Journal of High School Science

Using impartial games and greedy algorithms to generate codes with fixed distance

Yin B[1], Aydin N[2]

Submitted: September 11, 2025, Revised: version 1, October 27, 2025, version 2, October 30, 2025
Accepted: November 7, 2025

Abstract

B−greedy codes are codes with a fixed distance, defined by a greedy algorithm. Despite simple construction, *B*−greedy codes possess many properties that are useful in the study of coding theory. A subset of these codes, known as lexicographic codes, or lexicodes for short, has connections with impartial games, making them even more fruitful to study. By studying impartial games, we gain access to additional tools that can be used to study lexicodes, as winning positions in impartial games are related to codewords in lexicodes. This paper aims to study lexicodes in the context of Sprague-Grundy analysis and highlight the connection between winning strategies and greedy algorithms. We will also present a proof of the Gilbert-Varshamov bound using *B*−greedy codes.

Keywords

Lexicographic codes, Greedy codes, Impartial games, Gilbert-Varshamov bound, Sprague-Grundy analysis, Greedy algorithm

[1]Benjamin Yin, Pioneer Academics Research Program[1], 101 Greenwood Ave, Jenkintown, PA 19046, USA; Lexington High School[2], 251 Waltham St, Lexington, MA 02421, USA. benyin0828@gmail.com
[2]Corresponding author: Noah Aydin, Department of Mathematics, Kenyon College, 106 College Park Dr, Gambier, OH 43022, USA. aydinn@kenyon.edu

https://doi.org/10.64336/001c.155366

1 Introduction

One of the biggest challenges when sending information over a communication channel is alteration of data. When sending messages over a noisy channel, we wish to be able to recover the original message, even when alterations occur. Messages are usually sent in binary and are composed of words of certain lengths. A code is a set of words that can be used for messages. One of the biggest goals is to find codes that can resist changes, thus the study of coding theory.

In coding theory, one of the largest questions is how to construct codes with high distance. Although there are many algorithms that construct codes of a fixed distance, this paper will be concerned with codes constructed using a greedy algorithm, hence the name ”greedy codes”. In fact, many well-known codes like the binary Golay and Hamming code are greedy codes. In this paper, we will analyze how greedy codes can be created through impartial games like Kayles and Grundy’s Game.

Impartial games are two-player games in which one cannot tell which player is which, i.e., the options on each turn for both players are identical. The two players alternate taking turns until one player has no more legal moves. The first player with no remaining moves is the loser.

To best understand the impartial games of Kayles and Grundy’s Game, we first review a keystone game for Sprague-Grundy analysis, the game of nim.

In the game of nim, we start with any number of piles of beans. Each pile is called a *heap*. During each player’s turn, a valid move involves removing a nonnegative integer number of beans from a single heap. The game concludes when no legal moves remain. The player who makes the last move is the winner.

Nim is crucial to our understanding of impartial games, as the Sprague-Grundy Theorem uncovers the key to connecting all impartial games.

In the beginning parts of this paper, we will analyze a special case of the nim game and the corresponding Sprague-Grundy theorem.

In the next parts of this paper, we will highlight the significance of such impartial games in relation to our study of greedy codes. In particular, the analysis of winning positions in impartial games is related to greedy codes known as lexicographic codes, or lexicodes for short. This paper will aim to construct a certain class of greedy codes and later use them to prove many results, including the Gilbert-Varshamov Bound.

2 Results on impartial games

As mentioned in the introduction, all impartial games can be related back to the game of nim using the Sprague-Grundy Theorem (S-G Theorem). We first define the concept of a position, and then restate the S-G Theorem for clarity.

2.1 Definition 2.1
For an impartial game, a *position* is a specific state of the game at any fixed moment.

2.2 Sprague-Grundy Theorem (1)
Every position may be represented by a corresponding one-heap nim game.

Proof: This proof will be omitted, but a proof can be found in (2), while the foundational papers for the S-G Theorem can be found in (1) and (3).

The S-G Theorem allows us to describe positions in impartial games as a one-heap game of nim. We are thus inspired to define the *Grundy value*, or G-value for short, of an impartial game position to be the corresponding heap size of the nim game. The G-value will allow us to generalize winning and losing positions in any impartial game and will eventually aid us in selecting words for greedy codes. We now outline a few rules tying impartial games to the concept of G-values.

2.3 Definition 2.2
A *move* is an action taken by a player that starts at one position and ends at another.

2.4 Definition 2.3
A *winning position* is a position where the second player can always secure a victory, regardless of the first player's actions. In this context, the first player moves from the initial position, and the second player responds immediately after.

2.5 Properties of impartial games

1. The zero position has a G-value of 0.

2. From a position with G-value 0, there exists no legal move to another position of G-value 0.

3. From a position with G-value not equal to 0, there is always a move to a position with G-value 0.

4. In a winning position, the second player may win by always moving to positions of G-value 0.

The S-G Theorem provides an unobvious way of assigning G-values to positions in impartial games. At this point, the reader can interpret the G-value to be a value assigned to every position in an impartial game. In the next parts of this section we define the G-value in an alternate way through means of the *mex rule*. To be able to discuss the alternate definition of G-value, we must review the game of nim and introduce the nim-sum.

Pless (4) describes pile sizes in a nim game as binary vectors. The size of each pile in nim will be a nonnegative integer. We can express this integer in binary, and its digits in binary will form its binary vector. For example, $10 = 1010_2$, so

$$10 = (1, 0, 1, 0)$$

$$9 \oplus 10 = (1, 0, 0, 1) + (1, 0, 1, 0) = (0, 0, 1, 1) = 3$$

We define the *nim-sum* (denoted $\oplus$) of two nonnegative integers to be the binary addition of their binary vectors. An example of this definition can be found above. The following theorem is well-known in game theory, and the proof will be left as an exercise to those who are curious.

2.6 Theorem 2.1 (5)
In a winning position of nim, the nim-sum of all the pile sizes is equal to 0. Now, we present two rules used to compute G-values of arbitrary positions. For notation purposes, we denote $G(P)$ as the G-value of a position P.

2.7 Definition 2.4
The *minimal excluded element*, also known as *mex*, of a set is defined as the least nonnegative integer not present in the set. For instance, $\text{mex}\{2, 3, 5, 0, 1, 8\} = 4$ because the integers 0, 1, 2, and 3 are in the set but 4 is not.

2.8 Mex Rule (4)(6)
From a given position P, let $a, b, c, \ldots$ be the set of positions P can get to after one move. Then,

$$G(P) = \text{mex}\{G(a), G(b), G(c), \ldots\}$$

Now, we present a result on nim relating to the mex rule.

