Homework 2: Procedural Programming

CSC 600-01 Programming Languages

Spring 2017

Emilio Quiambao

3 / 1 / 2017

All programs are written in Java

(1) Plateau Program

My idea for the Plateau program is to traverse the array while checking the current index at each iteration. When checking the current index, we check if its value equates the previous integer or not. If it does, we increase the “count” of that integer by 1. If it does not, the count resets back to 1. If the current count of the integer surpasses the previous highest count, the highest count is replaced by the current one. Since the array is already sorted, we only need to traverse the array once, which in return, becomes simple and efficient for us. After traversing the array, the maxlen() function will return the highest count. I believe this is considerably a good solution since it is very simple, straightforward, efficient and not very complex. It is also minimal and only keeps the needed data for the end result of the function and nothing else.

```
public class Plateau
{
   public static void main(String[] args)
   {
       int arr[] = {1, 1, 1, 2, 3, 3, 5, 6, 6, 6, 6, 7, 9};
       System.out.println(maxlen(arr));
   }

   static int maxlen(int[] arr)
   {
       int highestCount = 0;
       //the final result, the highest count of identical numbers

       int currentCount = 0;
       //the current count of the integer we are counting in the array

       int value = 0;
       //the current value of the integer we are counting

       for (int i = 0; i < arr.length; i++)
       {
           if (arr[i] == value) currentCount++;
           //if number is same, increase the current counter

           else
           //if number is different...
           //change current value, reset current counter, check for highest count;
           {
               value = arr[i];
               if (currentCount > highestCount) highestCount = currentCount;
               currentCount = 1;
           }
       }
       return highestCount;
   }
}
```

Output:

```
4

Process finished with exit code 0
```

(2) Parabolic Approximation

Using method (a), we will have to deal with three linear equations. By using linear algebraic techniques such as the use of matrices, we can efficiently solve the equations and find the needed coefficients. Using this approach, we first need to create a matrix that will hold each of the three coordinates. We then have to also find the inverse of that matrix. In order to find the inverse of a 3x3 matrix we can find its determinant using Sarrus' rule. Since we will always deal with three coordinates, we can repeatedly use the same formula to find a 3x3 matrix's determinant. When multiplying the inverse of the matrix with a 3x1 matrix holding the y values, it will result in the needed coefficients for the parabolic equation. Since this program is restricted to the use of three points, the structure and needed data is refined and the computations are lessened by the use of mathematical formulas.

```
public class parabolicApproximation
{

   public static void main(String[] args)
   {
       for(int i = 1; i <= 40; i++)
       {
           System.out.printf("(%2d,%7.3f)\t", i, y(0,0, 12,16, 42,120, i));
           if(i % 5 == 0) System.out.println();
       }
   }

   static double y(double x1, double y1,
                   double x2, double y2,
                   double x3, double y3, double x)
   {
       //3x3 matrix A
       double[][] mA = {{x1*x1, x1, 1},
                        {x2*x2, x2, 1},
                        {x3*x3, x3, 1}};

       //3x1 matrix B
       double[] mB = {y1, y2, y3};

       //array of coefficients a, b, and c for parabolic equation
       double[] paraEq = new double[3];
```

```
      //the determinant of matrix A
      double d = mA[0][0] * mA[1][1] +
                 mA[0][1] * mA[2][0] +
                 mA[1][0] * mA[2][1] –
                 mA[2][0] * mA[1][1] –
                 mA[2][1] * mA[0][0] –
                 mA[1][0] * mA[0][1];

       //the inverse of matrix A
       double[][] invA = {{(mA[1][1] - mA[2][1]) /  d,
                           (mA[0][1] - mA[2][1]) / -d,
                           (mA[0][1] - mA[1][1]) /  d},

                          {(mA[1][0] - mA[2][0]) / -d,
                           (mA[0][0] - mA[2][0]) /  d,
                           (mA[0][0] - mA[1][0]) / -d},

                          {(mA[1][0] * mA[2][1] - mA[1][1] * mA[2][0]) /  d,
                           (mA[0][0] * mA[2][1] – mA[0][1] * mA[2][0]) / -d,
                           (mA[0][0] * mA[1][1] - mA[0][1] * mA[1][0]) /  d}};

       //we now multiply inverse of matrix A with matrix B
       for(int i = 0; i < 3; i++)
           for(int j = 0; j < b.length; j++) eq[i] += ai[i][j] * b[j];

       //parabolic equation ( y(x) =ax^2 + bx + c )
       return eq[0]*x*x + eq[1]*x + eq[2];
   }
}
```

