Question: https://leetcode.com/problems/knight-dialer/

[0] If k is at index (i, j), then in a single hop, k can move to 8 possible positions which are below.

a (i - 1, j - 2)

b (i - 2, j - 1)

c (i - 2, j + 1)

d (i - 1, j + 2)

e (i + 1, j + 2)

f (i + 2, j + 1)

g (i + 2, j - 1)

h (i + 1, j - 2)

Math behind the solution:

Consider a function paths(i, j, n) which calculates the total number of unique paths to reach index (i, j) for a given n, where n is the number of hops. From [0] or [1], we can recusively define paths(i, j, n) for all non-trivial (n > 1, that is, more than one hop) cases as follows,

paths(i, j, n) = paths(i - 1, j - 2, n - 1) +

paths (i - 2, j - 1, n - 1) +

paths (i - 2, j + 1, n - 1) +

paths (i - 1, j + 2, n - 1) +

paths (i + 1, j + 2, n - 1) +

paths (i + 2, j + 1, n - 1) +

paths (i + 2, j - 1, n - 1) +

paths (i + 1, j - 2, n - 1)

If we translate this to plain english, all we are saying is "the total number of unique paths to (i, j) for certain hops n is equal to the sum of total number of unique paths to each valid position from which (i, j) can be reached using n - 1 hops".

If you are confused why it is n - 1 hops, note that when we are at (i, j), we already made one hop and we have n - 1 hops more to take.

For the trivial case (n = 1, that is no hops), the problem states that this must be considered as one path. Therefore, paths(i, j, n) = 1, for n = 1.

Naive Recursive Code

public static final int max = (int) Math.pow(10, 9) + 7;

public int knightDialer(int n) {

long s = 0;

//do n hops from every i, j index (the very requirement of the problem)

for(int i = 0; i < 4; i++) {

for(int j = 0; j < 3; j++) {

s = (s + paths(i, j, n)) % max;

}

}

return (int) s;

}

private long paths(int i, int j, int n) {

// if the knight hops outside of the matrix or to \* return 0

//as there are no unique paths from here

if(i < 0 || j < 0 || i >= 4 || j >= 3 || (i == 3 && j != 1)) return 0;

//trivial case

if(n == 1) return 1;

//non trivial case

long s = paths(i - 1, j - 2, n - 1) % max + // jump to a

paths(i - 2, j - 1, n - 1) % max + // jump to b

paths(i - 2, j + 1, n - 1) % max + // jump to c

paths(i - 1, j + 2, n - 1) % max + // jump to d

paths(i + 1, j + 2, n - 1) % max + // jump to e

paths(i + 2, j + 1, n - 1) % max + // jump to f

paths(i + 2, j - 1, n - 1) % max + // jump to g

paths(i + 1, j - 2, n - 1) % max; // jump to h

return s;

}

If you run this code for n = 50 in your favorite programming language, you will realize that it takes at least an hour to get the answer.

This is because this problem not only has similar subproblems but each of those similar subproblems have overlapping subproblems.

So we use a 3 dimensional array to memorize the solved subproblems.

Below is the code.

public static final int max = (int) Math.pow(10, 9) + 7;

public int knightDialer(int n) {

// A 3D array to store the solutions to the subproblems

long M[][][] = new long[n + 1][4][3];

long s = 0;

//do n hops from every i, j index (the very requirement of the problem)

for(int i = 0; i < 4; i++) {

for(int j = 0; j < 3; j++) {

s = (s + paths(M, i, j, n)) % max;

}

}

return (int) s;

}

private long paths(long[][][] M, int i, int j, int n) {

// if the knight hops outside of the matrix or to \* return 0

//as there are no unique paths from here

if(i < 0 || j < 0 || i >= 4 || j >= 3 || (i == 3 && j != 1)) return 0;

if(n == 1) return 1;

//if the subproblem's solution is already computed, then return it

if(M[n][i][j] > 0) return M[n][i][j];

//else compute the subproblem's solution and save it in memory

M[n][i][j] = paths(M, i - 1, j - 2, n - 1) % max + // jump to a

paths(M, i - 2, j - 1, n - 1) % max + // jump to b

paths(M, i - 2, j + 1, n - 1) % max + // jump to c

paths(M, i - 1, j + 2, n - 1) % max + // jump to d

paths(M, i + 1, j + 2, n - 1) % max + // jump to e

paths(M, i + 2, j + 1, n - 1) % max + // jump to f

paths(M, i + 2, j - 1, n - 1) % max + // jump to g

paths(M, i + 1, j - 2, n - 1) % max; // jump to h

return M[n][i][j];

}

Github Link :<https://lnkd.in/ecwtJeaz>