2.9 Lemma 2.2
In nim, the G-value of a single heap is equal to the number of coins it has, that is, $G(P_n) = n$.

Proof: Let P_i represent the position with a single heap of size n. We will perform induction on the size of the single heap.

Claim: $G(P_i) = i$.

Base Case: When $i = 0$, we have $G(P_0) = 0$, which is true by the definition of G-values (See *Properties of impartial games* for a reminder).

Induction Hypothesis: Assume for a nonnegative integer k that for all $i \leq k$, $G(P_i) = i$.

Inductive Step: Starting from P_{k+1}, the only legal move is to remove some number of coins from the single heap. Thus, from P_{k+1}, after one single move, we can only reach the positions P_0, $P_1, \ldots, P_k$. So, by the mex rule, we have

$$\begin{aligned} G(P_{k+1}) &= \text{mex}\{G(P_0), G(P_1), \ldots, G(P_k)\} \\ &= \text{mex}\{0, 1, 2, \ldots, k\} \\ &= k + 1 \end{aligned}$$

Thus, the G-value of P_{k+1} is $k + 1$, and our induction is complete. We now present a second way to find G-values for arbitrary positions.

2.10 Decomposition Rule
(6) If a position P can be expressed as $P = P_1 + P_2 + P_3 + \ldots$, where $P_1, P_2, P_3, \ldots$ represent the individual heaps in P, then

$$G(P) = G(P_1) \oplus G(P_2) \oplus G(P_3) \oplus \ldots$$

The term "Decomposition Rule" is not used in any well-known source but will be used in this paper for ease of reference. As an example, we will illustrate how the Decomposition Rule can be used to calculate G-values in nim. In the game of nim, let P be the position with pile sizes of 2, 3, and 5. Let P_i be the position that is one pile with i coins. Then, by the Decomposition Rule, we have

$$G(P) = G(P_2) \oplus G(P_3) \oplus G(P_5)$$

$$G(P) = G(P_2) \oplus G(P_3) \oplus G(P_5) = 2 \oplus 3 \oplus 5 = 4$$

By Lemma 2.2, we know that $G(P_i) = i$, so, this simplifies to the definition above.

We will now introduce two impartial games, Kayles and Grundy's Game, and give G-values for select positions in these games.

2.11 Kayles

Pless introduces and briefly analyzes the game of Kayles (4). The game of Kayles is played with two players, and starts with a line of ten-pins. During their turn, a player may choose to either knock down one ten-pin or a pair of neighboring ten-pins. Victory is awarded to the player able to knock down the last ten-pin.

Let R_i denote the position consisting of a single row of i consecutive ten-pins. If we have a row of zero ten-pins, by definition, the G-value is 0, so $G(R_0) = 0$. For a row of one ten-pin, the only option is to knock it down. So, by mex rule we have

$$G(R_1) = \text{mex}\{0\} = 1$$

$$G(R_2) = \text{mex}\{0, G(1)\} = \text{mex}\{0, 1\} = 2$$

For a row of two ten-pins, we can either knock down one ten-pin leaving a row of one ten-pin, or we can knock down both ten-pins. So, by mex rule we have the definition above.

For a row of three ten-pins, we can knock down one ten-pin, either leaving a row of two ten-pins or two non-adjacent ten-pins. Alternatively, we could knock down two ten-pins, leaving a row of 1 ten-pin. So, by the mex rule we have

$$\begin{aligned} G(R_3) &= \text{mex}\{G(R_2),\ G(R_1) \oplus G(R_1),\ G(R_1)\} \\ &= \text{mex}\{2, 0, 1\} \\ &= 3 \end{aligned}$$

Note: $G(R_1 + R_1) = G(R_1) \oplus G(R_1)$ by the Decomposition Rule.

Additionally, we can compute a few more G-values. For simplicity, let us denote $G(R_i)$ as (i):

$$\begin{aligned} (4) &= \text{mex}\{(3), (2) \oplus (1), (2), (1) \oplus (1)\} \\ &= \text{mex}\{3, 3, 2, 0\} = 1, \\ (5) &= \text{mex}\{(4), (3) \oplus (1), (2) \oplus (2), (3), \\ &\quad (1) \oplus (2)\} \\ &= \text{mex}\{1, 2, 0, 3, 3\} = 4, \\ (6) &= \text{mex}\{(5), (4) \oplus (1), (3) \oplus (2), (4), \\ &\quad (1) \oplus (3), (2) \oplus (2)\} \\ &= \text{mex}\{4, 0, 1, 1, 2, 0\} = 3 \end{aligned}$$

Table 1. The first few G-values for a single row in Kayles

i	$G(R_i)$
0	0
1	1
2	2
3	3
4	1
5	4
6	3

In the next section we discuss a game that, similar to nim, contains heaps of beans and will thus result in fruitful analysis powered by the S-G Theorem.

3 Grundy's Game

As described in (4, 6), Grundy's Game is a two-player game played on a set of bean heaps. On their turn, a player can divide a single heap into two smaller heaps, if the sizes are different. The game concludes once a player is unable to play a valid move, resulting in their loss.

Recall that from the Decomposition Rule, we can determine the G-value of a general position in Grundy's Game by decomposing it into its individual heaps. For example, we would have $G(P_3 + P_4 + P_6) = G(P_3) \oplus G(P_4) \oplus G(P_6)$. So, it suffices to find the G-values of single heaps to determine the G-value of any position. As used in (6), we refer to positions with only a single heap as *atomic positions*. As shown in table 2, (6) gives the G-values of some atomic positions.

Table 2. G-values for atomic positions of size n

n	1	2	3	4	5	6	7	8	9	10
$G(n)$	0	0	1	0	2	1	0	2	1	0

$$P = c_1P_1 + c_2P_2 + c_3P_3 + \dots$$

Now, we will move on to describing the relationship between Grundy's Game and Greedy Codes. By considering an arbitrary position P in Grundy's Game, we can reach the relationship above, where we have c_1 heaps of 1 bean, c_2 heaps of 2 beans, c_3 heaps of 3 beans, and so on. Using the decomposition rule, we have

$$G(P) = \underbrace{G(P1) \oplus \dots \oplus G(P1)}_{c_1\ terms} \oplus \underbrace{G(P2) \oplus \dots \oplus G(P2)}_{c_2\ terms} \oplus \dots$$

Now, since $x \oplus x = 0$, the expression $G(C_iP_i) = \underbrace{G(P_i) \oplus \ldots \oplus G(P_i)}_{c_i \, terms}$ can be simplified to either 0 if c_i is even or $G(P_i)$ if c_i is odd. So, only the parities of the c_i's are taken into account when evaluating $G(P)$.

