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

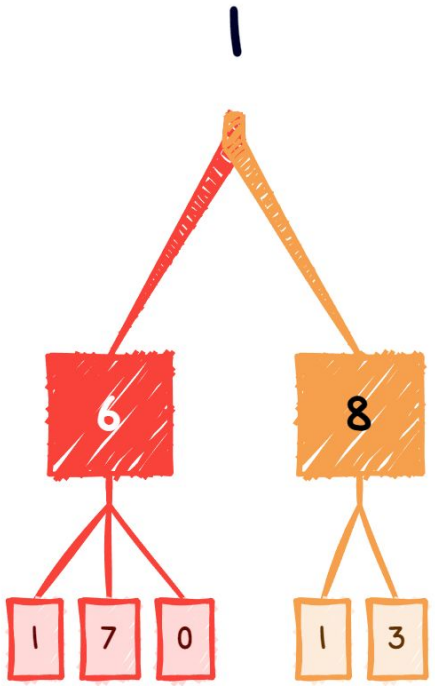
Naive solution takes about 67 seconds
and consumes up to 25GB for 20 steps route.

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?

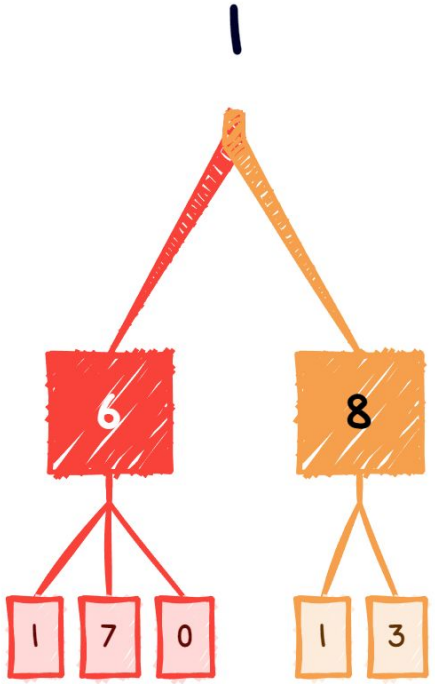


Leaf-based solution

Create a tree of nodes based on the Knight moves

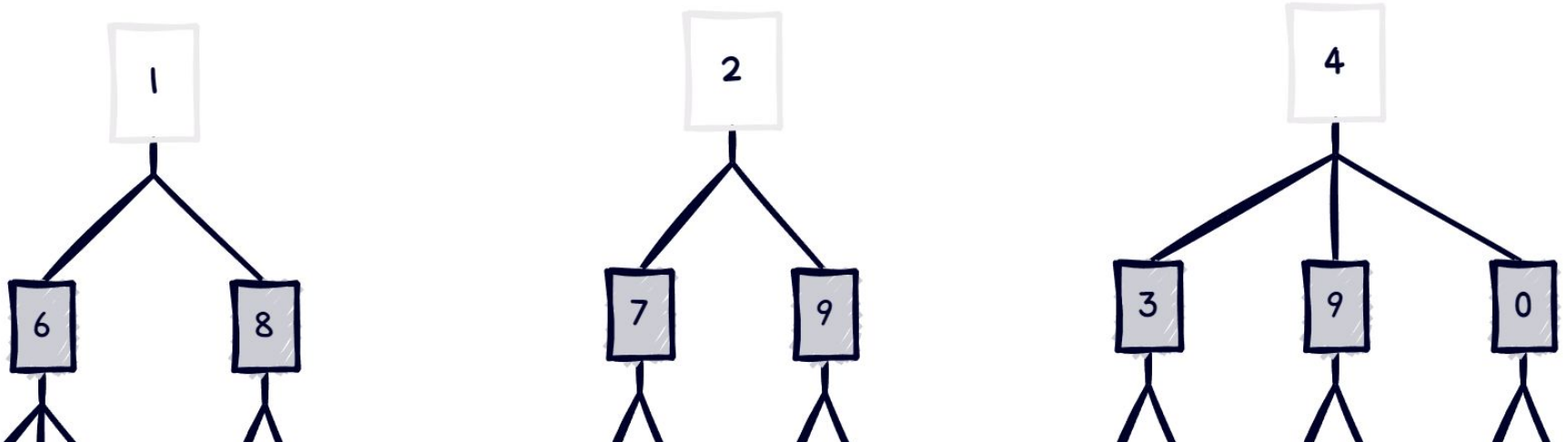
Count nodes on the target level N.

