자료구조 HW3 희소행렬 덧셈

CSE3080

20231604 정유연

1. 기본 구조체

```
#include <stdio.h>
#include <stdlib.h>
#pragma warning (disable:4996)
#define MAX_SIZE 50
#define MAX_TERMS 101

typedef enum { head, entry } tagfield;
typedef struct matrix_node* matrix_pointer;
matrix_pointer hdnode[MAX_SIZE];

typedef struct entry_node {
    int row;
    int col;
    int value;
}EntryNode;

typedef struct matrix_node {
    matrix_pointer down;
    matrix_pointer right;
    tagfield tag;
    matrix_pointer next;
    EntryNode entry;

}MatrixNode;

matrix_pointer mread(FILE*); // file에서 matrix 자료를 읽어 리스트에 저장한다.
void mwrite(matrix_pointer); // 행렬을 적절한 포맷 (2차원 행렬형태)으로 출력한다.
void merase(matrix_pointer*); // 행렬의 모든 노드를 지운다.
matrix_pointer madd(matrix_pointer, matrix_pointer); // matrix addition
```

문제로 주어진 기본 구조체이다.

matrix_node의 down은 같은 열의 다음 행 노드를 가리키고, right는 같은 행의 다음 열 노드를 가리킨다. 이번 코드에서 down은 구현은 돼있지만 사용하지는 않는다.

next는 각 행의 시작 노드로, next로 다음 행끼리 연결돼있다. entry에는 각 노드의 행,열, 값이 저장돼있다.

mread에서는 외부 file을 읽어 행렬을 저장하고, mwrite에서는 행렬을 출력한다. 이때 0

이 포함된 2차원 배열로 출력한다. merase에서는 매개변수로 들어온 행렬의 모든 원소를 지운다. 이때 기존 행렬의 사이즈만을 남겨둔다. madd에서는 매개변수로 들어온 두 행렬의 덧셈을 진행한다. 이때 행렬의 사이즈는 같은 것으로 전제한다.

2. mread함수

```
matrix_pointer mread(FILE* file) {
    int NumRows, NumCols, NumEntries = 0, NumHeads, i;
    int row, col, value;
    matrix_pointer temp, last, node;

    fscanf(file, "%d %d", &NumRows, &NumCols);
    NumHeads = (NumCols > NumRows) ? NumCols : NumRows;
    printf("%d, %d₩n", NumRows, NumCols);
    /* set up head node for the list of head nodes */
    node = (matrix_pointer)malloc(sizeof(MatrixNode));
    node->tag = entry;
    node->entry.row = NumRows;
    node->entry.col = NumCols;
```

numrows, numcols는 matrix의 행과 열의 값이다. node의 시작인 node->entry.row와 col에 그 값을 저장한다.

```
    if (!NumHeads)
        node->right = node;
    else {
        // 헤더노드 초기화
        for (i = 0; i < NumHeads; i++) {
            temp = (matrix_pointer)malloc(sizeof(MatrixNode));
            hdnode[i] = temp;
            hdnode[i]->tag = head;
            hdnode[i]->right = temp;
            hdnode[i]->next = temp;
        }
```

만약 희소행렬의 행,열값이 0이라면 노드의 right를 자기 자신으로 해서 노드가 시작하지 않게 한다. 그렇지 않을 경우 행과 열 둘 중 더 큰 수의 크기만큼 matrix_pointer를 동적

할당한다. 예를 들어 행이 6, 열이 7인 크기의 matrix라면 현재 위 코드를 거쳐 7개의 연속적인 matrix_pointer가 생성된 것이다.

```
last = hdnode[0];
for (row = 0; row < NumRows; row++) {
    for (col = 0; col < NumCols; col++) {
        fscanf(file, "%d", &value);
        if (value != 0) {
            NumEntries++;
            temp = (matrix_pointer)malloc(sizeof(MatrixNode));
            temp->tag = entry;
            temp->entry.row = row;
            temp->entry.col = col;
            temp->entry.value = value;

            last->right = temp;          // 행리스트에 연결
            last = temp;
            hdnode[col]->next->down = temp;  // 열리스트에 연결
            hdnode[col]->next = temp;
        }
    }
}
node->entry.value = NumEntries;
last->right = NULL;
```

last는 바로 이전의 노드를 가리키고, temp는 새로 만드는 노드이다.