Consider all words of the form $\ldots c_3c_2c_1$ such that $c_1, c_2, c_3, \cdots \in \{0, 1\}$ and the position $P = c_1P_1 + c_2P_2 + c_3P_3$ is a winning position for the second player. These codewords comprise the *winning code* for Grundy's Game. A codeword is part of the winning code when the G-value of its corresponding position is 0.

A useful table from (6) displays the first few codewords in the winning code.

Table 3. First few codewords in the Winning Code Along with Heap Sizes Source: (6)

Codeword	Heap Sizes
$\ldots$0000000	0
$\ldots$0000001	1
$\ldots$0000010	2
$\ldots$0000011	3
$\ldots$0001000	4
$\ldots$0001001	4,1
$\ldots$0001010	4,2
$\ldots$0001011	4,2,1
$\ldots$0100100	6,3
$\ldots$1000000	7

A challenge that makes the winning code difficult to use is that each codeword has an infinite length. We may define a winning code for a fixed length n by taking the rightmost n components from each codeword, where we get rid of all codewords that have their first nonzero component after the nth component counting from the right. An astute reader may already notice that the winning code appears to be linear. This observation turns out to be true not only for Grundy's Game, but for impartial games in general. The following theorem is stated in (6) without a formal proof, but we present one below.

3.1 Theorem 3.1 (6)
The winning code of length n for all $n \in Z^+$ for any impartial game is linear in F_2.

Proof: Let the winning code of length n be denoted W_n. Consider any two codewords C_1, C_2 in W_n. Let $C_1 = a_n \ldots a_3a_2a_1$, $C_2 = b_n \ldots b_3b_2b_1$, with $a_i, b_i \in \{0, 1\}$ and the corresponding positions $Q_1 = a_1P_1 + a_2P_2 + \cdots + a_nP_n$, $Q_2 = b_1P_1 + b_2P_2 + \cdots + b_nP_n$ respectively, where P_i is the atomic position with heap size i. Since C_1 and C_2 are in W_n, the conditions

$$G(Q_1) = G(a_1P_1) \oplus \cdots \oplus G(a_nP_n) = 0,$$

$$G(Q_2) = G(b_1P_1) \oplus \cdots \oplus G(b_nP_n) = 0,$$

are true.

Now, consider the nim-sum $C = C_1 \oplus C_2$. The components of C will be $a_n \oplus b_n, \ldots a_2 \oplus b_2, a_1 \oplus b_1$. Let Q be the corresponding position of C. Then we have

$$G(Q) = G((a_1 \oplus b_1)P_1) \oplus G((a_2 \oplus b_2)P_2) \oplus \\ \oplus \ldots G((a_n \oplus b_n)P_n)$$

Since $a_i, b_i \in \{0, 1\}$, one can quickly verify that $G((a_i \oplus b_i)P_i) = G(a_iP_i) \oplus G(b_iP_i)$. So, we have

$$G(Q) = G(a_1P_1) \oplus G(b_1P_1) \oplus \ldots G(a_nP_n) \oplus G(b_nP_n)$$

Since nim-sum is commutative, we can rearrange to get

$$\begin{aligned} G(Q) &= [G(a_1P_1) \oplus \cdots \oplus G(a_nP_n)] \\ &\quad \oplus [G(b_1P_1) \oplus \cdots \oplus G(b_nP_n)] \\ &= G(Q_1) \oplus G(Q_2) \\ &= 0 \oplus 0 \\ &= 0 \end{aligned}$$

So, since $G(Q) = 0$, we know $C = C_1 \oplus C_2$ is in W_n. Therefore, since the sum of any two elements of W_n is in W_n, and the zero word is always in the winning code of a fixed length, W_n is linear over F_2.

We have already seen that each position in Grundy's Game can be viewed as a binary number, so for any position P in Grundy's Game, we will discuss P as a position and a number interchangeably.

A position P in Grundy's Game can be expressed in the form $P = c_1P_1 + c_2P_2 + \ldots$, and more generally, for an impartial game, a position P can be expressed as $P = P_a + P_b + P_c + \ldots$, where $a, b, c, \ldots$ are not necessarily distinct. (6) defines a *turning set* to be a set

$$\{h, i, j, \ldots\} \qquad \text{with } h > i > j > \ldots$$

such that replacing P_h with $P_i + P_j + \ldots$ in the expression of P is a legal move. In the case of Grundy's Game, the set $\{5, 3, 2\}$ would be an example of a turning set because a legal move would be to replace P_5 with $P_3 + P_2$ (splitting a heap of 5 into a heap of 3 and a heap of 2).

Turning sets will be useful in our next discussion of winning codes and lexicodes.

Let $P = c_1P_1 + c_2P_2 + c_3P_3 + \ldots, P' = c'P_1 + c'P_2 + c'P_3 + \ldots$ for positions P and P' in Grundy's Game. From the rules of Grundy's Game, we can note that a move from P to P' is legal if (6):

1) $P' < P$ (positions are also binary numbers)
2) The set of i such that $c_i \neq c_i'$ is a turning set

In simpler terms, condition 1) states that we must move towards a ”smaller” position in order to guarantee that the game ends, and condition 2) states that the move must be one of the possible legal moves we defined when introducing Grundy's Game.

Now, we are fully equipped to construct the *lexicode*. The binary *lexicode* of length n for an impartial game is constructed using the following greedy algorithm (6):

3.2 Algorithm 3.1

1. Let L be the empty set
2. Add the zero word of length n to L
3. Consider the nonzero words of length n in lexicographic order (the binary numbers are in lexicographic order if when converting back to decimal, we get the order $1, 2, 3, \ldots$)
4. Add a word/position P to L if for every codeword P' already in L, the set of i with $c_i \neq c_i'$, denote this $\Delta(P, P')$, is **not** a turning set (in other words, the move from P to P' is not legal)
5. The resulting L is the lexicode

A key result that connects greedy algorithms to impartial games is that the lexicode is the same as the winning code for any impartial game, where the winning code can be generalized to represent all winning positions in an impartial game. (6) presents a more general version of this next theorem, but we state a version (Theorem 3.3) that will be sufficient for this paper.

3.3 Theorem 3.2 (6)

From our inductive step, we conclude that a position is only included in the lexicode only if it is a winning position. So, the lexicode consists of the winning positions of the impartial game and is therefore the winning code.

The winning moves in an impartial game are to the codewords in the lexicode.

