3장. 순차 데이터 표현

3.1 배열 추상 데이터 타입

3.2 배열의 표현

3.3 Java에서의 배열

3.4 선형 리스트

3.5 다항식 추상 데이터 타입

3.6 희소 행렬 추상 데이터 타입

3.7 희소 행렬 연산의 Java 구현

3.4 선형 리스트

리스트

- 리스트(list)
  - 원소들의 순열(sequence)로써 원소들을 일렬로 정렬해 놓은 것

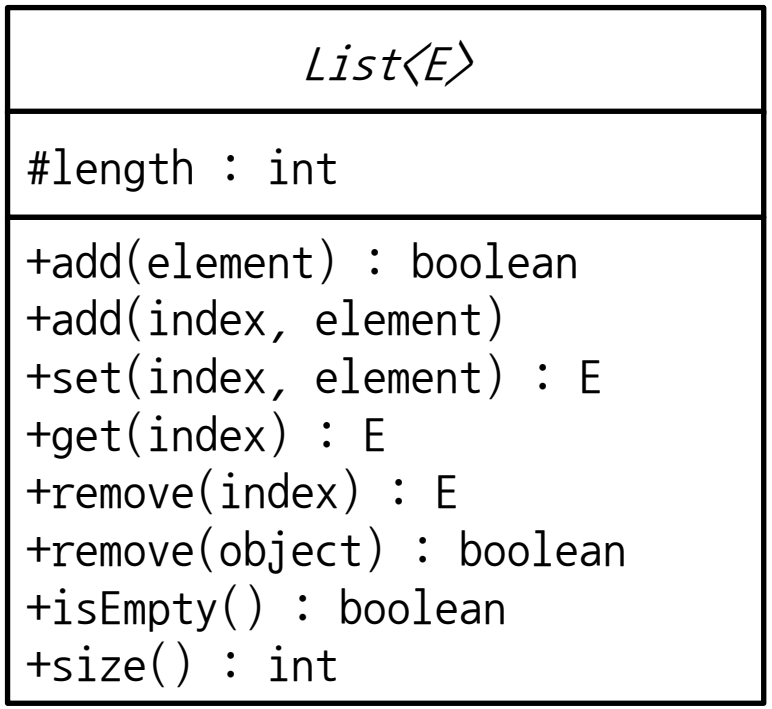
- 선형 리스트 (linear list)
  - 순서를 가진 원소들의 순열(sequence)
  - 물리적 순서가 아닌 원소의 특성에 의한 논리적 순서를 의미
  - 리스트는 기본적으로 순서 개념을 가지므로 선형 리스트라고 볼 수 있음

- 리스트는 보통 $L=(e_1, e_2, \ldots, e_n)$으로 표기
  - L은 리스트 이름, e_i는 리스트 원소
  - 공백 리스트(empty list, 원소가 하나도 없는 리스트)의 표현 : L=( )
  - 리스트의 각 원소는 선행자(predecessor)와 후속자(successor)를 가짐
  - 예
    - 자료 구조 강의 요일 = (월요일, 수요일, 금요일)
    - 토요일 강의 과목 = ( )

List<E> ADT – 리스트 처리 연산의 정의

- boolean add(E element)
  - 원소 element를 리스트의 제일 뒤에 삽입한다.
- void add(int index, E element)
  - 원소 element를 index 번째 삽입한다.
- E set(int index, E element)
  - 원소 element를 index 번째 저장하고, 기존에 저장되어 있던 원소를 반환한다.
- E get(int index)
  - index 번째 원소를 반환한다.
- E remove(int index)
  - index 번째 원소를 리스트에서 삭제하고 삭제한 원소를 반환한다.
- boolean remove(Object object)
  - 리스트에서 object 원소를 삭제한다. 삭제에 성공하면 true, 삭제할 원소가 없어 삭제에 실패하면 false를 반환한다.
- boolean isEmpty()
  - 리스트가 비어있으면 true, 그렇지 않으면 false를 반환한다.
- int size()
  - 리스트에 저장된 원소의 개수를 반환한다.

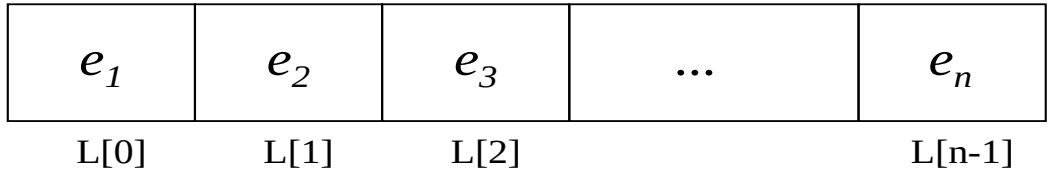
```
List list = …;
list.add(2,e);
list.remove(2);
```



리스트의 표현

- 배열을 이용한 표현: 순차 표현 리스트
  - 리스트 원소 e_i와 e_{i+1}이 인덱스 i-1과 i에 대응되게 연속적으로 저장
  - 저장된 원소의 물리적 순서가 논리적 순서를 나타냄 (순서를 표시하기 위한 특별한 장치가 필요 없음)
  - 삽입, 삭제시에 후속 원소들을 한자리씩 밀거나 당겨야 하는 오버헤드가 치명적인 약점

$$L = (e_1, e_2, \ldots, e_n)$$



List의 Java 표현

```
package hufs.dislab.util;

import java.util.NoSuchElementException;

public abstract class List<E> {

    public abstract boolean add(E element);
    public abstract void add(int index, E element);
    public abstract E set(int index, E element);
    public abstract E get(int index);
    public abstract E remove(int index);
    public abstract boolean remove(Object o);

    public boolean isEmpty() {
        return length == 0;
    }

    public int size() {
        return length;
    }

    public E getFirst() {
        if (length == 0)
            throw new NoSuchElementException();
        return get(0);
    }

    public E getLast() {
        if (length == 0)
            throw new NoSuchElementException();
        return get(length - 1);
    }
```

```
    public void addFirst(E e) {
        add(0, e);
    }

    public void addLast(E e) {
        add(e);
    }

    public E removeFirst() {
        if (length == 0)
            throw new NoSuchElementException();
        return remove(0);
    }

    public E removeLast() {
        if (length == 0)
            throw new NoSuchElementException();
        return remove(length - 1);
    }

    @Override
    public String toString() {
        StringBuffer str = new StringBuffer();
        str.append("(");
        for(int i = 0; i < size(); i++) {
            str.append(get(i));
            if (i < size() - 1)
                str.append(",");
        }
        str.append(")");
        return str.toString();
    }

    protected int length;
}
```

