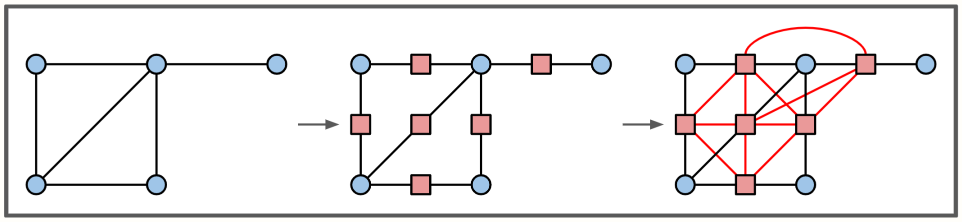
1. A *strongly independent* set is a subset of vertices S in a graph G such that for any two vertices in S, there is no path of length two in G. Prove that *Strongly Independent Set* is NP-hard.

Solution: To show that strongly independent set is NP-hard we do a reduction from independent set.

For any graph G we construct a new graph H where we add vertices that correspond to each edge, these edge vertices are connected to the vertices that the original edges were connected to and to any edge vertex where the original edges shared a vertex.



$\rightarrow$ Let G have an independent set X, then X is a strongly independent set in H. If u, v in G are 2 length away, then the path in H is u connects to the edge vertex of u, which connects to the edge vertex of v which connects to v, so u, v are 3 length away in H. Therefore if G has an independent set of size k, then H has a strongly independent set of size k.

$\leftarrow$ Let H have a strongly independent set Y. Then let w be an edge vertex in Y where the original edge in G connects to vertices a, b. The path from any vertex z (not a) to w is less than or equal to the path from z to a (or b). This is because a connects only to the edge vertices that represent the edges a connects to in G, but these edge vertices that connect to a are all connected to one another. So any path to a has to go through one of these edge vertices. If this edge vertex is w then the path to w is 1 shorter then the path to a. If this edge vertex is not w then this path could go to w instead of a next so the path to w is the same as the path to a. This mean for every edge vertex in Y we can swap it out for a vertex that the edge connected to in G and preserve strong independence. Let Y' be the strongly independent set obtained from swapping out all of the edge vertices. Y' is an independent set in G because if all of the paths from u to v in Y' are greater than 2 in H then u and v cannot be adjacent in G. Therefore if H has a strongly independent set of size k, then G has an independent set of size k.

We have proved both directions in the reduction and constructing the new graph is polynomial time. Therefore because independent set is NP-hard, strongly independent set must also be NP-hard.

■

2. The problem kPARTITION is defined as follows: Given a set S of kn positive integers, determine whether the elements of S can be split into n subsets, each with k elements, whose sums are all equal.

   (a) Describe and analyze a polynomial-time algorithm for 2PARTITION, or prove that it is NP-hard.

Solution: We give a polynomial-time algorithm for 2PARTITION. To define the main algorithm, we first define a recursive helper function as follows:

```
2PARTITIONRECURSIVE($A[1..2n]$):
  If $n = 1$
      return TRUE
  return $A[1] + A[2n] = A[2] + A[2n-1] \wedge$ 2PARTITION($A[2..2n-1]$)
```

With this definition of the recursive helper function, the top level call is:

```
2PARTITION($S$):
  $A[1..2n] \leftarrow$ MERGESORT($S$)
  return 2PARTITIONRECURSIVE($A$)
```

Observe that the running time of 2PARTITIONRECURSIVE($A[1..2n]$) satisfies the recurrence $T(n) = T(n-1) + O(1)$ which has solution $T(n) = O(n)$. Since MERGESORT(S) requires time $O(n \log n)$, the algorithm requires ***time*** $\boldsymbol{O(n \log n)}$, implying it is polynomial-time. ■

(b) Describe and analyze a polynomial-time algorithm for 12Partition, or prove that it is NP-hard.