Proof: We will perform a proof by induction. We induct on the position P for an impartial game. So far, we have only discussed positions in Grundy's Game. However, we can generalize what we know about positions in Grundy's Game to impartial games. Every position P in an impartial game may be written as $P = P_a + P_b + P_c + \ldots$ for some not necessarily distinct $a, b, c, \ldots$. Recall that the mex rule and decomposition rule still work for generalized impartial games.

Claim: Codewords in the lexicode correspond to winning positions, and winning moves are to codewords.

Base Case: When $P = 0$, P is included in the lexicode, and is trivially a winning position.

Induction Hypothesis: For some N, all positions $N' < N$ that correspond to winning positions are in the lexicode.

Inductive Step: There are two cases, N is either included or not included in the lexicode.

If N is included in the lexicode, then by our greedy algorithm, $\Delta(N, N')$ is not a turning set for all $N' < N$, where N' is in the lexicode. So, since there are no moves from N to a winning position, N must be a winning position.

If N is not included in the lexicode, then by our greedy algorithm, there exists some $N' < N$ where N' is in the lexicode such that $\Delta(N, N')$ is a turning set. So, since there exists a move from N to a winning position, N cannot be a winning position. The first player can move from N to a winning position, and thus the second player cannot win, making N not a winning position.

3.4 Theorem 3.3 (6)
For any impartial game where the winning code and the lexicode are in binary, the winning code is equal to the lexicode.

Proof: For an impartial game, by definition, the winning moves are to winning positions. By Theorem 3.2, the winning moves are to the codewords in the lexicode, which means the winning positions are the codewords in the lexicode. Thus, the winning code is equal to the lexicode.

From Theorems 3.1 and 3.3, we arrive at a particularly beautiful result.

3.5 Theorem 3.4
Binary lexicodes are linear.

Proof: From Theorem 3.3, the lexicode is equal to the winning code. From Theorem 3.1, the winning code is linear in F_2, so, the lexicode is also linear in F_2.

4 *B*-greedy Codes

We will begin by introducing a broader class of codes known as *B*−greedy codes to which lexicodes are a part of. Using *B*−greedy codes we will then have an alternate way of generating lexicodes.

Let $B = \{y_1, y_2, \ldots, y_n\}$ be an ordered basis of F_2^n. (7) describes a method for generating the corresponding *B*−order of F_2^n. We describe it in the same fashion, with more detail added for clarity.

Let $O_1 = 0, y_1$. Then, to construct O_i, we will append the words $O_{i-1} \oplus y_i$ to O_{i-1}, where $O_{i-1} \oplus y_i$ is defined in a similar way to when we defined $L_2^n \oplus w$. That is,

$$O_i = O_{i-1}, O_{i-1} \oplus y_i$$

Below, we show a couple of examples of how to construct the O_i's.

$$\begin{aligned}
O_1 &= 0, y_1 \\
O_2 &= 0, y_1, y_2, y_1 \oplus y_2 \\
O_3 &= 0, y_1, y_2, y_1 \oplus y_2, y_3, y_1 \oplus y_3, y_2 \oplus y_3, \\
&\quad y_1 \oplus y_2 \oplus y_3 \\
O_4 &= 0, y_1, y_2, y_1 \oplus y_2, y_3, y_1 \oplus y_3, y_2 \oplus y_3, \\
&\quad y_1 \oplus y_2 \oplus y_3, y_4, y_1 \oplus y_4, y_2 \oplus y_4, \\
&\quad y_1 \oplus y_2 \oplus y_4, y_3 \oplus y_4, y_1 \oplus y_3 \oplus y_4, \\
&\quad y_2 \oplus y_3 \oplus y_4, y_1 \oplus y_2 \oplus y_3 \oplus y_4
\end{aligned}$$

Finally, O_n will be our *B*−order of F_2^n for our given *B*.

Now, using a given *B*−order, we can find the corresponding *B*−greedy code as follows.

4.1 Algorithm 4.1

1. Give the first word (zero word) g-value 0
2. To assign the *i*th word a g-value, let *r* be the smallest nonnegative integer such that the *i*th word has distance *d* or more to every previous word given g-value *r*. If *r* exists, assign the *i*th word g-value *r*.
3. If the *i*th word has a distance less than *d* to every previous word, assign it the next smallest integer g-value not used.

Finally, we create a code composed of all the words with a g-value of 0. This code is known as the *B*−greedy code. An observation about using Algorithm 4.1 to generate *B*−greedy codes is that since we only care about words with g-value 0, if we take a *B*− order and swap two words that lie between the same two codewords in the *B*−greedy code, the resulting *B*−greedy code does not change. This is because Algorithm 4.1 will still assign the two swapped words nonzero g-values, so the codewords with g-value 0 do not change.

A key theorem about *B*−greedy codes comes from Brualdi and Pless (7).

4.2 Theorem 4.1 (7)(4)
For a fixed *n* and *d,* let *B* be an ordered basis of F_2^n. Let *C* be the *B*−greedy code. Let *m* be the number of digits in the binary representation of the largest g-value over all words in F_2^n. Then, the map

$$g : \mathrm{F}_2^n \rightarrow \mathrm{F}_2^m$$

is a homomorphism with kernel *C.* In particular, *C* is a linear code with parity-check matrix

$$H = [g(e_n) \ \ldots \ g(e_2) \quad g(e_1)]$$

where e_i is the unit vector with a one in the *i*th position, and $g(w)$ is the g-value of *w,* written in binary from bottom to top. In addition, $g(w)$ is the syndrome of *w* with respect to *H.*

Proof: See (7)

Using Theorem 4.1, we are able to prove another result which will be crucial in the proofs later in this paper.

4.3 Corollary 4.1
Let g_1 be a homomorphism mapping words in F_2^n to g-values based on a *B*−order (g_1 is a homomorphism due to Theorem 4.1). Let *v* and *w* be any two words in F_2^n, then,

$$g_1(v \oplus w) = g_1(v) \oplus g_1(w)$$

Proof: Since g_1 is a homomorphism, it preserves nim-addition. So,

$$g_1(v \oplus w) = g_1(v) \oplus g_1(w)$$

as desired. Now, we will dive deeper into the study of lexicodes using the lens of *B*−greedy codes.

4.4 How to Generate a Lexicode
For a fixed *n* and *d,* we wish to find the unique lexicode of length *n* and distance *d.*

To begin, we will lexicographically list out all the words of length n. We then assign *g-values* to each word using Algorithm 4.1.

We select words with g-value 0 and add it to our code. The resulting code is the unique lexicode. This algorithm for producing lexicodes can be extended to q-ary settings. For interested readers, Bogdanova and Todorov (8) explored q-ary lexicodes of constant weight in their paper. Another observation about lexicodes is that changing the distance for sufficiently large distances results in minor changes to the dimension, as can be seen in Appendix C. As an example, we have included a table with g-values for the first 16 words from F_2^5 in lexicographic order and their g-values, where

$n = 5, d = 3$.

