1) Applying Warshall's algorithm for this graph, numbering nodes clockwise from 0 (corresponding to a) at the top–left through 4 (corresponding to e) at the bottom–left, the WebApp yields as output:

$$\begin{pmatrix} - & 1 & - & 1 & 1 \\ - & - & 1 & - & 1 \\ - & - & - & - & - \\ - & - & 1 & - & - \\ - & - & - & 1 & - \end{pmatrix} \overset{\textit{Preprocessing}}{\rightarrow} A_0 = \left(\begin{array}{c|cccc} 0 & 1 & 0 & 1 & 1 \\ \hline 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right) \rightarrow A_1 = \left(\begin{array}{cc|ccc} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right) \rightarrow A_2 = \left(\begin{array}{ccc|cc} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right)$$

$$\rightarrow A_3 = \left(\begin{array}{ccc|c|c} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 \end{array}\right) \rightarrow A_4 = \left(\begin{array}{cccc|c} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 \end{array}\right) \rightarrow A_5 = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \overset{\textit{Solution}}{\rightarrow} \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

with an entry $A_{k\,ij}$ in matrix A_k being 1 iff there is a path from node i to node j through nodes $\{0, \ldots, k-1\}$ in the initial graph.

From the A_5/solution matrix we read off that $a\ R^+\ c$ holds (1 in the 3rd column of the 1st row), but also that neither $c\ R^+\ a$ (0 in the 1st column of the 3rd row) nor $d\ R^+\ b$ (0 in the 2nd column of the 4th row) hold. Indeed, there is a (non-empty) path from a to b in the initial graph, but there neither is a path from c to a nor a path from b to d.

The A_5/solution matrix here is obtained from the solution matrix D of 2nd exercise of the 1st Exercise sheet, by mapping every ∞ (corresponding to there being no path) and 0 (corresponding to the empty path being the shortest) in D to 0 (there is no non-empty path) and every positive natural number (corresponding to the shortest path being non-empty) in D to 1 (corresponding to there being a non-empty path).

2) We have $T = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa, aaab, aaba, aabb, abaa, abab, abba, abbb, baaa, baab, baba, babb, bbaa, bbab, bbba, bbbb\}$; there are 31 strings in total in T: $1 = 2^0$ of length 0, namely the empty string ϵ, $2 = 2^1$ of length 1, namely a and b, $4 = 2^2$ of length 2, $8 = 2^3$ of length 3 and $16 = 2^4$ of length 4.

To concisely write down the relation $\sqsubset$, we first observe that it relates every string to exactly one other string (of the same length). We can exploit this by abbreviating '$s \sqsubset t$ and $t \sqsubset u$' to '$s \sqsubset t \sqsubset u$'[1] and similarly for longer chains, to obtain: $\epsilon \sqsubset \epsilon$,[2] $a \sqsubset a$, $b \sqsubset b$, $aa \sqsubset aa$, $ab \sqsubset ba \sqsubset ab$, $bb \sqsubset bb$, $aaa \sqsubset aaa$, $aab \sqsubset baa \sqsubset aba \sqsubset aab$, $abb \sqsubset bab \sqsubset bba \sqsubset abb$, $bbb \sqsubset bbb$, $aaaa \sqsubset aaaa$, $aaab \sqsubset baaa \sqsubset abaa \sqsubset aaba \sqsubset aaab$, $aabb \sqsubset baab \sqsubset bbaa \sqsubset abba \sqsubset aabb$, $abab \sqsubset baba \sqsubset abab$, $abbb \sqsubset babb \sqsubset bbab \sqsubset bbba \sqsubset abbb$, $bbbb \sqsubset bbbb$.

[1] In the same way one usually abbreviates, say, '$15 < x$ and $x < 17$' to '$15 < x < 17$'.

[2] This is up for debate given the (ambiguous) specification provided in the exercise. We have interpreted it here as expressing: t is obtained from s by moving *every* letter that is the last letter of s, to the front. Then ϵ indeed is related to itself by $\sqsubset$. Alternatively, the specification can be interpreted as expressing: there *exists* a letter that is the last letter of s and that is moved to the front to obtain t. Then ϵ would not be $\sqsubset$-related to any string, in the absence of a last letter in it.

$\sqsubset$ is not reflexive since, e.g., $ab \not\sqsubset ab$. $\sqsubset$ is not irreflexive since, e.g., $aa \sqsubset aa$. $\sqsubset$ is not symmetric since, e.g., $aab \sqsubset baa$ but not $baa \sqsubset aab$. $\sqsubset$ is not anti-symmetric since, e.g., $ab \sqsubset ba \sqsubset ab$ but not $ab = ba$. $\sqsubset$ is not transitive since, e.g., $aab \sqsubset baa \sqsubset aba$ but not $aab \sqsubset aba$.

