

Problem 2

Buni Rabbit enjoys hopping up and down a set of stairs, one step at a time. In how many ways can Buni start on the ground, make a sequence of $2k$ hops, and end up back on the ground?

1. a)

$k = 3$ (10 points)

2. b)

$k = 5$ (5 points)

(Bonus Question) How to solve the problem when $k = n$ ($n > 0$) using a recursive algorithm? You need to give the main ideas of your algorithm and the recursive formula.

(20 points)

Thoughts:

- the Bunny has $2k$ number of hops to make it back down the steps
- If the bunny goes all the way up and all the way down it can only go k steps up and k steps down
 - Must make an equal number k up and down steps

Each step you have the option to either go up or down

- restraints:
 - Cannot go down at step 0
 - Cannot go up at step k
 - must make equal number of up and down moves

a) k = 3 | 5 different paths

n = 3 | Number of paths: 5

Elapsed time: 1.6927719116210938e-05 seconds

UDUDUD

UDUDD

UDDUD

UUDUD

UUUDD

b) k = 5 | 42 different paths

Algorithm: BRUTE FORCE

input: n the number of stairs on the staircase

global: Count \leftarrow 0

actual_paths \leftarrow an initially empty array that will contain all valid paths the bunny can take.

Output: A tuple containing the list of paths and the count

def'n bunnyHop(n):

k \leftarrow 2 * n # need to validate the path of 2*n, where n is the height of the stairs

UP \leftarrow 1

DOWN \leftarrow -1

paths \leftarrow an empty array

if n is 0: # no paths when there are no steps

return paths

validPaths([], 0, k)

return paths

input:

current_path \leftarrow The current path being considered

balance \leftarrow The balance of number of UP and DOWN moves

Output:

None. Adds paths to final list iff they are valid paths

def'n validPaths(current_path, balance, n)

if length of current path == n

if balance == 0

count \leftarrow count + 1

actual_paths.append(current_path)

return # if the length is n, we are done with this path

if balance > 0: # if the balance is positive i.e. favors up moves down jumps are valid.

current_path.append(DOWN)

validPaths(current_path, balance - 1, n)

path.pop(). # removes the previous move because the path is complete.

This is the back tracing step. Once we have completed the path we now want to get all other permutations

current_path.append(UP)

validPaths(current_path, balance + 1, n)

current_path.pop()

The main idea of this algorithm is to brute force the different paths that can be made, and based off the balancing criteria determine if the path is valid or not. Its corrects, but far from optimal.

Runtime:

The runtime of this algorithm is based off the number of recursive calls that are being made, and the choices that are available to the decision tree. The bunny can either go up, or down at any given point when the step number is greater than 0. Given that the path is not just n but $2n$ due to the problem requiring the bunny to finish on the same point it started on the algorithm is $O(2^{(2n-2)})$ because it must iterate through the entire path length. the constant -2 is in the exponent because the first move and last move must always be an UP and DOWN more respectively.

Space Complexity:

At any given moment the current path being stored will be at a maximum of $2n$ in size, and the final paths can be defined as the length $2n*m$ where m is the number of resulting paths

$O(2n*m+2n)$ or $(2n*m)$

I also figured out a dynamic programming approach that runs in $O(n^2)$. It is not a recursive formula, but gets the same answer much quicker. Take a look in the attached python file.