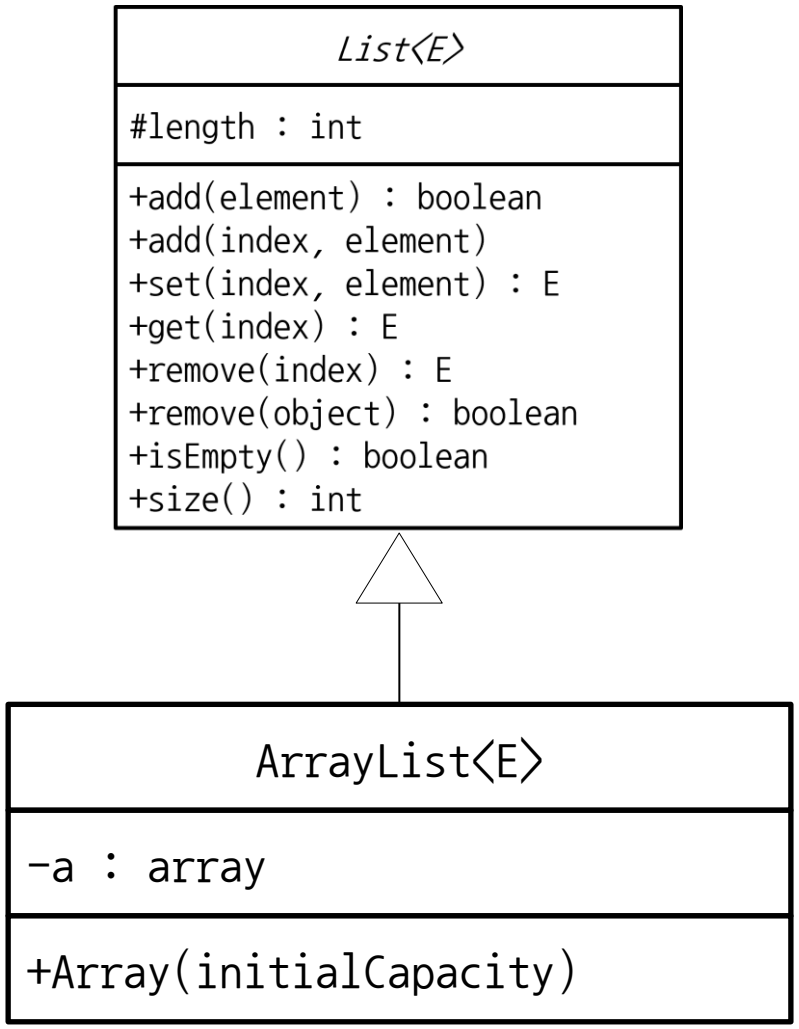
3.1 배열 추상 데이터 타입

배열과 인덱스

- 배열(array)의 특성
  - 순차적 메모리 할당 방식
  - <인덱스, 원소> 쌍의 집합 – 각 쌍은 인덱스와 그와 연관된 원소 값의 대응 관계를 나타냄
  - 원소들이 모두 같은 타입, 같은 크기

- 인덱스(index)
  - 순서를 나타내는 원소의 유한 집합
  - 집합 내에서의 상대적 위치 식별
  - 원소 수가 한정되어 있어 항상 마지막 원소가 존재
  - 인덱스만으로 원하는 원소를 직접 접근 (사용자는 내부 구현을 알 필요 없음 - 정보은닉)

배열 추상 데이터 타입




```
List<Integer> arr = new ArrayList<Integer>(5);
arr.add(2,5);
int i = arr.get(2);
```

Java를 이용한 배열 추상 데이터 타입

```java
package hufs.dislab.util;

public class ArrayList<E> extends List<E> {

    public ArrayList(int initialCapacity) {
        a = new Object[initialCapacity];
    }

    @Override
    public boolean add(E element) {
        if (length >= a.length)
            throw new IndexOutOfBoundsException();
        a[length++] = element;
        return true;
    }

    @Override
    public void add(int index, E element) {
        if (index < 0 || index > size() || index >= a.length)
            throw new IndexOutOfBoundsException();
        for(int i = size() - 1; i >= index; i--)
            a[i + 1] = a[i];
        a[index] = element;
        length++;
    }

    @Override
    @SuppressWarnings("unchecked")
    public E set(int index, E element) {
        if (index < 0 || index >= size())
            throw new IndexOutOfBoundsException();
        E oldElement = (E)a[index];
        a[index] = element;
        return oldElement;
    }
```

```java
    @Override
    @SuppressWarnings("unchecked")
    public E get(int index) {
        if (index < 0 || index >= size())
            throw new IndexOutOfBoundsException();
        return (E)(a[index]);
    }

    @Override
    @SuppressWarnings("unchecked")
    public E remove(int index) {
        if (index < 0 || index >= size())
            throw new IndexOutOfBoundsException();
        E oldElement = (E)a[index];
        for(int i = index; i < size() - 1; i++)
            a[i] = a[i + 1];
        length--;
        return oldElement;
    }

    @Override
    public boolean remove(Object o) {
        for(int i = 0; i < size(); i++) {
            if (get(i).equals(o)) {
                remove(i);
                return true;
            }
        }
        return false;
    }

    private Object[] a;
}
```

ArrayListTest

```java
package hufs.dislab.util;

   public class ArrayListTest {

    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>(5);

        list.add(1);
        list.add(1, 2);
        list.add(3);
        list.add(4);
        list.add(2, 5);
        // list.add(6);
        System.out.println(list);

        list.remove(1);
        System.out.println(list);

        list.remove(new Integer(3));
        System.out.println(list);
    }

}
```

3.2 배열의 표현

배열의 표현

- 인덱스
  - 배열 내에서 원소의 상대적 위치를 나타냄
  - 인덱스가 하나의 값으로 표현되면 1차원(one-dimensional), n개의 값으로 표현되면 n차원 (n-dimensional) 배열이라 함

