University of Washington
Department of Computer Science and Engineering
CSE 421, Winter 2019

January 25, 2019

Homework 3, Due Wednesday, January 30, 1:29 pm, 2019

Turnin instructions: Electronics submission on canvas using the CSE 421 canvas site. Each numbered problem is to be turned in as a separate PDF.

Problem 1 (10 points):

Suppose that an n-vertex undirected graph $G = (V, E)$ has vertices s and t with the distance from s to t strictly greater than $n/2$. Show that there must exist some vertex v, not equal to either s or t, such that deleting v destroys all $s - t$ paths. (This could be phrased as: show that a graph with *diameter* strictly greater than $n/2$ has an *articulation point*.) Give an algorithm that finds v in $O(n + m)$ time.

Problem 2 (10 points):

Consider a directed graph on n vertices, where each vertex has exactly one outgoing edge. This graph consists of a collection of cycles as well as additional vertices that have paths to the cycles, which we call the *branches*.

Describe a linear time algorithm that identifies all of the cycles and computes the length of each cycle. You can assume that the input is given as an array A, where $A[i]$ is the neighbor of i, so that the graph has the edge $\langle i, j \rangle$.

For clarity, make sure that you describe in English the main steps of your algorithm, and don't just provide code. Justify the correctness of your algorithm.

Problem 3:

In this problem, and in the next, you will implement algorithms for interval scheduling. For the test data, you will have to construct a generator to create random instances of interval scheduling. Two separate methods are proposed for generating an instance I that consists of n intervals. These give slightly different results, but capture at least one notion of randomness.

Method 1 (Permutation based): Start with an array of the endpoints of the intervals (which can be represented just by the number of the interval), and then randomly permute the array. The index of the first j is the start of interval j, and the index of the second j is the end of interval j. For example, if you start with the array $[1, 1, 2, 2, 3, 3, 4, 4]$ and randomly permute it to $[2, 1, 4, 2, 3, 4, 1, 3]$ the intervals are $\{(2, 7), (1, 4), (5, 8), (3, 6)\}$.

Method 2 (Random positions and lengths): Each interval is assigned a random start position in $[0.0, 1.0)$ and is assigned a random length in $[0.0, 1.0)$.

You are free to write in any programming language you like. The quality of your algorithm may be graded, but the actual quality of the code will not be graded. The expectation is that you write the algorithmic code yourself - but you can use other code or libraries for supporting operations (including generating random numbers or random permutations). Submit your code as a PDF.

Programming Problem 3a (5 points):

Write a random interval generator. You may use either of the methods above, or invent your own. If you invent your own, describe how it works.

For this problem, turn in a PDF of your code, and the output of two runs of size 10.

Problem 3b (0 points)[1]:

Before you complete parts c, and d, what is the expected size of a maximum set of disjoint intervals found by the greedy algorithm based on your random interval generator. Can you determine the growth rate of the solution as $O(f(n))$.

Programming Problem 3c (5 points):

Implement the greedy algorithm for the Interval Scheduling Problem to find a maximum set of disjoint intervals. Submit your code as a PDF. Your implementation should be $O(n \log n)$.

Programming Problem 3d (5 points):

How does the size of the the solution grow as a function of n. Run your program across a range of input sizes to get an estimate of the growth of the solution. How does the solution grow. (You should be able to run this on instances of up to size $n = 1,000,000$ without much difficulty, which is big enough to see a good trend.)

Problem 4:

Implement the algorithm for scheduling all intervals, and determine how many processors are needed for a random set of intervals.

Problem 4a (0 points):

Before you complete parts b, and c, what is the expected size of the minimum number of processors needed to schedule a random set of n intervals. Can you determine the growth rate of the solution as $O(f(n))$, or even determine what the constant factor is?

Programming Problem 4b (5 points):

Implement the algorithm for scheduling all intervals to determine the number of processors needed for scheduling the set of intervals. For this problem, you only need to compute the number of processors needed (the *depth*), and you do not need to find the actual assignment of intervals to processors. Submit your code as a PDF. Your implementation should be $O(n \log n)$.

Programming Problem 4c (5 points):

How does the size of the the solution grow as a function of n. Run your program across a range of input sizes to get an estimate of the growth of the solution. How does the solution grow. (You should be able to run this on instances of up to size $n = 1,000,000$ without much difficulty, which is big enough to see a good trend. Depending on computing hardware, you may be able to get up to $n = 100,000,000$.)

[1] Yes, 0 points. We want you to think about this before you run your algorithm, but can't think of any reasonable way of assigning credit.

Problem 5 (10 points):

The paragraphing problem is: Given a set of words $w_1, \ldots, w_n$ with word lengths $l_1, \ldots, l_n$, break the words into consecutive groups, such that the sum of the lengths of the words in each group is less than a fixed value K. (We will ignore the issue of putting spaces between words or hyphenation; these are minor details.) The words remain in the original order, so the task is just to insert line breaks to ensure that each line is less than length K.

The greedy algorithm for line breaking is to pack in as many words as possible into each line, e.g., to put words into a line one at a time until the length bound K is reached, and break the line before the word w_r that caused the the bound to be exceeded.

Prove that the Greedy Algorithm is optimal in the sense that it produces a paragraph with the smallest number of lines. For a formal proof, induction is recommended. The key to this problem is coming up with the right induction hypothesis.

Problem 6 (10 points):

Let $P = \{x_1, \ldots x_n\}$ be points on the X-axis in increasing order. Give an $O(n)$ time algorithm to determine the minimum number of intervals of length R to cover the points. Prove that your algorithm is correct.