Table 4. The g-values of words in lexicographic order for $n = 5, d = 3$ Source: (4)

Word	g-value
00000	0
00001	1
00010	2
00011	3
00100	3
00101	2
00110	1
00111	0
01000	4
01001	5
01010	6
01011	7
01100	7
01101	6
01110	5
01111	4

Word	g-value
10000	5
10001	4
10010	7
10011	6
10100	6
10101	7
10110	4
10111	5
11000	1
11001	0
11010	3
11011	2
11100	2
11101	3
11110	0
11111	1

Taking a look at Algorithm 4.1, we see that the method for assigning g-values to words is similar to the method for assigning G-values to positions via the mex rule.

4.5 Theorem 4.2 (4)
Let $G(P)$ denote the G-value of a position P and let $g(P)$ denote the g-value of the position P as a binary number using lexicographic order. Then, $G(P) = g(P)$ for all positions P in the following game: Start with all binary numbers listed in lexicographic order. Starting at one binary number x, moving to another binary number y is legal if and only if y comes before x in lexicographic order and the distance between x and y is less than d.

Proof: For this impartial game, the set of positions is already a set of binary numbers. So, we can use Algorithm 4.1 to assign g-values to each word (binary number). Now, we may consider the G-values of each word by considering the way the impartial game works:

1. Start with a list of all positions, listed lexicographically
2. A move from P to P' is legal if $P' < P$ and $\Delta(P, P')$ is a turning set, that is, $d(P, P') < d$
3. The first player not to have any moves left is the loser

Now, when we assign the G-value for a position P, we use mex-rule and only look at positions $P' < P$ where $d(P, P') < d$. Notice that in step 2 of Algorithm 4.1, we are also going through positions before P and checking their distance to P, the same algorithm used for mex-rule. Therefore, $G(P) = g(P)$.

Going back to our discussion of assigning g-values, we will now present three results, Lemma 4.1, Lemma 4.2, Theorem 4.3, that we believe no paper has explicitly stated before. Brualdi and Pless's findings (7) directly imply Theorem 4.3, however, Brualdi and Pless do not present a proof for it. Before presenting the results, we define some notation.

Let $g(w)$ denote the g-value of word w. Let L_2^n denote the words of F_2^n listed in lexicographic order. Then, we can take the ordered list, which we will call an ordering, L_2^n and replace every word with its nim-sum with some fixed word w. We denote the resulting ordering as $L_2^n \oplus w$. As a side result, we prove that $L_2^n \oplus w$ is actually an ordering of the words in F_2^n.

4.6 Lemma 4.1
$L_2^n \oplus w$ is an ordering of the words in F_2^n.

Proof: $L_2^n \oplus w$ contains 2^n words, so all that is left to show is that all 2^n words are distinct. We will now perform a proof by contradiction. For the sake of contradiction, assume there exist two words in $L_2^n \oplus w$ that are equal.

Let $u \oplus w$ and $v \oplus w$ be two words in $L_2^n \oplus w$ that are equal but at different positions in the ordering. When constructing $L_2^n \oplus w$, $u, v \in L_2^n$ became $u \oplus w$ and $v \oplus w$ respectively. Since $u \oplus w = v \oplus w$, we have

$$u \oplus w \oplus w = v \oplus w \oplus w \Rightarrow u = v$$

as $\oplus$ is commutative and $w \oplus w = 0$. However, since u and v were two distinct words in L_2^n, they could not have been equal, a contradiction.

Therefore, $L_2^n \oplus w$ does not contain two words that are equal and is therefore an ordering of F_2^n.

As an example of $L_2^n \oplus w$, we have:

$$L_2^3 \oplus 011 = 011, 010, 001, 000, 111, 110, 101, 100$$

4.7 Lemma 4.2
For any word $w \in F_2^n$, using the ordering $L_2^n \oplus w$ on Algorithm 4.1 will result in the same corresponding g-values. In other words, the g-value of v using L_2^n will be equal to the g-value of $v \oplus w$ using $L_2^n \oplus w$).

Table 5. The g-values of words in the ordering $L_2^5 \oplus 00101$ for $n = 5, d = 3$ Source: (4)

Word	g-value
00101	0
00100	1
00111	2
00110	3
00001	3
00000	2
00011	1
00010	0
01101	4
01100	5
01111	6
01110	7
01001	7
01000	6
01011	5
01010	4

Word	g-value
10101	5
10100	4
10111	7
10110	6
10001	6
10000	7
10011	4
10010	5
11101	1
11100	0
11111	3
11110	2
11001	2
11000	3
11011	0
11010	1

As an example, we perform Algorithm 4.1 on $L_2^5 \oplus 00101$. As Lemma 4.2 predicts, the column of g-values in Table 5 is identical to that of Table 4.

Proof: Let $g_1(v)$ denote the g-value of word v using the lexicographic order. Let $g_2(v)$ denote the g-value of word v using the ordering $L_2^n \oplus w$. Lemma 4.2 is equivalent to the statement below for all $v \in F_2^n$.

$$g_1(v) = g_2(v \oplus w)$$

We will now perform a proof by contradiction. Suppose that $g_1(v)$ is not equal to $g_2(v \oplus w)$ for some $v \in F_2^n$. Consider the earliest word v in lexicographic order such that $g_1(v) \neq g_2(v \oplus w)$.

Since $g_1(v)$ and $g_2(v \oplus w)$ are not equal, there are two cases to consider. Consider the case when $g_1(v) < g_2(v \oplus w)$. Then, during Algorithm 4.1 for $L_2^n \oplus w$, since $g_2(v \oplus w) > g_1(v)$, the word $v \oplus w$ must have had distance less than d to some word $u \oplus w$ with g-value $g_1(v)$. So, we have

$$d(v \oplus w, u \oplus w) < d$$
$$\Rightarrow wt((v \oplus w) \oplus (u \oplus w)) < d$$

and $wt((v \oplus w) \oplus (u \oplus w)) = wt(v \oplus u \oplus w \oplus w)$ since $\oplus$ is commutative. Then, since $w \oplus w = 0$, $wt(v \oplus u \oplus w \oplus w) = wt(v \oplus u)$. So,

$$wt(v \oplus u) < d \Rightarrow d(v, u) < d$$

Now, since $u \oplus w$ came before $v \oplus w$ in $L_2^n \oplus w$, the word $u \oplus w$ was assigned g-value $g_2(u \oplus w) = g_1(u)$. When defining $u \oplus w$, we stated that $g_2(u \oplus w) = g_1(v)$, so $g_2(u \oplus w) = g_1(v) = g_1(u)$.

