Lecture 7

배열을 활용한 리스트

2018년도 2학기

컴퓨터프로그래밍2

김 영 국
충남대학교 컴퓨터공학과

이번 주에 학습할 내용

- 자바의 배열 (복습)
- 자바에서 배열의 프린팅
- 순차 탐색과 이진 탐색
- java.util.Arrays 클래스
- 사용자 정의 IntArrays 클래스
- 순서 배열의 유지
- 간접 참조

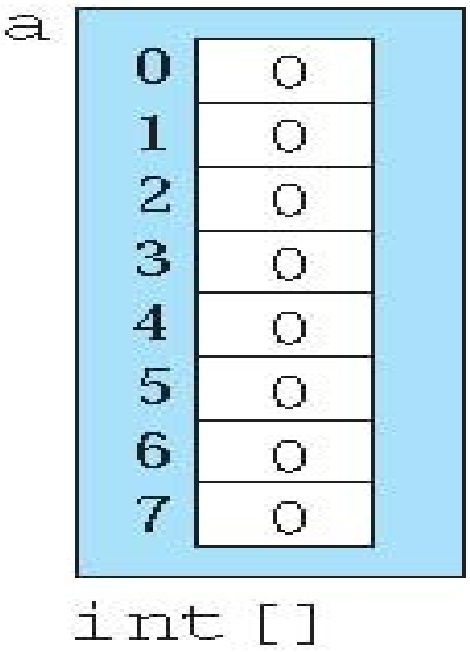
배열을 활용해서 리스트를 구현하는 방법을 살펴봅시다.

1. Java의 배열

- 배열은 객체임
- 배열의 타입은 t[] 형태
  - t는 배열의 원소 타입
  - 예: 원소 타입이 int이면 배열 타입은 int[]
- 유효한 배열 선언문
  - int[] a; // 원소는 프리미티브 타입 int임
  - String[] args; // 원소는 객체 타입 String임
  - List[] lists; // 원소는 인터페이스 타입 List임
  - double[][] matrix; // 원소는 배열 타입 double[]임
  - int[][][] x3d; // 원소는 배열 타입 int[][]임

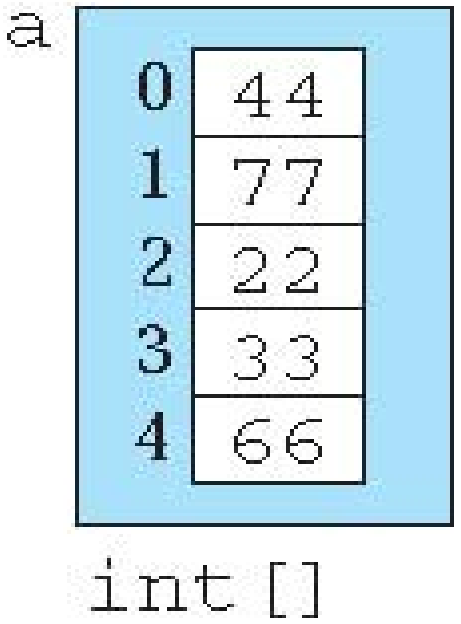
1. Java의 배열

- new 연산자를 이용해 메모리 할당
  - a = new int[8];
    // 8개의 int 원소로 된 배열을 할당
  - d = new Date(); // Date 객체를 할당
- 프리미티브 타입의 배열을 new 연산자로 할당하면 그 원소들은 자동적으로 그 타입의 0의 값으로 초기화됨
- 배열은 항상 0을 기반으로 인덱스가 부여됨
  - 배열의 길이가 n일 때 인덱스의 범위는 0에서 n-1이 됨



1. Java의 배열

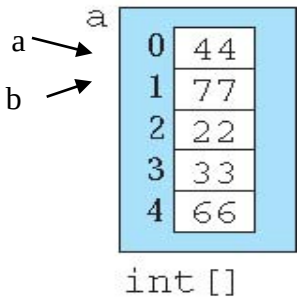
- 배열의 길이는 length 필드를 이용해서 접근
  - int n = a.length; // 배열 a의 원소의 수
- Java는 자동으로 배열의 인덱스를 검사함
  - 인덱스 범위를 벗어나면 ArrayIndexOutOfBoundsException 예외를 발생시킴
- 초기화 리스트를 이용해 배열을 초기화할 수 있음
  - int a = {44, 77, 22, 33, 66};




```
int[] a = new int[5];
a[0] = 44;
a[1] = 77;
a[2] = 22;
a[3] = 33;
a[4] = 66;
```

