Experimental Guide for Algorithms and Data Structures

Zhou Yongxia

College of Information Technology

China Jiliang University

2020-09-05

目录

Experiment 1. Algorithm Analysis

1. Write functions to sum N integers.
   a) Use iterative method
   b) Use recursive method
   c) Run program (a) for N=1000, 10000,100000; program (b) for N = 1000, 10000,100000,1000000.
   d) When will the exception java.lang.StackOverflowError occur of program (b)?
   e) Compare the actual running times.

2. Write functions to calculate factorial of N integers.
   a) Use iterative method
   b) Use recursive method
   c) Run program (a) for N=1000, 10000,100000; program (b) for N = 1000, 10000,100000,1000000.
   d) When will the exception java.lang.StackOverflowError occur of program (b)?
   e) Compare the actual running times.

3. Write functions to calculate the N-th Fibonacci number.
   a) Use iterative method
   b) Use recursive method
   c) Run program (a) for N=1000, 10000,100000; program (b) for N = 1000, 10000,100000,1000000.
   d) When will the exception java.lang.StackOverflowError occur of program (b)?
   e) Compare the actual running times.

4. Suppose you need to generate a random permutation of the first N integers. For example, {4,3,1,5,2} and {3,1,4,2,5} are legal permutations, but {5,4,1,2,1 is not, because one number （1） is duplicated and another (3) is missing. This routine is often used in simulation of algorithms. We assume the existence of a random number generator, **r**, with method **randInt(i,j)**, that generates integers between i and j with equal probability. Here are three algorithms:
   a) Fill the array a from a[0] to a[n-1] as follows: To fill a[i], generate random numbers until you get one that is not already in a[0], a[1],....., a[i-1].
   b) Same as algorithm (a), but keep an extra array called the **used array**. When a random number, **ran**, is first put in the array a, set **used[ran] = true**. This means that when filling a[i] with a random number, you can test in one step to see whether the random number has been used, instead of the (possibly) i steps in the first algorithm.
   c) Fill the array such that a[i] = i + 1. Then

```
for( i = 1; i<n; i++)
    swapReferences( a[i], a[ randInt(0,i) ] );
```

   d) Complete the following tasks.
      i. Write the time complexity of each algorithm.

ii. Write (separate) programs to execute each algorithm 10 times, to get a good average. Run program (a) for N=1000, 5000, 10000; program (b) for N=10000, 50000, 100000, 500000, 1000000; program (c) for N=1000000, 5000000, 10000000.

iii. Compare your analysis with the actual running times.

iv. What is the worst-case running time of each algorithm?

Experiment 2. Lists

1. You are given a list, L, and another list, P, containing integers sorted in ascending order. The operation **printLots**(L,P) will print the elements in L that are in positions specified by P. For instance, if P = 1,3,4,6, the elements in positions 1,3,4, and 6 in L are printed. Write the procedure **printLots**(L,P). **LinkedList** and **ArrayList** should be used. What are the time complexities of your procedures?
2. Given two sorted lists, L1 and L2, write a procedure to compute L1∩ L2 using only the basic list operations. What is the time complexity of your procedure? Compare the running times of your implementation with **retainAll**. **LinkedList** and **ArrayList** should be used.
3. Given two sorted lists, L1 and L2, write a procedure to compute L1∪ L2 using only the basic list operations.What is the time complexity of your procedure? Compare the running times of your implementation with **removeAll** + **addAll**. **LinkedList** and **ArrayList** should be used.
4. Compute the maximum, minimum and average value of a list. **LinkedList** and **ArrayList** should be used.
5. Write a program to print out the elements of a singly linked list.
6. Write a nonrecursive procedure to reverse a singly linked list in O(N) time.
7. Swap two adjacent elements by adjusting only the links(and not the data) using:
   a) Singly linked lists.
   b) Doubly linked lists.
8. Assume that a singly linked list is implemented with a header node, but no tail node, and that it maintains only a reference to the header node. Write a class that includes methods to:
   a) Return the size of the linked list
   b) Print the linked list
   c) Test if a value x is contained in the linked list
   d) Add a value x if it is not already contained in the linked list
   e) Remove a value x if it is contained in the linked list
9. Write a program with proper data structures to add and multiply(optional) two polynomials. The polynomial has the following form:

$$P(x) = a_1 x^{e1} + \cdots + a_n x^{en}$$

Where a_i is the coefficient, e_i is the exponent and e_i is nonnegative integer. Test your program using two example polynomials: $P_1(x) = 10x^{1000}+5x^{14}+1$ and $P_2(x) = 3x^{1990} - 2x^{1492}+11x+5$. And more test examples should be used.

