자료구조 실습 보고서

[제 4주] : 동전 가방

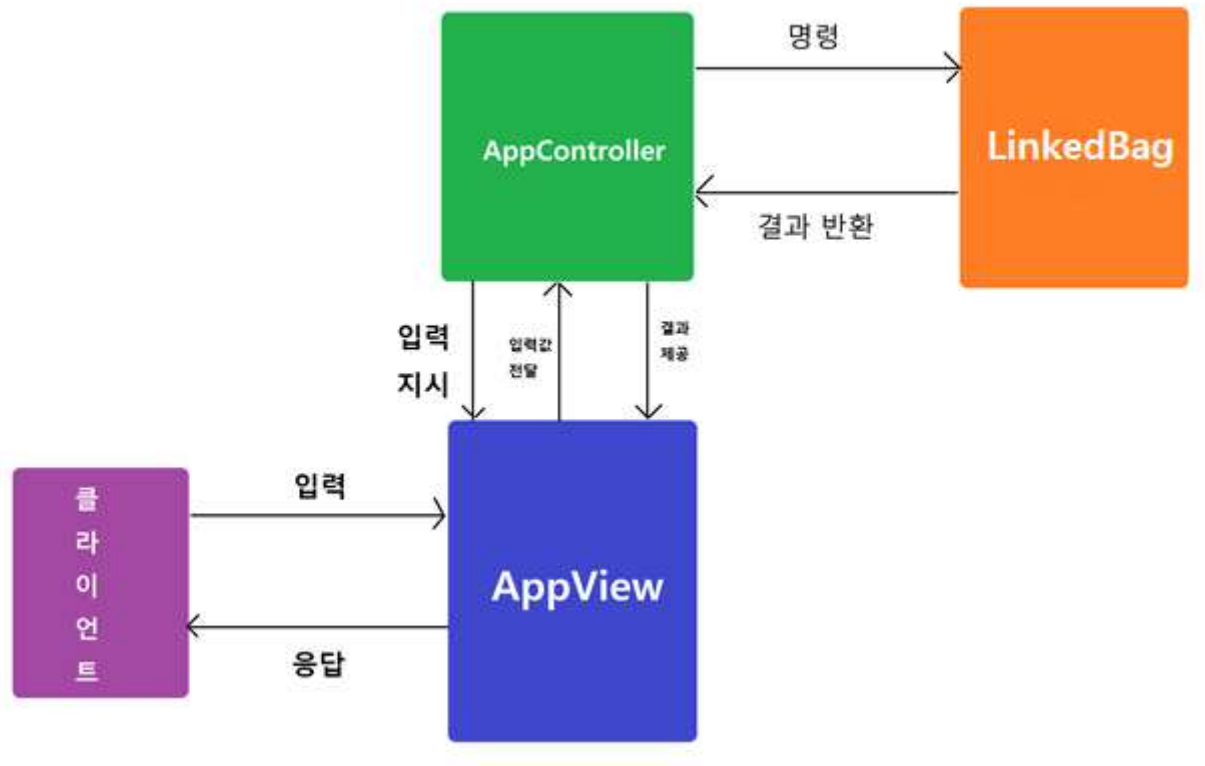
제출일: 2022-03-29(화)

201902708 신동훈

프로그램 설명

1. 프로그램의 전체 설계 구조

간단하게 표현하면 다음과 같다.



MVC(Model - View - Controller)구조를 따르며, 각각의 역할은 다음과 같다.

AppController

AppView에게 입력 요청을 보내며, 그에 대한 응답으로 반환된 입력값을 받아온다.
ArrayBag에게 Coin 객체를 전달하며 보관, 삭제 등의 계산을 요청하고, 결과를 받아온다.
AppView에게 출력 정보를 제공하며, 출력을 요청한다.

AppView

화면(콘솔)에 메시지를 출력하는 역할과, 화면으로부터 입력값을 받아와 다른 객체에게 전달하는 역할을 담당한다.