1. Java의 배열

- 배열과 관련된 기초적인 사항들
  - 다른 객체와 마찬가지로 배열도 그에 대해 여러 개의 참조를 가질 수 있다.
    - int[] aa = a;
  - 메소드의 매개변수 리스트에 배열 매개변수 선언 가능
    - public void print(int[] a)
  - 배열은 이름만 이용해서 메소드로 전달됨
    - print(a);
  - 배열 타입은 메소드에 대한 리턴 타입이 될 수 있음
    - public int[] randomArray(int n)
  - 한 배열을 다른 배열에 할당해도 실제로 복사되는 것은 아니다. 단지 다른 이름(즉, 다른 참조)만 부여하게 된다.
    - b = a; // a[]와 b[]는 동일 배열이 됨



1. Java의 배열

- 배열 복사:
  - System 클래스에 정의된 arraycopy() 메소드 이용
  - System.arraycopy(a, 0, b, 0, a.length);
- 중복 배열 생성:
  - Object 클래스에 정의된 clone() 메소드 이용
  - b = (int[]) a.clone();
  - clone()에 대한 리턴 타입은 Object이므로 타입을 배열로 변환(type casting)시켜야 한다.
- 배열은 대개 for 루프를 이용해서 처리

```
for (int i = 0; i < a.length; i++)
    a[i] = random.nextInt(1000);
```

- final로 선언된 배열은 재할당될 수 없다.
  - final int[] a = {22, 44, 66, 88};
  - a[3] = 99; // OK
  - a = new int[8]; // 오류 발생

배열의 테스팅

- **LISTING 1: Testing Arrays**

```
1 public class Main {
2       private static java.util.Random random = new java.util.Random();
4       public static void main(String[] args) {
5            int[] a = randomInts(10,1000);
6            int[] aa = (int[]) a.clone(); // creates a duplicate of a in aa
7            print(a);   print(aa);
9            a[0] = a[1] = a[2] = 888;
10           print(a);   print(aa);
12      }
14      public static int[] randomInts(int n, int range) {
15           int[] a = new int[n];
16           for (int i = 0; i < n; i++)
17                a[i] = random.nextInt(range);
18           return a;
19      }
21      public static void print(int[] a) {
22           System.out.print("{" + a[0]);
23           for (int i = 1; i < a.length; i++)
24                     System.out.print("," + a[i]);
25           System.out.println("}");
26      }
27 }
```

배열의 테스팅

- 출력 결과

{102,955,717,249,649,596,131,414,849,759}

{102,955,717,249,649,596,131,414,849,759}

{888,888,888,249,649,596,131,414,849,759}

{102,955,717,249,649,596,131,414,849,759}

2. Java에서 배열의 프린팅

- 배열의 이름은 실제로는 배열에 대한 참조 변수의 이름임
  - 이 변수는 메모리에서 배열의 시작 주소를 저장함
  - 이 변수를 프린트하면 메모리 주소를 16진수로 출력
  - 출력 문자열: [I@73d6a5
    - [I : 타입 int[]의 배열임을 의미
    - @73d6a5 : 배열이 저장된 메모리의 주소
- 배열의 원소 프린팅
  - 반복문을 이용해 한번에 한 원소씩 출력

배열 참조의 프린팅

- **LISTING 2: Printing an Array Reference**

```
1 public class Print {
2       public static void main(String[] args) {
3           int[] a = {66, 33, 99, 88, 44, 55, 22};
4           System.out.println(a);
5       }
6 }
```

- 출력 결과

```
[I@73d6a5
```

3. 탐색(Search)

- **순차 탐색 (선형 탐색 또는 직렬 탐색) (Sequential Search)**
  - 주어진 목표 값을 찾아 리스트를 앞에서부터 순차적으로 탐색
  - 목표가 발견된 첫 번째 위치를 리턴; 목표가 발견되지 않으면 음수를 리턴
- **이진 탐색 (Binary Search)**
  - 주어진 배열이 정렬되어 있을 때 주어진 배열을 반복적으로 반으로 나누어가며, 각 단계에서 그 범위에 목표를 포함하는 반쪽에 집중

순차 탐색

- 알고리즘

  입력: 배열과 목표 값 x

  출력: 인덱스 값 i

  후조건: ai = x 또는 모든 aj≠x일 때 i = -n

  1. 0에서 n-1의 각 i에 대해, 단계 2를 수행.
  2. ai = x이면, i를 리턴.
  3. -n을 리턴.

