


5-2017

Combinatorial Games on Graphs

Trevor K. Williams
Utah State University

Follow this and additional works at: <https://digitalcommons.usu.edu/etd>

 Part of the [Mathematics Commons](#), and the [Statistics and Probability Commons](#)

Recommended Citation

Williams, Trevor K., "Combinatorial Games on Graphs" (2017). *All Graduate Theses and Dissertations*. 6502.
<https://digitalcommons.usu.edu/etd/6502>

This Thesis is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact dylan.burns@usu.edu.



COMBINATORIAL GAMES
ON GRAPHS

by

Trevor K. Williams

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTERS OF SCIENCE

in

Mathematics

Approved:

David Brown
Major Professor

James Cangelosi
Committee Member

LeRoy Beasley
Committee Member

Mark R. McLellan
Vice President for Research and
Dean of the School for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2017

Copyright © Trevor Williams, 2017

All Rights Reserved

ABSTRACT

Combinatorial Games on Graphs

by

Trevor K. Williams, Master of Science

Utah State University, 2017

Major Professor: Dr. David Brown
Department: Mathematics and Statistics

Combinatorial games are intriguing and have a tendency to engross students and lead them into a serious study of mathematics. The engaging nature of games is the basis for this thesis. Two combinatorial games along with some educational tools were developed in the pursuit of the solution of these games.

The game of Nim is at least centuries old, possibly originating in China, but noted in the 16th century in European countries. It consists of several stacks of tokens, and two players alternate taking one or more tokens from one of the stacks, and the player who cannot make a move loses. The formal and intense study of Nim culminated in the celebrated Sprague-Grundy Theorem, which is now one of the centerpieces in the theory of impartial combinatorial games. We study a variation on Nim, played on a graph. Graph Nim, for which the theory of Sprague-Grundy does not provide a clear strategy, was originally developed at the University of Colorado Denver. Graph Nim was first played on graphs of three vertices. The winning strategy, and losing position, of three vertex Graph Nim has been discovered, but we will expand the game to four vertices and develop the winning strategies for four vertex Graph Nim.

Graph Theory is a markedly visual field of mathematics. It is extremely useful for graph

theorists and students to visualize the graphs they are studying. There exists software to visualize and analyze graphs, such as SAGE, but it is often extremely difficult to learn how use such programs. The tools in GeoGebra make pretty graphs, but there is no automated way to make a graph or analyze a graph that has been built. Fortunately GeoGebra allows the use of JavaScript in the creation of buttons which allow us to build useful Graph Theory tools in GeoGebra. We will discuss two applets we have created that can be used to help students learn some of the basics of Graph Theory.

The *game of thrones* is a two-player impartial combinatorial game played on an oriented complete graph (or tournament) named after the popular fantasy book and TV series. The game of thrones relies on a special type of vertex called a king. A king is a vertex, k , in a tournament, T , which for all x in T either k beats x or there exists a vertex y such that k beats y and y beats x . Players take turns removing vertices from a given tournament until there is only one king left in the resulting tournament. The winning player is the one which makes the final move. We develop a winning position and classify those tournaments that are optimal for the first or second-moving player.

(73 pages)

PUBLIC ABSTRACT

Combinatorial Games on Graphs

Trevor K. Williams

Combinatorial Games are intriguing and have a tendency to engross students and lead them into a serious study of mathematics. The engaging nature of games is the basis for this thesis. Two combinatorial games and some educational tools are presented which were developed by the author in the pursuit of the solution of these games.

ACKNOWLEDGMENTS

I would like to thank Dr. David Brown for his consistent pressure to help me become a better mathematician. I'm extremely grateful for the many opportunities he gave me to work on and be recognized for my research. I am also very grateful for my wife and family for their support and patience. I would also like to thank my committee members and the other professors at Utah State University that have pushed me in my pursuit of mathematics. Last, I would like to thank my peers at Utah State University for their support and guidance through my graduate schooling.

Trevor K. Williams

CONTENTS

ABSTRACT	iii
PUBLIC ABSTRACT	v
ACKNOWLEDGMENTS	vi
LIST OF TABLES	viii
LIST OF FIGURES	ix
CHAPTER	
0. MOTIVATION	1
1. NIM AND THE SPRAGUE-GRUNDY THEOREM	3
1.1 The Sprague-Grundy Theorem	5
1.2 Graph Theory Basics	6
1.3 The Sprague-Grundy Function	7
2. GRAPH NIM	10
2.1 Introduction	10
2.2 Graph Nim on a 3-vertex Graph	11
2.3 Graph Nim on a 4-vertex Graph	14
2.4 Graph Nim on Other Graphs	22
2.5 Sprague-Grundy Theorem and Graph Nim	24
2.6 Future Work	31
3. GEOGEBRA	35
3.1 Graphs and Adjacency Matrices	35
3.2 Tournaments, Score, and Kings	39
3.3 Conclusion	44
4. THE GAME OF THRONES	46
4.1 Game Play	47
4.2 A Winning Position	47
4.3 Tournament Optimality Classification	49
4.4 Sprague-Grundy Theorem and The Game of Thrones	52
4.5 Future Work	53
REFERENCES	55
APPENDICES	56
APPENDIX- SAGE code used to generate S_n up to order 8	57

LIST OF TABLES

Table		Page
2.1	A Summary of the Winning Strategies for Graph Nim on 4-vertex Graphs.	24
4.1	Sizes of the Sets Described in the Proof of Lemma 2, for Tournaments up to Order 8.	51

LIST OF FIGURES

Figure	Page
1.1 An Example Game of Nim	3
1.2 A Game of Nim in a Losing Position	4
1.3 A Graph with 4 Vertices and 5 Edges.	7
1.4 A Digraph with 4 Vertices.	7
1.5 An Example of a Sprague-Grundy Graph	8
1.6 A Labeled Sprague-Grundy Graph	8
1.7 Grundy Numbers Given by the mex Function	9
2.1 An Example of Generalizing Nim	10
2.2 An Example of Graph Nim	11
2.3 An Example of How Graph Nim is Played	12
2.4 A Losing Position for Graph Nim on 3 Vertices	13
2.5 Graph Nim on a 4-vertex Graph Using the Multiple Edge Representation.	13
2.6 Example Showing that Labeling is not Important for Graph Nim	15
2.7 Every Distinct 4-vertex Graph	16
2.8 $\overline{K_4}$	17
2.9 $2K_2$	17
2.10 $K_2 \cup 2K_1$	17
2.11 $K_{1,2} \cup K_1$	18
2.12 $K_{1,3}$	18
2.13 C_4	19
2.14 P_4	20
2.15 $K_3 \cup K_1$	20
2.16 $\overline{K_{1,2} \cup K_1}$	21
2.17 K_4	21
2.18 $\overline{K_2 \cup 2K_1}$	22
2.19 All Possible Positions of 3-vertex Graph Nim when Edge Weights are Less Than or Equal to 1	25
2.20 Sprague-Grundy Graph of 3-vertex Graph Nim when Edge Weights are Less Than or Equal to 1	25
2.21 New Positions that are Possible if the Edge Weights are Less Than or Equal to 2 in 3-vertex Graph Nim.	26
2.22 Sprague-Grundy Graph of 3-vertex Graph Nim when Edge Weights are Less Than or Equal to 2	27
2.23 Labeled Sprague-Grundy Graph of 3-vertex Graph Nim when Edge Weights are Less Than or Equal to 2	27
2.24 Labeled Sprague-Grundy Graph for the Starting Position in Figure 2.2	27
2.25 An Example of Game Play and the Sprague-Grundy Graph	30
2.26 All possible positions of 4-vertex Graph Nim when edge weights are less than or equal to 1	31
2.27 Sprague-Grundy Graph of 4-vertex Graph Nim when Edge Weights are Less Than or Equal to 1	32

2.28	Labeled Sprague-Grundy Graph of 4-vertex Graph Nim when Edge Weights are Less Than or Equal to 1	33
3.1	A Graph and its Adjacency Matrix	36
3.2	Deleting Previous Graphs and Matrices	36
3.3	Creating a Random Number of Vertices	37
3.4	Creating Random Edges	37
3.5	Analyzing the Graph and Building the Matrix	38
3.6	Placing the Matrix in GeoGebra	39
3.7	Adjacency Matrix Applet	39
3.8	A Tournament with 4 Vertices.	40
3.9	A Tournament with 4 Vertices and its Adjacency Matrix	41
3.10	Input Bar Code	42
3.11	Show Kings Button Code	43
3.12	Hide Kings Button Code	43
3.13	Show Score Button Code	44
3.14	Show Vertex Names Button Code	44
3.15	Tournament Applet	45
4.1	An Example Game in which the First Player Wins	47
4.2	$T_5^{(2)}$	49
4.3	$T_5^{(2)}$	50
4.4	All Tournaments in S_6	52
4.5	A 6-tournament	53
4.6	The Labeled Sprague-Grundy Graph for the Tournament in Figure 4.5	53
4.7	The SAGE Code used to Generate the Sets S_n up to Order 8.	58

CHAPTER 0

MOTIVATION

Combinatorial Games are intriguing and have a tendency to engross students and lead them into a serious study of mathematics. I, of course, speak from experience. The engaging nature of games is the basis for this thesis. As an undergraduate I took a discrete mathematics course from Dr. David Brown in which I was presented a game that was developed during a Research Experience for Teachers (RET) at the University of Colorado-Denver by Dr. Michael Ferrara. This game was a variation of the ancient game Nim played on a graph (this game will be discussed in detail later). During the RET, and as a homework assignment in Dr. Brown's class, a strategy was developed such that the first player could always win given the appropriate starting conditions of the game when the game was played on a three vertex graph.

This homework exercise ignited my desire to pursue mathematical research. I wanted to discover a winning strategy for Graph Nim on a four vertex graph. This desire drove me to learn the basics of combinatorial game theory, which is not taught in any class on campus, and to pursue a graduate degree in mathematics. I began attending research seminars held by Dr. Brown and began to pursue research opportunities. During a research seminar meeting Dr. Brown invited Dr. Larry Langley of the University of the Pacific to present some of his research. Dr. Langley presented a special class of vertices in tournaments that he, and Dr. Kim Factor of Marquette University, had invented. During this presentation a game was discussed that Dr. Brown named The Game of Thrones after the popular fantasy series (the motivation of this name will become clear when we discuss the game in more detail). This game intrigued me and I began to pursue a winning strategy. My pursuit of a winning strategy for The Game of Thrones required me to learn more tournament theory.

Because of the complexity of The Game of Thrones, it became clear early in my research that I would not be able to analyze the game without the aid of a computer. The visual nature of the game, and Graph Theory in general, suggested that I would need a program with a Graphic User Interface (GUI). Since I had little training in practical computer science it seemed unproductive for

me to write a program with a GUI. I searched for alternate solutions and found that the open source program GeoGebra allowed automated construction and analysis through the use of JavaScript. This proved invaluable to my research, but I realized that these tools may be of great educational benefit to Graph Theory students. As stated previously Graph Theory is a very visual subject, but it is hard for students to “play” with graphs because of the time it takes to draw and manipulate graphs. I developed a couple of applets in GeoGebra as an example of how an educator might allow students to “play” with graphs to facilitate learning. Dr. Brown found a journal devoted to the use of GeoGebra in education and I prepared a manuscript to submit to this journal. My goal was not to create a plethora of Graph Theory applets in GeoGebra, but rather, inform educators that this powerful tool exists and to provide some examples of how it might be used.

The chapters of this thesis may appear to be three unconnected projects, and they are, but this thesis tells a single story. This thesis is an illustration of the effects of introducing games to students. I hope that educators will realize the power of posing a problem as a game, and recognize that this can ignite the curiosity of their students. These games may not be important in and of themselves, but they can certainly lead students to a motivated study of extracurricular mathematics.

CHAPTER 1
NIM AND THE SPRAGUE-GRUNDY THEOREM

Solving Graph Nim requires a basic understanding of combinatorial game theory. The Sprague-Grundy Theorem[5, 10] is one of the centerpieces in the theory of impartial combinatorial games and will be presented here. In order to state The Sprague-Grundy Theorem the game of Nim must first be presented. Nim is at least centuries old, possibly originating in China, but noted in the 16th century in European countries. The game is played with several stacks of tokens called heaps, and two players alternate taking one or more tokens from a single heap. The player who makes the last move wins; this is called “normal play”. Nim has been solved for any number of initial heaps and tokens, meaning there is a winning strategy for the first player provided the game meets certain initial conditions. This chapter will present the theory of Nim and the celebrated Sprague-Grundy Theorem.

First, define an associative, commutative, binary operation called the *Nim-sum* or *Xor* (which is short for “exclusive or”), and denoted \oplus . The Nim-sum is used to calculate what is called the *Nimber* of a game of Nim. The Nimber is simply the Nim-sum of all the heaps of the game, and is vital in determining the winning strategy of the game. This operation will be defined using an example. Figure 1.1 shows a position in a game of Nim, it contains three heaps that have 6, 7, and

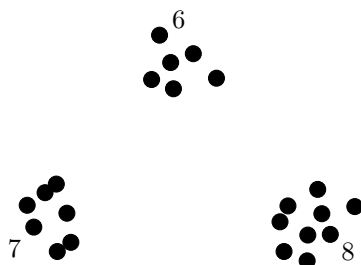


Figure 1.1: An Example Game of Nim

8 tokens. In order to calculate the Nimber of this game the heap sizes must first be converted to

binary numerals. Doing this for the example the following numerals are obtained

$$6 \rightarrow 0110$$

$$7 \rightarrow 0111$$

$$8 \rightarrow 1000$$

Now to calculate $(6 \oplus 7) \oplus 8$ the binary representations of 6, 7, and 8 are added digit-wise, but adding the digits as follows: $1 + 0 = 0 + 1 = 1$, $0 + 0 = 0$, and $1 + 1 = 0$. Another way to think of this is that digits that are different add to 1, and digits that are the same add to 0, so

$$\begin{array}{r} 0110 \\ \oplus 0111 \\ \hline 0001 \\ \oplus 1000 \\ \hline 1001 \end{array}$$

Next the result is converted back to a decimal numeral $1001 \rightarrow 9$ to obtain the Nimber, so $6 \oplus 7 \oplus 8 = 9$. If the Nimber of a game is equal to zero then the game is in a losing position, meaning that no matter what move a player makes, they cannot win. The game in Figure 1.1 is not in a losing position, but can be reduced to a losing position in a single move. If a player selects the heap with 8 tokens and removes seven of them the resulting game is in the losing position. This game, depicted in

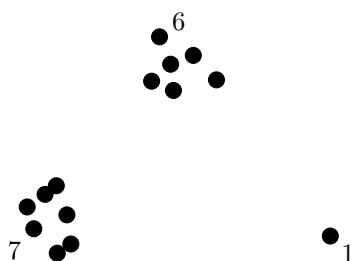


Figure 1.2: A Game of Nim in a Losing Position

Figure 1.2, has three heaps that have 1, 6, and 7 tokens, and $1 \oplus 6 \oplus 7 = 0$ using the Nim-sum operation. How does a player determine what tokens to remove in order to put the game into the losing position?

The beauty of Nim and The Sprague-Grundy Theorem is that there is an easy predictable strategy for the game. Now assume that the game in Figure 1.1 is the starting position of a game

of Nim, the first player must pick a heap and remove any number of tokens they like. In order to determine which heap to pick they will first calculate the Nimber of the game, which was previously calculated as $6 \oplus 7 \oplus 8 = 9$. Now that the player knows the Nimber, 9, they will calculate the Nim-sum of the size of each heap with the Nimber.

$$6 \oplus 9 = 15$$

$$7 \oplus 9 = 14$$

$$8 \oplus 9 = 1$$

When they did this only a single Nim-sum was less than the size of the initial heap, $8 \oplus 9 = 1$, then to put the game in the losing position they simply reduce the heap of size 8 to one of size 1, which is the position in Figure 1.2. It has been proven that if the starting position does not have a Nimber of 0, then the first player can always make a move to a position that has a Nimber of 0, and can therefore always win[1].

1.1

The Sprague-Grundy Theorem

The Sprague-Grundy Theorem only applies to impartial, sequential games with perfect information, or *impartial combinatorial games*. Impartial means that a player's move depends only on the position of the game. Another way to think about this is that the game does not have pieces for each player. Chess and Tic-tac-toe are examples of partisan games; Nim is an example of an impartial game. If a game has perfect information each player is perfectly informed of all the events that have previously occurred including the initial position of the game. Sequential means each position of the game is obtained by manipulating the previous position. Poker is an example of a game that is not sequential and does not have perfect information. It is not sequential because each hand does not rely on the previous hand and it does not have perfect information because players are not aware of which cards the other players have. Nim is both sequential and has perfect information. Now the Sprague-Grundy Theorem states:

Theorem (Sprague 1935, Grundy 1939). *Every impartial combinatorial game under the normal play convention is equivalent to a Nimber.*

The Sprague-Grundy Theorem means that there is some function that assigns numbers, called Grundy Numbers, to the positions of an impartial combinatorial game and each Grundy number

is equivalent to a Nimber. So if the appropriate function for a game can be determined, then the losing positions of the game, which are when the Grundy number is equal to zero, are known. In order to present this function some Graph Theory basics will first be introduced.

1.2

Graph Theory Basics

Graph Theory is the study of objects and the relationships between them. These objects and their relationships are studied using graphs. A *graph* in Graph Theory is a collection of vertices (which represent the objects) and edges (which represent the relationships between the objects). Graphs are usually defined as an ordered pair, $G = (V, E)$, where V is called the vertex set and is a set of objects, and E the edge set contains subsets of size two of V . Two vertices, u, v , are called *adjacent* if $\{u, v\} \in E$. The 2-set $\{u, v\}$ is an edge. An edge is *incident* to a vertex if the vertex is contained in the edge.

It is often convenient to represent a graph as a diagram. For example consider the graph, G , with vertex set

$$V = \{A, B, C, D\}$$

and edge set

$$E = \{\{A, B\}, \{B, C\}, \{C, D\}, \{A, D\}, \{B, D\}\}$$

This diagram is shown in Figure 1.3. It is common to refer to these diagrams as graphs themselves.

