CS 6301.502. Implementation of advanced data structures and algorithms

Fall 2017

Short Project 2: Lists, stacks and queues

Wed, Aug 30, 2017

Version 1.0: Initial description (Wed, Aug 30).

Due: 11:59 PM, Sun, Sep 10.

Submission procedure:

Create a folder named cs6301 (no spaces) and inside that a folder whose

name is your group number (e.g., g00). Place all files you are submitting

in that folder. There is no need to submit binary files created by your

IDE (such as class files). Make sure there is a "readme" file explaining

the contents of the files being submitted. Zip the contents into a single

zip or rar file. If the zip file is bigger than 1 MB, you have included

unnecessary files. Delete them and create the zip file again. If you

unzip, it should start with creation of the cs6301 folder.

Upload the zip or rar file on elearning. Submission can be revised before

the deadline. Only the final submission before the deadline will be graded.

Only one member of each group needs to submit project. Include the names

of all team members in ALL files.

Solve as many problems as you wish. Maximum score: 50

1. [30 points]

Given two linked lists implementing sorted sets, write functions for

union, intersection, and set difference of the sets.

public static<T extends Comparable<? super T>>

void intersect(List<T> l1, List<T> l2, List<T> outList) {

// Return elements common to l1 and l2, in sorted order.

// outList is an empty list created by the calling

// program and passed as a parameter.

// Function should be efficient whether the List is

// implemented using ArrayList or LinkedList.

// Do not use HashSet/Map or TreeSet/Map or other complex

// data structures.

}

public static<T extends Comparable<? super T>>

void union(List<T> l1, List<T> l2, List<T> outList) {

// Return the union of l1 and l2, in sorted order.

// Output is a set, so it should have no duplicates.

}

public static<T extends Comparable<? super T>>

void difference(List<T> l1, List<T> l2, List<T> outList) {

// Return l1 - l2 (i.e, items in l1 that are not in l2), in sorted order.

// Output is a set, so it should have no duplicates.

}

2. [30 points]

Write the Merge sort algorithm that works on linked lists. This will

be a member function of a linked list class, so that it can work with

the internal details of the class. The function should use only

O(log n) extra space (mainly for recursion), and not make copies of

elements unnecessarily. You can start from the SinglyLinkedList class

provided or create your own.

static<T extends Comparable<? super T>> void mergeSort(SortableList<T> list) { ... }

Here is a skeleton of SortableList.java:

public class SortableList<T extends Comparable<? super T>> extends SinglyLinkedList<T> {

void merge(SortableList<T> otherList) { // Merge this list with other list

}

void mergeSort() { Sort this list

}

public static<T extends Comparable<? super T>> void mergeSort(SortableList<T> list) {

list.mergeSort();

}

}

3. [20 points]

Extend the "unzip" algorithm discussed in class to "multiUnzip" method

in the SinglyLinkedList class:

void multiUnzip(int k) {

// Rearrange elements of a singly linked list by chaining

// together elements that are k apart. k=2 is the unzip

// function discussed in class. If the list has elements

// 1..10 in order, after multiUnzip(3), the elements will be

// rearranged as: 1 4 7 10 2 5 8 3 6 9. Instead if we call

// multiUnzip(4), the list 1..10 will become 1 5 9 2 6 10 3 7 4 8.

}

4. [20 points]

Write recursive and nonrecursive functions for the following tasks:

(i) reverse the order of elements of the SinglyLinkedList class,

(ii) print the elements of the SinglyLinkedList class, in reverse order.

Write the code and annotate it with proper loop invariants.

Running time: O(n).

5. [30 points]

Implement array-based, bounded-sized queues, that support the following

operations: offer, poll, peek, isEmpty (same behavior as in Java's Queue

interface). In addition, implement the method resize(), which doubles

the queue size if the queue is mostly full (over 90%, say), or halves it

if the queue is mostly empty (less then 25% occupied, say). Let the

queue have a minimum size of 16, at all times.

6. [20 points]

Implement array-based, bounded-sized stacks. Array size is specified

in the constructor and is fixed. When the stack gets full, push()

operation should throw an exception.

7. [20 points]

Write Merge sort algorithm without using recursion by maintaining your

own stack and simulating how the compiler implements function calls.

8. [30 points]

Implement the Shunting Yard algorithm:

https://en.wikipedia.org/wiki/Shunting-yard\_algorithm

for parsing arithmetic expressions using the following precedence rules

(highest to the lowest).

\* Parenthesized expressions (...)

\* Unary operator: factorial (!)

\* Exponentiation (^), right associative.

\* Product (\*), division (/). These operators are left associative.

\* Sum (+), and difference (-). These operators are left associative.

Output the equivalent expression in postfix.

9. [30 points]

Implement arithmetic with sparse polynomials, implementing the

following operations: addition, multiplication, evaluation.

Terms of the polynomial should be stored in a linked list, ordered by

the exponent field. Implement multiplication without using HashMaps.