자료구조 실습 보고서

[제04주] 가방 성능 비교

제출일 : 2015 – 03 – 26

201402395 이승희

1. 프로그램 설명서

   1) 주요 알고리즘 / 자료구조 / 기타

      : 2주에 걸쳐 진행되었던 동전 가방의 성능을 비교하였다. ArrayBag과 LinkedBag을 정렬된 경우와 그렇지 않은 경우로 나누어 각각의 성능을 비교하였다. 성능 측정을 할 행위는 데이터의 삽입과 최대값 검색이며 이 프로그램은 MVC패턴으로 구성되었다.

   2) 함수 설명서

      Public PerformanceMeasurement(int aMaxTestSize, int aNumOftests, int aFirstTestSize, int aSizeIncrement) // 생성자

      Public int numOfTests() // 현재 설정된 Test 횟수를 반환한다.

      Public void generateData() // 성능 측정에 필요한 데이터를 생성한다.

      Public TestResult unsortedArrayBag(int testCount) // Unsorted Array로 구현한 List의 성능을 측정한다.

      Public TestResult sortedArrayBag(int testCount) // Sorted Array로 구현한 List의 성능을 측정한다.

      Public TestResult unsortedLinkedBag(int testCount) // Unsorted Linked List로 구현한 List의 성능을 측정한다.

      Public TestResult sortedLinkedBag(int testCount) // Sorted Linked List로 구현한 List의 성능을 측정한다.

   3) 종합 설명서

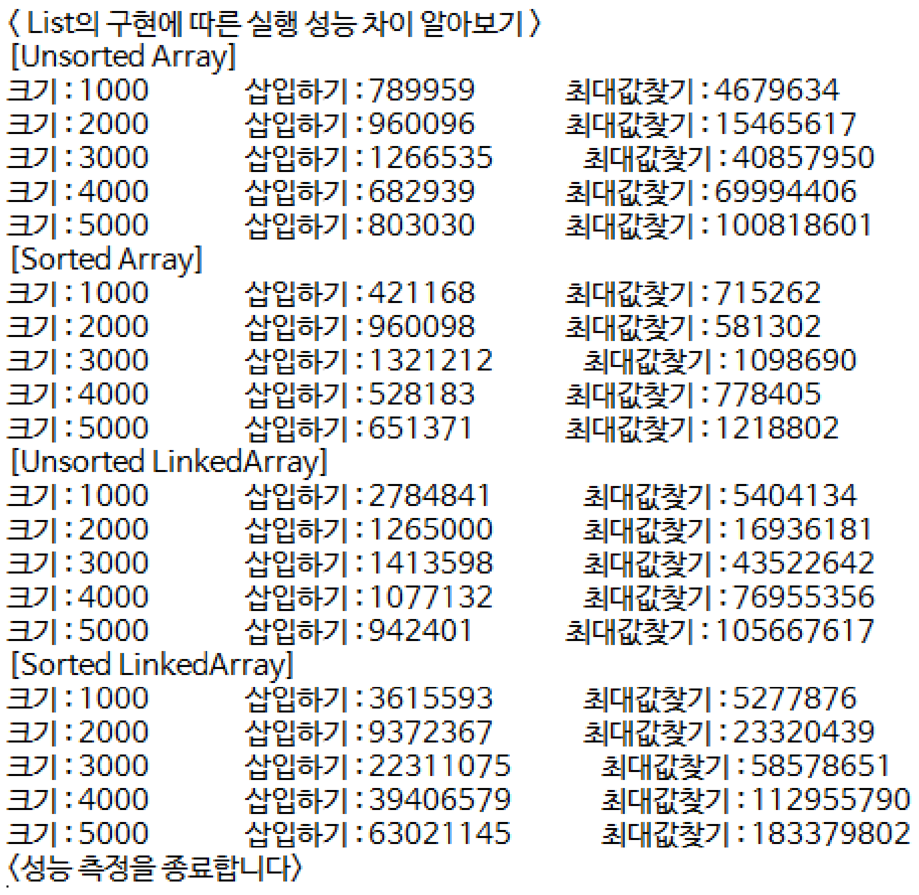
      : 랜덤함수를 이용하여 배열 안에 값을 넣고, 가방의 성능을 비교할 수 있도록 시간 측정 방법을 썼다. 랜덤함수는 Random 객체를 생성한 뒤 반복문을 이용하여 배열 안에 값을 저장하였고, 시간 측정은 nanoTime 메소드를 사용하였다.

