I realised these areas whilst performing my coding..

As per investigation whilst coding, I am quickly examining the entries. I am trying to ascertain if I can perform any cost savings by not storing certain entries in the set. Otherwise, my coding will utilize try and catch to eradicate a large subset.
It would be approximately in this section of the code:

```
for (int k: nMoves)
    {
      System.out.println("Elements in: " + i + " row of triangle:" + triangle[i][0].length);
      i++;
      try
      {
        HERE IT WOULD HAVE APPLIED THE FIRST ENTRY IN THE SUBSET TO ADDRESS ROW 0   TRIANGLE[0][0][X]

        HERE IT WOULD HAVE APPLIED THE FIRST ENTRY IN THE SUBSET TO ADDRESS ROW 1   TRIANGLE[1][0][X]

        HERE IT WOULD HAVE APPLIED THE THIRD ENTRY IN THE SUBSET TO ADDRESS ROW 2   TRIANGLE[2][0][X]
      }
      catch (ArrayIndexOutOfBoundsException e)
      {
        STATE INVALID SUBSET AND DISCARD IT
      }
```

I do not consider this as poor practice, but I think I will apply better logic if I identify any invalid subsets prior to this... reason is also for collating information during mid-execution in the set that accurately reflects issue in hand.. Ironically, to some extents the code above is significantly shorter and easier to follow and implement than changes I completed (to support the below).

My interpretations were as follows (and if it is correct, can be applied principally for larger triangles also.. This will be something I will need to test, and currently have no idea until Test cases):

(1,X,X) or (2,X,X) are of no use whatsoever = 66% entries = 18 entries...
//Also (0,2,0), (0,2,1) and (0,2,2) = 3 entries **TOTAL 21**

It has left 6 remaining entries (see overleaf).
I have adjusted my code with implementations to support this.
And fortunately it has brought the unique entries down to 6 also.
But it required lots tweaking.

//I have colour coded below any non-applicable entries since I had to perform lots of code changes in phases and eventually I removed all the coloured entries!!

**********ALL UNIQUE ENTRIES**********

0,0,1

0,1,0

0,0,0

0,1,2

0,2,1

0,0,2

0,1,1

0,2,0

0,2,2

2,1,2

2,2,1

2,1,1

2,0,2

2,2,0

2,2,2

2,0,1

2,1,0

2,0,0

1,2,2

1,2,1

1,1,2

1,0,2

1,2,0

1,1,1

1,0,1

1,1,0

1,0,0

27 unique combinations

//This is with adjusted code

**********ALL UNIQUE ENTRIES**********

0,1,0

0,0,1

0,0,0

0,1,2

0,0,2

0,1,1

6 unique combinations

TEST CASE 1: As per the challenge

```
int [ ][ ][ ] triangle = new int[ ][ ][ ] {
                                    {  {1}  },
                                    { {2,3}  },
                                   { {1,5,1}  },
                                  };
```

Welcome to Online IDE!! Happy Coding :)

P^R(3,3) = Math.pow(n,r)

PERMUTATIONS(WITH REPLACEMENT)

PR(n,r) = nr

**********THE TRIANGLE COUNT CHALLENGE**********

P^R(3,3) = Math.pow(n,r)

Testing Permutations: 27

There are triangles configured : 1

**********INITIAL VALUE OF CYCLES: 0

**********Contents of the backup set

**********Contents of the valuesSet

0,0,1 6 viable permutations

0,1,0

0,0,0

0,1,2

0,1,1

0,0,2

**********NEW VALUE CYCLES: 1080

**********RUNNING TOTAL CYCLES: 1080

***PROCESSING SET AT INDEX: 0

ENDING AT INDEX:*** 6

0,0,1 Subset: 1 at cycle number: 1080 // this should correspond to first entry above

TRIANGLE 0

Elements in: 0 row of triangle:1 //confirming expected number elements in each triangle row

Value at triangle: 0 [0][0][0]: 1

Elements in: 1 row of triangle:2 **//confirming expected number elements in each triangle row**

Value at triangle: 0 [1][0][0]: 2

Elements in: 2 row of triangle:3 **//confirming expected number elements in each triangle row**

Value at triangle: 0 [2][0][1]: 5

*******TOTAL: 8 //this is number of interest

Highest total triangle(0)is: 8

0,1,0 Subset: 2 at cycle number: 1080 **// this should correspond to second entry above**