- 배열 a[]는 인덱스와 원소의 쌍(<i, v>)의 집합으로 정의
  - a[i] : 인덱스 i에 대응하는 원소 v의 주소
  - <i, v> ∈ a[] 이면 a[i]=v
  - a[i] ← v : a[i]는 v가 저장될 공간
  - k ← a[i] : a[i]는 변수 k에 저장시킬 값을 검색해 올 주소, retrieve(a,i) 연산과 같은 기능 수행

1차원 배열

- 1차원 배열의 선언 : a[n]
  - a : 배열 이름, n : 원소의 최대 수, 인덱스는 {0, 1, ..., n-1}
- 메모리 표현(memory representation)
  - 연속적인 메모리 주소를 배열에 할당
  - 순차 사상(sequential mapping)
    - 배열의 논리적 순서와 메모리의 물리적 순서가 같도록 표현
  - 순차 표현(sequential representation)
    - 순차 사상(sequential mapping)을 이용하여 데이타를 표현
  - 원소는 할당된 메모리 내에서 인덱스 순서에 따라 저장
  - 1차원 배열의 순차 표현
    - a[0]의 주소(기준 주소: base address)가 α이면, a[i]의 주소는 α + i
    - 각 원소가 c개의 워드를 필요로 할 경우, a[i]의 주소는 α + c•i

주소	배열 a의 원소
α	a[0]
α+1	a[1]
. . .	. . .
α+i	a[i]
. . .	. . .
α+n-1	a[n-1]

2차원 배열 (1)

- 2차원 배열의 선언 : a[n_1, n_2]
  - n_1: 행(row)의 수, n_2: 열(column)의 수, 원소 수: $n_1 \cdot n_2$
  - 원소는 a[i,j]로 표현(i: 행 인덱스, j: 열 인덱스)

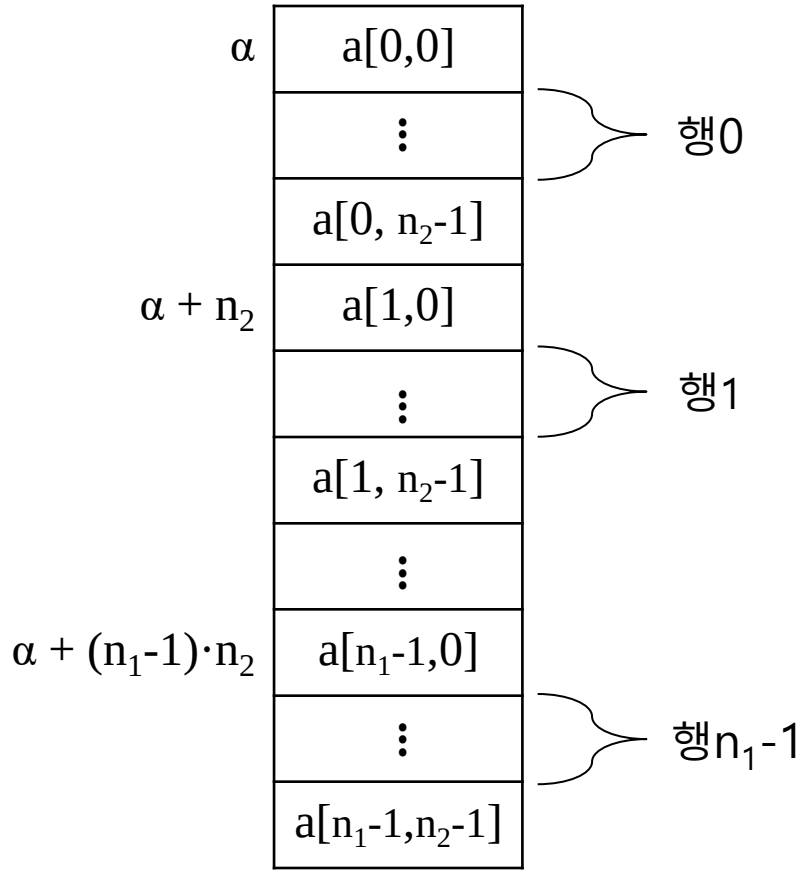
- 2차원 배열을 1차원 메모리로 사상
  - 행 우선 순서 (row major order): 일반적인 방법
  - 열 우선 순서 (column major order)

- a[2,3]의 행우선 순서의 표현
  - 3개의 원소로 된 2개의 행을 행 번호에 따라 한 줄로 연결
    즉, (<a[0,0], a[0,1], a[0,2]>, <a[1,0], a[1,1], a[1,2]> 순서
  - 원소를 차례로 접근 시 항상 열을 나타내는 인덱스가 먼저 변함
  - a[0,0]의 주소가 α라 할 때, 원소 a[1,0]의 주소는 $\alpha+1 \cdot 3+0 = \alpha+3$이 됨
    - 원소의 행 인덱스 값: 1, 배열의 열 수: 3, 원소의 열 인덱스 값: 0

2차원 배열 (2)

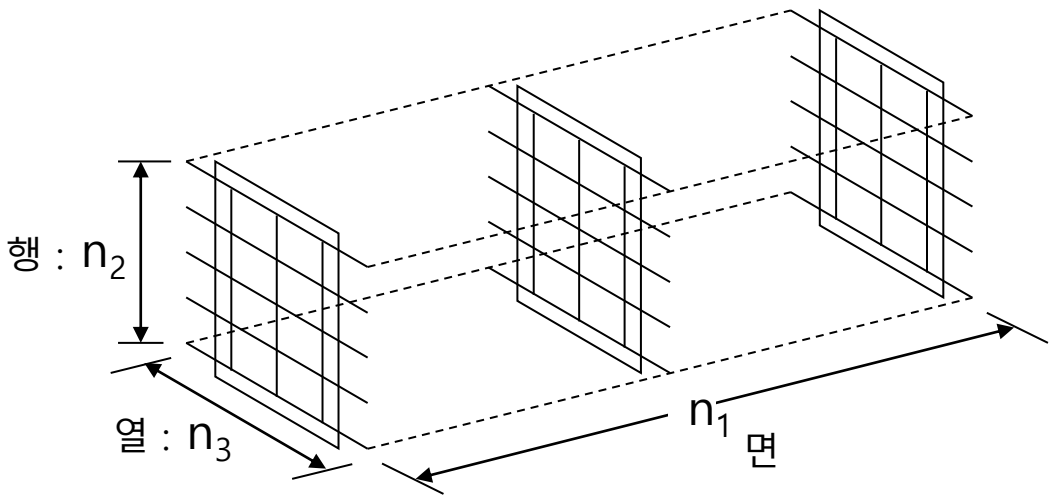
- 2차원 배열 $a[n_1, n_2]$
  - $a[0, 0]$의 주소가 α일 때, 원소 $a[i_1,i_2]$의 주소는 $\alpha + i_1 \cdot n_2 + i_2$가 된다