Solution (direct): We prove that 12Partition is NP-hard via a reduction from 3Partition. Let S be the instance to 3Partition and $m = \max S$. For ease of notation, define $[n] = \{1, 2, \ldots, n\}$ for any positive integer n. Define $T = \left\{5m^3 j + 4m^2 + 4ms \,\middle|\, s \in S, j \in [3]\right\}$ and $S' = S \uplus T$. We claim that there exists a 3-partition of S if and only if there exists a 12-partition of S'.

⇒ Suppose there is a 3-partition $(P_i)_{i=1}^n$ of S. For each $i \in [n]$, let $P_i = \left\{p_{i,j} \mid j \in [3]\right\}$. For all $i \in [n]$, define $T_i = \left\{5m^3 j + 4m^2 + 4mp_{i,j} \,\middle|\, j \in [3]\right\}$, $P'_i = P_i \uplus T_i$ and s'_i as the sum of the elements in P'_i. We have that, for $i, i' \in [n]$ with $i \neq i'$,

$$\begin{aligned}
s'_i &= \sum_{j=1}^{3} p_{i,j} + 5m^3 j + 4m^2 + 4mp_{i,j} \\
&= 30m^3 + 12m^2 + (4m+1)\sum_{j=1}^{k} p_{i,j} \\
&= 30m^3 + 12m^2 + (4m+1)\sum_{j=1}^{k} p_{i',j} \\
&= \sum_{j=1}^{3} p_{i,j} + 5m^3 j + 4m^2 + 4mp_{i',j} \\
&= s'_{i'}.
\end{aligned}$$

For $i, i' \in [n]$ with $i \neq i'$,

$$\begin{aligned}
P'_i \cap P'_{i'} &= (P_i \uplus T_i) \cap (P_{i'} \uplus T_{i'}) \\
&= (P_i \cap P_{i'}) \cup \left(T_i \cap T_{i'}\right) \\
&= \varnothing;
\end{aligned}$$

We show that the last step holds carefully, in particular that $T_i \cap T_{i'} = \varnothing$. Fix any $i, i' \in [n]$ and $j, j' \in [3]$ such that $p_{i,j} < p_{i',j'}$. If $j = j'$, then

$$5m^3 j + 4m^2 + 4mp_{i,j} = 5m^3 j' + 4m^2 + 4mp_{i,j} < 5m^3 j' + 4m^2 + 4mp_{i',j'}.$$

If $j \neq j'$, let $\underline{j} = \min\{j, j'\}$ and $\overline{j} = \max\{j, j'\}$, and let $\underline{i}$, $\overline{i}$, $\underline{i'}$ and $\overline{i'}$ be defined such that $p_{i,j} = p_{\underline{i},\underline{j}}$ and $p_{i',j'} = p_{\overline{i},\overline{j}}$. Since $\underline{j} + 1 \leq \overline{j}$,

$$\begin{aligned}
5m^3 \underline{j} + 4m^2 + 4mp_{\underline{i},\underline{j}} &\leq 5m^3 \underline{j} + 4m^2 + 4m^2 \\
&< 5m^3 \underline{j} + 5m^3 + 4m^2 \\
&= 5m^3(\underline{j} + 1) + 4m^2 \\
&\leq 5m^3 \overline{j} + 4m^2 \\
&< 5m^3 \overline{j} + 4m^2 + 4mp_{\overline{j},\overline{j}}.
\end{aligned}$$

This shows that $T_i \cap T_{i'} = \varnothing$ for all $i, i' \in [n]$ such that $i \neq i'$. Since

$$\begin{aligned} S' &= S \cup T \\ &= \left(\bigcup_{i=1}^{n} P_i\right) \cup \left\{(2k)^2 m^2 + 2kms \;\middle|\; s \in S, j \in [k]\right\} \\ &= \left(\bigcup_{i=1}^{n} P_i\right) \cup \left(\bigcup_{i=1}^{n} \left\{(2k)^2 m^2 + 2kmp_{i,j} \;\middle|\; j \in [k]\right\}\right) \\ &= \bigcup_{i=1}^{n} P_i \cup \left\{(2k)^2 m^2 + 2kmp_{i,j} \;\middle|\; j \in [k]\right\} \\ &= \bigcup_{i=1}^{n} P_i', \end{aligned}$$

we have that $(P_i')_{i=1}^n$ is a $2k$-partition of S'.

