

자료구조 2차과제 report

정보대학 컴퓨터학과

2015410090 이소라

1. 구현 환경

OS – Windows 10

TOOL – Microsoft Visual Studio 2017

2. 프로그램 사용 방법 및 실행 화면

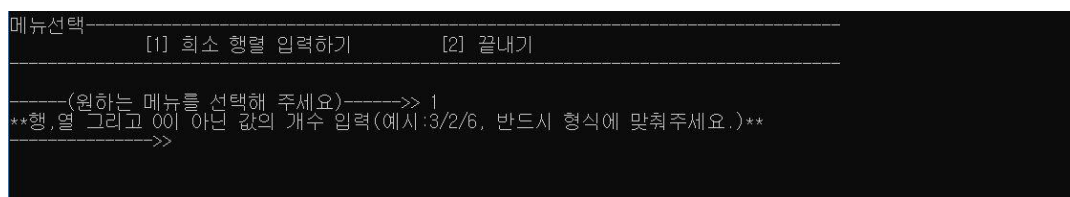
START



```
*****
                      최소행렬의 각종 연산                      자료구조 2차 Assignment
                                                                컴퓨터학과 이소라
*****
메뉴선택-----
                [1] 최소 행렬 입력하기      [2] 끝내기
-----
(원하는 메뉴를 선택해 주세요)----->> 1
```

처음 실행 시 다음과 같은 메뉴 선택 화면이 출력됩니다. 1과 2가 아닌 다른 값이 입력될 경우 "올바른 값을 다시 입력해 주세요."라는 문구가 뜹니다.

INPUT



```
*****
                      최소행렬의 각종 연산                      자료구조 2차 Assignment
                                                                컴퓨터학과 이소라
*****
메뉴선택-----
                [1] 최소 행렬 입력하기      [2] 끝내기
-----
(원하는 메뉴를 선택해 주세요)----->> 1
**행, 열 그리고 0이 아닌 값의 개수 입력(예시:3/2/6, 반드시 형식에 맞춰주세요.)**
----->> 3/2/6
```

1을 선택하면 행렬의 정보를 입력 받는 부분이 출력됩니다. 형식에 맞춰 출력하지 않으면 비정상적으로 작동합니다.

첫번째 행렬 입력

```
**행,열 그리고 0이 아닌 값의 개수 입력(예시:3/2/6, 반드시 형식에 맞춰주세요.**)
----->>5/4/6
**순서대로 행,열,값을 입력(예시:0/0/6, 반드시 형식과 순서를 지켜주세요.**)
1번째 원소----->> 0/0/2
2번째 원소----->> 1/0/4
3번째 원소----->> 1/3/3
4번째 원소----->> 3/0/8
5번째 원소----->> 3/3/1
6번째 원소----->> 4/2/6
```

입력하신 행렬 <행의수 = 5, 열의수 = 4, 인자수 = 6> **

입력하신(또는 연산) 행렬의 행과 열 그리고 그 값----->>

0	0	2
1	0	4
1	3	3
3	0	8
3	3	1
4	2	6

두번째 메뉴선택-----

[1] 두번째 최소 행렬 입력(덧셈, 곱셈) [2] 첫번째 입력 행렬 Transpose [3] 끝내기

-----<원하는 메뉴를 선택해 주세요>----->>

행렬을 입력하면 mwrite 함수가 행렬의 정보를 출력합니다. 그리고 다음 메뉴 선택 메시지가 출력됩니다.

Transpose

두번째 메뉴선택-----

[1] 두번째 최소 행렬 입력(덧셈, 곱셈) [2] 첫번째 입력 행렬 Transpose [3] 끝내기

-----<원하는 메뉴를 선택해 주세요>----->> 2

입력하신 행렬 <행의수 = 4, 열의수 = 5, 인자수 = 6> **

입력하신(또는 연산) 행렬의 행과 열 그리고 그 값----->>

0	0	2
0	1	4
0	3	8
2	4	6
3	1	3
3	3	1

(최소 행렬 연산이 종료되었습니다. 0을 입력하면 종료됩니다.)----->>

두번째 메뉴선택에서 2를 입력하면 처음 입력한 행렬의 transpose된 행렬이 출력됩니다.