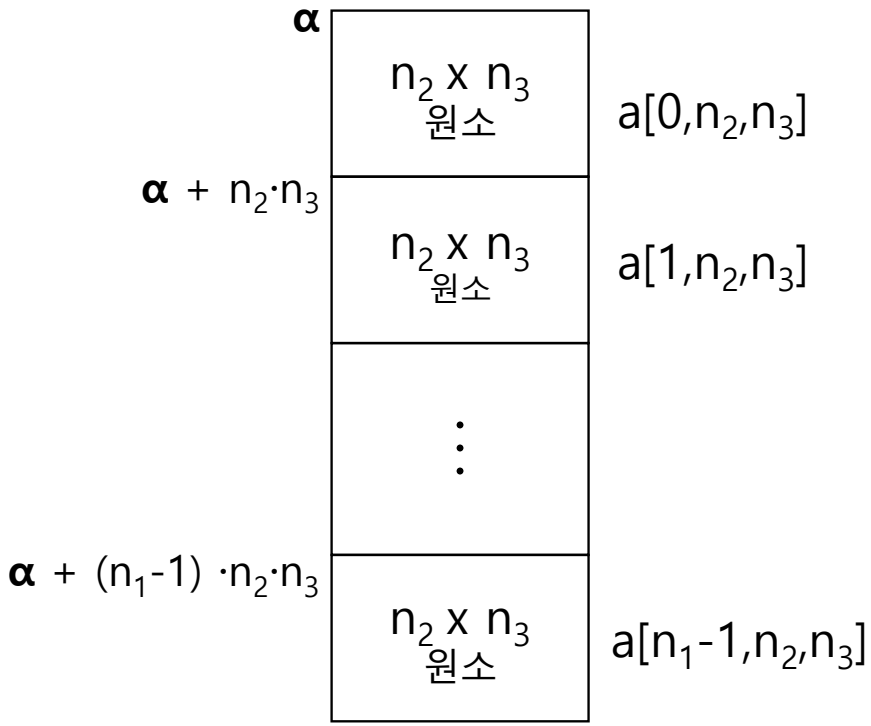
	열0	열1	…	열n_2-1
행0	a[0,0]	a[0,1]	…	a[0, n_2-1]
행1	a[1,0]	a[1,1]	…	a[1, n_2-1]
행2	a[2,0]	a[2,1]	…	a[2, n_2-1]
	⋮	⋮	…	⋮
행n_1-1	a[n_1-1,0]	a[n_1-1,1]	…	a[n_1-1, n_2-1]



3차원 배열

- 3차원 배열 a[n_1,n_2,n_3] (<면(plane), 행, 열>)의 메모리 표현
  - n_1개의 2차원 배열(n_2 X n_3)을 차례로 1차원 메모리에 순차 사상하는 방법을 이용
  - a[0,0,0]의 주소를 α라 할 때, a[i_1,i_2,i_3]의 주소는 $\alpha + i_1 \cdot n_2 \cdot n_3 + \mathbf{i_2} \cdot n_3 + i_3$





3.3 Java에서의 배열

Java에서의 배열 (1)

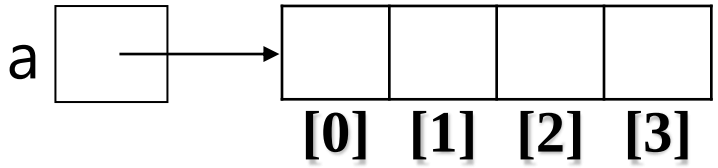
- Java의 배열
  - 일정 수의 컴포넌트를 순차적으로 정렬시킨 것
  - 정수(int), 불리언(boolean), 문자(char), 부동소수(double) 등 원시 타입은 물론, 객체 타입의 배열도 허용
  - 배열 변수는 객체를 참조하는 참조 변수와 똑같음

- 배열의 선언 예
  - 정수 배열 선언: `int[] a;`

Java에서의 배열 (2)

- 배열의 생성
  - **new** 연산자를 이용하여 객체를 생성
  - 예) `int[] a;`

```
a = new int[4];
```

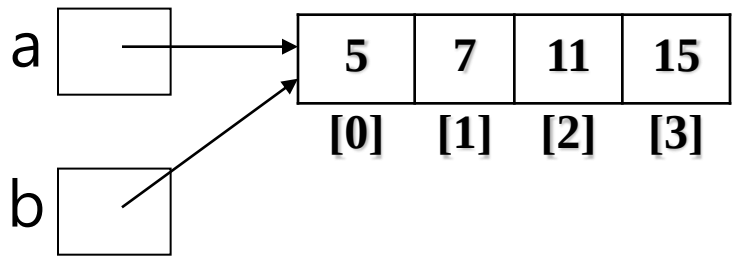


- 배열의 데이타 필드 length
  - 모든 배열이 생성될 때 배열 내부에 갖게 되는 데이타 필드
  - 원소 수를 표현
  - `a = new int[4];`가 실행된 뒤에 `a.length`는 4가 됨
  - `int[] a;`만 선언한 뒤에 `a.length`를 사용하면 a에 대한 객체가 생성되지 않았기 때문에 a는 null이 되어 length 접근은 불가. 따라서 시스템은 에러 메시지를 생성

Java에서의 배열 (3)

- 배열 참조
  - 배열 변수 a, b에 대해 b = a; 가 실행되면 b는 a가 참조하고 있는 배열을 똑같이 참조하게되어 a == b는 true가 됨
  - 예

