Sums of Palindromes: An Approach via Automata

Aayush Rajasekaran, Jeffrey Shallit, Tim Smith

University of Waterloo

September 28, 2017

Jon Borwein

Table of Contents

Alphabets and Strings

- An *alphabet*, Σ, is a finite collection of *symbols*
- We are primarily concerned with the *binary alphabet*, $\Sigma_2 = \{0, 1\}$
- A *string* is any finite list of symbols from an alphabet
- Example: $x = 0010101 \in \Sigma_2^*$
- For a positive integer N, we let $(N)_b$ denote the string that is its base-b representation
- Example: $(76)_2 = 1001100$

Definition

- A *palindrome* is any string that is equal to its reverse
- Examples are `RADAR`, `MALAYALAM`, and 10001
- We call an integer a *base-b palindrome* if its base-b representation is a palindrome
- Examples are $16_{10} = 121_3$ and $297_{10} = 100101001_2$
- Binary palindromes ($b = 2$) can be found as sequence A006995 in the *On-Line Encyclopedia of Integer Sequences* (OEIS)

$$0, 1, 3, 5, 7, 9, 15, 17, 21, 27, 31, 33, 45, 51, 63, \ldots$$

The problem

Question: Given a positive integer, N, what is the smallest number of base-b palindromes that add up to N?

Current results

- William Banks (2015) showed that every positive integer is the sum of at most 49 decimal palindromes
- Javier Cilleruelo, Florian Luca and Lewis Baxter (2017) showed that for any base $b \geq 5$, all positive integers are the sum of three base-b palindromes

Our result

We have the following result for the binary case:

Theorem

Every natural number N is the sum of at most 4 binary palindromes. The number 4 is optimal.

To note that 4 is optimal, we observe that 176 is not the sum of 3 binary palindromes (or fewer).

Previous proofs are complicated (1)

Screenshot from Banks (2015)

2.4. **Inductive passage from** $\mathbb{N}_{\ell,k}(5^+;c_1)$ **to** $\mathbb{N}_{\ell-1,k+1}(5^+;c_2)$**.**

LEMMA 2.4. *Let* $\ell, k \in \mathbb{N}$*,* $\ell \geqslant k+6$*, and* $c_\ell \in \mathcal{D}$ *be given. Given* $n \in \mathbb{N}_{\ell,k}(5^+;c_1)$*, one can find digits* $a_1, \ldots, a_{18}, b_1, \ldots, b_{18} \in \mathcal{D}\backslash\{0\}$ *and* $c_2 \in \mathcal{D}$ *such that the number*

$$n - \sum_{j=1}^{18} q_{\ell-1,k}(a_j, b_j)$$

lies in the set $\mathbb{N}_{\ell-1,k+1}(5^+;c_2)$*.*

Proof. Fix $n \in \mathbb{N}_{\ell,k}(5^+;c_1)$, and let $\{\delta_j\}_{j=0}^{\ell-1}$ be defined as in (1.1) (with $L := \ell$). Let m be the three-digit integer formed by the first three digits of n; that is,

$$m := 100\delta_{\ell-1} + 10\delta_{\ell-2} + \delta_{\ell-3}.$$

Clearly, m is an integer in the range $500 \leqslant m \leqslant 999$, and we have

$$n = \sum_{j=k}^{\ell-1} 10^j\delta_j = 10^{\ell-3}m + \sum_{j=k}^{\ell-4} 10^j\delta_j. \tag{2.4}$$

Let us denote

$$\mathcal{S} := \{19, 29, 39, 49, 59\}.$$

In view of the fact that

$$9\mathcal{S} := \underbrace{\mathcal{S} + \cdots + \mathcal{S}}_{\text{nine copies}} = \{171, 181, 191, \ldots, 531\},$$

it is possible to find an element $h \in 9\mathcal{S}$ for which $m - 80 < 2h \leqslant m - 60$. With h fixed, let $s_1, \ldots, s_9$ be elements of $\mathcal{S}$ such that

$$s_1 + \cdots + s_9 = h.$$

Previous proofs are complicated (2)

Screenshot from Cilleruelo et al. (2017)

II.2 $\boldsymbol{c_m = 0}$. We distinguish the following cases:

II.2.i) $\boldsymbol{y_m \neq 0}$.