The vertices in a graph can represent any set of things, or they may represent nothing and be objects in and of themselves. Graphs can be used to study one-way or two-way relationships. For example, if the vertices in a graph represent people and the edges represent marriage the graph represents a two-way relationship. If Joe is married to Sally, then Sally has to be married to Joe. Marriage is a two-way relationship, two-way relationships are called symmetric. If the vertices in a graph represent sports teams and the edges represent beating a team, then the graph is depicting one-way relationships. If The New York Yankees beat the Boston Red Sox, then the Boston Red Sox did not beat the New York Yankees. When edges represent a one-way relationship they are called arcs and are represented by an arrow to show which way the relationship is going; in other words the edges are ordered pairs of vertices. Graphs that show only one-way relationships are called Directed Graphs, or Digraphs for short. Figure 1.3 shows a graph with four vertices and five edges. In this graph vertices A and D are adjacent, but vertices A and C are not. Edges are generally named

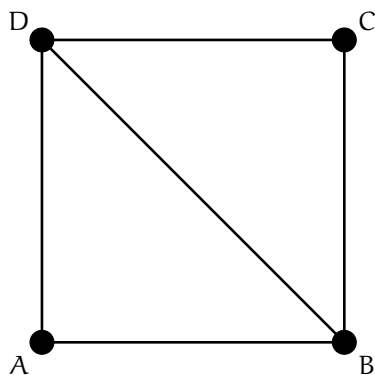


Figure 1.3: A Graph with 4 Vertices and 5 Edges.

based on the vertices they connect. For example, the edge that connects A and D, may be called edge AD. Also, note that AD is incident to vertex A and to vertex D.

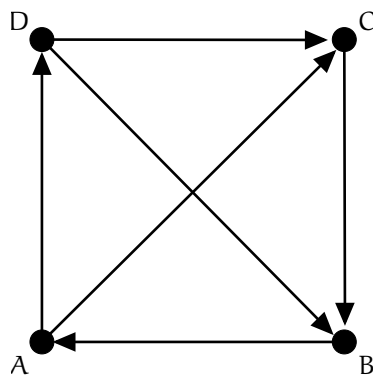


Figure 1.4: A Digraph with 4 Vertices.

1.3

The Sprague-Grundy Function

The Sprague-Grundy Theorem shows that there is some function that assigns Grundy Numbers to the positions of an impartial combinatorial game. The function requires the construction of a graph. This graph is built by creating a vertex for the initial position of the game. Next, a vertex for each position that can be obtained by a single move from the initial position of the game is created, and an arc from the first vertex to each of the new vertices is added. The process is continued for each new vertex until a vertex for each possible position of the game has been created. Figure 1.5 is an example of how such a graph would look. These graphs will be referred to as *Sprague-Grundy Graphs*.

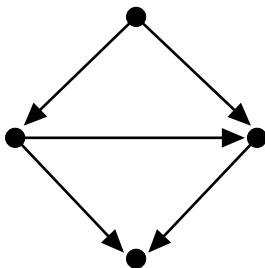


Figure 1.5: An Example of a Sprague-Grundy Graph

The vertices of this graph will be labeled using the following algorithm. Start by labeling all vertices with no arcs leaving them “P”, these are the terminal (final) positions of the game. Next label any vertex with an arc pointing to a “P” vertex as “N”. Then label any vertex only pointing to “N” vertices as “P”. Repeat these steps until every vertex is labeled. If this process is applied to the graph in Figure 1.5, the graph in Figure 1.6 is obtained.

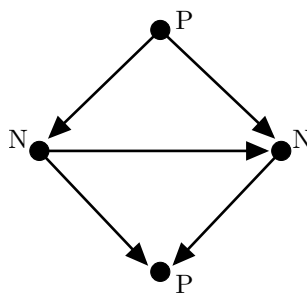


Figure 1.6: A Labeled Sprague-Grundy Graph

Now a function called *mex* (which is short for *minimum excluded value*), will be used to assign Grundy Numbers to these positions. The *mex* function takes a set and outputs the minimum excluded non-negative integer; for example:

$$\text{mex}(1, 3, 5, 6) = 0$$

$$\text{mex}(0, 2, 4, 7) = 1$$

Begin by assigning the number 0 to all terminal positions. Now all vertices that only point to terminal positions will receive a *mex* value of 1. Then for every other vertex assign the value given by the *mex* of all the vertices they point to[2]. Doing this for the example the values shown in Figure 1.7 are obtained. Then by The Sprague-Grundy Theorem, these Grundy Numbers are equivalent to a Nimber. Therefore, the positions that were assigned a value of zero are the losing positions of the

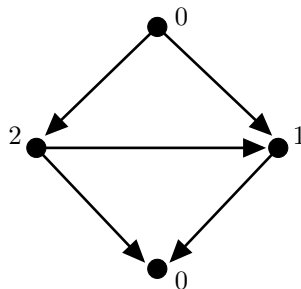


Figure 1.7: Grundy Numbers Given by the mex Function

game, note that all “P” vertices received a value of 0. The winning strategy is given by the graph, a player simply has to move the game to a position that has a Grundy Number of 0 on each turn. We know that this is always possible given that the game did not start at a position with Grundy Number 0, because each Grundy Number is equivalent to a Nimber.

CHAPTER 2

GRAPH NIM

2.1

Introduction

Recall the game of Nim described in the Introduction. Nim consists of several stacks of tokens called heaps, and two players alternate taking one or more tokens from a single heap. Now consider a generalization of this game in which the heaps are arranged in some order, and certain pairs of heaps are associated. Then, instead of selecting a single heap to remove tokens from a player can now select heaps that are associated and remove any number of tokens from any of those heaps. For example, Figure 2.1 shows a game of the example game of Nim from the last chapter. In this example the heaps are associated in a triangular shape. Now a player can select any two heaps to

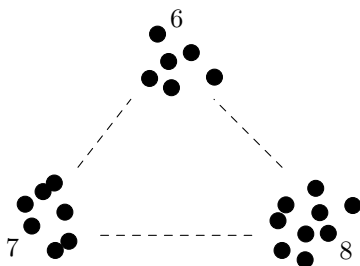


Figure 2.1: An Example of Generalizing Nim

remove tokens from. With more heaps we could develop even more complex arrangements.

This game can be visualized using a generalization of a graph which allows multiple edges between vertices. We may model this variation of Nim to with a graph by letting the heaps be edges, and associated heaps incident to some common vertex. Then on their turn each player selects a vertex and then removes any number of edges incident to that vertex. This game is called Graph Nim. An example of a position of this game is given in Figure 2.2.

This generalization of Nim is similar to another generalization of Nim called Circular Nim[3]. In Circular Nim the heaps are arranged in a circle and, for a given k , a player may select k consecutive

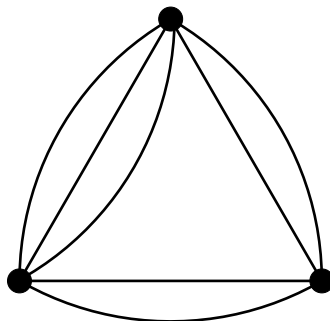


Figure 2.2: An Example of Graph Nim

heaps to remove tokens from. Graph Nim differs from this in that the arrangement of the heaps need not be circular and the number of heaps a player may remove tokens from is given by the adjacencies of the graph. Graph Nim is not easily solved with the use of the Sprague-Grundy Theorem (which will be discussed later), but we use arguments that rely on the structural properties of the graphs. However, winning strategies and losing positions must be discovered on a graph-by-graph basis.

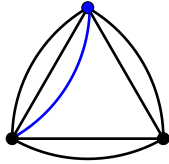
2.2

Graph Nim on a 3-vertex Graph

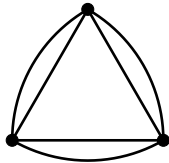
We will begin with Graph Nim on a 3-vertex graph, as shown in Figure 2.2. The winning strategy and losing position for this graph have been discovered at a Research Experience for Teachers at the University of Colorado-Denver. It is also commonly assigned as a homework problem in Dr. David Brown's Discrete Mathematics course, but this winning strategy and losing position have yet to be published, and will therefore be presented here.

Before presenting the losing position of 3-vertex Graph Nim, we will first present a detailed example of how this game is played for clarity. Let the starting position of this example game be the graph shown in Figure 2.2.

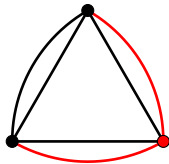
The first player begins by selecting a vertex and now can delete any edges incident to that vertex. Suppose the top vertex is selected and one edge is removed.



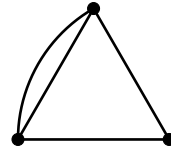
Now the second player has this new graph to play on:



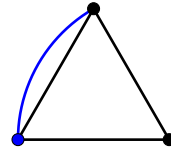
So the second player selects a vertex and can delete any edges incident to that vertex.



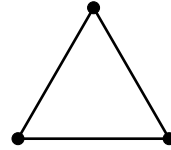
This gives Player 1 the following graph to play on:



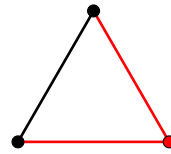
So Player 1 selects the bottom left vertex and deletes an edge incident to it.



Then Player 2 has this graph to play on:



They select the following vertex and delete two edges incident to it.



Player 1 plays on this graph and can simply remove the last edge and therefore wins.

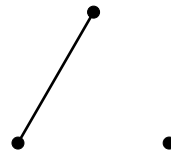


Figure 2.3: An Example of How Graph Nim is Played

This example also shows the winning strategy for Graph Nim on a 3-vertex graph. When the game was reduced to a triangle Player 2 had no way to win. No matter what vertex Player 2 had selected they could only remove 1 or 2 edges. In either case all edges left in the game are incident to the same vertex and therefore Player 1 can win. This leads to the following theorem.

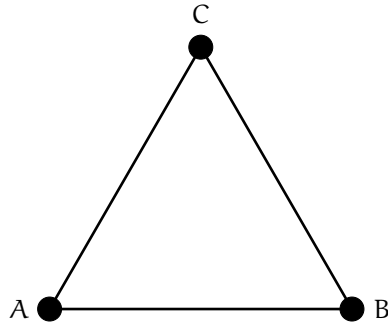


Figure 2.4: A Losing Position for Graph Nim on 3 Vertices

Theorem. *The losing position of Graph Nim played on 3-vertex graph is any 3-vertex graph such that there are an equal number of edges incident to every vertex.*

This theorem will be proved shortly, but first, some notation will be introduced. When playing Graph Nim on a 4-vertex graph it can be cumbersome to draw multiple edges between the vertices, as seen in Figure 2.5, therefore we will introduce a new way to represent the heaps.

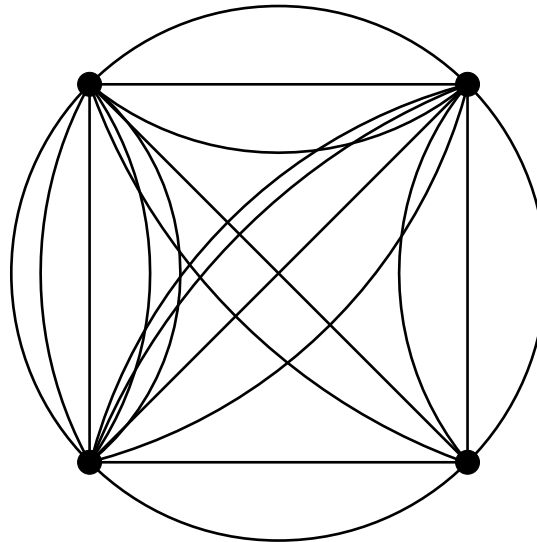


Figure 2.5: Graph Nim on a 4-vertex Graph Using the Multiple Edge Representation.

Definition. Define $w(x, y) \in \mathbb{N}$ to be the weight on the edge incident to the vertices x and y .

Now Graph Nim will be played as follows; on their respective turns each player selects a vertex and can reduce the weight of any number of edges incident to that vertex by non-negative integer amounts. On their turn a player must reduce the weight of at least 1 edge by at least 1, once an

edge is weighted zero it can no longer be played. The winning player is the player that reduces the last edge to a weight of zero. A position of Graph Nim is given by the function $p(G)$ that assigns weights to the edges of a graph, G . The losing position of a game will be denoted $\mathcal{Lp}(G)$. Also, let K_n denote the complete graph on n vertices, that is, the n -vertex graph with each pair of vertices adjacent. Now the previous theorem will be restated using this new language.

Theorem 1. $\mathcal{Lp}(K_3) = \{w(A, B), w(A, C), w(B, C) \mid w(A, B) = w(A, C) = w(B, C)\}$.

Proof. Examine the simplest losing position possible in the game, which is when all the edges are weighted 1. If all of the edges are weighted 1 on Player 1's turn they must pick a vertex, and can reduce the weight of either one or two edges. If Player 1 reduces the weight on two edges to zero, then there is only one left for Player 2 to reduce to zero. If Player 1 reduces only one edge weight to zero then the resulting graph will have a vertex that is incident to 2 edges and two vertices incident to 1 edge, Player 2 simply has to pick the vertex incident to 2 edges and can then reduce the weight of the final two edges to zero. By the same argument, if all the edges have weight 2 then Player 2 can either force all the edges to have weight 1 or win the game. Since on any move a player can reduce edge weights by any integer value greater than or equal to 1, any game position in which all edges have equal weight will also be a losing position. \square

2.3

Graph Nim on a 4-vertex Graph

The winning strategy and losing position for every 4-vertex graph will be presented in this section. Determining how many unique graphs exist for a given number of vertices is an interesting problem in and of itself and requires group theoretic results such as Burnside's Lemma and Pólya's Theorem[6]. A combinatorial argument gives the number of labeled graphs for a given number of vertices. First, determine the number of possible edges. Consider the maximum number of edges in a graph with n vertices, select a vertex, this vertex can be adjacent to at most $n - 1$ other vertices. Now there are n vertices that can be adjacent to $n - 1$ other vertices, but since each edge is incident to exactly 2 vertices every edge is counted twice, therefore the maximum number of edges in a graph with n vertices is given by

$$m = \frac{n(n - 1)}{2}.$$

Now, to count the number of labeled graphs consider that each edge may or may not be included, this means that for every edge there are exactly two options (include the edge or don't). Therefore,

the number of labeled graphs with n vertices is given by

$$2^m.$$

For Graph Nim the labeling of a graph is not important, although the graphs will be labeled for notation's sake. For example, Figure 2.6 shows two graphs that are labeled differently, and are therefore counted as different graphs in the counting method previously described. However, if $w(A, D)$ in graph G is equal to $w(B, D)$ in graph H and $w(B, C)$ in graph G is equal to $w(A, C)$ in graph H, then they are identical positions of Graph Nim. Therefore, the number of distinct unlabeled graphs is needed.

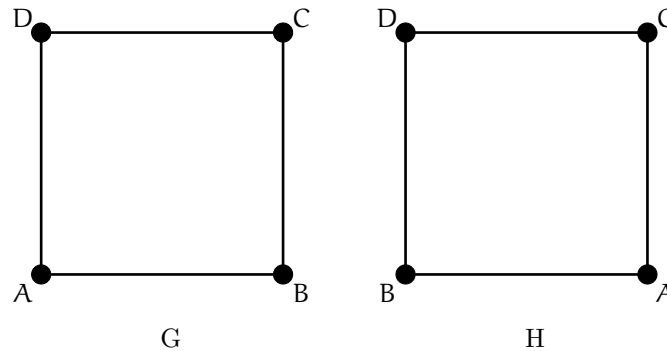


Figure 2.6: Example Showing that Labeling is not Important for Graph Nim

Frank Harary and Edgar M. Palmer describe the process of applying Pólya's Theorem to counting unlabeled graphs[6]. This process (which will be omitted here for simplicity) removes all duplicate graphs when the labeling is removed, and gives the number of distinct unlabeled graphs with 4 vertices by the polynomial

$$g_4(x) = 1 + x + 2x^2 + 3x^3 + 2x^4 + x^5 + x^6.$$

One interesting thing about this polynomial is that the coefficient of x^m gives the number of graphs with exactly m edges, and evaluating the polynomial at $x = 1$ is simply summing the coefficients, and hence gives the number of different graphs on 4 vertices.

By the given polynomial there are $g_4(1) = 11$ distinct graphs with 4 vertices and they are shown in Figure 2.7. In this section a winning strategy will be given for each of these 11 graphs. These 11 graphs will be referred to by their common graph theoretic names, and therefore some notation will be introduced. Recall from the last section that a complete graph with n vertices (the

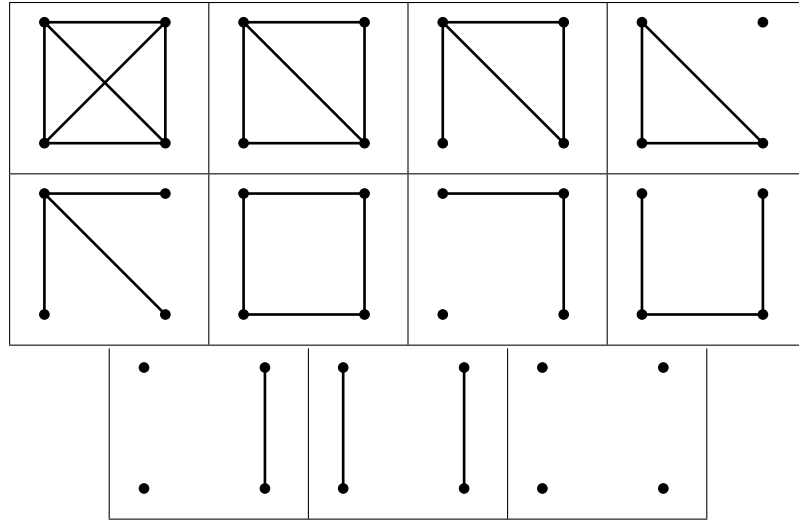
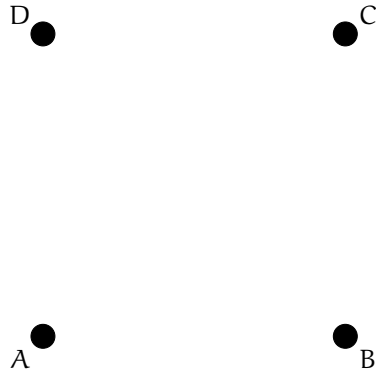


Figure 2.7: Every Distinct 4-vertex Graph

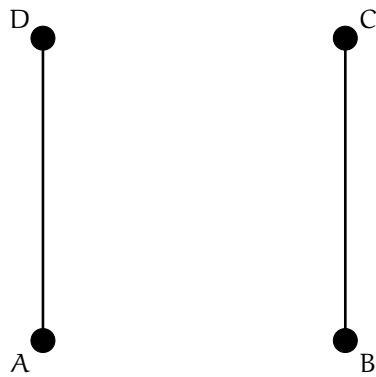
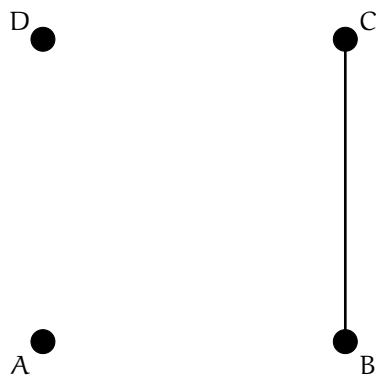
graph in which each vertex is adjacent to every other vertex) is denoted K_n . A graph G is a *path* on n vertices, denoted P_n , if it contains n vertices which can be ordered so that two vertices are adjacent if and only if they are consecutive in the ordering. A graph G is a *cycle* on n vertices, denoted C_n , if it contains n vertices and n edges and whose vertices can be placed around a circle so that two vertices are adjacent if and only if they appear consecutively along the circle. The complement of a graph G , denoted \overline{G} , is a graph with the same vertex set as G , but two vertices are adjacent in \overline{G} if and only if they are not adjacent in G . The union of two graphs, G and H , denoted $G \cup H = (V(G) \cup V(H), E(G) \cup E(H))$, is the graph whose vertex set is the union of the vertex sets of G and H and the edge set is the union of the edge sets of G and H . The union of t copies of the same graph, G , is denoted tG . A graph G is *bipartite* if its vertex set is the union of two disjoint independent sets called *partite* sets of G . A *complete bipartite graph* is a bipartite graph such that two vertices are adjacent if and only if they are in different partite sets. When the partite sets in a complete bipartite graph have sizes m and n , we denote this graph by $K_{m,n}$ [11].