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**



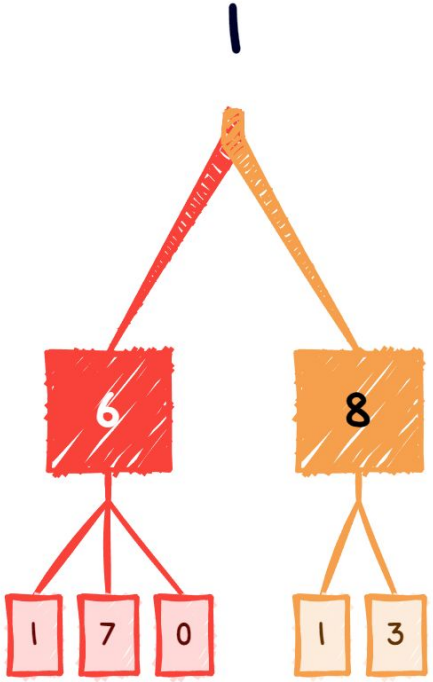
Predefined steps solution

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?



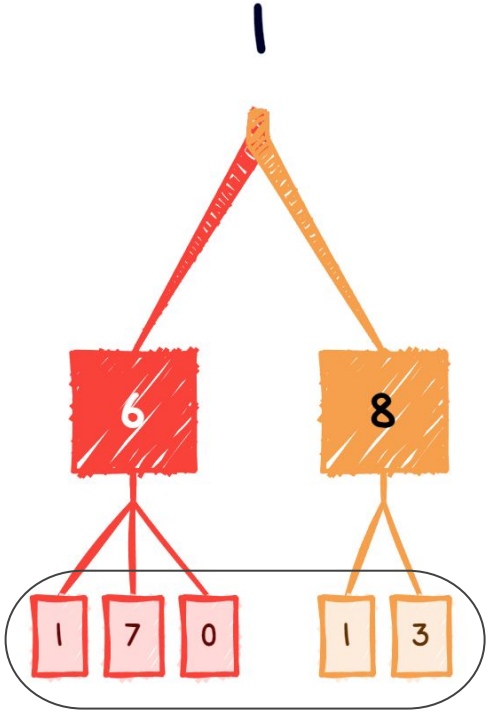
Sequences solution

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.

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

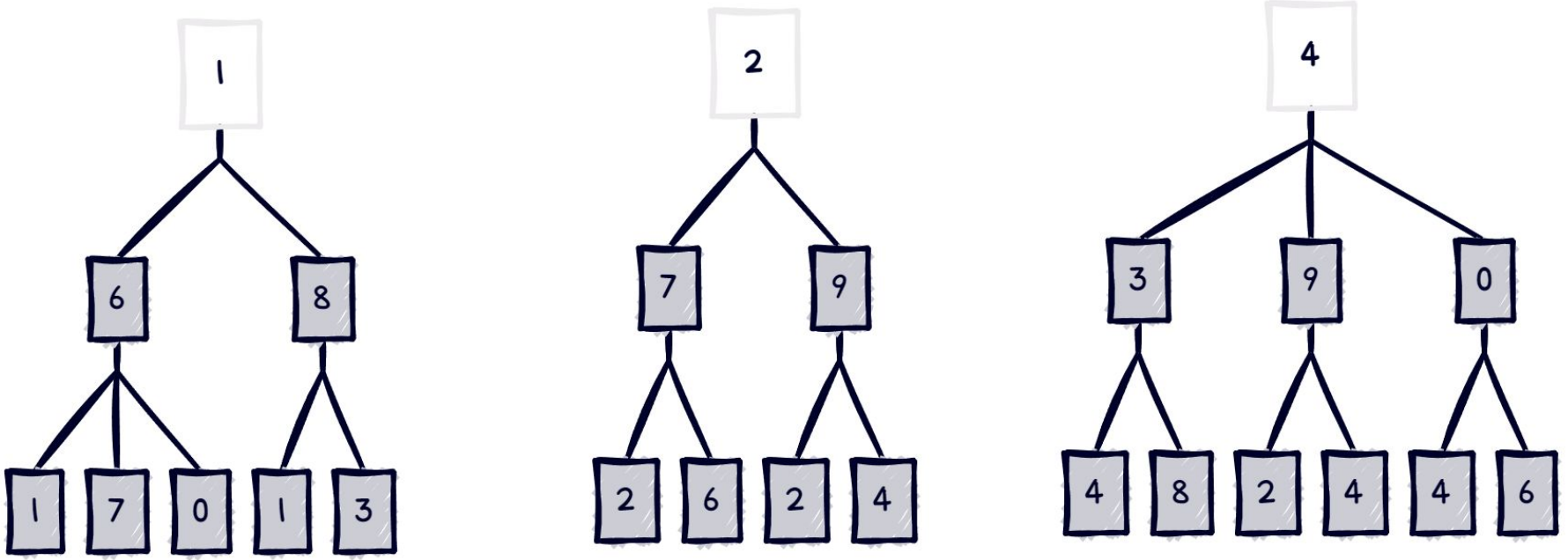
Do we really need to build the **last** level?