TRIANGLE 0

Elements in: 0 row of triangle:1

Value at triangle: 0 [0][0][0]: 1

Elements in: 1 row of triangle:2

Value at triangle: 0 [1][0][1]: 3

Elements in: 2 row of triangle:3

Value at triangle: 0 [2][0][0]: 1

*******TOTAL: 5

Highest total triangle(0)is: 8

0,0,0 Subset: 3 at cycle number: 1080

TRIANGLE 0

Elements in: 0 row of triangle:1

Value at triangle: 0 [0][0][0]: 1

Elements in: 1 row of triangle:2

Value at triangle: 0 [1][0][0]: 2

Elements in: 2 row of triangle:3

Value at triangle: 0 [2][0][0]: 1

*******TOTAL: 4

Highest total triangle(0)is: 8

0,1,2 Subset: 4 at cycle number: 1080

TRIANGLE 0

Elements in: 0 row of triangle:1

Value at triangle: 0 [0][0][0]: 1

Elements in: 1 row of triangle:2

Value at triangle: 0 [1][0][1]: 3

Elements in: 2 row of triangle:3

Value at triangle: 0 [2][0][2]: 1

*******TOTAL: 5

Highest total triangle(0)is: 8

0,1,1 Subset: 5 at cycle number: 1080

TRIANGLE 0

Elements in: 0 row of triangle:1

Value at triangle: 0 [0][0][0]: 1

Elements in: 1 row of triangle:2

Value at triangle: 0 [1][0][1]: 3

Elements in: 2 row of triangle:3

Value at triangle: 0 [2][0][1]: 5

*******TOTAL: 9

Highest total triangle(0)is: 9

0,0,2 Subset: 6 at cycle number: 1080

TRIANGLE 0

Elements in: 0 row of triangle:1

Value at triangle: 0 [0][0][0]: 1

Elements in: 1 row of triangle:2

Value at triangle: 0 [1][0][0]: 2

Elements in: 2 row of triangle:3

Value at triangle: 0 [2][0][2]: 1

*******TOTAL: 4

Highest total triangle(0)is: 9

**********SUMMARY HIGHEST RESULTS**********

*******TRIANGLES*********

There are triangles configured : 1

[[1]] //This is triangle configured

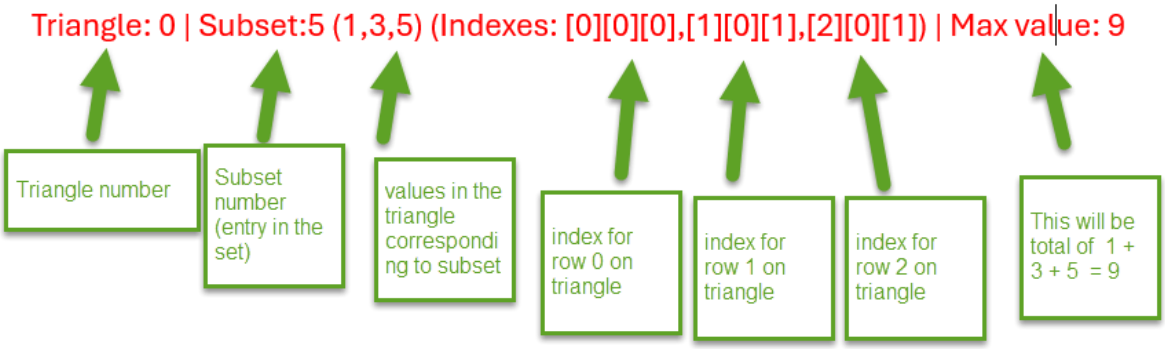
[[2, 3]]

[[1, 5, 1]]

****

Triangle: 0 | Subset:5 (1,3,5) (Indexes: [0][0][0],[1][0][1],[2][0][1]) | Max value: 9

** Process exited - Return Code: 0 **



TEST CASE 2: multiple identical paths

Expecting this to be identical across any triangle configuration.

```
Welcome to Online IDE!! Happy Coding :)
P^R(3,3) = Math.pow(n,r)
***PERMUTATIONS***(WITH REPLACEMENT)
PR(n,r) = nr
**********THE TRIANGLE COUNT CHALLENGE***********
P^R(3,3) = Math.pow(n,r)
Testing Permutations: 27
***There are triangles configured*** : 1
*************INITIAL VALUE OF  CYCLES: 0
*****************Contents of the backup set
*****************Contents of the valuesSet
0,0,1
0,1,0
0,0,0
0,1,2
0,0,2
0,1,1
```