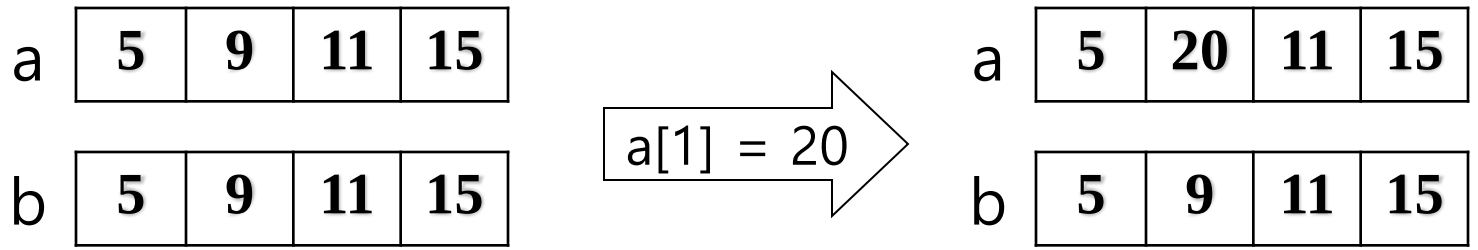
```
int[] a;
int[] b;
a = new int[4];   // 4개의 정수 원소에 대한 메모리 할당
a[0] = 5;   a[1] = 7;   a[2] = 11;   a[3] = 15;
b = a;
```



  - 배열a의 원소 변경은 다른 배열 변수 b에 영향을 줌

Java에서의 배열 (4)

- 배열 복제
  - Java에서는 객체를 복제할 수 있는 메소드 clone()을 제공
  - 예) `b = (int[]) a.clone();`
    - 여기서 (**int**[])는 clone() 메소드가 복제한 결과를 정수(**int**) 타입으로 변환(cast)하라는 것을 명세
    - 이제 두 개의 배열이 만들어졌기 때문에 한 배열의 원소 변경은 서로 영향을 주지 않음.



3.5 다항식 추상 데이터 타입

다항식 추상 데이타 타입 (1)

- 다항식(polynomial)
  - 수학적으로 ax^e형식을 가진 항(term)의 합으로 정의
    - a: 계수(coefficient), x: 변수(variable), e: 지수(exponent)
    - 다항식의 제일 큰 지수: 다항식의 차수(degree)
      - 예: $A(x) = 2x^3 + 4x^2 + 5$

List로 표현한 다항식 추상 데이터 타입

```java
package hufs.dislab.util;

import java.util.NoSuchElementException;

public abstract class Polynomial {

    protected List<Term> terms;

    public boolean isZero() {
        return terms.isEmpty();
    }

    public double coef(int exp) {
        for(int i = 0; i < terms.size(); i++) {
            Term term = terms.get(i);
            if (term.exp == exp)
                return term.coef;
        }
        throw new NoSuchElementException();
    }

    public int maxExp() {
        if (isZero())
            throw new NoSuchElementException();
        return terms.get(0).exp;
    }
```

```java
    public Polynomial addTerm(double coef, int exp) {
        terms.add(new Term(coef, exp));
        return this;
    }

    public Polynomial delTerm(int exp) {
        for(int i = 0; i < terms.size(); i++) {
            if (terms.get(i).exp == exp) {
                terms.remove(i);
                return this;
            }
        }
        throw new NoSuchElementException();
    }

    /**
     * Factory method
     * 적절한 객체의 생성을 하위 클래스에게 맡기기 위한
     * 메소드
     * @return 새로운 다항식 객체
     */
    public abstract Polynomial createPolynomial();

    @Override
    public String toString() {
        return terms.toString();
    }
```

```
    public Polynomial add(Polynomial p) {
        Polynomial p1 = this;
        Polynomial p2 = p;
        Polynomial p3 = createPolynomial();

        while(!p1.isZero() && !p2.isZero()) {
            if (p1.maxExp() < p2.maxExp()) {
                p3.addTerm(p2.coef(p2.maxExp()), p2.maxExp());
                p2.delTerm(p2.maxExp());
            } else if (p1.maxExp() == p2.maxExp()) {
                double sum = p1.coef(p1.maxExp()) + p2.coef(p2.maxExp());
                if (sum != 0)
                    p3.addTerm(sum, p1.maxExp());
                p1.delTerm(p1.maxExp());
                p2.delTerm(p2.maxExp());
            } else {
                p3.addTerm(p1.coef(p1.maxExp()), p1.maxExp());
                p1.delTerm(p1.maxExp());
            }
        }
           while (!p1.isZero()) {
            p3.addTerm(p1.coef(p1.maxExp()), p1.maxExp());
            p1.delTerm(p1.maxExp());
        }

        while (!p.isZero()) {
            p3.addTerm(p2.coef(p2.maxExp()), p2.maxExp());
            p2.delTerm(p2.maxExp());
        }

        return p3;
    }
}
```

다항식 연산의 구현

- 다항식 생성
  - 모든 다항식은 기본적으로 addTerm()을 이용해 생성할 수 있음
    - 예) $2x^3 + 4x^2 + 5$
    - p.addTerm(2,3).addTerm(4,2).addTerm(5,0)

- 다항식의 표현 시 가정
  - 모든 항은 지수에 따라 내림차순으로 정렬
  - 모든 항의 지수는 상이함
  - 계수가 0인 항은 포함하지 않음

- 다항식의 덧셈
  - 두 다항식의 항들의 지수를 비교하여 합병, 복사, 삭제를 수행

프로그램에서 다항식의 표현 방법

- 계수로만 표현
  - 지수의 내림차순에 따라 다항식의 계수만 표현
  - 차수가 n인 항은 지수 n+1개에 해당하는 계수만을 순서 리스트(배열)에 표현
  - 장점 : 다항식의 덧셈이나 곱셈 연산이 간단하고 지수를 별도로 저장할 필요가 없음
  - 단점 : 0인 항이 많은 희소 다항식인 경우 공간 낭비
    - 예) $x^{1000} + 1$: 원소 2개만 0이 아니고 나머지 999개가 모두 0

- 지수-계수 쌍으로 표현
  - 0이 아닌 항의 지수-계수 쌍만 차례로 저장
  - 단점 : 다항식 연산의 알고리즘이 복잡 (지수 검사 필요)
  - 장점 : 효율적으로 저장 공간 활용
  - 보통 이 방법을 더 선호

- 예: $2x^4 + 10x^3 + x^2 + 6$
  - 계수 표현 : (2, 10, 1, 0, 6): 1차원 배열로 표현
  - 지수-계수 표현 : (4, 2, 3, 10, 2, 1, 0, 6)또는 (4, 2), (3, 10), (2, 1), (0, 6)
    : 2차원 배열로 표현