Output:

```
(  1,  0.775)(  2,  1.651)(  3,  2.629)(  4,  3.708)(  5,  4.889)
(  6,  6.171)(  7,  7.556)(  8,  9.041)(  9, 10.629)( 10, 12.317)
( 11, 14.108)( 12, 16.000)( 13, 17.994)( 14, 20.089)( 15, 22.286)
( 16, 24.584)( 17, 26.984)( 18, 29.486)( 19, 32.089)( 20, 34.794)
( 21, 37.600)( 22, 40.508)( 23, 43.517)( 24, 46.629)( 25, 49.841)
( 26, 53.156)( 27, 56.571)( 28, 60.089)( 29, 63.708)( 30, 67.429)
( 31, 71.251)( 32, 75.175)( 33, 79.200)( 34, 83.327)( 35, 87.556)
( 36, 91.886)( 37, 96.317)( 38,100.851)( 39,105.486)( 40,110.222)

Process finished with exit code 0
```

(3) Array Processing

I believe the most efficient way of solving this problem requires going through the array at least twice: the first traversal is finding the three largest different integers and the second traversal is reducing the array. I do not think it is possible to traverse it less than two times since reducing the array requires examination first. In Java, you cannot remove an element in a basic array or reduce a basic array's size, so in this case, a new reduced array must be created by inputting the values of the original array while ignoring the three greatest different integer values that we have found. An easier and possibly more efficient way is either using ArrayList to remove elements of the original array or using a different language that passes by pointers instead of references. Other than that, I believe this method is a good solution since it passes the array the least times possible and it is not hard to understand.

```
public class arrayProcessing
{
    public static void main(String[] args)
    {
        int[] arr = {9,1,1,6,7,1,2,3,3,5,6,6,6,6,7,9};

        //prints original array and its size
        System.out.print("Original:\n");
        System.out.print("[ ");
        for(int i = 0; i < arr.length; i++) {System.out.print(arr[i] + " ");}
        System.out.print("]\n");
        System.out.print("n = " + arr.length);
        System.out.println("\n");

        arr = reduce(arr);

        //prints new reduced array and its size
        System.out.print("Reduced:\n");
        System.out.print("[ ");
        for(int i = 0; i < arr.length; i++) {System.out.print(arr[i] + " ");}
        System.out.print("]\n");
        System.out.print("n = " + arr.length);
        System.out.println();
    }

    static int[] reduce(int[] arr)
    {
        //where we will store the values of the greatest integers
        int g1 = 0;
        int g2 = 0;
        int g3 = 0;

        //new reduced array for correct size
        int[] reducedArr;

        //counter when we go through array second time
        int counter = 0;

        //find 3 greatest different integers and store them
        for(int i = 0; i < arr.length; i++)
        {
            if(arr[i] > g1)
                g1 = arr[i];
```

```
            else if (arr[i] > g2 && arr[i] != g1)
                g2 =arr[i];
            else if (arr[i] > g3 && arr[i] != g1 && arr[i] != g2)
                g3 = arr[i];
        }

        //reduce and rewrite original array, ignoring three greatest different integers
        for(int i = 0; i < arr.length; i++)
        {
            if(arr[i]!= g1 && arr[i]!= g2 && arr[i]!= g3)
            {
                arr[counter] = arr[i];
                counter++;
            }
        }

        //initialize size of new array
        reducedArr = new int[counter];

        //input rewritten original array into new array with correct size
        for(int i = 0; i < reducedArr.length; i++) reducedArr[i] = arr[i];
        return reducedArr;
    }
}
```

Output:

```
Original:
[ 9 1 1 6 7 1 2 3 3 5 6 6 6 6 7 9 ]
n = 16

Reduced:
[ 1 1 1 2 3 3 5 ]
n = 7

Process finished with exit code 0
```