- 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.

Summary

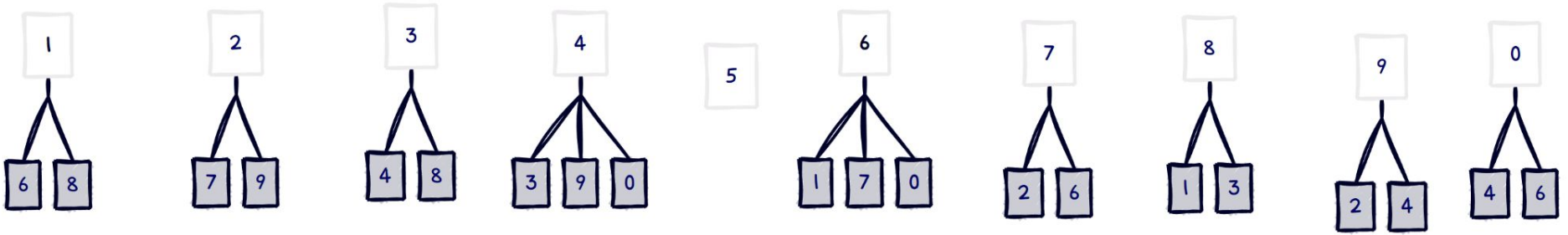
- Every unique number is a **path** from root to leaf in a tree of Knight hops.
- **Predefined** steps as a mapping (frozen dictionary): given digit and the next ones.
- Store in advance the mapping for **2 hops** and generate not each, but every second level.
- Do **not construct** the last level (N, or 2), just count the number of leafs on it.
- Use minimalistic data type (**byte**) for digits.
- Run each initial digit in **parallel**.
- Solution time and memory **complexity** is O((20/9)^N).

~~Summary~~ Eureka moment!