fscanf로 파일에서 값을 불러들이고 0이 아닌 값이라면 노드에 entry.row , col, value값을 새로 저장한다. 이렇게 만든 temp를 last의 right로 연결하고, hdnode배열을 통해 down으로도 연결한다. numentries에는 희소행렬의 0이 아닌 값의 크기를 저장하고, 맨 마지막 노드는 NULL을 가리키게 한다.

```
// close all column lists
        for (i = 0; i < NumCols; i++)
            hdnode[i]->next->down = hdnode[i];

        // link all head nodes together
        for (i = 0; i < NumHeads - 1; i++)
            hdnode[i]->next = hdnode[i + 1];

        hdnode[NumHeads - 1]->next = node;
        node->right = hdnode[0];
    }
    printf("FINISH READING₩n");
    return node;
```

반복문이 끝난 다음에는 hdnode배열을 통해 각 행과 열을 다 같이 이어준다. 이렇게 만든 node를 return한다.

3. madd함수

mread와 유사하게 새로운 노드를 만들어 반환하는 함수다. 이때 value값을 a, b matrix의 value를 더한 값으로 정한다.

```
matrix_pointer madd(matrix_pointer a, matrix_pointer b) {
    int NumRows, NumCols, NumEntries = 0, i;
    int row, col, value;
    matrix_pointer temp, last, node;

    int r, c, NumHeads, inda = 0, indb = 0;
    r = a->entry.row;
    c = a->entry.col;
    NumHeads = (c > r) ? c : r;
```

```
    /* set up head node for the list of head nodes */
    node = malloc(sizeof(MatrixNode));
    node->tag = entry;
    node->entry.row = r;
    node->entry.col = c;
 if (!NumHeads)
        node->right = node;
    else {
        // 헤더노드 초기화
        for (i = 0; i < NumHeads; i++) {
            temp = malloc(sizeof(MatrixNode));
            hdnode[i] = temp;
            hdnode[i]->tag = head;
            hdnode[i]->right = temp;
            hdnode[i]->next = temp;
        }
```

mread와 거의 같다. inda, indb는 각각 희소행렬 a, b의 현재 노드의 col인덱스를 뜻한다.

```
matrix_pointer acop, bcop;
        acop = a->right->right;
        bcop = b->right->right;
        last = hdnode[0];
        for (row = 0; row < r; row++) {
            for (col = 0; col < c; col++) {
                if (acop == NULL) {
                    inda = c + 3;
                }
                if (bcop == NULL) {
                    indb = c + 3;
                }
                else {
                    if (acop->entry.row == row && bcop->entry.row == row) {
```

```
                inda = acop->entry.col;
                indb = bcop->entry.col;
            }
            else {
                if (acop->entry.row != row) inda = c + 2;
                if (bcop->entry.row != row) indb = c + 2;
            }
        }
```

acop, bcop은 각각 행렬 a,b를 가리키는 포인터이다. 이중반복문을 통해 행렬의 크기만큼 탐색한다.

반복문 도중 행렬의 맨마지막 노드가 나온다면, 즉 acop또는 bcop이 NULL이라면 해당 인덱스를 콜룸 사이즈보다 크게 해서 아래 if비교문을 진입하지 못하게 한다. 만약 이렇게 하지 않는다면 segmentation fault가 발생할 수 있다.

