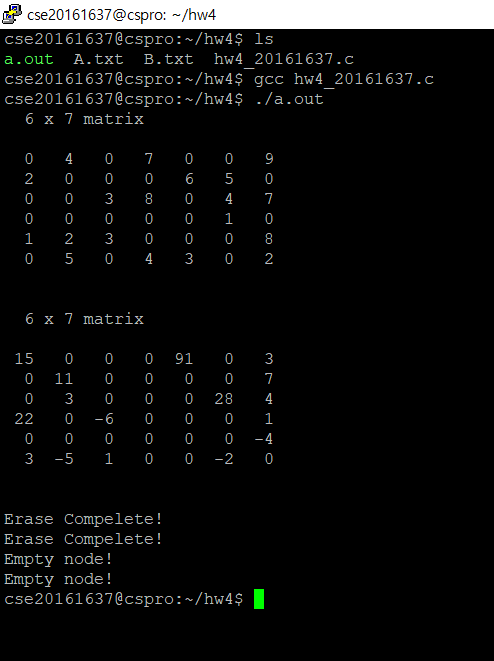
**2021 자료구조**

**오픈랩 과제 #4 : Matrix with linked list representation**

**20161637 장호영**

**<캡쳐 결과>**



#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 50

typedef enum { head, entry } tagfield;

typedef struct matrix\_node\* matrix\_pointer;

typedef struct entry\_node {

int row;

int col;

int value;

} entry\_node;

typedef struct matrix\_node {

matrix\_pointer down;

matrix\_pointer right;

tagfield tag;

union {

matrix\_pointer next;

entry\_node entry;

} u;

} matrix\_node;

matrix\_pointer hdnode[MAX\_SIZE];

matrix\_pointer new\_node();

matrix\_pointer mread(FILE\*);

void mwrite(matrix\_pointer);

void merase(matrix\_pointer\*);

int main()

{

FILE\* fp1, \* fp2;

fp1 = fopen("A.txt", "r");

fp2 = fopen("B.txt", "r");

//file open error 체크

if (fp1 == NULL || fp2 == NULL) {

printf("Check file A.txt or B.txt!");

return 0;

}

matrix\_pointer a, b;

a = mread(fp1);

b = mread(fp2);

mwrite(a);

mwrite(b);

merase(&a);

merase(&b);

mwrite(a);

mwrite(b);

return 0;

}

//node 새로 할당 받기

matrix\_pointer new\_node() {

matrix\_pointer temp = (matrix\_pointer)malloc(sizeof(matrix\_node));

temp->down = NULL;

temp->right = NULL;

return temp;

}

//.txt파일로부터 매트릭스 정보를 linked list 형태로 변환하여 저장하기

matrix\_pointer mread(FILE\* fp)

{

int num\_rows, num\_cols, num\_heads, i;

int row, col, value, current\_row;

int temp\_row=0, temp\_col=0,res;

matrix\_pointer temp, last, node;

fscanf(fp, "%d %d", &num\_rows, &num\_cols); //행렬 사이즈 정보 받기

num\_heads = (num\_cols > num\_rows) ? num\_cols : num\_rows; //row와 col 둘 중 큰 값으로 돌릴예정

node = new\_node();

node->u.entry.row = num\_rows;

node->u.entry.col = num\_cols;

if (!num\_heads) node->right = node; //사이즈가 없는 경우로 자기 자신을 가리키도록 설정

else {

for (i = 0; i < num\_heads; i++) {

//head설정해주기

temp = new\_node();

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

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

}

current\_row = 0; last = hdnode[0];

while(fscanf(fp, "%d", &res)!=EOF) {//나머지 .txt 파일을 읽어들이는 동안

row = temp\_row;

col = temp\_col;

if (res == 0) { //원소가 0인경우 skip해주며 temp\_row와 temp\_col을 설정함

temp\_col++;

if (temp\_col == num\_cols) {

temp\_row++;

temp\_col = 0;

}

continue;

}

else { //원소가 value가 있으면 value값 대입해주고

value = res;

temp\_col++;

if (temp\_col == num\_cols) {

temp\_row++;

temp\_col = 0;

}

}

if (row > current\_row) {//구조 만들어주기

last->right = hdnode[current\_row];

current\_row = row; last = hdnode[row];