2. 프로그램 장단점 분석

   : 실행할 때 마다 시간이 아무렇게 나오고, 시간순으로 정리가 되지 않았다.

3. 실행 결과 분석

1) 입력과 출력



```
〈 List의 구현에 따른 실행 성능 차이 알아보기 〉
[Unsorted Array]
크기 : 1000          삽입하기 : 789959          최대값찾기 : 4679634
크기 : 2000          삽입하기 : 960096          최대값찾기 : 15465617
크기 : 3000          삽입하기 : 1266535          최대값찾기 : 40857950
크기 : 4000          삽입하기 : 682939          최대값찾기 : 69994406
크기 : 5000          삽입하기 : 803030          최대값찾기 : 100818601
[Sorted Array]
크기 : 1000          삽입하기 : 421168          최대값찾기 : 715262
크기 : 2000          삽입하기 : 960098          최대값찾기 : 581302
크기 : 3000          삽입하기 : 1321212          최대값찾기 : 1098690
크기 : 4000          삽입하기 : 528183          최대값찾기 : 778405
크기 : 5000          삽입하기 : 651371          최대값찾기 : 1218802
[Unsorted LinkedArray]
크기 : 1000          삽입하기 : 2784841          최대값찾기 : 5404134
크기 : 2000          삽입하기 : 1265000          최대값찾기 : 16936181
크기 : 3000          삽입하기 : 1413598          최대값찾기 : 43522642
크기 : 4000          삽입하기 : 1077132          최대값찾기 : 76955356
크기 : 5000          삽입하기 : 942401          최대값찾기 : 105667617
[Sorted LinkedArray]
크기 : 1000          삽입하기 : 3615593          최대값찾기 : 5277876
크기 : 2000          삽입하기 : 9372367          최대값찾기 : 23320439
크기 : 3000          삽입하기 : 22311075          최대값찾기 : 58578651
크기 : 4000          삽입하기 : 39406579          최대값찾기 : 112955790
크기 : 5000          삽입하기 : 63021145          최대값찾기 : 183379802
〈성능 측정을 종료합니다〉
```

2) 결과 분석

: 난수를 생성하여 데이터를 저장하고 데이터 크기에 따라서 데이터 삽입의 시간과 최댓값을 찾는데 걸리는 시간을 출력한다. 결과를 보아 Linked보다 Array가 좀더 빠르고 정렬되지 않은 경우보다 정렬된 경우가 더 빠르다.

4. 소스코드

1) DS1_04_201402395_이승희

```java
public class DS1_04_201402395_이승희 {

  public static void main(String[] args) {
    AppController appController = new AppController();
    appController.run();
  }

}
```

