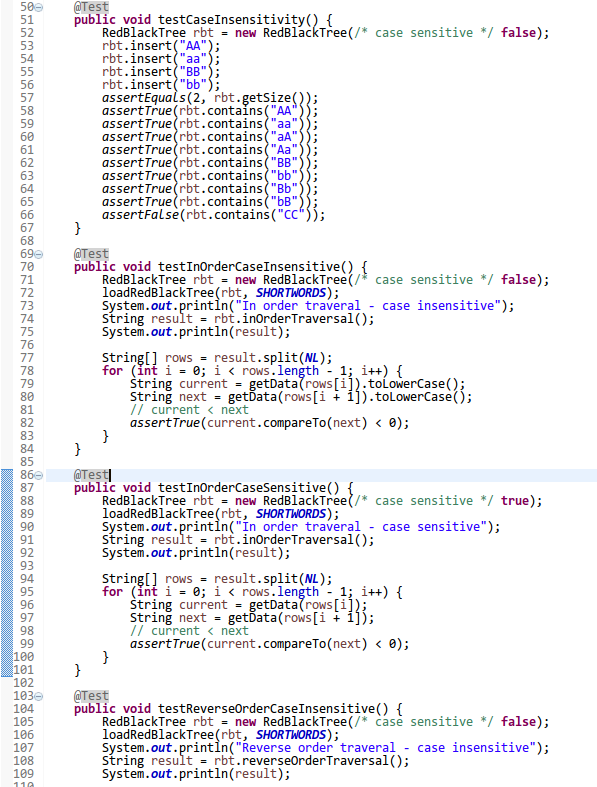
# Part 1. Red Black Tree

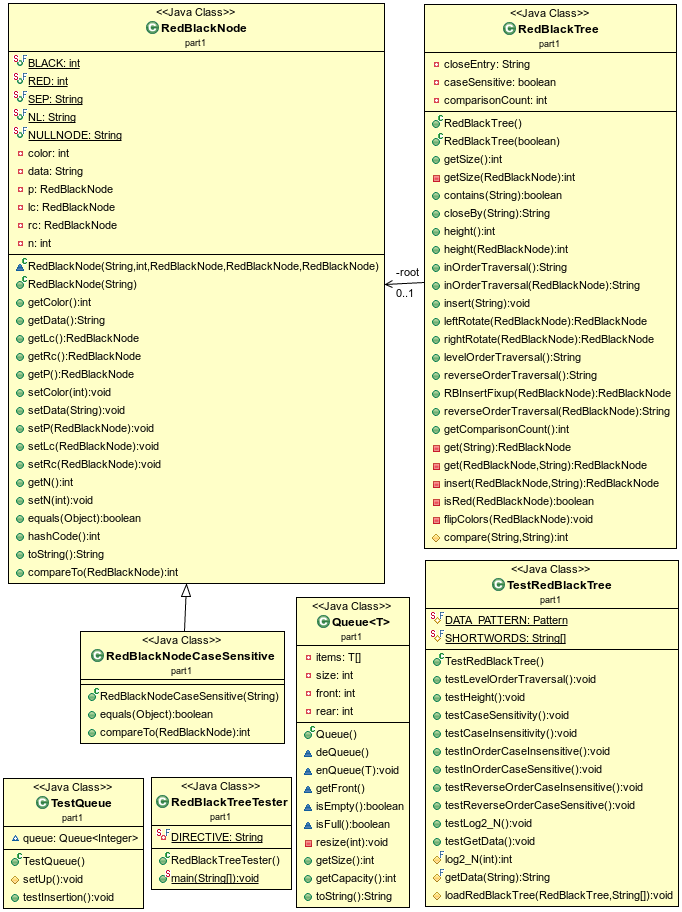
The [specification](https://www.andrew.cmu.edu/user/mm6/95-771/examples/RedBlackTreeTesterProject/dist/javadoc/index.html) requires that inOrderTraversal(), reverseOrderTraversal, and levelOrderTraversal() methods should have void return type. In my humble opinion, void methods are sink holes unless we have good verifications around them. They are harder to test, so we write less unit tests. Less unit tests lead to “[legacy code](http://www.objectmentor.com/resources/articles/WorkingEffectivelyWithLegacyCode.pdf)”. Legacy code leads to a program is hard to maintain and hard to change. I changed signatures of above 3 methods so that they would return String object. Corresponding Junit test has been developed, see part1.TestRedBlackTree.java for the details.



