C++ 프로그래밍 과제

제출일자: 2023.04.06
경영학과 2019126042 박민서

1. 문제 제기

(1) 다항식 클래스를 만들기 위해 1) 다항식 A, B 2개를 입력 받아 식을 출력하고, 2 두 다항식을 곱셈하여 새로운 다항식 C를 출력하시오.

- 임의의 x값을 입력 받아 다항식 C 에 x를 대입한 결과값을 출력
- 출력된 결과값이 50 미만이면 'A'를, 50 이상이면 'B' 를 출력하는 프로그램
- 입력 받을 다항식의 계수와 지수는 1~10까지의 자연수만 고려한다.
- 계수 및 지수가 1일 경우엔 출력 시 표현하지 않는다.
- 다항식 연산은 연산자 오버로딩을 활용한다.

(2) 행렬을 입력받아 A[n][m] 방식으로 저장하는것이 효율적인지 희소행렬 방식으로 저 장하는 것이 효율적인지 출력하고 전치행렬을 출력하는 프로그램을 구현하시오

- 저장 공간이 동일하다면 아무 방식으로 저장하여도 상관없다.
- 희소행렬 방식으로 저장할 시 행을 기준으로 오름차순 정렬이 되어있어야 한다.
- 행이 같으면 열을 기준으로 오름차순 정렬한다.
- 보고서에 더 효율적인 방법의 기준을 명시한다.

2. 문제 해결 과정

(1) 다항식 클래스를 만들기 위해 1) 다항식 A, B 2개를 입력 받아 식을 출력하고, 2 두 다항식을 곱셈하여 새로운 다항식 C를 출력하시오.

- Polynomial 클래스는 max_degree 변수와 coefs 배열로 구성됨
- max_degree 변수는 다항식의 최고차항의 지수를, coefs 배열은 다항식의 각 항의 계수를 저장

- Polynomial 클래스의 객체를 생성할 때, 계수 배열과 최고차항의 지수를 입력으로 받아 Polynomial(int coefs[], int max_degree) 생성자를 호출함
- eval() 메서드는 다항식을 출력. 계수 배열(coefs)로 표현된 다항식을 최고차항에서 최저차항으로 내림차순으로 반복문을 수행하면서 출력
- eval(int x) 메서드는 다항식의 x에 대한 값을 계산하여 반환
- mul(Polynomial &p) 메서드는 다항식 곱을 반환
- _split(char *tokens[], char s[], const char *sep) 메서드는 문자열 s를 받아서 토큰들로 나눠서 그것들을 tokens 배열로 저장하고, 나눠진 토큰의 개수를 반환
- _coef(int coefs{}, char *tokens[], int num_tokens) 함수는 토큰 리스트로 토큰들을 받고, 계수 배열을 생성. 그리고 계수 배열의 유의미한 사이즈, 즉 최대 차수를 반환

(2) 행렬을 입력받아 A[n][m] 방식으로 저장하는것이 효율적인지 희소행렬 방식으로 저 장하는 것이 효율적인지 출력하고 전치행렬을 출력하는 프로그램을 구현하시오

- input_transpose_matrix(): 입력 함수로서, 행렬의 크기와 각 원소 값을 입력받아 2차원 배열 A에 저장
- _count_non_zeros(): 2차원 배열 A에서 0이 아닌 값의 개수를 세어 반환합니다.
- is_dok_effecient(): 배열 A를 희소 행렬 방식으로 저장하는 것이 더 효율적인지를 판단하여, 저장 방식에 따라 true/false 값을 반환
- print_matrix(): 2차원 배열 A를 순회하여 각 원소 값을 출력
- print_dok_matrix(): 2차원 배열 A를 순회하여 0이 아닌 원소들만 (위치, 값)으로 출력
- A[N_MAX + 1][M_MAX + 1]: 입력된 행렬의 정보를 저장하는 2차원 배열
- R, C: 입력된 행렬의 행, 열 크기를 저장
- non_zeros: 2차원 배열 A에서 0이 아닌 값의 개수를 저장

3. 결과 (소스코드 및 실행화면)

(1) 다항식 클래스를 만들기 위해 다항식 A, B 2개를 입력 받아 식을 출력하고, 두 다항식을 곱셈하여 새로운 다항식 C를 출력하시오.

