CallMeKnight

dotnet chapter challenge #007 solution
by Illia Levandovskyi

C# coding challenge

Luxoft
A DXC Technology Company

The problem

Have you ever wondered how hard it is to live as a chess knight in the modern world? How to dial a number and make a phone call? And is it possible to dial the required number?

Task: How many distinct numbers can you dial?

Rules:

- You can start from any position.
- You can move as a chess knight.
- You can make exactly N hops.
- You are the knight!

Example:

- N is zero: 0 numbers.
- N is one: 10 numbers.
- N is two: let's count, 1-6, 2-9, 9-2, 0-4, ... many!

Goal:

Implement a function to count distinct dialable numbers for given N.

The map

```
1 2 3
4 5 6
7 8 9
  0
```

The knight uses a classic dial pad with only numbers.
A phone number is a simple sequence of digits of length N.

Naive solution

Make actual steps

Write down the path - digits

Select unique numbers

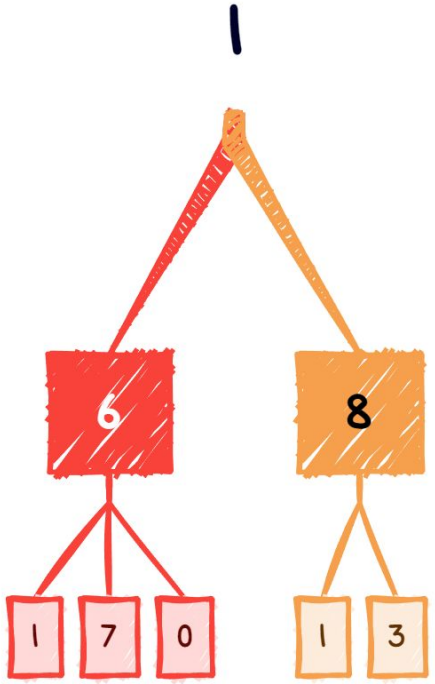
Naive solution takes about 67 seconds
and consumes up to 25GB for 20 steps route.

Naive solution insights

Knight steps form a tree

Each number is a unique path on the tree from the root to the leaf

Is the amount of numbers equal to the amount of leaf nodes?



Leaf-based solution

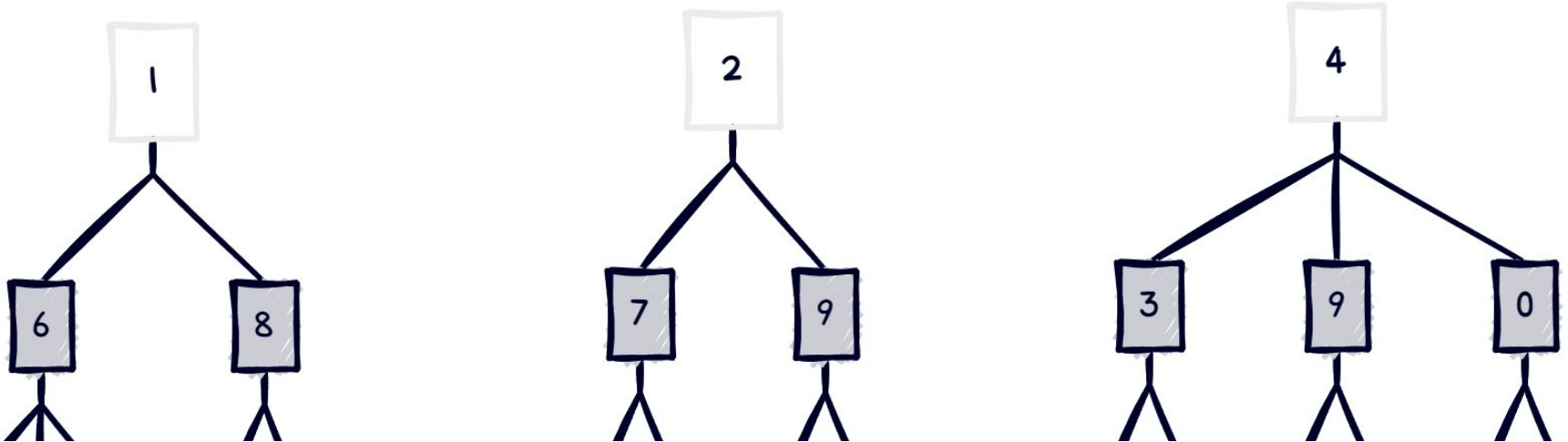
Create a tree of nodes based on the Knight moves

Count nodes on the target level N.