*Picture 1. Junit test case for RedBlackTree implementation.*

### Class Diagram

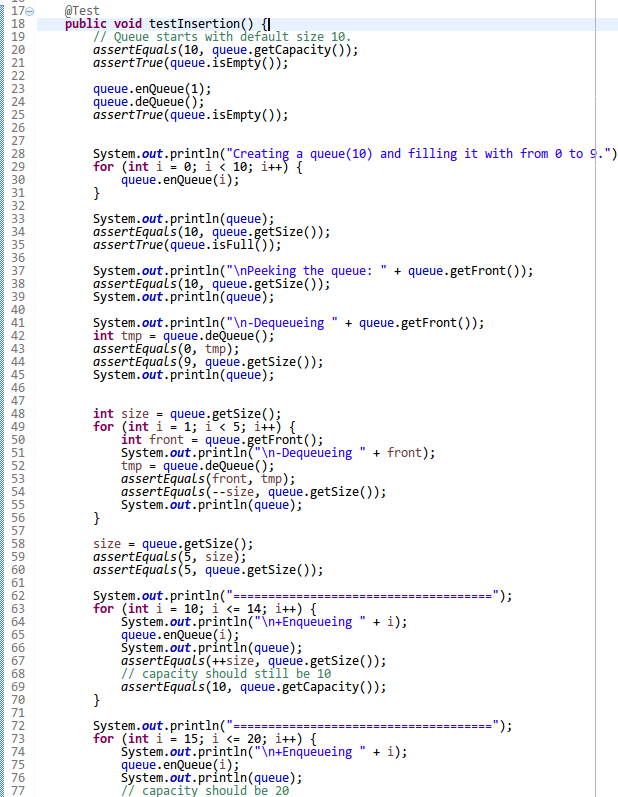
See below class diagram for project 2, part 1 program structure.



*Picture 2. Class diagram of the part 1 of homework.*

### Array Based, Self-Resizing, Circular Queue

I developed a test case for the array based, self-resizing, circular queue implementation. The initial queue size is 10. If you run part1.TestQueue.java Junit test, you will see how it handles front and tail indices, and how it expands/shrinks as we add/remove elements to/from the queue.



*Picture 3. Test case for the self-resizing, circular, fast queue.*

After resizing, the queue head starts from array index 0:

+Enqueueing 13

[10, 11, 12, 13<=r, null, f=>5, 6, 7, 8, 9]

+Enqueueing 14

[10, 11, 12, 13, 14<=r, f=>5, 6, 7, 8, 9]

=====================================

+Enqueueing 15

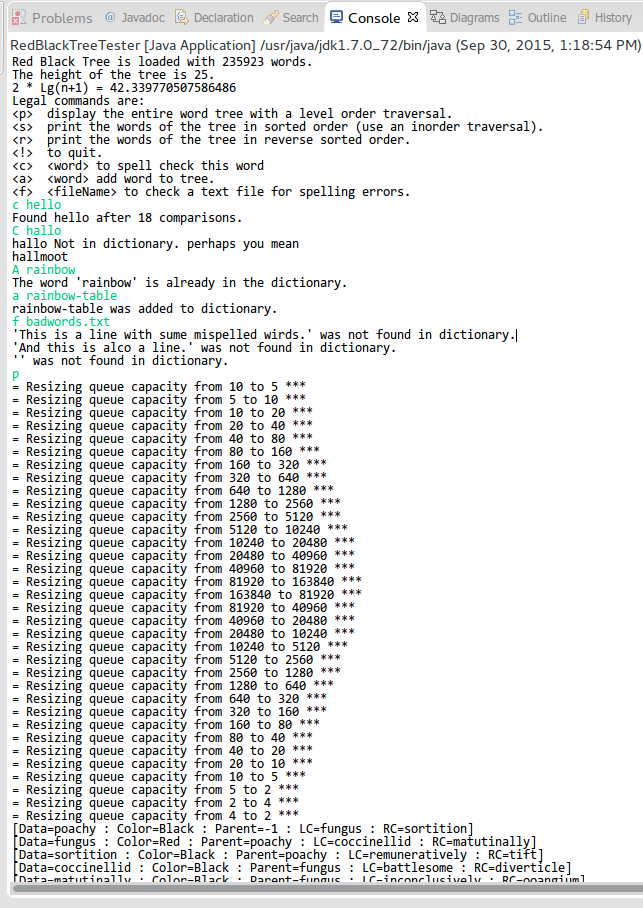
**= Resizing queue capacity from 10 to 20 \*\*\***