또는 현재 비교중인 col인덱스가 정해진 row값이 아닐 때, 즉 어느 한쪽이 이미 다음 row로 넘어간 상황일 때도 인덱스 값을 콜룸 사이즈보다 크게 해서 아래 if비교문을 진입하지 못하게 한다. 이렇게 하지 않는다면, 두 matrix가 다른 행을 가리키기에 열의 크기를 비교할 때 오류가 생길 수 있다.

그렇지 않고 노드의 row인덱스가 현재 비교 중인 row값과 같을 때 비로소 inda와 indb 값을 행렬 a,b의 현재 노드의 col인덱스로 정한다.

```
if (inda < indb) {
            if (inda == col) {
                value = acop->entry.value;
                acop = acop->right;
            }
            else { value = 0; }
        }
        else if (inda > indb) {
            if (indb == col) {
                value = bcop->entry.value;
                bcop = bcop->right;
```

```
        }
        else { value = 0; }
    }
    else {
        if (indb == col) {
            value = acop->entry.value + bcop->entry.value;
            acop = acop->right; bcop = bcop->right;
        }
        else { value = 0; }
    }
```

만약 inda가 indb보다 작은 상황이라면 inda인덱스를 먼저 추가해줘야 하기 때문에 value를 현재 a위치의 값으로 정한다. 그리고 a는 다음 노드로 넘어간다.(acop = acop->right) 이때 중요한 점은 해당 인덱스가 현재 반복문 속 진행중인 iterator, 즉 column 값과 같을 때(inda==col) value값을 정하는 것이다. 같지 않다면 아직 해당 위치에는 matrix원소를 추가하지 않기 때문에 value를 0으로 한다.

반대로 indb가 inda보다 작은 상황이면 indb인덱스를 추가해줘야 하기 때문에 value를 현재 b위치의 값으로 정한다. 그리고 b는 다음 노드로 넘어간다.(bcop = bcop->right)

else는 inda==indb인 상황이다. 이때도 위와 같은 원리로 동작한다.

```
    if (value != 0) {
        NumEntries++;
        temp = malloc(sizeof(MatrixNode));
        temp->tag = entry;
        temp->entry.row = row;
        temp->entry.col = col;
        temp->entry.value = value;

        last->right = temp;          // 행리스트에 연결
        last = temp;
```

```
                hdnode[col]->next->down = temp;  // 열리스트에 연결
                hdnode[col]->next = temp;
            }
        }
    }
    node->entry.value = NumEntries;
    // close all column lists
    for (i = 0; i < c; i++)
        hdnode[i]->next->down = hdnode[i];

    // link all head nodes together
    for (i = 0; i < NumHeads - 1; i++)
        hdnode[i]->next = hdnode[i + 1];

    hdnode[NumHeads - 1]->next = node;
    node->right = hdnode[0];
  }
  return node;
}
```

mread에서 행렬에 값을 추가한 코드와 같다. 이렇게 함으로써 모든 노드를 행과 열로 연결하고, 마지막에는 헤드노드인 node를 return 한다.

4. mwrite함수

```
5.  void mwrite(matrix_pointer node)
6.  {
7.      int r = node->entry.row;
8.      int c = node->entry.col;
9.      matrix_pointer head, temp;
10.     // matrix dimensions
11.     printf("₩nrows = %d, cols = %d, terms = %d₩n", node->entry.row,
12.         node->entry.col, node->entry.value);
```

```
13.     printf("< The Matrix >₩n");
14.
15.     //전체가 0인 행렬일 때
16.     if (node->right == NULL) {
17.         for (int i = 0; i < node->entry.row; i++) {
18.             for (int j = 0; j < node->entry.col; j++) {
19.                 printf("0 ");
20.             }
21.             printf("₩n");
22.         }
23.         return;
24.     }
```

mwrite는 주어진 node를 0이 포함된 희소행렬 형태로 출력하는 함수다.

