Term Project
IE 343

Ceren Gülkokan
ceren.gulkokan@ozu.edu.tr
Ahmet Said DAŞ
said.das@ozu.edu.tr

Department of Industrial Engineering
December 31, 2022

I. Project Introduction

French Lebanese trumpeter Ibrahim Maalouf needs to select his songs for his new album. Further information can be found in the ReadMe file. We have designed two algorithms to solve the given problems. We chose 2 Greedy algorithms in our design because enumeration algorithms would yield to high CPU times even though they are exact. On the other hand, other algorithms such as Dijkstra's algorithm were observed to consume too much time to implement.

II. Solution

For this project, in addition to readValues() and readSequential() functions we have created a function for the first part of the project, named as solveProblem1(). Which takes in valueList and weightList as arguments. Since the first part of the project wants us to determine which songs will be included in Ibrahim Maalouf's next album, solveProblem1(valueList, weightList) function uses greedy algorithm to solve the problem. This method takes two lists, valueList and weightList.

First the algorithm asserts if the size of valueList is equal to weightList. Then, we initialize sorted arrays that stores the sorted list of songs according to their gain by weight. For this reason, we have two arrays and for the inclusion we have another array, adding up to 3 new arrays. These arrays are: sortedGain, sortedIndices and X_i (inclusion) . The sortedGain sorts the gains (popularity/duration) for all songs. Not that this array only stores the gain values, therefore we needed to store the indices of these gains since after the sorting they will all be random to us. Thus, we used sortedIndices. In this array, the indices of the elements updated at every iteration of sorting, i.e. if the 27^{th} element in the original table has the most gain among all songs sortedIndices[0] will be 27. Lastly, the inclusion array is only 0 and 1.

Afterwards, we assigned initial array values. All are either unsorted or 0 for now. To get the songs that have the highest value and minimum duration, the algorithm checks all songs by their value per weight (gain by weight) and includes the songs with the most gain, meaning the maximum one of the ratios and its index number. Meanwhile, the algorithm sorts the gain ratios with a for loop implementing the swapping values technique. Our sorting algorithm is selection sort since a mergeSort or a HeapSort would be difficult to implement with 2 arrays: sortedGains and sortedIndices. After all the gains are sorted, we get our sortedIndices and sortedGain arrays ready for the inclusion.

The inclusion starts with the first element of the sortedGain and iterates until there is no time left. The duration of the selected songs is added under totalTime which is the cumulative total duration of included songs in ms. Using a for loop again if the total duration + the current iteration's song is below 1 800 000 ms (30 minutes) the song can be added in.

To calculate the objective function, we run a for loop and add every song's value if it is included (X_i[i] ==1 holds). Even though the popularity value is integer, it is initialized double to be safe in objective value calculation since the punishment will be double, just to be safe with operating with both. We determine popularity by adding each included songs popularity to variable named "popularity". At the end of the part 1, we print out the songs that are included and calculate the objective function value as popularity - punishment.

To use in the second part of the project, at the beginning of the project we created integer variables that store the index of the most popular song (MPS) and the index of second most popular song (SPS). After calculating the objective function, we assign MPS and SPS as the first two values of the sortedIndices.

In the second part of the project, we need to solve this track list problem with a relevant heuristic. Again, we have chosen a greedy algorithm to solve this problem as greedy algorithm is a heuristic approach to solve the problem. First, we initialized first and last songs meaning the most popular song must be the first song and the second most popular must be the last song. Then, to understand which songs have been included in the track list we have decided to use flag assigning technique (inclusion array) which gives 1 to the ith song if that song is included or 0 if that song is not included.

In the beginning of the algorithm, first we included the first song. For now, we did not include the second most popular song in the playlist, it will be added after every possible song is added and the inclusion is finished.

Then, we initialized the next song's index as jMax which will be selected from the best option of all j's. We created a for loop to select the best song possible From the MPS to SPS until no songs can be added. At every iteration maxValue is initialized as 0 and the loops search for a value to beat the maxValue. For this searching process the loop only checks the songs that are not included or that are not SPS. In order for an e_{ij} value to beat maxValue the duration of the song j should not exceed 30 minutes when added. As long as there is an addable song, every song will provide a value to beat maxValue.

When a state is reached that no more songs can be added algorithm stops. We used to determine the stopping of the algorithm with counters at our very first approach. However, we realised that we did not have to have these unnecessary counters in our computers. Instead, we switched to a simple if statement which, after the maximum enjoy among all possible songs are found if it is 0 the algorithm must terminate. This ensures that from song i, no song j exists such that song j is addable with an $e_{ij} > 0$.

SPS will be included as the last song, we created a condition which the index is not equal to SPS. During satisfaction of this condition, the algorithm scans every song j from song i. It finds the best enjoy value. If the song found, satisfies all conditions, (it is not included yet, its time will not make the album exceed 30 minutes or it is not the SPS) the song is included. After including the best option, we make sure that we add the enjoy value to the objective value and increase total duration of included songs.

After the algorithm finishes processing, we make sure that we include SPS as the last song. Then, punish by the missing time. Lastly, we report the objective function.

III. CPU Benchmark of Algorithms

The average CPU times for our algorithms are given below:

- For problem1 average is 15719469,95 nano seconds.
- For problem 2 average is 15778359,85 nano seconds.

Attempts for taking the average are also provided below (in nanoseconds).

Problem 1 Algorithm	Problem 2 Algorithm
15678001	16731600
14966200	15498801
15803301	17056699
17105100	14982700
16457701	18109900
15058599	13425699
15003999	15851100
14743300	15440800
14861000	14261101
15504100	16757899
14928200	13710900
15264100	16155700
16308900	18066400
15962599	19360799
13554099	13769699
15106200	15962100
15541200	15276200
20009200	13718300
16700999	15268900
15832601	16161900