

Data Structure HW4

20231515
컴퓨터공학과
김다은

<목차>

1. 문제 1
2. 문제 2
3. 문제 3

문제 1.

이 문제는 matrix 를 입력 받아 아래와 같은 matrix 자료구조로 관리하는 것이 중요했던 문제였다. 아래 matrix 구조체로 구성된 linked list 를 만들어 관리하는 것이 중요했다.

```
typedef struct matrix* matrix_pointer;

typedef struct matrix {
    int row;
    int col;
    int value;
    matrix_pointer link;
} matrix;
```

프로그램 실행 단계:

1. main 함수에서 readfile 함수를 호출한다.
 - A. input.txt 파일의 내용을 읽어와서 기존 매트릭스의 정보를 linked list 로 저장한다.
2. main 함수에서 mtranspose 함수를 호출한다.
 - A. 기존 matrix 의 원소에서 row 와 col 를 바꾸어 저장한 새로운 matrix 를 생성한다.
 - B. 생성한 matrix 를 parameter 로 가지는 insert_matrix 함수를 호출한다.
 - C. row 와 col 를 기준으로 정렬하여 transpose 된 linked list 에 해당 matrix 를 삽입한다.
3. main 함수에서 wrtiefile 함수를 호출한다.
 - A. output.txt 파일에 transpose 된 matrix 의 내용을 저장한다.
4. main 함수에서 freenode 함수를 호출하여, 기존 매트릭스 정보가 담긴 linked list 와 transpose 된 매트릭스 정보가 담긴 linked list 에 할당된 메모리를 해제한다.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct matrix* matrix_pointer;

typedef struct matrix {
    int row;
    int col;
    int value;
    matrix_pointer link;
} matrix;

matrix_pointer hdnnode = NULL;
matrix_pointer hdnnode_t = NULL;

void insert_matrix(matrix_pointer node) {
    // printf("insert 호출: %d %d %d\n", node->row, node->col, node->value);
    matrix_pointer ptr = hdnnode_t;
    matrix_pointer pre = NULL;
```

```

    if(hdnode_t == NULL) { // linked list의 첫 노드로 삽입되는 경우
        hdnode_t = node;
        return;
    }
    for(; ptr!=NULL; pre=ptr, ptr=ptr->link) {
        if(ptr->row > node->row) {
            if(pre == NULL) { // 맨 앞의 노드로 삽입되는 경우
                node->link = ptr;
                hdnode_t = node;
                return;
            }
            node->link = pre->link;
            pre->link = node;
            return;
        }
        else if(ptr->row == node->row) {
            if(ptr->col > node->col) {
                if(pre == NULL) { // 맨 앞의 노드로 삽입되는 경우
                    node->link = ptr;
                    hdnode_t = node;
                    return;
                }
                node->link = pre->link;
                pre->link = node;
                return;
            }
        }
    }
    if(ptr == NULL) { // linked list의 맨 마지막 노드로 삽입되는 경우
        pre->link = node;
    }
}

matrix_pointer mtranspose(matrix_pointer node) {
    matrix_pointer ptr = node;
    for(; ptr!=NULL; ptr=ptr->link) {
        matrix_pointer newmat = (matrix_pointer)malloc(sizeof(matrix));
        // row와 col 바꾸기
        newmat->row = ptr->col;
        newmat->col = ptr->row;
        newmat->value = ptr->value;
        newmat->link = NULL;
        insert_matrix(newmat);
    }

    return hdnode_t;
}

// 동적 할당된 메모리 해제
void freenode(matrix_pointer head) {
    matrix_pointer ptr = head;
    matrix_pointer del = NULL;
    while(ptr != NULL) {
        del = ptr;
        ptr = ptr->link;
        free(del);
    }
}

```

