Problem

Note: You need not include the problem statement in your solution. It is here only so that the solution below makes sense.

For each of the following, answer with the tightest upper bound from this list : \( O(\log n), O(n), O(n \log n), O(n^2), O(2^n) \). Justify your answers.

a) Space required for an adjacency list representation of a graph \( G = (V, E) \)

b) Time to determine whether a graph \( G = (V, E) \) is connected, assuming an adjacency list representation of \( G \)

c) Time to count the number of connected components of a graph \( G = (V, E) \), assuming an adjacency list representation of \( G \)

Solution

a) This representation consists of an array of length \( n = |V| \) with one entry for each vertex. The \( i^{th} \) entry includes a reference to a linked list of the edges incident with vertex \( v_i \). Each edge will appear in exactly two linked lists so the sum of the lengths of the linked lists is \( 2|E| \), for a total space requirement of \( O(|V| + |E|) \) (the array plus the linked lists). By the way, the length of the list corresponding to vertex \( v_i \) is \( \deg(v_i) \), so the sum of the lengths of the lists is also \( \sum_{i=1}^n \deg(v_i) \). So we get the following result for free:

For any graph \( G = (V, E) \),

\[
\sum_{v \in V} \deg(v) = 2|E|
\]

b) Performing either a breadth- or depth-first search from any vertex \( v \) of \( G \) will let you visit those vertices reachable from \( v \). \( G \) is connected if and only if you can visit all vertices starting from (any) single vertex. Since either search technique takes time \( O(|V| + |E|) \)—assuming that \( G \) is represented with the adjacency list structure—we can determine connectedness in this amount of time.

c) Using the ideas from the previous answer, we can modify the search so that the entire graph is visited in \( O(|V| + |E|) \) time.

- Starting with the first entry of the vertex array, perform a search as above. Each time you visit a vertex, mark that vertex in the vertex array.
- When the search has completed, move to the next vertex in the array, if it has not been visited, perform a search starting with it, otherwise move on to the next vertex.
- Continue until all vertices have been visited, then report how many searches you performed.

Each search finds a new component \( G_i = (V_i, E_i) \) of \( G \) and takes time \( O(|V_i| + |E_i|) \), so altogether the searches take time proportional to \( \sum_{i=1}^k O(|V_i| + |E_i|) \), where \( k \) is the number of connected components. Since the only other work performed is walking sequentially through the vertex array to find the next unmarked vertex, this sum is \( O(|V| + |E|) \).

Other Useful Bits of \LaTeX

Maybe you’d like a multi-line equation
\[
\sum_{i=1}^{n} (3i + 2) = \sum_{i=1}^{n} 3i + \sum_{i=1}^{n} 2
\]

\[
= 3 \sum_{i=1}^{n} i + 2n
\]

\[
= \frac{n(n+1)}{2} + 2n
\]

\[
= \frac{3n^2 + 7n}{2}
\]

Sometimes you might like to include pseudo-code in a solution. You might use the algorithm/algorithmic package (see the "use package" lines at the top of this file), like this:

Algorithm 1: Put your caption for the pseudo-code here.

Require: An array of integers \( A \) of length \( n \geq 0 \). Empty arrays return \(+\infty\).

1: \( m \leftarrow +\infty \)
2: \( \text{for } i \leftarrow 1 \text{ to } n \text{ do} \)
3: \( m \leftarrow \min(m, A[i]) \)
4: \( \text{end for} \)
5: \( \text{return } m \)

Feedback:

I found this problem to be (circle one):

1. Easy as pie
2. Moderately easy
3. About what I would expect on average
4. Challenging but fair
5. Are you kidding me? This is supposed to be a 200-level course!!!

Comments: Put comments here.

1. Feedback, civilly phrased, will not affect your grade.
2. But avoid excessive use of footnotes.
3. Really—they are distracting in large numbers!