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

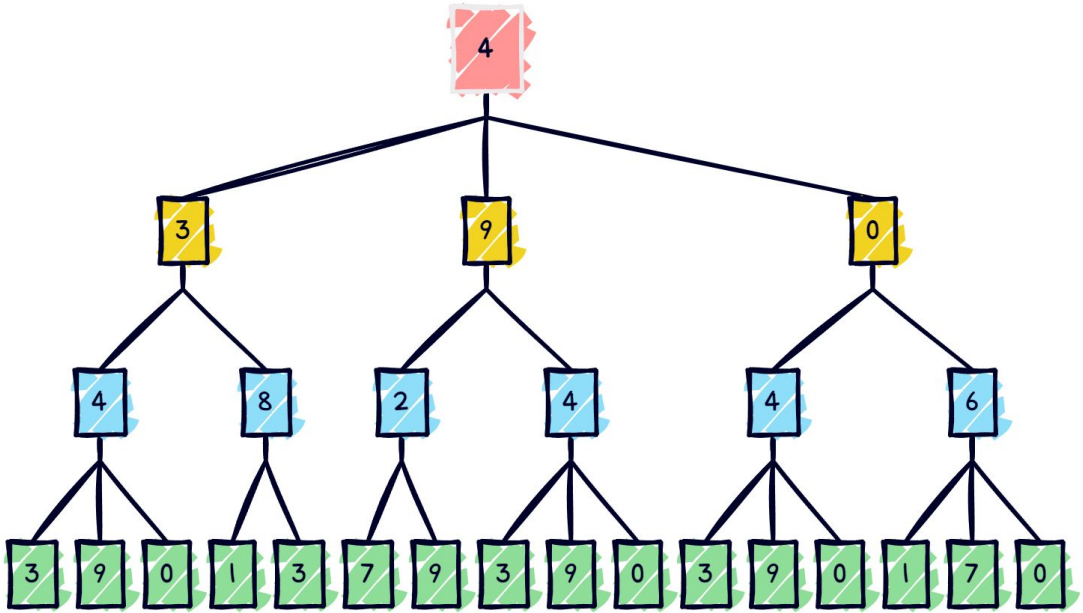
Frequency solution

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



Frequency solution: level 3 to 4

level3	amount
1	0
2	1
3	0
4	3
6	1
7	0
8	1
9	0
0	0



level4	amount
1	1+1
2	0
3	3+1
4	0
6	0
7	1+1
8	0
9	3+1
0	3+1

Frequency solution result

Frequency solution takes 17 um and 105 KB for 20 steps route (ulong data type).

0.5s for 10_000 steps route.

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

How many unique numbers could be generated in 10000 steps? (3596-digit number)

40261387160894992171908251341262275474301766569816850704240848158529013006220362865338766377375106913877549576326715738212070657673516310209189411692637
34022452473177812310765367130973882375044834000023208707938372192001121105060211714343615607158327257058794763309137913527096739107921659127937130806612
81636651142206245394873483299946472056099956275521604906644133102869439781522047276132792860551197781308259786557655633831092364240635808049392476804298
24189196645188838434780341112778262001189333541964121465654637529195649748555804295025983640991745755837712901797918788483625453345686655571154830086371
13324029278818957938532739910364119500337541981685082880419278356471857747568731879472777606354327119274267121694002534302976152367073644849174116728683
83127295575484868782284381868577652416138042451912046845633034261370799737960348847503852087130187788999499543016801762862243470238416078910412236881155
99278519316340916620795926496069466983007310236229231265784283030512647495101669844128143962473745998934404462183671376267785628203617947888449736693067
78622587064656677431930547395665081245886643798424555237871816266159517265732271554401571170048557139087826390916215417775242948290518886685796392205000
72734916942281826162731766047483535165902235823164948179185755854559582807313155229099099786694124662998311478612038052653995634363762376630597700958107
55430792329471235604365105072297005495873306225236612462458103231898419463031085051011137959118907351467829320566347534142260311407521856332647899234869
10577005445837993165776602760006750331105914374431918539895829292285997139598280243660868308834687805444666302360401979991085846449857470643508859405349
12881315625350938705491855723512125360116305643368582358065298999496126196844054475117236876461733048465409102660658606446132576985829188089404443553969
08734530578426603046006856278141786302082893575924600037128682461251980073224493343967696588343352779754463661581882887277247210248218168179456522738621
19699236855665649774178134230886523384730906908717072796120916011969245227465931322623060312352828313915306545151978758849635799536126038562323512314673
88067449878479313000003383171112691623843038290634190167749415502623576208872568015623351705507585089452911207216064897305528698468168835035825346603878
85327531175292691138138913692532332632072370814913984809594495095714523686939457566572294512245209291516415505489886220761962247649802067540667211929640
40746647920223022505885340309348748021419588840695954112622742968048173924789190072021363877712829500318860853989633357661846424093731313763368861075390
58609695978957404589558749575166257503332596553917579010679191931498830924729832817317027074232617824446250421329628090775536306550358418803999654026287
83975612282354282155983427203893394204772997981881221495156622837536975646246811881832800216103289212202000639457165234937753067087242201048872589678395
04745015742482058723995067533603623868625372898959427377979188785776707328259554251731816122716921887412670065150412456549250509096763342703797870257174
75525447490214218151224993019071414499051794644954172990577009522851396663344479809312381418360171289868217641455924296177302668410438972720258597081033
10842263929374640619462224230966907638268365001572276964015079971420758405381425517491123126288224693547732351524045082369322899084418298785909789542693
41324290123108531014006670266439314598729454838897388403102970183452390212785183790427596826457908677932726687970302349806469307191691188751583593499606
69326835635045377811596508596257537492045820593540028009223790525842359791872291514895069040649175 04

Lessons learnt

1. calculating the number of something
   and
   generating something to count it

   are completely different tasks (linear vs exponential complexity)

2. iterative optimization allows to understand the problem in greater depth
   and
   find essentially better solution