2) AppController

```
public class AppController {
  private AppView _appView;
  private PerformanceMeasurement _pml;

  public AppController() {
    this._appView = new AppView();
  }

  private void testUnsortedArrayBag() {
    this.showMessage(MessageID.Notice_UnsortedArrayStart);
    long testCount = 0;
    while (testCount < this._pml.numOfTests()) {
      TestResult testResult = this._pml.unsortedArrayBag(testCount);
      this._appView.outputResult(testResult);
      testCount++;
    }
  }

  private void testSortedArrayBag() {
    this.showMessage(MessageID.Notice_SortedArrayStart);
    long testCount = 0;
    while (testCount < this._pml.numOfTests()) {
      TestResult testResult = this._pml.sortedArrayBag(testCount);
      this._appView.outputResult(testResult);
      testCount++;
    }
  }


private void testUnsortedLinkedArrayBag() {
  this.showMessage(MessageID.Notice_UnsortedLinkedStart);
  long testCount = 0;
  while (testCount < this._pml.numOfTests()) {
    TestResult testResult = this._pml.unsortedLinkedArrayBag(testCount);
    this._appView.outputResult(testResult);
    testCount++;
  }

}

private void testSortedLinkedArrayBag() {
  this.showMessage(MessageID.Notice_SortedLinkedStart);
  long testCount = 0;
  while (testCount < this._pml.numOfTests()) {
    TestResult testResult = this._pml.sortedLinkedArrayBag(testCount);
    this._appView.outputResult(testResult);
    testCount++;
  }
}

public void run() {
  this._pml = new PerformanceMeasurement();
  this.showMessage(MessageID.Notice_StartProgram);
  this._pml.generateData();
  this.testUnsortedArrayBag();
  this.testSortedArrayBag();
  this.testUnsortedLinkedArrayBag();
  this.testSortedLinkedArrayBag();
  this.showMessage(MessageID.Notice_EndProgram);
}
```

```
private void showMessage(MessageID aMessageID) {
  switch (aMessageID) {
  case Notice_StartProgram:
    System.out.println("〈 List의 구현에 따른 실행 성능 차이 알아보기 〉");
    break;
  case Notice_EndProgram:
    System.out.println("〈성능 측정을 종료합니다〉");
    break;
  case Notice_UnsortedArrayStart:
    System.out.println("[Unsorted Array]");
    break;
  case Notice_SortedArrayStart:
    System.out.println("[Sorted Array]");
    break;
  case Notice_UnsortedLinkedStart:
    System.out.println("[Unsorted LinkedArray]");
    break;
  case Notice_SortedLinkedStart:
    System.out.println("[Sorted LinkedArray]");
    break;
  default:
    break;
  }
}
```

3) AppView

```
import java.util.*;

public class AppView {
  private Scanner _scanner;

  public AppView() {
    this._scanner = new Scanner(System.in);
  }

  public void outputResult(TestResult aTestResult) {
    System.out.println("크기 : " + aTestResult.testSize() + "          삽입하기 : "
        + aTestResult.testInsertTime() + "          최대값찾기 : "
        + aTestResult.testFindMaxTime());

  }
}
```

4) MessageID

```
public enum MessageID {
  // Message IDs for Notices:
  Notice_StartProgram,
  Notice_EndProgram,
  Notice_UnsortedArrayStart,
  Notice_SortedArrayStart,
  Notice_UnsortedLinkedStart,
  Notice_SortedLinkedStart,
  // message IDs for Errors:
  Error_WrongMenu

}
```