[소스코드]

```
#include    <cstdio>
```

```cpp
#include    <cstring>   // memset(), memcpy() , strtok()
#include    <cmath>     // pow()
#define     LINESZ      1000
#define     TOKENSZ     100
#define     COEFSZ      30

using namespace std;


class Polynomial {
public:
    int max_degree;         // 지수
    int coefs[COEFSZ + 1];  // 계수 배열

    // 인자로 받은 coefs 배열을 복사하여 새로운 Polynomial 객체를 만든다.
    Polynomial(int coefs[], int max_degree) {
        this->max_degree = max_degree;
        memcpy(this->coefs, coefs, sizeof(int) * (max_degree + 1));
    }

    // f(x) = x^2 + 2x


    void eval() {
        for (int i = max_degree; i >= 0; i--) {
            if (coefs[i] == 0)      continue; // 계수가 0 이 아니면, 현재 차수가
최고차항보다 작으면 더하기 기호(+)를 출력

            if (i < max_degree)     printf(" + ");

            if (coefs[i] != 1)      printf("%d", coefs[i]); // 1 이 아니면 계수를 출력

            if (i == 1)             printf("x"); // 현재 차수가 1 이면 변수 x 를 출력
            else if (i > 1)         printf("x^%d", i); // 현재 차수가 1 보다 크면 변수
x 의 현재 차수를 지수로 하는 식을 출력합
        }
    }
    void operator()() { return this->eval(); }          // 연산자 오버로딩

    // f(1) = 1^2 + 2*1 = 3
    int eval(int x) {
        int sum = 0;
        for (int i = max_degree; i >= 0; i--)
            sum += coefs[i] * pow(x, i);   // x^i
        return sum;
    }
    int operator()(int x) { return this->eval(x); }     // 연산자 오버로딩

    // Polynominal(this) * Polynomial(p)
```

```
    Polynomial mul(Polynomial &p) { // class Polynomail 의 메서드 mul -> 다항식 곱을
반환
        int max_degree = this->max_degree * p.max_degree;
        int coefs[max_degree + 1];
        memset(coefs, 0, sizeof(coefs));

        for (int i = this->max_degree; i >= 0; i--) {
            for (int j = p.max_degree; j >= 0; j--)
                coefs[i + j] += this->coefs[i] * p.coefs[j];
        }
        return Polynomial(coefs, max_degree);
    }
    Polynomial operator*(Polynomial &p) { return this->mul(p); }
};


// "1 2 3 4" -> ["1", "2", "3", "4"]
// 문자열 s 를 받아서 토큰들로 나눠서 그것들을 tokens 라는 리스트로 저장하고, 나눠진 토큰의
개수를 반환한다.
int _split(char *tokens[], char s[], const char *sep)
{
    int num_tokens = 0; // 나눠진 토큰의 개수
    char *p; // 자른 문자열

    p = strtok(s, sep); // strtok 함수(문자열, 구분자) / 문자열을 임시로 저장하기 위한
포인터 변수 p
    if (p != NULL) {
        tokens[num_tokens++] = p;

        while ((p = strtok(NULL, sep)) != NULL) // s 문자열에서 이전에 잘려지지 않은