Leaf-based solution takes 13 seconds and 11 GB for 20 steps route.

Leaf-based insights

We do not need to calculate next step every time - it could be precalculated



Predefined steps solution

Write down a map of next locations with respect to a given one.

Predefined steps solution takes 391 ms and 1.37 GB for 20 steps route.

Predefined steps solution insights

Do we really need Nodes and Locations?

Sequences solution

Simplify the tree - rely on the digits themself only.

Keep track of leaf number on a given level and process as many levels as needed (N).

Sequences solution takes 165 ms and 176 MB for 20 steps route.

Sequences solution with immutable arrays takes 150 ms and 176 MB for 20 steps route.

Sequences solution insights

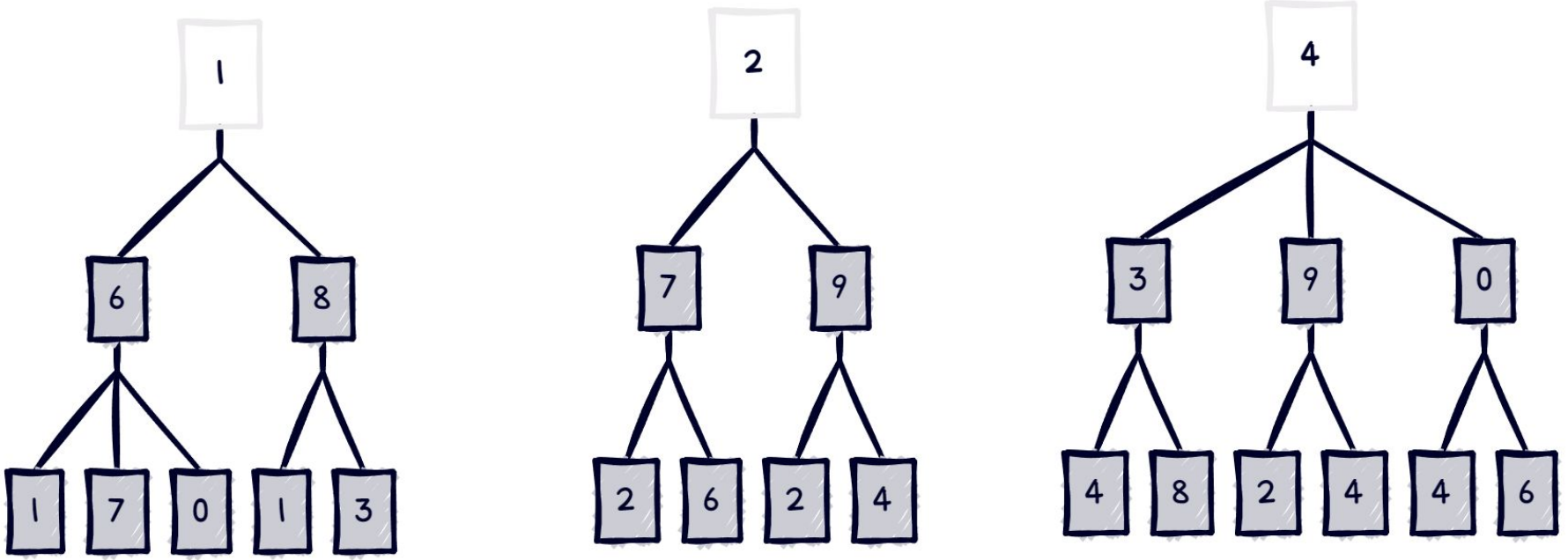
Do we really need to build the last level?

- The growth of the payload is exponential O(2.2 ^ N)
- Thus the last level costs as much as almost all levels before.

Can we enhance predefined steps?

- Knowing beforehand one step in advance is mandatory
- But knowing beforehand two steps upfront is cool!

Predefine 2 steps beforehand



Sequence 2 steps solution

Predefine 1 step upfront digits and 2 steps upfront digits.

Go over 2 steps on one iteration using the later predefined mapping.

Do not construct the last 2 levels, count the leaf amount instead.

Sequence 2 steps solution takes 114ms and 72 MB for 20 steps route.