First, consider $\overline{K_4}$ which is depicted in Figure 2.8. Clearly, since this graph contains no edges Graph Nim cannot be played on this graph, it may be useful to think of this graph as the terminal position for every game of Graph Nim on any 4-vertex graph.

Now examine $2K_2$, depicted in Figure 2.9, and $K_2 \cup 2K_1$, depicted in Figure 2.10. For both of these graphs every vertex is incident to at most 1 edge, so when a player selects any vertex they can reduce at most 1 edge weight. This means that both of these graphs are simply Nim with 2 and 1 heap respectively, because a player can only reduce one edge weight at a time. The winning strategy for Nim was given in Chapter 1. A player would Nim-sum the weights of every edge to obtain the

Figure 2.8: $\overline{K_4}$

Nimber for the position, then they would calculate the Nim-sum of each edge with the Nimber in order to determine which edge weight to reduce and how much to reduce it.

Figure 2.9: $2K_2$ Figure 2.10: $K_2 \cup 2K_1$

Now consider $K_{1,2} \cup K_1$, depicted in Figure 2.11, and $K_{1,3}$, depicted in Figure 2.12. In both

of these graphs all of the edges are incident to a single vertex. Therefore, the first player can simply select the vertex that is incident to every edge and reduce the edge weight for every edge to 0. These positions can be won in a single move.

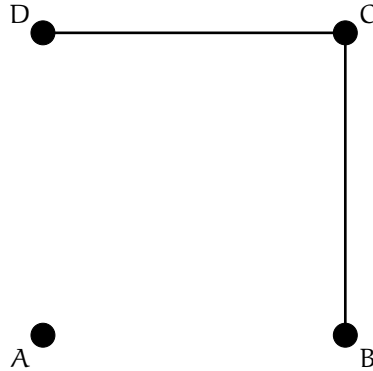


Figure 2.11: $K_{1,2} \cup K_1$

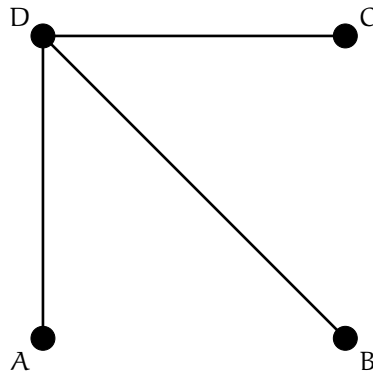
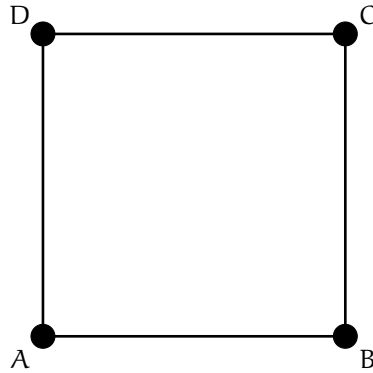


Figure 2.12: $K_{1,3}$

Now consider C_4 , depicted in Figure 2.13. This graph is nontrivial, so we prove the following theorem about its losing position.

Theorem 2. $\mathcal{Lp}(C_4) = \{w(A, B), w(B, C), w(C, D), w(D, A) \mid w(A, B) = w(C, D) \text{ and } w(B, C) = w(D, A)\}$, where the labeling is as in Figure 2.13

Proof. Examine the simplest losing position possible in the game, which is when all the edges are weighted 1. If all of the edges are weighted 1, on Player 1's turn they must pick a vertex, and can reduce the weight of either one or two edges. If Player 1 reduces the weight on two edges to zero, then there will be one vertex that is incident to the remaining 2 edges with weight greater than zero; Player 2 then simply has to select that vertex and reduce those 2 edges' weights to zero. If Player 1

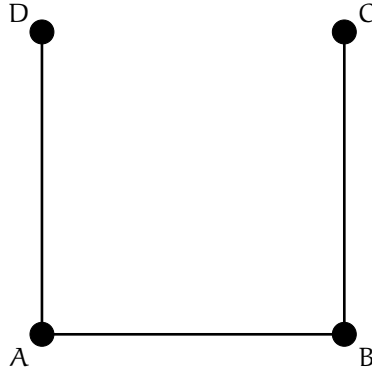
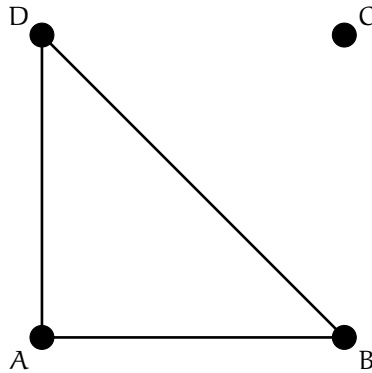
Figure 2.13: C_4

reduces only one edge weight to zero then there will be 2 vertices incident to 2 edges, and the other 2 vertices will be incident to 1 edge. Without loss of generality let A and B be the vertices incident to 2 edges. If Player 2 selects either A or B and sets $w(A, B) = 0$, then the resulting game will only consist of 2 edges which are not incident to the same vertex. Therefore, Player 1 must reduce 1 of the 2 edge weights to zero and Player 2 will be able to reduce the other edge weight to zero. Since on any move a player can reduce edge weights by any integer value greater than or equal to 1, any game position in which opposite edges have equal weight will also be a losing position (note: in a graph on 4 vertices, if an edge xy is incident to two vertices, x and y , then the opposite edge is the edge that is not incident to either x or y). \square

The winning strategy for Graph Nim on C_4 is to reduce the graph to the losing position. This can always be done: Player 1 simply has to select the vertex that is not incident to the 2 edges with the lowest weights and reduce the edges incident to that vertex to the same weight as the edge opposite it. If the initial position of the graph is the losing position, then Player 1 cannot win. This losing position was also found by M. Dufour and S. Heubach in the context of another variation of Nim called Circular Nim[3].

Now consider the strategy for Graph Nim on P_4 , depicted in Figure 2.14. This is a special case of $\mathfrak{p}(C_4)$ where one of the edges already has an edge weight of zero, therefore, the winning strategy for Graph Nim on this graph is given by the winning strategy for Graph Nim on C_4 . Without loss of generality let $w(A, D) > w(B, C)$, then Player 1 would simply select vertex A and reduce $w(A, B) = 0$ and $w(A, D) = w(B, C)$ the game is now in the losing position for Graph Nim on C_4 .

Now consider $K_3 \cup K_1$. This graph will have the same strategy as Graph Nim on K_3 because there is an isolated vertex. This strategy is discussed in the previous section.

Figure 2.14: P_4 Figure 2.15: $K_3 \cup K_1$

Next consider $\overline{K_{1,2} \cup K_1}$, which is depicted in Figure 2.16. None of the winning strategies that have been presented thus far apply to this Graph Nim on this graph, and so we prove the following nontrivial theorem

Theorem 3. *Without loss of generality $\mathcal{Lp}(\overline{K_{1,2} \cup K_1}) = \{w(A, D), w(D, C), w(D, B), w(C, B) \mid w(C, B) > w(A, D), w(D, C) \text{ and } w(D, B)\}$.*

Proof. If $w(C, B) > w(A, D), w(D, C)$ and $w(D, B)$ then Player 1 cannot reduce the game to $\mathcal{Lp}(K_3)$ or $\mathcal{Lp}(C_4)$ and therefore cannot win the game on the first move. \square

The winning strategy for $\overline{K_{1,2} \cup K_1}$ is determined by $\mathcal{Lp}(C_4)$. If the graph is labeled as in Figure 2.16 Player 1 will select vertex D, and reduce $w(D, C)$ and $w(D, B)$ to zero. Player 1 will also reduce $w(A, D)$ such that $w(A, D) = w(C, B)$. This will result in $\mathcal{Lp}(C_4)$ for Player 2 to play on. This strategy has the condition that $w(A, D) \geq w(C, B)$. If this condition is not met, there is an alternative strategy based on $\mathcal{Lp}(K_3)$. In this strategy Player 1 will again select vertex D, then reduce $w(A, D), w(D, B)$, and $w(D, C)$ such that $w(A, D) = 0$ and $w(D, C) = w(D, B) = w(C, B)$. This will

result in $\mathcal{Lp}(K_3)$ for Player 2 to play on. This strategy has the conditions that $w(D, B) \geq w(C, B)$ and $w(D, C) \geq w(C, B)$.

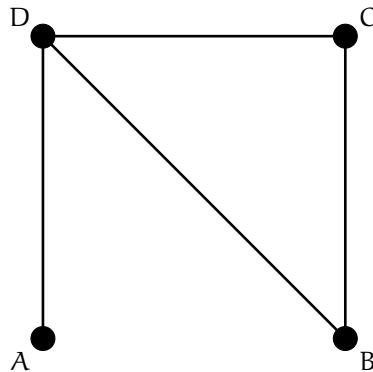


Figure 2.16: $\overline{K_{1,2} \cup K_1}$

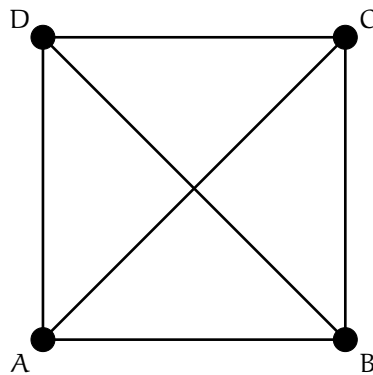
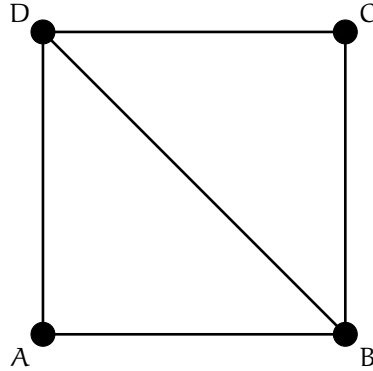


Figure 2.17: K_4

The winning strategy for Graph Nim on K_4 is based on $\mathcal{Lp}(\overline{K_{1,2} \cup K_1})$. Player 1 begins by finding the maximum edge weight in $\mathfrak{p}(K_4)$ the edge with this weight will be incident to 2 vertices, x and y ; Player 1 should select one of the 2 vertices, a and b , that this edge is not incident to. Without loss of generality suppose Player 1 selects vertex a . Player 1 should then reduce $w(a, x)$, $w(a, y)$ and $w(a, b)$ such that $w(a, x) = w(a, y) = 0$ and $0 < w(a, b) < w(x, y)$. Unless $w(x, y) = w(b, x) = w(b, y)$ the game will be reduced to $\mathcal{Lp}(\overline{K_{1,2} \cup K_1})$ for Player 2 to play on. This strategy will always work unless $w(x, y) = w(b, x) = w(b, y) = w(a, x) = w(a, y)$, but if $w(x, y) = w(b, x) = w(b, y) = w(a, x) = w(a, y)$ then Player 1 can select any vertex and remove all incident edges, the resulting position will be $\mathcal{Lp}(K_3)$.

The winning strategy for Graph Nim on $\overline{K_2 \cup 2K_1}$, depicted in Figure 2.18, is the same as the winning strategy for Graph Nim on K_4 , but with an extra condition. Again Player 1 begins by

Figure 2.18: $\overline{K_2 \cup 2K_1}$

finding the maximum edge weight as in the strategy presented for Graph Nim on K_4 the edge with this weight will be incident to 2 vertices, x and y , and not incident to 2 vertices, a and b . Because $\overline{K_2 \cup 2K_1}$ only has 5 edges there is the possibility that $w(a, b) = 0$ in which case Player 1 cannot reduce the game to $\mathcal{Lp}(\overline{K_{1,2} \cup K_1})$. If this is the case then Player 1 can only win if the game can be reduced to $\mathcal{Lp}(C_4)$ or $\mathcal{Lp}(K_3)$.

The losing positions and winning strategies for all 4-vertex graphs have now been developed. Some of these strategies and observations can be applied to graphs with more than 4 vertices.

2.4

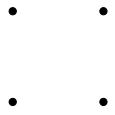
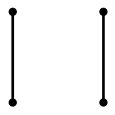
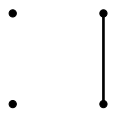
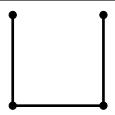
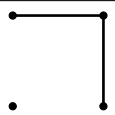
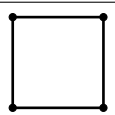
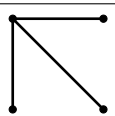
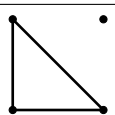
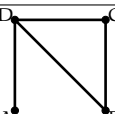
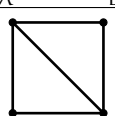
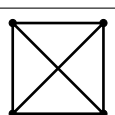
Graph Nim on Other Graphs

As mentioned above, some of the losing positions and winning strategies can be applied to graphs with more than 4 vertices. For example, in the last section it was observed that Graph Nim on $2K_2$ was essentially Nim with 2 heaps. This can be generalized to any disjoint union of K_2 's, because a player can only reduce the weight of a single edge weight in any given move. Also, Graph Nim on $K_{1,2} \cup K_1$, and $K_{1,3}$ could be won in a single move, because all the edges are incident to a single vertex. This can also be generalized to any graph where all the edges are incident to a single vertex, which can be denoted $K_{1,n}$.

Since all the edges in a $K_{1,n}$ are incident to a single vertex we can think of the entire graph as a single heap in a game of Nim. This leads to another generalization of the winning strategies discussed earlier. Since $K_{1,n}$ can be thought of as a single heap then Graph Nim on any disjoint unions of $K_{1,n}$ is essentially Nim. Therefore, a winning strategy for Graph Nim played on $K_{1,n} \cup K_{1,m} \cup \dots \cup K_{1,r}$ has been developed. Notice that a K_2 can also be denoted $K_{1,1}$, and therefore a disjoint union of

K_2 's is a special case of a disjoint union of complete bipartite graphs where one of the partite sets has size 1.

Table 2.1: A Summary of the Winning Strategies for Graph Nim on 4-vertex Graphs.
Summary of Strategies

	Trivial
	Equivalent to Nim
	Equivalent to Nim
	Reduce to $\mathcal{Lp}(C_4)$ by reducing edge weights of opposite edges to be equal
	Equivalent to Nim
	Reduce to $\mathcal{Lp}(C_4)$ by reducing edge weights of opposite edges to be equal
	Equivalent to Nim
	Reduce to $\mathcal{Lp}(K_3)$ by reducing all edge weights to be equal
	Reduce to $\mathcal{Lp}(\overline{K_{1,2} \cup K_1}) = \{w(C, B) > w(A, D), w(D, C) \text{ and } w(D, B)\}$
	Reduce to $\mathcal{Lp}(K_3)$, $\mathcal{Lp}(C_4)$, or $\mathcal{Lp}(\overline{K_{1,2} \cup K_1})$ depending on edge weights
	Reduce to $\mathcal{Lp}(K_3)$, $\mathcal{Lp}(C_4)$, or $\mathcal{Lp}(\overline{K_{1,2} \cup K_1})$ depending on edge weights

2.5

Sprague-Grundy Theorem and Graph Nim

Graph Nim is an impartial combinatorial game and as such The Sprague-Grundy Theorem applies. Recall from the Introduction:

Theorem (Sprague 1935, Grundy 1939). *Every impartial combinatorial game under the normal play convention is equivalent to a Nimber.*

The application of The Sprague-Grundy Theorem to Graph Nim will be presented here in detail.

Sprague-Grundy and 3-vertex Graph Nim

First examine The Sprague-Grundy Theorem applied to a graph with 3 vertices and edge weights restricted to be less than or equal to 1. Then the possible positions are simply the number of graphs with 3 vertices which is 4[6]. These positions are depicted in Figure 2.19.

The Sprague-Grundy Graph for these possible positions is the graph depicted in Figure 2.20. The careful observer might notice that this is the graph that was used as an example in the introduction, therefore the Grundy number associated with each vertex is already known.

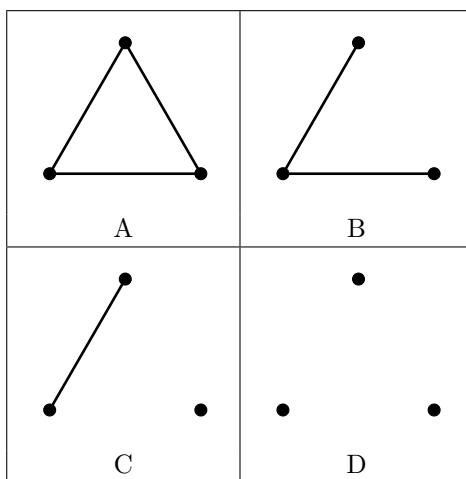


Figure 2.19: All Possible Positions of 3-vertex Graph Nim when Edge Weights are Less Than or Equal to 1

This confirms the losing position stated in Theorem 1, but the edge weights have been restricted to be less than or equal to 1. Now examine The Sprague-Grundy Graph with the edge weights less than or equal to 2. Then the number of possible positions has been expanded. Figure 2.21 shows the new positions that are possible if the edge weights are less than or equal to 2.