두번째 행렬 입력

```

두번째 메뉴선택----->>
[1] 두번째 회소 행렬 입력(덧셈, 곱셈) [2] 첫번째 입력 행렬 Transpose [3] 끝내기

----->> 1
**행,열 그리고 0이 아닌 값의 개수 입력(예시:3/2/6, 반드시 형식에 맞춰주세요.)**
>>5/4/8
**순서대로 행,열,값을 입력(예시:0/0/6, 반드시 형식과 순서를 지켜주세요.)**
1번째 원소----->> 0/0/3
2번째 원소----->> 0/1/5
3번째 원소----->> 1/0/4
4번째 원소----->> 1/1/7
5번째 원소----->> 1/3/2
6번째 원소----->> 3/0/8
7번째 원소----->> 4/1/7
8번째 원소----->> 4/3/3

입력하신 행렬 <행의수 = 5, 열의수 = 4, 인자수 = 8> **

입력하신(또는 연산) 행렬의 행과 열 그리고 그 값----->>

0 0 3
0 1 5
1 0 4
1 1 7
1 3 2
3 0 8
4 1 7
4 3 3

세번째 메뉴선택----->>
[1] 두 행렬 간의 덧셈 연산 [2] 두 행렬 간의 곱셈 연산 [3] 끝내기

----->>

```

두번째 메뉴선택에서 1을 입력하면 두번째 행렬을 입력할 수 있습니다. 첫번째 행렬과 마찬가지로 행렬의 정보가 출력됩니다.

ADD

```

세번째 메뉴선택----->>
[1] 두 행렬 간의 덧셈 연산 [2] 두 행렬 간의 곱셈 연산 [3] 끝내기

----->> 1
입력하신 행렬 <행의수 = 5, 열의수 = 4, 인자수 = 10> **

입력하신(또는 연산) 행렬의 행과 열 그리고 그 값----->>

0 0 5
0 1 5
1 0 8
1 1 7
1 3 5
3 0 16
3 3 1
4 1 7
4 2 6
4 3 3

(회소 행렬 연산이 종료되었습니다. 0을 입력하면 종료됩니다.)----->>

```

세번째 메뉴선택에서 1을 입력하면 첫번째 행렬과 두번째 행렬의 합이 출력됩니다.

```

세번째 메뉴선택-----
[1] 두 행렬 간의 덧셈 연산 [2] 두 행렬 간의 곱셈 연산 [3] 끝내기
-----
(원하는 메뉴를 선택해 주세요)----->> 1
***Warning!! 행렬 덧셈 연산 시 각각의 행의 수 또는 열의 수가 일치하지 않으면 연산이 불가능합니다!
<종료 후 다시 시작해주세요: 0입력>----->>

```

이때 연산이 불가능한 경우 이러한 메시지가 출력됩니다.

MULTIPLY

```

두번째 메뉴선택-----
[1] 두번째 희소 행렬 입력(덧셈, 곱셈) [2] 첫번째 입력 행렬 Transpose [3] 끝내기
-----
(원하는 메뉴를 선택해 주세요)----->> 1
**행,열 그리고 0이 아닌 값의 개수 입력(예시:3/2/6, 반드시 형식에 맞춰주세요.)**
-->>4/4/7
**순서대로 행,열,값을 입력(예시:0/0/6, 반드시 형식과 순서를 지켜주세요.)**

1번째 원소----->> 0/0/4
2번째 원소----->> 0/3/8
3번째 원소----->> 1/1/5
4번째 원소----->> 1/3/7
5번째 원소----->> 2/1/3
6번째 원소----->> 2/2/4
7번째 원소----->> 3/2/7

입력하신 행렬 <행의수 = 4, 열의수 = 4, 인자수 = 7> **
입력하신(또는 연산) 행렬의 행과 열 그리고 그 값----->>

0 0 4
0 3 8
1 1 5
1 3 7
2 1 3
2 2 4
3 2 7