순차 탐색 (구현)

- **LISTING 3: The Sequential Search**

```
1 public class TestSequentialSearch {
3       public static void main(String[] args) {
4           int[] a = {66, 44, 99, 33, 55, 22, 88, 77};
5           System.out.println("search(a," + 55 + "): " + search(a,55));
6           System.out.println("search(a," + 50 + "): " + search(a,50));
7       }
9       public static int search(int[] a, int target) {
10          for (int i = 0; i < a.length; i++)
11              if (a[i] == target) return i;
12          return -a.length;
13      }
14 }
```

순차 탐색 (구현)

- 출력 결과

{66,44,99,33,55,22,88,77}

search(a,55): 4

search(a,50): -8

이진 탐색

- 알고리즘

  입력: 배열과 목표 값 x

  출력: 인덱스 값 i

  선조건: 배열은 정렬되어 있음

  후조건: ai = x; 또는 모든 j<p에 대해서 aj<x이고 모든 j≥p에 대해서 aj>x일 때 i = -p-1

  1. p=0, q=n-1로 놓음.
  2. p≤q이면 단계 2-5를 반복.
  3. i = (p+q)/2로 놓음.
  4. ai = x이면, i를 리턴.
  5. ai < x이면, p = i+1로 놓음; 그렇지 않으면 q = i-1로 놓음.
  6. -p-1을 리턴.

이진 탐색 (구현)

- **LISTING 4: The Binary Search**

```
1 public class TestBinarySearch {
3       public static void main(String[] args) {
4           int[] a = {22, 33, 44, 55, 66, 77, 88, 99};
5           System.out.println("search(a," + 55 + "): " + search(a, 55));
6           System.out.println("search(a," + 50 + "): " + search(a, 50));
7       }
9       static int search(int[] a, int x) {
10          int p = 0, q = a.length-1;
11          while (p <= q) { // search the segment a[p..q]
12                  int i = (p+q)/2; // index of element in the middle
13                  if (a[i] == x) return i;
14                  if (a[i] < x) p = i+1; // search upper half
15                  else q = i-1; // search lower half
16          }
17          return -p-1; // not found
18      }
19 }
```

이진 탐색 (구현)

- 출력 결과

```
search(a,55): 3
search(a,50): -4
```

4. java.util.Arrays 클래스

- 배열 조작을 위한 여러 유틸리티 메소드를 제공
- 모두 static 선언으로 되어 있음 (접두어 "Arrays"에 의해 호출됨)
- 주요 메소드
  - public static int **binarySearch**(double[] a, double x)
  - public static boolean **equals**(double[] a, double[] b)
  - public static void **fill**(double[] a, double x)
  - public static void **fill**(double[] a, int lo, int hi, double x)

    // a[lo], a[lo+1], …, a[hi-1]을 x로 채움
  - public static void **sort**(double[] a)

4. java.util.Arrays 클래스

- **LISTING 5: Using the java.util.Arrays Class**

```
1 public class TestJavaUtilArrays {
3       public static void main(String[] args) {
4           int[] a = {77, 44, 55, 22, 99, 66, 33, 88};
5           print(a);
6           java.util.Arrays.sort(a);
7           print(a);
8           test(a, 55);  test(a, 60);  test(a, 88);  test(a, 90);
12      }
13      static void test(int[] array, int target) {
14          int i = java.util.Arrays.binarySearch(array, target);
15          System.out.print("target="+target+", i="+ i);
16          if (i >= 0) System.out.println("₩ta[" + i + "]=" + array[i]);
17          else System.out.println("₩tInsert " + target +" at a["+(-i-1)+"]");
18      }
20      static void print(int[] a) {  // See lines 21-26 in Listing 1  }
23 }
```

4. java.util.Arrays 클래스

- 출력 결과

a = {77,44,55,22,99,66,33,88}

a = {22,33,44,55,66,77,88,99}

target=55, i=3 a[3]=55

target=60, i=-5 Insert 60 at a[4]

target=88, i=6 a[6]=88

target=90, i=-8 Insert 90 at a[7]