(4) Integer Plot Function

This problem is simple. It is just a matter of defining each number from 1 through 9 and its 7x7 format when being read. This program just needs a reader that will match the number with its big 7x7 counterpart and the format of each counterpart. So first, we just create each big number in an array and how its formatted in 7x7 size by using a two dimensional char array, then create a reader that will read the inputted number and associate its bigger format when being printed. We also have to restrict the inputted number to ten digits since ints cannot go beyond that. I find this solution good since after we create each number's format, calling back the big numbers array is simple and it is now just a matter of reading and matching the inputted number.

```
public class integerPlotFunction
{
    public static void main(String[] args)
    {
        bigInt(1234567890);
    }

    static void bigInt(int n)
    {
        //this will hold the formatting of each big number from 1 to 9
        char[][][] bigNums = new char[10][0][0];

        bigNums[0] = new char[][] {
                {'@','@','@','@','@','@','#'},
                {'@','@',' ',' ',' ',' ','@','#'},
                {'@','@',' ',' ',' ',' ','@','#'},
                {'@','@',' ',' ',' ',' ','@','#'},
                {'@','@',' ',' ',' ',' ','@','#'},
                {'@','@',' ',' ',' ',' ','@','#'},
                {'@','@','@','@','@','@','#'}};

        bigNums[1] = new char[][] {
                {'@','@','@','@','@','#',' '},
                {' ','@','@','@','@','#',' '},
                {' ','@','@','@','@','#',' '},
                {' ','@','@','@','@','#',' '},
                {' ','@','@','@','@','#',' '},
                {' ','@','@','@','@','#',' '},
                {'@','@','@','@','@','@','#'}};

        bigNums[2] = new char[][] {
                {'@','@','@','@','@','@','#'},
                {'@','#',' ',' ','@','@','#'},
                {' ',' ',' ',' ','@','@','#'},
                {'@','@','@','@','@','@','#'},
                {'@','@','@','#',' ',' ',' '},
                {'@','@','@','#',' ',' ',' '},
                {'@','@','@','@','@','@','#'}};

        bigNums[3] = new char[][] {
                {'@','@','@','@','@','@','#'},
                {'@','#',' ',' ',' ','@','@','#'},
                {' ',' ',' ',' ',' ',' ','@','@','#'},
                {' ','@','@','@','@','#','#'},
                {' ',' ',' ',' ',' ',' ','@','@','#'},
                {'@','#',' ',' ',' ',' ','@','@','#'},
                {'@','@','@','@','@','@','#'}};


        bigNums[4] = new char[][] {
                {'@','@','#',' ',' ','@','@','#'},
                {'@','@','#',' ',' ','@','@','#'},
                {'@','@','#',' ',' ','@','@','#'},
```

```
        {'@','@','#',' ','@','@','#'},
        {'@','@','@','@','@','@','#'},
        {' ',' ',' ',' ','@','@','#'},
        {' ',' ',' ',' ','@','@','#'}};

bigNums[5] = new char[][] {
        {'@','@','@','@','@','@','#'},
        {'@','@','#',' ',' ',' ',' '},
        {'@','@','@','@','@','@','#'},
        {' ',' ',' ',' ','@','@','#'},
        {' ',' ',' ',' ','@','@','#'},
        {'@','@','#',' ','@','@','#'},
        {'@','@','@','@','@','@','#'}};

bigNums[6] = new char[][] {
        {'@','@','@','@','@','@','#'},
        {'@','@','#',' ','@','@','#'},
        {'@','@','#',' ',' ',' ',' '},
        {'@','@','@','@','@','@','#'},
        {'@','@','#',' ','@','@','#'},
        {'@','@','#',' ','@','@','#'},
        {'@','@','@','@','@','@','#'}};

bigNums[7] = new char[][] {
        {'@','@','@','@','@','@','#'},
        {'@','@','@','@','@','@','#'},
        {' ',' ',' ','@','@','@','#'},
        {' ',' ',' ','@','@','@','#'},
        {' ',' ',' ','@','@','@','#'},
        {' ',' ',' ','@','@','@','#'},
        {' ',' ',' ','@','@','@','#'}};

bigNums[8] = new char[][] {
        {'@','@','@','@','@','@','#'},
        {'@','#',' ',' ',' ','@','#'},
        {'@','@','@','@','@','@','#'},
        {'@','@','@','@','@','@','#'},
        {'@','#',' ',' ',' ','@','#'},
        {'@','@','@','@','@','@','#'},
        {'@','@','@','@','@','@','#'}};

bigNums[9] = new char[][] {
        {'@','@','@','@','@','@','#'},
        {'@','#',' ',' ',' ','@','#'},
        {'@','#',' ',' ',' ','@','#'},
        {'@','@','@','@','@','@','#'},
        {' ',' ',' ',' ',' ','@','#'},
        {'@','#',' ',' ',' ','@','#'},
        {'@','@','@','@','@','@','#'}};
```

```
        //how many digits to print
        int size = 0;

        //ints are restricted to 10 digits
        //this holds the value of each digit from the input
        int[] nums = new int[10];

        //find digit length
        //store each digit value
        for( ; n > 0; n /= 10, size++) nums[size] = n % 10;

        //print out the digits

        //i refers to the 7x7 format of the big nums
        //j refers to the size of the inputteed digist
        for(int i = 0; i < 7; i++)
        {
            for(int j = size - 1; j >= 0; j--)
            {
                System.out.print(bigNums[nums[j]][i]);
                System.out.print("  ");
            }
            System.out.print("\n");
        }
    }
}
```