```

두번째 메뉴 선택에서 1을 입력한 상황입니다.

```

세번째 메뉴선택-----
[1] 두 행렬 간의 덧셈 연산 [2] 두 행렬 간의 곱셈 연산 [3] 끝내기
-----
(원하는 메뉴를 선택해 주세요)----->> 2
입력하신 행렬 <행의수 = 5, 열의수 = 4, 인자수 = 10> **
입력하신(또는 연산) 행렬의 행과 열 그리고 그 값----->>

0 0 8
0 3 16
1 0 16
1 2 21
1 3 32
3 0 32
3 2 7
3 3 64
4 1 18
4 2 24

(희소 행렬 연산이 종료되었습니다. 0을 입력하면 종료됩니다.)----->>

```

이때 세번째 메뉴선택에서 2를 입력하면 첫번째 행렬과 두번째 행렬의 곱이 출력됩니다.

REMOVE

Merase를 통해 행렬의 저장에 사용했던 메모리들을 모두 free시켜 주었습니다.

3. 코드 설명

Mread, mwrite, merase는 교과서의 코드를 참고하였습니다. 달라진 부분은, hdnod 배열을 이차원 배열로 바꾸어 각각의 index와 하나의 행렬을 매핑하여 저장하였기 때문에 parameter에 hdnod의 index를 추가해 주었습니다.

mtranspose

```
matrixPointer mtranspose(matrixPointer a, matrixPointer c,
    int hdnod_idx_before, int hdnod_idx_after) {

    matrixPointer newNode, temp, last, nodeToCopy, head;

    int i, j, currentRow;
    int numCols = a->u.entry.col;
    int numRows = a->u.entry.row;
    int numTerms = a->u.entry.value;
    int numHeads = (numCols > numRows) ? numCols : numRows;

    attach(&c, numRows, numCols, numTerms, 1);
    int row, col, value;

    if (!numHeads) c->right = c;
    else {
        for (i = 0; i < numHeads; i++) {
            temp = (matrixPointer)malloc(sizeof(matrixNode));
            hdnod[hdnod_idx_after][i] = temp;
            hdnod[hdnod_idx_after][i]->tag = head;
            hdnod[hdnod_idx_after][i]->right = temp;
            hdnod[hdnod_idx_after][i]->u.next = temp;
        } // hdnod 세팅

        currentRow = 0;
        last = hdnod[hdnod_idx_after][0]; // last : currentRow의 마지막 node
        head = hdnod[hdnod_idx_before][0]; // head : currentRow의 header node
        nodeToCopy = head->down; // 원래 행렬에서 copy할 노드
```

```

for (j = 0; j < a->u.entry.col; j++) {
    nodeToCopy = head->down;
    if (nodeToCopy == head) {                // 다시 head로 돌아왔을 때
        head = head->u.next;                // 다음 head로 이동
        nodeToCopy = head->down;            // copy할 노드로 다음 head의 자식 설정
        continue;
    }
    if (head == a) break;
    for (i = 0; i < numTerms; i++) {

        row = nodeToCopy->u.entry.col;
        col = nodeToCopy->u.entry.row;
        value = nodeToCopy->u.entry.value;
        // 값은 그대로, 좌표만 뒤바꿔둬줌

        if (row > currentRow) {
            last->right = hdnode[hdnode_idx_after][currentRow];
            currentRow = row;
            last = hdnode[hdnode_idx_after][row];
        }
        // nodeToCopy의 row값이 currentRow보다 크면
        // currentRow 마무리하고 다음 currentRow로 row 채택

        attach(&temp, row, col, value, 0);
        last->right = temp;
        last = temp;
        hdnode[hdnode_idx_after][col]->u.next->down = temp;
        hdnode[hdnode_idx_after][col]->u.next = temp;
        // currentRow에 새로 입력 들어온 node 만들어서 붙임
    }
}