δ_m	δ_{m-1}
0	0
$*$	y_m
$*$	$*$

$\longrightarrow$

δ_m	δ_{m-1}
1	1
$*$	$y_m - 1$
$*$	$*$

II.2.ii) $\boldsymbol{y_m = 0}$.

II.2.ii.a) $\boldsymbol{y_{m-1} \neq 0}$.

δ_m	δ_{m-1}	δ_{m-2}
0	0	$*$
y_{m-1}	0	y_{m-1}
$*$	z_{m-1}	z_{m-1}

$\longrightarrow$

δ_m	δ_{m-1}	δ_{m-2}
1	1	$*$
$y_{m-1} - 1$	$g - 2$	$y_{m-1} - 1$
$*$	$z_{m-1} + 1$	$z_{m-1} + 1$

The above step is justified for $z_{m-1} \neq g - 1$. But if $z_{m-1} = g - 1$, then $c_{m-1} \geq (y_{m-1} + z_{m-1})/g \geq 1$, so $c_m = (z_{m-1} + c_{m-1})/g = (g - 1 + 1)/g = 1$, a contradiction.

II.2.ii.b) $\boldsymbol{y_{m-1} = 0, z_{m-1} \neq 0}$.

δ_m	δ_{m-1}	δ_{m-2}
0	0	$*$
0	0	0
$*$	z_{m-1}	z_{m-1}

$\longrightarrow$

δ_m	δ_{m-1}	δ_{m-2}
0	0	$*$
1	1	1
$*$	$z_{m-1} - 1$	$z_{m-1} - 1$

II.2.ii.c) $\boldsymbol{y_{m-1} = 0, z_{m-1} = 0}$.

If also $c_{m-1} = 0$, then $\delta_{m-1} = 0$, which is not allowed. Thus, $c_{m-1} = 1$.

Previous proofs are complicated (3)

- These proofs are highly case-based
- Difficult to establish, difficult to understand
- Difficult to check too: The original Cilleruelo et al. proof had some minor flaws that were only noticed when the proof was implemented as a `Python` program
- Idea! Could we automate such proofs?

The idea

- Build a finite-state machine that takes natural numbers as input, expressed in the desired base
- Allow the machine to "guess" representations of the input as a sum of palindromes
- The machine should accept an input if it verifies a guess, i.e., if the input has a valid representation as a sum of palindromes
- Use decidability algorithms to establish properties about the language of binary representations accepted by this machine

Introduction to VPAs

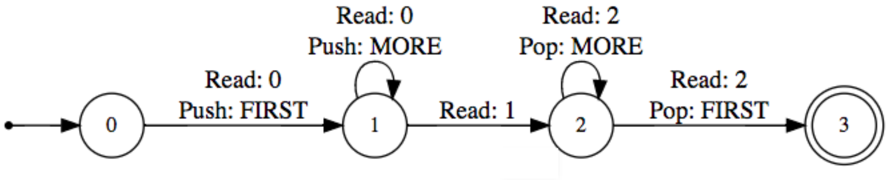
- Visibly-pushdown automata
- Popularized by Alur and Madhusudan in 2004, though similar ideas have been around for longer
- VPAs receive an input string, and read the string one letter at a time
- They have a (finite) set of states and a stack
- Upon reading a letter of the input string, the VPA can transition to a new state, and might modify the stack
- The states of the VPA are either *accepting* or *non-accepting*
- If the VPA can end up in an accepting state after it is done reading the input, then the VPA "accepts" the input, else it "rejects" it

