The door code problem

Sebastian Björkqvist

May 6, 2014

Abstract

This text considers the problem of finding the shortest possible string of numbers that contains as a substring every possible string of some length k. The problem is represented as a graph problem and it is shown that an optimal solution can be found using a greedy algorithm.

Acknowledgments. I would like to thank Lasse and Jeremias for their help in solving this problem.

Introduction

Consider the following scenario: You are about to enter a building, but the front door requires a code before it may be opened. You know that the required code consists of four digits from 0 to 9. You also know that when entering the code, the door will unlock at any time when the correct code has been entered, even if it is preceded by other digits. You thus wish to find the shortest string possible that contains as a substring every possible four-digit code. Codes may overlap in the string since the lock does not care about preceding digits once the correct code has been entered.

We may of course generalize the problem to any code length and any base, where we by the base refer to the number of different digits the code may use. In this text we denote by b the base and by k the length of the code in the problem; in the above scenario, we have $b = 10$ and $k = 4$. The general door code problem is the following: What is the shortest string that contains all strings of length k in base b as a substring, and how can we create this string?

It is easy to derive a lower bound of $b^k + k - 1$ (Lemma 3) for the length of the shortest string. The big question is if this lower bound can always be reached, and maybe a bit surprisingly it can. This means that we do not need to repeat any string of length k when constructing the string that contains all strings of length k as a substring.

Formal definition of the problem

We formalize the door code problem in the following way:

Definition 1. The number $D(b, k)$ is the length of the shortest string that contains all strings of length $k \in \mathbb{Z}_+$ in base $b \in \mathbb{Z}_+$ as a substring. We call the number k the *code length*.

It is clear that the number $D(b, k)$ is finite and well-defined for all b and k, since we may find the shortest string that meets the requirements by checking strings of base b in lexicographic order (starting from strings of length 1, then length 2 and so on) until we find one that meets the requirements.

The main result is the following:

Theorem 2. *$D(b, k) = b^k + k - 1$ for every base b and code length k.*

Our proof of Theorem 2 is constructive: an algorithm will be presented that constructs the desired string, and it will be shown that the string is of length $b^k + k - 1$.

The optimality of the algorithm to be presented follows from the following lemma, which gives a lower bound for the number $D(b, k)$:

Lemma 3. *$D(b, k) \geq b^k + k - 1$ for every base b and code length k.*

Proof. Let L be a string that contains every string of length k in base b as a substring. There are b^k different strings of length b in base k, and thus, because L contains every one of them as a substring, there must be at least b^k starting points for strings of length k in L, which means the starting point of the last string is at least at index b^k in L (when indexing starts from 1).

A string of length k that starts at some index i will end at index $i + k - 1$. Because the last string starts at index b^k or later, the length of L must be at least $b^k + k - 1$. □

Example. Consider the simple case where $b = 2$ and $k = 3$, i.e. the case where the codes are of length 3 and consist of digits 0 and 1. By Lemma 3, we know that $D(2, 2) \geq 2^3 + 3 - 1 = 10$. We may easily find a string of length 10 that meets the requirements; one possibility is the string 1110001011. It contains all eight strings of length 3 in base 2 as a substring:

- 1110001011
- 1110001011
- 1110001011
- 1110001011
- 1110001011
- 1110001011
- 1110001011
- 1110001011

Thus we've found an optimal solution in this case, and it follows that $D(2, 3) = 10$.

Graph-theoretic view

Instead of considering some large string L that has strings of length k as substrings, we may view the door code problem in a graph-theoretic setting. To accomplish this, we define a graph $G_{b,k}$ where there is a node for every code of length k in base b, and there is a directed edge from node $v = v_1 \cdots v_n$ to node $u = u_1 \cdots u_k$ if and only if the string $u_1 \cdots u_k$ can follow the string $v_1 \cdots v_k$ in L. This holds only if $u_i = v_{i-1}$ for all $i \in \{1, \ldots, k-1\}$; u_k may be any digit $0, 1, \ldots, b-1$.

It follows that every node v in $G_{b,k}$ has b incoming and b outgoing edges, but some nodes (the ones that repeat the same digit k times) have an edge that goes to itself. The graph $G_{b,k}$ is thus a b-regular directed graph that has b^k nodes and $b \cdot b^k = b^{k+1}$ edges. For an example of the graph $G_{b,k}$ when $k = 3$ and $b = 2$, see Figure A.