```

```

    if (nodeToCopy->down == head) {
        break;
    } // nodeToCopy가 currentRow의 마지막 노드면 break
    nodeToCopy = nodeToCopy->down;
    // currentRow의 마지막 노드가 아니면 다음 노드로 진행
}
head = head->u.next;
}

last->right = hdnode[hdnode_idx_after][currentRow];
// 맨 마지막 원소의 right에 그 row의 header node 넣음

for (i = 0; i < numCols; i++) {
    hdnode[hdnode_idx_after][i]->u.next->down = hdnode[hdnode_idx_after][i];
}
for (i = 0; i < numHeads - 1; i++) {
    hdnode[hdnode_idx_after][i]->u.next = hdnode[hdnode_idx_after][i + 1];
}
hdnode[hdnode_idx_after][numHeads - 1]->u.next = c;
c->right = hdnode[hdnode_idx_after][0];
}

return c;
}

```

Mtranspose는 transpose하기 전 행렬, transpose 하기 전 행렬의 hdnode index, 한 후의 hdnode index를 인자로 받는 함수입니다. 원래의 것과 row, col의 값을 바꿔 노드를 만들

었고, 행렬 a를 down 방향으로 확인하고 (return할) 행렬 c에는 right 방향으로 노드를 붙여주었습니다. 새로운 행렬 c를 return해줍니다.

Madd

```
matrixPointer madd(matrixPointer a, matrixPointer b,
    int hdnnode_idx_a, int hdnnode_idx_b) {

    int i, j;
    matrixPointer temp1, temp2, head_a, head_b, head_d, last;
    matrixPointer d, temp;
    int sum, value = 0;
    int currentRow;

    int numCols = a->u.entry.row;
    int numRows = a->u.entry.col;
    int numHeads = (numCols > numRows) ? numCols : numRows;

    attach(&d, a->u.entry.row, a->u.entry.col, 0, 1);

    if (!numHeads) d->right = d;
    else {
        for (i = 0; i < numHeads; i++) {
            temp = (matrixPointer)malloc(sizeof(matrixNode));
            hdnnode[2][i] = temp;
            hdnnode[2][i]->tag = head_;
            hdnnode[2][i]->right = temp;
            hdnnode[2][i]->u.next = temp;
        } // header node 세팅

        head_a = hdnnode[hdnnode_idx_a][0];    temp1 = head_a->right;
        head_b = hdnnode[hdnnode_idx_b][0];    temp2 = head_b->right;
        head_d = hdnnode[2][0];                temp = head_d;
        // 행렬 a와 b 그리고 답을 적을 d의 head 노드를 각각
        // head_a, head_b, head_d에 저장하고
        // 현재 보고 있는 노드를 temp1, temp2, temp로 나타냄

        last = hdnnode[2][0];
        // last에 row 0의 헤더를 넣어 hdnnode[2]에 새 노드 붙일 준비
        d->right = hdnnode[2][0];
        // d와 hdnnode[2] 연결
```



```

currentRow = 0;

for (i = 0; i < a->u.entry.value; i++) {
    for (j = 0; j < b->u.entry.value; j++) {
        if (temp1 == head_a) { // 행렬 a row에서 한바퀴 돌아 header가 되었을 때
            for (; temp2 != head_b; temp2 = temp2->right) {
                attach(&temp, temp2->u.entry.row, temp2->u.entry.col,
                    temp2->u.entry.value, 0); // 남은 b의 요소들을 추가
                last->right = temp;
                last = temp;
                value++; // 전체 element의 수를 알 수 없으므로 하나하나 세어줌
                hdnode[2][temp2->u.entry.col]->u.next->down = temp;
                hdnode[2][temp2->u.entry.col]->u.next = temp;
                // 새로 추가해준 b의 요소들을 끝에 붙임
            }
            last->right = hdnode[2][currentRow];
            // last의 오른쪽에 다시 헤더노드 붙여줌

            head_a = head_a->u.next;
            head_b = head_b->u.next;
            currentRow++;

            temp1 = head_a->right;
            temp2 = head_b->right;
            last = hdnode[2][currentRow];
            head_d = hdnode[2][currentRow];
            temp = head_d;
            // 다음 row로 이동

            if (head_a == a || head_b == b) break;
            // header가 한바퀴 돌아 header of header가 되는 경우 break
        }
    }
}