LinkedBag

Coin 객체를 받아서 저장하거나 삭제하는 등의 로직을 수행한다.

2. 함수 설명(주요 알고리즘/ 자료구조의 설명 포함)

2-1. 사용된 자료구조

LinkedBag은 내부적으로 LinkedNode객체를 사용하여 Coin 객체를 저장한다.
LinkedNode는 내부적으로 자신이 가진 원소인 element 와, 자신과 연결된, 즉 자신 이후의 LinkedNode에 대한 주소를 가지고 있다.
따라서 LinkedBag은 가장 앞부분의 LinkedNode만 알고 있어도, 해당 LinkedNode와 연결된 LinkedNode들을 통해, 모든 원소를 순회할 수 있으며, 원소를 추가할 때도 용량 제한 없이 계속해서 원소를 추가할 수 있다.

2-2. 주요 함수

(Coin과 AppController 등은 이전과 동일하기 때문에 따로 적지는 않고, LinkedNode와 LinkedBag에 대한 함수에 대해서 작성하겠다.)

간단한 설명

LinkedNode 클래스

public **LinkedNode()**: 새로운 LinkedNode 객체를 생성한다.
public **LinkedNode (E givenElement)**: 원소 givenElement 를 갖는 LinkedNode 객체를 생성한다
public **LinkedNode (E givenElement, LinkedNode<E> givenNext)**: 원소 givenElement 와 다음 노드 givenNext 를 갖는 LinkedNode 객체를 생성한다
public **E element()**: LinkedNode 객체에 있는 element 를 얻는다.
public **LinkedNode<E> next()**: LinkedNode 객체의 다음 LinkedNode 객체를 얻는다.
public void **setElement (E newElement)**: LinkedNode 에 있는 element 를 newElement 로 변경한다.
public void **setNext (LinkedNode<E> newNext)**: LinkedNode 객체의 next 를 newNext 로 변경한다.

자세한 구현

```
/**
 * LinkedNode 객체를 생성한다.
 */
public LinkedNode(){
    this._element = null;    //_element 를 null 로 초기화 한다.
    this._next = null;       //_next 를 null 로 초기화 한다.
}

/**
 * 원소 givenElement 를 갖는 LinkedNode 객체를 생성
 * @param givenElement LinkedNode 가 가지고 있을 요소
 */
public LinkedNode (E givenElement){
    this._element = givenElement;        //_element 를 givenElement 로 초기화 한다.
    this._next = null;                   //_next 를 null 로 초기화 한다.
}


/**
 * 원소 givenElement 와 다음 노드 givenNext 를 갖는 LinkedNode 객체를 생성
 * @param givenElement LinkedNode 가 가지고 있을 요소
 * @param givenNext 연결될 다음 노드
 */
public LinkedNode (E givenElement, LinkedNode<E> givenNext){
    this._element = givenElement;    //_element 를 givenElement 로 초기화 한다.
    this._next = givenNext;          //_next 를 givenNext 로 초기화 한다.
}

/**
 * LinkedNode 객체에 있는 element 를 얻는다.
 * @return 가지고 있는 요소
 */
public E element(){
    return this._element;
}
```

```
    /**
     * LinkedNode 객체의 다음 LinkedNode 객체를 얻는다.
     * @return 연결된 다음 Node
     */
    public LinkedNode<E> next(){
        return this._next;
    }

    /**
     * LinkedNode 에 있는 element 를 newElement 로 변경한다.
     * @param newElement 변경할 요소
     */
    public void setElement (E newElement){
        this._element = newElement;
    }

    /**
     * LinkedNode 객체의 next 를 newNext 로 변경한다.
     * @param newNext 연결될 다음 Node
     */
    public void setNext (LinkedNode<E> newNext){
        this._next = newNext;
    }
```

