Android Forensics : scheme locking mechanism, an algorithmic perspective

Abstract

This paper introduces a very simple recurcive algorithm that counts all possible Android scheme patterns for arbitrary matrice sizes.

Introduction

As the usage of smartphones has risen, the amount of personal/business information they carry is now unprecedented. Email-checking, social networking, messaging, online banking and many-many more things are now common among billions of people around the globe. As a result, phone-locking mechanisms, once very simple, have evolved to accommodate and secure information from unintended eyes. Two main operating systems dominates the market, iOS and Android. Both enable the user to unlock their phone using a shared-secret technique. iOS uses a 4-digit password for which the math are simple and well-known. On the other side, Android phones use an *a priori* more complex pattern-based mechanism that has 9 dots or *node* on the screen organized in a 3×3 matrix. To unlock the phone the user is required to connect nodes together in a specific order. The latter is difficult to model using traditional combinatorics tools because, to be legal, a pattern has to:

- connect at least four nodes;
- the connected nodes must all be distinct;
- if the line segment connecting any two consecutive nodes in the pattern passes through any other nodes, those must have previously been connected.

So how many valid patterns are there? In this paper, we will introduce an algorithm that counts them.

The algorithm

The algorithm consists of a very straightforward recursive function that heuristically discovers all possible paths starting from a given node in a matrix:

1. Mark all nodes as unvisited;
2. Push this node on the stack;
3. For each other nodes in the matrix:
   A. For each crossed nodes on the line segment between this node and the target node, check that it has been previously visited (on the stack), if not goto `step 3`;
   B. Has not been previously visited in the current pattern (not on the stack), if not goto `step 3`;
   C. Has not been previously visited from this node regardless of the current pattern, if not goto `step 3`;
   D. Accumulate to returned value of `step 2` with the target node as current node;
4. If there is more than 4 elements on the stack, increment the number of pattern found;
5. Pop this node from the stack;
6. Return the number of found patterns.

The final result of all possible combinations is obtained by accumulating the results of all nodes in the matrix.

Symmetry

Since the number of possible combinations starting from a node isn't correlated to it's absolute position but rather on a neighbor configuration (flipping a matrix or mirroring it would not change the results), it is possible to use symmetry to fasten the computation. This way, a corner would only be computed onces, etc.

Performance

This algorithm employs a stack as it's main data structure. It's depth is known at compile-time — as it relates to the matrix size. Hence, it can be allocated on the invocation stack, avoiding dynamic allocation and enabling the compiler to pre-compute the offsets of memory jumps.

Reference implementation

We propose a reference implementation in `C++` that is able to discover, list and count all legal patterns for given matrix. While Android proposes only 3x3 matrices, this program is not limited to those and can deal with arbitrary-sized matrices.

Results

Number of nodes/digits	PIN	PATTERN
4	10^4	1624
5	10^5	7152
6	10^6	26016
7	10^7	72912
8	10^8	140704
9	10^9	140704
Σ	**1111110000**	**389112**

As one can see from the above table, PIN locking always offers a better number of possibilities, making it an overall better security mechanism.