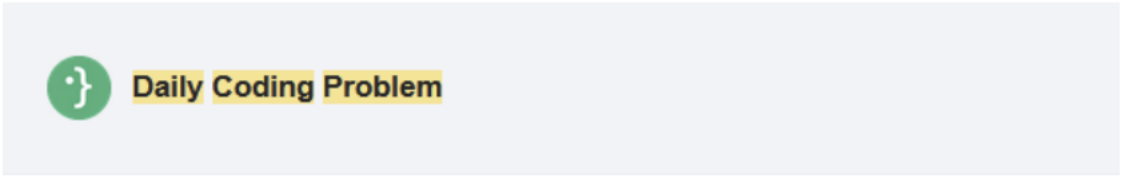


Good morning! Here's your coding interview problem for today.

This problem was asked by Google.

You are given an array of arrays of integers, where each array corresponds to a row in a triangle of numbers. For example, `[[1], [2, 3], [1, 5, 1]]` represents the triangle:

We define a path in the triangle to start at the top and go down one row at a time to an adjacent value, eventually ending with an entry on the bottom row. For example, 1 -> 3 -> 5. The weight of the path is the sum of the entries.

Write a program that returns the weight of the maximum weight path.

Again, the initial part of the code resembles understanding the levels on the triangle.
I can only foresee a solution by performing a for loop at each level of the triangle.

In terms of number of understanding the 3D array again, it is conceptually similar to the below:

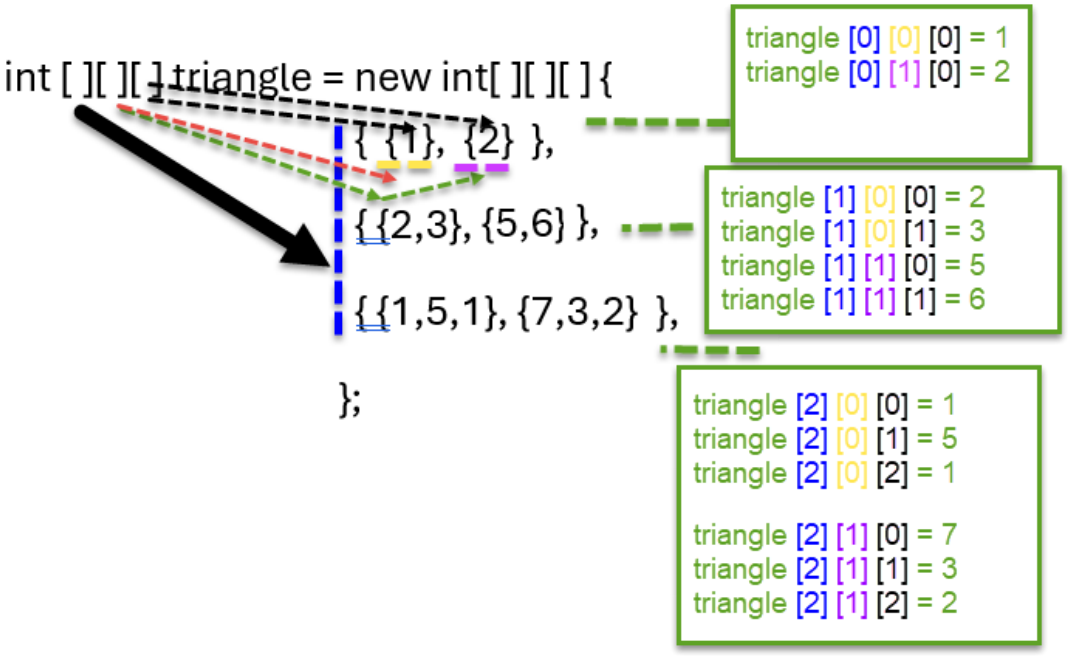
```
int [ ][ ][ ] gg = new int[ ][ ][ ] {
                        { {1} },
                        {{2,3} },
                        {{1,5,1} },
                       };
```

We know individual values are stored at this index int [ ][ ][X]
We know to access entire level of the triangle, it is accessed here int [ ][X][ ].
We know that identification of the required level is based on tier of the triangle. This is accessed via: int [X][ ][ ]

All in all its quite straight forward, should it required that multiple triangles are preloaded in the configuration, this is also possible in order to save lots of additional lines of code in the initialisation…

```
int [ ][ ][ ] triangle = new int[ ][ ][ ] {
                          { {1},  {2} },
                          {{2,3}, {5,6} },
                          {{1,5,1}, {7,3,2} },
                         };
```

The below shows in a colour coded fashion on how each item is accessible.



This is a major part in calculating the total from top to bottom.

In terms of each tier, the selection widens progressing downwards.
So this can only be completed.
The selection this time is based on the width of each tier.

We know this is the navy blue component

If there was <u>one</u> triangle stored in the configuration, it would be bit more straight forward:

triangle[0].length
triangle[1].length
up to n tiers
triangle[n].length

If there were <u>multiple</u> triangle stored in the configuration, it would be slightly more involved.

<u>For first triangle:</u>	**<u>For second triangle:</u>**
triangle[0][0].length	triangle[0][1].length
triangle[1] [0].length	triangle[1] [1].length
up to n tiers	up to n tiers
triangle[n][0].length	triangle[n][1].length

The indexes available ascertains the selection.

0 tier: triangle[0][0].length = 1

So, there will only be a single selection available from {0} due to zero index nature.

1 tier: triangle[1][0].length = 2

So, there will only be two selections available from {0,1} due to zero index nature.

nth tier: triangle[n][0].length = n+1

So, there will be n selection available from {0,n} due to zero index nature.

At this moment, it is not adhering to any of previous coding since the index of element can be denoted by the same number.

The following [ ] [ ] [X] index is of interest. This will be {0,n}

I think for this example, it is worthwhile starting with the statistical calculator and building outwards again to ensure there is no stale code.
Alternatively I will use the Google challenge completed on:
Friday 22 November 2024 & Monday 2 December 2024

What I know so far:
I will fill an array lets say X[] with values 0,1,2,n
I will perform a permutational calculation (with replacement) based on following:
P(3,3)
This will allow me to set the do while loop for the expected set size().