$\Leftarrow$ Suppose there is a $2k$-partition $(P_i')_{i=1}^n$ of S'. For all $i \in [n]$, define $P_i = P_i' \cap S$. For all $i \in [n]$, $|P_i| = 3$. Suppose otherwise. Then there exists $i, i' \in [n]$ with $i \neq i'$ such that $|P_i| > 3$ and $|P_{i'}| < 3$. For all $i \in [n]$, let s_i and $s_{i'}$ be the sum of P_i and P_i', respectively. We have that

$$\begin{aligned} s_i' &\leq 12m + \sum_{j=1}^{3} 8\left(5m^3 j + 4m^2 + 4m^2\right) \\ &< 12m + 241m^3 \\ &< 300m^3 \\ &< \sum_{j=1}^{3} 10(5m^3 j + 4m^2 + 4m) \\ &\leq s_{i'}', \end{aligned}$$

a contradiction that $(P_i')_{i=1}^n$ is a $2k$-partition of S'. For $i, i' \in [n]$, we have that $s_i = s_{i'}$. Suppose otherwise. Then for some $i, i' \in [n]$, we have that $s_i < s_{i'}$. But this contradicts that $s_i' = s_{i'}'$, as $s_{i'} - s_i \leq 3m$ but for *any* $x, x' \in S' \setminus S$ with $x < x'$, $x' - x \geq 4m > 3m$. Since for any $i, i' \in [n]$ with $i \neq i'$ $P_i' \cap P_{i'}' = \varnothing$, we have that $P_i \cap P_{i'} = \varnothing$. Because also

$$S = S' \cap S = \left(\bigcup_{i=1}^{n} P_i'\right) \cap S = \bigcup_{i=1}^{n} P_i' \cap S = \bigcup_{i=1}^{n} P_i,$$

we have that $(P_i)_{i=1}^n$ is a k-partition of S.

Assuming there are n elements in S, it requires time $O(n)$ to compute m and time $O(n)$ to compute $5m^3 j + 4m^2 + 4ms$ for all $s \in S$ and $j \in [3]$ thereafter. Thus the reduction requires ***time $O(n)$***, implying it is polynomial-time. ■

Solution (induction): We prove that 12Partition is NP-hard via an alternate strategy. In particular, we show that kPartition $\leq_p$ $2k$Partition for all $k \geq 3$. As 3Partition is NP-hard, this proves inductively a stronger claim than the one we require: $3 \cdot 2^i$Partition is NP-hard for all integers $i \geq 0$.

Let S be the instance to kPartition, $m = \max S$, $T = \left\{(2k)^2m^2 + 2kms \;\middle|\; s \in S\right\}$ and $S' = S \uplus T$. We claim that, for any $k \geq 3$, there exists a k-partition of S if and only if there exists a $2k$-partition of S'.

⇒ Suppose there is a k-partition $(P_i)_{i=1}^n$ of S. For each $i \in [n]$, let $P_i = \left\{p_{i,j} \mid j \in [3]\right\}$. Define $T_i = \left\{(2k)^2m^2 + 2kmp_i \;\middle|\; p_i \in P_i\right\}$ and $P'_i = P_i \uplus T_i$ and s'_i as the sum of the elements in P'_i. We have that, for $i, i' \in [n]$ with $i \neq i'$,

$$\begin{aligned} s'_i &= \sum_{j=1}^{k} p_{i,j} + (2k)^2m^2 + 2kmp_{i,j} \\ &= k(2k)^2m^2 + (2km+1)\sum_{j=1}^{k} p_{i,j} \\ &= k(2k)^2m^2 + (2km+1)\sum_{j=1}^{k} p_{i',j} \\ &= \sum_{j=1}^{k} p_{i,j} + (2k)^2m^2 + 2kmp_{i',j} \\ &= s'_{i'}. \end{aligned}$$