간단한 설명

LinkedBag 클래스

private LinkedNode<E> **head()** : head Node 를 반환한다.
private void **setHead(LinkedNode<E> newHead)** : head Node 를 설정한다
private void **setSize(int newSize)** : size 를 설정한다.
public **LinkedBag()** : size 는 0이고, head 는 null 인 LinkedBag 객체를 생성한다.
public int **size()** : 현재 가방의 사이즈를 반환한다
public boolean **isEmpty()** : 가방이 현재 비어있는지를 확인한다.
public boolean **isFull()** : 가방이 가득 찼는지의 여부를 반환한다.
public boolean **doesContain(E anElement)** : 주어진 원소와 동일한 원소가 가방 속에 존재하는지 확인한다
public int **frequencyOf(E anElement)**: 가방 안에 주어진 원소가 몇 개 있는지를 돌려준다.
public E **elementAt(int order)** : 주어진 순서에 있는 원소를 돌려준다
public boolean **add(E anElement)** : 주어진 원소를 가방에 추가한디.
public boolean **remove(E anElement)** : 주어진 원소를 가방에서 제거한다.
public E **removeAny()** : 가방 속 아무 원소를 삭제한다.
public void **clear()** : 가방을 비운다

자세한 구현

```
/**
 * head Node 를 반환한다.
 * @return head Node
 */
private LinkedNode<E> head(){
    return this._head;
}

/**
 * head Node 를 설정한다
 * @param newHead 설정할 head Node
 */
private void setHead(LinkedNode<E> newHead) {
    this._head = newHead;
}

/**
 * size 를 설정한다.
 * @param newSize 설정할 size
 */
private void setSize(int newSize) {
    this._size = newSize;
}

/**
 * size 는 0이고, head 는 null 인 LinkedBag 객체를 생성한다.
 */
public LinkedBag() {
    this.setSize(0);
    this.setHead(null);
}
```

```
/**
 * 현재 가방의 사이즈를 반환한다
 * @return 현재 가방의 사이즈
 */
public int size() {
    return this._size;
}

/**
 * 가방이 현재 비어있는지를 확인한다.
 * @return 가방이 비었으면 true, 비어있지 않다면 false
 */
public boolean isEmpty() {
    return size()==0;
}

/**
 * 가방이 가득 찼는지의 여부를 반환한다.
 * @return 항상 false 를 반환
 */
public boolean isFull() {
    return false;//LinkedBag 은 용량에 제한이 없으므로 항상 false 를 반환한다.
}



/**
 * 주어진 원소와 동일한 원소가 가방 속에 존재하는지 확인한다
 * @param anElement 포함되어있는지 확인할 원소
 * @return 포함되어 있다면 true, 포함되어있지 않다면 false
 */
public boolean doesContain(E anElement) {

    //같지 않다면 반복할 임시 노드인 temp 를 설정
    LinkedNode<E> currentNode = this.head();

    while (currentNode    != null){
        if(currentNode.element().equals(anElement)){
            return true;
        }
        currentNode = currentNode.next();
    }
    return false;
}
```

```
/**
 * 가방 안에 주어진 원소가 몇 개 있는지를 돌려준다.
 * @param anElement 빈도수를 확인할 원소
 * @return 가방 속에 포함된 주어진 원소의 개수
 */
public int frequencyOf(E anElement) {
    int frequencyCount = 0;

    //반복할 임시 노드인 temp 를 설정
    LinkedNode<E> currentNode = this.head();


    while (currentNode != null){
        if(currentNode.element().equals(anElement)){
            frequencyCount++;
        }
        currentNode = currentNode.next();
    }
    return frequencyCount;
}

/**
 * 주어진 순서에 있는 원소를 돌려준다
 * @param order 찾아낼 순서
 * @return order 위치에 존재하는 요소
 */
public E elementAt(int order){
    if(order == 0){
        return this.head().element();
    }

    int count = 0;
    LinkedNode<E> temp = this.head();

    while (temp.next() != null && count < order){
        temp = temp.next();
        count++;
    }
    return temp.element();
}
```

```
/**
 * 주어진 원소를 가방에 추가한디.
 * @param anElement 가방에 추가할 원소
 * @return 원소 추가 성공 여부를 반환한다.
 *
 */
public boolean add(E anElement){

    LinkedNode<E> newHead = new LinkedNode<>(anElement, this.head());
    this.setHead(newHead);
    setSize(this.size() + 1);

    return true;
}



/**
 * 주어진 원소를 가방에서 제거한다.
 * @param anElement 가방에서 제거할 원소.
 * @return 가방이 비어있거나, 주어진 원소를 가방에서 찾지 못한다면 false 를 반환한다.
 */
public boolean remove(E anElement){
    if(isEmpty()) return false;

    LinkedNode<E> previousNode = null;
    LinkedNode<E> currentNode = this.head();

    while (currentNode != null){
        if(currentNode.element().equals(anElement)){
            previousNode.setNext(currentNode.next());
            this.setSize(this.size()-1);
            return true;
        }
        previousNode = currentNode;
        currentNode = currentNode.next();
    }

    return false;
}
```

```
/**
 * 가방 속 아무 원소를 삭제한다.
 * @return 삭제된 원소
 */
public E removeAny(){
    if(isEmpty()) return null;

    //맨 앞 원소를 지운다.
    E removeElement = this.head().element();
    this.setHead(this.head().next());
    return removeElement;
}


/**
 * 가방을 비운다
 */
public void clear(){
    this.setSize(0);
    this.setHead(null);
}
```