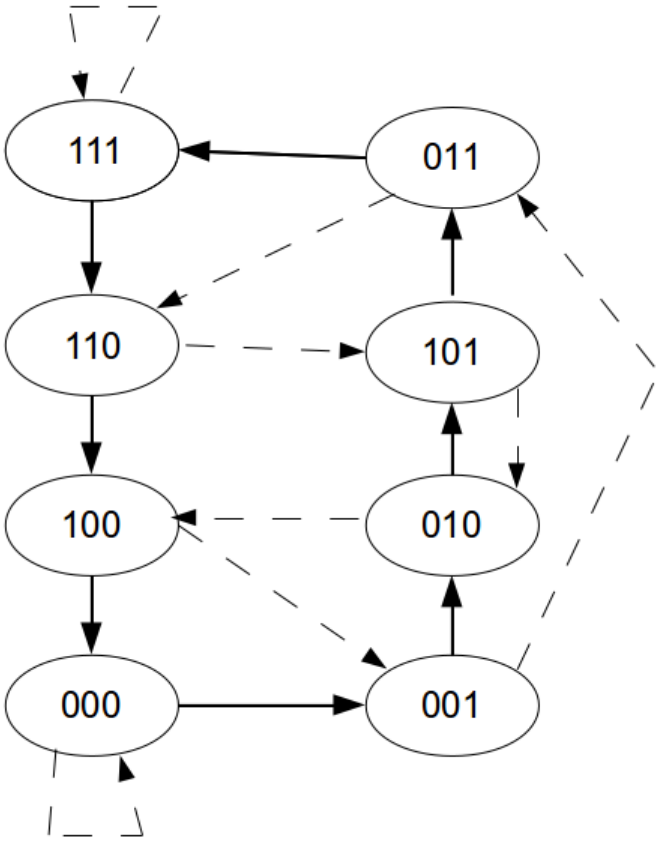

Figure A: The graph $G_{2,3}$ corresponding to code length $k = 3$ and base $b = 2$.

The problem of constructing a string of length $b^k + k - 1$ containing all codes of length k as a substring corresponds to finding a Hamiltonian path in the graph $G_{b,k}$, since for the string to be of optimal length we may not repeat any code, and thus we may not visit any node in $G_{b,k}$ more than one time. For instance, the highlighted edges in Figure A form a Hamiltonian cycle, and we obtain an optimal code 1110001011 by taking all three digits from the node 111 and after that the last digit of every subsequent node in the cycle.

This observation does not help us immediately, since it is well known that the Hamiltonian path problem is NP-complete. On the other hand, it has been shown that almost all regular graphs are Hamiltonian ([1]), which means that we might well find one in this setting also.

The algorithm and proof of the main result

Remark. In this section the base b and code length k is fixed. We also use the notation $(x_1, \ldots, x_k)$ for the string/node $x_1 \cdots x_k$. Here $x_i \in \{0, 1, \ldots, b-1\}$ for all $i \in \{1, \ldots, k\}$.

The algorithm that finds a string of optimal length, or equivalently a Hamiltonian path in the corresponding graph $G_{b,k}$ is simple: Start with the string $(b-1, \ldots, b-1)$, which we denote by s. After that choose at each step the lowest digit possible that does not repeat any string of length k that already is a substring. In the graph $G_{b,k}$ this corresponds to starting from the node $s = (b-1, \ldots, b-1)$ and after that going to the first node (lexicographically) that hasn't already been visited. For instance, the Hamiltonian path in Figure A can be created by the algorithm.

We may now prove Theorem 2 by showing that the algorithm described above will always create a string of length $b^k + k - 1$, i.e. that we may at all times move to a new node until we've visited them all. We begin our analysis by proving the following lemma:

Lemma 4. *Let $x = (x_1, \ldots, x_k)$ be any node other than the starting node s. When the algorithm enters the node x, we have already visited all nodes $x' = (x_1, \ldots, x_k')$, where $x_k' < x_k$.*

Proof. Since x isn't the starting node, we were in some node $y = (y_1, x_1, \ldots, x_{k-1})$ before entering x. Because the algorithm chooses the lowest available node, the lowest available node must have been the node x. Thus we have already visited all nodes $x' = (x_1, \ldots, x_k')$, where $x_k' < x_k$. □

Using the above lemma we may prove the following:

Lemma 5. *If the algorithm has visited the nodes $(i, b-1, \ldots, b-1, j)$ for all digits $i, j \in \{0, 1, \ldots, b-1\}$, it has visited every node in $G_{b,k}$.*

Proof. Let $i \in \{0, 1, \ldots, b-1\}$ be fixed. We consider two cases:

1) $i \neq b-1$: By the assumption, we have visited the nodes $(i, b-1, \ldots, b-1, j)$ for all $j \in \{0, 1, \ldots, b-1\}$. We now make the following observation: we may enter nodes $(i, b-1, b-1, \ldots, j)$ only from nodes $(x_0, i, b-1, \ldots, b-1)$, where $x_0 \in \{0, 1, \ldots, b-1\}$. Because $i \neq b-1$, there is a one-to-one correspondence between the nodes $(x_0, i, b-1, \ldots, b-1)$ and the nodes $(i, b-1, \cdots, b-1, j)$, and we know that none of the nodes $(i, b-1, \ldots, b-1, j)$ are the starting node. Thus we must have visited all nodes $(x_0, i, b-1, \ldots, b-1)$, because otherwise we couldn't have visited all nodes $(i, b-1, \ldots, b-1, j)$! See Figure B for an illustration.

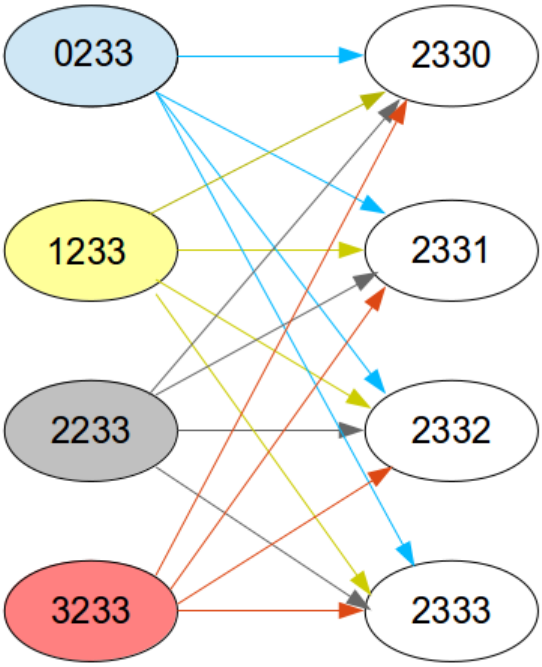

Figure B: An illustration of the above reasoning in the case $k = b = 4$, $i = 2$. If we have visited all the white nodes, we must have visited all the coloured nodes.

Thus we have visited the nodes $(x_0, i, b-1, \ldots, b-1)$ for all $x_0 \in \{0, 1, \ldots, b-1\}$. It follows from Lemma 4 that we've also visited the nodes $(x_0, i, b-1, \ldots, b-1, x_{k-1})$ for all $x_0, x_{k-1} \in \{0, 1, \ldots, b-1\}$. By repeating the argument made above, it follows that we've also visited nodes $(x_{-1}, x_0, i, b-1, \ldots, b-1)$ for all $x_{-1}, x_0 \in \{0, 1, \ldots, b-1\}$, and thus by Lemma 4 the nodes $(x_{-1}, x_0, i, b-1, \ldots, b-1, x_{k-2})$ for all $x_{-1}, x_0, x_{k-2} \in \{0, 1, \ldots, b-1\}$. It follows by induction that we've visited all nodes

$$y = (y_1, \ldots, y_k), \text{where } y_l = i \text{ for some } l \in \{1, \ldots, k\}.$$

2) $i = b - 1$: By going through all values $i < b - 1$ and using the reasoning above, we covered all nodes y where there is at least one index that doesn't contain the digit $b-1$. If all indices have the digit $b-1$, we have the node $s = (b-1, \ldots, b-1)$, which we've visited by the assumption (choose $i = j = b - 1$).

Thus by combining both cases above, we conclude that we've visited all nodes in $G_{b,k}$. □

The main result follows from the following lemma:

Lemma 6. *When the algorithm gets stuck for the first time, i.e. we can't move to any node that hasn't already been visited, we have visited all nodes.*

Proof. Assume the node $x = (x_1, \ldots, x_k)$ is the first one from which we can't move to a node we haven't already visited. This means that the nodes $x' = (x_2, \ldots, x_{k+1})$ have already been visited for all $x_{k+1} \in \{0, 1, \ldots, b-1\}$. As in the proof of Lemma 5, we

know that we can enter nodes $(x_2, \ldots, x_{k+1})$ only from nodes $\bar{x} = (a, x_2, \ldots, x_k)$, where $a \in \{0, 1, \ldots, b-1\}$. Because we've visited all the nodes x', we must also have visited all the nodes $\bar{x}$. There are, in addition to the node x, $b-1$ nodes of the type $(a, x_2, \ldots, x_k)$. Since we've visited each of them only once, they have as a group only $b-1$ successors in the path the algorithm has chosen, and all successors are nodes of type $(x_2, \ldots, x_{k+1})$. On the other hand, since all b nodes of type $(x_2, \ldots, x_{k+1})$ are covered, the only remaining possibility is that we started from one of them, i.e. that $x_2 = \cdots = x_k = b-1$ (see Figure C for an illustration).