For $i, i' \in [n]$ with $i \neq i'$,

$$\begin{aligned} P'_i \cap P'_{i'} &= (P_i \uplus T_i) \cap (P_{i'} \uplus T_{i'}) \\ &= \left(P_i \uplus \left\{(2k)^2m^2 + 2kmp_{i,j} \;\middle|\; j \in [k]\right\}\right) \\ &\qquad \cap \left(P_{i'} \uplus \left\{(2k)^2m^2 + 2kmp_{i',j} \;\middle|\; j \in [k]\right\}\right) \\ &= (P_i \cap P_{i'}) \cup \Big(\left\{(2k)^2m^2 + 2kmp_{i,j} \;\middle|\; j \in [k]\right\} \\ &\qquad \cap \left\{(2k)^2m^2 + 2kmp_{i',j} \;\middle|\; j \in [k]\right\} \Big) \\ &= \varnothing. \end{aligned}$$

Since

$$\begin{aligned} S' &= S \cup T \\ &= S \cup \left\{(2k)^2m^2 + 2kms \;\middle|\; s \in S\right\} \\ &= \left(\bigcup_{i=1}^{n} P_i\right) \cup \left(\bigcup_{i=1}^{n} \left\{(2k)^2m^2 + 2kmp_{i,j} \;\middle|\; j \in [k]\right\}\right) \\ &= \bigcup_{i=1}^{n} P_i \cup \left\{(2k)^2m^2 + 2kmp_{i,j} \;\middle|\; j \in [k]\right\} \\ &= \bigcup_{i=1}^{n} P_i \cup T_i \\ &= \bigcup_{i=1}^{n} P'_i, \end{aligned}$$

we have that $(P'_i)_{i=1}^n$ is a $2k$-partition of S'.

$\Leftarrow$ Suppose there is a $2k$-partition $(P'_i)_{i=1}^n$ of S'. For all $i \in [n]$, define $P_i = P'_i \cap S$. For all $i \in [n]$, $|P_i| = k$. Suppose otherwise. Then there exists $i, i' \in [n]$ with $i \neq i'$ such that $|P_i| > k$ and $|P_{i'}| < k$. For all $i \in [n]$, s'_i be the sum of the elements of P'_i. We have that

$$\begin{aligned} s'_i &\leq (k+1)m + (k-1)\left((2k)^2m^2 + 2km^2\right) \\ &= (k+1)m + (k-1)(2k)^2m^2 + 2(k-1)km^2 \\ &< (k+1)(2k)^2m^2 + 2k(k+1)m \\ &= (k+1)\left((2k)^2m^2 + 2km\right) \\ &\leq s'_{i'}, \end{aligned}$$

a contradiction that $(P'_i)_{i=1}^n$ is a $2k$-partition of S'. For all $i \in [n]$, let s_i and $s_{i'}$ be the sum of P_i and P'_i, respectively. For $i, i' \in [n]$, we have that $s_i = s_{i'}$. Suppose otherwise. Then for some $i, i' \in [n]$, we have that $s_i < s_{i'}$. But this contradicts that $s'_i = s'_{i'}$, as $s_{i'} - s_i \leq km$ but for *any* $x, x' \in S' \setminus S$ with $x < x'$, $x' - x \geq 2km > km$. Since for any $i, i' \in [n]$ with $i \neq i'$ $P'_i \cap P'_{i'} = \varnothing$, we have that $P_i \cap P_{i'} = \varnothing$. Because also

$$S = S' \cap S = \left(\bigcup_{i=1}^{n} P'_i\right) \cap S = \bigcup_{i=1}^{n} P'_i \cap S = \bigcup_{i=1}^{n} P_i,$$

we have that $(P_i)_{i=1}^n$ is a k-partition of S.

Assuming there are n elements in S, it requires time $O(n)$ to compute m and time $O(n)$ to compute $(2k)^2m^2 + 2kms$ for all $s \in S$ thereafter. Thus the reduction requires ***time $O(n)$***, implying it is polynomial-time. ■

3. A *domino* is a 1×2 rectangle divided into two squares, each of which is labeled with an integer[1]. In a *legal arrangement* of dominos, the dominos are lined up end-to-end so that the numbers on adjacent ends match. An example of a legal arrangement of dominos is given below:

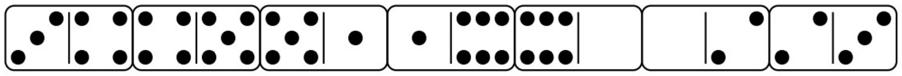


Figure 1. A legal arrangement of dominos in which every integer between 0 and 6 appears twice.