**[f=>5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15<=r, null, null, null, null, null, null, null, null, null]**

**Test case in action:**

|  |
| --- |
| Creating a queue(10) and filling it with from 0 to 9.  [f=>0, 1, 2, 3, 4, 5, 6, 7, 8, 9<=r]  Peeking the queue: 0  [f=>0, 1, 2, 3, 4, 5, 6, 7, 8, 9<=r]  -Dequeueing 0  [null, f=>1, 2, 3, 4, 5, 6, 7, 8, 9<=r]  -Dequeueing 1  [null, null, f=>2, 3, 4, 5, 6, 7, 8, 9<=r]  -Dequeueing 2  [null, null, null, f=>3, 4, 5, 6, 7, 8, 9<=r]  -Dequeueing 3  [null, null, null, null, f=>4, 5, 6, 7, 8, 9<=r]  -Dequeueing 4  [null, null, null, null, null, f=>5, 6, 7, 8, 9<=r]  =====================================  +Enqueueing 10  [10<=r, null, null, null, null, f=>5, 6, 7, 8, 9]  +Enqueueing 11  [10, 11<=r, null, null, null, f=>5, 6, 7, 8, 9]  +Enqueueing 12  [10, 11, 12<=r, null, null, f=>5, 6, 7, 8, 9]  +Enqueueing 13  [10, 11, 12, 13<=r, null, f=>5, 6, 7, 8, 9]  +Enqueueing 14  [10, 11, 12, 13, 14<=r, f=>5, 6, 7, 8, 9]  =====================================  +Enqueueing 15  = Resizing queue capacity from 10 to 20 \*\*\*  [f=>5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15<=r, null, null, null, null, null, null, null, null, null]  +Enqueueing 16  [f=>5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16<=r, null, null, null, null, null, null, null, null]  +Enqueueing 17  [f=>5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17<=r, null, null, null, null, null, null, null]  +Enqueueing 18  [f=>5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18<=r, null, null, null, null, null, null]  +Enqueueing 19  [f=>5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19<=r, null, null, null, null, null]  +Enqueueing 20  [f=>5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20<=r, null, null, null, null]  =====================================  -Dequeueing 5  [null, f=>6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20<=r, null, null, null, null]  -Dequeueing 6  [null, null, f=>7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20<=r, null, null, null, null]  -Dequeueing 7  [null, null, null, f=>8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20<=r, null, null, null, null]  -Dequeueing 8  [null, null, null, null, f=>9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20<=r, null, null, null, null]  -Dequeueing 9  [null, null, null, null, null, f=>10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20<=r, null, null, null, null]  -Dequeueing 10  [null, null, null, null, null, null, f=>11, 12, 13, 14, 15, 16, 17, 18, 19, 20<=r, null, null, null, null]  -Dequeueing 11  [null, null, null, null, null, null, null, f=>12, 13, 14, 15, 16, 17, 18, 19, 20<=r, null, null, null, null]  -Dequeueing 12  [null, null, null, null, null, null, null, null, f=>13, 14, 15, 16, 17, 18, 19, 20<=r, null, null, null, null]  -Dequeueing 13  [null, null, null, null, null, null, null, null, null, f=>14, 15, 16, 17, 18, 19, 20<=r, null, null, null, null]  -Dequeueing 14  [null, null, null, null, null, null, null, null, null, null, f=>15, 16, 17, 18, 19, 20<=r, null, null, null, null]  -Dequeueing 15  = Resizing queue capacity from 20 to 10 \*\*\*  [f=>16, 17, 18, 19, 20<=r, null, null, null, null, null]  -Dequeueing 16  [null, f=>17, 18, 19, 20<=r, null, null, null, null, null]  -Dequeueing 17  [null, null, f=>18, 19, 20<=r, null, null, null, null, null]  -Dequeueing 18  = Resizing queue capacity from 10 to 5 \*\*\*  [f=>19, 20<=r, null, null, null]  -Dequeueing 19  = Resizing queue capacity from 5 to 2 \*\*\*  [f=>20<=r, null] |

