Path Existence in Directed Graphs

A decision problem: *Given a directed graph G and nodes s and t in G, is there a directed path from s to t in G?*

The corresponding language:

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph having a path from node } s \text{ to node } t\}$$

Brute-force decider for $PATH$: Given directed graph G having m nodes, try every sequence of $k \leq m-2$ distinct nodes $r_1, r_2, \ldots, r_k$, all different from s and t, and test whether $(s, r_1), (r_1, r_2), \ldots, (r_k, t)$ are all directed edges in G.

There are more than $(m-2)!$ such sequences of nodes, so this is clearly not a polytime decider for $PATH$ (but it *is* a decider).

Theorem. $PATH \in \mathrm{P}$.

Proof. Here is a polytime decider for $PATH$:

$M_1 =$ "On input $\langle G, s, t\rangle$, where G is a directed graph with nodes s and t:

1. Place a mark on node s.
2. Repeat until no additional nodes are marked:
3. For every directed edge (a, b) in G:
   If node a is marked and node b is unmarked, mark node b.
4. If node t is marked, *accept*; otherwise, *reject*."

Correctness of M_1: Obvious – uses breadth-first search.

Time complexity of M_1:

- Stages 1 and 4 run only once, and each obviously runs in time polynomial in m (= number of nodes in G)
- Each time it runs, stage 3 simply requires scanning the edges (and possibly re-scanning back to the nodes to check which ones are marked). Regardless of the precise details of its implementation, it can clearly be implemented so that each time it runs it requires time polynomial in the number of edges, which is $O(m^2)$, making its running time polynomial in m.
- The remaining issue is to determine how many iterations occur of the repeat loop. Since it marks at least one node on each iteration except the last, it must run at most m times. (Just noting it's $O(m)$ or, more generally, polynomial in m, is good enough.)

Therefore this is a polytime decider for $PATH$, which proves that $PATH \in \mathrm{P}$.

Relatively Prime Numbers

Two natural numbers x and y are *relatively prime* if the only common factor (divisor) they have is 1.

Examples:

- 9 and 20 are relatively prime
- 14 and 21 are not relatively prime

A decision problem: *Given natural numbers x and y, are they relatively prime?*

The corresponding language:

$$RELPRIME = \{\langle x, y\rangle \mid \ x \text{ and } y \text{ are relatively prime natural numbers}\}$$

For convenience in what follows, assume that $x > y$. Obviously the values of x and y could be exchanged if they don't start out this way.

Brute-force decider for *RELPRIME*: Given x and y, try dividing both of them by every number from 2 through $\min(x, y) = y$.

The number of divisors to try is one less than y. Since the base-b representation (for $b \geq 2$) of y is of size $n = O(\log y)$, this algorithm runs in $O(y) = O(2^n)$ time, so it's an exponential-time, not a polytime, decider.

Relatively Prime Numbers (Continued)

Theorem. *RELPRIME* $\in$ P.

Proof. Here is a polytime decider for *RELPRIME*:

$M_2 =$ "On input $\langle x, y \rangle$, where x and y are natural numbers:
1. Repeat until $y = 0$:
2. Assign $x \leftarrow x \bmod y$.
3. Exchange x and y.
4. If $x = 1$, *accept*; otherwise, *reject*."

$x \bmod y$ is the remainder after performing integer division of x by y.

Correctness of M_2: Stages 1-3 represent the Euclidean algorithm for finding the *greatest common divisor* (gcd) of x and y, which is the largest factor they have in common. We don't prove here that it works, although it's quite easy. When the Euclidean algorithm terminates, x is the gcd. Taking for granted this is true, this algorithm is correct since the gcd of x and y is 1 iff they're relatively prime.

Time complexity of M_2:

- After stage 2, $x < y$ because of the mod function.
- After stage 3, $x > y$ since they've been exchanged.
- What happens to x the next time through the loop?
  - If $y \leq x/2$, then $x \bmod y < y \leq x/2$.
  - If $y > x/2$, then $x \bmod y = x - y < x/2$
- Thus in either case, x drops by at least half.
- Since x and y are exchanged each time through the loop, they both drop by at least half on every two iterations.
- Therefore the maximum number of iterations of the repeat loop is $\min(2 \log_2 x, 2 \log_2 y) = 2 \log_2 y$ since we've assumed $x > y$.
- Thus the number of iterations of the repeat loop is $O(\log y) = O(n)$, where n is the size of the base-b representation of y for $b \geq 2$.

Thus M_2 runs in time polynomial in the size of its input, so it's a polytime decider for *RELPRIME*. Therefore *RELPRIME* $\in$ P.

Eulerian Cycles in Undirected Graphs

An *Eulerian cycle* in an undirected graph is a path that starts and ends at the same node and uses each edge exactly once.

A decision problem: *Given undirected graph G, does it have an Eulerian cycle?*

The corresponding language:

$$EULERCYCLE = \{\langle G \rangle \mid G \text{ is an undirected graph having an Eulerian cycle}\}$$

Brute-force decider for $EULERCYCLE$: For each possible permutation of the edges, test to see if every two adjacent edges (and the last and the first) share a node in the graph. This is clearly not a polytime algorithm in the size of the graph.

Theorem. $EULERCYCLE \in$ P.

Proof. Here is a polytime decider for $EULERCYCLE$:

$M_3 =$ "On input $\langle G \rangle$, where G is an undirected graph:
1. If G is not connected, *reject*.
2. Check the parity of the degree of every node in G.
3. If every node has even degree, *accept*; otherwise *reject*."

The *degree* of a node is the number of edges connected to it.

Correctness of M_3: Rests on the following well-know result from graph theory, which we don't bother to prove here:

Theorem. An undirected graph has an Eulerian cycle iff it is connected and every node has even degree.

Time complexity of M_3:

- Stage 1 easily runs in polynomial time using a marking algorithm.
- Stage 2 is also easily implemented in polynomial time.

Therefore M_3 is a polytime decider for $EULERCYCLE$, proving that $EULERCYCLE \in$ P.