만약 merase를 거쳐 아무것도 없는 함수라면 node->right가 NULL이어서 잘못 접근할 때 segmentation fault가 나오므로 따로 정의 해준다. if(node->right==NULL)

이때는 지워지기 전 기존 row,col사이즈만큼의 0을 반복해서 출력한다.

```
head = node->right;
temp = head->right;
for (int i = 0; i < r; i++) {
    //print out the entries in each row
    for (int j = 0; j < c; j++) {
        if (j == temp->entry.col) {
            printf("%d ", temp->entry.value);
            if (temp->right != NULL) {
               temp = temp->right;
            }
        }
        else printf("0 ");
        //printf("j = %d, temp->entry.col = %d₩n", j, temp->entry.col);
    }
    printf("₩n");
 }
```

이중 반복문으로 0이 포함된 희소행렬을 출력한다. 그 방법은 위 madd에서 썼던 방법과 유사하다. 현재 노드인 temp의 col값이 반복되는 interator, 즉 현재 콜룸값과 같을 때만

temp의 value를 출력하고, 다음 노드로 넘긴다. (temp=temp->right;) 현재 출력할 위치가 아닌 경우에는 0을 출력한다. 현재 iterator의 위치에는 행렬의 값이 0이라는 뜻이기 때문이다.

5. merase함수

행렬의 모든 값을 동적할당 해제하고 기존 row, col값만 남겨두는 함수이다.

```
void merase(matrix_pointer* node)
{
    matrix_pointer head = (*node)->right;
    matrix_pointer x, y, next, newnode;
    // 새로운 노드 할당
    newnode = malloc(sizeof(MatrixNode));
    newnode->tag = entry;
    newnode->entry.row = (*node)->entry.row;
    newnode->entry.col = (*node)->entry.col;
    newnode->entry.value = 0;
newnode->right = NULL;
```

head는 지워야할 행렬의 맨 처음을 가리키는 matrix_pointer이다. 매개변수로 들어온 node를 모두 지우고 node는 빈 행렬을 가리키게 하기 위해 새로운 matrix_pointer를 만든다. 새로운 matrix_pointer인 newnode에는 기존 matrix의 행의 크기와 열의 크기만 저장해둔다.

```
// free each cols
    for (int i = 0; i < (*node)->entry.row; i++) {
        y = head;
        while (y != head) {
            next = y->right;
            free(y);
            y = next;
```

```
        }
        head = head->next;
    }
    // free each rows
    y = head;
    while (y != *node) {
        x = y;
        y = y->next;
        free(x);
    }
    // change node
    free(*node);
    (*node) = newnode;
    printf("FINISH ERASING!₩n");
}
```

먼저 row에 있는 각 col를 해제하고, 남은 row들도 free해준다. 마지막으로 *node까지 free해준 다음 node포인터가 newnode를 가리키게 한다.

6. main함수

```
int main() {
    matrix_pointer a, b, d;
    //파일 오픈한다.

    FILE* fileA = fopen("A.txt", "r");
    if (fileA == NULL) {
        printf("Error opening file A.txt.₩n");
        return 1;
    }
    FILE* fileB = fopen("B.txt", "r");
    if (fileB == NULL) {
        printf("Error opening file B.txt.₩n");
        fclose(fileA);
        return 1;
    }

    a = mread(fileA);
    b = mread(fileB);
    mwrite(a);
```

```
mwrite(b);
d = madd(a, b);
mwrite(d);
merase(&d);
merase(&a);
merase(&b);
mwrite(a);
mwrite(b);
mwrite(d);

fclose(fileA);
fclose(fileB);
```

파일 A.txt와 B.txt를 mread로 읽어서 mwrite로 출력해주고, madd한 값을 d에 저장하고 mwrite로 출력한다. 이후 d,a,b 를 모두 merase로 지우고 a,b,d를 출력한다.

맨 처음에 fopen을 해줬기 때문에 fclose도 잊지 않는다.

7. 실행결과