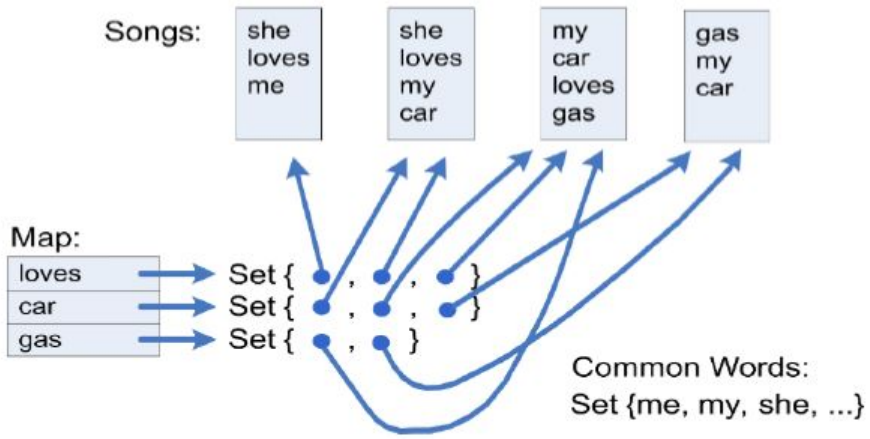
TreeMaps and TreeSets



Objectives: Designing an algorithm and data structure for ranking matches

We have just seen that searching by key words can sometimes give you a lot of matches. We are now going to rank those matches so that we can display the best matches first. This assumes that the user has a certain phrase in mind, such as "she loves you", perhaps from hearing it on the radio, and desiring to find that song. The matched songs must have all words in the search phrase in order, but they might have extra intervening words or punctuation.

Here are a few search results for the phrase "she loves you":
The Beatles: ... She loves you, yeah, yeah, yeah ...
rank = 13
Bonnie Raitt: ... She just loves you ...
rank = 18
Elvis Presley: ... You're asking if she loves me \r \n Well, you don't know the half ...
rank = 23

The rank is the length of the substring from the start of the first word to the end of the last word. The phrase "She loves you" is 13 characters long, so that is the best possible rank for that phrase. There may be many possible matches in a song, and the rank is based on the best match within a song. Phrase matches may span multiple lines such as in the Elvis Presley song above where "\r \n" are the return and newline characters separating two lines of the lyrics. (These characters may differ on different platforms, so you may get a slightly different rank, depending on how you read the song file and how you built the lyrics string.)

Searches are case insensitive. Each word in the search phrase must be matched in order, however any punctuation or spacing is not matched. You are matching whole words so "love" should not match words such as "glove" or "lovelorn". The search phrase may contain common words or even consist entirely of common words. Common words also need to be

matched.

Task 1:

- Create a class PhraseRanking
- Create within it a method:

  `static int rankPhrase(String lyrics, String lyricsPhrase)`

  This takes in 2 strings, the lyrics and the lyrics phrase being looked for. It returns the ranking of the lyrics with respect to a search for lyricsPhrase.
- (optional) You may wish to create a class RankedSong that has 2 properties, the integer rank and a Song object reference. You may do this, but remember to hand it in along with PhraseRanking. It only needs a constructor, a toString() and a compareTo().

Task 2:

- Add a main() method for testing your rankPhrase method. As in the testing methods for your search classes, the first command line argument is the name of the song file, and the second argument is now the lyrics phrase. In NetBeans, to pass in more than one word as a single command line argument you will need to enclose the lyrics phrase in quotes.
- Have it read in the SongCollection, and then rank every song based on the lyricsPhrase.
- Print a list of (ranking, artist and title) for all songs that have valid rankings.
- For the phrase "she loves you" the correct output is:

13 Aerosmith "Beyond Beautiful"
68 Aerosmith "Girls Of Summer"
52 Aerosmith "Walk On Down"
112 Alabama "The Woman He Loves"
147 Alabama "This Love's On Me"
309 Aretha Franklin "A Rose Is Still a Rose"
79 Beatles, The "Devil In Her Heart""
174 Beatles, The "Every Little Thing"
39 Beatles, The "I Got A Woman"
13 Beatles, The "She Loves You"
. . . total of 58 songs

What to turn in:

Written part:

1. Explain your algorithm for calculating the rank of a song.
2. Turn in your ranking results for the phrases:
   - "she loves you" (expect 58 songs)
   - "Time can bring you down" (expect 9 songs)
   - "You can't always get what you want" (expect 5 songs)
3. Any incomplete parts, or any known bugs in your code.

Electronic submission:

- Submit and your write up part9.pdf file.
- PhraseRanking.java (and RankedSong.java if you created one)
- Do not in any way combine, compress, zip, tar or jar your files!

Grading:

Written explanation of your ranking algorithm and ranking results	20 points
Correctly calculating rankings	80 points

I expect your programs to be properly commented and use good style. Points may be deducted for egregious disregard of these matters. Every method in every class must have complete documentation including authorship. A full an complete revision history should be at the top of every class file!