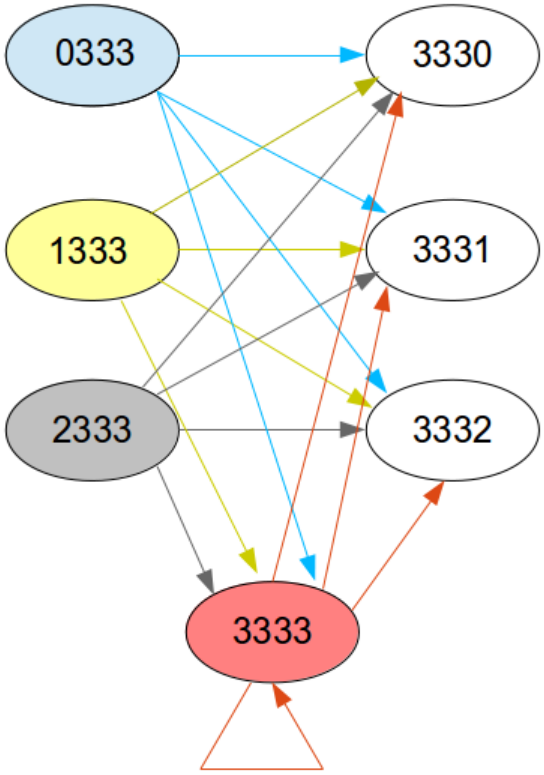

Figure C: An illustration in the case $k = b = 4$. It's impossible for three nodes to have four successors, so one node must be covered by some other way, i.e. by being the starting node.

This means that the node x is of the type $(x_1, b-1, \ldots, b-1, b-1)$ for some $x_1 \in \{0, 1, \ldots, b-1\}$, and we have visited the nodes $(a, b-1, \ldots, b-1, b-1)$ for all $a \in \{0, 1, \ldots, b-1\}$. By Lemma 4, we have visited all nodes $(a, b-1, \ldots, b-1, c)$ for all $a \in \{0, 1, \ldots, b-2\}$, $c \in \{0, 1, \ldots, b-1\}$ (remember that the lemma doesn't cover the case $s = (b-1, \ldots, b-1)$). When we to this add the fact that we've visited the nodes $x' = (b-1, \ldots, b-1, x_{k+1})$ for all $x_{k+1} \in \{0, 1, \ldots, b-1\}$, we have satisfied the condition of Lemma 5. The algorithm has thus visited all the nodes. □

Corollary 7. *The algorithm creates a Hamiltonian path in the graph $G_{b,k}$, and thus also creates a string of length $b^k + k - 1$ that solves the door code problem.*

Proof. Since we have visited all the nodes when we get stuck for the first time, and the algorithm always picks a node that hasn't been visited, the path traversed by the algorithm is a Hamiltonian path. Also, because the node we get stuck at is of the type $(x_1, b-1, \ldots, b-1, b-1)$, we may complete the path to a Hamiltonian cycle by moving to the starting node s.

As previously mentioned, the way to convert the path to a string is to pick the entire string of length k that corresponds to the starting node, and after that pick the last digit from every subsequent node. Since the graph has b^k nodes, we thus get a string of length $b^k + k - 1$. □

We have shown that the algorithm solves the door code problem, and thus proved Theorem 2, i.e. that $D(b, k) = b^k + k - 1$ for all positive integers b and k.

Open problems

We noticed that the algorithm creates an Hamiltonian cycle in the graph $G_{b,k}$, which means that we may actually start the string with any valid code of length k, and after that traverse the cycle until we return to the starting node. Also, when we have found a string that works, we may of course transpose the digits in the string to create a new solution. An open problem is thus the following: Is there for each instance of the door code problem only one solution, i.e. can we from one solution create all the other solutions by either transposing the digits or by rotating the string?

The algorithm gives us an easy way of finding Hamiltonian cycles in certain special types of regular directed graphs. Might there be some way to generalize the algorithm so it could create Hamiltonian cycles in more general types of graphs?

References

[1] Robinson, R.W., Worwald, W.C. *Almost All Regular Graphs are Hamiltonian* `https://www.math.uwaterloo.ca/~nwormald/papers/hamreg.pdf`