EXPLAINED

```
*************SUMMARY HIGHEST RESULTS*************
*******TRIANGLES*********
***There are triangles configured*** : 1

[[1]]
[[3, 3]]
[[1, 5, 1]]
****************

Triangle: 0 | Subset:1 (1,3,5) (Indexes: [0][0][0],[1][0][0],[2][0][1]) | Max value: 9
Triangle: 0 | Subset:6 (1,3,5) (Indexes: [0][0][0],[1][0][1],[2][0][1]) | Max value: 9
```

```
Welcome to Online IDE!! Happy Coding :)
P^R(3,3) = Math.pow(n,r)
***PERMUTATIONS***(WITH REPLACEMENT)
PR(n,r) = nr
**********THE TRIANGLE COUNT CHALLENGE**********
P^R(3,3) = Math.pow(n,r)
Testing Permutations: 27
***There are triangles configured*** : 1
*************INITIAL VALUE OF  CYCLES: 0
*****************Contents of the backup set
*****************Contents of the valuesSet
0,0,1
0,1,0
0,0,0
0,1,2
0,0,2
0,1,1
```

```
*************SUMMARY HIGHEST RESULTS*************
*******TRIANGLES*********
***There are triangles configured*** : 1

[[1]]
[[3, 3]]
[[1, 5, 1]]
****************

Triangle: 0 | Subset:1 (1,3,5) (Indexes: [0][0][0],[1][0][0],[2][0][1]) | Max value: 9
Triangle: 0 | Subset:6 (1,3,5) (Indexes: [0][0][0],[1][0][1],[2][0][1]) | Max value: 9
```

I have just stated subset 1 (0,0,1). I have not even conveyed this information in results to ensure no information overload..
BUT IT CAN BE SEEN THE VALUES TRANSLATE HERE:

I have just stated subset 6 (0,1,1). I have not even conveyed this information in results to ensure no information overload..
BUT IT CAN BE SEEN THE VALUES TRANSLATE HERE:

IT CAN ALSO BE SEEN THAT TOTALS ARE EXACTLY THE SAME

TEST CASE 3: ALL paths same total, but different values... FAILED

```
int [ ][ ][ ] triangle = new int[ ][ ][ ] {
                                  { {1} },
                                  { {8,8} },
                                 { {1,1,1} },
                                };
```

```
Triangle: 0 | Subset:1 (1,8,1) (Indexes: [0][0][0],[1][0][0],[2][0][1]) | Max value: 10
Triangle: 0 | Subset:2 (1,8,1) (Indexes: [0][0][0],[1][0][1],[2][0][0]) | Max value: 10
Triangle: 0 | Subset:3 (1,8,1) (Indexes: [0][0][0],[1][0][0],[2][0][0]) | Max value: 10
Triangle: 0 | Subset:4 (1,8,1) (Indexes: [0][0][0],[1][0][1],[2][0][2]) | Max value: 10
Triangle: 0 | Subset:5 (1,8,1) (Indexes: [0][0][0],[1][0][1],[2][0][1]) | Max value: 10
Triangle: 0 | Subset:6 (1,8,1) (Indexes: [0][0][0],[1][0][0],[2][0][2]) | Max value: 10
```

I have been slightly surprised by this outcome, but once again, there is always value in performing trivial testing..

So I have taken opportunity to investigate this:
It is quite embarrassing that since I put lots of focus into so many areas, I have missed out significant logic in the code.

```
Welcome to Online IDE!! Happy Coding :)
P^R(3,3) = Math.pow(n,r)
***PERMUTATIONS***(WITH REPLACEMENT)
PR(n,r) = nr
**********THE TRIANGLE COUNT CHALLENGE**********
P^R(3,3) = Math.pow(n,r)
Testing Permutations: 27
***There are triangles configured*** : 1
*************INITIAL VALUE OF  CYCLES: 0
******************Contents of the backup set
******************Contents of the valuesSet
0,0,1
0,1,0
0,0,0
0,1,2
0,0,2
0,1,1
```

```
int [ ][ ][ ] triangle = new int[ ][ ][ ] {
                                  { {1} },
                                  { {8,8} },
                                  { {1,1,1} },
                                 };
```

```
Triangle: 0 | Subset:1 (1,8,1) (Indexes: [0][0][0],[1][0][0],[2][0][1]) | Max value: 10
Triangle: 0 | Subset:2 (1,8,1) (Indexes: [0][0][0],[1][0][1],[2][0][0]) | Max value: 10
Triangle: 0 | Subset:3 (1,8,1) (Indexes: [0][0][0],[1][0][0],[2][0][0]) | Max value: 10
Triangle: 0 | Subset:4 (1,8,1) (Indexes: [0][0][0],[1][0][1],[2][0][2]) | Max value: 10
Triangle: 0 | Subset:5 (1,8,1) (Indexes: [0][0][0],[1][0][1],[2][0][1]) | Max value: 10
Triangle: 0 | Subset:6 (1,8,1) (Indexes: [0][0][0],[1][0][0],[2][0][2]) | Max value: 10
```

We know on a triangle there are 4 valid paths. I completely bypassed this when performing evaluation on my previous logic documentation, It is considered a big error, but if the question had given all paths in the example, I would have identified this earlier...
With respect to the triangle, we know that the adjacent aspect is totally irrelevant up to row 2.
From row 2 to row 3, we can see the following paths are invalid (see triangle configuration).

