Project 2: Dynamic vs. Exhaustive - Crane unloading problem

CPSC 335 - Algorithm Engineering

Spring 2023

Instructor: Himani Tawade

Everette D. Webber 886894732

GitHub Repository: <https://github.com/Digx7/Crane-Problem>

Exhaustive Algorithm Solution Pseudocode

Given max\_steps

Given empty vector allValidPaths

Given empty queue permutations

Given empty path start

Permutations.push(start)

While (permutations is not empty)

Path current = path at front of permutations

Var length = number of steps of current

If (length > 0)

allValidPaths.add(current)

if (length <= max\_steps)

if (current + one step right IS VALID)

permutations.push( current + one step right )

if (current + one step down IS VALID)

permutations.push( current + one step down )

path best

for ( i = 0 to allValidPaths.size() )

if ( allValidPaths[i].total\_cranes > best.total\_cranes )

best = allValidPaths[i]

Time Analysis

The above code runs through two distinct loops. A while loop and a for loop. The while loop runs as long as the queue permutations is not empty. This loop also sets the size of the vector allValidPaths. The for loop depends on the size of the vector allValidPaths. Together the combined time complexity of both loops makes the time complexity of the whole Exhaustive Algorithm Solution.

Exhaustive Algorithm Solution time = While loop + for loop

Where

For loop = O(n) with n = allValidPaths.size()

While loop = number of all permuations of a set of 2 elements with variable size from 1 to max\_steps

Number of permutations with repetition = n^r

Where n = number of elements & r = length of the permutation

Therefore the number of permutations with repetition and variance = The Sum of n^r from r = j to r = k

Therefore the While loop = The Sum of 2^r from r = 1 to r = max\_steps

Meaning the While loop has a time complexity of O(2^n) where n = max\_steps

This gives the entire algorithm a time complexity of O(2^max\_steps) + O(allValidPaths.size)

Or O((2^max\_steps) + allValidPaths.size)

This gives us a worst case of O(2(2^n)) when 2^max\_steps = allValidPaths.size

And a best case of O(2^n) when allValidPaths.size = 0

Graph for time vs. input size

through

Dynamic Algorithm Solution Pseudocode

Given a 2D vector called A of the same dimensions as a 2D grid G

A[0][0] = an empty path on G

Path best = an empty path on G

For (row = 0 to G.rows)

For (column = 0 to G.columns)

If (G[row][column] is a building)

Continue

Path from\_above = an empty path

Path from\_left = an empty path

If ( row != 0 AND G[row – 1][column] is NOT a building )

From\_above = A[row-1][column] + one step down

If ( column != 0 AND G[row][column - 1] is NOT a building )

From\_left = A[row][column - 1] + one step right

If (from\_above OR from\_left are NOT empty paths)

A[row][column] = from\_above or from\_left whichever has more total cranes

If (A[row][column] total cranes > best total cranes )

Best = A[row][column]

Time Analysis

Given that the entire algorithm is nested for loop we get a time complexity of O(n^2) where n = the length of the side of the grid.

The README for this assignment implies that we should get a time complexity of O(n^3). This seems to be because the README suggests that once you fill out the 2D Vector A with all the paths you should loop through that again and find the best path. However, we can instead do this at the same time as filling out the 2D vector A, giving us the reduced time complexity of O(n^2).

Graph for time vs. input size

Through

Questions

1. Is there a noticeable difference in the performance of the two algorithms? Which is faster, and by how much? Does this surprise you?
2. Are your empirical analyses consistent with your mathematical analyses? Justify your answer.
3. Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.
4. Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.