ArrayPolynomial 클래스

```java
package hufs.dislab.util;

public class ArrayPolynomial extends Polynomial {


    public ArrayPolynomial() {
        terms = new ArrayList<Term>(20);
    }

    @Override
    public Polynomial createPolynomial() {
        return new ArrayPolynomial();
    }

}
```

```java
package hufs.dislab.util;

public class PolynomialTest {

    public static void main(String[] args) {
        Polynomial p1, p2;

        p1 = new ArrayPolynomial();
        p2 = new ArrayPolynomial();
        createPolynomial(p1, p2);
        addTest(p1, p2);
        System.out.println();
    }

    public static void createPolynomial(Polynomial p1, Polynomial p2) {
        p1.addTerm(2, 3).addTerm(4, 2).addTerm(5, 0);
        p2.addTerm(-4, 2).addTerm(8, 1);
    }

    public static void addTest(Polynomial p1, Polynomial p2) {
        System.out.println(p1);
        System.out.println(p2);

        Polynomial p3 = p1.add(p2);

        System.out.println(p1);
        System.out.println(p2);
        System.out.println(p3);
    }

}
```

```
((2.0,3),(4.0,2),(5.0,0))
((-4.0,2),(8.0,1))
()
()
((2.0,3),(8.0,1),(5.0,0))
```

add 메소드가 개선된 RevisedArrayPolynomial 클래스

```
package hufs.dislab.util;

public class RevisedArrayPolynomial extends ArrayPolynomial {

    @Override
    public Polynomial createPolynomial() {
        return new RevisedArrayPolynomial();
    }

    @Override
    public Polynomial add(Polynomial p) {
        RevisedArrayPolynomial p1 = this;
        RevisedArrayPolynomial p2 = (RevisedArrayPolynomial)p;
        Polynomial p3 = createPolynomial();

        int i1 = 0, i2 = 0;

        while(i1 < p1.terms.size() && i2 < p2.terms.size()) {
            Term t1 = terms.get(i1);
            Term t2 = terms.get(i2);

            if (t1.exp < t2.exp) {
                p3.addTerm(t2.coef, t2.exp);
                i2++;
            } else if (t1.exp == t2.exp) {
                double sum = t1.coef + t2.coef;
                if (sum != 0)
                    p3.addTerm(sum, t1.exp);
                i1++;
                i2++;
            } else {
                p3.addTerm(t1.coef, t1.exp);
                i1++;
            }
        }
```

```
        for( ; i1 < p1.terms.size(); i1++) {
            Term term = p1.terms.get(i1);
            p3.addTerm(term.coef, term.exp);
        }

        for( ; i2 < p2.terms.size(); i2++) {
            Term term = p2.terms.get(i2);
            p3.addTerm(term.coef, term.exp);
        }

        return p3;
    }
}
```

RevisedArrayPolynomial 테스트

```
package hufs.dislab.util;

public class PolynomialTest {
    public static void main(String[] args) {
        Polynomial p1, p2;
        p1 = new RevisedArrayPolynomial();
        p2 = new RevisedArrayPolynomial();
        createPolynomial(p1, p2);
        addTest(p1, p2);
        System.out.println();

    }

    public static void createPolynomial(Polynomial p1, Polynomial p2) {
        p1.addTerm(2, 3).addTerm(4, 2).addTerm(5, 0);
        p2.addTerm(-4, 2).addTerm(8, 1);
    }

    public static void addTest(Polynomial p1, Polynomial p2) {
        System.out.println(p1);
        System.out.println(p2);

        Polynomial p3 = p1.add(p2);

        System.out.println(p1);
        System.out.println(p2);
        System.out.println(p3);
    }
}
```

```
((2.0,3),(4.0,2),(5.0,0))
((-4.0,2),(8.0,1))
((2.0,3),(4.0,2),(5.0,0))
((-4.0,2),(8.0,1))
((4.0,3),(8.0,2),(5.0,0))
```

3.6 희소 행렬 추상 데이터 타입

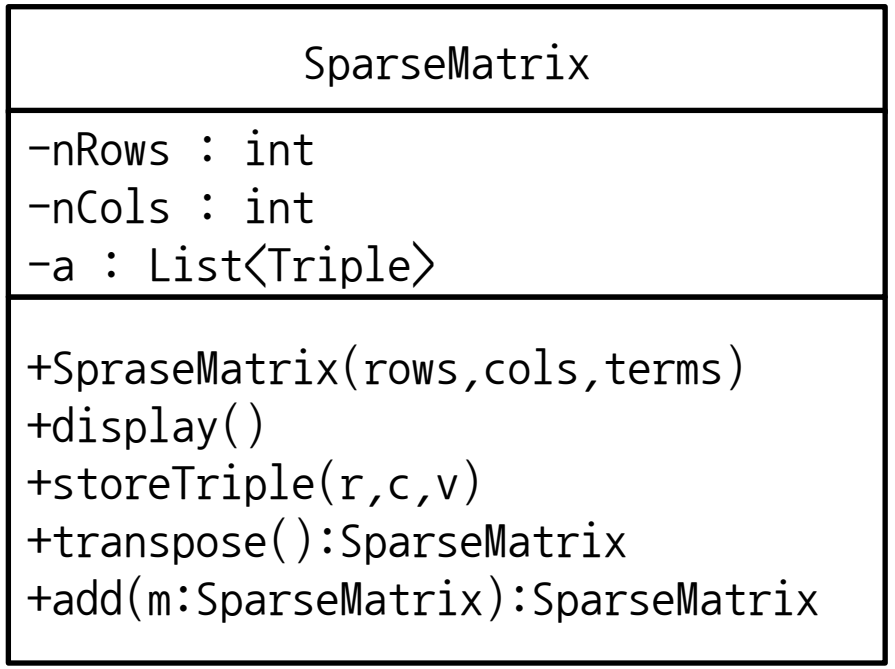
희소 행렬 (sparse matrix)

- 행렬 (matrix)
  - m X n 행렬 : m개의 행과 n개의 열로 구성
  - m X n: 행렬의 차수
  - m과 n이 같은 행렬을 정방 행렬(square matrix)
  - 2차원 배열로 표현
  - 행렬을 a[m, n]으로, 각 원소는 a[i, j] (i는 행, j는 열)로 표현