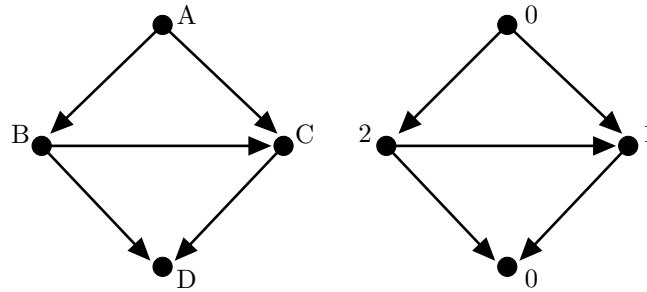


Figure 2.20: Sprague-Grundy Graph of 3-vertex Graph Nim when Edge Weights are Less Than or Equal to 1

Now adding these new positions into our Sprague-Grundy Graph, depicted in Figure 2.22, the graph has become a lot more complicated. Six vertices and 22 arcs were added to the graph, but through labeling the graph and using the mex function to calculate the Grundy Numbers, shown in Figure 2.23, there is only added one new vertex with Grundy Number zero, that is position E. This, also, confirms Theorem 1.

The Sprague-Grundy Graph confirms our losing position. Now an example of how this graph informs our winning strategy will be shown. Start with the labeled Sprague-Grundy Graph if the starting position is the graph in Figure 2.2.

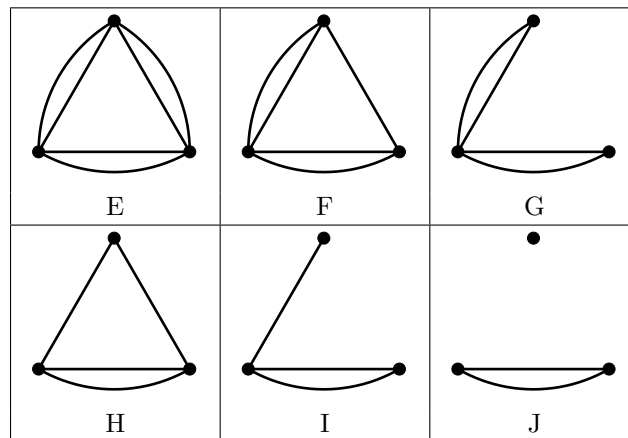


Figure 2.21: New Positions that are Possible if the Edge Weights are Less Than or Equal to 2 in 3-vertex Graph Nim.

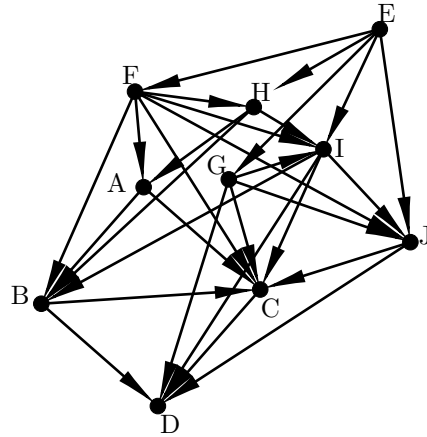


Figure 2.22: Sprague-Grundy Graph of 3-vertex Graph Nim when Edge Weights are Less Than or Equal to 2

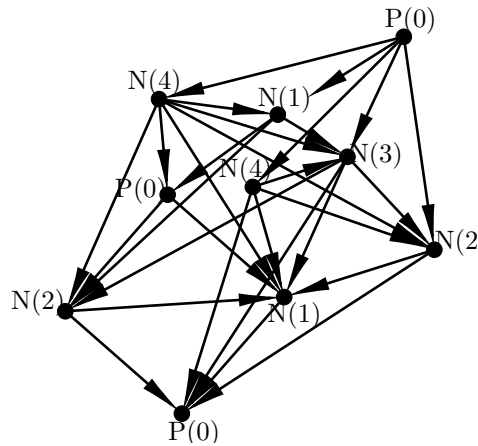


Figure 2.23: Labeled Sprague-Grundy Graph of 3-vertex Graph Nim when Edge Weights are Less Than or Equal to 2

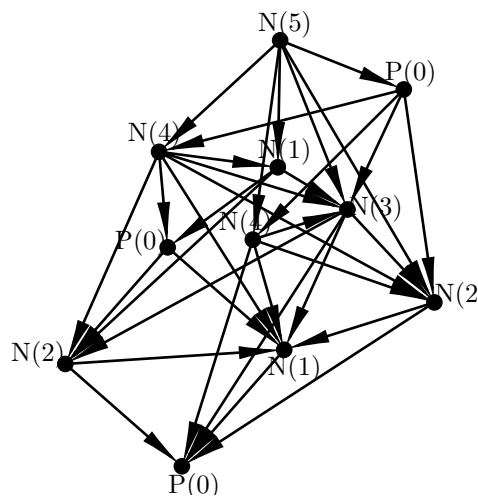
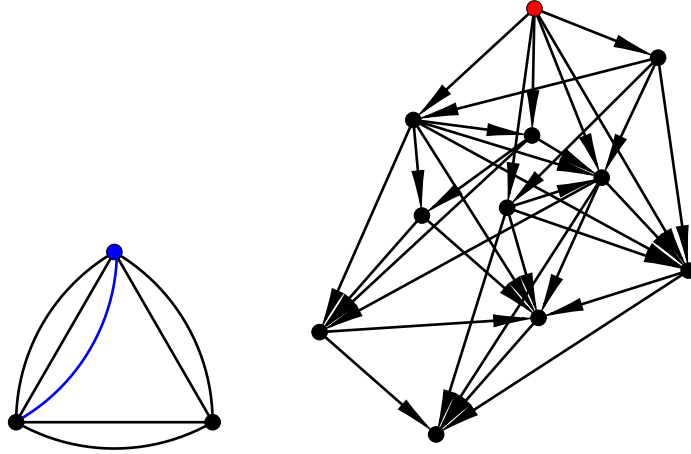


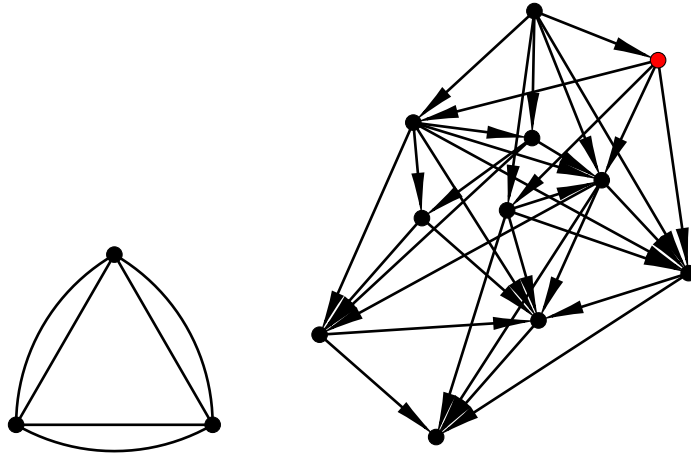
Figure 2.24: Labeled Sprague-Grundy Graph for the Starting Position in Figure 2.2

We will walk through the example in Figure 2.3, but we will keep track of where we are on the Sprague-Grundy Graph. We will depict what vertex we are at by coloring the vertex red.

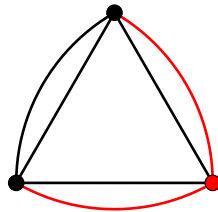
The first player begins by selecting a vertex and now can delete any edges incident to that vertex



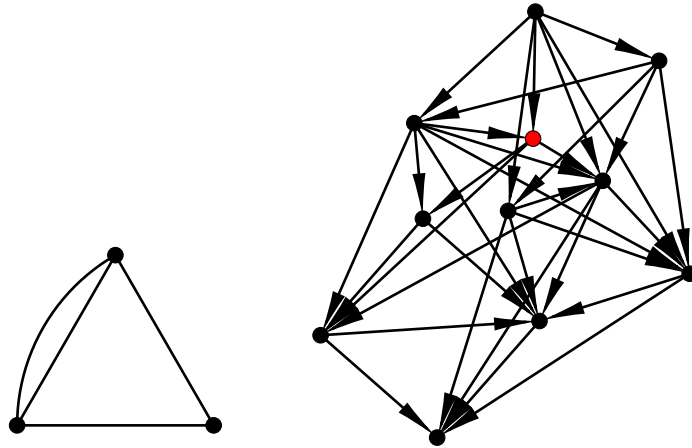
Now the second player has this new graph to play on



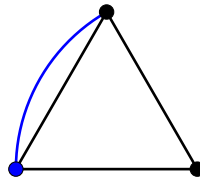
So the second player selects a vertex and can delete any edges incident to that vertex



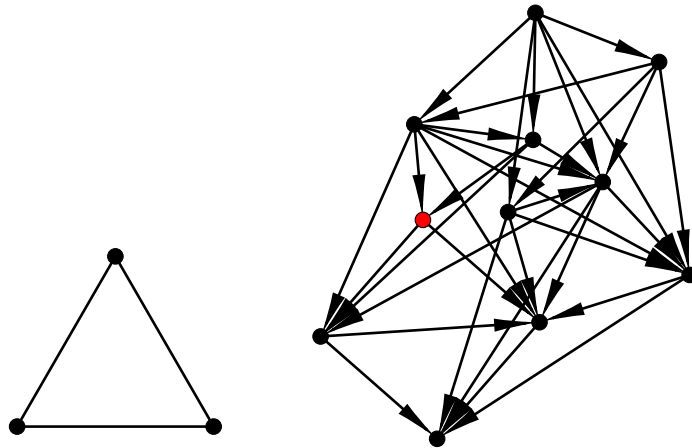
This gives Player 1 the following graph to play on



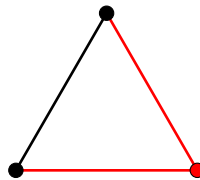
So Player 1 selects the following vertex and deletes an edge incident to it



Then Player 2 has this graph to play on



They select the following vertex and delete two edges incident to it



Player 1 plays on this graph and can simply remove the last edge and therefore wins.

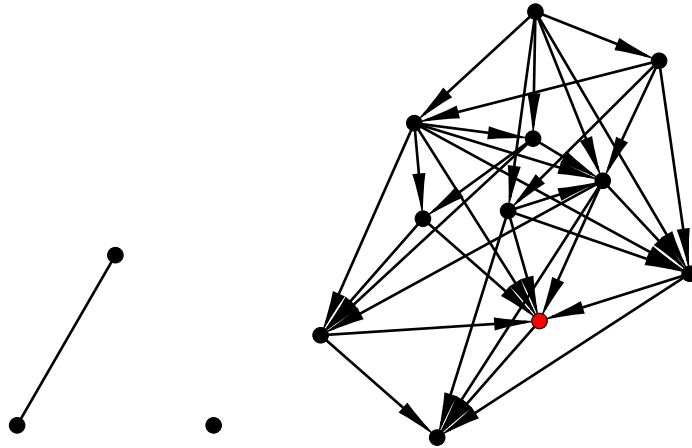


Figure 2.25: An Example of Game Play and the Sprague-Grundy Graph

Player 1 used the winning strategy in this example. The Grundy Numbers weren't shown in the example, but referencing Figure 2.24. Player 1 always moved the game to a “P” position, or a position with Grundy Number 0, and Player 2 was always forced to move to a “N” position. This strategy allowed Player 1 to win regardless of Player 2's moves. This also helps illustrate why if the game starts in a losing position the game is Player 2 optimal.

Sprague-Grundy and 4-vertex Graph Nim

Now do the same thing for 4-vertex Graph Nim. Let the edge weights be restricted to be less than or equal to 1. Then the possible positions are simply the number of graphs with 4 vertices which is 11[6]. These positions are depicted in Figure 2.26. The Sprague-Grundy Graph for these positions is the graph in Figure 2.27.

Labeling the graph and assigning Grundy Numbers with the mex function, the graph in Figure 2.28 is obtained. The graph shows that positions D, F, and J are losing positions which confirm the findings in Theorem 1 and Theorem 2.

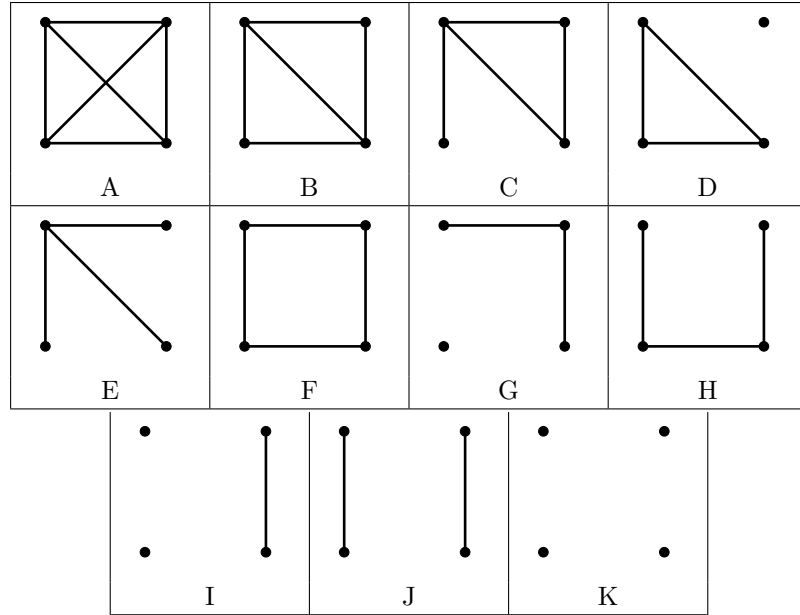


Figure 2.26: All possible positions of 4-vertex Graph Nim when edge weights are less than or equal to 1

Conclusion

It is possible to use The Sprague-Grundy Theorem on Graph Nim. Examples have been shown for 3 and 4-vertex Graph Nim. The problem is that The Sprague-Grundy Graphs are huge and therefore are not practical. Arguments can be made about the adjacency in these graphs to deduce which vertices have to be a “P” vertex and have Grundy Number zero, but they would be similar to the arguments made without using The Sprague-Grundy Theorem.

2.6

Future Work

The winning strategies and losing positions for Graph Nim on 3 and 4-vertex graphs have been discovered, but there are many more graphs to work on. The obvious next step is to find winning strategies and losing positions for Graph Nim on 5-vertex graphs. Any 5-vertex graph with an isolated vertex is equivalent to a position of Graph Nim on 3 or 4 vertices. Also, any 5-vertex graph that is a disjoint union of complete bipartite graphs where one of the partite sets is size 1 has been solved. This takes care of a lot of the 5-vertex graphs, but doesn’t solve them all. The Sprague-Grundy Theorem might be useful for discovering the “base” losing positions. The Sprague-Grundy Graph for 5-vertex Graph Nim when edge weights are less than or equal to 1 could be built,

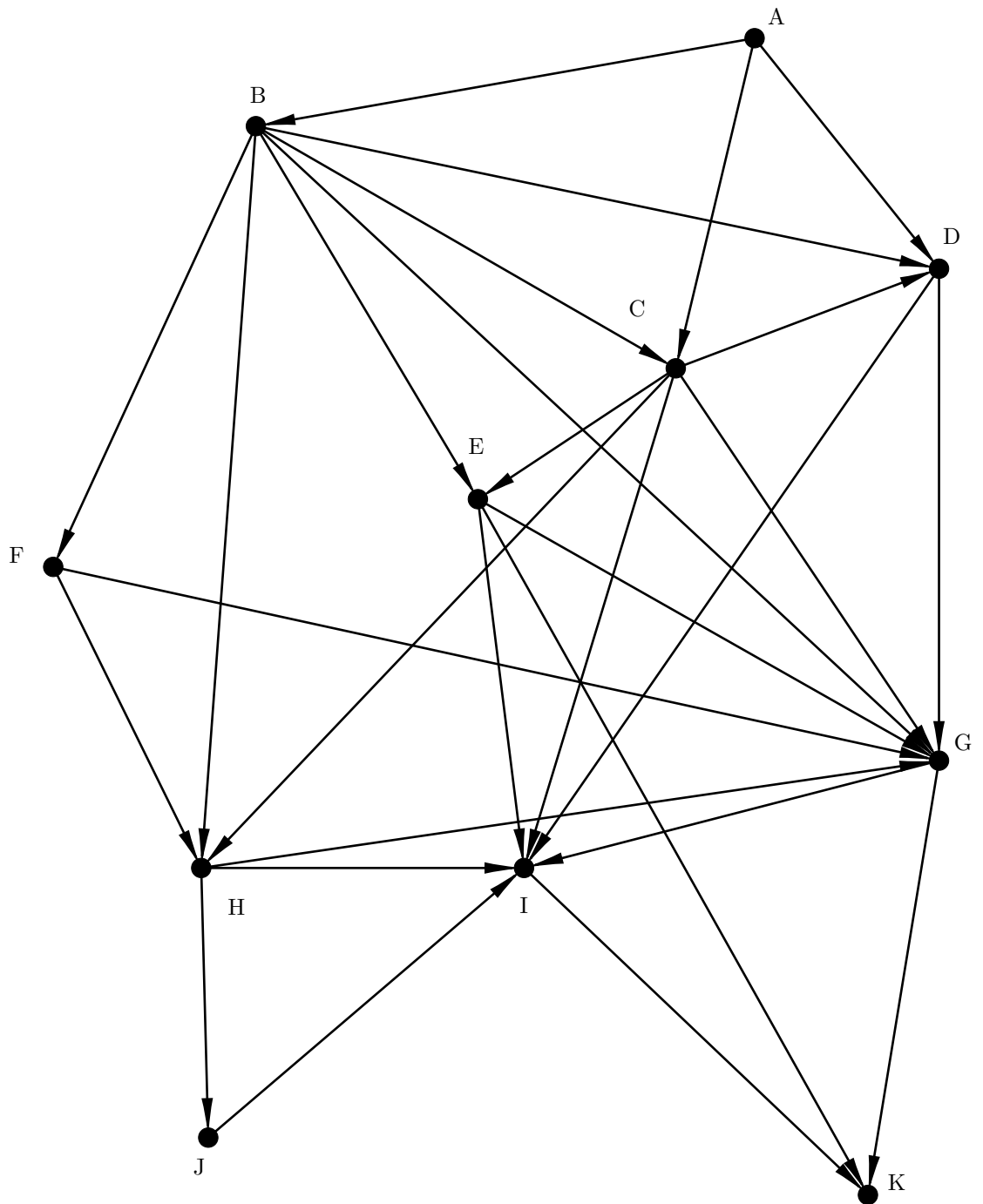


Figure 2.27: Sprague-Grundy Graph of 4-vertex Graph Nim when Edge Weights are Less Than or Equal to 1