```

    hnode_t = NULL;
}

void readfile(char* filename, int *row, int *col, int *elem) {
    FILE* fin = fopen("input.txt", "r");
    matrix_pointer ptr = hnode;
    int a, b, c;

    if(fin == NULL) printf("Error");

    fscanf(fin, "%d %d %d", row, col, elem);
    for(int i=0; i<*elem; i++) {
        matrix_pointer newnode = (matrix_pointer)malloc(sizeof(matrix));
        fscanf(fin, "%d %d %d", &a, &b, &c);
        newnode->row = a;
        newnode->col = b;
        newnode->value = c;
        newnode->link = NULL;
        if(hnode == NULL) { // 맨 첫 노드인 경우
            // printf("fir: %d %d %d\n", a,b,c);
            hnode = newnode;
            ptr = hnode;
        }
        else { // 맨 뒤에 newnode 삽입
            // printf("add: %d %d %d\n", a,b,c);
            ptr->link = newnode;
            ptr = ptr->link;
        }
    }

    fclose(fin);
}

void writefile(char* filename, int row, int col, int elem) {
    FILE* fout = fopen(filename, "w");

    fprintf(fout, "%d %d %d\n", col, row, elem);
    for(matrix_pointer ptr = hnode_t; ptr!=NULL; ptr=ptr->link) {
        fprintf(fout, "%d %d %d\n", ptr->row, ptr->col, ptr->value);
    }

    fclose(fout);
}

int main() {
    int row, col, elem;
    readfile("input.txt", &row, &col, &elem);

    // printf("hnode 출력\n");
    // for(matrix_pointer ptr = hnode; ptr!=NULL; ptr=ptr->link) {
    //     printf("%d %d %d\n", ptr->row, ptr->col, ptr->value);
    // }

    mtranspose(hnode);

    writefile("output.txt", row, col, elem);
}

```

```
// 동적할당된 메모리 해제
freenode(hdnode);
freenode(hdnode_t);
}
```

실제 입력 결과:

```
cse20231515@cspro:~/cse3080/HW4/problem1$ gcc -g p1-1.c -o p1
cse20231515@cspro:~/cse3080/HW4/problem1$ ./p1
```

input.txt	output.txt
1 5 6	1 4 6
2 0 2 11	2 0 1 12
3 0 4 6	3 1 1 7
4 1 0 12	4 1 2 -4
5 1 1 7	5 2 0 11
6 2 1 -4	6 3 3 -15
7 3 3 -15	7 4 0 6

문제 2.

a.txt 와 b.txt 로 주어진 a, b polynomial 의 곱을 d.txt 에 저장하는 문제였다. 해당 문제의 경우 a와 b 다항식을 아래 poly 를 원소로 하는 linked list 형태로 저장하여 해결하였다. 또한 각 a 의 원소와 b 의 원소에 대해 곱을 계산한 뒤, 해당 곱의 coefficient 와 exponential 의 값에 따라 d linked list 에 삽입하는 것이 핵심이었다.

```
typedef struct poly* poly_pointer;