Using the VPA's stack

- The VPA can only take very specific stack actions
- The input alphabet, Σ, is partitioned into three disjoint sets
  - Σ_c, the push alphabet
  - Σ_l, the local alphabet
  - Σ_r, the pop alphabet
- If the letter of the input string we read is from the push alphabet, the VPA pushes *exactly* one symbol onto its stack
- If the letter of the input string we read is from the pop alphabet, the VPA pops *exactly* one symbol off its stack
- If the letter of the input string we read is from the local alphabet, the VPA does not look at its stack at all

Example VPA

Here's a VPA for the language $\{0^n 1 2^n \ : \ n \geq 1\}$



The push alphabet is $\{0\}$, the local alphabet is $\{1\}$, and the pop alphabet is $\{2\}$.

Determinisation and Decidability

- A nondeterministic VPA can have several matching transition rules for a single input letter
- Nondeterministic VPAs are as powerful as deterministic VPAs
- VPAs are closed under union, intersection and complement. There are algorithms for all these operations.
- Testing emptiness, universality and language inclusion are decidable problems for VPAs
- But a nondeterministic VPA with n states can have as many as $2^{\Theta(n^2)}$ states when determinized!

Example

- As an example, let us see how a VPA could verify that 2189 is the sum of 7 binary palindromes of length 9
- We are using 7 summands for this example because the machine is easier to understand when all summands are of the same length
- The machine for the optimal bound of 4 is very similar, but needs a lot more states

The machine

- Note that guessing 7 palindromes over the alphabet $\{0,1\}$ is equivalent to guessing one palindrome over the alphabet $\{0, 1, \ldots 7\}$
- The first symbol guessed must be 7, because all our palindromes must start (and end) with 1
- The idea is to nondeterministically guess every possible combination of 7 palindromes that could sum to 2189
- We push our guesses onto the stack, and use the states to keep track of the carry

The input string

- We set $\Sigma_c = \{a, b\}$, $\Sigma_l = \{c, d\}$, and $\Sigma_r = \{e, f\}$
- The symbols a, c, and e correspond to 0, while b, d, and f correspond to 1.
- We feed the input string to the machine starting with the least-significant digit
- The first half of the input string is composed of symbols from Σ_c, while the second half is composed of symbols from Σ_r
- We use the local alphabet for the last 3 symbols and for the "middle" letter
- $(2189_2) = 100010001101$
  $(2189_2)^R = 101100010001$
  $x = babbceefeccd$

Step 0

We start out with an empty stack, and a carry of 0

Remaining input : *babbceefeccd*

Current state: 0

Stack: EMPTY

Step 1

Current state: 0

Stack: EMPTY

On reading the b, we must guess 7 because the most-significant digit of our summands must be 1

Remaining input : *abbceefeccd*

New state: 3

New stack: EMPTY, 7

Step 2

Current state: 3

Stack: EMPTY, 7

On reading the *a*, we can guess 1 since $1 + 3$ produces an output bit of 0

Remaining input : *bbceefeccd*

New state: 2

New stack: EMPTY, 7, 1

Step 3

Current state: 2

Stack: EMPTY, 7, 1

On reading the b, we can guess 3 since $3 + 2$ produces an output bit of 1

Remaining input : *bceefeccd*

New state: 2

New stack: EMPTY, 7, 1, 3

Step 4

Current state: 2

Stack: EMPTY, 7, 1, 3

On reading the b, we can guess 1 since $1 + 2$ produces an output bit of 1

Remaining input : *ceefeccd*

New state: 1

New stack: EMPTY, 7, 1, 3, 1

Step 5

Current state: 1

Stack: EMPTY, 7, 1, 3, 1

On reading the c, we can guess 1 since $1 + 1$ produces an output bit of 0. This is not pushed onto the stack since it is a local transition.

