B505: Applied algorithms

HW4 (Due: **Apr. 2 Monday**)

http://darwin.informatics.indiana.edu/col/courses/B505-18

(This HW contains 400 points for midterm makeup if you answer all questions correctly.)

1. (20 pts) Given an undirected graph G and a particular edge e in it, devise a linear-time algorithm O(m+n) that determines whether G has a cycle containing e.

2. (20 pts) Consider the following statement: if (u,v) is an edge in an undirected graph, and during the depth-first search post(u) < post(v), then v must be an ancestor of u in the DFS tree. Is this statement true or false? If it is true, prove it; otherwise, give a counterexample.

3. (20 pts) The police department in the city of Axeville has made all streets one-way. The mayor contends that there is still a way to drive legally from any intersection in the city to any other intersection, but the opposition is not convinced. (1) Formulate a graph algorithmic problem to check whether this statement is true or not, and devise a linear time algorithm to solve it; (2) Suppose it turns out the mayor's claim is false. She next claims something weaker: if you start driving from the town hall, then no matter where you reach, there is always a way to drive legally back to the town hall. Formulate this weaker statement, and devise a linear time algorithm to check if it is true or not.

4. (20 pts) Devise a linear O(m+n) time to solve the following problem: given a directed acyclic graph G (with n vertices and m edges), check if G has a directed path that visits every vertex once and only once.

5. (20 pts) A feedback edge set of an undirected graph G=(V, E) is a subset of edges E'⊆E that intersects every cycle of the graph. Thus, removing the edges in E' will render the graph acyclic. Give an efficient algorithm for the following problem:
   Input: Undirected graph G=(V, E) with positive edge weights w_e
   Output: A feedback edge set E'⊆E of minimum total weight $\Sigma_{e \in E'} w_e$

Make up questions (40 pts * 10 = 400 pts; accounting for 20 marks for the final grade):

6. Given two arrays of *n* integers, the *all-pair-sum* S is defined as the sum of every pair of elements: $S = \Sigma_{i,j} a_i b_j$, where a_i and b_j are the integers in the two respective arrays. Given an array of n integers A, we want to find an array of integers B, in which each element $b_j \in \{1, -1\}$, such that the *all-pair-sum* between A and B is maximized. How to find array B?

7. Given two <u>sorted</u> arrays, each containing n integers, A[1..n] and B[1..n], and an integer N, we want to find two numbers a and b in each of these input arrays, respectively, $a \in A$ and $b \in B$, such that $|a-b|=N$; if there are no such two numbers, a message "not found" should be output. Here, we want to avoid any additional storage with the size O(n). Devise an algorithm to find the two numbers in O(n) time under this constraint. (Note that you can still use additional constant memory as temporary storage.)

8. Given an array of n positive integers and an integer N, we want to find if it has a consecutive subarray (i.e., a subarray with consecutive elements between a two positions) with the sum of N. Devise an O(n) algorithm to solve this problem.

9. Given k arrays each with n integers, devise a divide-and-conquer algorithm in O(nklog(nk)) time to find all integers that each appears at least once in each input array. You should use only comparison of the integers, but not use advanced data structures such as hash tables or counting arrays.

10. In an array of n integers A[1..n], the *increasing* subsequence is a subsequence of k consecutive elements in the array, A[i], A[i+1], …, A[i+k], such that $A[i] \leq A[i+1] \leq \ldots \leq A[i+k]$. Devise a linear time O(n) algorithm to find the longest *increasing* subsequence of a given array of n integers.

11. The *overlap* between two intervals i and i' is defined as: if $i \cap i' \neq \varnothing$, that is, if $low[i] \leq low[i']$, overlap = max(0, high[i] - low[i']); if $low[i'] \leq low[i]$, overlap = max(0, high[i'] - low[i]). Given a set of intervals S and a query interval q, we want to find the interval in S with the greatest overlap with q. Devise a data structure based on binary search tree (BST) to maintain the interval set S (so that they can be inserte/delected, etc), and a search algorithm using the data structure to solve the above problem in O(log n), where n = |S|. You may modify the BST data structure to incorporate additional auxiliary information.

12. Given n integers, and a query integer k, we want to computer how many integers among the n integers are greater than k. Assuming we maintain a binary search tree (BST) of the n integers (so that they can be inserted/deleted, etc), we want to solve the above problem in O(h) time, where h is the height of the BST, and for a balanced BST, h=O(log n). Devise an algorithm in O(h) based on the BST of the n integers. You may modify the BST data structure to incorporate additional

auxiliary information.

13. Given two set of elements, A with m elements and B with n elements ($n \geq m$), devise an algorithm to check if A is a subset of B. Note that you can only compare the elements to tell if they are the same or not. What is the run time of your algorithm in big-O notation?

14. You want to schedule a subset of *n* given jobs on a resource. Each job i needs to run on the resource for t_i hours, and has a benefit of b_i. You cannot schedule more than one job on the resource at a time. Devise an algorithm to schedule the subset of the jobs on the resource for a total of N hours (N is given in addition to t_i and b_i).

15. Alice wants to organize a party and is deciding whom to call. She has *n* people to choose from, and she has made up a list of which pairs of these people know each other. She wants to pick as many people as possible, subject to two constraints: at the party, each person should have at least five other people whom they know and five people whom they don't know. Given as input the list of *n* people and the list of pairs who know each other, devise an algorithm to output the best choice of party invitees.

16. Consider the following variation on the Scheduling Problem. You have a processor that can operate 24 hours a day, every day. People submit requests to run daily jobs on the processor. Each such job comes with a start time and an end time; if the job is accepted to run on the processor, it must run continuously, every day, for the period between its start and end times. Given a list of *n* such jobs, your goal is to accept as many jobs as possible (regardless of their length), subject to the constraint that the processor can run at most one job at any given point in time. Provide an algorithm to do this with a running time that is polynomial in *n*. You may assume for simplicity that no two jobs have the same start or end times.

    Example. Consider the following four jobs, specified by (start time, end-time) pairs. (6 P.M., 6 A.M.), (9 P.M., 4 A.M.), (3 A.M., 2 P.M.), (1 P.M., 7 P.M.). The optimal solution would be to pick the two jobs (9 P.M., 4 A.M.) and (1P.M., 7 P.M.), which can be scheduled without overlapping.