- 희소 행렬 (sparse matrix)
  - 전체 원소 수에 비하여 극소수의 원소만이 0이 아닌 행렬
  - 행렬의 원소 값이 대부분 0으로 구성
  - 일반적인 2차원 배열로 저장하면 공간의 낭비가 심함.
    - 행렬의 원소가 0이 아닌 원소만 저장하는 방법이 필요
    - 0이 아닌 원소를 <행,렬, 값> 3원소 쌍으로 저장하면 효율적임

희소 행렬 추상 데이타 타입

```java
package hufs.dislab.util;

public class SparseMatrix {

    private int nRows, nCols;
    private List<Triple> a;

    public SparseMatrix(int rows, int cols, int terms);

    public void display();

    public void storeTriple(int r, int c, int v);

    public SparseMatrix transpose();

}
```



희소 행렬의 표현 방법

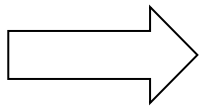
- 희소 행렬의 0이 아닌 원소는 <행, 열, 값> 3원소 쌍으로 식별
  - 0이 아닌 원소들에 대해 열이 3인 2차원 배열로 표현
  - 효율적인 연산을 위해 행과 열을 오름차순으로 저장

※ 배열의 첫 번째 행은 희소 행렬의 정보를 저장
배열의 행 수는 (0이 아닌 원소 수 + 1)이 됨

- c[0, 0], c[0, 1]은 각각 희소 행렬 b의 행수와 열수, c[0, 2]는 0이 아닌 원소 수를 나타냄

$$b_{7X6} = \begin{bmatrix} 76 & 0 & 0 & 0 & 13 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 25 & 0 & 0 & 0 & 0 \\ -19 & 0 & 0 & 56 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 13 \\ 0 & 0 & 3 & 0 & 0 & 0 \end{bmatrix}$$

c[9,3]



	0	1	2
0	7	6	8
1	0	0	76
2	0	4	13
3	2	5	3
4	3	1	25
5	4	0	-19
6	4	3	56
7	5	5	13
8	6	2	3

행렬의 전치

- 전치 행렬 (transposed matrix)
  - 원소의 행과 열을 교환시킨 행렬
  - 원소 <i,j,v> → **<j,i,v>**
    - <0,4,13> → <4,0,13>

- 간단한 전치 행렬 알고리즘 (행우선 표현)

```
for (j ←0; j <= n-1 ; j ←j+1 ) do
      for (i ←0; i <=m-1 ; i ←i+1 ) do
            b[j, i] ← a[i, j];
```

- 희소 행렬의 전치
  - 0이 아닌 원소만 표현
  - 행 우선이면서 행 내에선 열 오름차 순이 되도록 저장
  - 전치 연산: 주어진 행렬에서 각 열 별로 모든 원소를 찾아 차례로 행의 오름차 순으로 저장

Java로 표현한 간단한 전치

```java
package hufs.dislab.util;

public class Matrix {
    public Matrix(int m, int n) {
        this.m = m;
        this.n = n;
        a = new int[m][n];
    }

    public Matrix set(int i, int j, int value) {
        a[i][j] = value;
        return this;
    }

    public int get(int i, int j) {
        return a[i][j];
    }

    public Matrix transpose() {
        Matrix mat = new Matrix(n, m);
        for(int j = 0; j <= n - 1; j++) {
            for(int i = 0; i < m - 1; i++) {
                mat.a[j][i] = a[i][j];
            }
        }
        return mat;
    }

    public void display() {
        for(int i = 0; i < a.length; i++) {
            for(int j = 0; j < a[i].length; j++)
                System.out.printf("%4d", a[i][j]);
            System.out.println();
        }
    }

    private int[][] a;
    private int m;
    private int n;
}
```

```java
package hufs.dislab.util;

public class MatrixTest {

    public static void main(String[] args) {
        Matrix m = new Matrix(7, 6);

        m.set(0, 0, 76);
        m.set(0, 4, 13);
        m.set(2, 5, 3);
        m.set(3, 1, 15);
        m.set(4, 0, -19);
        m.set(4, 3, 56);
        m.set(5, 5, 13);
        m.set(6, 2, 13);

        m.display();
        System.out.println();

        Matrix m2 = m.transpose();
        m2.display();
    }
}
```

```
  76   0   0   0  13   0
   0   0   0   0   0   0
   0   0   0   0   0   3
   0  15   0   0   0   0
 -19   0   0  56   0   0
   0   0   0   0   0  13
   0   0  13   0   0   0

  76   0   0   0 -19   0   0
   0   0   0  15   0   0   0
   0   0   0   0   0   0   0
   0   0   0   0  56   0   0
  13   0   0   0   0   0   0
   0   0   3   0   0  13   0
```

희소 행렬 전치 알고리즘

```java
package hufs.dislab.util;

public class SparseMatrix {

    public class Triple {
        int row;
        int col;
        int value;

        public Triple() {
            row = col = value = 0;
        }

        public Triple(int r, int c, int v) {
            row = r;
            col = c;
            value = v;
       }
    }

    private int nRows, nCols;
    private List<Triple> a;

    public SparseMatrix(int rows, int cols, int terms) {
        nRows = rows;
        nCols = cols;
        a = new ArrayList<Triple>(terms);
    }

    public void display() {
        System.out.println("Number of rows : " + nRows);
        System.out.println("Number of columns : "
                           + nCols);
        System.out.println("Number of non-zero terms : "
                           + a.size());
        for(int i = 0; i < a.size(); i++)
            System.out.println("[" + i + "]" + " "
                + a.get(i).row + " "
                + a.get(i).col + " " + a.get(i).value);
    }

    public void storeTriple(int r, int c, int v) {
        a.add(new Triple(r, c, v));
    }



    public SparseMatrix transposeS() {
        int nTerms = a.size();
        SparseMatrix m = new SparseMatrix(nCols, nRows, nTerms);

        m.nCols = nRows;
        m.nRows = nCols;

        if (nTerms > 0) {
            for(int p = 0; p < nRows; p++) {
                for(int i = 0; i < nTerms; i++) {
                    if (a.get(i).col == p) {
                        Triple t = a.get(i);
                        m.storeTriple(t.col, t.row, t.value);
                    }
                }
            }
        }

        return m;
    }
}
```

```java
package hufs.dislab.util;

public class SparseMatrixTest {

    public static void main(String[] args) {
        SparseMatrix a = new SparseMatrix(7, 6, 8);

        a.storeTriple(0, 0, 76);
        a.storeTriple(0, 4, 13);
        a.storeTriple(2, 5,  3);
        a.storeTriple(3, 1, 25);
        a.storeTriple(4, 0, -19);
        a.storeTriple(4, 3, 56);
        a.storeTriple(5, 5, 13);
        a.storeTriple(6, 2, 13);

        a.display();

        SparseMatrix b = a.transposeS();
        b.display();
    }

}
```

```
Number of rows : 7
Number of columns : 6
Number of non-zero terms : 8
[0] 0 0 76
[1] 0 4 13
[2] 2 5 3
[3] 3 1 25
[4] 4 0 -19
[5] 4 3 56
[6] 5 5 13
[7] 6 2 13
Number of rows : 6
Number of columns : 7
Number of non-zero terms : 8
[0] 0 0 76
[1] 0 4 -19
[2] 1 3 25
[3] 2 6 13
[4] 3 4 56
[5] 4 0 13
[6] 5 2 3
[7] 5 5 13
```

