1, 5, 2 },

{ 3, 1, 3 },

{ 3, 9, 4 },

{ 4, 0, 5 },

{ 4, 4, 6 },

{ 4, 9, 7 } },5,10,7 }; //SparseA 구조체를 선언하고 초기화후, 데이터를 넣는다.

//row 는 5 col 은 10 terms 는 7이다.

sparse\_matrix SparseB = { {{ 0, 0, 5 },

{ 0, 4, 6 },

{ 0, 9, 7 },

{ 1, 1, 3 },

{ 1, 5, 1 },

{ 1, 9, 4 },

{ 2, 2, 1 },

{ 3, 5, 2 },

{ 4, 4, 1 },

{ 4, 9, 1 }},5,10,10 }; //SparseB 구조체를 선언하고 초기화후, 데이터를 넣는다.

//row 는 5 col 은 10 terms 는 10이다.

sparse\_matrix SparseC = add(SparseA, SparseB); //SparseA 와 SparseB 를 더해서 SpareseC 에 저장한다.

print(SparseC); //저장된 SparseC 를 출력한다.

}

sparse\_matrix add(sparse\_matrix matrix1, sparse\_matrix matrix2) { //더하고 구조체로 반환하기 때문에 sparse\_matrix 를 쓴다.

sparse\_matrix temp; //반환할 구조체 선언

int it = 0, i1 = 0, i2 = 0;

//temp 의 index matrix1 의 index matrix2 의 index 변수 it i1 i2 를 선언한다.

//index 의 초기값은 0이다.

if (matrix1.rows != matrix2.rows || matrix1.cols != matrix2.cols) {

printf("희소행렬 크기 에러");

exit(1);

}

temp.rows = matrix1.rows;

//알고리즘 설명에 있던 c 배열의 row 는 matrix 1 의 row 가 된다,..

temp.cols = matrix1.cols;

//C배열의 col 은 a b 의 col matrix1 의 col 이 된다.

temp.terms = 0; //데이터가 0 개 삽입되었으므로 0 이다.

int Matrix1Postion, Matrix2Postion;

while (matrix1.terms > i1 && matrix2.terms > i2) {

//matrix1의 데이터 보관수가 i1 보다 클때까지만 해야된다, 그렇지 않으면 data 의 이상한 값도 저장되기 때문

//마찬가지로 i2 도 matrix2 의 데이터 보관수보다 작을때 까지만 해야된다.

Matrix1Postion = matrix1.data[i1].row \* matrix1.cols + matrix1.data[i1].col; //받아온 희소행렬 a matrix1 의 위치

Matrix2Postion = matrix2.data[i2].row \* matrix2.cols + matrix2.data[i2].col; //받아온 희소행렬 b matrix2 의 위치

if (Matrix1Postion == Matrix2Postion) { //데이터가 같은 row 값 col 값일 경우

temp.data[it].col = matrix1.data[i1].col; //matrix1 이나 2 의 col 값 할당

temp.data[it].row = matrix1.data[i1].row;//matrix1 이나 2 의 row 값 할당

temp.data[it].value = matrix1.data[i1].value + matrix2.data[i2].value; //matrix1 과 2 의 더한값 할당

i1++, i2++, it++; //matrix1 의 i1 데이터 matrix2 의 i2 데이터를 사용했으므로 증가

//it 도 마찬가지로 데이터를 옆에 저장해야하므로 1 증가

}

//row 가 다른 경우, col 값은 볼 필요가 없다.

else if (Matrix1Postion > Matrix2Postion) { //데이터가 다른 row 값 col 값일 경우, matrix1 이 더 뒤에 있는경우다.

//해당 케이스는 2 1 3 과 1 8 2 인 경우인데

//1 8 2 가 저장되어야되므로 matrix2.data[i2] 정보가 그대로 가야된다.

temp.data[it].col = matrix2.data[i2].col; //matrix2 의 col 데이터 할당

temp.data[it].row = matrix2.data[i2].row; //matrix2 의 row 데이터 할당

temp.data[it].value = matrix2.data[i2].value; //matrix2 의 value 데이터 할당

i2++, it++; //matrix2 의 i2 인덱스는 사용했으므로 1 증가

//it 도 마찬가지로 데이터를 옆에 저장해야하므로 1 증가

}

else {

//해당 케이스는 1 8 2 과 2 8 2 인 경우인데

//1 8 2 가 저장되어야되므로 matrix1.data[i2] 정보가 그대로 가야된다.

temp.data[it].col = matrix1.data[i1].col;//matrix1 의 col 데이터 할당

temp.data[it].row = matrix1.data[i1].row;//matrix1 의 row 데이터 할당

temp.data[it].value = matrix1.data[i1].value;//matrix1 의 value 데이터 할당

i1++, it++;//matrix2 의 i2 인덱스는 사용했으므로 1 증가

//it 도 마찬가지로 데이터를 옆에 저장해야하므로 1 증가

}

}

temp.terms = it; //it 이 데이터가 증가될때마다 증가되므로 terms 는 it 와 같다.

return temp;//반환할 구조체 temp 를 반환한다.

}

void print(sparse\_matrix matrix) {//구조체를 출력하는 함수 print 인자로는 matrix 를 받는다.

int index = 0; //matrix.data 의 index 를 설정해준다.

for (int i = 0; i < matrix.rows; i++) { //받아온 matrix 의 row 만큼,

for (int j = 0; j < matrix.cols; j++) { //받아온 matrix 의 col 만큼 반복한다.

if (matrix.data[index].row == i && matrix.data[index].col == j) { //만약 matrix data 의 index 번쨰의 row 와 col 이 i j 일 경우

printf("%d ", matrix.data[index].value); //데이터가 있다는 말이므로 출력한다.

index++; //index 를 늘려 옆 배열을 확인한다.

}

else {

printf("0 "); //그렇지 않으면 0을 출력한다.

}

}

printf("\n");

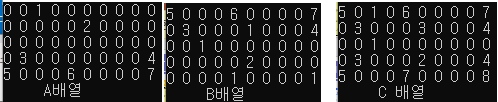
}

}

Github.com/wookoo/data-Stricture 에 가시면 깔끔한 소스를 보실수 있습니다.

1. 검증 및 테스트

A . rowSize colSize 가 같을 경우



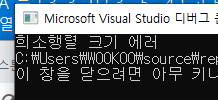
A 배열 B 배열 모두 rowSize ColSize 가 5 10 으로 같기에 C 배열도 5행 10 열만 출력된다.

B. A 의 rowSize (8 )가 B 의 rowSize(5) 보다 더 큰 경우



A B 배열 모두 rowSize ColSize 가 같아야 덧셈을 할 수 있다.

C. A의 colSize (8) 가 B 의 colSize(5) 보다 더 큰 경우



A B 배열 모두 colSize rowSize 가 같아야 덧셈을 진행 할 수 있다.

1. 의견

희소행렬을 출력을 구현 할떄, row \* 배열의열크기 + col 을 사용해서 구현을 하라고 문제에 제시되어있었다. 그러나 내 생각엔 이 방법 없이 element 구조체의 data 배열에서 row 와 col 값이 2중 for 문의 row 와 col 값이 일치하면 출력 하고 index 를 늘리는 방법이 더 직관적이라 생각된다,

아직도 왜 row \* 배열의열크기 + col 을 사용하는지 모르겠다.