5) PerformanceMeasurement

```
import java.util.Random;

public class PerformanceMeasurement {
  private static final int MaxTestSize = 10000;
  private static final int NumOfTests = 5;
  private static final int FirstTestSize = 1000;
  private static final int SizeIncremenet = 1000;

  private int _maxTestSize;
  private int _numOfTests;
  private int _firstTestSize;
  private int _sizeIncrement;
  private long [] _testSizes;
  private long [] _data;

  public PerformanceMeasurement() {
    this._maxTestSize = MaxTestSize;
    this._numOfTests = NumOfTests;
    this._firstTestSize = this.FirstTestSize;
    this._sizeIncrement = this.SizeIncremenet;

    this._data = new long [MaxTestSize] ;
    this._testSizes = new long [NumOfTests] ;

    int i = 0;
    while (i < this._numOfTests) {
      this._testSizes [i] = this._firstTestSize + this._sizeIncrement * i;
      i++;
    }
  }


public PerformanceMeasurement (int aMaxTestSize, int aNumOfTests,
    int aFirstTestSize, int aSizeIncrement) {
  this._maxTestSize = aMaxTestSize;
  this._numOfTests = aNumOfTests;
  this._firstTestSize = aFirstTestSize;
  this._sizeIncrement = aSizeIncrement;
}

public void generateData () {
  int i = 0;
  Random random = new Random ();
  while (i < this._maxTestSize) {
    this._data [i] = random.nextInt (this._maxTestSize);
    i++;
  }
}

public int numOfTests () {
  return this._numOfTests;
}

public TestResult unsortedArrayBag (long testCount) {
  UnsortedArrayBag bag;
  long max;
  long testSize;
  long timeForAdd, timeForMax;
  long start, end;
  int i;

  testSize = this._testSizes [(int) testCount] ;
  bag = new UnsortedArrayBag (testSize);
  i = 0;
  timeForAdd = 0;
  timeForMax = 0;
  while (i < testSize) {
    start = System.nanoTime();
    Coin aCoin = new Coin(this._data[i]);
    bag.add(aCoin);
    end = System.nanoTime();
    timeForAdd += (double) (end - start);

    start = System.nanoTime();
    max = bag.maxCoinValues();
    end = System.nanoTime();
    timeForMax += (double) (end - start);

    i++;

  }
  return new TestResult(testSize, timeForAdd, timeForMax);

}

public TestResult sortedArrayBag(long testCount) {
  SortedArrayBag bag;
  long max;
  long testSize;
  long timeForAdd, timeForMax;
  long start, end;
  int i;

  testSize = this._testSizes[(int) testCount];
  bag = new SortedArrayBag(testSize);

  i = 0;
  timeForAdd = 0;
  timeForMax = 0;
  while (i < testSize) {
    start = System.nanoTime();
    Coin aCoin = new Coin(this._data[i]);

    bag.add(aCoin);
    end = System.nanoTime();
    timeForAdd += (double) (end - start);

    start = System.nanoTime();
    max = bag.maxCoinValues();
    end = System.nanoTime();
    timeForMax += (double) (end - start);

    i++;

  }
  return new TestResult(testSize, timeForAdd, timeForMax);

}
```

```
public TestResult unsortedLinkedArrayBag(long testCount) {
  UnsortedLinkedArrayBag bag;
  long max;
  long testSize;
  long timeForAdd, timeForMax;
  long start, end;
  int i;

  testSize = this._testSizes[(int) testCount];
  bag = new UnsortedLinkedArrayBag();

  i = 0;
  timeForAdd = 0;
  timeForMax = 0;
  while (i < testSize) {
    start = System.nanoTime();
    Coin aCoin = new Coin(this._data[i]);

    bag.add(aCoin);
    end = System.nanoTime();
    timeForAdd += (double) (end - start);

    start = System.nanoTime();
    max = bag.maxCoinValue();
    end = System.nanoTime();
    timeForMax += (double) (end - start);

    i++;

  }
  return new TestResult(testSize, timeForAdd, timeForMax);
}

  public TestResult sortedLinkedArrayBag(long testCount) {
    SortedLinkedArrayBag bag;
    long max;
    long testSize;
    long timeForAdd, timeForMax;
    long start, end;
    int i;

    testSize = this._testSizes[(int) testCount];
    bag = new SortedLinkedArrayBag();

    i = 0;
    timeForAdd = 0;
    timeForMax = 0;
    while (i < testSize) {
      start = System.nanoTime();
      Coin aCoin = new Coin(this._data[i]);

      bag.add(aCoin);
      end = System.nanoTime();
      timeForAdd += (double) (end - start);

      start = System.nanoTime();
      max = bag.maxCoinValue();
      end = System.nanoTime();
      timeForMax += (double) (end - start);
      i++;
    }
    return new TestResult(testSize, timeForAdd, timeForMax);
  }
}
```

6) Node

```
public class Node {
  private Coin _element;
  private Node _next;

  public Node() {
    this._element = null;
    this._next = null;
  }

  public Node(Coin anElement) {
    this._element = anElement;
    this._next = null;
  }

  public Node(Coin anElement, Node aNode) {
    this._element = anElement;
    this._next = aNode;
  }

  public Coin element() {
    return this._element;

  }

  public Node next() {
    return this._next;
  }

  public void setElement(Coin anElement) {
    this._element = anElement;
  }

  public void setNext(Node aNode) {
    this._next = aNode;
  }

}
```

