Nijenhuis' Algorithm to Calculate Frobenius Number

Michael Angel

April 2019

1 Introduction

The topic of this paper is computations that solve the Frobenius Problem (also known as the *coin change* problem) which can be asked as follows: given integers $a_1, a_2, ..., a_n$ that have greatest common divisor of 1, what is the largest number that cannot be represented as a combination of those numbers? This is the calculation of the **Frobenius number** of set $\{a_1, a_2, ..., a_n\}$. This paper will present an algorithm by Nijenhuis that calculates the Frobenius problem using the idea of a directed graph.

For example, if a video game arcade coin machine only dispenses two types of coins, 25¢, a quarter, and 3¢, a trime (which was minted in the United States until 1873 [1]). Then the machine can give out the amounts 9¢ (three trimes), 34¢ (a quarter and a trime) and 53¢ (two quarters and a trime). But the coin machine cannot give 12¢ nor 26¢. The Frobenius problem asks what is the largest number of cents that the machine cannot give?

If the question is simply asked about two integers, a_1, a_2, then the Frobenius number, $g(a_1, a_2)$, has a simple formula:

$$g(a_1, a_2) = a_1a_2 - a_1 - a_2$$

Thus, in our example, the arcade coin machine can dispense any number of cents higher than 47.

Calculations of the Frobenius number for $\{a_1, a_2, ..., a_n\}$ become more difficult when n is larger than 2 and simple formulas are not available, so we turn to algorithms to calculate the $g(a_1, a_2, ..., a_n)$. A naïve algorithm might simply go about listing every number that is a combination of $a_1, a_2, ..., a_n$ in an attempt to find the highest number that is not a combination. Indeed this algorithm will terminate, specifically when it come across a_i consecutive numbers, where a_i is the minimum of $a_1, a_2, ..., a_n$, and the output will be the maximum number not in the list, but this algorithm suffers from being computationally inefficient.

Relevant to this matter is the computational complexity of algorithms that calculate the Frobenius number. Better algorithms reduce the total number of operations necessary relative to number of the inputs, n.

Ramírez Alfonsín was able to prove that a general algorithm to calculate the Frobenius number, $g(a_1, a_2, ..., a_n)$, is computationally difficult, NP-Hard [Ram96][Ram2005]. The method of proof was to use the Frobenius number algorithm to solve for the Integer Knapsack Problem, a known NP-Hard problem, in polynomial time.

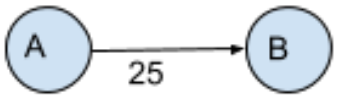
Because the general problem is provably difficult, algorithms of interest involve a lot of computations relative to the number of inputs. Also, algorithms to calculate the Frobenius number quickly for special cases are important, as seen above in the case of two integers. The following section will describe an algorithm that calculates the Frobenius for any integers $a_1, a_2, ..., a_n$ with greatest common divisor 1.

[1] https://www.govmint.com/1851-1873-3-cent-silver-trime-vg
[Ram96] J. Ramírez Alfonsín, Complexity of the Frobenius problem, *Combinatorica* 16, 1996, 143-147.
[Ram2005] J. Ramírez Alfonsín, *The Diophantine Frobenius Problem* Oxford University Press, 2005, 24-28.

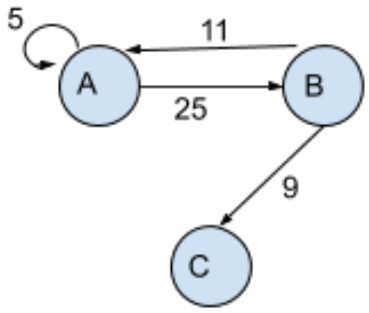
2 Nijenhuis' Algorithm

A **directed graph** is a set of vertices and weighted edges that each connect two vertices (or a vertex to itself) in a direction. The concept of a graph is inherently visual.

Here is an illustration of a graph with two vertices, $\{A, B\}$, and one edge from A to B with weight 25, (A, B, 25).