Remaining input : *eefeccd*

New state: 1

New stack: EMPTY, 7, 1, 3, 1

Step 6

Current state: 1

Stack: EMPTY, 7, 1, 3, 1

On reading the *e*, we pop 1 off the stack and verify that $1 + 1$ produces an output bit of 0

Remaining input : *efeccd*

New state: 1

New stack: EMPTY, 7, 1, 3

Step 7

Current state: 1

Stack: EMPTY, 7, 1, 3

On reading the *e*, we pop 3 off the stack and verify that $3 + 1$ produces an output bit of 0

Remaining input : *feccd*

New state: 2

New stack: EMPTY, 7, 1

Step 8

Current state: 2

Stack: EMPTY, 7, 1

On reading the f, we pop 1 off the stack and verify that $1 + 2$ produces an output bit of 1

Remaining input : *eccd*

New state: 1

New stack: EMPTY, 7

Step 9

Current state: 1

Stack: EMPTY, 7

On reading the e, we pop 7 off the stack and verify that $7 + 1$ produces an output bit of 0

Remaining input : ccd

New state: 4

New stack: EMPTY

Step 10

- We next read *ccd* which corresponds to 4, which is exactly the carry we have, so the VPA accepts the input string
- Note that our guessed palindrome is 713111317. We have:
  $111\ 111\ 111 = 511$
  $101\ 000\ 101 = 325$
  $101\ 000\ 101 = 325$
  $100\ 000\ 001 = 257$
  $100\ 000\ 001 = 257$
  $100\ 000\ 001 = 257$
  $100\ 000\ 001 = 257$
- And, we have $511 + 325 \cdot 2 + 257 \cdot 4 = 2189$

Proof Strategy

- To prove our result, we built 2 VPAs:
  - For all n, A accepts all n-bit odd integers that are the sum of three palindromes whose lengths are either $n-1$, $n-2$, $n-3$ or $n-4$
  - B accepts all valid representations of odd integers over the alphabet $\{a, b, c, d, e, f\}$
- We then prove that all inputs accepted by B are accepted by A
- To prove that we use the ULTIMATE Automata Library
- We simply have to say `assert(IsIncluded(B, A))` in ULTIMATE

Results

For bases 3 and 4, we establish a bound of 3 for every sufficiently large positive integer.

Theorem

Every natural number $N > 256$ *is the sum of at most three base-3 palindromes.*

Theorem

Every sufficiently large natural number $N > 64$ *is the sum of at most three base-4 palindromes.*

Proof details

- The VPAs for bases 3 and 4 get very large, and the `IsIncluded` assertion times out in ULTIMATE
- We use NFAs to prove our results for these bases
- This poses a challenge: NFAs do not have a stack, which is what made VPAs good for this problem
- The solution is to "fold" our input and provide the machine with two bits of input at the same time
- For example, if our input is 12011210, then we give the machine $[1,0][2,1][0,2][1,1]$ as its input

Definition

- A *square* is any string that is some smaller string repeated twice
- Examples are `TARTAR`, `PATPAT`, and 100100
- We call an integer a *base-b square* if its base-b representation is a square
- Examples are $36_{10} = 100100_2$ and $3_{10} = 11_2$.
- Binary squares ($b = 2$) can be found as sequence A020330 in the OEIS

$$3, 10, 15, 36, 45, 54, 63, 136, 153, 170, 187, 204, 221, \ldots$$

Results

In an analogy of Lagrange's 4-square theorem, we have

Theorem

Every natural number $N > 686$ is the sum of at most 4 binary squares.

We also have the following result

Theorem

Every natural number is the sum of at most two binary squares and at most two powers of 2.

Acknowledgments

- Matthias Heizmann, Alexander Nutz, and Christian Schilling, the developers of ULTIMATE
- Dirk Nowotka and Parthasarathy Madhusudan, for independently suggesting the “folding” trick which lets us use NFAs
- Rein Otsason, for his help with the research and assistance in preparing these slides