5. 사용자 정의 IntArrays 클래스

- int 배열을 위한 "집에서 만든(homegrown)" IntArrays 클래스: 리스팅 6

5. 사용자 정의 IntArrays 클래스

- **LISTING 6: A User-Defined Utility IntArrays Class**

```
1   public class IntArrays {
2      private static java.util.Random random = new java.util.Random();
4        /** Determines whether the specified array is in ascending order.
7         * @param a the array.
8         * @return true if a[] is in ascending order, otherwise false.   */
10     public static boolean isSorted(int[] a) {
11        for (int i = 1; i < a.length; i++)
12                  if (a[i] < a[i-1]) return false;
13        return true;
14     }
16        /**  Prints all the elements in the specified array. * @param a the array.   */
21     public static void print(int[] a) {
22        System.out.print("{" + a[0]);
23        for (int i = 1; i < a.length; i++)
24                  System.out.print("," + a[i]);
25        System.out.println("}");
26     }
```

5. 사용자 정의 IntArrays 클래스

```
28 /** Returns a new array of n random integers. If range>0, then
30 * the elements will be uniformly distributed in the range
31 * 0 <= a[i] < range; otherwise, they will range over all int values.
34 * @param n the length of the new array.
35 * @param range determines the range of the element values.
36 * @throws IllegalArgumentException.
37 * @return the new array.
   */
39    public static int[] randomInts(int n, int range) {
40        if (n<0 || range<2) throw new IllegalArgumentException();
41        int[] a = new int[n];
42        for (int i=0; i<n; i++)
43          a[i] = random.nextInt(range);
44        return a;
45    }
47 /** Returns a new array of size n that contains the elements of
49 * the specified array a and 0s thereafter.
51 * @param a the array to be copied.
52 * @param n the length of the new array.
53 * @throws IllegalArgumentException.
54 * @return the new array.
   */
```

5. 사용자 정의 IntArrays 클래스

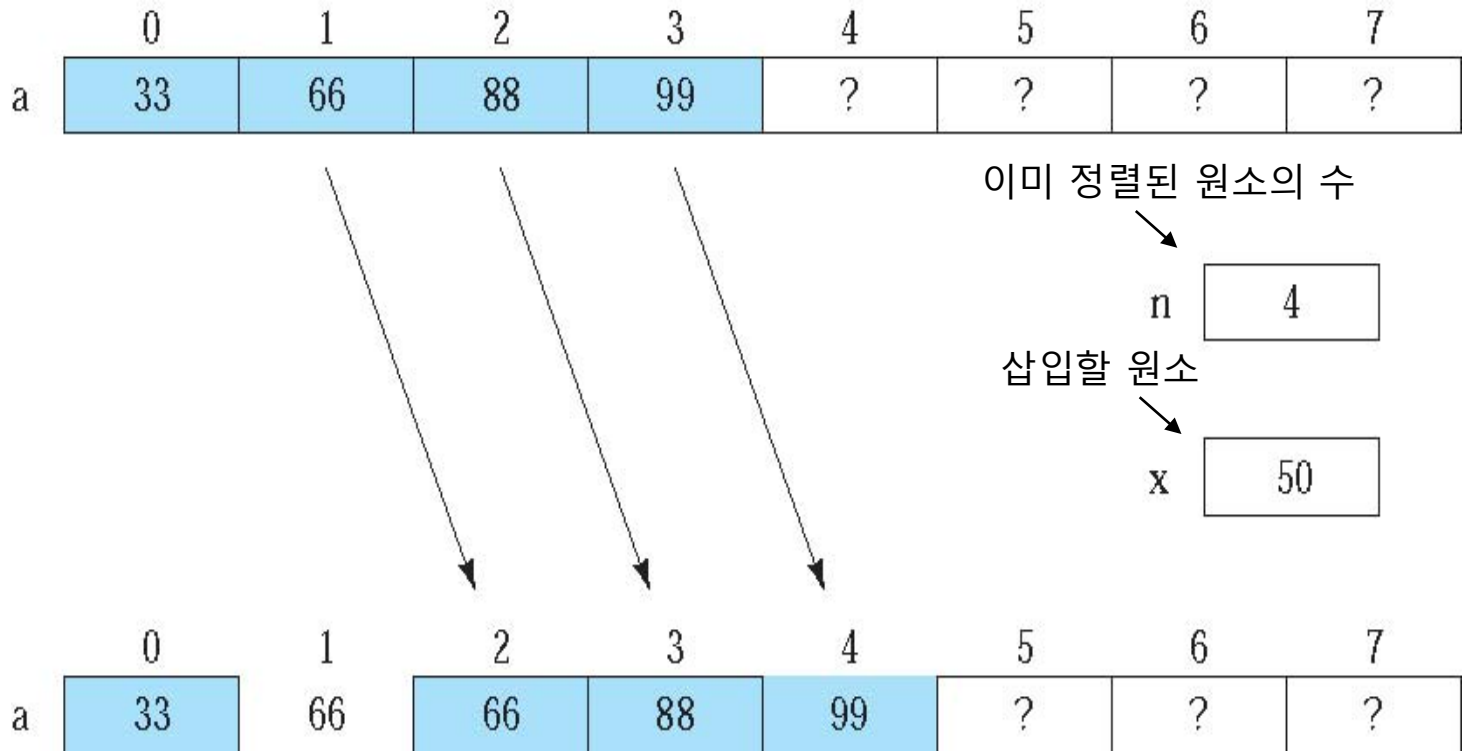
```
56      public static int[] resize(int[] a, int n) {
57          if (n<a.length) throw new IllegalArgumentException();
58          int[] aa = new int[n];
59          System.arraycopy(a, 0, aa, 0, a.length);
60          return aa;
61      }
63  /** Interchanges element i with element j in the specified array.
66  * @param a the array.
67  * @param i index of one element.
68  * @param j index of the other element.
69  */
70      public static void swap(int[] a, int i, int j) {
71          int ai = a[i], aj = a[j];
72          if (ai == aj) return;
73          a[i] = aj;
74          a[j] = ai;
75      }
76  }
```