문자열의 다음 위치에서부터 구분자 sep 을 찾아서 해당 위치에서부터 자른 문자열의 포인터를 반환
            tokens[num_tokens++] = p; // strtok() 함수가 반환하는 포인터를 tokens 배열에
저장하고, num_tokens 변수를 증가시켜 나눠진 토큰의 개수를 세는 과정을 반복
    }

    return num_tokens; // strtok() 함수가 더 이상 자를 문자열이 없을 때, NULL 을
반환하므로 이를 체크하여 반복문을 종료하고, 나눠진 토큰의 개수를 반환
}

// ["1", "2", "3", "4"] -> Coef([0, 0, 1, 0, 3]) (= 3x^4 + x^2)
// tokens 리스트로 토큰들을 받고, 계수 배열을 생성한다. 그리고 계수 배열의 유의미한 사이즈, 즉
최대 차수를 반환한다. (다항식의 문자열 표현에서 계수와 차수를 추출하고 이를 배열로 저장)
int _make_coefs(int coefs[], char *tokens[], int num_tokens)
{
    int max_degree = 0; // 다항식의 최고 차수

    int degree, coef; // 차수, 개수
    for (int i = 0; i < num_tokens; i += 2) {
```

```
        coef = atoi(tokens[i]), degree = atoi(tokens[i + 1]); // atoi = char to int = 문자열을 정수 타입으로
        coefs[degree] = coef;

        if (degree > max_degree)    max_degree = degree;
    }

    return max_degree;
}

Polynomial input_polynomial() // class Polynomal의 메서드 input_polynomial -> 다항식 입력 시 계수, 지수 순서로 입력 받음
{
    // 한 줄 받기 : "1 2 3 4"
    char line[LINESZ];
    fgets(line, LINESZ, stdin);     // 문자열 입력함수 fgets(저장할 배열, 행 최대 문자수, 포인터 이름)를 사용하면 공백문자가 포함되어 있는 문장을 입력받아 저장할 수 있다.  , 행단위로 취득한 값은 문자열 배열로 저장
                                    // 즉 stdin(표준입력)으로 부터 문자열을 입력받아 배열 line에 저장하되, LINESZ의 길이 만큼만 저장
    // "1 2 3 4" -> ["1", "2", "3", "4"]'
    int num_tokens; // 문자열인 한 줄을 공백을 기준으로 자르기 위한 변수 num_tokens
    char *tokens[TOKENSZ]; // 포인터 tokens
    num_tokens = _split(tokens, line, " "); // 공백을 기준으로 자른 문자열 저장

    // ["1", "2", "3", "4"] -> Coef([0, 0, 1, 0, 3]) (= 3x^4 + x^2)
    int max_degree;
    int coefs[COEFSZ + 1]; // 계수를 저장하기 | 위한 정수형 계수 배열
    max_degree = _make_coefs(coefs, tokens, num_tokens); // 최고 차수 변수 max_degree

    return Polynomial(coefs, max_degree);
}


int main()
{
    // 2 1 1 2  (계수, 지수)
    Polynomial A = input_polynomial(); // 다항식 A

    // 1 1 3 2
    Polynomial B = input_polynomial(); // 다항식 B

    // x = 1
    int x;
    printf("x = "), scanf("%d", &x);

    // A : 2x + x^2
    printf("A : "), A(), printf("\n");
    // B : x + 3x^2
```

```
    printf("B : "), B(), printf("\n");

    // C : 3x^4 + 7x^3 + 2x^2
    Polynomial C = A * B;           // A.mul(B);
    printf("C : "), C(), printf("\n");

    // 12
    int c = C(x);
    printf("%d\n", c);
    // A
    printf("%s\n", (c < 50) ? "A" : "B");
}
```