For each of the following problems, either describe and analyze a polynomial-time algorithm or prove that the problem is NP-*complete*:

(a) Given an arbitrary bag D of n dominos, is there a legal arrangement of *all* the dominos in D?

Solution: We describe a polynomial-time algorithm by using graph modeling. In particular, we reduce the given problem to the problem of determining if there is an *Eulerian* walk (i.e., a walk that traverses every edge) in some graph. Construct an undirected unweighted *multigraph* $G(V, E)$ as follows:

- $i \in V \iff$ there is a domino in D with one of its squares labeled i.
- $(i, j) \in E \iff$ there is a domino in D where one of its squares is labeled i and the other is labeled j.

In particular, if multiple dominos have the same two numbers, there will be multiple edges between the same two vertices in G. G can be constructed by brute force in time $O(V + E)$. We find if there is an Eulerian walk in G in two parts:

- We first check that the graph is connected by using WHATEVER-FIRST-SEARCH starting from any vertex in G. This requires time $O(V + E)$. If the graph is not connected, there is no such walk.
- Otherwise, compute $\deg(v)$ for all $v \in V$. In doing so, for a fixed $v \in V$, a self-loop on v contributes **2** to $\deg(v)$ and multiple edges between two vertices u and v are counted *separately* in computing $\deg(u)$ and $\deg(v)$. This requires time $O(V + E)$ as it only requires scanning the adjacency list of G once. Let $V_{\text{odd}} = \{v \mid \deg(v) \text{ is odd}\}$. There is an Eulerian walk in G in this case (i.e., G is connected) if and only if $|V_{\text{odd}}| = 0$ or $|V_{\text{odd}}| = 2$.

If there is an Eulerian walk in G, return TRUE. Otherwise, return FALSE. In terms of the *graph* G, this algorithm requires time $O(V + E)$. Since $\max\{V, E\} \leq 2n$, construction of G can be done by brute force in time $O(n)$ in terms of the *dominos instance* D. Thus, this algorithm requires ***time*** $\boldsymbol{O(n)}$ and is indeed polynomial-time. ■

[1]These integers are usually represented by pips, exactly like dice. On a standard domino, the number of pips on each side is between 0 and 6, although one can buy sets with up to 9 or even 12 pips on each side; we will allow arbitrary integer labels. A standard set of dominos contains exactly one domino for each possible unordered pair of labels; we do *not* assume that the inputs to our problems have this property.

(b) Given an arbitrary bag D of n dominos and an integer k, is there a legal arrangement of dominos from D in which every integer between 1 and k appears exactly twice?

Solution: We prove this problem $\mathscr{P}$ is NP-hard by a polynomial-time reduction from the undirected Hamiltonian cycle problem. Suppose $G(V, E)$ is the instance to Hamiltonian cycle. We construct a dominos instance to $\mathscr{P}$ as follows:

- Fix an arbitrary bijection $\ell : V \to \{1, 2, \dots, V\}$. ℓ can be thought of as an assignment of each vertex a *unique* arbitrary integer from 1 to V.
- Domino d with one square labeled $\ell(u)$ and the other labeled $\ell(v)$ is in $D \iff (u, v) \in E$.

Suppose $\mathscr{A}$ is an oracle for $\mathscr{P}$. Input D as constructed above to $\mathscr{A}$ and output the output of $\mathscr{A}$ as the solution to the Hamiltonian cycle with instance G. The correctness of this reduction follows from the following claim: G has a Hamiltonian cycle $\iff (D, V)$ is a positive instance to $\mathscr{P}$. This is proven below:

$\Rightarrow$ Suppose G has a Hamiltonian cycle $v_0, v_1, \dots, v_{V-1}, v_0$. Then $(v_i, v_{i+1 \bmod V}) \in E$ for *all* $0 \le i \le V-1$ by the definition of cycle as a walk. For each $1 \le i \le V$, d_i be a domino with one of its squares labeled $\ell(v_i)$ and its other square labeled $\ell(v_{i+1 \bmod V})$. By construction, D contains d_i for *all* $0 \le i \le V-1$. But $(d_i)_{i=0}^{V-1}$ is a legal arrangement of all dominos in D in which every integer between 1 and V appears exactly twice. Thus, (D, V) is a positive instance to $\mathscr{P}$.