7) Coin

```
public class Coin {
  private long _value;

  public Coin() {
    this._value = 0;
  }

  public Coin(long aValue) {
    this._value = aValue;
  }

  public long value() {
    return this._value;
  }

  public void setValue(long aValue) {
    this._value = aValue;
  }

  public boolean equals(Coin aCoin) {
    if (this._value == aCoin._value)
      return true;
    else
      return false;
  }

}
```

8) TestResult

```
public class TestResult {
  private long _testSize;
  private long _testInsertTime;
  private long _testFindMaxTime;

  public TestResult() {
    this._testSize = 0;
    this._testInsertTime = 0;
    this._testFindMaxTime = 0;
  }

  public TestResult(long aTestSize, long aTestInsertTime,
      long aTestFindMaxTime) {
    this._testSize = aTestSize;
    this._testInsertTime = aTestInsertTime;
    this._testFindMaxTime = aTestFindMaxTime;
  }

  public long testSize() {
    return this._testSize;

  }

  public void setTestSize(long aTestSize) {
    this._testSize = aTestSize;
  }

  public long testInsertTime() {
    return this._testInsertTime;
  }
```

```
public void setTestInsertTime(long aTestInsertTime) {
  this._testInsertTime = aTestInsertTime;
}

public long testFindMaxTime() {
  return this._testFindMaxTime;
}

public void setTestFindMaxTime(long aTestFindMaxTime) {
  this._testFindMaxTime = aTestFindMaxTime;
}
```

9) UnsortedArrayBag

```
public class UnsortedArrayBag {

  private static final int DEFAULT_MAX_SIZE = 100;
  private long _maxSize;
  private int _size;
  private Coin _elements[];

  public void UnsortedArraryBag() {
    this._maxSize = DEFAULT_MAX_SIZE;
    this._elements = new Coin[(int) this._maxSize];
    this._size = 0;
  }

  public UnsortedArrayBag(long givenMaxSize) {
    this._maxSize = givenMaxSize;
    this._elements = new Coin[(int) this._maxSize];
    this._size = 0;
  }

  public int size() {

    return _size;
  }

  public boolean isEmpty() {
    if (this._size == 0)
      return true;
    else
      return false;
  }
```

```
public boolean isFull() {
  if (this._size == this._maxSize)
    return true;
  else
    return false;
}

public boolean doesContain(Coin anElement) {
  boolean found = false;
  for (int i = 0; i < this._size && found == true; i++) {
    if (this._elements[i].equals(anElement))
      found = true;
  }
  return found;
}

public int frequencyOf(Coin anElement) {
  int frequencyCount = 0;
  for (int i = 0; i < this._size; i++) {
    if (this._elements[i].equals(anElement))
      frequencyCount++;
  }
  return frequencyCount;

}

public int sumElementValue() {
  int sumValue = 0;
  for (int i = 0; i < this._size; i++) {
    sumValue += this._elements[i].value();
  }

  return sumValue;

}

public boolean add(Coin anElement) {
  if (this.isFull() == true)
    return false;
  else {
    this._elements[this._size] = anElement;
    this._size++;
    return true;
  }
}

public boolean remove(Coin anElement) {
  if (this.isEmpty() == true)
    return false;
  else {
    for (int i = 0; i < this._size; i++) {
      if (this._elements[i].equals(anElement)) {
        this._elements[i] = null;
        for (int j = i; j < this._size - 1; j++) {
          this._elements[j] = this._elements[j + 1];
        }
        this._size--;

      }
    }
    return true;
  }
}
```

```
  public void clear() {
    this._size = 0;
  }

  public long maxCoinValues() {
    long maxValue = 0;
    for (int i = 0; i < this._size; i++) {
      if (maxValue < this._elements[i].value())
        maxValue = this._elements[i].value();
    }
    return maxValue;
  }
```

