Recitation - Complexity and Graphs

1 Bits and Pieces

Determine which of the following problems can be computed in worst-case polynomial-time, i.e. $O(n^k)$ time for some constant k, where n denotes the number of bits in the binary representation of the input. If you think the problem can be solved in polynomial time, give an algorithm, explain briefly why it gives the correct answer, and argue carefully why the running time is polynomial. If you think the problem cannot be solved in polynomial time, then provide a proof.

- Give an input positive integer N, output $N!$.
- Given as input a positive integer N, output True iff $N = M!$ for some positive integer M.
- Given as input a positive integer N, output True iff $N = M^2$ for some positive integer M.

Solution. https://www.youtube.com/embed/0CsyjnMy7Ig

Problem 1: To represent the value of $N!$, we will need $\log_2(N!)$ bits. As shown in the course notes, $\log_2(N!) = \Theta(N \log N)$. It takes at least one step to write down a single bit of the output. Therefore, writing out the whole output would take exponential time with respect to the input length n, since $n = O(\log N)$.

Problem 2: The polynomial-time algorithm is as follows. We iteratively compute the values $1!, 2!, 3!, \ldots$. In particular, at iteration M, in order to compute $M!$, we take the result from previous iteration and multiply it by M. In every iteration we check whether

$M! < N$. When $M! \geq N$, we exit the loop and check whether $M! = N$ or $M! > N$. If they are equal, then return true, and false otherwise.

The analysis of the algorithmic complexity is as follows: First, a crude upper bound on the number of iterations is $O(\log N)$. This is because after the second iteration, $M!$ grows by a factor of at least 2 in each iteration. So after $\log_2 N$ iterations, the value of $M!$ exceeds N. In each iteration we do a multiplication operation. If we use the quadratic-time multiplication algorithm, then the running time of the multiplication operation would be at most $O(\log^2 N)$. Therefore the total running time of the loop is $O(\log^3 N)$. The comparison between two numbers $M!$ and N will take $O(\log N)$ time. Thus, the whole algorithm takes $O(\log^3 N) = O(n^3)$ time, which is polynomial in the input length n.

Problem 3: We will perform binary search to pin down the value of x such that $x^2 = N$. At each iteration, we have a guess x for the square root of N. We compute $x \cdot x$ to see if our guess is correct, or whether x^2 is above N or below N. Depending on that, we update our range and calculate a new guess x. Here we will not give the details about the implementation of binary search (that is not the point of the exercise).

Let's assume lo and hi are the two variables in the binary search algorithm that keep track of the range we are interested in (the midpoint of lo and hi corresponds to our guess x). At the beginning of each iteration of the while loop, the invariant $lo \leq \sqrt{N} \leq hi$ will be maintained. Since lo and hi are integers, and $hi - lo$ is decreasing at each iteration, we will find $\sqrt{N}$ if it is also an integer.

To analyze the run-time, we first look at the number of iterations, and then look at the amount of work we do in each iteration.

In terms of the number of iterations, observe that $hi - lo$ decreases by at least a half at the end of each iteration of the loop. Since we start with a range of length N initially, and we decrease the size of this range by a factor of 2 at every iteration, we have at most $O(\log(N))$ iterations.

In terms of the work done in each iteration, note that we compute the product $x \cdot x$, and the value of x is at most N. Hence, we can do this computation in $O(\log^2 N)$ time (just using the grade-school multiplication algorithm). We also perform some comparisons and assignments, each of which only takes $O(\log N)$ time. So the work done in each iteration is $O(\log^2 N)$.

We can conclude that the total time is $O(\log^3 N)$, which is polynomial in $n = \log N$. ■

2 "Clearly" Correct

A connected graph with no cycles is a tree. Consider the following claim and its proof.