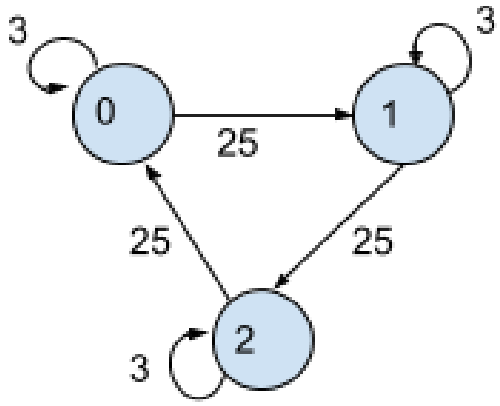


This is a graph with vertices: $\{A, B, C\}$, and edges: $\{(A, A, 5), (A, B, 25), (B, A, 11), (B, C, 9)\}$.

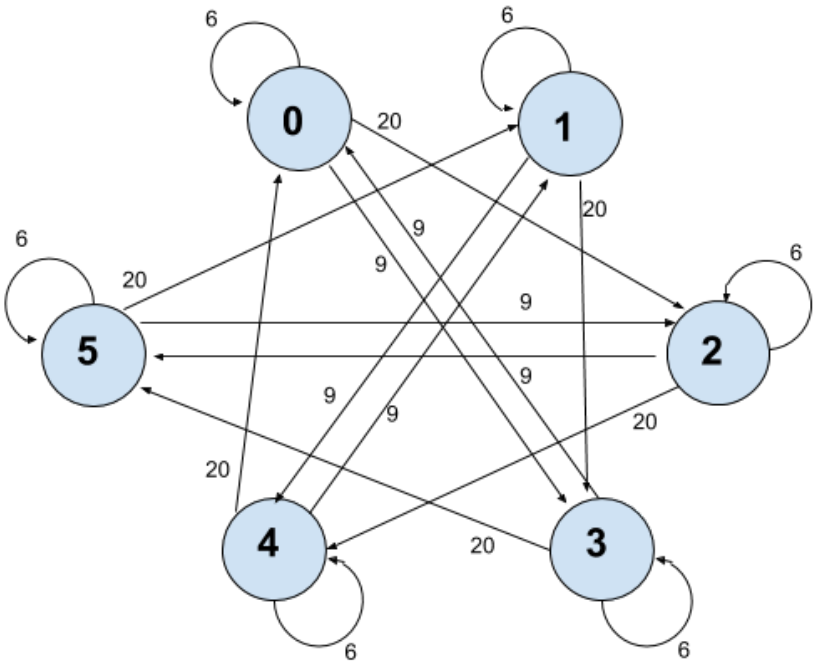


Nijenhuis' Algorithm to calculate $g(a_1, a_2, ..., a_n)$ begins by considering the **Nijenhuis graph** defined as follows:
For natural numbers $a_1, a_2, ..., a_n$, where a_1 is the smallest,
- Vertices: the equivalence classes of the natural numbers modulo a_1, $\{\overline{0}, \overline{1}, ..., \overline{a_1 - 1}\}$.
- Edges: An edge connects two vertices, u to v, if $u + a_i \equiv v \bmod a_1$ for any a_i in $\{a_1, a_2, ..., a_n\}$ and that edge has weight a_i.

For example, consider the arcade coin machine from above. The Nijenhuis graph has vertices: $\{\overline{0}, \overline{1}, \overline{2}\}$ and edges: $\{(\overline{0}, \overline{0}, 3), (\overline{1}, \overline{1}, 3), (\overline{2}, \overline{2}, 3), (\overline{0}, \overline{1}, 25), (\overline{1}, \overline{2}, 25), (\overline{2}, \overline{0}, 25)\}$.



Below is the Nijenhuis graph for $\{6, 9, 20\}$,



A **path** from vertex u to vertex v is a traversal from u to v along the edges and the length of the path is the sum of the weights of the edges.

For example, in the above graph for $\{6, 9, 20\}$, a path from the 0 vertex to the 5 vertex can be through edges $(\overline{0}, \overline{3}, 9)$, and then $(\overline{3}, \overline{5}, 20)$. The length of this path is 29 and it is the shortest path from the 0 vertex to the 5 vertex.