Output:

```
 @@@@@#   @@@@@@#  @@@@@@#  @@# @@#  @@@@@@#  @@@@@@#  @@@@@@#  @@@@@@#  @@@@@@#  @@@@@@#
  @@@@#   @# @@@#  @#  @@#  @@# @@#  @@#      @@# @@#  @@@@@@#  @#   @#  @#   @#  @@   @#
  @@@@#      @@@#     @@#  @@# @@#  @@@@@@#  @@#         @@@#  @@@@@@#  @#   @#  @@   @#
  @@@@#   @@@@@@#   @@@@##  @@# @@#      @@#  @@@@@@#     @@@#  @@@@@@#  @@@@@@#  @@   @#
  @@@@#   @@@#         @@#  @@@@@@#      @@#  @@# @@#     @@@#  @#   @#       @#  @@   @#
  @@@@#   @@@#     @#  @@#      @@#  @@# @@#  @@# @@#     @@@#  @@@@@@#  @#   @#  @@   @#
@@@@@@#  @@@@@@#  @@@@@@#      @@#  @@@@@@#  @@@@@@#     @@@#  @@@@@@#  @@@@@@#  @@@@@@#

Process finished with exit code 0
```

(5) Iteration vs Recursion

From prior knowledge and from the guest lecturer on this subject we have had not too long ago, I have made the hypothesis that the iterative method will be faster. Making two simple binary search algorithms, one being an iterative and the other being recursive, I have found my hypothesis to be correct as the recursive approach has been almost a second (sometimes even more) slower after multiple runs. I did not know how to find the maximum number of elements allowed for an array since Java varies from system to system. I inputted very large numbers until I hit a MemoryError and then backtracked, replacing and testing different numbers. After getting run times of more than twenty seconds each, I decided the number

is reasonably large to test. While I find recursion easier to understand and can somewhat condense the amount of lines of code needed, the work needed for it to run is much more than an iterative approach. I find that recursive methods are easier on the eyes for humans, but iterative (while sometimes messy) can be much more efficient for machines. Recursive algorithms just make too many function calls. Iteration vs Recursion is almost negligible for smaller computations, but when increasing the numbers and size of the problem, recursive tends to get far slower while an iterative approach is far superior, beating recursion by seconds.

```
public class iterationVsRecursion
{
    public static void main(String[] args)
    {
        //largest possible array size
        int n = 178000009;
        int[] arr = new int[n];

        //filling array with values
        for (int i = 0; i < arr.length; i++) arr[i] = i;

        //testing runtimes
        int k = 1;

        long sec;
        long dec;
        long t;

        //ITERATION TEST
        //time starts
        t = System.nanoTime();
        for(int j = 0; j < k; j++)
            for(int p = 0; p < n; p++)
                if(iterativeBinSearch(arr, n-1, p) != p) System.out.print("\nERROR");

        //time ends
        t= System.nanoTime() - t;
        sec = t / 1000000000;
        dec = t % 1000000000;
        System.out.print("Iterative runtime: " + sec + "." + dec + " seconds\n");

        //RECURSIVE TEST
        //time start
        t = System.nanoTime();
        for(int j = 0; j < k; j++)
            for(int p = 0; p < n; p++)
                if(recursiveBinSearch(arr, 0, n-1, p) != p) System.out.print("\nERROR");

        //time ends
        t = System.nanoTime() - t;
        sec = t / 1000000000;
        dec = t % 1000000000;
        System.out.print("Recursive runtime: " + sec + "." + dec + " seconds\n");
    }
```

```
    //ITERATIVE SEARCH
    static int iterativeBinSearch(int[] arr, int n, int x)
    {
        int min = 0, max = n;

        while (max >= min)
        {
            int mid = (min + max) / 2;

            //x is in the middle
            if (arr[mid] == x) return mid;

            //search in upper half
            else if (arr[mid] < x) min = mid + 1;

            //search in lower half
            else max = mid - 1;
        }

        return -1; //x does not exist in the array
    }

    //RECURSIVE SEARCH
    static int recursiveBinSearch(int[] a, int min, int max, int x)
    {

        if (max < min) return -1; //x does not exist in the array
        else
        {
            int mid = (min + max) / 2;

            //search upper half
            if (a[mid] < x) return recursiveBinSearch(a,mid+1,max,x);

            //search lower half
            else if (a[mid] > x) return recursiveBinSearch(a, min, mid - 1, x);

            else return mid; //x is in the middle
        }
    }

}
```

Output:

```
Iterative runtime: 20.837871224 seconds
Recursive runtime: 21.300905283 seconds

Process finished with exit code 0
```