HW2

Friday, February 10, 2023 12:02 AM

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: _____ Wisc id: _____

Asymptotic Analysis

1. *Kleinberg, Jon. Algorithm Design (p. 67, q. 3, 4).* Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

   (a) $f_1(n) = n^{2.5}$
   $f_2(n) = \sqrt{2n}$
   $f_3(n) = n + 10$
   $f_4(n) = 10n$
   $f_5(n) = 100n$
   $f_6(n) = n^2 \log n$

   $f_2(n) = \sqrt{2n}$ $f_3(n) = n + 10$ $f_4(n) = 10n$ $f_5(n) = 100n$ $f_6(n) = n^2 \log n$ $f_1(n) = n^{2.5}$

   (b) $g_1(n) = 2^{\log n}$
   $g_2(n) = 2^n$
   $g_3(n) = n(\log n)$
   $g_4(n) = n^{4/3}$
   $g_5(n) = n^{\log n}$
   $g_6(n) = 2^{(2^n)}$
   $g_7(n) = 2^{(n^2)}$

   $g_1(n) = 2^{\log n}$ $g_3(n) = n(\log n)$ $g_4(n) = n^{4/3}$ $g_5(n) = n^{\log n}$ $g_2(n) = 2^n$ $g_7(n) = 2^{(n^2)}$ $g_6(n) = 2^{(2^n)}$

2. *Kleinberg, Jon. Algorithm Design (p. 68, q. 5).* Assume you have a positive, non-decreasing function f and a positive, increasing function g such that $g(n) \geq 2$ and $f(n)$ is $O(g(n))$. For each of the following statements, decide whether you think it is true or false and give a proof or counterexample.

(a) $\log_2 f(n)$ is $O(\log_2 g(n))$

If *f(n)* is *O(g(n))*, then:
f(n) <= cg(n) for some value n >= n_0.
If we take the log of both sides, we have:
$\log_2(f(n)) <= \log_2(cg(n))$ for some value n >= n_0.
Thus, the statement above is True.

(b) $2^{f(n)}$ is $O(2^{g(n)})$

If *f(n)* is *O(g(n))*, then:
f(n) <= cg(n) for some value n >= n_0.
If we raise each side to the power of 2, we have:
2^(f(n)) <= 2^(cg(n)) for some value n >= n_0.
Thus, the statement above is True.

(c) $f(n)^2$ is $O(g(n)^2)$

If *f(n)* is *O(g(n))*, then:
f(n) <= cg(n) for some value n >= n_0.
If we square both sides, we have:
$f(n)^2 <= (cg(n))^2$ for some value n >= n_0.
$f(n)^2 <= c^2g(n)^2$ for some value n >= n_0.
Thus, the statement above is True.

3. *Kleinberg, Jon. Algorithm Design (p. 68, q. 6).* You're given an array A consisting of n integers. You'd like to output a two-dimensional n-by-n array B in which $B[i, j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$ — that is, the sum $A[i] + A[i + 1] + ... + A[j]$. (Whenever $i \geq j$, it doesn't matter what is output for $B[i, j]$.) Here's a simple algorithm to solve this problem.

```
for i = 1 to n
  for j = i + 1 to n
    add up array entries A[i] through A[j]
    store the result in B[i, j]
  endfor
endfor
```

(a) For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).

$O(n^2)$

(b) For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asympto tically tight bound of $\Theta(f(n))$ on the running time.)

Because this algorithm has a nested for loop where the iterations are entirely dependent on *n*, the best case is no better than the worst case. The algorithm will always take the same amount of operations to complete for a given array *A* of size *n*. Thus, it is $\Omega(n^2)$ as well, making it $\Theta(n^2)$.

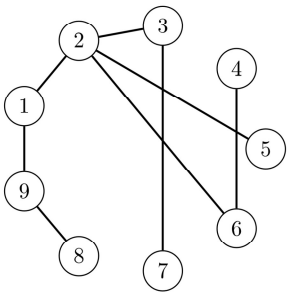
(c) Although the algorithm provided is the most natural way to solve the problem, it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$.

Initialize B to an n-by-n array of zeros
For i = 1 to n
    Add A[i]

    I could not come up with a more efficient algorithm in time

Graphs

4. Given the following graph, list a possible order of traversal of nodes by breadth-first search and by depth-first search. Consider node 1 to be the starting node.



BFS: 1, 2, 9, 3, 5, 6, 8, 7, 4

DFS: 1, 2, 3, 7, 5, 6, 4, 9, 8

5. *Kleinberg, Jon. Algorithm Design (p. 108, q. 5).* A binary tree is a rooted tree in which each node has at most two children. Show by induction that in any binary tree the number of nodes with two children is exactly one less than the number of leaves.

