4주차 3차시 다항식, 행렬의 순차 자료구조 표현

【학습목표】

1. 선형리스트의 응용 방법에 대하여 설명할 수 있다.
2. 다항식의 1,2차원배열을 이용한 구현을 구분할 수 있다.

학습내용1 : 다항식의 1차원 배열을 이용한 구현

1. 다항식이란?

* aXe 형식의 항들의 합으로 구성된 식
- a : 계수(coefficient)
- X : 변수(variable)
- e : 지수(exponent)

* 다항식의 특징
- 지수에 따라 내림차순으로 항을 나열
- 다항식의 차수 : 가장 큰 지수
- 다항식 항의 최대 개수 = (차수 +1)개

2. 다항식의 추상 자료형

ADT Polynomial
데이터 : 지수(e_i)-계수(a_i)의 순서쌍 <e_i, a_i>의 집합으로 표현된
다항식 $p(x) = a_0x^{e0} + a_1x^{e1} + \dots + a_nx^{en}$. ($e_i$는 음이 아닌 정수)

연산 : p,p1,p2 ∈ Polynomial; a ∈ Coefficient; e ∈ Exponent;
// p,p1,p2는 다항식이고, a는 계수, e는 지수를 나타낸다.

zeroP() ::= **return** polynomial p(x)=0;
// 다항식p(x)를 0으로 만드는 연산

isZeroP(p) ::= **if** (p) **then false;**
else return true;
// 다항식p가 0인지 아닌지 검사하여 0이면 true를 반환하는 연산

coef(p,e) ::= **if** (<e,a> ∈ p) **then return** a;
else return 0;
// 다항식p에서 지수가 e인 항의 계수a를 구하는 연산

maxExp(p) ::= **return** max(p.Exponent);
// 다항식p에서 최대 지수를 구하는 연산

```
addTerm(p,a,e) ::= if (e ∈ p.Exponent) then return error;
                   else return p after inserting the term <e,a>;
    // 다항식p에 지수가 e인 항이 없는 경우에 새로운 항<e,a>를 추가하는 연산

delTerm(p,e) ::= if (e ∈ p.Exponent) then return p after removing the term <e,a>;
                 else return error;
    // 다항식p에서 지수가 e인 항<e,a>를 삭제하는 연산

multTerm(p,a,e) ::= return (p * ax^e);
    // 다항식p의 모든 항에 ax^e항을 곱하는 연산

addPoly(p1,p2) ::= return (p1 + p2);
    // 두 다항식p1과 p2의 합을 구하는 연산

multPoly(p1,p2) ::= return (p1 * p2);
    // 두 다항식p1과 p2의 곱을 구하는 연산
End Polynomial
```

3. 다항식의 표현

* 각 항의 지수와 계수의 쌍에 대한 선형리스트
- 예) $A(x)=4x^3+3x^2+2 \Rightarrow p1= (3,4,\ 2,3,\ 0,2)$

* 1차원 배열을 이용한 순차 자료구조 표현
- 차수가 n인 다항식을 (n+1)개의 원소를 가지는 1차원 배열로 표현
- 배열 인덱스 i : 지수(n-1)을 의미
- 배열 인덱스 I의 원소 : 지수(n-1)항의 계수
    - 다항식에 포함되지 않은 지수의 항에 대한 원소에 0을 저장