Since $u \oplus w$ came before $v \oplus w$ in $L_2^n \oplus w$, u came before v in L_2^n. We know that $d(v, u) < d$, so using Algorithm 4.1, we know that v and u can not have the same g-value. So, $g_1(v) \neq g_1(u)$, contradiction.

A symmetric argument shows that the case when $g_1(v) > g_2(v \oplus w)$ also results in a contradiction.

4.8 Theorem 4.3

Use Algorithm 4.1 to generate g-values using the lexicographic order and distance d. Then, over all words in F_2^n, there are exactly 2^{n-k} unique g-values, and each g-value corresponds to exactly 2^k words, where k is the dimension of the lexicode.

Proof 1: Let L be the lexicode with length n and distance d. Let $L \oplus w$ denote the code where every word in L is replaced with the nim-sum of the word and w.

Similar to the proof of Lemma 4.2, let $g_1(v)$ denote the g-value of word v using the lexicographic order. Let $g_2(v)$ denote the g-value of word v using the ordering $L_2^n \oplus w$.

Let w be a word with $g_1(w) = g$. From Lemma 4.2, after applying Algorithm 4.1 to the ordering $L_2^n \oplus w$, the words with g-value 0 will be codewords in $L \oplus w$.

In fact, the codewords in $L \oplus w$ are the set of words with g-value g using L_2^n. To see why this is true, consider a word u with $g_1(u) = g$. We wish to show that $u \in L \oplus w$. This is equivalent to showing that $u \oplus w \in L$.

From Corollary 4.1, we know

$$g_1(u \oplus w) = g_1(u) \oplus g_1(w)$$

So, we have

$$g_1(u \oplus w) = g_1(u) \oplus g_1(w) = g \oplus g = 0$$

and thus $u \oplus w$ is in L. Now, consider a word v with $g_1(v) \neq g$. We wish to show that $v \oplus w \notin L$.

From Corollary 4.1,

$$g_1(v \oplus w) = g_1(v) \oplus g_1(w) = g_1(v) \oplus g$$

Since $g_1(v) \neq g$, $g_1(v) \oplus g \neq 0$, and therefore, since $g_1(v \oplus w) \neq 0$, $v \oplus w$ is not in L. Therefore, $L \oplus w$ is equal to the set of words with g-value g.

Now, the code $L \oplus i$ for any word i has 2^k codewords. Since every word in F_2^n has a g-value, the set of $L \oplus i$'s must cover F_2^n. There are 2^n words in F_2^n and each $L \oplus i$ covers 2^k words. Therefore, there are 2^{n-k} distinct codes $L \oplus i$, and therefore 2^{n-k} distinct g-values. This completes our proof of Theorem 4.3.

Proof 2: Let g: $F_2^n \rightarrow F_2^m$ be the map from all words in F_2^n to their corresponding g-value, in binary. Then, from Theorem 4.1, g is a homomorphism.

Let u be an arbitrary word from F_2^n with $g(u) = v$. Then, proving Theorem 4.3 is equivalent to showing

$$\{w \in F_2^n : g(w) = v\} = L \oplus u$$

where $L \oplus u$ is the lexicode with every word replaced with its nim-sum with u.

Claim 1: $L \oplus u \subseteq \{w \in F_2^n : g(w) = v\}$
Take an arbitrary codeword l from L. Then, from Corollary 4.1, $g(l \oplus u) = g(l) \oplus g(u) = 0 \oplus v = v$.

Therefore, every codeword in $L \oplus u$ has g-value v, as desired.

Claim 2: $\{w \in F_2^n : g(w) = v\} \subseteq L \oplus u$

Take an arbitrary word w from F_2^n with $g(w) = v$. Then, since g is a homomorphism, it preserves nim-addition. So,

$$g(w) \oplus g(u) = g(w \oplus u)$$

Now, since $g(w) = g(u) = v$, we also have

$$g(w) \oplus g(u) = v \oplus v = 0$$

Therefore, $g(w \oplus u) = 0$, meaning $w \oplus u \in L \Rightarrow w \in L \oplus u$, as desired.

From Claim 1 and Claim 2, we have $L \oplus u \subseteq \{w \in F_2^n : g(w) = v\}$ and $\{w \in F_2^n : g(w) = v\} \subseteq L \oplus u$, so $\{w \in F_2^n : g(w) = v\} = L \oplus u$, which completes our proof of Theorem 4.3.

5 Bounds on Greedy Codes

The lexicode and B−greedy code constructions can sometimes result in optimal codes. As examples, the binary and extended Hamming codes and the binary and extended Golay codes are all lexicodes (2)(6). To construct these codes, we can simply set n and d to our desired values and use the algorithm to produce lexicodes introduced in Section 4.

Instead of discussing B−greedy codes and lexicodes as separate entities, we have an interesting result.

5.1 Lemma 5.1
Lexicodes are B−greedy codes

Proof: Take the ordered basis $B = \{e_1, e_2, \ldots e_n\}$. This basis describes a B−order equivalent to the lexicographic order. Thus, lexicodes are a type of B−greedy code.

Now, we present two more results in B−greedy codes.

5.2 Claim 5.1
Let C be a B−greedy code with distance d. The covering radius of C is at most $d - 1$.

Proof: We wish to show that there is no word not in C that has a distance of d or more from every codeword. For the sake of contradiction, assume there exists a word $w \notin C$ that has a distance of d or more from every codeword in C. By the greedy algorithm that constructed C, since w had distance of d or more from every codeword, it is included in C, contradiction.

5.3 Lemma 5.2
B−greedy codes satisfy the Gilbert-Varshamov (G-V) Bound.

Proof: Let C be any B−greedy code. Let n be the length of C and let d be the distance. Since C has covering radius $d - 1$, every word in F_2^n will have distance at most $d - 1$ to some codeword in C. Therefore, if we add any word not in C to C, the distance of the code would decrease, as the added word would have distance less than d to some word in C.

So, C is a maximal code and therefore satisfies the Gilbert-Varshamov Bound.

Now, we will introduce a result on maximal codes, which in tandem with our knowledge of B−greedy codes will lead us to a proof of a version of the G-V Bound.

Brualdi and Pless's work in (7) resulted in the proof that B−greedy codes were linear. For the first time, we will extend this finding, and eventually use our discoveries to tackle a version of the G-V Bound.