In terms of implementation , it needs something in the code where stepcounter ==3
But issue is, there just isn't any logic out there since whilst it is performing the permutations in the set, it has no idea that it will be transposed onto a triangle.
There is no pattern whatsoever that I can visualize..

There is one option, it can be seen that logically on triangle, adjacency is valid if directly beneath it.
I can ONLY hardcode logic so that if the subset {0,1,0} or {0,0,2} appears to discard it.
This is only way since there is no violation at the indexing level, there is no other way to put adjacency logic into practice..

triangle[1][0][**0**]
triangle[2][0][**0**] triangle[2][0][**1**] triangle[2][0][2]

triangle[1][0][**1**]
triangle[2][0][0] triangle[2][0][**1**] triangle[2][0][**2**]

A big issue with this approach it fixes triangles with three rows, but it would need this logic moving into larger triangles.

It can be seen above that the adjacent value on next row has index either same or greater by 1...
AND Also not lower..

VIABLE SOLUTION 1: Not versatile

```
if (q==r-1) // if its on the last loop..
{
    //critical for testing... can not proceed if stepsCounter != triangles.leng
    //System.out.println("value of stepsCounter: " + stepsCounter);


for (int h=0; h<1; h++)
{
    if (stepStore[h]==0 && stepStore[h+1]==1 && stepStore[h+2]==0)
    {
        invalidIndex=true;
        break;
```

```
    }

    if (stepStore[h]==0 && stepStore[h+1]==0 && stepStore[h+2]==2)
    {
        System.out.println("1EVER HERE!");
        System.out.println(sj.toString());
        invalidIndex=true;
        break;

    }


  }
```

```
if (!invalidIndex)
{
    num=0;
    stepStore=new int[rowsTriangle];
    //This would be the point where code is changed.
    //it would only perform this condition.

    st.add(sj.toString()); // it will only add it if the conditions are met abov
    //preparing for next execution. StringJoiner has not been modified yet,
    //so just performing a cleanse incase...
    stepsCounter=0;
    sj = new StringJoiner(",");
}
```

I also had to reset the num variable.

```
if (invalidIndex)
{
    sj=new StringJoiner(",");
    invalidIndex=false;
    stepsCounter=0;
    num=0;
}
```

however a more comprehensive fix is this:

VIABLE SOLUTION 2: versatile

```
if (q==r-1) // if its on the last loop..
{
    //critical for testing... can not proceed if stepsCounter != 
    //System.out.println("value of stepsCounter: " + stepsCounter
for (int h=0; h<1; h++)
{
    //It can be seen above that the adjacent value on next row ha
    //AND Also not lower..
        if ((Math.abs(stepStore[h+2]-stepStore[h+1])>1))
        {
        invalidIndex=true;
```

```
        break;
        }

        if (stepStore[h+2]<stepStore[h+1])
        {
        invalidIndex=true;
        break;
        }
```

And that is it

TEST CASE 3 (re-tested): ALL paths same total, but different values... PASSED

```
*********
0,0,1
0,0,0
0,1,2
0,1,1
*********
```

```
*************SUMMARY HIGHEST RESULTS*************
*******TRIANGLES*********
***There are triangles configured*** : 1

[[1]]
[[8, 8]]
[[1, 1, 1]]
***************

Triangle: 0 | Subset:1 (1,8,1) (Indexes: [0][0][0],[1][0][0],[2][0][1]) | Max value: 10
Triangle: 0 | Subset:2 (1,8,1) (Indexes: [0][0][0],[1][0][0],[2][0][0]) | Max value: 10
Triangle: 0 | Subset:3 (1,8,1) (Indexes: [0][0][0],[1][0][1],[2][0][2]) | Max value: 10
Triangle: 0 | Subset:4 (1,8,1) (Indexes: [0][0][0],[1][0][1],[2][0][1]) | Max value: 10
```

TEST CASE 4: Trying incorrect triangle configuration

```
//This is defining two triangles, it can be increased
int [ ][ ][ ] triangle = new int[ ][ ][ ] {
                                  {  {1,15}  },
                                  { {8,9,70}  },
                                 { {1,1,1,99,6}  },
                                };
```

```
TRIANGLE 0
Elements in: row 0 of triangle:  2
Ensure correct triangle configuration.


** Process exited - Return Code: 0 **
```

TEST CASE 5: Trying incorrect triangle configuration

```
int [ ][ ][ ] triangle = new int[ ][ ][ ] {
                                    {  {}  },
                                    { {8,9,70}  },
                                   { {1,1,1,99,6}  },
                                  };
```

```
0,0,1    Subset: 1  at cycle number: 1080
TRIANGLE 0
Elements in: row 0 of triangle:  0
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0
        at Staircase.performMoves(Permutation.java:462)
        at Staircase.obtainMoves(Permutation.java:425)
        at Staircase.<init>(Permutation.java:385)
        at Permutation.main(Permutation.java:687)
```