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x^1 + a_0 x^0$$

인덱스	[0]	[1]	···	[n−1]	[n]
P	a_n	a_{n-1}	···	a_1	a_0

[그림 4-13] n차 다항식 P(x)의 순차 자료구조 표현

* $A(x)=4x^3+3x^2+2$ 의 자료구조 표현

	[0]	[1]	[2]	[3]
A	4	3	0	2

[그림 4-14] A(x)의 순차 자료구조 표현

- 희소 다항식에 대한 1차원 배열 저장
    - $B(x)=3x^{1000} + x + 4$

	[0]	[1]	[2]	[3]	···	[997]	[998]	[999]	[1000]
B	3	0	0	0	···	0	0	1	4

[그림 4-15] B(x)의 순차 자료구조 표현

- 차수가 1000이므로 크기가 1001인 배열을 사용하는데 항이 단 3개로 배열의 원소 중에서 3개만이 사용되어 메모리 공간 낭비

학습내용2 : 다항식의 2차원 배열을 이용한 구현

1. 2차원 배열을 이용한 순차 자료구조 표현

* 다항식의 각 항에 대한 <지수, 차수>쌍을 2차원 배열에 저장
- 2차원 배열의 행의 개수 : 다항식의 항의 개수
- 2차원 배열의 열의 개수 : 2
- B(x)=$3x^{1000}$ + x + 4 의 2차원 열 표현
- 1차원 배열을 사용하는 방법보다 메모리 사용 공간량 감소

	[0]	[1]	
[0]	1000	3	☞ $3x^{1000}$
[1]	1	1	☞ x
[2]	0	4	☞ 4

[그림 4-16] 희소 다항식 B(x)의 순차 자료구조 표현

2. 다항식의 덧셈 알고리즘 : A(x)+B(x) = C(x)

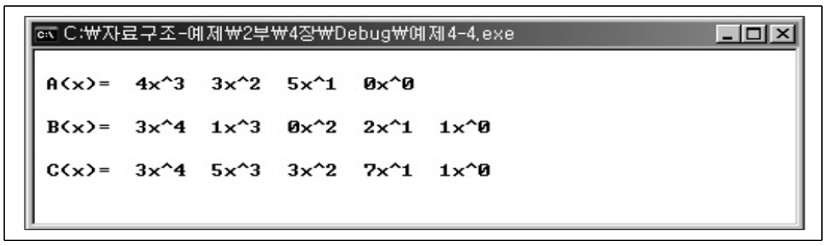
```
addPoly(A, B)
// 주어진 두 다항식 A와 B를 더하여 결과 다항식 C를 반환하는 알고리즘
  C ← zeroP();
  while (not isZeroP(A) and not isZeroP(B)) do {
    case {
        maxExp(A) < maxExp(B) :
              C ← addTerm(C, coef(B, maxExp(B)), maxExp(B));
              B ← delTerm(B, maxExp(B));
        maxExp(A) = maxExp(B) :
              sum ← coef(A, maxExp(A)) + coef(B, maxExp(B));
              if (sum≠0) then C ← addTerm(C, sum, maxExp(A));
              A ← delTerm(A, maxExp(A));
              B ← delTerm(B, maxExp(B));
        maxExp(A) > maxExp(B) :
              C ← addTerm(C, coef(A, maxExp(A)), maxExp(A));
              A ← delTerm(A, maxExp(A));
        }
  }
  if (not isZeroP(A)) then A의 나머지 항들을 C에 복사;
  else if (not isZeroP(B)) then B의 나머지 항들을 C에 복사;
  return C;
End  addPoly()
```

3. 다항식의 덧셈 C 프로그램

```
01  #include <stdio.h>
02  #define MAX(a,b) ((a>b)?a:b)
03  #define MAX_DEGREE 50
04
05  typedef struct{
06      int degree;
07      float coef[MAX_DEGREE];
08  } polynomial;
09
10  polynomial addPoly(polynomial A, polynomial B)
11  {
12      polynomial C;
13      int A_index=0, B_index=0, C_index=0;
14      int A_degree=A.degree, B_degree=B.degree;
15      C.degree=MAX(A.degree, B.degree);
16
17      while(A_index<=A.degree && B_index<=B.degree){
18          if(A_degree > B_degree){
19              C.coef[C_index++] = A.coef[A_index++];
20              A_degree--;
21          }
22          else if(A_degree == B_degree){
23              C.coef[C_index++] = A.coef[A_index++]+B.coef[B_index++];
24              A_degree--;
25               B_degree--;
26          }
```

```
27          else{
28              C.coef[C_index++] = B.coef[B_index++];
29              B_degree--;
30          }
31      }
32      return C;
33  }
34
35  void printPoly(polynomial P)
36  {
37      int i, degree;
38      degree=P.degree;
39
40      for(i=0; i<=P.degree; i++)
41          printf("%3.0fx^%d", P.coef[i], degree--);
42      printf("\n");
43  }
44
45
46  void main()
47  {
48      polynomial A={3, {4,3,5,0}};
49      polynomial B={4, {3,1,0,2,1}};
50
51      polynomial C;
52      C= addPoly(A,B);
53
54      printf("\n A(x)=");  printPoly(A);
55      printf("\n B(x)=");  printPoly(B);
56      printf("\n C(x)=");  printPoly(C);
57
58      getchar();
59  }
```