6. 순서 배열의 유지

- 정렬된 배열에 새 원소 삽입
  - 배열 구현의 경우 새 원소보다 큰 원소들을 모두 이동시켜야 함

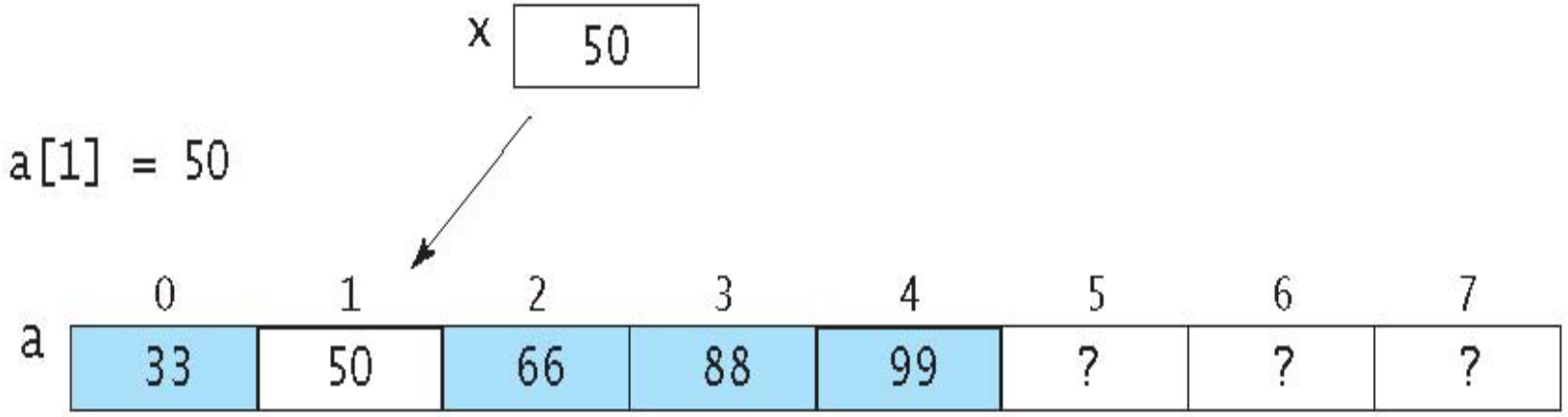
6. 순서 배열의 유지

- 새 원소에 대한 공간 확보



6. 순서 배열의 유지

- x를 올바른 위치에 복사



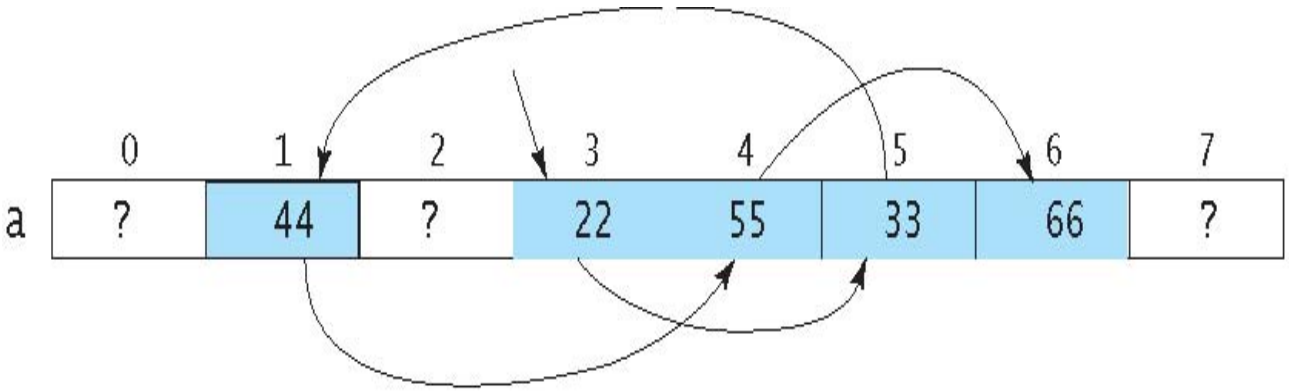
순서 배열로의 삽입

- **LISTING 7: Inserting into an Ordered Array**

```
1 void insert(int[] a, int n, int x) {
2     // preconditions: a[0] <= ... <= a[n-1], and n < a.length;
3     // postconditions: a[0] <= ... <= a[n], and x is among them;