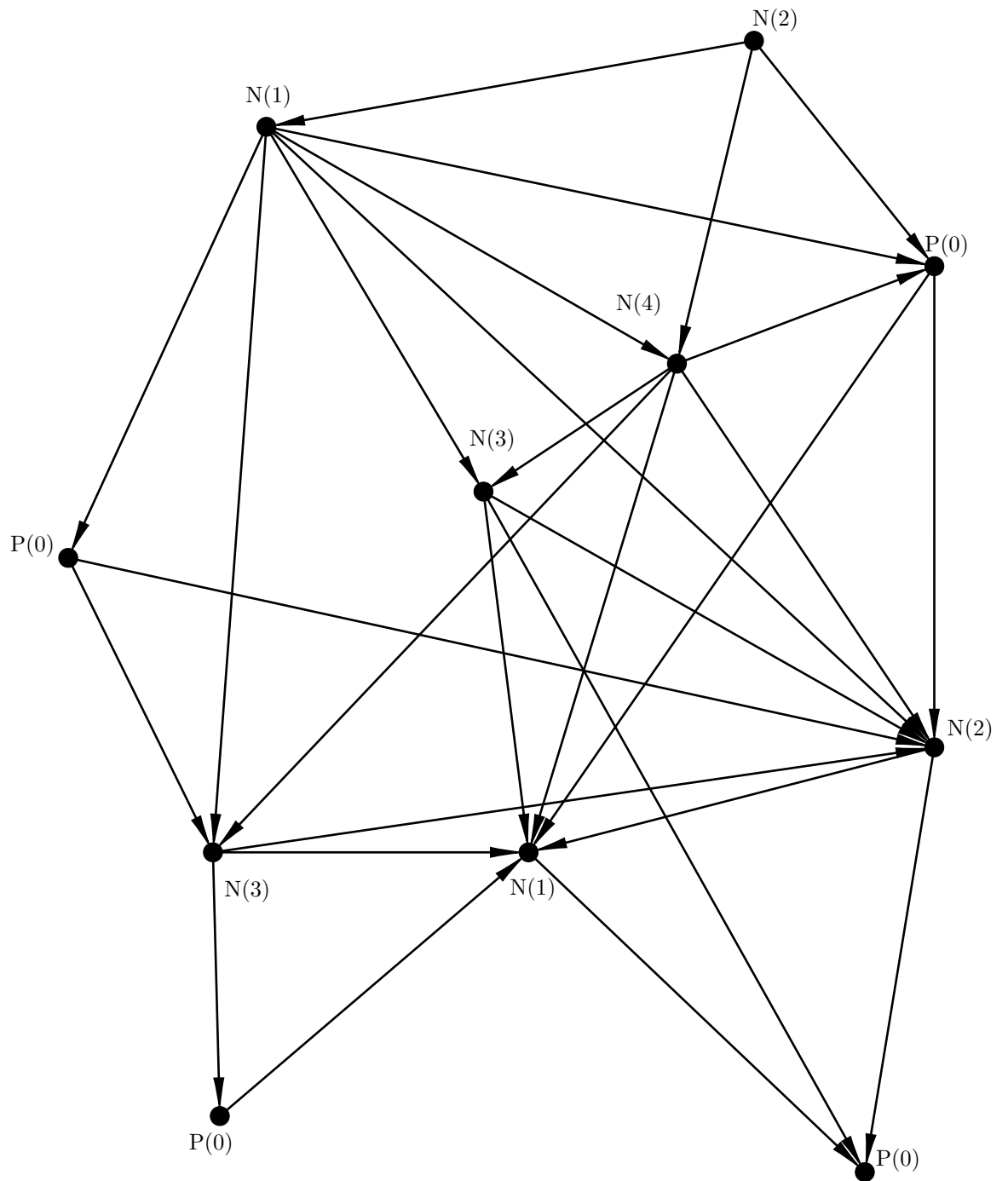


Figure 2.28: Labeled Sprague-Grundy Graph of 4-vertex Graph Nim when Edge Weights are Less Than or Equal to 1

and used to find the “P” positions. This would be difficult, however, because there are 34 possible positions, even under the edge weight restriction. The Sprague-Grundy Theorem can also be applied to Circular Nim to confirm and extend the results of Dufour and Heubach.

There are also variations of the game that might be interesting to pursue. In standard Graph Nim the weighting of our edges was restricted to non-negative integers, but variations of the game could be played with other weight schemes, for example:

- Integer weighting and on a player’s move they must choose a vertex and an operation (either addition or subtraction) then the player can add or subtract any integer amount from any edge incident to that vertex.
- Rational weighting and on a player’s turn they must pick a vertex and must multiply the weights of any edges incident to the vertex by some fixed integer that they can split any way they see fit. Instead of reducing the weights to zero the goal is to get the weights to 1.

CHAPTER 3

GEOGEBRA

My research on graph games led to a search for a program that could build and manipulate graphs in an automated fashion. GeoGebra suits those needs perfectly. While this was invaluable to this research there is also great potential in this tool for educators.

Graph Theory is a markedly visual field of mathematics. It is extremely useful for graph theorists and students to visualize the graphs they are studying. There exists software to visualize and analyze graphs, such as SAGE, but it is often extremely difficult to learn how use such programs. The tools in GeoGebra make pretty graphs, but there is no automated way to make a graph or analyze a graph that has been built. Fortunately GeoGebra allows the use of JavaScript in the creation of buttons which allow us to build useful Graph Theory tools in GeoGebra. We will discuss two applets that were created to serve as an example of how GeoGebra can be used to help students learn some of the basics of Graph Theory.

Remark. The JavaScript code used to generate these applets is displayed along with the text. In each piece of code there are comment lines meant to inform the reader how each piece of code is accomplishing our ultimate goal.

3.1

Graphs and Adjacency Matrices

There are many ways a graph can be represented, so far we have been looking at visual representations of graphs. Another way a graph can be represented is by an adjacency matrix. Let G be a graph on $V(G) = \{v_1, \dots, v_n\}$. The adjacency matrix of G is the $(0, 1)$ -matrix $A = [a_{ij}]$ with $a_{ij} = 1$ if and only if $\{v_i, v_j\} \in E(G)$. Figure 3.1 shows an example of a graph and its adjacency matrix.

The first Applet creates a random graph with 1 to 10 vertices and will generate the corresponding adjacency matrix. This applet consists of two JavaScript buttons. The first creates

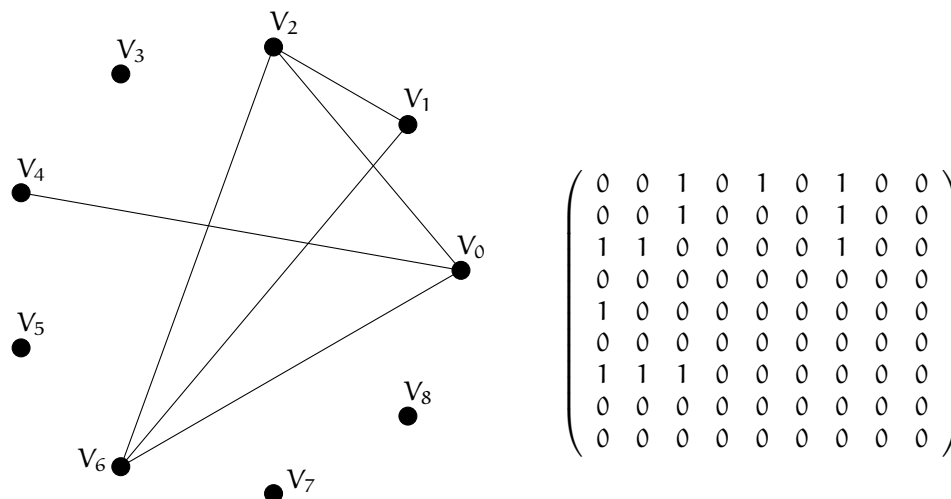


Figure 3.1: A Graph and its Adjacency Matrix

the random graph. The Matrix is created using a separate button so that students can practice converting a graph to an adjacency matrix and check.

Generating the Graph

In order for this applet to be useful students need to be able to use it repeatedly, therefore the first thing the button must do is delete any previous graphs and adjacency matrices.

```

1 for (var i=0; i < 11; i++)
2 {
3   ggbApplet.deleteObject("V_{"+i+"}");
4 }
5 ggbApplet.deleteObject("text1");
6 ggbApplet.deleteObject("text2");

```

Figure 3.2: Deleting Previous Graphs and Matrices

Next a random number between 1 and 10 is created. This will be the number of vertices in the graph. This number will need to be stored in GeoGebra so that it can be used in the Matrix button. Then the vertices are created and polar coordinates are used to space them evenly.

```

1 //Generate a random number of vertices
2 var n= Math.floor((Math.random() * 10) + 1);
3
4 //Store the number of vertices in GeoGebra for other button

```

```

5 ggbApplet.evalCommand("n="+n)
6
7 //Create the vertices so that they are evenly spaced (using polar coordinates)
8 for(var k =0;k<n;k++) {
9 ggbApplet.evalCommand("V_{"+k+"}=(4;("+k+"*(360/"+n+")) )");
10 }
11 //Make all the points bigger
12 for(var m=0;m<n;m++){
13 ggbApplet.setPointSize("V_{"+m+"}", 5)
14 }

```

Figure 3.3: Creating a Random Number of Vertices

After creating the vertices randomly and getting them spaced nicely then edges are created randomly.

```

1
2 for (var q=0; q<n; q++){
3 //Generate a random number associated with vertex q.
4 var r= Math.random()
5 for(var w=0; w<n; w++){
6 //Generate a random number associated with vertex w.
7 var t = Math.random()
8 //If there is already an edge between the vertices, do nothing
9 if(ggbApplet.exists("e_{"+w+", "+q+"}")){
10 break;}
11 //If the edge doesn't exist do the following
12 else{
13 //Condition on the random numbers for vertices q and w to create random edges
14 if( r < .7 && t < .7){
15 //if w and q are the same vertex do nothing
16 if(w==q){
17 break;}
18 //if not make the edge
19 else{
20 ggbApplet.evalCommand("e_{"+q+", "+w+"}=Segment [V_{"+q+"},V_{"+w+"}] ");}}
21 else{
22 break;}}}}

```

Figure 3.4: Creating Random Edges

Generating the Matrix

Now that a graph has been created randomly the associated adjacency matrix needs to be created. This would not be possible in GeoGebra if JavaScript code had not been used to name the edge segments so that incidence was implied by the name of the edge. This code checks segment names to determine if two vertices are adjacent and builds a matrix.

```

1 //Get the number of vertices from geogebra
2 var n= ggbApplet.getValue("n");
3 //Create an empty string to put our matrix values in later
4 var matrix = ""
5 for (var i=0; i<n; i++){
6 //Create an empty array so that the edges can be given a value
7 var e=[]
8 for (var j=0; j<n; j++){
9 //If there is an edge between vertex i and j then make variable e[i,j]=1
10 if (ggbApplet.exists("e_{"+i+"+"+j+"}") || ggbApplet.exists("e_{"+j+"+"+i+"}")){
11 e[i,j] = "1"}
12 //Otherwise make the value of e[i,j]=0
13 else{
14 e[i,j]="0"}
15 }
16 //Make an empty variable so that the rows can be built
17 var row = []
18 //Make the first row
19 row[i]=e[i,0]
20 //Make the remaining rows and get them in the form that geogebra will use
21 for (var k=1; k<n; k++){
22 row[i]=row[i] + ","+ e[i,k]}
23 row[i]= "{" + row[i] +"}"
24 //Gather all our rows into one variable in the form geogebra will use
25 if(i==0){
26 matrix = matrix + row[i];}
27 else{
28 matrix = matrix +"," + row[i];}
29 }

```

Figure 3.5: Analyzing the Graph and Building the Matrix

Now that the Matrix has been created this code tells GeoGebra what to do with it and where to put it.

```

1 //Tell geogebra to make the matrix
2 ggbApplet.evalCommand("text1=FormulaText[{" + matrix + "}]")
3 //Make the matrix invisible
4 ggbApplet.setVisible("text1", false)
5 //Copy the matrix and tell geogebra where to place it
6 ggbApplet.evalCommand("text2=Text[ text1, (5.58, 1.7), false, true ]")

```

Figure 3.6: Placing the Matrix in GeoGebra

With the use of these two buttons a simple effective applet was created that might help students understand graphs and their adjacency matrices.

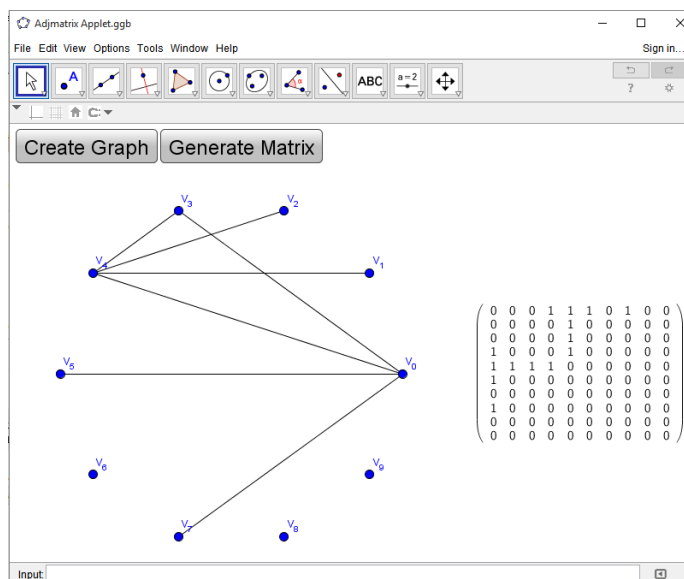


Figure 3.7: Adjacency Matrix Applet

3.2

Tournaments, Score, and Kings

This applet, presented now, was built to study an impartial combinatorial game called The Game of Thrones. The Game of Thrones is played on a specific class of graphs called tournaments. Tournaments are possibly the most well-known and heavily studied class of directed graphs. A *tournament* is a directed graph in which there is an arc between every pair of vertices; that is,

between any two vertices x and y either there is an arc from x to y , denoted $x \rightarrow y$, or an arc from y to x , denoted $y \rightarrow x$. If there is an arc from x to y we may say x *beats* y . A well-known application of a tournament in the sports world is known as a round robin tournament, whence the “beats” terminology. In a round robin tournament each team plays every other team with no ties. The nature of a tournament leads to an ambiguous interpretation of the most dominant vertex. For example, examine the tournament in Figure 3.8. If the vertices represent sports teams and the arcs represent games played between the teams, then it is unclear which team is the “best”. Team A and team D both won two games. Team A beat team D, however, team B, which was beaten by team D, beat team A.

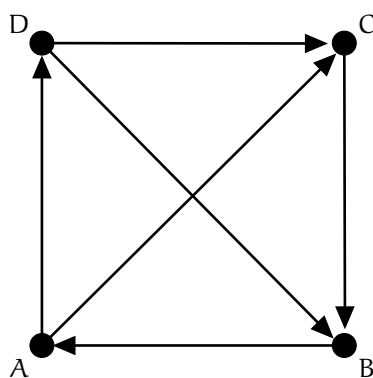


Figure 3.8: A Tournament with 4 Vertices.

In an attempt to determine the dominant vertices in a tournament mathematical sociologist H. G. Landau developed the notion of a king and proved that every tournament has at least one[7]. A *king* in a tournament is any vertex, v , such that for every other vertex, x , either $v \rightarrow x$, or there exists a vertex, y , such that $v \rightarrow y$ and $y \rightarrow x$. It is also valuable to note how many vertices a vertex beats. This is called the score of a vertex.

This applet can determine which vertices are kings and the score of each vertex. This applet was built so that data from the website of Brendan McKay of Australian National University can be used, <http://users.cecs.anu.edu.au/~bdm/data/digraphs.html>. Dr. McKay stores tournaments as a string of 1’s and 0’s. These strings are the upper triangle of the tournament’s adjacency matrix. Figure 3.8 is an example of a tournament and its adjacency matrix. Notice that the diagonal entries are all 0 and if the entry (i, j) is 0, then the entry (j, i) is 1. For example, there is a 0 in row 1 column 2, and there is a 1 in row 2 column 1. This is because the beats relationship in a tournament is anti-symmetric. If the upper triangle of the matrix is known then the entire matrix can be generated. Dr. McKay would store the tournament in Figure 3.8 as the string “011000”

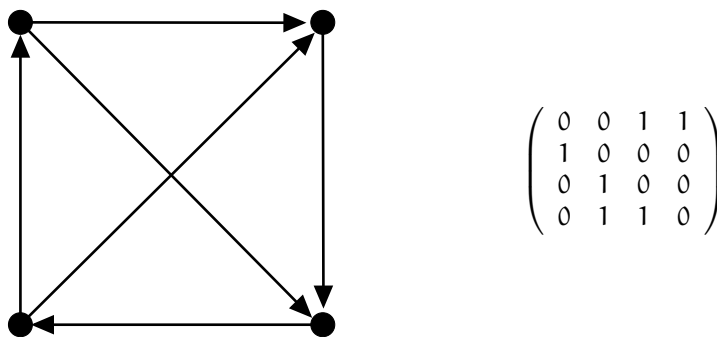


Figure 3.9: A Tournament with 4 Vertices and its Adjacency Matrix

which is simply the upper triangle of the matrix excluding the diagonal entries.

The applet consists of 4 buttons, an input bar, and instructions. The buttons are: Show Kings, Hide Kings, Show Scores, and Show Vertex Names. Each button and the input bar use JavaScript code to accomplish their tasks, each will be presented.