[실행화면]

```
→  minseo-workspace cd "/Users/minseopark/minseo-workspace/" && g++ cloneprob1.cpp -o cl
oneprob1 && "/Users/minseopark/minseo-workspace/"cloneprob1
2 1 1 2
1 1 3 2
x = 1
A : x^2 + 2x
B : 3x^2 + x
C : 3x^4 + 7x^3 + 2x^2
12
A
```

(2) 행렬을 입력받아 A[n][m] 방식으로 저장하는것이 효율적인지 희소행렬 방식으로 저 장하는 것이 효율적인지 출력하고 전치행렬을 출력하는 프로그램을 구현하시오

[소스코드]

```
// 배열을 이용하여 행렬을 표현하는 2 가지 방법
// 1. 2 차원 배열을 이용해 전체요소를 저장하는 방법 -> 연산구현 간단하지만 메모리 공간 낭비
// 2. 0 이 아닌 요소들만 저장하는 방법 (위치, 요소) -> 메모리 공간 절약되지만, 연산구현 어려움

#include <cstdio>
#define N_MAX 100 // 행 크기 100
#define M_MAX 100 // 열 크기 100

int A[N_MAX + 1][M_MAX + 1], R, C; // N_MAX + 1, M_MAX + 1 인 이유-> sparce matrix 의
0 번째 요소는 row, col, value 에 대한 정보를 담기 때문


/** input matrix */ // 행렬 입력
void input_transpose_matrix()
{
    scanf("%d %d", &R, &C); // 공백을 기준으로 행 개수 R, 열 개수 C
```

```
    // 전치행렬이므로 열행
    for (int x = 1; x <= C; x++) {
        for (int y = 1; y <= R; y++) {
            scanf("%d", &A[y][x]);  // 행렬 내 원소 채우기
        }
    }
}



// sparcematrix 희소행렬  배열 내에서 0 이 아닌 개수 세기
int _count_non_zeros()
{
    int count = 0;
     // 전치행렬이므로 열행
    for (int y = 1; y <= R; y++) {
        for (int x = 1; x <= C; x++) {
            if (A[y][x] != 0) { // 행렬
                count += 1; // count 변수에 0 이 아닌 값의 개수를 세어 담는다.
            }
        }
    }
    return count; // 0 이 아닌 개수 리턴
}

// A[n][m] 과 희소행렬 중 효율 판단 bool (0 or 1)
bool is_dok_effecient()
{
    int non_zeros = _count_non_zeros(); // 변수 non_zeros 에 0 이 아닌 개수 저장
    if (3 * non_zeros < R * C) { // 0 이 아닌 행, 열, 값의 공간을 가지므로 공간 복잡도 3*t
        return true; // 1
    } else {
        return false; // 0
    }
}

/** print matrix */
// A[n][m] 저장방식 (1)이 유리 할 때 전치행렬
void print_matrix()
{
    for (int y = 1; y <= R; y++) {
        for (int x = 1; x <= C; x++) {
            printf("%d ", A[y][x]);
        }
        printf("\n");
    }
}
// 희소 행렬 저장 방식(2)이 유리할 때 전치행렬
void print_dok_matrix()
```

```
{
    for (int y = 1; y <= R; y++) { //열행
        for (int x = 1; x <= C; x++) {
            if (A[y][x] != 0) {
                printf("%d %d %d\n", y, x, A[y][x]);
            }
        }
    }
}

int main()
{
    /** input matrix */
    input_transpose_matrix(); // 행렬 입력

    /** metric -> print matrix */
    if (is_dok_effecient()) {
        printf("2\n"); // return 1 -> 희소배열 저장 방식이 유리하므로 2 출력
        print_dok_matrix(); // 희소 행렬 저장 방식(2)이 유리할 때 전치행렬
    }
    else {
        printf("1\n"); // return 0 -> A[n][m] 저장방식이 유리하므로 1 출력
        print_matrix();
    }
}
```

[실행 화면]

```
➜  minseo-workspace cd "/Users/minseopark/minseo-workspace/" && g++ cloneprob2.cpp -o cl
oneprob2 && "/Users/minseopark/minseo-workspace/"cloneprob2
5 5
0 1 0 2 0
3 0 4 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
2
1 2 3
2 1 1
3 2 4
4 1 2
➜  minseo-workspace
➜  minseo-workspace cd "/Users/minseopark/minseo-workspace/" && g++ cloneprob2.cpp -o cl
oneprob2 && "/Users/minseopark/minseo-workspace/"cloneprob2
5 2
1 2
1 2
1 2
1 2
1 2
1
1 2
2 1
1 2
2 1
1 2
```