Resolved by adding following code:

```
try
{
System.out.println("Value at triangle: " + j + "  [" + i +  "]" + "["
}
catch (ArrayIndexOutOfBoundsException e)
{
    System.out.println("Ensure correct triangle configuration.");
    System.exit(0);
}
```

TEST CASE 6: Trying larger triangle configuration

```
int [ ][ ][ ] triangle = new int[ ][ ][ ] {
                                    {  {1}  },
                                    { {2,3}  },
                                   { {1,5,1}  },
                                   { {1,9,7,8}  },
                                  };
```

Once again, I needed adjustment on how I populated X[ ] with all the index values..

Once this was complete, I generated results...

```
for (int q=0; q<rowsTriangle; q++)
{
    X[q]=q;
//int [] X = new int []{0,1,2};
// we know on top row of triangle 0 index will cover only 1 element
//we know on 2nd row, indexes are 0 and 1  = 2 elements
//we know on 3rd row, index is 0,1,2  = 3 elements.
}
```

I will need to manually validate this since I know P(4,4) = 256 and there are 16 entries.
It can already be seen the benefits...
In row 0 expect value 0 only (this validates)
In row 1, expect values 1 and 0 (this validates)
In row 2, expect values 0 and 1 and 2 (this validates)
In row 3, expect values 0 and 1 and 2 and 3 (this validates)

```
*****************Contents of the valuesSet
0,0,1,0
0,0,0,1
0,1,2,3
0,0,0,0
0,0,0,3
0,1,1,3
0,0,1,2
0,1,2,2
0,0,0,2
0,1,1,2
0,0,1,1
0,1,2,1
0,1,2,0
0,1,1,1
0,0,1,3
0,1,1,0
```

```
*************SUMMARY HIGHEST RESULTS*************
*******TRIANGLES*********
***There are triangles configured*** : 1

[[1]]
[[2, 3]]
[[1, 5, 1]]
[[1, 9, 7, 8]]
****************

Triangle: 0 | Subset:14 (1,3,5,9) (Indexes: [0][0][0],[1][0][1],[2][0][1],[3][0][1]) | Max
value: 18
```

Analysis:

```
*************SUMMARY HIGHEST RESULTS*************
*******TRIANGLES*********
***There are triangles configured*** : 1

[[1]]
[[2, 3]]
[[1, 5, 1]]
[[1, 9, 7, 8]]
****************

Triangle: 0 | Subset:14 (1,3,5,9) (Indexes: [0][0][0],[1][0][1],[2][0][1],[3][0][1]) | Max
value: 18
```

And it can be seen that it is the highest total

```
Subset 1: 0,0,1,0
Subset 2: 0,0,0,1
Subset 3: 0,0,0,0
Subset 4: 0,1,2,3
Subset 5: 0,1,2,2
Subset 6: 0,0,0,3
Subset 7: 0,0,1,2
Subset 8: 0,1,1,3
Subset 9: 0,1,1,2
Subset 10: 0,1,2,1
Subset 11: 0,0,1,1
Subset 12: 0,0,0,2
Subset 13: 0,1,2,0
Subset 14: 0,1,1,1
Subset 15: 0,0,1,3
Subset 16: 0,1,1,0
```

TEST CASE 7: Trying two valid triangles in the configuration. First will try both the same rows.. FAILED

```
*************SUMMARY HIGHEST RESULTS*************
*******TRIANGLES*********
***There are triangles configured*** : 2

[[1], [4]]
[[2, 3], [6, 8]]
[[1, 5, 1], [9, 2, 3]]
[[1, 9, 7, 8], [10, 17, 12, 23]]
***************

Triangle: 0 | Subset:15 (1,3,5,9) (Indexes: [0][0][0],[1][0][1],[2][0][1],[3][0][1]) | Max value: 18
Triangle: 1 | Subset:7 (4,6,9,23) (Indexes: [0][1][0],[1][1][0],[2][1][0],[3][1][3]) | Max value: 42
```

I can spot issue here. it should have navigated via 8 and not 6

Again, this was the first attempt at dual triangles.. At this point I am unsure if it related to extra logic either.

I will first do sensible approach first and try single triangle again, but maintain the one that has been affected.

It has still made wrong decision..

I will reduce the triangle by removing bottom layer (same issue on 2nd row).

Good news is that it is isolated to row 2. I tried moving highest number in other rows and program had no issues.