$\Leftarrow$ Suppose (D, V) is a positive instance to $\mathscr{P}$, and let $\mathsf{A} = (d_i)_{i=0}^{V-1}$ be a legal arrangement of all dominos in D in which every integer between 1 and V appears exactly twice. By the definition of legal arrangement, d_i has a label ℓ_{i+1} in common with d_{i+1} for *all* $0 \le i \le V-2$; note carefully the indexing here. Since labels ℓ_i for $1 \le i \le V-1$ appear exactly twice in A, we have that A is a sequence of dominos $(d_i)_{i=0}^{k}$ where each d_i has labels ℓ_i and ℓ_{i+1} for $1 \le i \le V-2$; again, note carefully the indexing. But d_{V-1} must have a label ℓ_V in common with d_0 that is not a label of either square in d_j for *any* $j \in \{0, 1, \dots, V-1\} \setminus \{0, V-1\}$; otherwise, we contradict that each integer from 1 to V appears exactly twice in A. Observe that the inverse ℓ^{-1} of ℓ is a well-defined function as ℓ is a bijection. Define $v_i = \ell^{-1}(\ell_i)$ for all $1 \le i \le V$ and the resulting sequence of vertices $C = v_1, v_2, \dots, v_V, v_1$ in G. By construction, each (v_i, v_{i+1}) is an edge in G, implying C is a closed walk. Since ℓ^{-1} is also a bijection and $\ell_i \neq \ell_j$ for $i, j \in \{1, 2, \dots, V\}$ with $i \neq j$, C does not repeat vertices in G and goes through every vertex in G exactly once. This implies that C is a Hamiltonian cycle in G.

This reduction requires ***time*** $\boldsymbol{O(V+E)}$ as (D, V) can be constructed by brute force in a single pass through G's adjacency list. This verifies that the reduction is indeed polynomial-time and thus that $\mathscr{P}$ is NP-hard.

We now show that $\mathscr{P}$ is NP. We assume that $\mathscr{P}$Certifier below receives as input a positive instance (D, k) of $\mathscr{P}$ as well as an sequence of distinct dominos $\mathsf{A} = (d_i)_{i=0}^{k-1}$ in D and each $d_i = (d_{i,1}, d_{i,2})$ for $0 \le i \le k-1$ is a domino with labels $d_{i,1}$ and $d_{i,2}$. It is defined as follows:

```
$\mathcal{P}$Certifier($D, k$, A):
    for $i \leftarrow 1$ to $k$:
        $labelCount[i] \leftarrow 0$
    for $i \leftarrow 1$ to $k$:
        for $j \leftarrow 1$ to 2:
            if $1 \le d_{i,j} \le k$:
                $labelCount[d_{i,j}] \leftarrow labelCount[d_{i,j}] + 1$
    for $i \leftarrow 1$ to $k$:
        if $labelCount[i] \neq 2$:
            return False
    for $i \leftarrow 0$ to $k-1$:
        if $d_{i,2} \neq d_{i+1 \bmod k,1}$:
            return False
    return True
```

$\mathcal{P}$Certifier is a correct certifier for $\mathcal{P}$ as $\mathcal{P}$Certifier returns True if and only if A is a legal arrangement of dominos in D where every integer from 1 to k appears exactly twice. Moreover, $\mathcal{P}$Certifier runs in ***time $O(k)$*** in terms of the dominos instance (D, k), verifying that $\mathcal{P}$Certifier is a polynomial-time certifier of $\mathcal{P}$. The existence of a polynomial-time certifier for $\mathcal{P}$ proves it is NP.