### Sample Program Output



*Picture 4. Sample program output. Words.txt file was loaded.  
Command directives (such <c>,<a>, <f>) are case insensitive. But words after directives are case sensitive.*

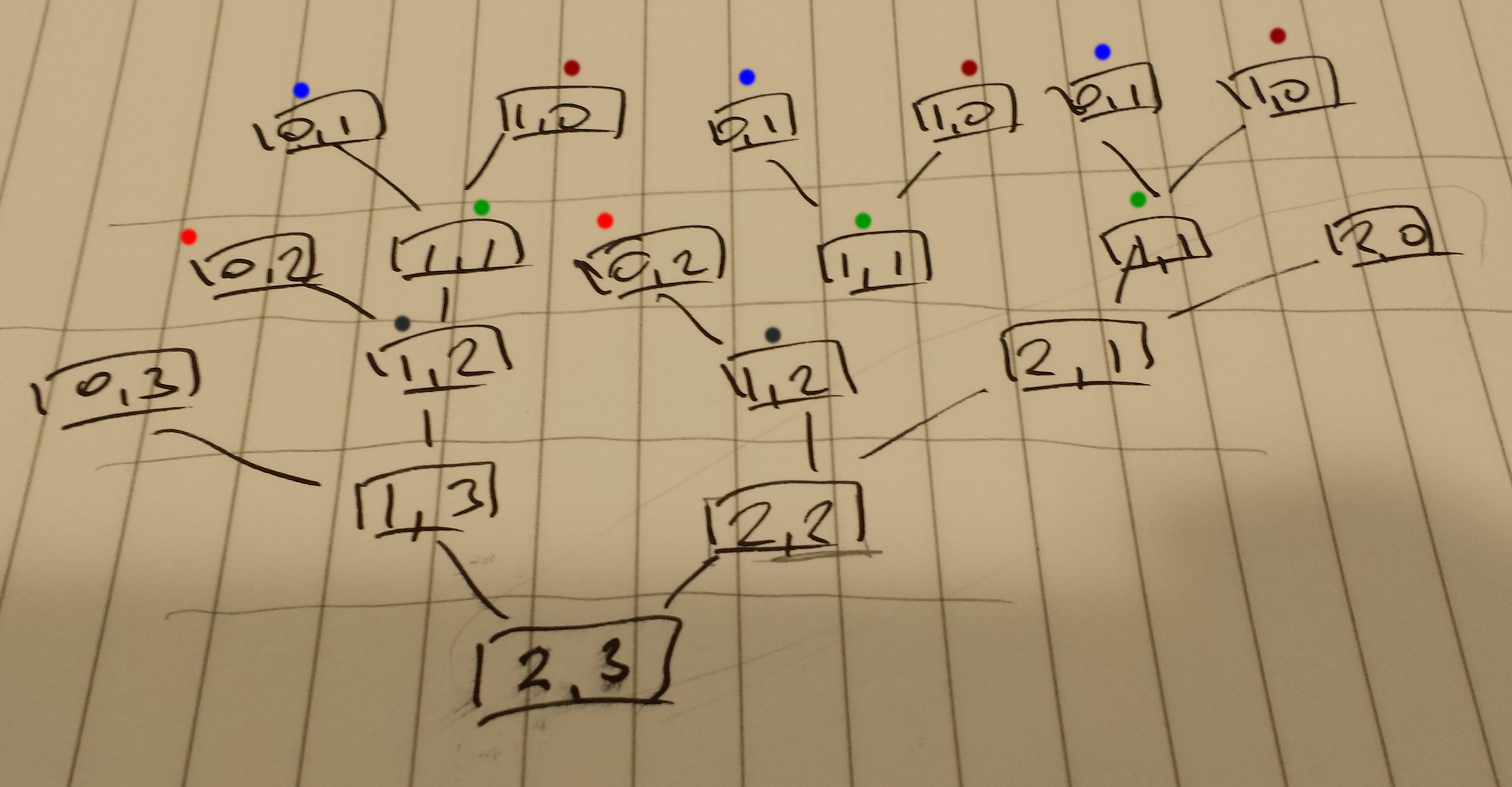
# Part 2. Dynamic Programming

## Recursive Computing Method

Running time: Exponential.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Problem Set | Value | Time in milliseconds | Number of method calls |
| 1 | P(2,3) | 0.6875 | 0 | 13 |
| 2 | P(4,7) | 0.828125 | 0 | 659 |
| 3 | P(7,6) | 0.38720703125 | 0 | 3,431 |
| 4 | P(10,12) | 0.6681880950927734 | 9 | 1,293,291 |
| 5 | P(20,23) | 0.6780155218148138 | 4,314,024  (~71 minutes) | 1,921,133,836,439 |
| 6 | P(30,15) | 0.011314420602957398 | 1,598,094  (~26 minutes) | 689,734,851,167 |
| 7 | P(50,40) | ? | ? | ? |

The running time is exponential because the recursive method computes the same set of problems again and again. See below picture for a demonstration of P(2,3) computation:



*Picture 5. The inefficient recursiveComputeP() computation for P(2,3).*

## Dynamic Computing Method

Running time: Linear or BigO(N).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Problem Set | Value | Time in milliseconds | Number of method calls |
| 1 | P(2,3) | 0.6875 | 0 | 13 |
| 2 | P(4,7) | 0.828125 | 0 | 45 |
| 3 | P(7,6) | 0.38720703125 | 0 | 37 |
| 4 | P(10,12) | 0.6681880950927734 | 0 | 149 |
| 5 | P(20,23) | 0.6780155218148138 | 1 | 681 |
| 6 | P(30,15) | 0.011314420602957398 | 0 | 301 |
| 7 | P(50,40) | 0.1445480403480301 | 1 | 2,781 |

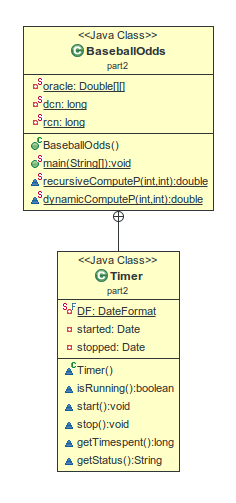
**Host machine information:**

* Memory: 16 GB
* CPU: Intel® Core™ i5-4300M CPU @ 2.60GHz × 4
* OS: Fedora 21 (64 bit)

### Program Output

|  |
