1. 30 20 56 75 31 19 are hashed to the following values, respectively:

\[ 8 \ 9 \ 1 \ 10 \ 0 \ 2 \]

2. (a) The chained hash table looks as follows:

\[
\begin{aligned}
0 & : \quad \_\_ \\
1 & : \quad 1 \rightarrow 8 \\
2 & : \quad 2 \\
3 & : \quad 3 \rightarrow 10 \\
4 & : \quad \_\_ \\
5 & : \quad \_\_ \\
6 & : \quad \_\_ \\
\end{aligned}
\]

(b) 1 3 2 8 10 are hashed to the following values, respectively:

\[ 1 \ 3 \ 2 \ 4 \ 5 \]

In particular, in case of 8, the first attempt is \(8 \mod 7 = 1\), the second attempt is \(1 + 1 \mod 7 = 2\), the third attempt is \(2 + 1 \mod 7 = 3\), and the third attempt is \(3 + 1 \mod 7 = 4\), which is an empty slot.

And in case of 10, the first attempt is \(10 \mod 7 = 3\), the second attempt is \(3 + 1 \mod 7 = 4\), and the third attempt is \(4 + 1 \mod 7 = 5\), which is an empty slot.

(c) 1 3 2 8 10 are hashed to the following values, respectively:

\[ 1 \ 3 \ 2 \ 4 \ 0 \]

In particular, in case of 8, \(h'(8) = 6 - 3 = 3\). So the first attempt is \(8 \mod 7 = 1\),
and the second attempt is \(1 + 3 \mod 7 = 4\), which is an empty slot.

And in case of 10, \(h'(10) = 6 - 0 = 6\). So the first attempt is \(10 \mod 7 = 3\),
the second attempt is \(3 + 6 \mod 7 = 2\), the third attempt is \(2 + 6 \mod 7 = 1\), and the
fourth attempt is \(1 + 6 \mod 7 = 0\), which is an empty slot.

3. This yields an uneven distribution of the words in the English language over the 26 slots.
For instance, relatively few words start with a q or x, so that relatively few words are hashed to the two slots connected to these two letters.

4. \((\frac{1}{m})^{n-1}\)

5. 23 persons, because

\[
\frac{364 \times 363 \times 362 \times 361 \times 360 \times 359 \times \ldots \times 1}{365^{22}} = 0.493
\]

**Remark:** The so-called *birthday attack* uses this phenomenon to reduce the complexity of cracking a cryptographic hash function.

6. To detect cycles in a directed graph using depth-first search, we modify depth-first search. A node \(u\) is kept on the stack until we backtrack from \(u\) for the last time, meaning that all descendants of \(u\) have been discovered.

The idea of cycle detection is: if we encounter a node which is already on the stack, we have found a cycle (the stack now contains nodes on a path, so if we see a node while it is on the stack, the path contains a cycle). And vice versa, if there is a cycle, then depth-first search will at some point encounter which is already on the stack.

We use three colours: white for unvisited nodes, gray for nodes on the stack, and black for nodes that have been popped from the stack (we backtracked from this node because we have seen all its descendants).

The modified depth-first search with cycle detection works as follows:

- Initially only the root node is on the stack \(S\), and this node is gray, while all other nodes are white.
- While \(S\) is nonempty do
  - Peek at the top \(u\) of \(S\).
  - If \(u\) has a gray child, then there is a cycle.
  - Else, if \(u\) has a white child \(v\), then colour \(v\) gray and push it onto \(S\).
  - Else, colour \(u\) black and pop it from \(S\).

An example where this cycle detection may take long to detect a cycle close to the root is a directed graph consisting a long branch \(u_0 \rightarrow u_1 \rightarrow \cdots \rightarrow u_n\) and a short cycle at the root: \(u_0 \rightarrow v\) and \(v \rightarrow u_0\).

If depth-first search starts with exploring the long branch (and not the short cycle), then it continues until the very end \(u_n\) of this branch, colouring \(u_0, \ldots, u_n\) gray. Next depth-first search backtracks, colouring \(u_n, \ldots, u_1\) black, to reach \(u_0\) again (which remains gray). Finally \(v\) is put on the stack, and its gray child \(u_0\) is discovered, so that the cycle is detected.
7. In this context a drawback of breadth-first search is that one has to keep track of the
different points at the current exploration depth from which one step has to be taken
to the next depth. While this is no problem for a computer on a graph data structure
(these points are kept in a queue), for a human physically walking through a labyrinth
this is problematic. One would need to put e.g. pebbles at all those different points,
and find one’s way back to these pebbles over and over again to make one step and put
another pebble.

Depth-first search is therefore much easier to use in a labyrinth. One only has to take
care not to run through a cycle forever. Therefore one needs to permanently mark the
places where one has been (using e.g. paint on the floor or a sequence of stones like
Little Thumb). If one runs into paint or stones, one needs to backtrack to a point where
a yet unexplored branch can be taken.

8. A possible scenario starting at $p$. First depth-first search is applied to the graph, starting
in $p$.

\[
d[p] = 1 \quad d[q] = 2 \quad d[r] = 3 \quad f[r] = 4 \quad d[t] = 5 \quad d[s] = 6 \quad f[s] = 7 \quad f[t] = 8 \quad f[q] = 9 \quad f[p] = 10 \quad d[u] = 11 \quad f[u] = 12
\]

Next depth-first search is applied to the graph with all edges reversed, starting each
new exploration in the unvisited node with the highest $f$-value.

\[
d[u] = 1 \quad f[u] = 2: \text{SCC} \{u\}
\]

\[
d[p] = 3 \quad d[s] = 4 \quad d[t] = 5 \quad d[q] = 6 \quad f[q] = 7 \quad f[t] = 8 \quad f[s] = 9 \quad f[p] = 10: \text{SCC} \{p, q, s, t\}
\]

\[
d[r] = 11 \quad f[r] = 12: \text{SCC} \{r\}
\]

A possible scenario starting at $r$. First depth-first search is applied to the graph, starting
in $r$.

\[
d[r] = 1 \quad f[r] = 2 \quad d[p] = 3 \quad d[q] = 4 \quad d[t] = 5 \quad d[s] = 6 \quad f[s] = 7 \quad f[t] = 8 \quad f[q] = 9 \quad f[p] = 10 \quad d[u] = 11 \quad f[u] = 12
\]

Next depth-first search is applied to the graph with all edges reversed, starting each new
exploration in the unvisited node with the highest $f$-value. This proceeds in exactly the
same way as the previous application.

A possible scenario starting at $u$. First depth-first search is applied to the graph, starting
in $u$.

\[
d[u] = 1 \quad d[t] = 2 \quad d[s] = 3 \quad d[p] = 4 \quad d[q] = 5 \quad d[r] = 6 \quad f[r] = 7 \quad f[q] = 8 \quad f[p] = 9 \quad f[s] = 10 \quad f[t] = 11 \quad f[u] = 12
\]

Next depth-first search is applied to the graph with all edges reversed, starting each new
exploration in the unvisited node with the highest $f$-value.

\[
d[u] = 1 \quad f[u] = 2: \text{SCC} \{u\}
\]

\[
d[t] = 3 \quad d[q] = 4 \quad d[p] = 5 \quad d[s] = 6 \quad f[s] = 7 \quad f[p] = 8 \quad f[q] = 9 \quad f[t] = 10: \text{SCC} \{p, q, s, t\}
\]

\[
d[r] = 11 \quad f[r] = 12: \text{SCC} \{r\}
\]