10) SortedArrayBag

```
 public boolean add(Coin aCoin) {
   if (aCoin.value() > DEFAULT_MAX_SIZE || aCoin.value() < 0) {
     return false;
   }

   if (isFull()) {
     return false;
   } else {
     int index;
     for (index = 0; index < this._size; index++) {
       if (this._elements[index].value() > aCoin.value())
         break;
     }
     int check;
     for (check = index; check < this._size; check++) {
       this._elements[check + 1] = this._elements[check];
     }
     this._elements[index] = aCoin;
     this._size++;
     return true;
   }

 }
```

(이하 UnsortedArrayBag 클래스와 동일)

11) UnsortedLinkedBag

```
public class UnsortedLinkedArrayBag {
  private int _size;
  private Node _head;

  public UnsortedLinkedArrayBag() {
    this._size = 0;
    this._head = null;
  }

  public int size() {
    return this._size;
  }

  public boolean isEmpty() {
    return (this._size == 0);
  }

  public boolean isFull() {
    return false;
  }

  public boolean doesContain(Coin anElement) {
    boolean found = false;

    Node searchNode = this._head;
    while (searchNode != null && !found) {
      if (searchNode.element().equals(anElement))
        found = true;
      else
        searchNode = searchNode.next();
    }
    return found;


}

public int frequencyOf(Coin anElement) {
  int frequencyCount = 0;
  Node currentNode = this._head;
  while (currentNode != null) {
    if (currentNode.element().equals(anElement))
      frequencyCount++;
    currentNode = currentNode.next();
  }
  return frequencyCount;
}

public void clear() {
  this._size = 0;
  this._head = null;
}

public long maxCoinValue() {
  long maxValue = 0;
  Node searchNode = this._head;
  for (int i = 0; i < this._size; i++) {
    if (maxValue < searchNode.element().value())
      maxValue = searchNode.element().value();
    searchNode = searchNode.next();
  }
  return maxValue;
}

public int sumElementValues() {
  int sumValues = 0;
  Node searchNode = this._head;
  for (int i = 0; i < this._size; i++) {
    sumValues += searchNode.element().value();
    searchNode = searchNode.next();
  }
  return sumValues;

}

public Coin any() {
  if (this.isEmpty())
    return null;
  else
    return this._head.element();
}

public boolean add(Coin aCoin) {
  if (this.isFull())
    return false;
  else {
    Node newNode = new Node();
    newNode.setElement(aCoin);
    newNode.setNext(this._head);
    this._head = newNode;
    this._size++;
    return true;
  }

}

public boolean remove(Coin anElement) {
  if (this.isEmpty())
    return false;

   else {
     Node previousNode = null;
     Node currentNode = _head;
     boolean found = false;

     while (currentNode != null && !found) {
       if (currentNode.element().equals(anElement)) {
         found = true;
       } else {
         previousNode = currentNode;
         currentNode = currentNode.next();
       }
     }

     if (!found)
       return false;
     else {
       if (currentNode == this._head)
         this._head = this._head.next();
       else
         previousNode.setNext(currentNode.next());
       this._size--;
       return true;
     }
   }
 }

 public Coin removeAny() {
   if (this.isEmpty())
     return null;
   else {
     Coin removedElement = this._head.element();
```

```
            this._head = this._head.next();
            this._size--;
            return removedElement;
          }
        }

      }
```

12) SortedLinkedBag

```
      public boolean add(Coin aCoin) {
        Node search = this._head;
        Node previous = null;

        while (search != null) {
          if (search.element().value() > aCoin.value()
              || search.element().value() < 0)
            break;
          else {
            previous = search;
            search = search.next();
          }
        }
        if (search == this._head) {
          Node newNode = new Node(aCoin);
          newNode.setNext(this._head);
          this._head = newNode;
        } else {
          Node newNode = new Node(aCoin);
          newNode.setNext(search);
          previous.setNext(newNode);
        }
        this._size++;
        return true;
      }
```

(이하 UnsortedLinkedBag 클래스와 동일)