Revised 1.2.3 Part 1 Array Bugs and Algorithms

Otakar Andrysek

I will look at 17c, Part V #2, 25, 26

Introduction

In the last activity, *1.2.2 Today's Top 40*, you used an *array algorithm* to determine the average cost of a number of songs. In this activity, you will learn other common array algorithms and practice writing loops. Knowing the common array algorithms is an important part of becoming a computer scientist, but these algorithms can cause some of the most difficult and frustrating bugs in a program. Trying to access elements that don't exist, using the wrong index values, iterating too many or too few times, can all cause problems that occur during runtime. You will learn what causes some of these problems and how to fix them.

Materials

- Computer with BlueJ and Android™ Studio
- Android™ tablet and USB cable, or a device emulator

Activity:

Part I: Fix Some Bugs

To begin the activity, first experiment with some of the most common mistakes programmers make when dealing with arrays.

1. Review the most **Common MiAstakes** regarding arrays.
2. Open your BlueJ MediaLib project and create a class called ArrayMistakes. Replace the content created by BlueJ with the following (buggy) content:

```
1  public class ArrayMistakes
2  {
3    public static void main()
4    {
5      int [] nums;
6
7      System.out.print("\nForward");
8      for (int i = 1; i <= nums.length(); i++) {
9         System.out.print(": " + nums[i]);
10     }
11
12     System.out.print("\nBackward" );
13     for (int i = nums.length; i > 0; i--) {
14        System.out.print( ": " + nums[i]);
15     }
16 }
```

This code has many problems with it—some are syntax errors, others occur as runtime errors.

3. Fix the bugs in this code so that the following output occurs:
   Forwards: 6: 9: 14: 19
   Backwards: 19: 14: 9: 6

   Fixed.

Part II: Fix Logic Error 1

In *Activity 1.2.2 Today's Top 40*, you learned to use a for-each loop and a for loop to iterate over an array called songs. In this part, you will look more closely at these two approaches to iteration.

4. Observe each construct side-by-side:

```
1
2
3
4       for (Song s : songs)
{
    s.setTitle("");
}
```

```
1
2
3
4       for (int i=0; i < songs.length; i++)
{
    songs[i].setTitle("");
}
```

5. Compare the body of each loop.
   - In the for-each loop, you access a method using a temporary variable s.
   - In the standard for loop, you access the method with the array name songs and its index i.

Standard for loops are used when you need to know the index of an element in your array.

6. Open your MediaLib project and create a new class called SongFinder. Replace the code generated by BlueJ with the following:

```
1  public class SongFinder
2  {
3     public static int
   findTitle(Song[] songs, String tar
4  for (int i = 0; i < songs.length; i+
   +){
5
   get){
6
        for (int i = 0; i <
7  songs.length; i++){
8
9           word = songs[i].getTitle();
1           if (word.equals(target)) {
0
                return i;
1
1           }
1           else return -1;
2
        }
1
3       return -1;
1     }
4
   }
```

The code above is similar to the findString method you just read about, with these notable differences:

- Line 3: The songs array is passed as the first parameter.
- Line 7: Because you have a for loop, the algorithm uses the new syntax array[index].methodname() as in songs[i].getTitle().
- The algorithm has a bug!

7. In ArrayMediaLib, call SongFinder.findTitle(…) with the title of your third song and show the result, using the following code:

```
1  int index = SongFinder.findTitle(topTenSongs, "Mack the Knife");
2  if (index >=0 ) {
3      System.out.println("Found " + topTenSongs[index].getTitle());
4  }
5  else  {
6      System.out.println("Song not found!");
7  }
```

(If you have a different title, be sure to use it in your call to findTitle.)
The song is not found, even though you know it is there.

To find the bug, a **code walk-through** may help. A code walk-through is the manual process of writing down what each line of code does. The following code walk-through represents what findTitle *should* do, *without* the bug.