5.4 Lemma 5.3
Let C be a linear code of length n. Any basis of C is a subset of some basis of F_2^n.

Proof: Let $\{v_1, v_2, \ldots, v_k\}$ be an arbitrary basis of C. This means all v_i are linearly independent, in F_2^n, and span C. So, by the Basis Extension Theorem, we know we can find a basis of F_2^n that contains $\{v_1, v_2, \ldots, v_k\}$. Thus, any basis of C is a subset of some basis of F_2^n.

5.5 Definition 5.1
A *maximal* code is a $[n, k, d]$ code which is not contained by a $[n, k+1, d]$ code. If any word not in a maximal code is added to the code, the distance will decrease.

5.6 Theorem 5.4
All maximal linear codes are B−greedy codes.
Proof: Let our maximal linear code be C with dimension k and distance d, and let $\mathrm{B} = (b_1, \ldots, b_k)$ be some ordered basis of C. Using Lemma 5.3, we can extend B to an ordered basis $B = (b_1, \ldots, b_k, b_{k+1}, \ldots, b_n)$ of F_2^n.

5.7 Weak Gilbert-Varshamov Bound
Then, taking the B−greedy code using the ordered basis B, we will add the first 2^k words to our code. That is, our code will include all words in C, as the first 2^k words are all the words in C. Now, since C is a maximal code, we claim that as we continue with our greedy algorithm, no other word may be added to the code. For the sake of contradiction, assume a word x is added to C after the first 2^k words are added. Then, since $x \notin C$, the span of $C \cup \{x\}$ will have distance less than d by maximality of C. So, there will exist a codeword $c \in C$ such that $d(c, x) < d$. However, by the greedy algorithm, x would not have been added to the code, as it has distance less than d from one of the previous codewords added, contradiction. Therefore, any word outside the first 2^k words cannot be assigned g-value 0, or else it would've been added to C. So, the greedy algorithm terminates once it adds all of C's codewords, therefore, C will be the B−greedy code. Thus, all maximal linear codes are also B−greedy codes.

The proof of Theorem 5.4 rules out the possibility of a non-linear extension, which implies that no further g-value can be assigned a zero label. This work is only concerned with linear codes resulting from greedy algorithms hence the separation of linear versus non-linear cases is not deemed necessary.

Finally, we will use the results developed in this paper to present a proof of a slightly weaker version of the binary Gilbert-Varshamov Bound.

For maximal linear binary codes C with length n, distance d, and dimension k, we have

$$\frac{2^n}{\sum_{i=0}^{d-2}\binom{n}{i}} \geq 2^k$$

Proof: From Theorem 5.4, we know that maximal linear binary codes are also B−greedy codes. Now, using Theorem 4.1,

$$H = [g(e_n) \ \ldots \ g(e_2) \ \ g(e_1)]$$

will be the parity-check matrix of C. Now, since C has distance d, we have the following conditions/restrictions:

$$g(e_{i1}) \neq 0$$
$$g(e_{i1} \oplus e_{i2}) \neq g(e_{i3})$$
$$g(e_{i1} \oplus e_{i2} \oplus e_{i3}) \neq g(e_{i4})$$
$$\ldots$$
$$g(e_{i1} \oplus e_{i2} \cdots \oplus e_{i(d-2)}) \neq g(e_{i(d-1)})$$

For any unit vectors $e_{i1}, e_{i2}, \ldots, e_{i(d-1)}$. Notice that we have each restriction because if for example, $g(e_{i1} \oplus e_{i2}) = g(e_{i3})$, then by Theorem 4.3, $g(e_{i1} \oplus e_{i2} \oplus e_{i3}) = 0$. This is not possible when $d \geq 3$ since the distance is d, and $d(0, e_{i1} \oplus e_{i2} \oplus e_{i3}) = 3 < d$.

Now, with these restrictions in mind, notice that each column in H is a binary number, and we have the restriction that no $1, 2, \ldots, d-2$ column vectors can sum to another column vector as well as no 0 vectors. A well-informed reader should notice that this set-up is almost identical to that for the proof of Gilbert-Varshamov. Indeed, the rest of this proof may look familiar to those who have seen this proof.

We have n column vectors, and we defined the parity-check matrix to have dimensions $m \times n$, so each column vector has length m. We also have the restrictions given above. So, we have the inequality

$$n + \binom{n}{2} + \cdots + \binom{n}{d-2} \leq 2^m - 1$$

as each vector in F_2^n is either one of the n column vectors, covering n vector, a sum of two column vectors, covering $\binom{n}{2}$ vectors, and so on. Finally, a vector could also not be able to be expressed as the sum of $d-2$ or less column vectors. So, we have

$$1 + n + \binom{n}{2} + \cdots + \binom{n}{d-2} \leq 2^m$$

since both sides are positive, we have

$$\frac{2^n}{\sum_{i=0}^{d-2}\binom{n}{i}} \geq \frac{2^n}{2^m}$$

Now, since H was a $m \times n$ matrix, the dimension k is $n - m$. So,

$$\frac{2^n}{\sum_{i=0}^{d-2}\binom{n}{i}} \geq 2^k$$

as desired.

6 Conclusion

This paper started by introducing how any impartial game could be transformed into a lexicode by taking its winning positions and turning them into codewords. We analyzed a couple of impartial games and assigned G-values using Sprague-Grundy analysis, powered by the mex and decomposition rules. After constructing the winning code, we showed that they were linear, which then led us to discover that lexicodes are also linear. We then discovered that lexicodes were a part of a bigger class of B−greedy codes. By constructing B−greedy codes, we were able to discover many brilliant results, many of which were powered by Theorem 4.1. Each B−greedy ordering also corresponds to a mapping between words in F_2^n and g-values. We found that the algorithm for assigning g-values to words in F_2^n had similarities to the algorithm for assigning G-values to positions. For g-values in the lexicographic case, we also found the number of g-values and corresponding words with that g-value. Finally, we finished the paper by presenting the proof of the Gilbert-Varshamov Bound using our knowledge of B−greedy codes. In this work, we have limited our scope to the simple cases of lexicodes and B-greedy codes resulting from B-orderings or $L_n \oplus w$. Therefore, a broader sensitivity analysis over bases, thresholds d, and q-ary alphabets is not attempted, and neither is the robustness of derived dimensions across many instances assessed.

For the future, we would like to investigate the question of determining a closed-form formula for the dimension of a lexicode given its length and distance, as this would directly give us a way to compare the usefulness of lexicodes with other codes.

