University of Greenwich Vietnam
BSc Computing (Top-up)

Data Structures and Algorithms Assignment 1

Student No: GT00570
Student Name: Dat Dinhquoc
Student Email: datdqgt00570@fpt.edu.vn

May 2017

1. Understanding Data Structures and Algorithms

1.1 Design Specification

a. Product Data Structure

```
Product {
    pcode:    string,
    pro_name: string,
    quantity: integer,
    sale:     integer,
    price:    double
}
```

b. Product Field Descriptions

```
pcode:    Product code
pro_name: Product name
quantity: Available quantity of product in warehouse
sale:     Number of units sold
price:    Price of the product in default currency
```

c. Customer Data Structure

```
Customer {
    ccode:    string,
    cus_name: string,
    phone:    string
}
```

d. Customer Field Descriptions

```
ccode:    Customer code
cus_name: Customer full name
phone:    Customer's phone number
```

e. Order Data Structure

```
Order {
    ccode:    string,
    pcode:    string,
    quantity: integer
}
```

f. Order Filed Descriptions

```
ccode:    Customer code
pcode:    Product code of purchased product
quantity: Number of product units ordered
```

1.2 Performance of Searching and Sorting Algorithms

a. Sequential Search

Mechanism:
Loop through all entries in array/list and match 1 by 1

Complexity (N is the number of entries):
O(N)

Pros:
Simple, no ordering maintenance needed

Contros:
Slow

b. Binary/Radix Search

Mechanism:
Binary/radix search does searching on a key, from the key, other properties of object can be retrieved. Given an array/list sorted on a key, divide the key list into sections (depending on the radix), check the searching key to see in which section it falls into. Recursively divide sections into subsections until the searching key is in a section with a single entry. Take the other properties.

Complexity (N is the number of entries, log here is the logarithm of the radix):
O(logN)

Pros:
Very fast searching, number of entries increases linearly doesn't matter.

Contros:
A bit complex, maintaining the ordering of key values is required.

c. Bubble Sort

Bubble sort is the most basic sorting algorithm everybody knows from first.

Mechanism:
Find all possible pairs, swap 2 values in a pair if they are not in correct order.

Complexity:
O(N^2)

Pros:
Simple

Contros:
Slow

d. Quick Sort

Quicksort is the most common high performance sorting algorithm which is implemented everywhere in every programming language.

Mechanism:
Get a pivot value, put those values less than pivot value to the left, put those values greater than pivot value to the right. Apply the same mechanism to left partition and right partition recursively until the a sub-list can't be divided into left and right partitions any more.

Complexity:
O(N*logN)

Pros:
Fast in multiple situations

Contros:
A bit more complex compared to bubble sort.

1.3 Operations of Recursive Algorithms

Recursive algorithms apply the idea that the same operation can be applied to subsets of data, and the results from operations on subset data can be used to make the result for the original data set. Directory listing is an example for recursion: get the directory list of root directory, with each child directory do the same thing to get the grand-child directories, and so on.

Many problems can be solved using recursion because most of all problems have a tree of steps to final solution. The number of nodes in recursion tree is expanded exponentially, so brute-force recursion is usually not recommended. Pure recursion can be applied to small problems only. For big problems, at least solution tree branch elimination should be applied along. When exact answer is not needed, heuristic methods can be put into recursion to give approximate answers.

2. Implementation of Data Structures and Algorithms

My Java code implementation for DSA Assignment 1 can be found at my Git repository on GitHub at (default branch '**working**'):
https://github.com/johnlowvale/sms

The assignment is implemented in Java SE with AngularJS for UI. Please run it as a normal Maven project and point browser to http://localhost to start using it.

2.1 Implemented Features

Product list:
1.1. Load data from file
1.2. Input & add new item
1.3. Display data
1.4. Save product list to file
1.5. Search by pcode
1.6. Delete by pcode
1.7. Sort by pcode
1.8. Delete the node after the node having code = xCode

Customer list:
2.1. Load data from file
2.2. Input & add new item
2.3. Display data
2.4. Save customer list to file
2.5. Search by ccode
2.6. Delete by ccode

Order list:
3.1. Input data
3.2. Display data with total value
3.3. Sort by pcode and ccode

2.2 Error Handling in the Implementation

Input error handling:
- Only proper .json file for products, customers, orders can be loaded.
- Empty and number value checking when entering data for new product, new customer.

Some other error handling are also implemented in code.

2.3 Testing Process

The code has been tested with 100 row data files and 10,000 row data files provided by lecturer. Some test cases are listed below.

Case 1.
Input: Load a bad data file
Error message is shown

Case 2.
Input: Enter values to create new product
Result message: Product added

Case 3.
Input: Click product display data button
Result: A list of products is shown on the web page

Case 4.
Input: Click on product save button
Result: A pop-up is shown to ask where to save the file

Case 5.
Input: Search product with code 'dd'
Result: 6 rows should be shown when using 100 row data file

Case 6.
Input: Create a product with code 'asdf', and click remove link
Result: Successful message should be shown

Case 7.
Input: Click sort product button
Result: A list of products sorted by product code should be shown.

Many other cases have been tested on features on customer list, order list.
Unit tests are not included in the Java code of the assignment.

3. Notes

Git repository of this assignment:
https://github.com/johnlowvale/sms

Development environment:
Eclipse Neon for Java Core with Maven

Web application access point:
http://localhost

This document:
https://github.com/johnlowvale/sms/docs/dsa-assignemnt1-datdq.pdf

//end of file