Jessica Salguero, UIN: 937005006 **Homework 3** February 21, 2025

Problem Statement

Yuckdonald's is considering opening a series of restaurants along Quaint Valley Highway (QVH). The n possible locations are along a straight line, and the distances of these locations from the start of QVH are, in miles and in increasing order, $m_1, m_2, \cdots, m_n$. The constraints are as follows:

1. At each location, Yuckdonald's may open at most one restaurant. The expected profit from opening a restaurant at location m_i is p_i, where p_i ¿ 0 and i = 1, 2, $\cdots$, n.
2. Any two restaurants should be at least k miles apart.

Give an efficient algorithm to compute the maximum expected total profit subject to the given constraints.

Main Idea

Describe here how you are planning to solve the problem.

We know that the Yuckdonald's locations are going to be in a straight line and in increasing order, and the goal is to maximize profit. The main constraints are that there can only be one restaurant at each location and each restaurant must be at least k distance apart, while trying to maximize profit.

When approaching this problem, it became obvious that there would be a recurring question that needed to be asked at each of the (n) total locations:

1. Should we open a restaurant at this location?
   (a) Yes, open a restaurant at location (m_i)
   (b) No, do not open a restaurant at location (m_i)

Additionally, if we do decide to open a restaurant at this location, we then need to make sure that this new location would satisfy the constraint where restaurants need to be at least k miles apart. Since we're dealing with a distance that can potentially be larger than k, there is going to need to be comparative searches while iterating through the list of potential locations and their respective profits (p_i) to identify the locations that will maximize the overall profit.

Greedy algorithms would not be a good choice for solving this problem because a greedy algorithm would determine the maximum profit for each step while iterating through possible locations, and then assume that these are also the correct choices for ensuring overall maximum profits. However, with dynamic programming, this algorithm would be able to consider all possible arrangements of restaurant locations along QVH to identify the actual best locations for maximizing profits. With this approach in mind, dynamic programming is the best choice for this algorithm.

Pseudocode

Algorithm 1 Maximize Yuckdonald's Profit

```
1: function MAX RESTAURANT PROFIT(void)
2:     Initialize global variables:
         • maxProfit = [0]. Initialize this variable into an array of size n+1 (range is from 0 to n); all values should be zero because no locations is equivalent to zero profit. This will be where the profit at the locations correlated with maximum profit will be stored.
         • possibleLocations = array of size n. Initialize this array to be equal to all possible restaurant locations ($m_1$ to $m_n$).
         • allProfits = array of size n. Initialize this array to be equal to profits corresponding to each restaurant location ($p_1$ to $p_n$).
3:
         • For loop iterating through all possible Yuckdonald's locations: for i in range(2, n), loop through possibleLocations. Start range at 2 because second loop will be accessing a previous item in array.
             – maxProfit[i] = maxProfit[i-1]. We need to compare with the results from the previous iteration
             – Second nested for loop to find suitable location for another restaurant: For j in range(1, i-1):
                 ∗ if (possibleLocations[i] - possibleLocations[j] is greather than or equal to k), check to see if adding location will benefit maxProfit:
                     · if ((allProfits[j] + maxProfit[j]) is greater than or equal to maxProfit[i]):
                     · maxProfit[i] = allProfits[j] + maxProfit[j]
                     · else:
                     · maxProfit[i] remains unchanged
     print("Maximum profit is equal to ", maxProfits[n]). Should be maxProfits[n] because the range of that array is from 0 to n; this is the last item in the array.
4: end function
```

Proof of Correctness

Clearly articulate the key steps and reasoning behind your algorithm's correctness. Prove that your algorithm always finds a correct solution for all possible instances. ***Specific examples may be used to help explain your proof, but they cannot replace the rigorous proof.***

First, to deal with the base case where there may be 0 possible locations, maxProfit is initialized to be of size n+1. If there are zero possible locations, then the maxProfit will default to zero. Additionally, if there is only one possible location, then the algorithm will still be able to conduct the nested for loops and determine the maximum profit being equal to p_1.

Second, the inner loop is integral to ensuring that the correct maxProfit is calculated. If we assume that maxProfit[a] was correctly calculated for locations ranging from 1 to a, and now want to calculate the maxProfit[a+1]. This algorithm will ensure the constraints are being correctly met:

1. First, it will ensure that the k distance constraint has been met. If the location is too close to another restaurant, it will not continue.

2. Second, it will check the other constraint regarding the maximum profits. If a restaurant should be added at that location, then you need to add the current profit $p_(a+1)$ to what the previously recorded maximum profit was. Then, the algorithm will determine the maximum value between maxProfit[a] and maxProfit[a+1] + $p_(a+1)$.

   Since maxProfit[a] had already been previously maximized during previous iterations, then when the algorithm chooses the maxProfit value between maxProfit[a] and maxProfit[a+1] + $p_(a+1)$, it will also be making the optimal decision for this current iteration. As the algorithm continues choosing the optimal choice during each iteration, we know that the final maximum profit will be found at the end of the loop (maxProfit[n]). Thus by induction, if each portion of the algorithm is correct, then the overall algorithm should also been correct.

Time Complexity

State and explain (if needed) the time complexity of your solution.

To calculate the time complexity of this algorithm, we need to take into account that there is a nested for loop. Since the outermost loop iterates through n times, this gives an O(n) for that specific portion of the algorithm. The inner loop will iterate from 1 to i-1, which means it will have a maximum number of iterations of n; the inner loop also has a time complexity of O(n). The overall algorithm will thus have a time complexity of O(n*n) = O(n^2).