3. 종합 설명

해당 프로그램은 동전을 가방에 넣고 빼는 동작과, 해당 과정들이 끝날 때 가방의 상태를 보여주는 프로그램이다.
앱을 실행하기 위해서는 '_DS04_Main_201902708_신동훈' 이라는 이름의 클래스에 속한 main 메서드를 실행시켜주면 된다.

ArrayBag은 시작할 때 항상 가방의 용량을 정해주었으나 LinkedBag은 용량에 제한이 없으므로 용량을 정해주지 않고 바로 가방에 대하여 수행할 수 있는 행위를 선택지를 제공한다. 가방에 동전을 추가(add)하거나, 가방에서 동전을 제거(remove)할 수 있으며, 입력받은 값의 동전이 가방속에 존재하는지 검색(search)할 수 있고, 동전의 빈도수(frequancy)를 확인할 수 있다.

마지막으로 9를 입력하면 현재 가방에 들어있는 동전의 개수와, 동전 중 가장 큰 값, 모든 동전 값의 크기를 보여주며 프로그램은 종료된다.

프로그램 장단점/특이점

1. 프로그램의 장점

MVC 패턴을 사용하였기에 유지보수성과, 이후 요구사항의 변경 시 코드의 변경의 파급효과가 적을 뿐더러 LinkedBag 클래스는 제네릭을 사용하여 다양한 객체들을 가방에 담을 수 있음과 동시에 잘못된 객체를 담게되었을 때 런타임이 아닌 컴파일 시예 예외가 발생하여 실수를 방지할 수 있다는 장점이 있다. 현재 프로그램에서는 Coin 객체만 담았지만, Coin 뿐만이 아니라 어떤 객체도 담을 수 있기에 프로그램을 확장하는 데 있어서 간편하다는 장점을 가진다.
또한 내부적으로 배열을 사용한 것이 아닌 LinkedNode 객체를 사용하여 가방의 용량 제한이 없다는 장점도 가진다.

2 프로그램의 단점

제네릭을 사용하였긴 했지만, 가방에는 담을 수 없는 것들, 혹은 담아서는 안 되는 것들이 존재한다. 그러나 현재 ArrayBag은 어떠한 객체던 차별 없이 가방에 보관할 수 있다는 문제점을 갖는다.
이를 해결하기 위해서는 간단한 예시로 '담을 수 있는'이라는 속성을 인터페이스로 정의해둔 후, 제네릭의 extends 키워드를 사용하여 상한 제한을 걸 수 있다.