$\sqsubset^+$ is reflexive since $\epsilon \sqsubset \epsilon$ and any string s of length $n > 0$ is related to itself via a 'path' $s \sqsubset \ldots \sqsubset s$ of length n; n times rotating a string of length n yields the string again (possibly sooner), as seen above. $\sqsubset^+$ is not irreflexive since it extends $\sqsubset$ which is not irreflexive. $\sqsubset^+$ is symmetric since $\sqsubset$ relates any string to exactly one other string (it represents the 'rotation' function) and since we have already seen that $\sqsubset^+$ is reflexive, i.e. that $\sqsubset$ is cyclic, any 'path' $s \sqsubset^+ t$ can be extended by $t \sqsubset^+ s$ into a cycle on s. $\sqsubset^+$ is not anti-symmetric since it extends $\sqsubset$ which is not anti-symmetric. $\sqsubset^+$ is transitive by definition of it being the transitive closure of $\sqsubset$.

3) a) The respective sets of pairs are, from left to right:

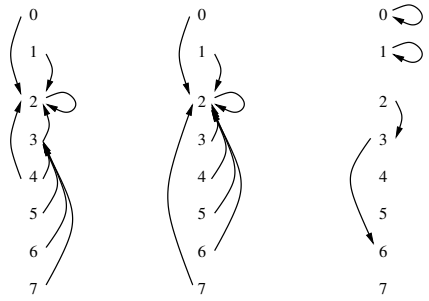
$$G_1 = (0,0), (1,0), (2,1), (3,1), (4,2), (5,2), (6,3), (7,3), \ldots$$

$$G_2 = (0,1), (1,2), (2,3), (3,4), (4,5), (5,6), (6,7), (7,8), \ldots$$

$$G_3 = (0,1), (0,2), (2,3), (2,4), (4,5), (4,6), (6,7), (6,8), \ldots$$

The left two graphs/relations G_1 and G_2 represent functions since for every natural number n, there is *exactly one* pair of shape $(n, \ldots)$ in the relation. In Haskell they can be written as `g1 = (`div` 2)` respectively `g2 = (+1)`. The right graph G_3 is not a function, for two reasons: for every even n there is *more than one* (namely 2) pair of shape $(n, \ldots)$ in the relation, and for every odd n there are *fewer than one* (namely 0) pairs of shape $(n, \ldots)$ in the relation.

b) The three graphs can be drawn, from left to right, as:



R_1 is not a function since both $(4,2)$ and $(4,3)$ are in R_1. R_2 is a function since $m+2 = m^2$ only has $m = 2$ as solution in the natural numbers.[3] The corresponding Haskell function is `f2 n = 2`. For any n, whether the number n is related by R_3 to $n + m$ is uniquely determined by whether the *smaller* number $n - 1$ is related by R_3 to m, yielding that we can construct for any n a unique output by starting constructing them from input 0 giving output 0, then 1 gives output 1 $(1 + 0)$ by using the former, etc.. R_3 corresponds to the Haskell function `f3 n = if (n == 0) then 0 else n + f3 (n-1)`;

4∗) The answer to this question depends on whether or not we include 0 among the natural numbers.

If we do include 0 among the natural numbers, then the specification of f for input 0 expresses only that $f(0) = f(\frac{0}{2}) = f(0)$, i.e. the specification puts no constraints on what the value of

[3]The same would not hold for the integers as then $m = -1$ would be another solution.

$f(0)$ could be. Thus, if there is a function satisfying the specification at all, it will never be unique (the value for $f(0)$ could be changed then, and still the specification would be satisfied).

If we do not include 0 among the natural numbers, then for f to be a function, for all n, the computation of $f(n)$ should have shape $f(n) = f(\ldots) = \ldots = f(1) = 1$, i.e. it should eventually evaluate to 1. But whether this is true or not, for arbitrary n is currently an open problem. It might be the case that for some n, computing $f(n)$ never terminates and then f is not a function, as for that n there is no pair $(n, \ldots)$, or stated more precisely, any pair (n, m) would meet the specification so the specification does not specify a unique function.

From this, we learn that even given a seemingly fine specification of an algorithm it may be not be clear whether that in fact specifies a mathematical function. In Haskell it is:

```
f n = if (n == 1) then 1 else f (if (n `mod` 2 == 0) then n `div` 2 else 3*n+1)
```

Experimenting with that code, although the result always *seems* to be 1, and quite quickly so, it is currently an open problem whether this holds for *all* n.