/*
Jason Wilson
Algorithm Analysis and Design
Professor Bowu
5/14/2020
*/

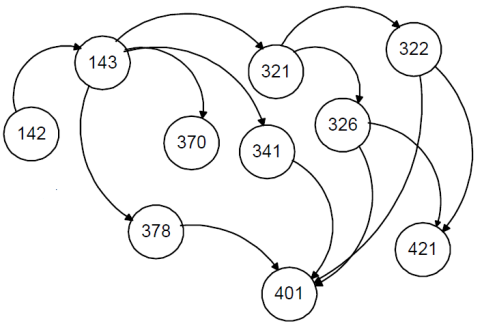
Final Assignment

Important: The final exam is completed by this assignment, and it is worth 5% of your final grade. Please read turn-in checklist at the end of this document before you start doing exercises.

Section 1: Pen-and-paper Exercises

1. Consider the following problem.

Assume you have to take n courses, which are labeled from 0 to n - 1. Some courses have prerequisites, for example to take course 143 you have to first take course 142, which can be expressed as a pair: [142, 143]



Given the courses and a list of prerequisite pairs, is it possible for you to finish all courses?

For example:
2, [0,1]
There are two courses to take. To take course 1 you should have finished course 0. So it is possible to finish all courses.

2, [0,1],[1,0]
There are two courses to take. To take course 1 you should have finished course 0, however, to take course 0 you should have finished course 1 -- it is impossible.

Outline an algorithm to solve this problem.

(i) **describe the idea behind your algorithm in English (2 points);**

Since the pairs are stored in a list, search through the list on each pair to make sure there is no pair where [x1,x2] -> [x2,x1]. Use a nested for loop with I in 0 to n - 1 and j in i + 1 to n

(ii) provide pseudocode (5 points);

INPUT : n , A[][]

```
for i in 0, n - 1

    x0 = A[i][0]
    x1 = A[i][1]

    for j in i + 1, n

        if A[j][0] == x1 && A[j][1] == x0

            OUTPUT : False

        end if

    end for

End for

OUTPUT : True
```

(iii) analyze its running time (3 points).

Time complexity is n log n because the algorithm will run through the given array n times plus the decreasing iteration with the for j in i + 1

N = 3

I = 0
J = 1

I = 0
J = 2

I = 0
J = 3

I = 1
j = 2

I = 1
J = 3

I = 2
J = 3

End
Therefor:

O (n log n)

2. Consider the following problem about student grade of some test (e.g., SAT) over a long time.

   (a) insert(***date, score***): insert into the data structure that a student has received score ***score*** at date ***date***. For example, assume a student took the test on January 16, 2020, and received 88, the following four operations should be performed: insert(20200116,88).

   (b) average(***date1, date2***): return the average of all scores that are received between date1 and date2 (inclusive).

   (c) delete(***date1, date2***): delete scores received between date1 and date2 (inclusive).

   The time per each operation should be at most O(log n), where n is the total number of scores. Note that if k scores were received between date1 and date2, delete(date1, date2) should take at most O(k log n) time.

   Select a data structure that supports the above operations.
   (i) **data structure (2 points).**

   Binary search tree

   (ii) justification for your choice: analyze the time complexity of insert, average, and delete on the data structure selected (5 points).

   Because a binary search tree's time complexity depends on the height of a tree, if a tree is balanced then by definition, their search, delete and average functions will be O(logn).

   Insert is dependent on the height of the tree being O(log n), then when instering, it traverses to the bottom and inserts a node.
   O(log n)

   Average would be O (log n) because search is defined as O(log n). In the worst case, the algorithm will have to search to the bottom of the tree which will take O(log n) time to search for both elements to average. Averaging a range of elements would take O(k log n) time because the iterating through the BST to only find those k elements would O(k) time.

   Delete would be O(log n) because in its worst case, the algorithm will traverse to the bottom of the tree O(log n) and remove and re balance if needed. As far as deleting a range of elements goes, the deletion for each element would take O(log n) time, but the gathering of the elements within that range would take k time, making it O (k log n) time.

   Full credit (7 points) will be awarded for a data structure that supports O(logn) insert, average, and delete. Data structures slower will be scored out of 2 points.

