

Resposta aos Requisitos

Item 1.

Da minha perspectiva a solução segue o padrão “Linear Time “, pois é necessário percorrer toda a matriz de forma linear coluna por coluna e linha por linha. Para tornar o algoritmo mais eficiente, ao invés de percorrer todos os caminhos possíveis, apenas o caminho com maior probabilidade de maiores retornos são percorridos e no final a maior quantidade de bananas possíveis é apresentada.

Item 2.

Para compreensão da solução lancei mão de papel e caneta, primeiro para ter modelo de matriz e poder verificar as regras de movimentação, após compressão, lancei mão do uso de uma planilha em busca de quantidade de interações que seriam necessárias na matriz para percorrer todos os caminhos possíveis, ao fazer isso percebi que a melhor maneira não era percorrer todos os caminhos e sim identificar qual o melhor caminho, além de tornar a solução mais inteligente isso iria ajudar na eficiência da solução, tanto no consumo de memória quando no de tempo de execução, para chegar ao raciocínio final levei mais ou menos três horas de análise para implementar a solução foram mais 4 horas entre escrita e testes.

Item 3.

Esta é a solução:

```
package javaapplication8;
```

```
/**
```

```
*
```

```
* @author Josberto
```

```
*/
```

```
public class JavaApplication8 {
```

```
    /**
```

```
    * @param args the command line arguments
```

```
    */
```

```
    public static void main(String[] args) {
```

```
        int[][] matriz = {{10,33,13,15},{22,21,4,1},{5,0,2,3},{0,6,14,2}};
```

```
        System.out.println(way(matriz));
```

```
    }
```

```
    /**
```

```
    * way - Processa a matriz em busca da maior somatória para caminho percorrido na matriz
```

```
    * @param matriz
```

```
    * @return
```

```
    */
```

```
    public static int way(int[][] matriz){
```

```
        int _x = 0;int _y = 0;int pos = 0; int maxBananas = 0;
```

```

for(int w = 0; w < matriz.length; w++){
    int count = 0;
    for(int h = 0; h < matriz.length; h++){
        if(_x == 0 && _y == 0 ){
            count += matriz[_x][_y];
            pos = posInicioTop( matriz[_x][_y+1], matriz[_x+1][_y+1] );
            if(pos == 2){
                _x += 1;
                _y +=1;
            } else if(pos == 0){
                _y +=1;
            }
            count += matriz[_x][_y];
        }else if((_x == 0) && _y < matriz[_x].length-1){
            pos = posInicioTop( matriz[_x][_y+1], matriz[_x+1][_y+1] );
            if(pos == 2){
                _x += 1;
                _y +=1;
            } else if(pos == 0){
                _y +=1;
            }
            count += matriz[_x][_y];
        }
        if(_x > 0 && _x != matriz.length - 1){
            if(_y == 0){
                count += matriz[_x][_y];
            }
            if(_y < matriz[_x].length-1){
                pos = posIntermediarias( matriz[_x-1][_y+1], matriz[_x][_y+1], matriz[_x+1][_y+1]);
                if(pos == 0){
                    _y+=1;
                }else if(pos == 1){
                    _x-=1; _y+=1;
                }else if(pos == 2){
                    _x+=1; _y+=1;
                }
                count += matriz[_x][_y];
            }
        }

        if (_x > 0 && _x == matriz.length - 1){
            if( _y < matriz[_x].length - 1 ){
                pos = posInicioBottom( matriz[_x-1][_y+1], matriz[_x][_y+1]);
                if(pos == 0){
                    _y+=1;
                }else if(pos == 1){
                    _x -= 1; _y+=1;
                }
            }
        }
    }
}