Referring back to this logic:

	triangle[0][0][**0**]	**stepStore[0]**
triangle[1][0][**0**]	triangle[1][0][**1**]	**stepStore[1]**

My logic in code stated:

```
if ((Math.abs(stepStore[2]-stepStore[1])>1))
{
invalidIndex=true;
break;
}
```

```
if (stepStore[2]<stepStore[1])
{
invalidIndex=true;
break;
}
```

It can be seen immediately that is has used the decision making in stepStore [2] , one layer down, to make decision on the entire subset. This in itself is totally illogical, but it

seemed correct at the moment of coding!
Also using reference to a hard index as such was beneficial when testing three layers.
However it will be out of bounds operating on a two row triangle...

I have fixed issue as follows (will explain in my documentation also). It works successfully now for a single triangle with four layers, whereas it was taking wrong route before.

```
for (int h=1; h<(rowsTriangle-1); h++)
{
    //It can be seen above that the adjacent value on next row ha
    //AND Also not lower..
        if ((Math.abs(stepStore[h+1]-stepStore[h])>1))
        {
        invalidIndex=true;
        break;
        }
```

```
    if (stepStore[h+1]<stepStore[h])
    {
    invalidIndex=true;
    break;
    }
```

TEST CASE 7 (re-testing): Trying two valid triangles in the configuration. First will try both the same rows - **FAIL**

```
int [ ][ ][ ] triangle = new int[ ][ ][ ] {
                                    {        {1},        {4}          },
                                    {       {2,3},      {6,8}         },
                                    {      {1,5,1},    {9,2,3}        },
                                    {    {1,9,7,8}, {10,17,12,23}     },
                                  };
```

```
*************SUMMARY HIGHEST RESULTS*************
*******TRIANGLES*********
***There are triangles configured*** : 2

[[1], [4]]
[[2, 3], [6, 8]]
[[1, 5, 1], [9, 2, 3]]
[[1, 9, 7, 8], [10, 17, 12, 23]]
***************

Triangle: 0 | Subset:8 (1,3,5,9) (Indexes: [0][0][0],[1][0][1],[2][0][1],[3][0][1]) | Max
value: 18
Triangle: 1 | Subset:3 (4,8,3,23) (Indexes: [0][1][0],[1][1][1],[2][1][2],[3][1][3]) | Max
value: 38
```

AGAIN, IT HAS TAKEN WRONG PATH!

It now seems performing dual triangles has caused issue. But at moment, I am totally unsure of the cause...

My first line of thought is to check if the path exists:

[0,1,0, 3] = expected path
[0,1,2,3] = taken path

It can seen that {0,1,0,3} is not present...

```
Subset 1: 0,0,0,1
Subset 2: 0,1,2,3
Subset 3: 0,0,0,0
Subset 4: 0,1,2,2
Subset 5: 0,0,1,2
Subset 6: 0,1,1,2
Subset 7: 0,0,1,1
Subset 8: 0,1,1,1
```

```
for (int h=1; h<(rowsTriangle-1); h++)
{
    //It can be seen above that the adjacent value on next row ha
    //AND Also not lower..
        if ((Math.abs(stepStore[h+1]-stepStore[h])>1))
        {
        invalidIndex=true;
        break;
        }
```

This is not applicable

stepStore[h]=1

stepStore[h+1]=0

```
    if (stepStore[h+1]<stepStore[h])
    {
    invalidIndex=true;
    break;
    }
```

This is not applicable

[0,1,0, 3] = expected path
[0,1,2,3] = taken path

```
Subset 1: 0,0,0,1
Subset 2: 0,1,2,3
Subset 3: 0,0,0,0
Subset 4: 0,1,2,2
Subset 5: 0,0,1,2
Subset 6: 0,1,1,2
Subset 7: 0,0,1,1
Subset 8: 0,1,1,1
```

This clearly suggests my logic is incorrect in the initial phase (prior to this) in below:

```
//holds random number
temp1 = rand.nextInt(X.length);
```

```
for (int q=0; q<r;q++)
{
```

We know that q=2 during this selection. We know that temp1 would be 0
temp1<=q (0<=2) = true (SO NO ISSUES).

```
    if (temp1<=q)
  {

      p=temp1;

      //increase the counter
      stepsCounter++;

  }
```

There is no chance of it entering the else statement given the above validation. It only suggests a variable is not reset properly. But once again, just not possible to figure it out...

```
  else
  {   //it also needs to set a flag here...
      //since the q=0 will only take effect once it has reached
      //end of execution of the for loop..
      //note performing a break is costing one cycle also in
      //do while loop, this has to be considered if triangle format
      //improbable.. for instance several rows...
      //We know only 0 is applicable for first row..

      invalidIndex=true;
      break;
```