}

//row col value넣고 노드를 생성/삽입

temp = new\_node(); temp->tag = entry;

temp->u.entry.row = row; temp->u.entry.col = col;

temp->u.entry.value = value; last->right = temp; last = temp;

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

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

}

// close last row

last->right = hdnode[current\_row];

// close all column lists

for (i = 0; i < num\_cols; i++)

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

// link all head nodes together

for (i = 0; i < num\_heads - 1; i++)

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

hdnode[num\_heads - 1]->u.next = node;

node->right = hdnode[0];

}

return node;

}

void mwrite(matrix\_pointer node)

{

int i;

int temp\_row=0, temp\_col=0;

if (node == NULL) //빈 노드일 경우

{

printf("Empty node! \n");

return;

}

matrix\_pointer temp, head = node->right;

printf(" %d x %d matrix\n\n",node->u.entry.row, node->u.entry.col);

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

temp\_col = 0;

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

while (1) {

//linked list값을 보고 해당 row와 col 직전까지는 0을 출력

if (temp\_col++!= temp->u.entry.col) {

printf(" 0 ");

}

else { //값이 있는 위치에서는 value 출력

printf("%3d ", temp->u.entry.value);

break;

}

}

}

//value 출력 후 나머지 비어있는 원소들에 0 출력

for (int j = temp\_col; j < node->u.entry.col; j++) {

printf(" 0 ");

}

head = head->u.next; // 다음줄로

printf("\n");

}

printf("\n\n");

return ;

}

void merase(matrix\_pointer\* node)

{

// 비어있는 노드라면 삭제처리

if((\*node)==NULL){

printf("Already NULL\n");

return ;

}

int i, num\_heads;

matrix\_pointer x, y, head = (\*node)->right;

for (i = 0; i < (\*node)->u.entry.row; i++) { //row 기준으로 돌면서 삭제

y = head->right;

while (y != head) {

x = y; y = y->right; free(x);

}

x = head; head = head->u.next; free(x); //헤드노드까지 삭제

}

// 남아있는 헤드 삭제

y = head;

while (y != \*node) {

x = y; y = y->u.next; free(x);

}

free(\*node); \*node = NULL;

printf("Erase Compelete!\n");

return;

}

주석으로 코드를 자세하게 설명했습니다. 알고리즘적인 측면에서 설명을 덧붙이겠습니다.

이번 과제는 두 개의 행렬을 .txt파일로부터 입력을 받아 mread함수를 통해 링크드 리스트 형태로 자료구조를 만들고 mwrtie 와 merase를 통해 자료구조의 출력과 삭제를 해보고 확인하는 것이 목적입니다. 굳이 mread처럼 복잡하게 할 필요는 없어보이지만, 수업 시간에 배운 점들을 활용하기 위해서 수업에서의 자료구조 형태를 그대로 가져왔습니다. dense한 matrix일 경우에는 배열로 해도 연산이나 다른 알고리즘을 돌리는데 시간 측면에서 손해를 보지는 않겠지만, sparse한 경우이렇게 linked list 형태로 짜주어야 더 효율적이기 때문에 이런 방식을 채택했습니다.

mread함수에서는 tag의 value를 entry or head로 지정해주어 노드가 어떤 노드를 저장하는 형태인지를 먼저 명시합니다. head노드일 경우에는 head로 해주어 뼈대를 만들고 entry일 경우에는 row, col, value 정보를 넣고 포인터를 연결해주어 실제 matrix의 정보들을 담습니다.

이를 토대로 mwrite와 merase를 진행합니다. 강의 자료에서 몇 가지 수정한 점들이 있는데 일단, mwrite같은 경우 parameter로 들어오는 노드가 null일 경우 에러가 발생하여 erase직후의 write를 진행하지 못했습니다. 따라서 null일 경우를 처리해주어 삭제 직후에도 입력된 노드가 비어있다는 상태를 출력하게끔 해주었습니다, merase 함수 같은 경우는 \*node ==NULL인 경우에 문제 발생하여 예외처리를 해주었고 그 이외에는 깔끔하게 문제없이 작동하기에 Erase Complete 라는 구문 정도만 함수 완료 확인을 위해 추가해주었습니다.