```

```

if (temp2 == head_b) { // 행렬 b row에서 한바퀴 돌아 header 되었을 때
    for (; temp1 != head_a; temp1 = temp1->right) {
        attach(&temp, temp1->u.entry.row, temp1->u.entry.col,
            temp1->u.entry.value, 0);
        // 남은 a element들을 추가해줌
        value++;
        last->right = temp;
        last = temp;
        hdnode[2][temp1->u.entry.col]->u.next->down = temp;
        hdnode[2][temp1->u.entry.col]->u.next = temp;
        // 이 부분 제외하면 위의 내용과 동일
    }
    last->right = hdnode[2][currentRow];

    head_a = head_a->u.next;
    head_b = head_b->u.next;
    currentRow++;

    temp1 = head_a->right;
    temp2 = head_b->right;

    last = hdnode[2][currentRow];
    head_d = hdnode[2][currentRow];
    temp = head_d;

    if (head_a == a || head_b == b) break;
}

```

```

else {
    int t = 0;
    t = COMPARE(temp1->u.entry.col, temp2->u.entry.col);

    if (t == 1) { // 왼쪽이 더 큼
        attach(&temp->right, temp2->u.entry.row, temp2->u.entry.col,
            temp2->u.entry.value, 0);
        // col이 작은 쪽은 바로 붙여줌
        value++;
        temp = temp->right;
        last->right = temp;
        last = temp;
        temp2 = temp2->right;
        // col 작았던 쪽을 다음 칸으로 이동시킴
        temp->right = hdnode[2][currentRow];
    }
    else if (t == 0) { // 두 값이 같음
        attach(&temp->right, temp1->u.entry.row, temp1->u.entry.col,
            temp1->u.entry.value + temp2->u.entry.value, 0);
        // col이 같으면 value의 합을 붙여줌
        value++;
        temp = temp->right;
        last->right = temp;
        last = temp;
        temp1 = temp1->right; temp2 = temp2->right;
        temp->right = hdnode[2][currentRow];
    }
    else if (t == -1) { // 오른쪽이 더 큼
        attach(&temp->right, temp1->u.entry.row, temp1->u.entry.col,
            temp1->u.entry.value, 0);
        // col이 작은 쪽은 바로 붙여줌
        value++;
        temp = temp->right;
        last->right = temp;
        last = temp;
        temp1 = temp1->right;
        // col 작았던 쪽을 다음 칸으로 이동시킴
        temp->right = hdnode[2][currentRow];
    }
}

```

```

    }
    }
    last->right = hdnode[2][currentRow];
}
for (i = 0; i < numCols; i++) {
    hdnode[2][i]->u.next->down = hdnode[2][i];
}
for (i = 0; i < numHeads - 1; i++) {
    hdnode[2][i]->u.next = hdnode[2][i + 1];
}
hdnode[2][numHeads - 1]->u.next = d;
d->right = hdnode[2][0];

d->u.entry.value = value;
return d;
}
};

```

Madd는 행렬 a와 행렬 b 그리고 각각의 hdnode index를 인자로 받습니다. 이때 답을 넣

을 hdnode index는 2로 고정됩니다. 행렬 a와 b를 row 기준으로 동시에 진행하면서 현재 row의 a의 노드가 더 이상 없을 때는 남은 b의 노드를 (리턴할) 행렬 d에 넣어줍니다. 현재 row의 b의 노드가 더 이상 없을 때는 남은 a의 노드를 d에 넣어줍니다. 같은 row의 노드일 경우 col의 값을 비교하여 col이 작은 쪽의 값을 d에 넣고 다음 칸으로 진행합니다. Col이 같으면 value의 합을 d에 넣어주고 a와 b 모두 다음 칸으로 이동합니다. 마지막에는 d를 return합니다.