TEST CASE 8 : Taking this challenge to a basic level (one triangle) and trying two row triangle

```
int [ ][ ][ ] triangle = new int[ ][ ][ ] {
                                    {        {1}                    },
                                    {       {2,3}                   },

                                        };
```

TEST CASE 9 : Taking this challenge to a basic level (two triangles) and trying two row triangle. Also keeping same numbers as previously for consistency to eradicate this finding.

```
int [ ][ ][ ] triangle = new int[ ][ ][ ] {
                                    {        {1},     {4}            },
                                    {       {2,3},   {6,8}           },

                                        };
```

```
*************SUMMARY HIGHEST RESULTS*************
*******TRIANGLES*********
***There are triangles configured*** : 2

[[1], [4]]
[[2, 3], [6, 8]]
***************

Triangle: 0 | Subset:3 (1,3) (Indexes: [0][0][0],[1][0][1]) | Max value: 4
Triangle: 1 | Subset:3 (4,8) (Indexes: [0][1][0],[1][1][1]) | Max value: 12
```

TEST CASE 10 : Taking this challenge to a basic level (two triangles) and trying three row triangle - FAIL

So problem starts when it reaches 3 rows in a triangle for same reason (for same reason not having {0,1,0} in configuration... We also know the subset was reduced from 6 to 4 when I started to increase further validation. Perhaps this validation as shown above (comparing stepStore needs to be removed again). Just to understand again why this was introduced as part of Test Case 3.... I will try to re-visit this test case and see what

influenced my decision making.

```
int [ ][ ][ ] triangle = new int[ ][ ][ ] {
                                    {        {1},      {4}          },
                                    {       {2,3},    {6,8}         },
                                    {     {1,5,1},  {9,2,3}         },
```

```
*************SUMMARY HIGHEST RESULTS*************
*******TRIANGLES*********
***There are triangles configured*** : 2

[[1], [4]]
[[2, 3], [6, 8]]
[[1, 5, 1], [9, 2, 3]]
***************

Triangle: 0 | Subset:5 (1,3,5) (Indexes: [0][0][0],[1][0][1],[2][0][1]) | Max value: 9
Triangle: 1 | Subset:3 (4,6,9) (Indexes: [0][1][0],[1][1][0],[2][1][0]) | Max value: 19
```

I have taken a step back for a second and realised that whilst I was presented with so much numerical output, I had misled myself into looking at the System output of the array and confused my mindset.

If we examine the example above, it can appears that 4=>8=>9 are adjacent. This is because it is just not possible to format the content in the array.

BUT if analysing the triangle arrangement (which is exactly what was required), it can be seen that 4=>8=>9 is not adjacent (index 0,1,0). **It can now be seen that it was correctly removed!**

A bit more obvious is 4=>6=>3 from the array output above. It can be visually seen that it is not adjacent. And this reflects index (0,0,2).

```
0,0,1
0,1,0
0,0,0
0,1,2
0,0,2
0,1,1
```

So infact after all these failed test cases, I consider all of them to be correct!

I can just continue a few more scenarios and this will be a complete solution.

TEST CASE 11 : Taking two triangles and incorrect configuration in second triangle ONLY

```
int [ ][ ][ ] triangle = new int[ ][ ][ ] {
                                    {        {1},      {4}          },
                                    {       {2,3},    {6,8,33}       },
                                    {      {1,5,1},  {9,2,3}         },

                                  };
```

```
TRIANGLE 0
Elements in: row 0 of triangle:  1
Value at triangle: 0  [0][0][0]:  1
Elements in: row 1 of triangle:  2
Value at triangle: 0  [1][0][0]:  2
Elements in: row 2 of triangle:  3
Value at triangle: 0  [2][0][1]:  5
*******TOTAL: 8

TRIANGLE 1
Elements in: row 0 of triangle:  1
Value at triangle: 1  [0][1][0]:  4
Elements in: row 1 of triangle:  3
Ensure correct triangle configuration.
```

I now have exhausted all testing.. Only will explore larger triangle arrangements... in a triple triangle configuration and also with failed configuration...