Other Requirements:

1) A report should be written according to the template as the attachment, and should be converted to PDF format before submitted.

Experiment 3. Stack and Queue

1. Implement a **stack** class using a linked list and write a program to test it.
2. Write routines to implement **queues** using the following data structures and test your queues.
   a) A linked list
   b) An array
   c) A circular array

   Requirement:
   a) use java.util.Queue to test the same operation sequence
3. Write a program to get a file's content, and check the content for the following balancing symbols:
   (), [], {}, /**/

   Requirement:
   a) use customized class **MyStack**.
   b) use java.util.**Stack**
   c) In the test file, there must be English letters and symbols.

4. Write a program to evaluate an infix expression.
5. Design and develop a calculator. The input should be an infix expression, then convert it to a postfix notaion, and evaluate the postfix expression. Operations such as addition, subtraction, multiplication, division, parenthesis etc. should be realized.
6. Write a program to output the result of these operations on a stack and a queue, respectively using java.util.Stack and java.util.Queue: insert integer sequence {1,2,3 4} to a queue, then delete two integers and insert integer 5.

Experiment 4. Binary Trees

1. Insert 3,1,4,6,9,7,5,2 into an initially empty binary search tree,
   a) Show the process of the insertions (You can use Microsoft Word, Visio, WPS or other software firstly, and finally you must convert it to a pdf file)
   b) Use the class of BinarySearchTree in the textbook,
      i. Write a method to calculate the number of leaves.
      ii. Write methods of preorder, inorder and postorder traversal.
      iii. Write a method to calculate the height of a binary search tree.
      iv. Write a method to calculate the number of nodes.
   c) In the main function,
      i. construct a binary search tree
      ii. call the above methods and print results of these callings.

2. Insert 3, 1,4,6,9,7,5,2 into an initially empty AVL tree.
   a) Show the process of the insertions (You can use Microsoft Word, Visio, WPS or other softwares firstly, and finally you must convert it to a pdf file)
   b) Design a class of AVL tree by using the class of avlNode in the textbook,
      i. Write a program to calculate the number of leaves.
      ii. Write methods of preorder, inorder and postorder traversal.
      iii. Write a method to calculate the height of a binary search tree.
      iv. Write a method to calculate the number of nodes.
   c) In the main function,
      i. construct a AVL tree
      ii. call the above methods and print results of these callings.

3. Count each word in a English file.
   a) Create a file and copy an English article to the file.
   b) Write a program to count each word in the file
   c) TreeMap must be used.

4. *Design and develop a text editor with the following functions:
   a) Open a text file and display its content on the GUI of your Java application.
   b) Modify the content of the text file (deletion and insertion).
   c) Search a word in the whole text content and highlight founded words
   d) Replace a word by a specified string.
   e) Save file's content to a file.

Experiment 5. Hashing

1. Given input {4371, 1223, 6173, 4199, 4344, 9679, 1989} and a hash function
   h(x) = x % 11 ,collision should be resolved by the following solutions
   a) Draw the results with the following different hash tables, calculate the average search lengths in successful searches and load factors.
   b) Write programs to implement the following different hash tables and test it with the above data sequence;
      i. Separate chaining hash table, and the class SeparateChainingHashTable in the textbook must be used.
      ii. Open addressing hash table using linear probing, f(i)=i, and the class QuadraticProbingHashTable in the textbook must be modified and used.
      iii. Open addressing hash table using quadratic probing, f(i) = f(i-1) + 2i – 1, and the class QuadraticProbingHashTable in the textbook must be used.
      iv. Open addressing hash table with double hashing, h2(x) = 7 – (X % 7), f(i)=i*h2(x), and the class QuadraticProbingHashTable in the textbook must be modified and used.
      v. Open addressing hash table using rehashing.
   b) Put the same data to an AVL tree and a list, draw the final tree and list, calculate the average search lengths in successful searches, sort the time complexities of searching on Hash table, AVL tree and List.
2. Add some same data to HashMap, HashSet, TreeMap and TreeSet objects, then delete 1~2 data and print all data in those objects. Try to understand these two data structures: binary trees and hash tables