```

```

        }
    }
}
_x = 1 + w;
_y = 0;
if(maxBananas < count){
    maxBananas = count;
}
}
return maxBananas;
}

/**
 * Identifica qual a direção o código deve seguir quando a leitura está na primeira linha da matriz
 * @param x
 * @param y
 * @return
 */
public static int posInicioTop(int x, int y){
    int retorno = 0;
    if (x == y){
        retorno = 0;
    }else if (x < y){
        retorno = 2;
    }else if (x > y){
        retorno = 0;
    }
    return retorno;
}

/**
 * Identifica qual a direção o código deve seguir quando a interação está na última linha da matriz
 * @param x
 * @param y
 * @return
 */
public static int posInicioBottom(int x, int y){
    int retorno = 0;
    if (x == y){
        retorno = 0;
    }else if (x < y){
        retorno = 0;
    }else if (x > y){
        retorno = 1;
    }
    return retorno;
}

```

```

/**
 * Identifica a direção que o código deve seguir quando a leitura estiver em uma linha interna da matriz
 * @param x
 * @param y
 * @param z
 * @return
 */
public static int posIntermediarias(int x, int y, int z){
    int retorno = 0;
    if(x == y && x == z){
        retorno = 0;
    }else if(x == y && y > z){
        retorno = 0;
    } else if(x == y && z > y){
        retorno = 2;
    } else if( x == y && x > z){
        retorno = 0;
    } else if(x == z && y > z){
        retorno = 0;
    } else if(x == z && z > y){
        retorno = 2;
    } else if(y == z && x > y){
        retorno = 1;
    } else if(y == z && x < y){
        retorno = 0;
    } else if(z > x && z > y){
        retorno = 2;
    } else if( y > x && y > z){
        retorno = 0;
    }else if( x > y && x > z){
        retorno = 1;
    }
    return retorno;
}
}

```

Código de teste básico

```

package javaapplication8;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Josberto
 */
public class JavaApplication8Test {

```

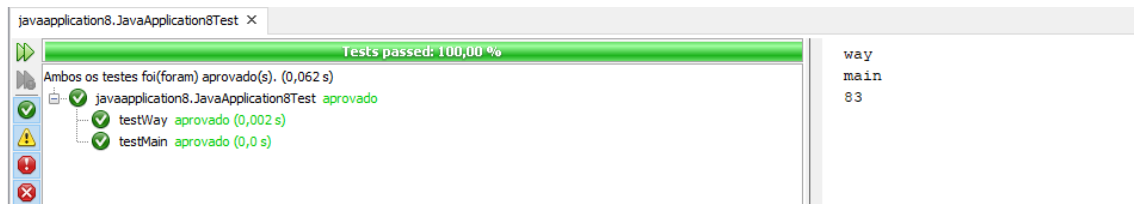
```

/**
 * Test of main method, of class JavaApplication8.
 */
@Test
public void testMain() {
    System.out.println("main");
    String[] args = null;
    JavaApplication8.main(args);
}

/**
 * Test of way method, of class JavaApplication8.
 */
@Test
public void testWay() {
    System.out.println("way");
    int[][] matriz = {{10,33,13,15},{22,21,4,1},{5,0,2,3},{0,6,14,2}};
    int expResult = 83;
    int result = JavaApplication8.way(matriz);
    assertEquals(expResult, result);
}
}

```

Resultado



The screenshot shows a test runner window for the class `javaapplication8.JavaApplication8Test`. The window has a title bar with the class name and a close button. Below the title bar, a green progress bar indicates "Tests passed: 100,00 %". The main area displays the test results in a tree view. The root node is `javaapplication8.JavaApplication8Test`, which is expanded to show two sub-nodes: `testWay` and `testMain`. Both sub-nodes are marked with a green checkmark and the text "aprovado" (approved), indicating they passed. The execution times for each test are shown in parentheses: `testWay` took 0,002 s and `testMain` took 0,0 s. On the left side of the window, there is a vertical toolbar with icons for running, debugging, and other actions. On the right side, there is a console area showing the output of the tests: "way" and "main" on separate lines, followed by the value "83" on the last line.

Test Method	Status	Execution Time
testWay	aprovado	0,002 s
testMain	aprovado	0,0 s