예를 들면 다음과 같다.
'public class ArrayBag<E extends Containable> {...}'

실행 결과 분석

1. 입력과 출력

```
<<< 동전 가방 프로그램을 시작합니다 >>>


? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 1
? 동전 값을 입력하시오: 5
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다.

? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 1
? 동전 값을 입력하시오: 10
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다.

? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 1
? 동전 값을 입력하시오: 20
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다.

? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 1
? 동전 값을 입력하시오: 5
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다.

? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 1
? 동전 값을 입력하시오: 30
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다.

? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 1
? 동전 값을 입력하시오: 5
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다.
```

```
? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 1
? 동전 값을 입력하시오: 100
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다.

? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 2
? 동전 값을 입력하시오: 50
- 주어진 값을 갖는 동전은 가방 안에 존재하지 않습니다.

? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 2
? 동전 값을 입력하시오: 5
- 주어진 값을 갖는 동전 하나가 가방에서 정상적으로 삭제되었습니다.

? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 2
? 동전 값을 입력하시오: 1000
- 주어진 값을 갖는 동전은 가방 안에 존재하지 않습니다.

? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 3
? 동전 값을 입력하시오: 20
- 주어진 값을 갖는 동전이 가방 안에 존재합니다.

? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 3
? 동전 값을 입력하시오: 70
- 주어진 값을 갖는 동전은 가방 안에 존재하지 않습니다.

? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 4
? 동전 값을 입력하시오: 5
- 주어진 값을 갖는 동전의 개수는 2개 입니다.

? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 4
? 동전 값을 입력하시오: 80
- 주어진 값을 갖는 동전의 개수는 0개 입니다.

? 수행하고자 하는 메뉴 번호를 선택하시오 (add: 1, remove: 2, search: 3, frequency: 4, exit: 9) : 9
가방에 들어 있는 동전의 개수: 6
동전 중 가장 큰 값: 100
모든 동전 값의 합: 170

<<< 동전 가방 프로그램을 종료합니다 >>>
```

2. 결과 분석

프로그램을 시작하면 AppController는 AppView의 outputLine 메서드를 통해 프로그램을 시작한다는 문구를 출력시키며, 사용자에게 번호를 받아와서 해당 번호에 해당하는 행동들을 수행한다.
LinkedBag은 용량에 제한이 없기에 용량을 설정하지 않아도 되며, 계속해서 coin을 add 해보아도 오류 없이 계속해서 추가된다.
동전을 제거하거나, 동전이 존재하는지의 여부, 혹은 빈도수를 구할 때도 정삭적으로 작동한다.
추가로 Coin를 5000000개 가방에 담고, elemetAt을 통해 3000000번째 원소를 찾도록 실행을 해보았는데, ArrayBag의 경우에는 실행 시간이 거의 걸리지 않았는데, LinkedBag의 경우에는 4초 정도 소요되었다. ArrayBag은 배열을 사용하기 때문에 임의의 원소에 접근하는 것이 굉장히 빠르지만, LinkedBag의 경우에는 처음 노드부터 특정 순서의 원소까지 주소를 참조하여 접근하는 데 많은 시간이 소요되기 때문이라 생각한다.

생각해 볼 점에 대한 의견

1. ArrayBag 과 LinkedBag 의 구현의 차이점

ArrayBag은 내부적으로 1차원 배열을 사용하였고, LinkedBag은 내부적으로 LinkedNode 객체를 사용하여 구현하였다.
이제부터 배열과 LinkedNode의 차이점에 대해 서술해보겠다.

자바에서는 배열을 생성하는 당시 배열의 자료형과, 배열의 크기를 가지고 배열을 저장할 공간을 미리 할당해 준다.
배열에 담을 요소의 저장공간을 연속적으로 배치해두는 특성 덕분에 배열의 특정한 인덱스의 원소에 접근하는 것은 매우 빠르다. 배열의 시작 주소값에 (특정 인덱스 * 값 하나의 메모리 크기)를 더해주면 바로 특정한 인덱스의 주소가 나오기 때문이다.

