Project 1 Report– Address Book   
with Multiple Field Search

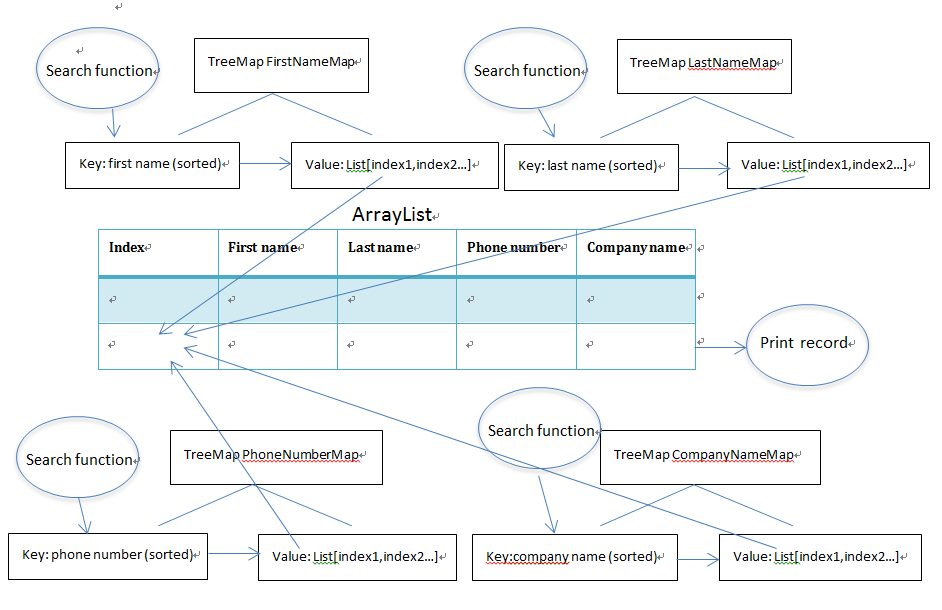


* Introduction:

This project is to implement search functionality on multiple fields of an address book which contains four attribute: first name, last name, phone number and company name.

I used ArrayList to store the original data list and created four TreeMap to store four attributes and their corresponding indexes in original data list. The structure TreeMap can automatically implement sorting function. And then I used subMap function to implement the partial search function using “begin with” syntax.

* Structure:



Class AddressInfo: Create the class to get the data from file.

ArrayList<AddressInfo> AddressList: Store the original data in an ArrayList

TreeMap<String, ArrayList<String>> FirstNameMap: Just Store the first names. Use the first name as key, and it’s corresponding index in AddressList as the value. The index is stored in a list to avoid data missing in case that several records have same first names.

TreeMap<String, ArrayList<String>> LastNameMap

TreeMap<String, ArrayList<String>>PhoneNumberMap

TreeMap<String, ArrayList<String>>CompanyNameMap

* Process:

1. Read file data into an ArrayList.

2. Create four treemap, put attributes and corresponding index list into threemap.

3. Search by different attributes. For each case, use subMap function to get subtreeMap which contains those matched records.

4. From the subtreeMap, get each key’s value, it’s a list of the key’s index in original Arraylist. Then use these index to retrieve whole information from original Arraylist.

* Results:

1. For small AddressBook:

Initialize time: 59ms

|  |  |  |  |
| --- | --- | --- | --- |
| Search Field | Input | Total matches | Search time |
| F | Reb | 3 | 2ms |
| L | Lin | 5 | 0ms |
| P | 202 | 4 | 0 |
| C | B | 33 | 3ms |
| Firstname | Invalid Selection | | |

2. For MediumAddressBook

Initialize time: 213ms

|  |  |  |  |
| --- | --- | --- | --- |
| Search Field | Input | Total matches | Search time |
| F | Reb | 7 | 2ms |
| L | Lin | 9 | 1ms |
| P | 202 | 30 | 2ms |
| C | B | 348 | 16ms |

3. For LargeAddressBook

Initialize time: 814ms

|  |  |  |  |
| --- | --- | --- | --- |
| Search Field | Input | Total matches | Search time |
| F | Reb | 77 | 5ms |
| L | Lin | 112 | 4ms |
| P | 202 | 649 | 18ms |
| C | B | 3553 | 184ms |

4. For HugeAddressBook

Initialize time: 7401ms

|  |  |  |  |
| --- | --- | --- | --- |
| Search Field | Input | Total matches | Search time |
| F | Reb | 549 | 26ms |
| L | Lin | 746 | 17ms |
| P | 202 | 4057 | 92ms |
| C | B | 24998 | 577ms |

5. For SuperHugeAddressBook:

Exception in thread "main" java.lang.OutOfMemoryError: GC overhead limit exceeded

* Analysis and Summary:

1. All the test cases are passed, and as the data scale becomes bigger, the Initialize and search time also increase.

2. Binary Search is implemented by structure TreeMap and subMap function. Sort is implemented by applying TreeMap structure.

3. The data was stored in four TreeMap, so for each search operation, there is no need to sort the data again.

4. The four TreeMap stored different attributes, using index to link with original data list. So there is no duplicate data in multiple sorted lists.