> **Claim:** Any graph with n vertices and $n - 1$ edges is a tree.
> **Proof:** We prove the claim by induction. The claim is clearly true for $n = 1$ and $n = 2$. Now suppose the claim holds for $n = k$. We'll prove that it also holds for $n = k + 1$. Let G be a graph with k vertices and $k - 1$ edges. By the induction hypothesis, G is a tree (and therefore by definition connected). Add a new vertex v to G by connecting it with any other vertex in G. So we create a new graph G' with $k + 1$ vertices and k edges. The new vertex we added is a leaf, so it does not create a cycle. Also, since G was connected, G' is also connected. A connected graph with no cycles is a tree, so G' is also a tree. The claim follows by induction.

Explain why the given proof is incorrect.

Solution. https://www.youtube.com/embed/d7K8nE0gpr4

The proof relies on a specific construction to obtain bigger graphs from smaller graphs. This really only proves the statement for all graphs that can be constructed in this way (taking a smaller graph with the property and connecting a new vertex to an existing

vertex). In this case it is easy to see how and why this fails. Consider any counterexample to the claim and observe that it cannot be obtained by connecting a new vertex to a smaller tree.

The general format for induction of graphs is something like this. Suppose we want to prove that all graphs that satisfy hypothesis A have property B.

1. Assume what is to be proven for all graphs on n vertices or less.
2. Consider an **arbitrary** graph on $n+1$ vertices that satisfies hypothesis A.
3. Remove some vertices to obtain a smaller graph that still satisfies hypothesis A.
4. Cite the induction hypothesis to conclude that the smaller graph has property B.
5. Reconstruct the original larger graph and argue that property B is maintained.

■

3 2 3 Proofs 4 You

Give two proofs (one using induction and another using a degree counting argument) for the following claim: the number of leaves in a tree with $n \geq 2$ vertices is

$$2 + \sum_{\substack{v \in V \\ \deg(v) \geq 3}} (\deg(v) - 2).$$

Solution. https://www.youtube.com/embed/pBGlJuLKJbE

As a reminder, any graph satisfying two of the following three properties is a tree: (i) the graph is connected; (ii) the graph is acyclic; (iii) the number of edges is one less than the number of vertices.

We prove the statement by induction on n, the number of vertices. For the base case, the only tree with $n = 2$ vertices is a single edge, which has 2 leaves and no vertices with degree at least 3. Therefore,

$$2 + \underbrace{\sum_{\substack{v \in V \\ \deg(v) \geq 3}} (\deg(v) - 2)}_{=0} = 2,$$

as desired.

For the inductive case, suppose that for some $n \geq 2$, the statement is true for all trees on n vertices. Consider an *arbitrary* tree T on $n+1$ vertices. Every tree with at least two vertices has at least 2 leaves (this is an exercise problem in the textbook). Let u be a leaf in T and let w be its neighbor. Then $T' := T \setminus \{u\}$ is a tree on n vertices since it is connected and acyclic. Note that we cannot have $\deg_T(w) < 2$ since w has u as a neighbor as well as another vertex in T'.

Case 1: $\deg_T(w) = 2$. Then $\deg_{T'}(w) = 1$ and so w is a leaf in T'. It follows that T and T' have the same number of leaves. So

$$\begin{aligned}
\text{Number of leaves in } T &= \text{Number of leaves in } T' \\
&= 2 + \sum_{\substack{v \in V(T') \\ \deg_{T'}(v) \geq 3}} (\deg_{T'}(v) - 2) \quad \text{(by IH)} \\
&= 2 + \sum_{\substack{v \in V(T) \\ \deg_T(v) \geq 3}} (\deg_T(v) - 2),
\end{aligned}$$

where the last equality holds because in both the summation for T and the summation for T', neither u nor w appear.