3. Optimal storage problem:

Input:
We are given n articles that are to be stored on a computer tape of length L.
Each article i has a length of Li, i.e. article 1's length is L1, and article 2's length is L2.
L >= (L1+L2+…+Ln), in other words, the computer tape has enough capacity to store all n articles.

Output:
A permutation of the n articles so that when they are stored on tape in this order, the MRT (mean retrieval time) is minimized assuming that all articles are retrieved equally often.

Suppose the articles are stored in the order (o1, o2, … on).

Then the time needed to retrieve article oj is : Lo1 + Lo2 + …+ Loj. This is because to retrieve article oj, we need to retrieve every article before oj on the tape.

Then the mean retrieval time is
MRT =

{(Lo1) + (Lo1 + Lo2) + (Lo1 + Lo2 + Lo3) + (Lo1 + Lo2 + Lo3 + Lo4) + … + (Lo1 + …+ Lon)}/n

Example:

Let n = 3, (L1, L2, L3) = (8, 12, 2), L = 50.

All the possible permutations and their respective retrieval time are given below:

Permutation	**Mean Retrieval Time**
1, 2, 3	{8 + (8+12) + (8+12+2)}/3 = 16.67
1, 3, 2	{8 + (8 + 2) + (8 + 2 + 12)}/3 = 13.33
2, 1, 3	{12 + (12 + 8) +(12 + 8 + 2)}/3 = 18
2, 3, 1	{12 + (12 +2) + (12 + 2 + 8)}/3 = 16
3, 1, 2	**{2 + (2 + 8) + (2 + 8+ 12)}/3 = 11.33**
3, 2, 1	{2 + (2 +12) + (2 + 12 + 8)}/3 = 12.67

The optimal permutation is 3, 1, 2, as it has the minimum mean retrieval time. Our goal is to find a permutation of n articles that can minimize the mean retrieval time **(10 points).**

(a) Consider the following greedy algorithm. Sort articles by their length in ascending order, such that Lo1<=Lo2<=Lo3…<=Lon. Store articles on the computer tape in this order. Prove or disprove that this algorithm is correct.

This algorithm is correct because it will always put the smallest number first and all preceding larger numbers in order after it. This way, when the calculation is preformed for the time taken, the number that is most added will be Lo1 which is the smallest number, adding the least amount on each iteration.

Ex. Lo1 = 2 Lo2 = 8 Lo3 = 12 stored in that order

The optimal solution: {2 + (2 + 8) + (2 + 8+ 12)}/3 = 11.33 is achieved

(b) Consider the following greedy algorithm. Consider the following greedy algorithm. Sort articles by their length in descending order, such that Lo1>=Lo2>=Lo3…>=Lon. Store articles on the computer tape in this order. Prove or disprove that this algorithm is correct.

This algorithms is incorrect because it will be adding on the largest number each iteration resulting in the worst case storage solution.

Lo1 = 12 Lo2 = 8 Lo3 = 2

Resulting in : {12 + (12 + 8) +(12 + 8 + 2)}/3 = 18

12 should not be first but this algorithm will place it in the beginning of the disk making the largest item also need to be searched for. The worst case.

HINT: One of the above greedy algorithms is correct and the other is not.

Section 2: Java Implementation

4. Implement Problem 2 in Java.
   Note:
   In your program, you should
   a) use the data structure you selected in Problem 2 to perform insert, average, and delete operations on student test scores.

      Note: If you use BST/Graph/Linked List, data structures discussed in this class, feel free to use BST.java (Assignment 7)/Graph.java (Assignment 8)/LinkList.java (Assignment 6) provided in previous assignments, and modify the program to solve this problem.

   b) create test cases, and display test results.

      Note: You need to create test cases for all insert, average, and delete operations.

Full credit (30 points) will be awarded for an implementation of a data structure that supports O(logn) insert, average, and delete. Programs slower will be scored out of 10 points.

TURN-IN CHECKLIST:

1. **Answers to Section 1 (.doc/.txt), and to Section 2 (all your source Code (.java files)). Remember to include your name, the date, and the course number in comments near the beginning of your code/report.**

2. **Create a folder and name it 'FirstName_LastName_Final'. In the newly created folder copy and paste your files (.doc/.txt/.java files). Then compress the folder, and submit to iLearn.**