Input Bar

The input bar is used to copy and paste the tournament string and contains the code to build the tournament.

```

1 //Delete Previously Existing Tournament
2 for (var i=0; i < 50; i++){
3   ggbApplet.deleteObject("V_"+i+"");}
4 //Determine number of vertices from uppertriangle of adjacency matrix
5 var str= ggbApplet.getValueString("t");
6 var n = Math.floor((1 + Math.sqrt(8 * str.length() + 1)) / 2);
7 //Create n vertices evenly spaced and correct size
8 var v = 0;
9 for(var k =0;k<n;k++) {
10  ggbApplet.evalCommand("V_"+k+"=(31;("+k+"*(360/"n+")) )");}
11 for(var m=0;m<n;m++){
12  ggbApplet.setPointSize("V_"+m+", 5)}
13 //Use the uppertriangle of adajacency matrix to build relationships
14 for (var i=0; i < n; i++){
15  for (var j=i+1; j<n; j++){
16  var p=String(str.charAt(v))
17  if (p == 49){
18  //GeoGebra changes the 1 in the matrix to a 49. Don't know why.

```

```

19 ggbApplet.evalCommand("v_{"+i+"},"+j+"}=Vector[V_{"+i+"},V_{"+j+"}]");}
20 else{
21 ggbApplet.evalCommand("v_{"+j+"},"+i+"}=Vector[V_{"+j+"},V_{"+i+"}]");}
22 v++;
23 }
24 }

```

Figure 3.10: Input Bar Code

Show Kings

The Show Kings button analyzes the tournament and determines which vertices are kings, it then highlights those vertices. Recall that a king is a vertex, k , in a tournament, such that for every other vertex in the tournament, x , either $k \rightarrow x$ or there exists a vertex, y , such that $k \rightarrow y$ and $y \rightarrow x$.

```

1 //Find out how many vertices are in the tournament
2 var str= ggbApplet.getValueString("t");
3 var n = Math.floor((1 + Math.sqrt(8 * str.length() + 1)) / 2);
4 //For each vertex i does it beat each other vertex j?
5 //if not does it beat a vertex that does beat j?
6 function isKingOf(i, j){
7 if(ggbApplet.exists("V_{"+i+"}") == false){
8 return true;}
9 if(ggbApplet.exists("V_{"+j+"}") == false){
10 return true;}
11 if(i == j){
12 return true;}
13 if (ggbApplet.exists("v_{"+i+"},"+j+"}")){
14 return true;}
15 for (var k=0; k < n; k++){
16 if (ggbApplet.exists("v_{"+i+"},"+k+"}") && ggbApplet.exists("v_{"+k+"},"+j+"}"))
17 return true; }
18 return false;
19 }
20 function isKing(i){
21 for (var j=0; j < n; j++){
22 if (isKingOf(i, j) == false)
23 return false;
24 }

```

```

25 return true;
26 }
27
28 //If vertex i is a king change it's color to red
29 for (var i=0; i < n; i++){
30 if (isKing(i)) {
31 ggbApplet.setColor("V_{i+}",255,0,0);}
32 else{
33 ggbApplet.setColor("V_{i+}",0,0,255);}
34 }

```

Figure 3.11: Show Kings Button Code

Hide Kings

This button changes all the vertices back to the same color.

```

1 //Find out how many vertices are in the tournament
2 var str= ggbApplet.getValueString("t");
3 var n = Math.floor((1 + Math.sqrt(8 * str.length() + 1)) / 2);
4 //Turn all vertices blue
5 for(var i=0;i < n; i++){
6 ggbApplet.setColor("V_{i+}",0,0,255);
7 }

```

Figure 3.12: Hide Kings Button Code

Show Score

This button analyzes the tournament to determine the score of each vertex. The score of a vertex is the number of vertices it beats.

```

1 //Find out how many vertices are in the tournament
2 var str= ggbApplet.getValueString("t");
3 var n = Math.floor((1 + Math.sqrt(8 * str.length() + 1)) / 2);
4 //Find out how many other vertices each vertex beats
5 for(var i=0; i < n; i++){
6 var t = 0;
7 for (var j=0; j < n; j++) {

```

```

8  if (ggbApplet.exists("v_{"+i+"","+j+"}")) {
9  t = t + 1;}}
10 //Change the label of the vertex to be the score
11 ggbApplet.evalCommand("SetCaption[V_{"+i+"},\""+ t+"\"]");}

```

Figure 3.13: Show Score Button Code

Show Vertex Names

This button changes the visible labels on the vertices to their given names.

```

1 //Find out how many vertices are in the tournament
2 var str= ggbApplet.getValueString("t");
3 var n = Math.floor((1 + Math.sqrt(8 * str.length() + 1)) / 2);
4 //For each vertex change it's label to it's name
5 for(var i=0; i < n; i++){
6 ggbApplet.setLabelStyle("V_{"+i+"}", 0)}

```

Figure 3.14: Show Vertex Names Button Code

3.3

Conclusion

With the use of JavaScript, GeoGebra can be a very useful tool for visualizing and analyzing graphs and tournaments. This may be useful for students. These applets represent a small portion of what is possible in GeoGebra. The applets can be accessed here: <https://tube.geogebra.org/johndoe314?p=materials>, but must be downloaded as they do not function properly on the website.

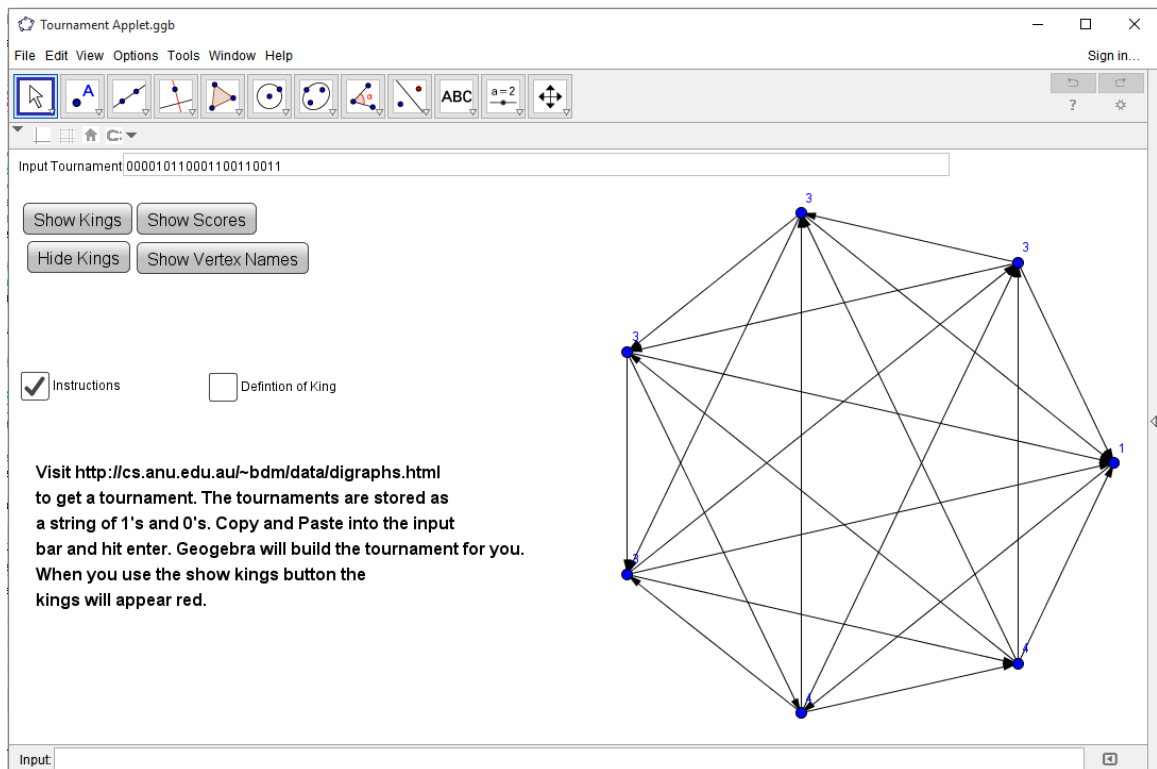


Figure 3.15: Tournament Applet

CHAPTER 4
THE GAME OF THRONES

Now we discuss an impartial combinatorial game played on tournaments, the directed graphs presented in the previous chapter. The Game of Thrones is an impartial combinatorial game played on a tournament. There are some classic tournament results that form the basis of The Game of Thrones. These results and their proofs will be presented here for completion.

Theorem. (*Landau 1951*) *Every tournament contains a king.*

Proof. Let T be a tournament. Let $x \in V(T)$ be a vertex with maximum score. Assume x is not a king, then there is some vertex y that beats x and there is no vertex that beats y and is beaten by x . Therefore, y beats every vertex that x beats, plus it beats x , so the score of y is strictly greater than the score of x . This is a contradiction, therefore, x is a king. \square

Definition. A vertex with score $n - 1$ is called a *source*.

Theorem. (*Maurer 1980*) *A tournament has exactly one king if and only if that king is a source.*

Proof. Let T be a tournament. Then if $x \in V(T)$ is a source we know that x beats every vertex, and no other vertex will satisfy the conditions of a king.

Now suppose $y \in V(T)$ is the only king, but is not a source. Then there is at least one vertex that beats y , by the proof of the previous theorem then we know that since y is beaten it is beaten by a king. This is a contradiction, therefore y is a source. \square

Definition. A *induced sub-tournament* of a tournament, T , is a tournament, H , that can be obtained by removing one or more vertices from T . This is denoted $H \trianglelefteq T$

These theorems and definitions are the basis for The Game of Thrones, a description of this game follows.

4.1

Game Play

The Game of Thrones is a two-player game played on a tournament. Players take turns removing vertices from the tournament. When a vertex is removed from the tournament the next player will play on the resulting induced sub-tournament. The game concludes when there is exactly one king remaining in the tournament. The winning player is the last player to move. An example game is shown in Figure 4.1, in each image the kings are colored light blue.

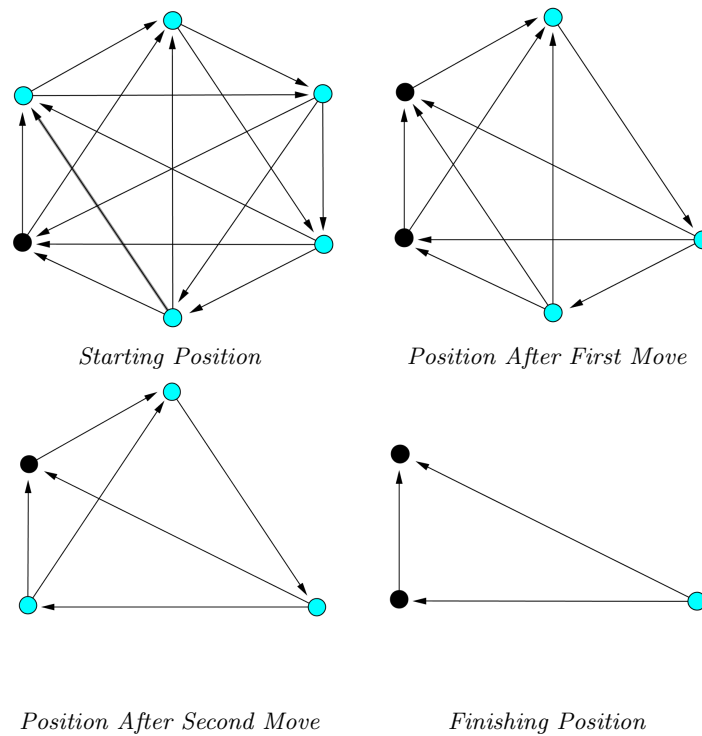


Figure 4.1: An Example Game in which the First Player Wins

4.2

A Winning Position

In order to win The Game of Thrones the induced sub tournament after your turn should contain a source, or a vertex with score $n - 1$ where n is the number of vertices in the current tournament. Therefore, if a tournament contains a vertex, x , of score $n - 2$ then x is only beaten by 1 vertex and can therefore become a source with the deletion of a single vertex, although, there may be more than one vertex of score $n - 2$ in a tournament.

Sarah Mousley, a student of Dr. David Brown, proved in an Undergraduate Honors Thesis that the maximum number of vertices with score m in a tournament with n vertices is given by a function $f(n, m)$.

Theorem 4 (Mousley, 2013[9]). *Let m, n be integers with $0 \leq m \leq n - 1$ and $n \geq 1$. Then*

$$f(m, n) = \begin{cases} 2m + 1 & \text{if } m \leq \frac{n-1}{2} \\ 2n - 2m - 1 & \text{if } m > \frac{n-1}{2} \end{cases}$$

This leads to the winning position of The Game of Thrones.

Theorem 5. *Any tournament with at least one vertex of score $n - 2$ is a winning position for the first player (Player 1).*

Proof. By Theorem 5, a tournament may have as many as 3 vertices of score $n - 2$. Now if a tournament only has a single vertex, x , of score $n - 2$ Player 1 simply has to delete the vertex that beats x . If a tournament has two vertices, x, y , of score $n - 2$ then either x beats y or y beats x . If x beats y then if Player 1 deletes x then y will be a source. If y beats x , then if Player 1 deletes y then x will be a source. Finally, if a tournament contains three vertices, x, y , and z , of score $n - 2$ then they must have the following relationships: $x \rightarrow y, y \rightarrow z$, and $z \rightarrow x$ or $y \rightarrow x, x \rightarrow z$, and $z \rightarrow y$. Now if Player 1 were to delete x, y , or z from the tournament one of the remaining vertices would be a source. □

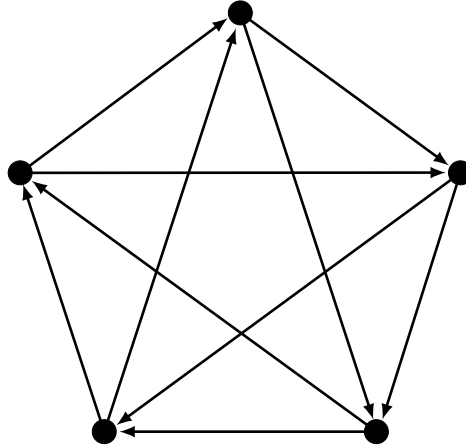
Definition. A tournament is called *regular* if every vertex has the same score.

Lemma 1. *The regular tournament of order 5, denoted $T_5^{(2)}$, is the smallest Player 2 optimal tournament.*

Proof. All tournaments with 3 or 4 vertices contain a vertex of score $n - 1$, or $n - 2$ and are either in the terminal or winning position respectively by Theorem 6. Every tournament with 5 vertices except $T_5^{(2)}$ also contains a vertex of score $n - 1$ or $n - 2$ and is either in the terminal or winning position respectively by Theorem 6. □

An Application

Theorem 6 can provide some interesting sociological and political science applications. Restricting the possible moves to only allow a player to remove kings from the tournament, creates

Figure 4.2: $T_5^{(2)}$

a variant of the game that models something like assassination and captures the idea that some object wants to be the sole king. Other versions of the game grant the model the flexibility to conform to many different applications. Define a tournament as *unstable* if it can be won in a single move. An unstable tournament representing a real power structure clearly facilitates competition for complete dominance over other objects in the tournament via the single object whose deletion yields a source. Define a tournament as *stable* otherwise. Then Theorem 6 may suggest strategic considerations regarding power structures developed to prevent singular domination (a single king), such as a democratic government. There are no stable tournaments that have 3 or 4 objects. The smallest stable tournament contains 5 objects. Suggesting that a stable democratic government should be built on 5 branches rather than 3.

4.3

Tournament Optimality Classification

In this section a way to iteratively determine if a tournament is Player 1 optimal (N position in the parlance of The Sprague- Grundy Theorem) or Player 2 optimal (P position). In order to present this, some notation and definitions will be presented. First, $\Delta(T)$ is the highest score in the tournament, T . The *order* of a tournament is the number of vertices in the tournament. A tournament of order n may be called an n -tournament.

Definition. Let T be a tournament of order n such that $\Delta(T) \leq n - 3$, then define S_n by:

$$S_n = \{T : H \not\triangleleft T, \forall H \in S_{n-1}\}$$

and

$$S_5 = \{T_5^{(2)}\}$$

Where $T_5^{(2)}$ is the regular 5-tournament.

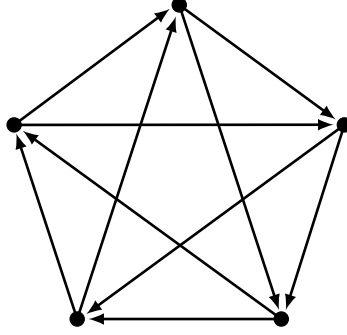


Figure 4.3: $T_5^{(2)}$

Lemma 2. *The elements of S_n are the only Player 2 optimal tournaments of order n .*

Proof. By Lemma 1, $T_5^{(2)}$ is the only Player 2 optimal tournament of order 5. Now consider the set, A , containing all tournaments of order n . Then A can be partitioned into the following sets

$$B = \{T : T \in A \text{ and } \Delta(T) = n - 1\}$$

$$C = \{T : T \in A \text{ and } \Delta(T) = n - 2\}$$

$$D = \{T : T \in A, T \notin B, \text{ and } T \notin C\}$$

The elements of B are terminal positions of the game and by definition are not in S_n . The elements of C are Player 1 optimal by Theorem 6, and by definition are not in S_n . Now partition D into two sets

$$S_n = \{T : \text{does not contain } H \forall H \in S_{n-1}\}$$

$$W_n = \{T : \exists H \text{ such that } T \text{ contains } H \text{ for } H \in S_{n-1}\}$$

The elements of W_n are Player 1 optimal because by the induction hypothesis $H \in S_{n-1}$ are Player 2 optimal. This implies that Player 1 can delete some vertex in T to obtain H and therefore Player 2, now playing on H , cannot win. Likewise, the elements of S_n are Player 2 optimal since by the induction hypothesis S_{n-1} contains all Player 2 optimal games of order $n - 1$. Therefore, there is no vertex in T that Player 1 can delete to force Player 2 to play on a Player 2 optimal tournament.

Therefore, by induction the elements of S_n are the only Player 2 optimal tournaments of order n . □

The sets described in the proof of Lemma 2 have been generated up to order 8 using SAGE. The code and the tournaments in S_n will be provided in Appendix A. A summary of these sets is provided in Table 4.1.

Table 4.1: Sizes of the Sets Described in the Proof of Lemma 2, for Tournaments up to Order 8.

n	$ A $	$ S_n $	$ W_n $	$ B $	$ C $	$ S_n / A $
5	12	1	0	4	7	$\frac{1}{12}$
6	56	5	5	12	34	$\approx \frac{1}{11}$
7	456	46	114	56	240	$\approx \frac{1}{10}$
8	6880	1277	2447	456	2700	$\approx \frac{1}{5}$

Conjecture. As $n \rightarrow \infty$,

$$\frac{|S_n|}{|A|} \rightarrow \frac{1}{2}.$$

Theorem 6. A tournament, T , of order $n \geq 6$ is Player 2 optimal if and only if T does not contain a vertex of score $n - 2$ and does not contain a subgraph in S_{n-1} .