Assume target is the title of the third song in the array:

i	word	return value
0	the title of your first song	Target does not match, n/a
1	the title of your second song	n/a
2	the title of your third song	2

8. To see what findTitle *really* does, perform your own walk-through of findTitle with your third song title as target:

i	word	return value
1	*THE TWIST*	*Song not found!*
2	PHYSICAL	Song not found!
3	SMOOTH	Song not found!

Your code walk-through should have shown you where the problem exists in findTitle.

9. Fix the bug and test to see whether your algorithm works using the test cases below. Also, fix any new problems that turn up.

   - When a target can be found in the array of songs.
   - When a target cannot be found.
   - When one or both parameters are null as in SongFinder.findTitle(null,null);

Part III: Fix Logic Error 2

Learning to walk through code to identify logic errors is a very valuable programming skill, and one that gets better with practice. Follow the steps below to explore another common logic error.

10. In your SongFinder class, create a new method getIndexLastDiscount, which introduces another logic error common to arrays and loops.

```
1  public static int getIndexLastDiscount(Song[] songs, double compare){
2      if (songs == null) return -1;
3
4      int found = -1;
5      for (int i = songs.length - 1; i >=0; i--) {
6          if (songs[i].getPrice() < compare) {
7              found = i;
8          }
9          else {
10             found = -1; // to show none found
11         }
12     }
13     return found;
14 }
```

11. Call this method in ArrayMediaLib somewhere after you have applied the $.99 discounted price to some of your songs:

```
1  index = SongFinder.getIndexLastDiscount(topTenSongs, 1.00);
2  if (index >= 0 ) {
3      System.out.println("Discount found " +
   topTenSongs[index].getTitle());
4  }
5  else {
6      System.out.println("No songs are discounted");
7  }
```

The call to `getIndexLastDiscount` indicates a discounted song could not be found, even though you have a few.

What is the logic error in the `getIndexLastDiscount` algorithm? Remember, you can use a code walk-through to help you find the bug.

Incorrect looping bounds.

12. Fix the logic error in `getIndexLastDiscount`.

    Done.

Part IV: Fix Logic Error 3

In Part IV, you will explore index values in an array and gain experience with what happens when the algorithm tries to access an index that is “out of bounds”.

13. Create your last, buggy method in SongFinder:

```
1  /**
2   * Search through all songs, checking for blank titles.
3   * If blank title is found, return -1 to indicate an error.
4   * If all titles are "well-defined", return the last index in the array.
5   */
6  public static int getIndexLastTitle(Song[] songs)
7  {
8      int i;
9      if (songs == null) {
10         return -1;
11     }
12     for (i = 0; i < songs.length; i++) {
13         // skip if no title
14         if (songs[i].getTitle().equals("")) {
15             return -1;
16         }
17     }
18     return i;
19 }
```

14. Read the comments to understand the purpose of this method.

    Understood.

15. Invoke getIndexLastTitle method like you did with your other SongFinder methods.

    Done.

```
System.out.println("--Find last song --");
index = SongFinder.getIndexLastTitle(topTenSongs);
if (index >= 0 ) {
    System.out.println("Last title: " + topTenSongs[index].getTitle());
}
else {
    System.out.println("You have a blank title!");
}
```

16. Run your program to test your algorithm.
    There is a problem in getIndexLastTitle, it causes an “array index out of bounds” exception.

17. To find this type of problem, here are a few suggestions:
    a. First, confirm that i is initialized correctly.
    b. Next, check that the for loop uses songs.length properly.
    c. Then, determine the actual value of array index that is out of bounds. What is the value?

       10.

18. Look at your for loop and try to determine why i is *one more than* the last index of your array.
    In getIndexLastTitle, the condition statement in the for is i < songs.length. At the end of the iteration, this condition *must evaluate to false* so that the loop terminates. A walk-through can make this clearer:

i	result
0	0 is < 10, code in the for loop executes index++ executes
1	1 is < 10, code in the for loop executes index++ executes
2	2 is < 10, code in the for loop executes index++ executes
…	…
9	9 is < 10, code in the for loop executes index++ executes

`i`	result
10	10 is **not** < 10; code in for loop does not execute; the for loop terminates and execution resumes at the line after the for loop's closing }.