References

1. Grundy, P. M. (1939). Mathematics and games. *Eureka*, *2*, 6-8.

2. Berlekamp, E. R., Conway, J. H., & Guy, R. K. (2004). *Winning ways for your mathematical plays* (Vol. 4). AK Peters/CRC Press. https://doi.org/10.1201/9780429487309

3. Sprague, R. P. (1935). Über mathematische Kampfspiele. *Tohoku Mathematical Journal, First Series*, *41*, 438-444.

4. Pless, V. (2011). *Introduction to the theory of error-correcting codes*. John Wiley & Sons.

5. Bouton, C. L. (1901). Nim, a game with a complete mathematical theory. *Annals of Mathematics*, *3*(1-4), 35-39.

6. Conway, J. H., Sloane, N. J. A. (1986). Lexicographic codes: Error-correcting codes from game theory. *IEEE Transactions on Information Theory*, *32*(3), 337-348. https://doi.org/10.1109/TIT.1986.1057187

7. Brualdi, R. A., Pless, V. S. (1993). Greedy codes. *Journal of Combinatorial Theory, Series A*, *64*(1), 10-30. https://doi.org/10.1016/0097-3165(93)90085-M

8. Bogdanova, G., Todorov, T. (2022). On the construction of q-ary constant-weight lexicodes. *International Journal of Mathematics and Computer Science*, *17*(2), 923-930.

9. Grassl, M. (2007). *Bounds on the minimum distance of linear codes and quantum codes*. http://www.codetables.de

Appendix A. Python Code for *B*−order Generation

The following python code generates the complete *B*−order for a given basis.

```
import itertools
import numpy as np

def xor(list1,list2):
#returns the xor of two binary numbers
expressed #as lists of 0s and 1s
  return [i ^ j for i, j in zip(list1, list2)]

def strtolist(string):
#converts a string of 0s and 1s to a list of
0s #and 1s
  list1=[]
  for num in string:
    list1.append(int(num))
  return list1

def lexiorder(length):
#returns a list of all binary numbers of
given #length in lexicographic order
  lexiorder = []
  for i in  range(0,2 ** length):
    word =
    strtolist(bin(i)[2:].zfill(length))
    lexiorder.append(word)
  return lexiorder

def b_order(basis):
#returns the B-order for a given
  basis length = len(basis) #
  order = []

  lexi_order =
  lexiorder(length) for
  coeffvector in lexi_order:
      word = [0] * length
      for idx, digit in
          enumerate(coeffvector): if digit
          == 1:
              word = xor(word, basis[idx])
      order.append(word)
    #We took every coefficient vector in
    the #lexicographic order, and used
    it to #generate its corresponding
    word in the #B-order using the basis
    vectors.
  return order
```

By inputting a list of basis vectors and calling the b-order function, we can generate the $B-$order for any given basis. We present a sample here:

```
#Input:
basis=[[0,1,0],[1,1,1],[1,1,0]
]
print(b_order(basi
s)) #Output:
[[0,0,0], [1,1,0], [1,1,1], [0,0,1], [0,1,0],
[1,0,0], [1,0,1], [0,1,1]]
```

Appendix B. Python Code for Lexicode Generation

The following Python code generates the lexicode for a given dimension and length.

```
import itertools
import numpy as np

def xor(list1,list2):
#returns the xor of two
binary #numbers expressed
as lists of #0s and 1s
 return [i ^ j for i, j
                      in
      zip(list1, list2)]

def strtolist(string):
#converts a string of 0s
and #1s to a list of 0s
and 1s
  list1=[]
  for num in string:
    list1.append(int(num))
  return list1

def lexiorder(length):
#returns a list of all binary
#numbers of given length in
#lexicographic order
  lexiorder = []
  for i in range(0,2 ** length):
  word = strtolist(bin(i)[2:].zfill(length))
    lexiorder.append(word)
  return lexiorder

def lexicode(d,l):
   #returns the lexicode with
   #dimension d and length l
   lexicode = []
   for word in
     lexiorder(l): for
     codeword in lexicode:
       if
         xor(word,codeword).count(1)
         <d: break
     else:
       lexicode.append(word)

   return lexicode
```

By using the lexicode function, we can input dimension and length, and the code will output a list of vectors, those being the codewords in the lexicode. We present a sample here:

```
#Input:
print(lexicode(3,5
)) #Output:
[[0,0,0,0,0], [0,0,1,1,1], [1,1,0,0,1],
[1,1,1,1,0]]
```

Appendix C. Comparing Dimension to Best Known Bounds

The following table will display dimensions for lexicodes and in the parentheses, best known dimensions for linear codes of the given length and distance. The table was constructed with the help of data from CodeTables (9).

Table 6. Dimensions for Lexicodes and Best Known Linear Codes

n\d	4	5	6	7	8
4	1(1)				
5	1(1)	1(1)			
6	2(2)	1(1)	1(1)		
7	3(3)	1(1)	1(1)	1(1)	
8	4(4)	2(2)	1(1)	1(1)	1(1)
9	4(4)	2(2)	2(2)	1(1)	1(1)
10	5(5)	3(3)	2(2)	1(1)	1(1)
11	6(6)	4(4)	3(3)	2(2)	1(1)
12	7(7)	4(4)	4(4)	2(2)	2(2)
13	8(8)	5(5)	4(4)	3(3)	2(2)
14	9(9)	6(6)	5(5)	4(4)	3(3)
15	10(10)	7(7)	6(6)	5(5)	4(4)
16	11(11)	8(8)	7(7)	5(5)	5(5)
17	11(11)	9(9)	8(8)	6(6)	5(5)
18	12(12)	9(9)	9(9)	7(7)	6(6)
19	13(13)	10(10)	9(9)	8(8)	7(7)
20	14(14)	11(11)	10(10)	9(9)	8(8)
21	15(15)	12(12)	11(11)	10(10)	9(9)
22	16(16)	12(13)	12(12)	11(11)	10(10)
23	17(17)	13(14)	12(13)	12(12)	11(11)
24	18(18)	14(14)	13(14)	12(12)	12(12)
25	19(19)	15(15)	14(14)	12(12)	12(12)
26	20(20)	16(16)	15(15)	12(13)	12(12)
27	21(21)	17(17)	16(16)	13(14)	12(13)
28	22(22)	18(18)	17(17)	13(14)	13(14)
29	23(23)	19(19)	18(18)	14(15)	13(14)
30	24(24)	19(20)	19(19)	15(16)	14(15)
31	25(25)	20(21)	19(20)	16(17)	15(16)
32	26(26)	21(22)	20(21)	16(18)	16(17)

The table illustrates that the dimension of lexicodes tend to be very close to the best known dimensions. In fact, the very first length of a lexicode that results in a different dimension from the best known dimension for ($4 \leq d \leq 8$) is $n = 22$.