$\mathcal{P}$ being NP-hard and NP gives that $\mathcal{P}$ is NP-complete. ■

4. For each of the following decision problems, either sketch an algorithm or prove that the problem is undecidable. Recall that w^R denote the reversal of string w. For each problem, the input is an encoding $\langle M, w\rangle$ of a Turing machine M and its input string w.

   (a) Does M either accept w or reject w^R?

Solution: Let L be the language corresponding to this problem. For the sake of argument, suppose there is an algorithm there exist an algorithm DecideL that decides the language L. Then we can solve the halting problem as follows:

```
DecideHalt($\langle M, w\rangle$):
    Encode the following Turing machine $M'$:
        $M'(x)$:
            run $M$ on input $w$
            return True
    return DecideL($\langle M', w\rangle$)
```

We prove this reduction correct as follows:

$\Longrightarrow$ Suppose $\langle M, w\rangle \in$ Halt.
Then M halts on input w.
Then M' accepts *every* input string x.
In particular, M' accepts w.
So $\langle M', w\rangle$ is in L.
So DecideL accepts $\langle M', w\rangle$.
So DecideHalt accepts $\langle M, w\rangle$.

$\Longleftarrow$ Suppose $\langle M, w\rangle \notin$ Halt.
Then M does *not* halt on input w.
Then M' diverges on *every* input string x.
Then M' accepts *no* string and rejects *no* string.
In particular, M' does not accept w or reject w.
So $\langle M', w\rangle$ is *not* in L.
So DecideL rejects $\langle M', w\rangle$.
So DecideHalt rejects $\langle M, w\rangle$.

In both cases, DecideHalt is correct. But that's impossible, because Halt is undecidable. We conclude that the algorithm DecideL cannot not exist. So L must be undecidable. ■

(b) If we run M on input w, does M ever change a symbol on its tape?

Solution: Let L be the language corresponding to this problem and let Q be the states of M. To decide L, we can build a Turing machine M' that simulates M for $|Q|(|w|+1)$ steps. If M changes a symbol within the $|Q|(|w|+1)$ steps, M' returns True. Otherwise, M' returns False. M' decides L so L is decidable.

The reasoning for the construction of M' is as follows. There are only $|Q|(|w|+1)$ distinct configurations for M, because the current state and the symbol at the current tape head location completely determine the transition, of which there are $|w|+1$ choices. Therefore, if M does not change a symbol on its tape within $|Q|(|w|+1)$ steps, then M would *never* change a symbol on its tape. ■

(c) If we run M on input w, does M ever reenter its start state?

Solution: Let L be the language corresponding to this problem. For the sake of argument, suppose there is an algorithm there exist an algorithm DECIDEL that decides the language L. Then we can solve the halting problem as follows:

```
DECIDEHALT($\langle M, w\rangle$):
  Encode the following Turing machine $M'$:
    $M'(x)$:
      leave start
      run $M$ on input $w$ while
        not reentering start
      reenter start
      return TRUE
  return DECIDEL($\langle M', w\rangle$)
```

The Turing machine M' encoded in DECIDEHALT is well-defined as we can enforce that M' simulates M on w using states outside of its start state. We prove this reduction correct as follows:

$\Longrightarrow$ Suppose $\langle M, w\rangle \in$ HALT.

Then M halts on input w.

Then M' reenters its start state if run on input w.

So $\langle M', w\rangle$ is in L.

So DECIDEL accepts $\langle M', w\rangle$.

So DECIDEHALT accepts $\langle M, w\rangle$.

$\Longleftarrow$ Suppose $\langle M, w\rangle \notin$ HALT.

Then M does *not* halt on input w.

Then M' diverges on *every* input string x.

Then M' does *not* reenter its start state.

So $\langle M', w\rangle$ is *not* in L.

So DECIDEL rejects $\langle M', w\rangle$.

So DECIDEHALT rejects $\langle M, w\rangle$.

In both cases, DECIDEHALT is correct. But that's impossible, because HALT is undecidable. We conclude that the algorithm DECIDEL cannot not exist. So L must be undecidable. ■