Performance overview

Method	N	Mean	Allocated
Calculate	20	67.5 s	25 GB
CalculateLeafs	20	13.4 s	11 GB
CalculateLeafsPredefinedSteps	20	392 ms	1.37 GB
CalculateSequences	20	165 ms	176 MB
CalculateSequencesImmutable	20	150 ms	176 MB
CalculateSequences2Steps	20	115 ms	72 MB

The 2Steps solution -> 19_453_763_584 unique numbers for 27 steps in 17s and consumes 8.5 GB RAM.

BenchmarkDotNet v0.14.0, Windows 11 (10.0.22631.4460/23H2/2023Update/SunValley3)

Intel Core i7-8750H CPU 2.20GHz (Coffee Lake), 1 CPU, 12 logical and 6 physical cores

.NET SDK 9.0.100

Ideas so far

Every unique number is a path from root to leaf in a tree of Knight hops.

Predefined steps as a mapping (frozen dictionary) between initial digit and the next digits.

Store in advance the mapping for 2 hops and generate not each, but every second level.

Do not construct the last level (N, or 2), just count the number of leafs on it.

Use minimalistic data type (byte).

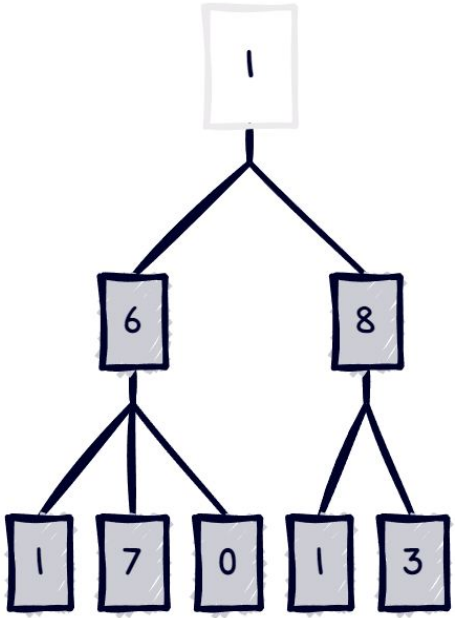
Run each initial digit in parallel.

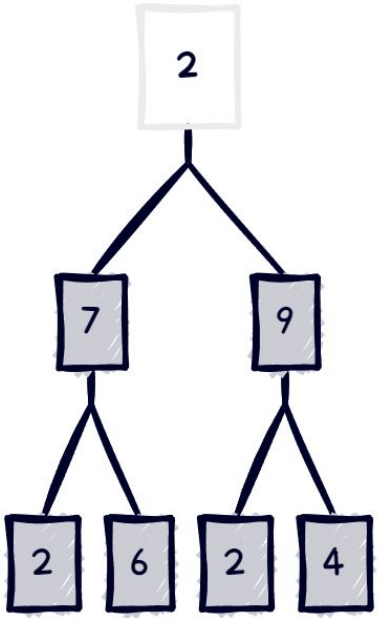
Solution complexity is O((20/9)^N).

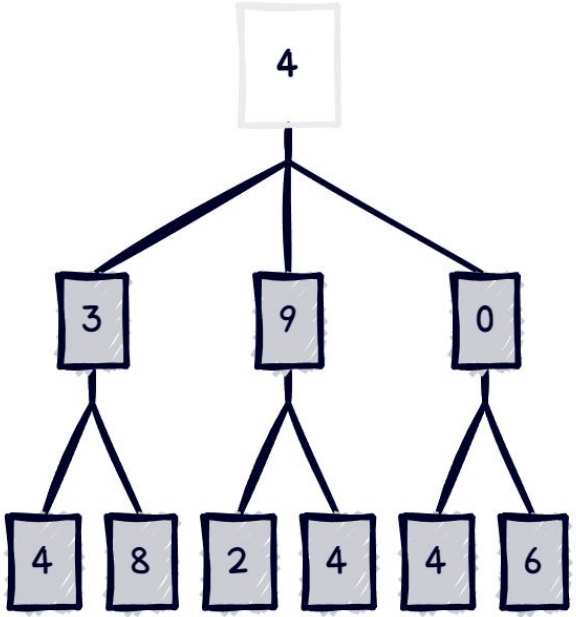
Do we really need to generate any trees, layers, nodes?