Experiment 6. Search

1. Write a program to generate N random integers (N = 10000, 50000, 100000,500000,1000000,5000000,10000000,) , save them to appropriate data structures, then search a random integer in the data structures by the following algorithms, and record the times used by these algorithms in table 1, draw the curves of running times with N (the axis X is the values of N, the axis Y is the values of the running times) by Microsoft Excel or WPS sheet.
   a) The Sequential search
   b) The binary search
   c) AVL tree
   d) The hash table

Table 1 Performances of search

N	10000	……	10000000
sequential search			
binary search			
AVL tree			
hash table			

Experiment 7. Sorting

1. Compare sorting performances of different algorithms:

Write programs to sort N random integers (N = 100, 1000, 5000, 10000, 50000, 100000,1000000) using selection, bubble, insertion, Shell, heap, merge, quick, bucket and radix algorithms. And record the times used by these algorithms in table 1, draw the curves of times with N (the X axis is the values of N, the Y axis is the values of the running times) by Microsoft Excel or WPS.

You need ensure that different algorithms have the same input data sequence. For example, for N=100, all the sorting algorithms should sort the same 100 integers. And the running time excludes the process of generating N random integers. What's more, please introduce your run environment, including CPU, RAM, and Operation system.

You need submit the source code files, an Excel or WPS file which includes table 1, curves and the run environment.

Table 1 Performances of sorting

N	100	……	1000000
Selection sort			
……			
Bucket sort			

Experiment 8. Graph

1. Using the graph as figure 7.1,
   a) Write out the results of topological sorting
   b) Write a program to perform a topological sort according to the pseudocode showed in Figure 9.7 in the textbook.

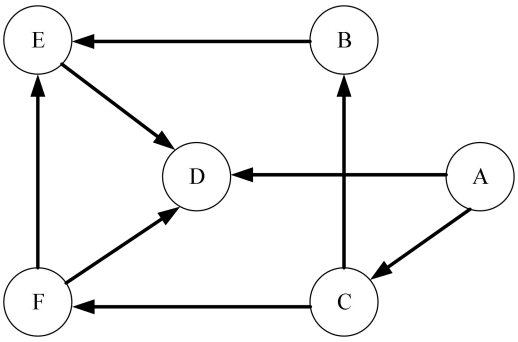

Figure 7.1 a directed graph

2. Write a program to implement Dijkstra's algorithm to solve the single-source shortest path problem(the source is A) of the graph in Figure 7.2, and write out the results.

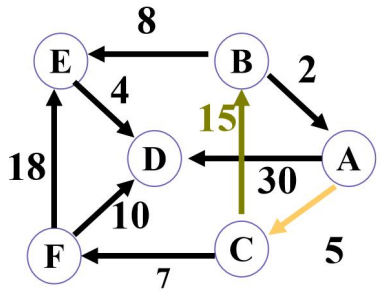

Figure 7.2 a directed weighted graph

3. Write a program to implement Prim's algorithm to get a minimum spanning tree from the graph in Figure 7.3, and draw the process.

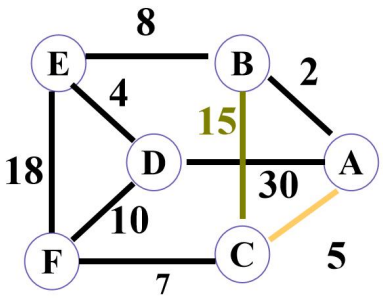

Figure 7.3 an undirected weighted graph

4. Write a program to implement Kruskal's algorithm to get a minimum spanning tree from the graph in Figure 7.3, and and draw the process.
5. Write a program to implement depth-first search from vertex A in the graph of Figure 7.3, and write out the result.
6. Write a program to implement breadth-first search from vertex A in the graph of Figure 7.3, and write out the result.