* 실행 결과



학습내용3 : 행렬의 순차자료구조 표현

1. 행렬이란?

* 행과 열로 구성된 자료구조
* 예) m × n 행렬은

- m : 행의 개수
- n : 열의 개수
- 원소의 개수 : (m×n) 개

$$A = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{vmatrix}$$

2. 전치행렬

- 행렬의 행과 열을 서로 교환하여 구성한 행렬
- 행렬 A의 모든 원소의 위치(i, j)를 (j, i)로 교환
- m×n 행렬을 n×m 행렬로 변환한 행렬 A'은 행렬 A의 전치행렬

$$A = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{vmatrix} \qquad A' = \begin{vmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \vdots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{vmatrix}$$

* 예)

$$A = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{vmatrix} \quad \text{전치행렬 변환} \longrightarrow \quad A' = \begin{vmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{vmatrix}$$

3. 행렬의 순차 자료구조 표현

* 2차원 배열 사용
- m×n 행렬을 m행 n열의 2차원 배열로 표현

$$A = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{vmatrix} \Rightarrow$$

A[3][4]

	[0]	[1]	[2]	[3]
[0]	1	2	3	4
[1]	5	6	7	8
[2]	9	10	11	12

(a)3×4 행렬 A와 배열 A[3][4]

$$B = \begin{vmatrix} 0 & 0 & 2 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 7 & 0 & 0 \\ 23 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 31 & 0 & 0 & 0 \\ 0 & 14 & 0 & 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6 \\ 52 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 11 & 0 & 0 \end{vmatrix} \Rightarrow$$

B[8][7]

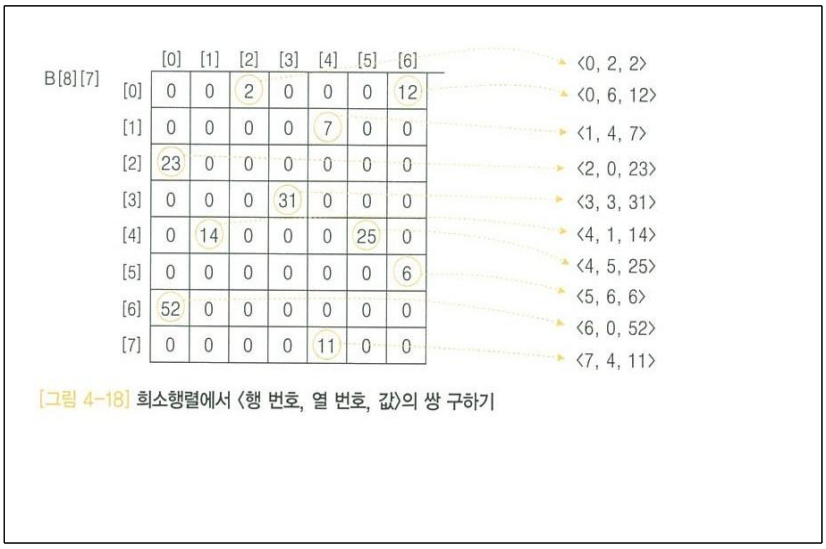
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
[0]	0	0	2	0	0	0	12
[1]	0	0	0	0	7	0	0
[2]	23	0	0	0	0	0	0
[3]	0	0	0	31	0	0	0
[4]	0	14	0	0	0	25	0
[5]	0	0	0	0	0	0	6
[6]	52	0	0	0	0	0	0
[7]	0	0	0	0	11	0	0