행렬 전치 알고리즘의 시간 복잡도

- 일반 m X n행렬에 대한 전치 행렬 알고리즘
  - 중첩된 for 루프로 인해 O(m • n)

- 희소 행렬의 전치 알고리즘 TransposeS
  - 중첩된 for 루프로 인해 O(n • t)
    - t는 0이 아닌 원소 수
  - 최악의 경우 t = m • n이 될 수 있으므로
    O(n • t) = O(m • n^2)이 되어 O(m • n)보다 커질 수 있음

- 보다 효율적인 전치 알고리즘
  - 각 열에 대해 먼저 0이 아닌 원소 수를 계산하면 이것이 곧 전치 행렬을 표현하는 배열 b[]에 대한 각 행의 원소수가 됨.
  - 이 정보로 배열 b[]에 저장시킬 각 행의 시작점(인덱스)을 결정
  - 그런 다음 배열 a[]의 원소를 하나씩 차례로 전치시켜 그 행의 인덱스 값에 따라 배열 b[]에 저장
  - 시간 복잡도는 O(n + t)

3.7 희소 행렬 연산의 Java 구현

희소 행렬 전치 알고리즘의 Java 구현

- 3원소 쌍의 표현
  - 2차원 배열이 아니라 클래스(Triple)로 표현
  - 필요한 만큼 3원소 객체를 생성하기 때문에 임의의 수의 0이 아닌 원소를 처리할 수 있음: 융통성 제공
- Triple 클래스
  - <행, 열, 값> 에 대한 데이터 필드와 생성자를 포함한 클래스

```java
package hufs.dislab.util;

public class SparseMatrix {
    ...

    public SparseMatrix transpose() {
        int[] rowTerms = new int[nCols];
        int[] rowBegins = new int[nCols];

        int nTerms = a.size();
        SparseMatrix m = new SparseMatrix(nCols, nRows, nTerms);
        if (nTerms > 0) {
            int i;
            for(i = 0; i < nCols; i++)
                rowTerms[i] = 0;
            for(i = 0; i < nTerms; i++)
                rowTerms[ a.get(i).col ]++;

            rowBegins[0] = 0;
            for(i = 1; i < nCols; i++)
                rowBegins[i] = rowBegins[i - 1] + rowTerms[i - 1];

            for(i = 0; i < nTerms; i++)
                m.a.add(null);

            for(i = 0; i < nTerms; i++) {
                Triple t = a.get(i);
                int no = rowBegins[ t.col ]++;
                m.a.set(no, new Triple(t.col, t.row, t.value));
            }
        }
        return m;
    }
}
```

```
(0, 0, 76)
(0, 4, 13)
(2, 5,  3)
(3, 1, 25)
(4, 0, -19)
(4, 3, 56)
(5, 5, 13)
(6, 2, 13)
```

row :	[0]	[1]	[2]	[3]	[4]	[5]
rowTerms =	2	1	1	1	1	2
rowBegins =	0	2	3	4	5	6

전치 행렬 실행 프로그램

```java
package hufs.dislab.util;

public class SparseMatrixTest {

    public static void main(String[] args) {
        SparseMatrix a = new SparseMatrix(7, 6, 8);

        a.storeTriple(0, 0, 76);
        a.storeTriple(0, 4, 13);
        a.storeTriple(2, 5,  3);
        a.storeTriple(3, 1, 25);
        a.storeTriple(4, 0, -19);
        a.storeTriple(4, 3, 56);
        a.storeTriple(5, 5, 13);
        a.storeTriple(6, 2, 13);

        a.display();

        SparseMatrix b = a.transpose();
        b.display();
    }

}
```

```
Number of rows : 7
Number of columns : 6
Number of non-zero terms : 8
[0] 0 0 76
[1] 0 4 13
[2] 2 5 3
[3] 3 1 25
[4] 4 0 -19
[5] 4 3 56
[6] 5 5 13
[7] 6 2 13
Number of rows : 6
Number of columns : 7
Number of non-zero terms : 8
[0] 0 0 76
[1] 0 4 -19
[2] 1 3 25
[3] 2 6 13
[4] 3 4 56
[5] 4 0 13
[6] 5 2 3
[7] 5 5 13
```

결과

- 원래의 행렬

Number of rows : 7
Number of columns : 6
Number of non-zero terms : 8
[0] 0 0 76
[1] 0 4 13
[2] 2 5 3
[3] 3 1 25
[4] 4 0 -19
[5] 4 3 56
[6] 5 5 13
[7] 6 2 3

- 전치된 행렬

Number of rows : 6
Number of columns : 7
Number of non-zero terms : 8
[0] 0 0 76
[1] 0 4 -19
[2] 1 3 25
[3] 2 6 3
[4] 3 4 56
[5] 4 0 13
[6] 5 2 3
[7] 5 5 13

희소 행렬 전치 프로그램의 시간 복잡도

- 시간 복잡도 : O(Ncols + Nterms)
  - 각각 Ncols, Nterms, Ncols-1, Nterms번 실행되는 4개의 루프가 있으므로