Proof. Let T be a tournament of order $n \geq 6$ that does not contain a vertex of score $n - 2$ and does not contain a subgraph in S_{n-1} . Then by Lemma 2 we know that $T \in S_n$ and is therefore Player 2 optimal.

Let T be a Player 2 optimal tournament then by Lemma 2 $T \in S_n$ and by the definition of S_n we know that T does not contain a vertex of score $n - 2$ and does not contain a subgraph in S_{n-1} . □

Remark. Theorem 6 also provides a winning strategy, although this strategy is dependent on the construction of the set S_{n-1} . The strategy is as follows:

For any given T Player 1 should take the following steps

1. Determine if $\Delta(T) = n - 1$. If this is the case the game is in the terminal position and cannot be played.
2. Determine if $\Delta(T) = n - 2$. If this is the case Player 1 should identify a vertex, x , of score $n - 2$, and delete the vertex that beats x . Then $T - x$ now contains a source and the game is finished.
3. If (1) and (2) do not apply to T then determine if T contains H for any $H \in S_{n-1}$.

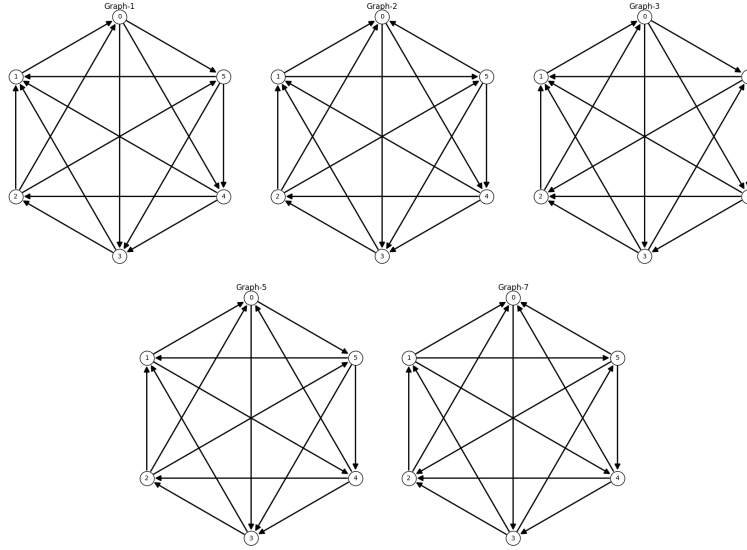


Figure 4.4: All Tournaments in S_6

- If T does contain some $H \in S_{n-1}$ then Player 1 can delete a vertex of T to obtain H . Then Player 2 cannot win because H is Player 2 optimal
- If T does not contain any $H \in S_{n-1}$ then Player 1 can not win because T is Player 2 optimal

4.4

Sprague-Grundy Theorem and The Game of Thrones

The Game of Thrones is an impartial combinatorial game, and as such is subject to The Sprague-Grundy Theorem. An example of how to apply The Sprague-Grundy Theorem to The Game of Thrones will be presented. Examine the tournament in Figure 4.5.

Recall, the vertices of the Sprague-Grundy graph will be labeled using the following algorithm. Start by labeling all vertices with no arcs leaving them “P”, these are the terminal (final) positions of the game. Next label any vertex with an arc pointing to a “P” vertex as “N”. Then label any vertex only pointing to “N” vertices as “P”. Repeat these steps until every vertex is labeled. The Sprague-Grundy Graph for this particular tournament is depicted in Figure 4.6.

The graph in Figure 4.6, shows that all “P” positions are terminal positions except one. In this case the “P” that is not terminal is $T_5^{(2)}$. The strategy suggested from The Sprague-Grundy Theorem is the same as the strategy suggested from our Optimality Classification, namely, that we search a tournament for specific sub-tournaments. Building The Sprague-Grundy Graph gives

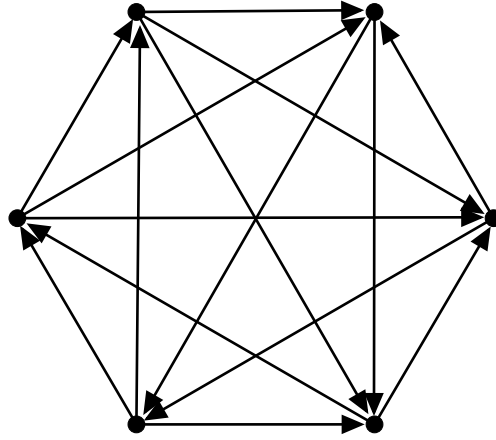


Figure 4.5: A 6-tournament

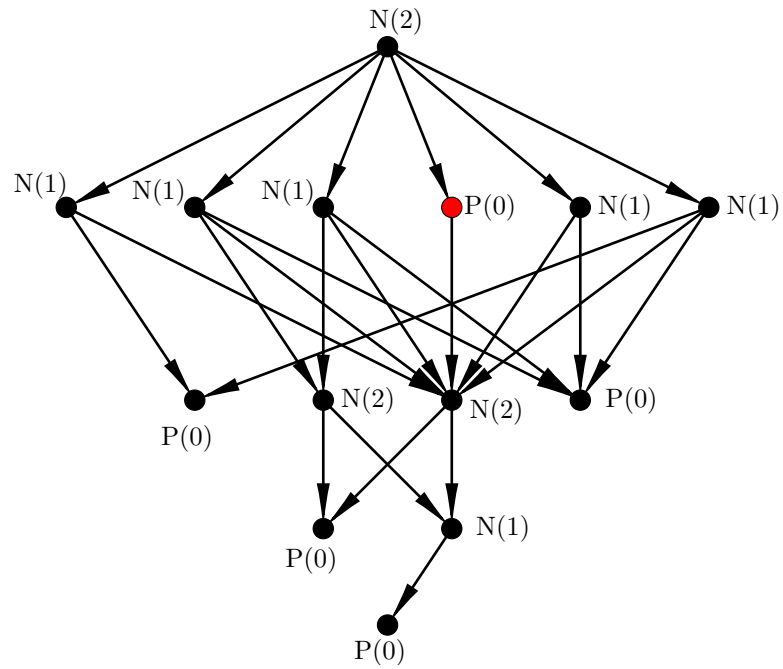


Figure 4.6: The Labeled Sprague-Grundy Graph for the Tournament in Figure 4.5

another way to build the set S_n . Therefore, while The Sprague-Grundy Theorem certainly applies to The Game of Thrones we did not learn anything new about the game.

4.5

Future Work

There are many directions future work on The Game of Thrones can take. First, if S_n can be characterized then a winning strategy that does not depend on the construction of $S_{n-1}, S_{n-2}, \dots, S_5$ can be discovered. Second, as described in Chapter 0 this work was spawned from a presentation given by Larry Langley. At this presentation Dr. Langley defined a new class of vertices in a tournament that he called Heirs.

Definition. An *heir* is a vertex that is not a king, but when a particular king is removed, it becomes a king[4].

Heirs may play a role in the winning strategy or losing position of The Game of Thrones, but it has not been discovered. This is a possible area of future work. Third, we can try to develop, or use an existing algorithm to allow a computer to play the game quickly. Algorithms such as alpha-beta pruning may be useful. This pursuit might also lead to some structural clues about the game. Last, there are many possible variations of this game, a select few follow.

Definition. A tournament is *strong* if for any two vertices, x and y , there is a path from x to y and there is a path from y to x .

Definition. Given a tournament T a set $S \subseteq V(T)$ is called a *dominating set* if every element $x \in V(T)$ is either in S or is beat by a vertex $y \in S$.

- Starting with a strong tournament players remove vertices until the tournament is no longer strong.
- Starting with a strong tournament players remove vertices of a dominating set until the tournament is no longer strong.
- Starting with any tournament players remove vertices until there is exactly one royal pair as defined by McKenna, Morton, and Sneddon[8].

REFERENCES

- [1] Charles L Bouton. Nim, a game with a complete mathematical theory. *The Annals of Mathematics*, 3(1/4):35–39, 1901.
- [2] Erik Demaine, Melissa Gymrek, and Jing Li. Theory of impartial games. MIT Open Courseware, Feb 2009. <http://web.mit.edu/sp.268/www/nim.pdf>.
- [3] Matthieu Dufour and Silvia Heubach. Circular nim games. *The Electronic Journal of Combinatorics*, 20(2), 2013.
- [4] Kim A.S. Factor and Larry J. Langley. Kings and heirs: A characterization of the (2,2)-domination graph of tournaments. *Discrete Applied Mathematics*, 204:142–149, 2016.
- [5] Patrick Michael Grundy. Mathematics and games. *Eureka*, 2:6–8, 1939.
- [6] Frank Harary and Edgard M. Palmer. *Graphical enumeration*. Academic press, 1973.
- [7] H. G. Landau. On dominance relations and the structure of animal societies i.: effect of inherent characteristics. *The Bulletin of Mathematical Biophysics*, 13(1):1–19, 1951.
- [8] Patricia McKenna, Margaret Morton, and Jamie Sneddon. New domination conditions for tournaments. *Australasian Journal of Combinatorics*, 26:171–182, 2002.
- [9] Sarah Mousley. Tournament directed graphs. Utah State University, 2013. Undergraduate Honors Thesis.
- [10] Roland Percival Sprague. Über mathematische kampfspiele. *Tohoku Mathematical Journal*, 41:438–444, 1935.
- [11] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

APPENDICES

APPENDIX

SAGE code used to generate S_n up to order 8.

```

1
2 sage: for g in digraphs.tournaments_nauty(5,max_out_degree=2,min_out_degree=2):
3 ...     S5 = g
4 sage: Check6=list()
5 sage: for g in digraphs.tournaments_nauty(6):
6 ...     x=vector(g.out_degree())
7 ...     if max(x)<4:
8 ...         Check6.append(g)
9 ...
10 ...
11 sage: for g in range(len(Check6)):
12 ...     Check6[g].name("6Graph-"+str(g))
13 ...
14 ...
15 sage: len(Check6)
16 //Output: 10
17 sage: S6=list()
18 sage: W6=list()
19 sage: for g in Check6:
20 ...     if g.subgraph_search(S5) is None:
21 ...         S6.append(g)
22 ...     else:
23 ...         W6.append(g)
24 ...
25 sage: print len(S6)
26 sage: print len(W6)
27 //Output: 5
28 //Output: 5
29 sage: for g in S6:
30 ...     g.dig6_string()
31
32 sage: Check7=list()
33 sage: for g in digraphs.tournaments_nauty(7):
34 ...     x=vector(g.out_degree())
35 ...     if max(x)<5:
36 ...         Check7.append(g)
37 ...
38 ...
39 sage: for g in range(len(Check7)):
40 ...     Check7[g].name("7Graph-"+str(g))
41 ...
42 ...
43 sage: len(Check7)
44 //Output: 160
45 sage: tempW7=list()
46 sage: for g in Check7:
47 ...     for h in S6:
48 ...         if g.subgraph_search(h) is not None:
49 ...             tempW7.append(g)
50 ...
51 sage: W7=list()
52 sage: for i in tempW7:
53 ...     if i not in W7:
54 ...         W7.append(i)
55 ...

```

```

56 sage: S7=list()
57 sage: for i in Check7:
58 ...     if i not in W7:
59 ...         S7.append(i)
60 ...
61 sage: print len(S7)
62 sage: print len(W7)
63 //Output: 46
64 //Output: 114
65 sage: for g in S7:
66 ...     g.dig6_string()
67
68 sage: Check8=list()
69 sage: for g in digraphs.tournaments_nauty(8):
70 ...     x=vector(g.out_degree())
71 ...     if max(x)<6:
72 ...         Check8.append(g)
73 ...
74 ...
75 sage: for g in range(len(Check8)):
76 ...     Check8[g].name("8Graph-"+str(g))
77 ...
78 ...
79 sage: len(Check8)
80 //Output: 3724
81 sage: tempW8=list()
82 sage: for g in Check8:
83 ...     for h in S7:
84 ...         if g.subgraph_search(h) is not None:
85 ...             tempW8.append(g)
86 ...
87 ...
88 sage: W8=list()
89 sage: for i in tempW8:
90 ...     if i not in W8:
91 ...         W8.append(i)
92 ...
93 sage: S8=list()
94 sage: for i in Check8:
95 ...     if i not in W8:
96 ...         S8.append(i)
97 ...
98 sage: print len(S8)
99 sage: print len(W8)
100 //Output: 1277
101 //Output: 2447
102 sage: for g in S8:
103 ...     g.dig6_string()

```

Figure 4.7: The SAGE Code used to Generate the Sets S_n up to Order 8.

The SAGE code not only counts the number of tournaments in S_n , but it also generates all of the tournaments. These tournaments can be encoded in a format called DiGraph6, that represents each graph as a string of characters. The tournaments in each S_n are included here in DiGraph6 format, these tournaments can be generated in SAGE from the DiGraph6 string using the DiGraph() command.

S6

'EF_pW[U'
'EE'pW[e'
'EF_oX[Y'
'EDapWkU'
'ECboXki'

S7

'F?OK^Fpkr?'
'FBO[F@q{1?'
'FBOKN@q{t?'
'FAO{FDp[j?'
'FAOK^Dpkr?'
'FBokN@o[V?'
'FBO{FDo[j?'
'FBO{F@q{1?'
'FBokN@q{t?'
'FBokFDq{x?'
'FBokFDq{x?'
'FBokF@o\?'
'FBokFDo\y?'
'FBokFDq{x?'
'FBo{N@o[F?'
'FBo[V@okN?'
'FBoK^@okV?'
'FBOK^Dokr?'
'FBO[V@qk1?'
'FBOK^@qkt?'
'FBOKVDqkx?'
'FBOK^@olu?'
'FBOKVDoly?'
'FBOKV@q1{?'
'F@pKFEo{Z?'
'F?PK^NAr{t?'
'F?PK^Epk?'
'F?P{^ApKf?'
'F?P{^VEpkj?'
'FEOKBLrsx?'
'FEo{BHpSN?'
'FEO{BLpSj?'
'FEOkJLpSr?'
'FEOkBLrSx?'
'FEokBHpT{?'
'FEokBLpTy?'
'FEokBHRt{?'
'FDpKBAq{\?'
'FJOSF@q{1?'
'FIOsFDp[j?'
'FIOC^Dpkr?'
'FIOsVDpkj?'
'FIOC^Dpkr?'

'FIOC^@rKt?'
'FIOCv@pL{?'
'FIOsV@pLm?'
'FIOCv@rL{?'

S8

'G@wB?wNDw^FW'
'G@GB@wNDw^LW'
'G@wB?wN@x^Fg'
'G@WJ?wNDwnFW'
'G@GJ@wNDwnLW'
'G@GN?wN@xnJg'
'G@GJ@wN@xnLg'
'G@GJ?w^@xnMg'
'G@GBBwN@xvLg'
'G@GBAw^@xvMg'
'G@WB?wnDwzFW'
'G@GF?wnDwzJW'
'G@GB@wnDwzLW'
'G@GB?w^@xzMg'
'G@GNBwN@wfHw'
'G@WJAwNDwfFW'
'G@GJAw^DwfMW'
'G@GNAwN@xfJg'
'G@GJBwN@xfLg'
'G@GJAw^@xfMg'
'G@GJAwNDxfNG'
'G@WJAwN@wfvO'
'G@GJBwN@wfv\o'
'G@GJAw^@wfv]o'
'G@GJAwNDwfv^o'
'G@GJAwN@xf^_'
'G@GN?w^@wjIw'
'G@GJ@w^@wjKw'
'G@WJ?wnDwjFW'
'G@GJ@wnDwjLW'
'G@GN?wn@xjJg'
'G@GJ@wn@xjLg'
'G@GJ?w^@xjMg'
'G@GJ?wnDxjNG'
'G@WJ?wn@wjVo'
'G@GJ@wn@wj\o'
'G@GJ?w^@wj]o'
'G@GJ?wnDwj^o'
'G@GJ?wn@xj^_'
'G?gV?wNEw^JW'
'G?gR@wNEw^LW'
'G?gR@wNAX^Lg'
'G?gR?w^Ax^Mg'
'G?G^?wNEwNjW'
'G?G^?wNAXnJg'

'G?GVAwNEwvJW'
'G?GRBwNAXvLg'
'G?GRAw^AxvMg'
'G?GV?wnEwzJW'
'G?GR@wnEwzLW'
'G?GR?w^AxzMg'
'G?w^?wNAwNBw'
'G?gZ@w^AwNKw'
'G?g^?wNEwNjW'
'G?gZ@wNEwNLW'
'G?gZ?w^EwNMW'
'G?wZ?wNAXNFg'
'G?g^?wNAXnJg'
'G?g^?wNAwNZo'
'G?gZ@wNAwN\o'
'G?gZ?w^AwN]o'
'G?gZ?wNEwN^o'
'G?gZ?wNAXn^_'
'G?wRBwNAwVDw'
'G?gVAwNEwvJW'
'G?gRAw^EwVMW'
'G?gRBwNAXvLg'
'G?gRAw^AxVMg'
'G?gRAwNEwVNG'
'G?gVAwNAwVZo'
'G?gRBwNAwV\o'
'G?gRAw^AwV]o'
'G?gRAwNEwV^o'
'G?gRAwNAXv^_'
'G?gFCwNFw^JW'
'G?gBCw^BX^Mg'
'G?GJCw^BXnMg'
'G?WBwEwNFwvFW'
'G?GFwEwNFwvJW'
'G?GBFwNFwvLW'
'G?GBwEw^FwvMW'
'G?GBFwNBXvLg'
'G?WBCwNFwzFW'
'G?GFCwNFwzJW'
'G?gBCw^BXzMg'
'G?GJCw^BXZJg'
'G?gBCw^BXZMg'
'G?gBCwNFXZNG'
'G?gBCwNBWZZo'
'G?gBDwNBWZ\o'
'G?gBCw^BWZ]o'
'G?gBCwNFwz^o'
'G?gBCwNBXZ^_'
'G?WNCwNBWjBw'
'G?WJDwNBWjDw'
'G?WJCw^BWjEw'
'G?GNCw^BWjIw'
'G?WJCwNFwjFW'
'G?GJDwNBXNLg'
'G?gJCw^BXNMg'
'G?gJCwNFXNNG'
'G?gNCwNBWNZo'
'G?gJDwNBWN\o'
'G?gJCw^FwNMW'
'G?gJDwNBXNLg'
'G?gJCw^BXNMg'
'G?gJCwNFXNNG'
'G?gNCwNBWNZo'
'G?gJDwNBWN\o'
'G?gJCw^BWN]o'
'G?gJCwNFwN^o'
'G?gJCwNBXN^_'
'G?wBFwNBWVDw'
'G?gFFwNBWVHw'

