Danial Rizvi
DR28944

Lab 3 Report

Part 1a: Give the pseudocode for an efficient algorithm determining the maximum fun level that can be achieved based on the given input. State the runtime complexity of this algorithm in Big-O notation.

This is similar to the 0/1 Knapsack problem in which the risk level of an activity is the weight of the item, the fun level of the activity is the value of the item, and the maximum risk is the maximum weight that can be carried. The problem can be solved by creating a two-dimensional array where the rows correspond to items and the columns correspond to risk (0 to maximum). By iterating through the array and choosing the activities that maximize the fun value while remaining below the risk level we can keep track of the most optimal activities to select. The element in the last row and last column of the array will be the maximum fun level that can be achieved based on the activities and the maximum risk level. Since the algorithm must iterate through all of the different activities and risk levels the runtime complexity will be a pseudo-polynomial of O(*m**n*) where *m* is the number of activities and *n* is the maximum risk level.

Part 1b: Discuss the changes you made to your algorithm from part (a), and how the runtime complexity may or may not have changed.

In order to decide which activities to select a second two-dimensional array of the same size as the first must keep track of whether the activity is selected or not when maximizing the fun level at the specific index. By keeping track of whether or not to select the item, once we have constructed both arrays we can iteratively loop back through the rows of the second array to determine whether the corresponding item should be selected or not. The run-time of this updated algorithm is O(*m**n*) + O(*m*) which can be simplified to O(*m**n*) from part a.

Part 2: Give the pseudocode for an efficient algorithm determining a schedule that minimizes the total vacation cost based on the given input. State the runtime complexity of this algorithm in Big-O notation.

The approach for solving this problem was similar to that of the previous problem in that a two-dimensional array is used to keep track of the minimum cumulative cost of staying at hotels between the two islands. The two-dimensional array is iterated by columns which represent the day. At each index in the array Fruitcake can arrive there by either transferring islands or remaining at the current island. If he transfers islands the cost of the transfer and hotel is added to the index which is one row above and one column to the left. If he remains at the island the cost of the hotel is added to the index which is in the same row and one column to the left. The minimum of these two is the minimum cumulated cost of that day. Because the algorithm moves diagonally and to the right, all indices below the diagonal are ignored. Two additional arrays of the same size keep track of the previous index of each element and which island is associated with each index. This is shown in the figure below. All of this is done twice, once for assuming Fruitcake starts at the first island and a second time for assuming Fruitcake starts at the second island. Once all of the arrays have been constructed the minimum cost is the minimum of the cost values in the last column of both arrays. Once the index of the minimum cost has been found, the columns of the other two arrays are iterated to select the island at the corresponding day which minimizes the total cost. The run-time complexity of this algorithm is

Danial Rizvi
DR28944

O(2**k***n*^{2}) where $0.5 < k < 0.75$ and *n* is the length of the island hotels array. This simplifies to a pseudo-polynomial of O(*n*^{2}).