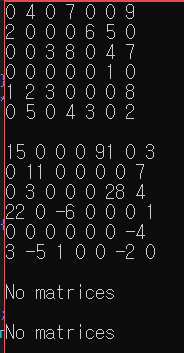
자료구조 과제 7

* Sparse Matrix List –

20151523

김동현

- 실행결과



- 코드 및 알고리즘 분석

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

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;

struct entry\_node entry;

} u;

} matrix\_node;

matrix\_pointer hdnode[MAX\_SIZE];

matrix\_pointer mread(FILE\*);

void mwrite(matrix\_pointer);

void merase(matrix\_pointer\*);

: 필요한 구조체 타입 선언과 전역변수, 함수 선언을 해준다. 구조체 matrix\_node에서 down은 같은 열에서 0이 아닌 다음항을 연결하는 포인터, right는 같은 행에서 0이 아닌 다음 항을 연결하는 포인터, 그리고 tag에 따라 헤드 노드라면 next를 통해 다음 헤더 노드를 가르키고, element 노드라면 entry를 통해 값을 갖는다. 배열 변수 hdnode[i]는 행I, 열I 에 대한 헤더 노드를 가리키는 역할로 사용되는 포인터 변수이다.

void main() {

matrix\_pointer a, b;

FILE \*fa, \*fb;

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

if (fa == NULL) {

printf("A file open error!\n");

return 0;

}

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

if (fb == NULL) {

printf("B file open error!\n");

return 0;

}

a = mread(fa);

b = mread(fb);

mwrite(a);

mwrite(b);

merase(&a);

merase(&b);

mwrite(a);

mwrite(b);

fclose(fa);

fclose(fb);

return 0;

}

: A.txt와 B.txt를 각각 파일포인터 fa, fb에 지정해서 열어준다. 그리고 mread, mwrite, merase의 각 함수를 main에서 호출하고 수행한다.

matrix\_pointer mread(FILE \*fp) : sparse matrix 파일을 읽어서 링크드 리스트 형식으로 저장하는 함수

{

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

int row, col, value, current\_row;

matrix\_pointer temp, last, node;

fscanf(fp, "%d %d", &num\_rows, &num\_cols);

num\_heads = (num\_rows > num\_cols) ? num\_rows : num\_cols;

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

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

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

: 첫 노드에 matrix의 row, col의 number의 정보를 저장해준다. 그리고 헤더 노드의 개수는 행과 열 중 더 큰 값을 따른다.

if (!num\_heads

node->right = node;

else {

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

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

hdnode[i] = temp;

hdnode[i]->tag = head;

hdnode[i]->right = temp;

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

}

: 헤더 노드들의 생성과 연결을 해준다.

current\_row = 0;

last = hdnode[0];

for (row = 0; row < num\_rows; row++) {

for (col = 0; col < num\_cols; col++) {

fscanf(fp, "%d", &value);

if (value) {

if (row > current\_row)

{

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

current\_row = row;

last = hdnode[row];

}

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

}

}

}

: 값 들을 차례대로 읽어들여서 0이 아닌 경우 그 값들을 새로운 node를 생성하여 해당하는 헤더 노드에 따라 저장해준다. 저장할 때 row, col의 정보도 함께 저장하고 last를 계속 새로 들어온 node로 초기화해주어서 반복적으로 연결되게끔 한다.

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

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

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

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;

}

: 헤더 노드들을 circular하게 다 연결해주고 첫 노드를 반환한다.

void mwrite(matrix\_pointer node) : Sparse Matrix를 출력하는 함수

{

matrix\_pointer tmp;

int last;

if (node != NULL)

{ matrix\_pointer head = node->right;

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

last = 0;

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

for (int j = last; j < tmp->u.entry.col; j++)

printf("0 ");

: 해당 노드의 값 출력에 앞서 그 빈 공간만큼 0을 출력해야 하므로 col개수 만큼 0을 출력해준다.

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

last = tmp->u.entry.col + 1;

}

: 그리고 해당 값을 출력해주면 행렬의 형식으로 출력이 된다. 그리고 last를 그 다음 col로 넘긴다.

for (int j = last; j < node->u.entry.col; j++)

printf("0");

head = head->u.next;

printf("\n");

}

}

: 그 다음 마지막 node 출력 이후, 비어 있는 공간에 0을 채워준다.

else

printf("No matrices\n");

printf("\n");

}

: 만일 node가 NULL이 들어와 있다면 erase되었으므로 에러 메세지를 출력한다.

void merase(matrix\_pointer\* node) : sparse matrix를 삭제하는 함수

{

int i;

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

for (i = 0; i < (\*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);

}

: row 순서로 각 헤더와 엔트리 노드들의 메모리 할당을 해제해준다.

y = head;

while (y != \*node) {

x = y;

y = y->u.next;

free(x);

}

free(\*node);

\*node = NULL;

}

: 그리고 마지막 노드와 첫 노드를 해제해주고 끝난다.