Case 2: $\deg_T(w) \geq 3$. Then $\deg_{T'}(w) = \deg_T(w) - 1 \geq 2$ and so

$$\begin{aligned}
&\text{Number of leaves in } T \\
&= \text{Number of leaves in } T' + 1 \\
&= \left(2 + \sum_{\substack{v \in V(T') \\ \deg_{T'}(v) \geq 3}} (\deg_{T'}(v) - 2)\right) + 1 \quad \text{(by IH)} \\
&= \left(2 + \sum_{\substack{v \in V(T) \\ \deg_T(v) \geq 3}} (\deg_T(v) - 2)\right) + \underbrace{\big(\deg_{T'}(w) - 2\big) - \big(\deg_T(w) - 2\big)}_{=-1} + 1 \\
&= 2 + \sum_{\substack{v \in V(T) \\ \deg_T(v) \geq 3}} (\deg_T(v) - 2).
\end{aligned}$$

■

Solution. https://www.youtube.com/embed/9XlPD3J7Vqg

We now prove the statement via a degree counting argument. Let $T = (V, E)$ be an arbitrary tree with n vertices, m edges and L leaves. Since T is a tree, we know $n = m+1$. By the handshake lemma,

$$\begin{aligned}
2m = \sum_{v \in V} \deg(v) &= \underbrace{\sum_{\substack{v \in V \\ \deg(v) = 1}} \deg(v)}_{=L} + \sum_{\substack{v \in V \\ \deg(v) \geq 2}} \deg(v) \\
&= L + \left(\sum_{\substack{v \in V \\ \deg(v) \geq 2}} (\deg(v) - 2)\right) + 2(n - L) \\
&= L + \left(\sum_{\substack{v \in V \\ \deg(v) \geq 2}} (\deg(v) - 2)\right) + 2(m + 1) - 2L \\
&= L + \left(\sum_{\substack{v \in V \\ \deg(v) \geq 3}} (\deg(v) - 2)\right) + 2(m + 1) - 2L,
\end{aligned}$$

where the last equality follows because the summation over $\deg(v) \geq 2$ is equivalent to $\deg(v) \geq 3$ (the $\deg(v) = 2$ terms are all 0). Rearranging the terms we get

$$L = 2 + \sum_{\substack{v \in V \\ \deg(v) \geq 3}} (\deg(v) - 2).$$

■

4 (Extra) How About Strictly Less Than?

We use $O(\cdot)$, $\Omega(\cdot)$ and $\Theta(\cdot)$ notation to compare functions in computer science. In particular, $O(\cdot)$ corresponds to $\leq$, $\Omega(\cdot)$ corresponds to $\geq$, and $\Theta(\cdot)$ corresponds to $=$. We now would like to define what "strictly less than" and "strictly greater than" mean with respect to comparing functions. How should we define them?

Solution. In computer science, the standard notation for "strictly less than" is $o(\cdot)$, which is known as the little-o notation. We usually define "strictly less than" to mean "not greater than or equal to". So we can try to define $f(n) = o(g(n))$ to mean "not $f(n) = \Omega(g(n))$". In other words, we cannot find constants $c, n_0 > 0$ such that for all $n \geq n_0$, $f(n) \geq c \cdot g(n)$. Unfortunately this definition does not quite work. This is because we can have functions that are incomparable to each other in the sense that neither $f(n) = O(g(n))$ holds nor $f(n) = \Omega(g(n))$ (try to come up with an example for this). And for such functions, we have "not $f(n) = \Omega(g(n))$", but we can't say $f(n) = o(g(n))$ because $f(n) = o(g(n))$ implies $f(n) = O(g(n))$.

Another way to define "strictly less than" is "less than or equal to but not equal to". So we can try to define $f(n) = o(g(n))$ to mean "$f(n) = O(g(n))$ but not $f(n) = \Theta(g(n))$". This is the definition we can indeed use. Equivalently, we want "$g(n)$ grows asymptotically faster than $f(n)$", or in other words, $\lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty$, which is how little-o notation is usually defined. As a very simple example, we can write $n = o(n \log n)$.

The definition for "strictly greater than" is analogous, and we use little-omega $\omega(\cdot)$ for this. As a simple example, we can write $n \log n = \omega(n)$. ■