(b)8×7 행렬 B와 배열 B[8][7]

[그림 4-17] 행렬에 대한 2차원 배열 표현

* 희소행렬에 대한 2차원 배열 표현

- [그림4-17]과 같이 배열의 원소 56개 중에서 실제 사용하는 것은 10개로 메모리 공간 활용도가 떨러지는 배열을 희소행렬이라 함
- 0이 아닌 원소만 추출하여 <행번호, 열 번호, 원소> 쌍으로 배열에 저장



[그림 4-18] 희소행렬에서 <행 번호, 열 번호, 값>의 쌍 구하기

- 추출한 순서쌍을 2차원 배열의 행으로 저장
- 원래의 행렬에 대한 정보를 순서쌍으로 작성하여 0번 행에 저장
  - < 전체 행의 개수, 전체 열의 개수, 0이 아닌 원소의 개수>

[그림 4-19] 희소행렬에 대한 2차원 배열

4. 희소행렬의 추상 자료형

```
ADT Sparse_Matrix
데이터 : 3원소쌍 <행 인덱스, 열 인덱스, 원소 값>의 집합

연산 :  a,b∈Sparse_Matrix; c∈Matrix; u,v∈value; i∈Row;  j∈Column;
      // a, b는 희소행렬, c는 행렬, u, v는 행렬의 원소값을 나타내며, i와 j는 행 인덱스와 열 인덱스를 나타냄
   smCreate(m,n) ::= return an empty sparse matrix with m× n;
      // m× n의 공백 희소행렬을 만드는 연산
   smTranspose(a) ::= return c where c[i,j]=v when a[i,j]=v;
      // 희소행렬 a[i,j]=v를 c[i,j]=v로 전치시킨 전치행렬 c를 구하는 연산
   smAdd(a,b) ::= if (a.dimension = b.dimension)
                     then return c where c[i,j]=v+u where a[i,j]=v and b[i,j]=u;
                  else return error;
      // 차수가 같은 희소행렬 a와 b를 합한 행렬 c를 구하는 연산
   smMulti(a,b) ::= if (a.n = b.m) then return c where c[i,j]=a[i,k]× b[k,j];
                    else return error;
      // 희소행렬 a의 열의 개수(n)와 희소행렬 b의 행의 개수(m)가 같은 경우에 두 행렬의 곱을 구하는 연산
End Sparse_Matrix
```

* 희소 행렬의 전치 연산에 대한 C 함수

```
01  typedef struct{
02        int row;
03        int col
04        int value;
05  } term;
06
```

```
07  void smTranspose(term a[], term b[]) {
08        int m, n, v, i, j, p;
09        m = a[0].row;    // 희소 행렬 a의 행 수
10        n = a[0].col;     // 희소 행렬 a의 열 수
11        v = a[0].value;  // 희소 행렬 a에서 0이 아닌 원소 수
12        b[0].row = n;     // 전치 행렬 b의 행 수
13        b[0].col = m;     // 전치 행렬 b의 열 수
14        b[0].value = t;  // 전치 행렬 b의 원소 수
15        if (v > 0) {       // 0이 아닌 원소가 있는 경우에만 전치 연산 수행
16            p = 1;
17            for (i = 0; i < n; i++)         // 희소 행렬 a의 열 별로 전치 반복 수행
18                for (j = 1; j <= v; j++)  //0이 아닌 원소 수에 대해서만 반복 수행
19                    if (a[j].col == i) {   // 현재의 열에 속하는 원소가 있으면 b[ ]에 삽입
20                        b[p].row = a[j].col
21                        b[p].col = a[j].row
22                        b[p].value = a[j].value
23                        p++;
24                    }
25        }
26  }
```

【학습정리】

1. 다항식을 배열을 사용하여 표현할 수 있는데 배열의 인덱스는 다항식 항의 지수를 표현하고 배열 원소에는 다항식 항의 계수를 지정한다.

2. 희소 다항식의 경우에는 메모리 효율성을 위해서 항의 개수에 따라 배열의 크기를 결정하는 방법을 사용한다.