The indexing variable `i` ends up 1 greater than the last index of the array. This illustrates an important concept with for loops: *When a for loop visits every element of an array, at the termination of the loop, the value of its index variable is one greater than the last index of the array*.

19. Fix the bug. Be sure to define some test conditions and test your algorithm with them.

    Done.

Part V: Bugs Mean Security Risks

Have you ever heard of a Zero Day vulnerability? It is a term used for a bug in a program that the author doesn't know about. Java, the JVM in particular, has had many bugs, and people are finding them all the time. In 2015 alone, users and programmers found 119 new vulnerabilities. But Java is not alone. QuickTime and iTunes also have bugs and vulnerabilities.

One Java bug led to a large breach called a Trojan horse. Similar to the Greek Trojan horse, users were invited to install friendly- or useful-looking software that actually contained harmful software called **malware**. The malware was discovered by an anti-virus program and the bug was fixed, but not before more than 600,000 computers were infected.

1. Research other types of malware and report on two of the types. Describe how the malware works and how users can prevent malware from harming their systems.

   Adware
   Adware is malware that automatically delivers advertisements to the user. Examples include pop-up ads on websites and advertisements that are displayed by software. To avoid this malware users should browse and download files they know are reputable and avoid clicking 'free' links.

   Ransomware
   Ransomware is a form of malware that essentially holds a computer system captive while demanding a ransom. The system might block some programs or even encrypt the whole hard drive. To avoid this exploit users should constantly update their software.

2. When the bugs are discovered or reported, companies issue a "patch" to fix the problem and users are supposed to install the fix. Many users do not.
   a. Why do you think users do not install patches?

      It takes a lot of time, often notification to update appear at inappropriate times, and people learn to use the current system and don't want anything to change. This is getting much better nowadays where Windows 10 and Mac operating system auto update at night.

   b. Why is it a bad idea not to install a patch?

      The system is still venerable to an exploit. Because the export is well knows at this point there is large chance an unpacked system will be compromised.

Part VI: Common Array Algorithms – Highest and Lowest

Now that you have some practice with arrays (and their problems!), you will write some common array algorithms.

20. Create a new class in your MediaLib project called Algorithms and create a main method as you have done before.

    Done.

21. Copy/reuse code from your ArrayMediaLib class that creates 10 songs in an array and initializes it. Modify the initialization list so that each song has another parameter for rating and rate the songs 1–10. Mix up the ratings so that songs are *not* rated in order 1–10. This will require yet another Song constructor, one that takes a title and a rating as its parameters.

    Done.

22. Use a for or a for-each loop to display all songs with their rating, one song per line.

    Done.

Consider how you might find your favorite song, whose rating would be #1, the lowest value. (I know this should probably be "ranking"...they already had us use a rating where higher meant a better song. Oh well, just go with it.)

You would need to check each song's rating and use a **best-so-far** variable to store the lowest value. You would iterate to check each song and determine whether a particular song's rating is *better* than the best rating so far. This is demonstrated in the **pseudocode** below. Pseudocode is a form of documentation that is structured like a programming language but without the syntax of the language.

```
1  declare and initialize best-so-far
2  for all songs in an array
3      if this song's rating is lower than best-so-far
4          best-so-far = this song's rating
```

Think about what might be a good initial value for bestRating. A very high number might be a good value, but users may choose a scale of 1 to 100, 1 to 1000, or whatever. Rather than trying to predict a specific value, you can initialize bestRating to the rating of the *first* song and then compare all other ratings to it.

23. Continuing in your Algorithms class, write the "find best" algorithm (make it a separate method) and print the best rated song in your array of songs.