TEST CASE 12 : Taking three triangles with several rows

```
*************SUMMARY HIGHEST RESULTS*************
*******TRIANGLES*********
***There are triangles configured*** : 3

[[1], [4], [4]]
[[2, 3], [6, 8], [5, 6]]
[[1, 5, 1], [9, 2, 3], [7, 8, 9]]
[[15, 7, 9, 12], [5, 9, 15, 8], [2, 7, 11, 8]]
[[11, 6, 7, 9, 3], [9, 5, 3, 6, 10], [1, 5, 6, 2, 9]]
****************
```

```
Triangle: 0 | Subset:3 (1,2,1,15,11) (Indexes: [0][0][0],[1][0][0],[2][0][0],[3][0][0],[4]
[0][0]) | Max value: 30
Triangle: 1 | Subset:6 (4,8,3,15,6) (Indexes: [0][1][0],[1][1][1],[2][1][2],[3][1][2],[4]
[1][3]) | Max value: 36
Triangle: 2 | Subset:4 (4,6,9,11,6) (Indexes: [0][2][0],[1][2][1],[2][2][2],[3][2][2],[4]
[2][2]) | Max value: 36
Triangle: 2 | Subset:8 (4,6,9,8,9) (Indexes: [0][2][0],[1][2][1],[2][2][2],[3][2][3],[4][2]
[4]) | Max value: 36
```

```
s is defining two triangles, it can be increased
][ ][ ] triangle = new int[ ][ ][ ] {
                    {        {1},          {4},          {4}
                    {       {2,3},        {6,8},        {5,6}
                    {      {1,5,1},      {9,2,3},      {7,8,9}
                    {    {15,7,9,12},  {5,9,15,8},  {2,7,11,8}
                    {   {11,6,7,9, 3}, {9,5,3, 6,10},{1,5,6, 2,9}
```

It can be seen the congestion in the array output.
As a final measure, I am validating to ensure that subset 4, subset 7, subset 5 and subset 9 are infact correct.

```
Subset 1: 0,0,0,1,1
Subset 2: 0,0,0,1,2
Subset 3: 0,0,0,0,0
Subset 4: 0,1,2,2,2
Subset 5: 0,0,0,0,1
Subset 6: 0,1,2,2,3
Subset 7: 0,1,2,3,3
Subset 8: 0,1,2,3,4
Subset 9: 0,1,1,2,2
Subset 10: 0,1,1,2,3
Subset 11: 0,1,1,1,1
Subset 12: 0,1,1,1,2
Subset 13: 0,0,1,1,1
Subset 14: 0,0,1,1,2
Subset 15: 0,0,1,2,2
Subset 16: 0,0,1,2,3
```

NOTE: I introduced padding in the code otherwise it is very possible I will get confused for what is functional

Subset 3 on Triangle 0 = 30
(these are adjacent)

Triangle[0][0]**[0]** = 1
Triangle[1][0]**[0]** = 2
Triangle[2][0]**[0]** = 1
Triangle[3][0]**[0]** = 15
Triangle[4][0]**[0]** = 11

Subset 6 on Triangle 1 =36
(these are adjacent)

Triangle[0][1]**[0]** = 4
Triangle[1][1]**[1]** = 8
Triangle[2][1]**[2]** = 3
Triangle[3][1]**[2]** = 15
Triangle[4][1]**[3]** = 6

Subset 4 on Triangle 2 = 36
(these are adjacent)

Triangle[0][2]**[0]** = 4
Triangle[1][2]**[1]** = 6
Triangle[2][2]**[2]** = 9
Triangle[3][2]**[2]** = 11
Triangle[4][2]**[2]** = 6

Subset 8 on Triangle 2 = 36

Triangle[0][2]**[0]** = 4
Triangle[1][2]**[1]** = 6
Triangle[2][2]**[2]** = 9
Triangle[3][2]**[3]** = 8
Triangle[4][2]**[4]** = 9

TEST CASE 13 : FINAL TEST – Different size triangles

```
triangle = new int[ ][ ][ ] {
                    {        {1},          {4},        {4}
                    {       {2,3},        {6,8},      {5,6}
                    {      {1,5,1},      {9,2,3},    {7,8,9}
                    {    {15,7,9,12},  {5,9,15,8},
                    {   {11,6,7,9, 3},

                        };
```

I am expecting this to fail since it has performed same execution at beginning rowsTriangle which assists in performing lots logic.
If I was to extend this software code further, I could look into permitting heterogeneous triangle rows.
But at moment, it is just best to catch this exception safely.

```
TRIANGLE 1
Elements in: row 0 of triangle:  1
Value at triangle: 1  [0][1][0]:  4
Elements in: row 1 of triangle:  2
Value at triangle: 1  [1][1][0]:  6
Elements in: row 2 of triangle:  3
Value at triangle: 1  [2][1][0]:  9
Elements in: row 3 of triangle:  4
Value at triangle: 1  [3][1][1]:  9
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: Index 1 out of bounds for length 1
        at Staircase.performMoves(Permutation.java:455)
        at Staircase.obtainMoves(Permutation.java:427)
        at Staircase.<init>(Permutation.java:387)
        at Permutation.main(Permutation.java:711)
```

It is expecting row here that does not exist

Additional code:

```
try
{
System.out.println("Elements in: " +  "row " + i + " of triangle:  " + triangle[
}
catch (ArrayIndexOutOfBoundsException e)
{
    System.out.println("There are no elements in row: " + i + " on this triangle
    System.exit(0);
}
```