typedef struct poly {
    int exp;
    int coef;
    poly_pointer link;
} poly;
```

프로그램 실행 단계:

1. main 함수에서 pread 함수를 호출하여 a.txt 의 내용을 읽어온다.
 - A. a polynomial 의 각 항을 poly 구조체의 형태로 저장한다. 각 poly 들을 linked list 형태로 구성한다.
 - B. a polynomial 의 항의 개수를 return 한다.
2. main 함수에서 pread 함수를 호출하여 b.txt 의 내용을 읽어온다.
 - A. b polynomial 의 각 항을 poly 구조체의 형태로 저장한다. 각 poly 들을 linked list 형태로 구성한다.
 - B. b polynomial 의 항의 개수를 return 한다.
3. main 함수에서 pmult 함수를 호출한다.
 - A. 각 a linked list 와 b linked list 의 원소를 2 중 for 문을 통해 전체 순회한다.
 - B. newpoly 를 만들어 동적 할당한 뒤 현재 계산 중인 a linked list 의 원소와 b linked list 의 원소의 곱의 결과를 newpoly 에 저장한다.
 - C. 이후 pinsert 함수를 호출한다.
 - i. coef 가 0 인 경우 d linked list 에 추가하지 않는다.
 - ii. 해당 exp 를 가지는 항이 d linked list 에 존재하는 경우 해당 항에 newpoly 의 coef 를 더한다.
 - iii. 만약 해당 exp 를 가지는 항이 없는 경우, d linked list 에 새로운 항을 생성하고 저장한다.
4. main 함수에서 pwrite 함수를 호출한다.
 - A. d.txt 에 d linked list 의 내용을 저장한다.

5. main 함수에서 freepoly 함수를 호출하여 a, b, d polynomial 의 정보가 저장된 linked list 에 할당된 메모리를 해제한다.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct poly* poly_pointer;

typedef struct poly {
    int exp;
    int coef;
    poly_pointer link;
} poly;

int pinsert(poly_pointer* head, poly_pointer newpoly) {
    if(newpoly->coef == 0) {
        free(newpoly);
        return -1;
    }

    poly_pointer ptr = *head;
    poly_pointer pre = NULL;
    if(*head == NULL) { // 첫 노드!
        *head = newpoly;
        return 0;
    }
    for(; ptr!=NULL; pre=ptr, ptr=ptr->link) {
        // 해당 exp 인 항이 이미 존재하는 경우
        if(ptr->exp == newpoly->exp) {
            if(ptr->coef + newpoly->coef == 0) {
                free(newpoly);
                // delnode;
                poly_pointer delnode = ptr;
                pre->link = ptr->link;
                ptr = pre;
                free(delnode);
                return -2;
            }
            ptr->coef += newpoly->coef;
            free(newpoly);
            return -1;
        }
        else if(ptr->exp < newpoly->exp) {
```

```

        if(pre == NULL) { // 맨 앞 노드로 삽입
            newpoly->link = (*head)->link;
            *head = newpoly;
        }
        else {
            newpoly->link = pre->link;
            pre->link = newpoly;
        }
        return 0;
    }
}
// 맨 마지막 노드로 삽입
pre->link = newpoly;
return 0;
}

int pmult(poly_pointer* head, poly_pointer a, poly_pointer b) {
    int exp, coef;
    int elem = 0;
    for(poly_pointer ptra = a; ptra!=NULL; ptra=ptra->link) {
        for(poly_pointer ptrb = b; ptrb!=NULL; ptrb=ptrb->link) {
            poly_pointer newpoly = (poly_pointer)malloc(sizeof(poly));
            newpoly->exp = (ptra->exp)+(ptrb->exp);
            newpoly->coef = (ptra->coef)*(ptrb->coef);
            newpoly->link = NULL;
            elem += pinsert(head, newpoly);
        }
    }
    // printf("elem: %d\n", elem);
    return elem;
}

int pread(char* filename, poly_pointer* head, poly_pointer* tail) {
    FILE* fin = fopen(filename, "r");
    int elem;
    int coef, exp;
    fscanf(fin, "%d", &elem);
    for(int i=0; i<elem; i++) {
        poly_pointer newnode = (poly_pointer)malloc(sizeof(poly));
        fscanf(fin, "%d %d", &coef, &exp);
        newnode->coef = coef;
        newnode->exp = exp;
        newnode->link = NULL;
        if(*head == NULL) {
            *head = newnode;

```



```

        *tail = newnode;
    }
    else {
        (*tail)->link = newnode;
        *tail = newnode;
    }
}
fclose(fin);
return elem;
}

void pwrite(poly_pointer head, int elem) {
    FILE* fout = fopen("d.txt", "w");

    poly_pointer ptr = head;
    fprintf(fout, "%d\n", elem);
    // printf("%d\n", elem);
    for(; ptr!=NULL; ptr=ptr->link) {
        // printf("%d %d\n", ptr->coef, ptr->exp); // for test
        fprintf(fout, "%d %d\n", ptr->coef, ptr->exp);
    }

    fclose(fout);
}

void freepoly(poly_pointer head) {
    poly_pointer ptr = head->link;
    poly_pointer delnode = head;
    while(ptr != NULL) {
        delnode = ptr;
        ptr = ptr->link;
        free(delnode);
    }
}

int main() {
    int aelem, belem, delem;
    poly_pointer ahead = NULL;
    poly_pointer atail = NULL;
    poly_pointer bhead = NULL;
    poly_pointer btail = NULL;
    poly_pointer dhead = NULL;

    aelem = pread("a.txt", &ahead, &atail);
    belem = pread("b.txt", &bhead, &btail);