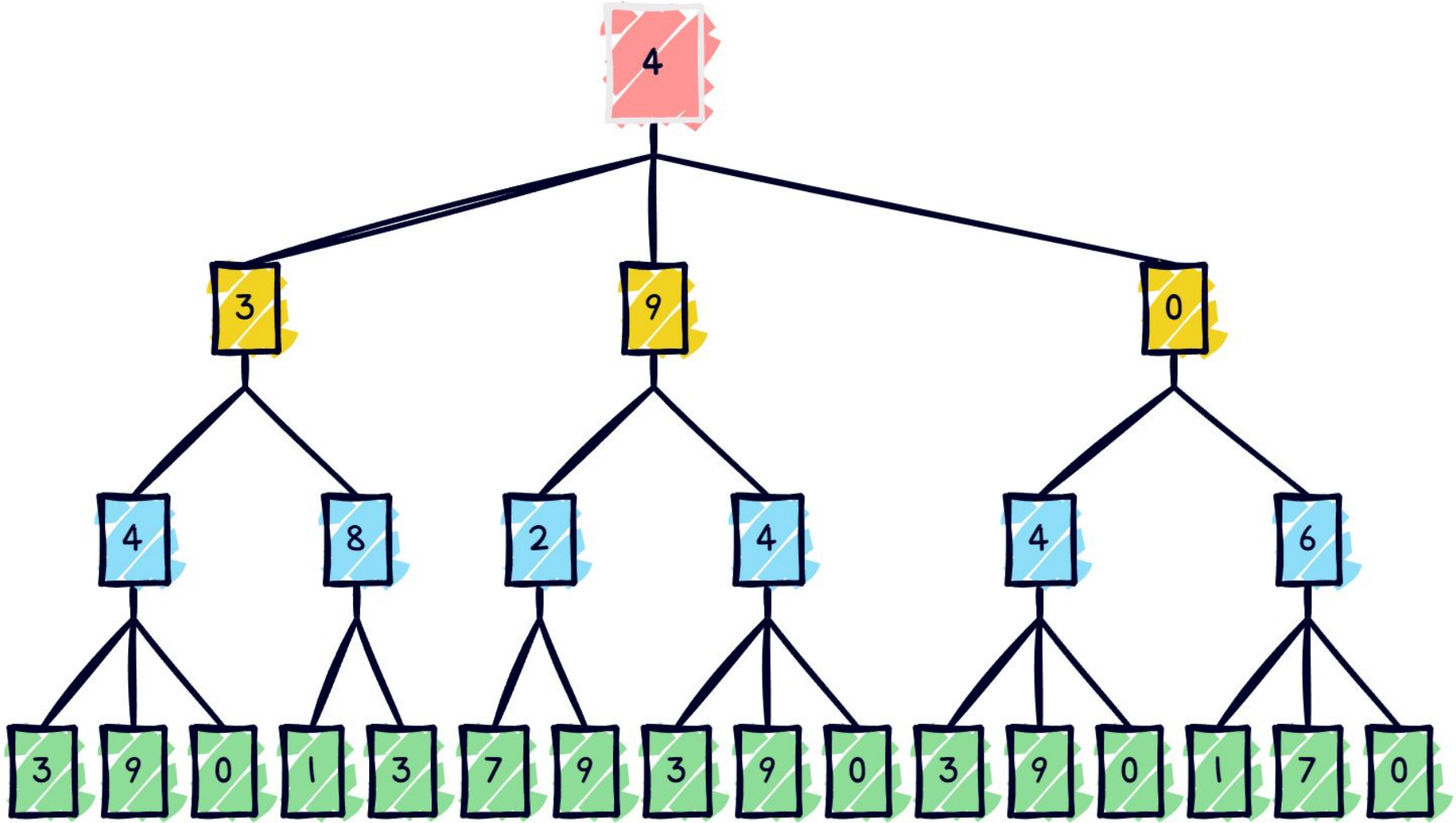
Frequency solution

Every digit allows to produce certain amount of next digits.
Just keep track of next level digits in a **frequency table**!








4
3
9
0
4
8
2
4
4
6
3
9
0
1
3
7
9
3
9
0
3
9
0
1
7
0

Frequency solution result

Frequency solution takes 80 um and 113 KB for 25 step route.
And 0.5s and 113 KB for 10_000 step route (to print the result, mostly).

The solution has time complexity O(N) and space complexity O(1).