28th Jan 2021

Neo4j Technical assessment

Submission commentary Jonathan Brincat

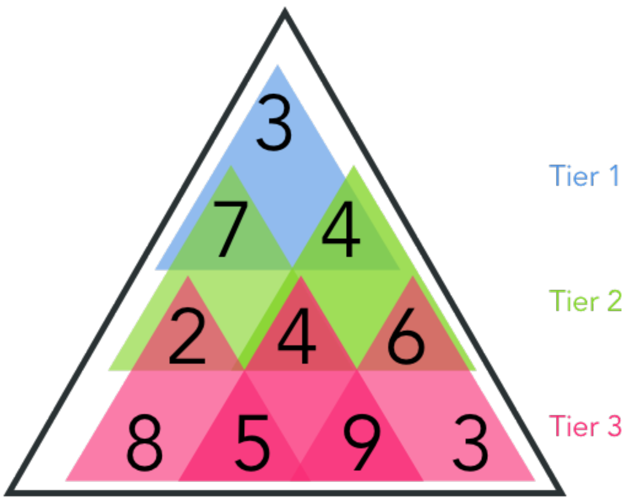
https://codesandbox.io/s/neo4j-technical-assessment-xtl74

The pyramid structure here is a bit of a red herring that will snare those with a fixed mindset. If you elect to focus at the natural starting point i.e. the top of the pyramid, then you will engage in a fool's errand. The only way to achieve a solution thinking this way is by brute force. Traversing every possible route down the tree to recursively solve every possible combination – storing the output in an Array discarding the highest value if a lower one is found on each pass. This would be a greedy approach that is computationally very expensive at scale. This is because starting from the beginning you don't benefit from hindsight. You can't safely assume the sum of the lowest combination at a particular level trumps all ahead because a higher sum may actually grant access to a lower overall path sum. So consequently you have to proof every plausible route. But why care about looking ahead if you can look behind? After all, the problem is laid bare in completeness from beginning to end. No. Rather you need to turn the pyramid on it's head and work from the bottom upwards to eliminate redundancies in operations and to claim any efficiencies in process. This I reasoned pretty quickly.

Bearing in mind, regardless of the proposed solution these factors remain constant.

- The first two rows is always a single segment with a fixed outcome.
- The last row is superfluous and likely doesn't need to be encapsulated - it will be absorbed by the preceding row's computation.

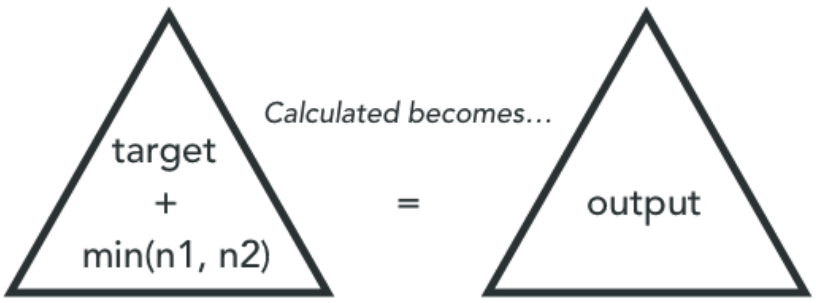
If you ignore the comprising triangle of the pyramid itself you can dissect the problem into granular triangular segments that individually form discreet units of this problem. Each unit represents a sum that produces an output. If you think about it this way then that numeric output is merely a step in the sequence and you can use this mantra to systematically flatten the parent pyramid - negating the need to wastefully traverse combinations that would result in a higher numeric output.



What I mean by flattening the pyramid is probably best illustrated as at this point I baulked with the realisation this was actually a mathematical exercise which was probably best teased with some frenzied chicken scratch before I even attempted to write a line of code…

For each segment you are effectively dealing with a simple summing operation, albeit there is an ancillary step that filters the smallest value from a pair – however this is trivial to the critical thinking. This can be represented as so:

x = target + min(n1, n2)



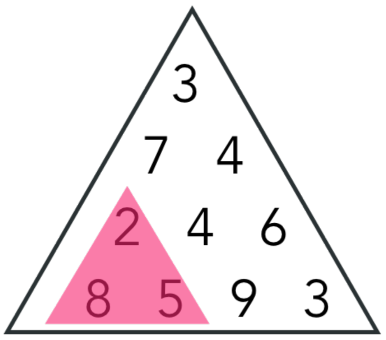
Expanding this further to evaluate a series of these segments laid out in columns this becomes:

x[col] = target[col] + min(n[col], n[col+1])

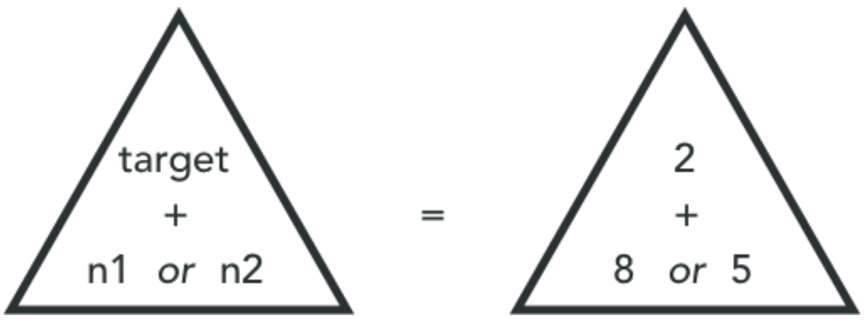
Adding a further dimension with rows to accommodate segments that form the tiers of a pyramid this then becomes:

x[row][col] = target[row][col] + min(n[row+1][col], n[row+1][col+1])

<u>Worked example</u>



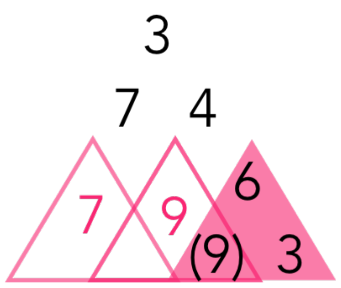
Starting from the bottom and taking the first segment.



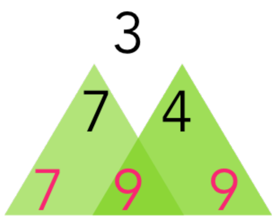
Passed through the x = target + min(n1, n2) operation this becomes

x = 2 + min(8, 5) **5 is the lower value. 8 will be discarded. Operation is 2 + 5.*
x = 7

And for brevity I will execute the second segment too but skip the workings. So now out pyramid looks like this

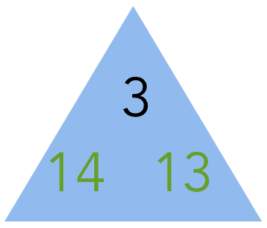


The output from these two operations can now replace the values previously contained in those two segments. Notice that by doing so we have more or less eliminated the 4th row and 'flattened' the pyramid. The completed first pass would look like this:



Now we can repeat this procedure onwards and upwards until we get to the top of the pyramid.

Second sweep
x[i] = 7 + min(7, 9) = 14
x[i] = 4 + min(9, 9) = 13



Although I neglected to show **x** assigned to a keyed reference in the very first example just to keep it clean this would be the case as we worked towards the top. However it is not necessary to retain previous calculated outputs these can be overwritten so the second 'row' key can actually be discarded.

Final pass
x[i] = 3 + min(14, 13) = 16

Result = 16 is the path of least resistance!