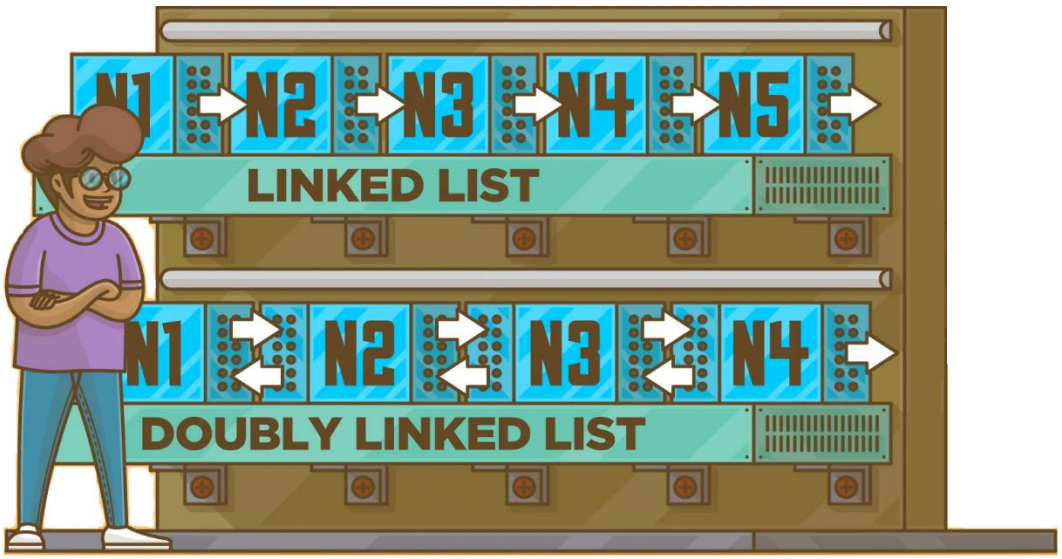
Data Structures And Algorithms

HOMEWORK - 2

Name	ID
İSAK ÇOBAN	1242***0116
OĞUZHAN ERÇELİK	1083***0016
SEVDA ERGÜN	2860***0092



REPORT

THE PURPOSE OF THIS PROJECT:

The purpose of the homework is to create a song program like Spotify. This program has 8 commands. The user can do operations shown below pressing buttons. Moreover, it doesn't matter if the key words are lower or upper case.

C	• *Create a person profile*
S	• *Add a song for a person profile*
E	• *Delete a song for a profile*
L	• *List the songs of the person*
N	• *List all names of registered people*
M	• *List all the songs that liked by anyone*
R	• *Recommend the most popular 3 different songs*
Stop	• *Exit the program*

HOW DID YOU SOLVE IT?

We have 4 classes in this homework. According to commands (***S, C*** , ***L*** etc.) that the user enters, ***Main*** class prints requested information reaching methods in other classes. ***Person*** and ***Song*** class creates node objects for our linked list. These classes have data fields like name, pointers etc. Names came from the user. ***LinkedList*** class determines how these nodes are stored by using its own methods. The commands entered by the user go through some processes in this class. ***create()*** method creates a person node object in the LinkedList. ***searchPerson()*** method helps ***create()*** method. If name that the user entered wasn't entered before, ***create()*** method creates a person with name taken from the user. ***addsong()*** method adds a song to the likedSongs(which is a data field in the Person class) of a person. Person name and song name are entered by the user. ***searchSong()*** method helps ***addsong()*.**

If the name that the user entered wasn't entered before, ***addSong()*** method prints a warning otherwise adds the song. Thanks to ***deletesong()*** method, the user can enter a person and the song that the person doesn't like. Therefore, this song is to be deleted from the list.

printListAsPersonName() method prints all names of registered people. ***printListAsSongName()*** method prints all the songs that liked by anyone. ***printSongsForPerson()*** gets a person name and prints that person's likedSongs list. ***recSongs()*** prints the most liked 3 songs in the whole environment. If the user presses ***Stop*** anytime, the program exits.

create() : First we checked if the given name is registered before and if there is, it prints a warning. If there is no person with the entered name, it create a new person node. Then we find the last node of the LinkedList and add that person to the end of the list. If there is no person in the list then the method add it to head of the list.

addsong() : First we checked if the given song is in the entered person before and if there is, it prints a warning. If there isn't, it creates a new song node and finds the last node of the LinkedList. Then add it to the end. If there is no song in the list then the method add it to head of the list.

deletesong() : First we search for the entered name in the list. If the person couldn't find the method prints a warning. Otherwise, we find the entered song in the entered person. The method assigns previous pointer of the song which is wanted to delete, to the the songs next's pointer.

printListAsPersonName() : If there is no person registered before, it prints a warning. Otherwise, we search linearly in the LinkedList via its pointer and print every node's name(with getPersonName in the Person class).

printListAsSongName() : If there is no son added before in the envorinment, it prints a warning. Otherwise, we search linearly in the LinkedList of every person in the environment and add them in a hashSet. Because a song can be in several people's likedSong list. To get rid of this duplication we used set. After this adding operation is finished, we search linearly in the set and print every element's name (with getSongName in the Song class)

printSongsForPerson() : First we find the entered person in the LinkedList. If there is no such person, it prints a warning. Otherwise we search linearly in the LinkedList (likedSongs data field) via its pointer and print every node's name(with getSongName in the Song class).

recSongs() : To fiqure out how many people has added every single song we have a map. Key is the song's name and value are the occurrence number. Firstly, we add all songs in the environment with duplicates in a ArrayList with searching in both lists linearly and get copies of person's LinkedList's head and that person's likedSong LinkedList's head. After that, we do linear search in the ArrayList. While moving in the ArrayList we check if an element of the list is equal to another element on the list. We count that occurance number and put the song and this number in the map. After putting we change all songs equal to the song that we just put in the map, in the ArrayList with a whitespace(" "). This is why we check if the selected element of the ArrayList is a whitespace. If it is we skip it, if it isn't we do calculation and find occurrence number.

searchSong() : First we copy LikedSong's head. Then we search linearly and if found the entered song is in the list, it returns that Song object.

searchPerson() : First we copy list of Person's head. Then we search linearly and if found the entered person name is in the list, it returns that Person object.

NOTE :

We use ***Intellij*** as IDE to code all classes. Moreover, we use our private repositories in ***Git*** to share our codes with our teammates.

TIME COMPLEXITY:

Methods	***BIG O***
create() :	N
addsong() :	N
deletesong() :	N
printListAsPersonName() :	N
printListAsSongName() :	N^2
printSongsForPerson():	N
recSongs() :	N^2
searchSong() :	N
searchPerson() :	N

Contributions as a percent of each member of a group:

İsak	Oğuzhan	Sevda
33%	***34%***	***33%***