```

```

    delem = aelem*belem+pmult(&dhead, ahead, bhead);

    pwrite(dhead, delem);

    freepoly(ahead);
    freepoly(bhead);
    freepoly(dhead);
}

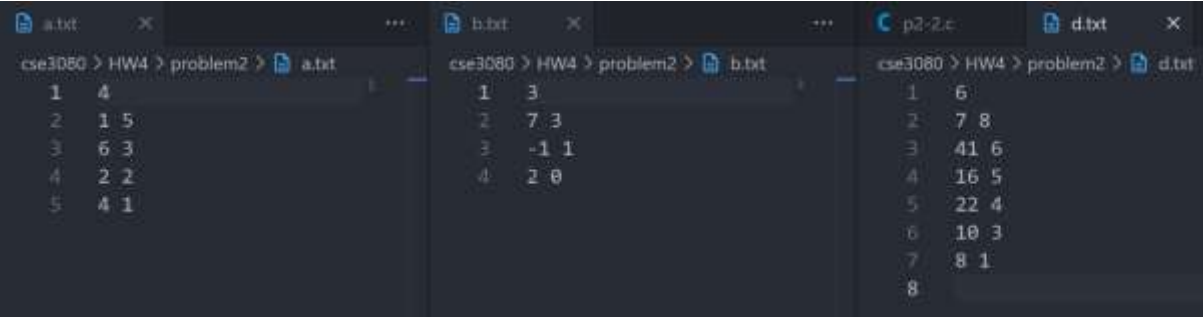
```

실제 입력 결과

```

cse20231515@cspiro:~/cse3080/HW4/problem2$ gcc -g p2-2.c -o p2
^[[A cse20231515@cspiro:~/cse3080/HW4/problem2$ ./p2

```



File	Line	Value 1	Value 2
a.txt	1	4	
	2	1	5
	3	6	3
	4	2	2
	5	4	1
b.txt	1	3	
	2	7	3
	3	-1	1
	4	2	0
d.txt	1	6	
	2	7	8
	3	41	6
	4	16	5
	5	22	4
	6	10	3
	7	8	1
	8		

문제 3.

자료구조 수업 시간에는 array 로 stack 를 만들어 미로 문제를 해결하였다. 본 문제에서는 아래와 같은 노드로 구성된 doubly linked list 를 사용하여 스택을 만들어 사용했다.

```
typedef struct node {
    int x, y, dir;
    struct node* llink;
    struct node* rlink;
} node;
```

미로에서 이동 방향은 다음과 같은 dx, dy 배열을 사용하였다.

```
int dx[] = {-1, -1, 0, 1, 1, 1, 0, -1};
int dy[] = {0, 1, 1, 1, 0, -1, -1, -1};
```

dx[], dy[]를 이용해 가능한 방향의 이동을 고려하기 위해 다음과 같이 코드를 작성할 수 있다.

```
int nx = x + dx[dir];
int ny = y + dy[dir];
```

프로그램의 진행 단계는 다음과 같다.

1. main 함수에서 readfile 함수를 호출하여 maze.txt 를 입력받는다.
2. main 함수에서 searchpath 함수를 호출한다.
 - A. (1, 1) 위치를 방문 처리한다.(mark[1][1] = '1')
 - B. 각 이동 방향에 대해 가능한지 여부를 조사한다.
 - C. 이동 가능한 경우 linked list 에 x, y, ++dir 를 가지는 node 를 push 한다.
 - D. 목적지에 도달할 경우 found = 1 로 만들고 종료한다.
3. main 함수에서 writefile 함수를 호출하여 결과값을 path.txt 에 저장한다.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10

typedef struct node {
    int x, y, dir;
    struct node* llink;
    struct node* rlink;
} node;
```

```

int dx[] = {-1, -1, 0, 1, 1, 1, 0, -1};
int dy[] = {0, 1, 1, 1, 0, -1, -1, -1};

char maze[SIZE][SIZE];
char mark[SIZE][SIZE];
node* head = NULL;
node* tail = NULL;

