First we present a general algo that computes all the vertices that are reachable (i.e. exist paths to) from any given vertex v. (This was discussed in the class).

1. Create a set \( S = \{v\} \). From each vertex in \( S \), compute \( \text{compute}(w) \) on \( v \) is an edge. Add this set to \( S \). If the size of \( S \) increases then repeat the above procedure. We called it the "contiguous method" or "breadth-first search". Since the number of vertices is \( n \) \& since repetition happens only after the size of \( S \) increases by at least 1, there are at most \( n-1 \) iterations. The overall speed is \( O(nm) \). A more careful way results in \( O(m) \) steps.

Now test that all vertices are reachable from the given vertex. This takes \( O(nm) \) steps. Then test that from each of the other vertices \( u \) is reachable. This takes \( O(nm) \) steps. The overall speed is \( O(n^2m) \). Since the algo is in \( P \).

After doing that there is a path from \( u \) to every vertex, the rest of the computation can
be speeded up by first computing
\[ G^2 = \{ (u, v) | (u, w) \text{ is an edge of } G \} \text{ in } O(n^2) \]
steps. Then test whether every vertex is reachable from \( u \) in \( G^2 \).

You don't need to worry about this refinement.

4) Given an NFA \( M \) and a string \( x \), is \( x \in L(M) \)?

We cannot assume that the length of \( |M| \) is for smaller than \( |x| \). So we cannot convert the NFA \( M \) to a DFA. The size of the DFA can be exponentially larger than \( |M| \).

Let \( x = a_1 \ldots a_m \).
For each \( 0 \leq i \leq m \) we compute
\[ S_i = \{ q | \text{NFA } M \text{ can end up in state } q \text{ on the input } a_1 \ldots a_i \} \]
\[ S_0 = \{ q_0 \} \text{ is the initial state of } M \]

Computation of \( S_i \) from \( S_{i-1} \)
\[ S_i = \{ q | q \in S_{i-1} \text{ and } a_i \cdot q \cdot q \rightarrow q \text{ in } M \} \]

After computing \( S_m \), we check whether one of the final states of \( M \) is in \( S_m \).
We will grade this as a BONUS PROBLEM since you are not required to know any aspects of the dynamic programming technique.

6. Algorithm: Let the elements be \( x_0, \ldots, x_n \).

   Compute \( W = \sum_{i=1}^{n} x_i \).

   Check whether any \( x_i > \frac{2W}{3} \). If so, output "yes" else output "no" and halt.

   Clearly the algo runs in \( O(n) \) steps.

   Now we argue that the algo is correct.

   If one \( x_i > \frac{2W}{3} \), then in any partition the part that doesn't contain this \( x_i \) is added up to \( < \frac{W}{3} \).

   If no \( x_i > \frac{2W}{3} \) then we argue that the required partition exists.

   Case 1: If there exists an \( x_k \) s.t. \( \frac{W}{3} < x_k \leq \frac{2W}{3} \), then \( x_0, x_1, x_2, \ldots, x_k \) is OK.

   Case 2: If every \( x_k < \frac{W}{3} \).

   Start with \( S = \{ x_0 \} \).

   Greedily repeatedly & greedily include the next element into \( S \) until the sum of the elements is \( \geq \frac{W}{3} \). This sum must be \( < \frac{2W}{3} \) since each element is \( < \frac{W}{3} \) - now the remaining elements from block 2...
8. CFG G, is \( L(G) = \phi \)?

We first compute \( A = \{ A | \exists x \in \Sigma^+ \text{ (A } \Rightarrow x) \} \).

Then we check whether \( S \in A \), if \( S \notin A \), then:

- \( \text{computation of } \Delta \) (For simplicity, we assume \( G \) in normal form.
- \( A \Rightarrow a \), \( a \in \Sigma \), is a production then include \( A \) in \( G \).
- Repeat the following procedure as long as the size of \( \Delta \) increases.
- For every \( A \Rightarrow BC \), if \( B, C \in \Delta \) then add \( A \) to \( \Delta \).

Since the size of \( \Delta \) is bounded by the number of nonterminals, if there are \( t \) nonterminals, the number of repetitions is \( t - 1 \).

Argue that it runs in \( O(t) \)

If the size of \( G \) is \( n \), then the number of nonterminals is \( \leq n \) and the number of productions is \( \leq n^2 \).

Each repetition takes \( O(n) \) steps. Hence the overall algorithm runs in \( O(n^2) \) steps.

1. digraph \( G, K \)
new guess $v_1, v_2, \ldots, v_k$

and check that $v_1, v_2, \ldots, v_k, v_2$ are distinct

and check that $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k), (v_k, v_1)$ are edges.

If all the tests succeed, output 'yes'; else output 'no' & halt.

Argue the correctness.

Verifier method: Proof needed is $v_1, v_2, \ldots, v_k, v_2$ is the real $g$

the steps for the det. verifier.

$z = a_1 a_2 \ldots a_n$, so $z$ not a prime.

Verifier guess $y = b_1 b_2 \ldots b_m$, $z = c_1 \ldots c_n$, $m \leq n$.

Then check $z = y z$. If so, answer 'yes'; else answer 'no'.

Verifier method: Proof needed is $z, 3$.

Then the verifier is as above.

(If course, in both cases $y$ suffices. Then we divide $z$ by $y$ & test that there is no remainder.)