그러나 배열은 크기가 고정적이라는 단점이 있다. 위에서 언급했던 것처럼 배열은 생성할 때 배열의 자료형과, 배열의 크기값을 가지고 저장공간을 할당하기에 배열을 생성할 당시 배열의 크기를 지정해 주어야 하며, 이는 이후 수정이 불가능하다.

또한 배열의 특성 상, 배열의 맨 뒤에 원소를 추가하거나, 수정할 때는 인덱스에 접근해서 값을 설정해주면 되기 때문에 매우 빠른 속도로 연산이 가능하다.
그러나 중간 부분에 원소를 추가하거나, 삭제하는 경우, 이후 존재하는 원소들의 위치를 모두 이동시켜 주어야 하므로 시간이 오래 걸리게 된다. 예를 들어 길이가 10000인 배열이 있을 때, 2번째 원소를 제거한 경우 3번째 원소부터 9999번째 원소까지 모두 다 왼쪽으로 한 칸씩 옮겨주어야 하므로 거의 10000번에 가까운 연산이 필요한 식이다.

즉 배열은 원소의 접근에 있어서 빠른 속도를 보여주지만, 배열의 중간에 있는 원소를 삭제하거나, 중간에 원소를 추가하는 경우 오랜 시간이 걸리게 된다.

배열과 달리 LinkedNode는 아래와 같은 특성을 지녔다.
LinkedNode는 내부적으로 자신과 연결된, 즉 자신 이후 순서의 LinkedNode에 대한 주소 정보를 가지고 있으며, 원소를 추가한다면 마지막 LinkedNode에 새롭게 추가하는 원소를 가진 LinkedNode의 주소값을 설정해주면 되기 때문에, 용량에 제한 없이 컴퓨터의 메모리가 버텨주는 한 끝없이 원소의 추가가 가능하다.

배열은 메모리를 연속적으로 사용하기에 특정한 인덱스에 대한 접근이 매우 빠르지만, LinkedNode는 메모리를 연속적으로 사용하지 않기에, LinkedNode를 사용하는 LinkedBag 경우 특정한 순서의 LinkedNode에 접근하기 위해서는 처음부터 순차적으로 노드들을 순회하여야 한다. 따라서 접근할 index의 크기가 커지면 커질수록 시간도 더 오래 걸리게 된다.

원소를 추가하거나 삭제하는 경우, 그 위치에 따라 속도가 달라지는데 다음과 같다.
맨 앞에 원소를 추가하거나, 맨 앞부분의 원소를 삭제하는 경우에는 매우 빠른 속도로 추가와 삭제가 가능하지만, 맨 처음이 아닌 경우, 해당 위치의 크기가 커지면 커질수록 해당 순서의 원소에 접근하는데 시간이 걸리므로 속도가 느려진다.
그러나 해당 원소에 접근한 이후에는 삭제하는 원소를 가진 Node의 이전 노드와 이후 노드를 연결시켜주기만 하면 되기 때문에, 배열과 달리 이후 존재하는 원소들의 순서를 바꿔줄 필요가 없어 매우 빠른 속도로 삭제가 가능하다. 삽입도 마찬가지이다.

정리하면 다음과 같다.

특정 인덱스의 원소에 접근하는 연산은 LinkedBag보다 ArrayBag이 빠르다.
맨 뒤에 원소를 추가하는 경우에는 ArrayBag이 LinkedBag 보다 빠르다.
맨 앞부분의 원소를 삭제하는 경우 LinkedBag이 ArrayBag 보다 빠르다
중간부분의 원소를 삭제하는 경우에는 LinkedBag이 ArrayBag 보다 빠르다.
마지막 원소를 삭제하는 경우 ArrayBag이 LinkedBag 보다 빠르다.