    Done.

24. Once you have found the best song, modify the algorithm to print the title.

    Done.

25. ~~Use~~ Copy the same loop to make a method to find the least favorite song and its title. You will need some new variables and choose a good initial value for the worst rating.
    Paste a snip of your bestSong and worstSong methods.
    Paste a snip of your output.

```
Song 0 | Name: THE TWIST | Rating 4
Song 1 | Name: SMOOTH | Rating 7
Song 2 | Name: MACK THE KNIFE | Rating 5
Song 3 | Name: HOW DO I LIVE | Rating 10
Song 4 | Name: PARTY ROCK ANTHEM | Rating 1
Song 5 | Name: I GOTTA FEELING | Rating 4
Song 6 | Name: MACARENA (BAYSIDE BOYS MIX) | Rating 8
Song 7 | Name: PHYSICAL | Rating 5
Song 8 | Name: YOU LIGHT UP MY LIFE | Rating 3
Song 9 | Name: HEY JUDE | Rating 9

The best song is
Name: HOW DO I LIVE | Rating 10

The worst song is
Name: PARTY ROCK ANTHEM | Rating 1
```

```
public static int bestSong(Song[] songs)
{
    int bestSong = -1;
    int position = -1;
    for (int i = 0; i < songs.length; i++)
    {
        if (songs[i].getRating() > bestSong)
        {
            bestSong = songs[i].getRating();
            position = i;
        }
    }
    System.out.println("\nThe best song is");
    System.out.println("Name: " + songs[position].getTitle() + " | Rating " + songs[position].getRating());
    return songs[position].getRating();
}
```

```
public static int worstSong(Song[] songs, int bestSongValue)
{
    int worstSong = bestSongValue;
    int position = -1;
    for (int i = 0; i < songs.length; i++)
    {
        if (songs[i].getRating() < worstSong)
        {
            worstSong = songs[i].getRating();
            position = i;
        }
    }
    System.out.println("\nThe worst song is");
    System.out.println("Name: " + songs[position].getTitle() + " | Rating " + songs[position].getRating());
    return songs[position].getRating();
}
```

26. Which for loop (standard or enhanced) did you use? Why?

I used a for loop. I used a for loop because I needed to use the index inside of the for itself. This document itself states: “Standard for loops are used when you need to know the index of an element in your array.”

That's all you need to turn in. If you are done before everyone else, feel free to work on anything you need to, or help others out. If you are still looking for some challenges, consider these more advanced algorithms I've worked on recently. They should be able to work efficiently with strings millions or even billions of characters long; however, you can always test out the concepts on smaller strings to see if they work. I hope you will see that are within your reach. (My point is not to have you do these -- unless you want to: I mostly want you to see how the algorithms you are able to create now have very relevant uses.)

1. Make an algorithm (in Java or Python) that will determine the length of the shortest subsequence that occurs exactly once in a genomic sequence of length N.

2. Bacterial genomes are often circular. To transform to a linear form, some genome assembly programs will pick a random location in the genome to break the circle. Thus, it is possible that running the same program multiple times we would get different answers, corresponding to different circular rotations of the same string. Make an algorithm (in Java or Python) that will determine if two DNA strings are circular rotations of each other. For example TTGATC is a circular rotation of ATCTTG.

3. We can define a set of distinct substrings of a string S that includes all substrings. However, each repeat is only represented once. For example, for the string S = AATATT, this set is: {A, T, AA, AT, TA, TT, AAT, ATA, TAT, ATT, AATA, ATAT, TATT, AATAT, ATATT, AATATT} You are given a suffix tree of S. Provide the pseudo-code for an algorithm that counts the number of distinct substrings of S. For full credit, this should run in O(n) time. (I'll let you look up what a suffix tree is.)

I will be working on this with Andrew.

Our code will be available here:
https://github.com/otakar-sst/csa-genetics