Evan Joseph

January 31, 2017

CIS 350-001, Program 1, Memo

Space analysis O() of every function

**findIndex**: O(1) because we only need space for the local variable. This only compares two ints at a time and returns the index if the two its are the same, or -1 if none of them are.

**nextOneOfEachName**: O(1) because we only need space for the local variables. lastPermute was resized in the main, so we’re not pushing anything back on it. It’s just modifying the contents of it.

**setUpUniqueNames**: O(n), n is locations.size() in this case, because we’re pushing back a number of indices equal to the total number of locations

**checkDistance**: Nothing is being stored here, other than a single variable which is returned. So, O(1)

**main**: we have 6 vectors that we use to push\_back certain values to, so the space required is the sum of the number of elements in locations, permute, uniqueNames, allCombinations, tempIntVector, holdsTheShortestPermutation. O(m+n+o+p+q+r)

Time analysis O() of every function

**findIndex**: O(n), n is uniqueNames.size() in this case; It steps through the for loop up to uniqueNames.size() times.

**nextOneOfEachName**: O(n\*m), which is lastPermute.size() \* uniqueNames.size() in this case; The for loop executes up to lastPermute.size() number of times, and it calls findIndex each time, which has O(uniqueNames.size())

**setUpUniqueNames**: O(n\*m), which is locations.size() \* indexUniqueNames.size() in this case; The nested for loops step through each element in locations, which is the total number of pokemon, and also through every element of indexUniqueNames, which has a size equal to the total number of unique pokemon

**checkDistance**: O(n), which is indices.size() in this case; Steps through each element in the for loop to calculate the sum of the distances

**main**: The worst case for the main occurs when we enter the for loop nested in a do-while loop which is nested in another for loop. O(n\*m\*p), which are allCombinations.size()\* next\_permutation(tempIntVector.begin(), tempIntVector.end()) \* tempIntVector.size() in this case

Test Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Reason for test | Input | Expected output | Actual output |
| This will test having more than one Pokemon for just one certain type of Pokemon. Pokemon that are not unique should not be visited in the same run. | 6  0 1 Eevee  2 2 Jolteon  5 5 Umbreon  9 4 Eevee  1 9 Snorlax  15 12 Eevee | 1 2 3 5 28 | 1 2 3 5 28 |
| This will test having more than one Pokemon for two or more types of Pokemon. | 5  1 0 Pikachu  4 3 Magmar  7 6 Pikachu  9 10 Electabuzz  15 15 Magmar | 1 2 4 38 | 1 2 4 38 |
| This will test having just one of each type of Pokemon. This is important because it should only print out the data once. | 3  4 7 Bulbasaur  5 8 Ivysaur  20 26 Blastoise | 1 2 3 92 | 1 2 3 92 |
| Testing just a single location for a single pokemon, as a boundary case. The program should know to go there and straight back. | 1  5 5 Eevee | 1 20 | 1 20 |