4      int i = 0; // find the first index i for which a[i] > x:
5      while (i < n && a[i] <= x)
6          ++i;
7      // shift {a[i],...,a[n-1]} into {a[i+1],...,a[n]}:
8      System.arraycopy(a, i, a, i+1, n-i);
9      // insert x into a[i]:
10     a[i] = x;
11 }
```

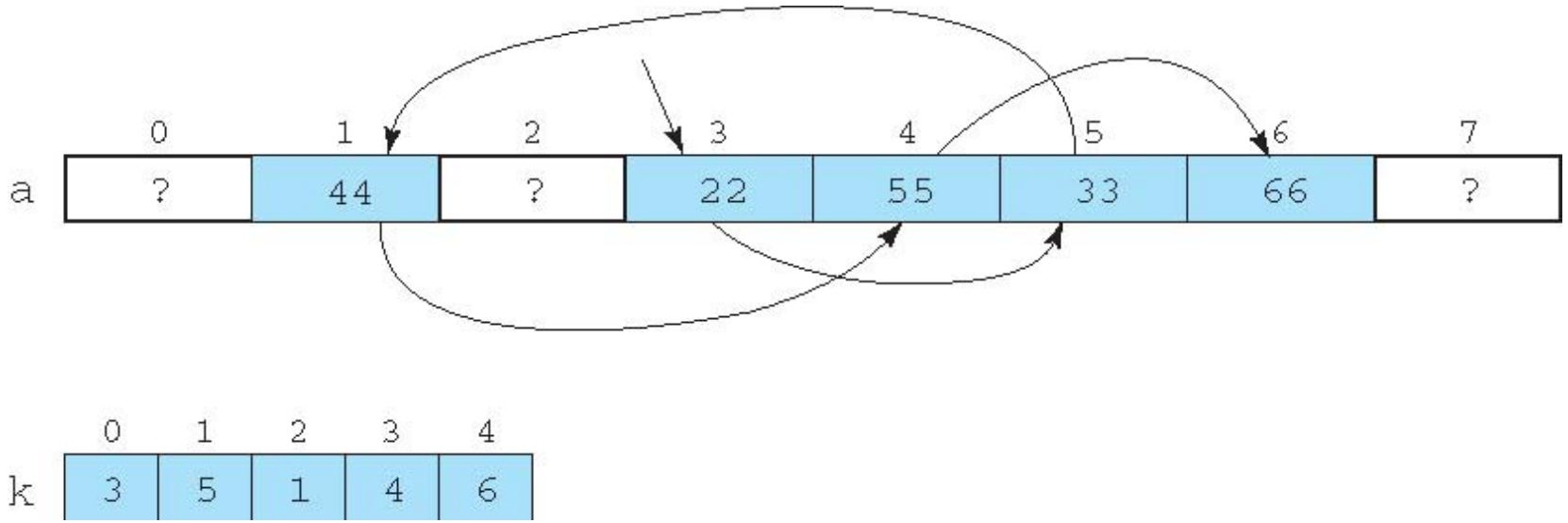
7. 간접 참조

- 원소의 실제 위치를 추적하기 위한 보조 인덱스 배열 사용
  - 추가의 공간(두번째 배열)을 필요로 함
  - 원소를 이동시킬 필요는 없음
  - 원소 {22,33,44,55,66}은 배열 a[]의 임의에 위치에 저장되어 있으며 그 순서는 어떤 보조 메커니즘에 의해서 결정



