Exercise 1.2 The minimal number of parameters (pairwise distances) that is required to completely specify a TSP instance is $\binom{n}{2}$, the binomial coefficient n over 2. Given a set with n elements it describes the number of subsets with exactly k (in our case 2) elements. (Sub*sets*, thus the order is irrelevant, as required.)

Exercise 1.4 The number of possible routes as a function of the number of cities is $f(n) = \left[{n \atop 1}\right]$, the nth stirling number of the first kind with $k = 1$. There are $n!$ different ways of arranging the n cities into a route, but many of them are identical from our perspective. See for example all $4! = 24$ permutations of $\{1, 2, 3, 4\}$,

$$
\begin{aligned}
(1,2,3,4) &= (2,3,4,1) = (3,4,1,2) = (4,1,2,3)\\
(2,1,3,4) &= (3,4,2,1) = (4,2,1,3) = (1,3,4,2)\\
(3,1,2,4) &= (1,2,4,3) = (2,4,3,1) = (4,3,1,2)\\
(1,3,2,4) &= (4,1,3,2) = (2,4,1,3) = (3,2,4,1)\\
(2,3,1,4) &= (4,2,3,1) = (3,1,4,2) = (1,4,2,3)\\
(3,2,1,4) &= (4,3,2,1) = (1,4,3,2) = (2,1,4,3)
\end{aligned}
$$

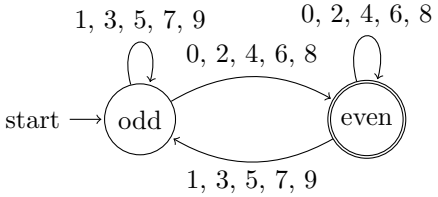
which only admit six different *cycles* of length four. The expression $\left[{n \atop 1}\right]$ reprents the number of permutations over n elements with one cycle, which is what we are looking for.

Problem 1.9 See Exercises 1.2 and 1.4.

Problem 1.11 We want to show that: „Every symbol from a given alphabet can be represented as a bitstring. This representation is one-to-one and only scales logarithmically in alphabet size.“ Consider an alphabet A with n symbols where each symbol $a_0, a_1, \ldots, a_{n-1}$ can be represented by an unique integer. Because we can use the index of an element as its integer counterpart, at least one such representation exists

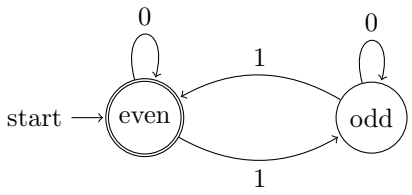
We can uniquely represent an integer c as a bitstring b through the following algorithm: Zero all bits in b. Find the largest x such that 2^x divides c. Set b_x to one and set c to $c - 2^x$. Continue until c is zero. The length of such a bitstring is determined by the x of the first iteration, so the first x such that $2^x \mid c$. Given only c, the length of the resulting bitstring will thus be $\lfloor log_2(c) \rfloor$.

Exercise 2.1 The relevant state diagram is



The runtime of this machine is equal to the length of the decimal encoding, thus $\lfloor \log_{10}(n) \rfloor + 1$. This is very similar to the runtime of a binary parity checking machine since both of them have logarithmic growth, but because $\log_{k+1}$ grows slower than $\log_k$ our machine is slightly faster.

Exercise 2.2 The relevant state diagram is



For $n = 2$ it is equivalent to XNOR. (Choosing the odd state to be the accepting state would make it equivalent to XOR.)

Exercise 3.3 By definition we call a finite string $x = x_0 \cdots x_{n-1}$ a palindrome if

$$x_{n-1}x_{n-2} \cdots x_1 x_0 = x_0 x_1 \cdots x_{n-2} x_{n-1}.$$

From this we can write down $n - 1$ constraints:

$$x_{n-1} = x_0 \qquad x_{n-2} = x_1 \qquad x_{n-3} = x_2 \qquad \cdots \qquad x_{n-k} = x_{k-1}$$

for $k = 1, \ldots, n-1$. But it is clear that some of these constraints are effectively duplicates. Some consideration reveals that exactly $\lfloor n/2 \rfloor$ of them are unique. This is effectively what Lemma 3.2 is about. *I suspect the concrete equation given in Lemma 3.2 is false, didn't continue.*

Exercise 3.5

a) Consider the binary alphabet and let $n \in \mathbb{N}$ be an odd number. Each palindrome x of length n must obey exactly $\lfloor n/2 \rfloor$ constraints, see Exercise 3.3. In particular, only the first $\lfloor n/2 \rfloor$ bits are relevant for determining whether or not x is a palindrome. There are a total of $2^{\lfloor n/2 \rfloor}$ distinct bitstrings of length $\lfloor n/2 \rfloor$.

   The number of palindromes of length n is thus $2^{\lfloor n/2 \rfloor}$.

b) The only part of the above that depends on alphabet length is determining the amount of unique strings of a given length. It can be generalized by letting N be our alphabet size, then there are a total of $N^{\lfloor n/2 \rfloor}$ strings of length $\lfloor n/2 \rfloor$.

Exercise 3.7 For $M = 0$ we have $0 = 0$. Assume that $\sum_{j=0}^{M} j = M(M+1)/2$ holds. We show

$$\sum_{j=1}^{M+1} j = \sum_{j=0}^{M} + M + 1 = \frac{M(M+1)}{2} + M + 1 = \frac{M(M+1) + 2M + 2}{2} = \frac{M^2 + 3M + 2}{2}$$
$$= \frac{(M+1)(M+2)}{2}$$