Mmult

```
matrixPointer mmult(matrixPointer a, matrixPointer b, int hdnode_idx_a, int hdnode_idx_b)
{
    matrixPointer newB, d;
    newB = (matrixPointer)malloc(sizeof(matrixNode));

    newB = mtranspose(b, newB, 1, 3);
    //newB의 헤더노드 인덱스는 3

    int i, j;
    matrixPointer temp1, temp2, head_a, head_b, head_d, last, temp;

    int sum = 0, value = 0;
    int currentRow, currentCol;

    int numRows = a->u.entry.row;
    int numCols = b->u.entry.col;
    int numHeads = (numCols > numRows) ? numCols : numRows;

    attach(&d, a->u.entry.row, b->u.entry.col, 0, 1);

    if (!numHeads) d->right = d;
    else {
        for (i = 0; i < numHeads; i++) {
            temp = (matrixPointer)malloc(sizeof(matrixNode));
            hdnode[2][i] = temp;
            hdnode[2][i]->tag = head_;
            hdnode[2][i]->right = temp;
            hdnode[2][i]->u.next = temp;
        } // header node setting

        head_a = hdnode[hdnode_idx_a][0]; head_b = hdnode[3][0]; head_d = hdnode[2][0];
        last = hdnode[2][0];
        d->right = hdnode[2][0];
        temp1 = head_a->right; temp2 = head_b->right; temp = head_d;

        currentRow = 0;
        currentCol = 0;
    }
}
```

```

for (i = 0; i < a->u.entry.value;) {
    for (j = 0; j < b->u.entry.value;) {
        if (temp1 == head_a || temp2 == head_b) {
            if (sum) {
                attach(&temp, currentRow, currentCol, sum, 0); value++;
                last->right = temp;
                last = temp;
                last->right = hdnode[2][currentRow];
            } // sum이 0이 아니면 currentRow, currentCol에 붙여줌
            sum = 0; // sum 초기화

            hdnode[2][currentCol]->u.next->down = temp;
            hdnode[2][currentCol]->u.next = temp;
            currentCol++;

            if (currentCol == numCols) { // currentRow에 대해 column 계산 끝
                currentRow++; currentCol = 0;
                head_b = hdnode[3][0];
                if (currentRow == numRows) break;
                // currentRow도 끝났으면 모든 계산 끝
            } else {
                head_a = hdnode[hdnode_idx_a][currentRow];
                head_d = hdnode[2][currentRow];
                last = hdnode[2][currentRow];
            } // currentCol만 끝났으면 새로운 row 계산할 준비
        }
        head_b = hdnode[3][currentCol];
        temp1 = head_a->right; temp2 = head_b->right;
    }
}

```

```

else {
    int t = COMPARE(temp1->u.entry.col, temp2->u.entry.col);
    // a의 row와 b의 col 비교
    if (t == 1) { // a의 col > b의 row
        temp2 = temp2->right;
    }
    else if (t == 0) { // a의 col == b의 row
        sum += temp1->u.entry.value * temp2->u.entry.value;
        temp1 = temp1->right;
        temp2 = temp2->right;
    }
    else if (t == -1) { // a의 col < b의 row
        temp1 = temp1->right;
    }
}

for (i = 0; i < numCols; i++) {
    hdnode[2][i]->u.next->down = hdnode[2][i];
}
for (i = 0; i < numHeads - 1; i++) {
    hdnode[2][i]->u.next = hdnode[2][i + 1];
}
hdnode[2][numHeads - 1]->u.next = d;
d->right = hdnode[2][0];

d->u.entry.value = value;
merase(newB, 3);
return d;
}

return d;
};

```

Mmult는 madd와 마찬가지로 행렬 a, 행렬 b, a의 hdnode index, b의 hdnode index를 인자로 받습니다. 값을 넣기가 용이하도록 b를 transpose한 newB를 사용하여 진행합니다. Madd와 비슷하게 행렬 a와 행렬 newB를 currentRow를 기준으로 동시에 진행합니다. 다만 한쪽의 row가 끝나면 그때 지금까지 모은 sum을 새 노드로 (return할) 행렬 d에 붙여줍니다. 같은 row이면서 같은 col을 가질 때만 sum에 더해줄 수 있습니다 d를 return합니다.