Lemma: Given a Nijenhuis graph for set of integers $\{a_1, a_2, ..., a_n\}$ with greatest common divisor 1 and minimal element a_1, the shortest path from the 0 vertex to the j vertex is the smallest combination of $\{a_1, a_2, ..., a_n\}$ that sums to j mod a_1.

Therefore, given $\{a_1, a_2, ..., a_n\}$ with greatest common divisor 1 and minimal element a_i, we can think of each natural number as belonging to a congruence class modulo a_i. Let w_p be the shortest path in the Nijenhuis graph from the 0 vertex to the pth vertex, which represents the $\overline{p}$ congruence class modulo a_i. Any natural number greater than $\max\limits_{1 \leq p \leq a_i - 1} \{w_p\} - a_i$ can be represented by a combination of $\{a_1, a_2, ..., a_n\}$. Therefore

$$g(a_1, a_2, ..., a_n) = \max_{1 \leq p \leq a_i - 1} \{w_p\} - a_i.$$

Nijenhuis' algorithm to calculate $g(a_1, a_2, ..., a_n)$**:**
1] Create the Nijenhuis graph.
2] Calculate the shortest path from the 0 vertex to each non-zero vertex.
Output the maximum shortest path minus $\min\{a_1, a_2, ..., a_n\}$.

3 Numerical Semigroups

The theory of numerical semigroups provides an alternative way to define these concepts.

A **numerical semigroup**, S, is a subset of the positive integers, Z^+, such that:

1. 0 is in S,
2. $|Z^+ \backslash S|$ is finite,
3. For all s_1, s_2 in S, $s_1 + s_2$ is in S.

A numerical semigroup can be represented with a minimum set of **generators**, $g_1, g_2, ..., g_n$, that serve as a basis such that every element of the semigroup can be represented as a linear combination of $g_1, g_2, ..., g_n$. The greatest common divisor of these generators is 1. $S = \langle g_1, g_2, ..., g_n \rangle$.

The **Frobenius number** of numerical semigroup S is the maximum element of $|Z^+ \backslash S|$, i.e. the largest number that cannot be represented as a linear combination of the generators of S.

For numerical semigroup S with generator m, the **Apéry Set**, $A(m, S)$ of S is $\{s \in S, s - m \notin S\}$.

If m is the smallest generator of numerical semigroup S, then $|A(m, S)| = m$ and each element of $A(m, S)$ is from a different congruence class modulo m. Furthermore, the Frobenius number of S is $\max A(m, S) - m$.

Connection between Apéry set and shortest paths on the Nijenhuis graph:

Given numerical semigroup $S = \langle g_1, g_2, ..., g_n \rangle$ where g_1 is the smallest generator. The shortest paths on the Nijenhuis graph of $\{g_1, g_2, ..., g_n\}$ from the 0 vertex to every other vertex are the elements of $|Ap(g_1, S)|$. Therefore the Frobenius number of $\{g_1, g_2, ..., g_n\}$ is the maximum shortest path minus g_1.

4 Implementation

Let $\{a_1, a_2, ..., a_n\}$ be ordered such that a_1 is minimal.

Step **1**] to create the Nijenhuis is straight forward.
Vertices can be represented as the list of integers from 0 to a_1 - 1.
To compute edges:
For each vertex a_i, where i ranges from 0 to $n-1$,
for each a_j, where j ranges from 1 to n, create edge $(a_i,\ a_i + a_j \bmod a_1,\ a_j)$.

Step **2**] involves finding shortest paths of graphs. This is at the heart of our computation and Dijkstra's algorithm famously can calculate this with good speed [1].

Overall, the number of computations in this algorithm are at most a constant multiple of $na_1 \log a_1$, i.e. $\mathcal{O}(na_1 \log a_1)$. This is because Dijkstra's algorithm has time complexity $\mathcal{O}(n \log a_1)$ and runs a_1 times.

Nijenhuis' Algorithm is a fast algorithm in practice but what makes it especially valuable is the added insight we get from considering the Frobenius problem in terms of shortest paths on graphs. To consider faster implementations we can examine the structure of the graph to specialize our technique.

[1] https://www.geeksforgeeks.org/dijkstras-algorithm-for-adjacency-list-representation-greedy-algo-8/