// stack 의 맨 뒤에 node 추가
void stpush(node* node) {
    // printf("head: %p, %p\n", head, tail);
    // printf("push: %p %d %d %d %p %p\n", node, node->x, node->y,
node->dir, node->llink, node->rlink);

    if(head == NULL) {
        head = node;
        tail = node;
        // printf("test: %p %d %d %d %p %p\n", tail, tail->x, tail->y,
tail->dir, tail->llink, tail->rlink);
    }
    else {
        // printf("test: %p\n", tail);
        // printf("test: %p %d %d %d %p %p\n", tail, tail->x, tail->y,
tail->dir, tail->llink, tail->rlink);
        node->llink = tail;
        tail->rlink = node;
        tail = node;
    }
}

// stack 의 맨 뒤 node 리턴
// 따로 free 해줘야 할 듯
node* stpop() {
    // printf("pop: %p %d %d %d %p %p\n", tail, tail->x, tail->y, tail-
>dir, tail->llink, tail->rlink);
    node* tmp = tail;
    if(head == tail) head = NULL;
    tail = tail->llink;
    return tmp;
}

int isempty() {
    if(head == NULL) return 1;
    else return 0;
}

```

```

}

void readfile(char* filename) {
    FILE* fin = fopen(filename, "r");
    int i = 0;
    while(fscanf(fin, "%s", maze[i++]) != EOF) {};
    fclose(fin);
}

void writefile(node* head, char* filename) {
    FILE* fout = fopen(filename, "w");

    while(head != NULL) {
        fprintf(fout, "%d %d\n", head->x, head->y);
        head = head->rlink;
    }
    fclose(fout);
}

node* newnode(int x, int y, int dir) {
    node* new = (node*)malloc(sizeof(node));
    new->x = x; new->y = y; new->dir = dir;
    new->llink = NULL; new->rlink = NULL;
    return new;
}

void searchpath() {
    // 출발지
    node* current = NULL;
    int found = 0;
    int x, y, dir;

    mark[1][1] = '1';
    stpush(newnode(1, 1, 1));

    while(!found && !isempty()) {
        // for(int i=0; i<SIZE; i++) {
        //     for(int j=0; j<SIZE; j++) {
        //         printf("%c", mark[i][j]);
        //     }
        //     printf("\n");
        // }
        // printf("\n");

        current = stpop();
    }
}

```

```

    x = current->x; y = current->y;
    dir = current->dir;
    while(dir<8 && !found) {
        int nx = x + dx[dir];
        int ny = y + dy[dir];
        // printf("search %d %d %d\n", nx, ny, dir);

        // 목적지 도달
        if(nx == SIZE-1 && ny == SIZE-1) {
            stpush(newnode(x, y, ++dir));
            found = 1;
        }
        // 이동 가능한 경우
        else if(maze[nx][ny] == '0' && mark[nx][ny] == '0'){
            // printf("cur: %d %d %d\n", nx, ny, dir);
            mark[nx][ny] = '1';
            stpush(newnode(x, y, ++dir));
            x = nx; y = ny; dir = 0;
        }
        // 이동 불가능한 경우
        else dir++;
    }
}

int main() {
    // mark 초기화
    for(int i=0; i<SIZE; i++) {
        for(int j=0; j<SIZE; j++) {
            mark[i][j] = '0';
        }
    }
    readfile("maze.txt");
    searchpath();
    writefile(head, "path.txt");
}

```

실제 입력 결과

```

cse20231515@cspro:~/cse3080/HW4/problem3$ gcc -g p3-1.c -o p3
cse20231515@cspro:~/cse3080/HW4/problem3$
cse20231515@cspro:~/cse3080/HW4/problem3$ ./p3

```

cse3080 > HW4 > problem3 > maze.txt

```
1 1111111111
2 1011111011
3 1100010111
4 1000100011
5 1100001111
6 1010010001
7 1101001011
8 1011111001
9 1011000101
10 1111111111
```

cse3080 > HW4 > problem3 > path.txt

```
1 1 1
2 2 2
3 2 3
4 2 4
5 3 5
6 2 6
7 3 7
8 3 6
9 4 5
10 5 6
11 5 7
12 5 8
13 6 7
14 7 8
15 8 8
16
```