| --- |
| Computing with \*dynamic\* method.  P(2,3) = 0.6875 : 0ms : dynamicComputeP() has been called 13 times  P(4,7) = 0.828125 : 0ms : dynamicComputeP() has been called 45 times  P(7,6) = 0.38720703125 : 0ms : dynamicComputeP() has been called 37 times  P(10,12) = 0.6681880950927734 : 0ms : dynamicComputeP() has been called 149 times  P(20,23) = 0.6780155218148138 : 1ms : dynamicComputeP() has been called 681 times  P(30,15) = 0.011314420602957398 : 0ms : dynamicComputeP() has been called 301 times  P(50,40) = 0.1445480403480301 : 1ms : dynamicComputeP() has been called 2781 times  Computing with \*recursive\* method.  P(2,3) = 0.6875 : 0ms : recursiveComputeP() has been called 19 times  [ Started at: 2015/09/29 21:37:56. Completed at: 2015/09/29 21:37:56. Spent 0 milliseconds. ]  P(4,7) = 0.828125 : 0ms : recursiveComputeP() has been called 659 times  [ Started at: 2015/09/29 21:37:56. Completed at: 2015/09/29 21:37:56. Spent 0 milliseconds. ]  P(7,6) = 0.38720703125 : 0ms : recursiveComputeP() has been called 3431 times  [ Started at: 2015/09/29 21:37:56. Completed at: 2015/09/29 21:37:56. Spent 0 milliseconds. ]  P(10,12) = 0.6681880950927734 : 9ms : recursiveComputeP() has been called 1293291 times  [ Started at: 2015/09/29 21:37:56. Completed at: 2015/09/29 21:37:56. Spent 9 milliseconds. ]  P(20,23) = 0.6780155218148138 : 4314024ms : recursiveComputeP() has been called 1921133836439 times  [ Started at: 2015/09/29 21:37:56. Completed at: 2015/09/29 22:49:50. Spent 4314024 milliseconds. ]  P(30,15) = 0.011314420602957398 : 1598094ms : recursiveComputeP() has been called 689734851167 times  [ Started at: 2015/09/29 22:49:50. Completed at: 2015/09/29 23:16:28. Spent 1598094 milliseconds. ] |