Base case:

number of nodes, n, = 1. In a binary tree with 1 node, there are 0 nodes with two children and 1 leaf. (a leaf is a node without any children, so the root can be a leaf)

Inductive Step:

Assume for a binary tree with k nodes, the number of nodes with two children is exactly one less than the number of leaves. We will prove that for a binary tree with k+1 nodes, the number of nodes with two children is exactly one less than the number of leaves.

There are two cases when adding a node to a binary tree:

Case 1: The new node is being added as a child of a leaf node. In that case, the total number of leaf nodes doesn't change because you are replacing one leaf with another. Also, the number of nodes with two children doesn't change because the former leaf node now only has one child.

Case 2: The new node is being added as a child of a node that has one other child. In that case, both the total number of leaf nodes and total number of nodes with two children increase by 1, as the parent node now has 2 children and the new node is an additional leaf.

Thus, a binary tree always has exactly 1 more leaf node that it does nodes with two children.

6. *Kleinberg, Jon. Algorithm Design (p. 108, q. 7).* Some friends of yours work on wireless networks, and they're currently studying the properties of a network of n mobile devices. As the devices move around, they define a graph at any point in time as follows:

   > There is a node representing each of the n devices, and there is an edge between device i and device j if the physical locations of i and j are no more than 500 meters apart. (If so, we say that i and j are "in range" of each other.)

   They'd like it to be the case that the network of devices is connected at all times, and so they've constrained the motion of the devices to satisfy the following property: at all times, each device i is within 500 meters of at least $\frac{n}{2}$ of the other devices. (We'll assume n is an even number.) What they'd like to know is: Does this property by itself guarantee that the network will remain connected?

   Here's a concrete way to formulate the question as a claim about graphs.

   Claim: Let G be a graph on n nodes, where n is an even number. If every node of G has degree at least $\frac{n}{2}$, then G is connected.

   Decide whether you think the claim is true or false, and give a proof of either the claim or its negation.

Let us assume that G is not connected and that it has 2 disconnected subgraphs, g_1 and g_2. Since every node must have degree of at least n/2, a node in g_1 must be connected to n/2 other nodes in g_1, giving us (n/2)+1 nodes in g_1. The same must be true for a node in g_2. If we try to find the total number of nodes, we must add together the number of nodes in g_1 and g_2:

$$2 \times ((n / 2) + 1) = n + 2$$

We find that there are n+2 total nodes in G, which is a contradiction because we know that there are only n nodes in G. Thus, G must be connected if every node of G has degree at least n/2.

Coding Question

7. Implement depth-first search in either C, C++, C#, Java, or Python. Given an undirected graph with n nodes and m edges, your code should run in $O(n + m)$ time. Remember to submit a makefile along with your code, just as with week 1's coding question.

   Input: the first line contains an integer t, indicating the number of instances that follows. For each instance, the first line contains an integer n, indicating the number of nodes in the graph. Each of the following n lines contains several space-separated strings, where the first string s represents the name of a node, and the following strings represent the names of nodes that are adjacent to node s. You can assume that the nodes are listed line-by-line in lexicographic order (0-9, then A-Z, then a-z), and the adjacent nodes of a node are listed in lexicographic order. For example, consider two consecutive lines of an instance:

   ```
   0, F
   B, C, a
   ```

   Note that 0 < B and C < a.

   Input constraints:

   - $1 \leq t \leq 1000$
   - $1 \leq n \leq 100$
   - Strings only contain alphanumeric characters
   - Strings are guaranteed to be the names of the nodes in the graph.

   Output: for each instance, print the names of nodes visited in depth-first traversal of the graph, *with ties between nodes visiting the first node in input order*. Start your traversal with the first node in input order. The names of nodes should be space-separated, and each line should be terminated by a newline.

   Sample:

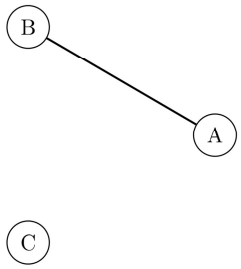
   Input:

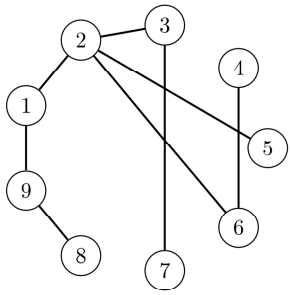
   ```
   2
   3
   A B
   B A
   C
   9
   1 2 9
   2 1 6 5 3
   4 6
   6 2 4
   5 2
   3 2 7
   7 3
   8 9
   9 1 8
   ```

   Output:

   ```
   A B C
   1 2 6 4 5 3 7 9 8
   ```

   The sample input has two instances. The first instance corresponds to the graph below on the left. The second instance corresponds to the graph below on the right.


B
A
C


3
2
4
1
5
9
6
8
7