7. 간접 참조

- 인덱스 배열의 사용



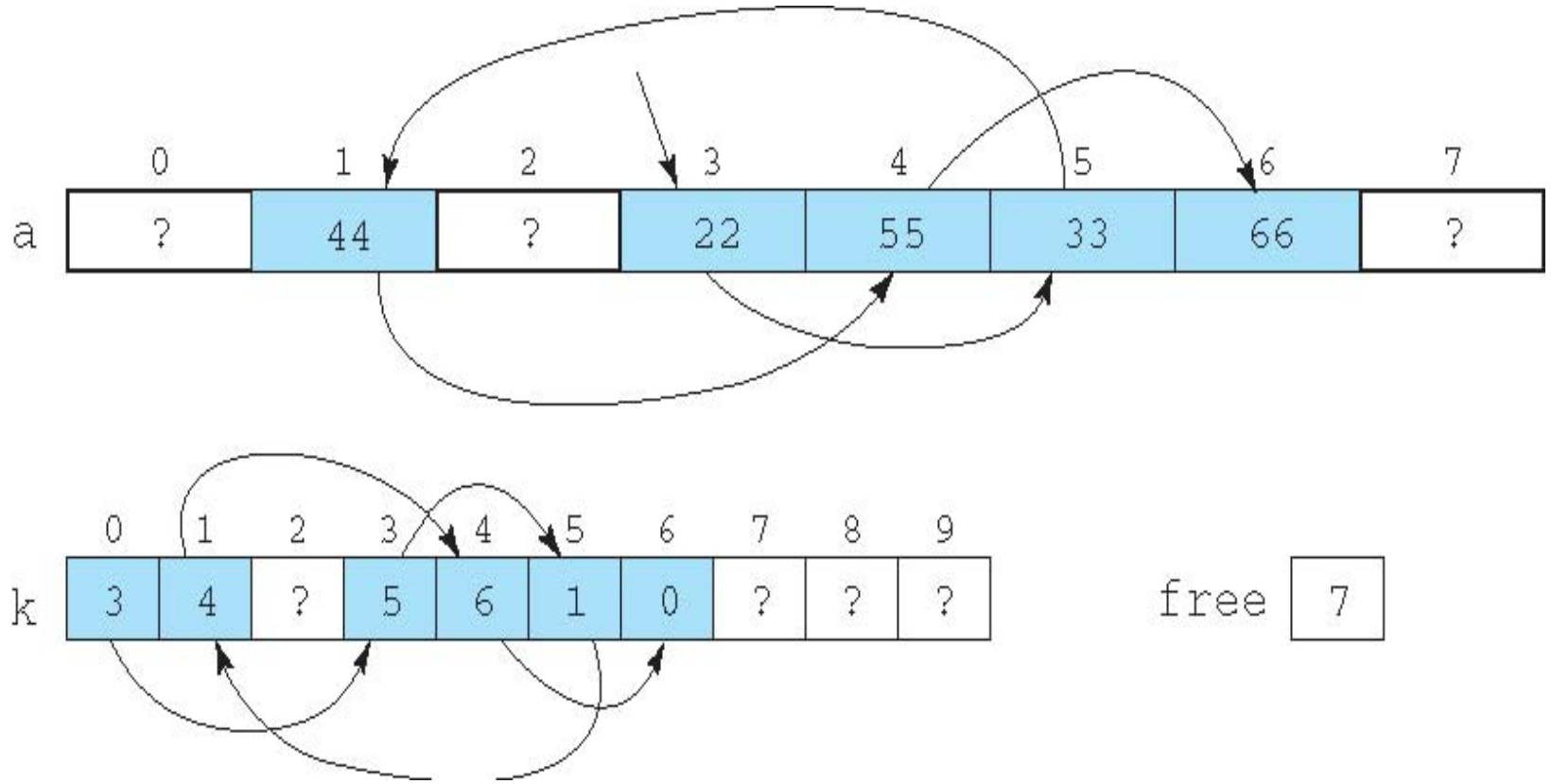
- 원소의 삽입과 삭제시 인덱스 배열 원소들의 이동 필요
- 문제를 a[]에서 k[]로 전가시킨데 불과함