## Implementation



*Picture 6. Class diagram for the part 2 of homework.*

|  |
| --- |
| package part2;  import java.text.DateFormat;  import java.text.SimpleDateFormat;  import java.util.Date;  /\*\*  \* Part 2. Dynamic Programming.  \* @author bsanchin  \*/  public class BaseballOdds {    // Let the oracle matrix has N+1 x N+1 dimension, so that we will not need  // to subtract 1 to compute an array index. Row 0 and column 0 will not be used.  private static Double[][] oracle = new Double[51][51];  private static long dcn = 0;  private static long rcn = 0;    public static void main(String[] args) {    // Initialize the timer.  Timer timer = new Timer();  String label;      // Define the problem set here.  int[][] problemSet = new int[][] {  {2,3}  ,{4,7}  ,{7,6}  ,{10,12}  ,{20,23}  ,{30,15}  ,{50, 40}  };  // Solve the problem set using dynamic method.  // Running time: Linear or BigO(N)  System.out.println("Computing with \*dynamic\* method.");  for (int i = 0; i < problemSet.length; i++) {  int[] pair = problemSet[i];  label = "P(" + pair[0] + "," + pair[1] + ")";    dcn = 0;  timer.start();  double result = dynamicComputeP(pair[0], pair[1]);  timer.stop();    System.out.println(" " + label + " = " + result + " : " + timer.getTimespent()  + "ms : dynamicComputeP() has been called " + dcn + " times");  }    // Solve the problem set using recursive method.  // Running time: Exponential  System.out.println("\nComputing with \*recursive\* method.");  for (int i = 0; i < problemSet.length; i++) {  int[] pair = problemSet[i];  label = "P(" + pair[0] + "," + pair[1] + ")";  rcn = 0;  timer.start();  double result = recursiveComputeP(pair[0], pair[1]);  timer.stop();    System.out.println(" " + label + " = " + result + " : " + timer.getTimespent()  + "ms : recursiveComputeP() has been called " + rcn + " times");  System.out.println(" " + timer.getStatus());  }    }    /\*\*  \* Computes probability of team Private eventually win the series.  \* @param i - number of games Private needs to win  \* @param j - number of games Cardinals needs to win  \* @return computed probability  \*/  static double recursiveComputeP(int i, int j) {  rcn ++;  if (i == 0) { return 1; }  if (j == 0) { return 0; }  return (recursiveComputeP(i-1, j) + recursiveComputeP(i, j-1))/2;  }  /\*\*  \* Computes probability of team Private eventually win the series.  \* @param i - number of games Private needs to win  \* @param j - number of games Cardinals needs to win  \* @return computed probability  \*/  static double dynamicComputeP(int i, int j) {  dcn++;  if (i == 0) { return 1; }  if (j == 0) { return 0; }    // See if oracle knows the answer.  if (oracle[i][j] != null) {  return oracle[i][j];  }    // Oracle did not have the answer, we need to compute then!  double result = (dynamicComputeP(i-1, j) + dynamicComputeP(i, j-1))/2;    // Let oracle knows the result, she will reply in O(1) next time :)  oracle[i][j] = result;    return result;  }    /\*\*  \* Simple timer tool for measuring time elapse.  \* @author bsanchin  \*/  static class Timer {    private static final DateFormat DF = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");  private Date started;  private Date stopped;    Timer() {}    boolean isRunning() {  return started != null && stopped == null;  }    void start() {  started = new Date();  stopped = null;  }    void stop() {  stopped = new Date();  }    long getTimespent() {  return stopped.getTime() - started.getTime();  }    String getStatus() {  StringBuilder sb = new StringBuilder();  if (isRunning()) {  sb.append("[ Started at: ").append(DF.format(started)).append(".");  sb.append(" Running for " + ((new Date()).getTime() - started.getTime()) + " milliseconds. ]");  }  else {  sb.append("[ Started at: ").append(DF.format(started)).append(".");  sb.append(" Completed at: ").append(DF.format(stopped)).append(".");  sb.append(" Spent " + (stopped.getTime() - started.getTime()) + " milliseconds. ]");  }  return sb.toString();  }    }  } |