7. 간접 참조

- 개선안
  - 데이터 배열 a[]에서 사용하는 것과 동일한 배열 위치를 인덱스 배열 k[]에서 사용하도록 함
  - 인덱스 배열 k[]는 a[]의 원소의 순서를 추적
  - 추가적인 변수 free는 인덱스 배열 k[]와 데이터 배열 a[] 양쪽의 자유 위치의 인덱스를 저장

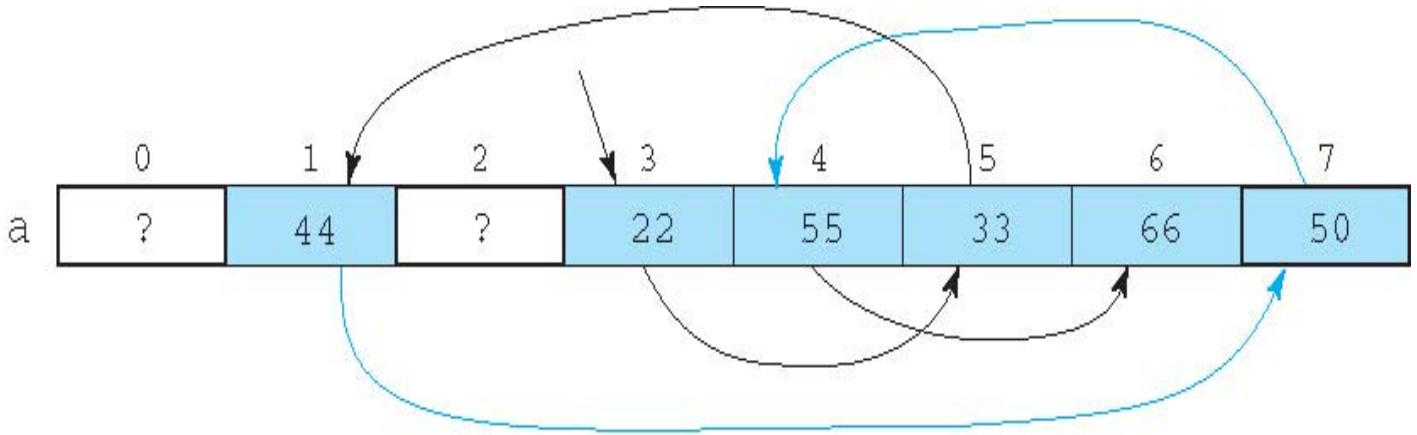
7. 간접 참조

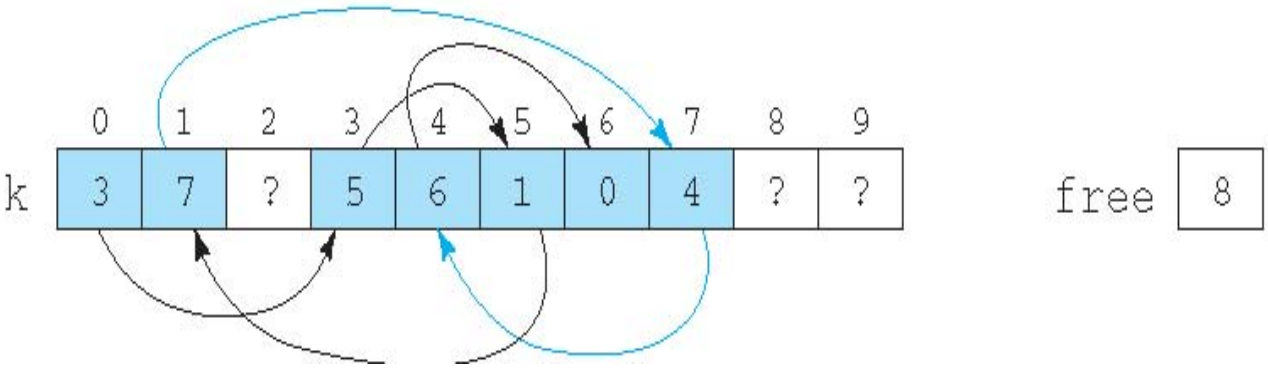
- 인덱스 배열의 사용



7. 간접 참조

- 원소의 삽입





정렬된 배열에 간접적으로 삽입

- **Listing 8: Inserting into an Sorted Array with Indirection**

```
1 void insert(int[] a, int[] k, int x, int free) {
2       int i = 0;
3       while (k[i]!=0 && a[k[i]]<x)
4           i = k[i];
5       a[free] = x;
6       k[free] = k[i];
7       k[i] = free++;
8 }
```

Q & A