'G?gFEw^BwvIw'
'G?gFEwNFwvJW'
'G?gBFwNFwvLW'
'G?gBEw^FwvMW'
'G?wBEwNBXVfG'
'G?gFEwNBXVJg'
'G?gBFwNBXVLg'
'G?gFEwNBWVZo'
'G?gBFwNBWV\o'
'G?gBEw^Bwv]o'
'G?gBEwNFwv^o'
'G?gBEwNBXV^_'
'G?WNEwNBwfvBw'
'G?WJFwNBwfvDw'
'G?GNFwNBwfvHw'
'G?WJEw^BwfvEw'
'G?WJEwNFwfvFW'
'G?GJFwNFwfvLW'
'G?GJEw^FwfvMW'
'G?WJEwNBXfFg'
'G?GNEwNBXfJg'
'G?GJFwNBXfLg'
'G?WJEwNBwfvVo'
'G?GNEwNBXfJg'
'G?GJFwNBwfv\o'
'G?GJEw^Bwfv]o'
'G?GJEwNFwfv^o'
'G?GJEwNBXf^_'
'G?gFDwNBWZHw'
'G?wBCw^BWZEw'
'G?gFCw^BWZJw'
'G?gFCwNFwzJW'
'G?gBDwNBWZLW'
'G?gFCwNBXZJg'
'G?gBCw^BXZMg'
'G?gBCwNFXZNG'
'G?gFCwNBWZZo'
'G?gBDwNBWZ\o'
'G?gBCw^BWZ]o'
'G?gBCwNFwz^o'
'G?gBCwNBXZ^_'
'G?WNCwNBWjBw'
'G?WJDwNBWjDw'
'G?WJCw^BWjEw'
'G?GNCw^BWjIw'
'G?WJCwNFwjFW'
'G?GJDwNBXNLg'
'G?gJCw^BXNMg'
'G?gJCwNFXNNG'
'G?gNCwNBWNZo'
'G?gJDwNBWN\o'
'G?gJCw^FwNMW'
'G?gJDwNBXNLg'
'G?gJCw^BXNMg'
'G?gJCwNFXNNG'
'G?gNCwNBWNZo'
'G?gJDwNBWN\o'
'G?gJCw^BWN]o'
'G?gJCwNFwN^o'
'G?gJCwNBXN^_'
'G?wBFwNBWVDw'
'G?gJCwNBXj^_'

'G?wn?xNBgNBw'
'G?gJ@x^BgNKw'
'G?gJ?x^FgNMW'
'G?gN?xNBgNZo'
'G?gJ@xNBgN\o'
'G?gJ?x^BgN]o'
'G?wBBxNBgVDw'
'G?gFBxNBgVHw'
'G?gFAx^BgvIw'
'G?gBAX^FgVMW'
'G?gFAxNBgVZo'
'G?gBBxNBgV\o'
'G?gBAX^Bgv]o'
'G?WNAxNBgfvBw'
'G?WJBxNBgfvDw'
'G?GNBxNBgfvHw'
'G?WJAX^BgfvEw'
'G?GJAX^FgfvMW'
'G?WJAXNBgfvVo'
'G?GJBxNBgfv\o'
'G?GJAX^Bgfv]o'
'GBwB?wF@x^Fg'
'GBGJ@wF@xnLg'
'GBGBwF@xvLg'
'GBGB@wfv@xZLg'
'GBGB?wv@xZMg'
'GBGB@wFHX|Lg'
'GBGFBwF@wVHw'
'GBwBAwV@wVEw'
'GBGFAwV@wVIw'
'GBwBAwFDwVFW'
'GBGBAwFDxVNG'
'GBGFAwF@wVZo'
'GBGFBwV@wVGw'
'GBWJBwF@wfvDw'
'GBGNBwF@wfvHw'
'GBWJAwV@wfvEw'
'GBWJAwFDwfvFW'
'GBGNAwF@xfJg'
'GBGJBwF@xfLg'
'GBGJAwV@xfMg'
'GBWJAwFDxfNG'
'GBGJBwF@wfv\o'
'GBGJAwV@wfv]o'
'GBGJAwFDwfv^o'
'GBGJAwF@xf^_'
'GBGNBwFDwfvHw'
'GBwB@wfv@wZDw'
'GBGFBwF@wZHw'
'GBwB?wv@wZEw'
'GBGFBwF@wZKw'
'GBwB?wfvDwZFW'

'GBwB?wf@xZfG' 'GAGZ@wVIw1Kw' 'GBgZ@wV?wNKw' 'G?grBwJAxVLg' 'G?gbCwzBWZ]o'

'GBgB?wv@xZMg' 'GAG^?wFMw1JW' 'GBg^?wF?xNJg' 'G?grAwZAxVMg' 'G?gbCwjFWZ^O'

'GBgB?wfDxZNG' 'GAGZ@wFMw1LW' 'GBgZ@wF?wN\o' 'G?grAwJExVNG' 'G?gbCwjBXZ^_'

'GBwB?wf@wZVo' 'GAGZ?wVMw1MW' 'GBgZ?wF?xN^_'

'GBgF?wf@wZZo' 'GAWZ?wFIx1fG' 'GBw^?wFCwNBW' 'G?grAwJAwVZo' 'G?WnCwjBwjBw'

'GBgB@wf@wZ\o' 'GAG^?wFIx1Jg' 'GBw^?wF?xNBg' 'G?grAwZAwV]o' 'G?WjCwzBwjEw'

'GBgB?wv@wZ]o' 'GAGZ?wFMx1NG' 'GBWZAwV?wfEw' 'G?grAwJAwV^O' 'G?GnCwzBwjIw'

'GBwB@wFHw\DW' 'GAG^?wFIw1Zo' 'GBWZAwF?wfVo' 'G?grAwJAxV^_'

'GBgF@wFHw\Hw' 'GAGZ@wFIw1\o' 'GBG^BwF?xfHg' 'G?wr?wzAwZEw' 'G?GnDwjFWjHW'

'GBwB?wVHw\Ew' 'GAGZ?wVIw1]o' 'GBG^AwFCxfJG' 'G?gv?wzAwZlw' 'G?GjCwzBXjMg'

'GBgF?wVHw\Iw' 'GAGZ?wFMw1^O' 'G@wb?wJ@x^Fg' 'G?gr@wzAwZKw' 'G?GnDwjFWjHW'

'GBgB@wVHw\Kw' 'GAGVAwFMwtJW' 'G@Gn?wJ@xnJg' 'G?gv?wjEwZJW' 'G?GjCwzBXjMg'

'GBwB?wFLw\FW' 'GAGRBwFMwtLW' 'G@GfAwJ@xvJg' 'G?gr@wjAxZLg' 'G?GnDwjFWjHW'

'GBgB?wFLx\NG' 'GAGRAwVMwtMW' 'G@GbBwJ@xvLg' 'G?gr?wzAxZMg' 'G?GnDwjFWjHW'

'GAGR@wFAX^Lg' 'GAWRAwFIxtfG' 'G@Gf?wj@xxZjg' 'G?gr?wjExZNG' 'G?GjDwZ@WNKw'

'GAWZ?wFAXnFg' 'GAGRBwFIxtLg' 'G@Gb@wj@xxZlG' 'G?gv?wjAwZZo' 'G?GjCwJDWNFW'

'GAG^?wFAXnJg' 'GAGRAwVIxtMg' 'G@Gb?wz@xxZmG' 'G?gr@wjAwZ\o' 'G?GnDwJDWNJW'

'GAGRBwFAXvLg' 'GAGRAwFMxtNG' 'G@Gb?wjDxxZNG' 'G?gr?wzAwZ]o' 'G?GjDwJDWNLW'

'GAWR?wFAXzFg' 'GAGVAwFIwtZo' 'G@Gf?wJHx|Jg' 'G?gr?wjEwZ^O' 'G?GjCwZDWNMW'

'GAGR@wFAXzLg' 'GAGRAwVIwt]o' 'G@Gb@wJHx|Lg' 'G?gr?wjAxZ^_'

'GAGR?wvAxzMg' 'GAGRAwFMwt^O' 'G@Gb?wJLx|NG' 'G?WvAwjAwRbw' 'G?GjDwJ@XNfg'

'GAWR?wFIx|Fg' 'GAGRAwFIxt^_'

'G@Gj@wZ@wNKw' 'G?WrAwjAwRbw' 'G?GjDwJ@WN\o'

'GAGR@wFIx|Lg' 'GAGBFwFBXvLg' 'G@Gn?wJDwNJW' 'G?GvAwjAwRlw' 'G?GjCwJDWN^O'

'GAGR?wFMx|NG' 'GAGBCwvBXzMg' 'G@wj?wJ@xNFg' 'G?GrBwzAwRkw' 'G?GnDwZ@WNGw'

'GAWZ?wVAwNEw' 'GAGBCwFNx|NG' 'G@Gj@wJ@wN\o' 'G?GrAwzAxRmg' 'G?GjCwZ@XNEg'

'GAgZ@wVAwNKw' 'GAWNCwFBwNBw' 'G@Gj?wJDwN^O' 'G?GfCwJBX^Jg' 'G?GwBGwL@x^Fg'

'GAWZ?wFAXNFg' 'GAgJDwFBXNLg' 'G@GfBwJ@wVHw' 'G?GfEwJBXvJg' 'G?GNGwL@xnJg'

'GAG^?wFAXNJg' 'GAgJCwVBXNMg' 'G@GfAwZ@wVIw' 'G?GbFwJBXvLg' 'G?GFIwL@xvJg'

'GAgZ@wFAwN\o' 'GAgJCwFFXNNG' 'G@GbAwZDwVMW' 'G?GfCwJBXzJg' 'G?GGBJwL@xvLg'

'GAgZ?wFAXN^_'

'GAgNCwFBwNZo' 'G@wbAwJ@xVfG' 'G?GbCwzBXzMg' 'G?GbCwzBXzMg' 'G?wNGwL@wNBw'

'GAgVBwFAwVHw' 'GAgJCwFBXN^_'

'G@GfAwJ@wVZo' 'G?GfCwJXX|Jg' 'G?GfJHw\@wNKw'

'GAgRBwFAXvLg' 'GAgFDwFBWZHw' 'G@WjBwJ@wfdw' 'G?GbCwJNX|NG' 'G?GwJGwL@xNFg'

'GAgRAwVAXVMg' 'GAWBCwvBWZEw' 'G@GnAwJ@xfJg' 'G?wnCwJBwNBw' 'G?GNGwL@wNZo'

'GAgRAwFEXVNG' 'GAgFCwvBWZlw' 'G@GjBwJ@xfLg' 'G?wjDwJBwNDw' 'G?G^GwLAXnJg'

'GAgVAwFAwVZo' 'GAgFCwFBXZJg' 'G@GjAwZ@xfMg' 'G?GjDwJFwNLW' 'G?GRJwLAXvLg'

'GAgRAwFAXv^_'

'GAgBCwvBXZMg' 'G@GjAwJDxfNG' 'G?GjCwZFWNMW' 'G?G^GwLAWNBw'

'GAWZBwFAwfdw' 'GAgBCwFBXZNG' 'G@WjAwJ@wfvO' 'G?GjDwJBXNLg' 'G?GZHW\@wNKw'

'GAWZAwVAwfdw' 'GAgFCwFBWZZo' 'G@GjAwJ@xf^_'

'G?GjCwZBXNMg' 'G?GjCwZBXNMg' 'G?wZGwLAXNFg'

'GAG^AwFEwfJW' 'GAgBDwFBWZ\o' 'G@WfAwJHwtBw' 'G?GjCwJFXNNG' 'G?G^GwLAXnJg'

'GAGZAwVEwfMW' 'GAgBCwvBWZ]o' 'G@GbBwJLwtLW' 'G?gnCwJBwNZo' 'G?wZGwLAWNVo'

'GAWZAwFAXfFg' 'GAgBCwFBXZ^_'

'G@GbAwZLwtMW' 'G?GjDwJBwN\o' 'G?G^GwLAWNZo'

'GAWZAwFAwfvO' 'GAWJCwvBwjEw' 'G@GfAwJHxtJg' 'G?GjCwZBwN]o' 'G?GZHWLAWN\o'

'GAWZBwVAwfdw' 'GAGNCwvBwjIw' 'G@GbBwJHxtLg' 'G?GjCwJFWN^O' 'G?GZGwLAXN^_'

'GAgV@wFAwZHw' 'GAGNCwfwjJW' 'G@GbAwZHxtMg' 'G?GjCwJBXN^_'

'GAWR?wvAwZEw' 'GAGJDwfwjLW' 'G@GbAwJLxtNG' 'G?gnDwJFWNHw' 'G?GZHWL?wNDw'

'GAGR@wvAwZKw' 'GAWJCwFBXjFg' 'G@WbAwJHwtVo' 'G?wfCwJBWZBw' 'G?G^HwL?wNHw'

'GAGR@wFAxZLg' 'GAGJCwvBXjMg' 'G@GbAwZHwt]o' 'G?wbCwzBWZEw' 'G?GZHW\@wNKw'

'GAGR?wvAxZMg' 'GAGJCwFBXjNG' 'G@GbAwJLwt^O' 'G?GfCwzBWZlw' 'G?GwZGwL@xNFg'

'GAGR?wFEXZNG' 'GAWJCwFBwjVo' 'G@GbAwJHxt^_'

'G?gbDwjFWZLW' 'G?G^GwL@xNJg'

'GAgV?wFAwZZo' 'GAGJDwFBwj\o' 'G?G^?wJAXnJg' 'G?GfCwJBXZJg' 'G?GwZGwL?wNVo'

'GAGR@wFAwZ\o' 'GAGJCwvBwj]o' 'G?Gr?wzAxzMg' 'G?gbCwzBXZMg' 'G?G^GwL?wNZo'

'GAGR?wvAwZ]o' 'GAGJCwfwj^O' 'G?wrBwJAwVdw' 'G?gbCwjFXZNG' 'G?GZHWL?wN\o'

'GAGR?wFAxZ^_'

'GAGJCwFBXj^_'

'G?gfCwJBWZZo' 'G?GZGwL@xN^_'

'GAW^?wFIw1Bw' 'GBwZ?wV?wNEw' 'G?gvAwJEwVJW' 'G?gbDwjBWZ\o' 'G?w^HwL?wN@w'

'G@G^Hw\\?wNGw'	'GCG^?WNAxnJg'	'GCGJCW^FWNMW'	'GFwB?WF@x^Fg'	'GEG^?WfEwjJW'
'G@gZHw\\CwNKW'	'GCCRBWNAxvLg'	'GCGJDWNBXNLg'	'GFWJBWF@wfdw'	'GEGZ@WfEwjLW'
'G@w^GwL?xNBg'	'GCGV?WnEwzJW'	'GCGJCW^BXNMg'	'GFGNBWF@wfhw'	'GEWZ?WfAxjFg'
'GDwB?WNDw^FW'	'GCCGR@WnEwzLW'	'GCGJCWNFXNNG'	'GFGJAWVDwfmw'	'GEGZ@WfAwj\\o'
'GDwB?WN@x^Fg'	'GCCGR@WnAxzLg'	'GCGNCWNBWNZo'	'GFGNAWF@xfJg'	'GEWRBwfAwrDw'
'GDgB?W^@x^Mg'	'GCCGR?W^AxzMg'	'GCGJCWNFWN^O'	'GFGJBWF@xfLg'	'GEGVAwfEwrJW'
'GDWJ?WNDwnFW'	'GCG^?WNEwnJW'	'GCGJCWNBXN^_'	'GFGJAWV@xfMg'	'GEWRAwfAxrFg'
'GDGJ@WNDwnLW'	'GCGZ@WNEwnLW'	'GCWNCWnBwjBw'	'GFGJAWFDxfNG'	'GEGRBwfAxrLg'
'GDGJ?W^DwnMW'	'GCGZ?W^EwnMW'	'GCWJDWnBwjDw'	'GFWJAWF@wfvO'	'GEGRAWvAxrMg'
'GDGJ@Wn@xnLg'	'GCwZ?WNAxNFg'	'GCGNCW^BwjIw'	'GFGJBWF@wfv\\o'	'GEGRAWfExrNG'
'GDGJ?W^@xnMg'	'GCG^?WNAxNJg'	'GCWJCWnFwjFW'	'GFGJAWV@wfv]o'	'GEGVAwfAwrZo'
'GDWBAWNDwvFW'	'GCGZ@WNAwn\\o'	'GCGJDWnFwjLW'	'GFGJAWFDwfv^O'	'GEGRAWfAxr^_'
'GDGBAW^DwvMW'	'GCGZ?WNEwn^O'	'GCWJCWnBXjFg'	'GFGJAWF@xf^_'	'GEWJCWFBXnFg'
'GDGBBWN@xvLg'	'GCGZ?WNAxN^_'	'GCGJCW^BXjMg'	'GEWZ?WFAxnFg'	'GEGJCWVBXnMg'
'GDWB?WnDwzFW'	'GCwRBWNAwVDw'	'GCGJCWnFXjNG'	'GEG^?WFAxnJg'	'GEWBWFBXvFg'
'GDGB@Wn@xzLg'	'GCGVAWNEwVJW'	'GCGJDWnBwj\\o'	'GEWRAWFAxvFg'	'GEGBFWFBXvLg'
'GDgFBWN@wHw'	'GCGRAW^EwVMW'	'GCGJCW^Bwj]o'	'GEGRBWFAxvLg'	'GEWBCWfBXzFg'
'GDwBAW^@wVEw'	'GCGRBWNAxVLg'	'GCGJCWnFwj^O'	'GEGRAWVAxvMg'	'GEGBCWvBXzMg'
'GDgFAWNDwVJW'	'GCGRAW^AxVMg'	'GCGJCWnBXj^_'	'GEWR?WfAxzFg'	'GEWBCWFJX Fg'
'GDgBAW^DwVMW'	'GCGRAWNEwVNG'	'GCGB@XNFg^LW'	'GEGR?WvAxzMg'	'GEGBCWVJX Mg'
'GDgBAW^@xVMg'	'GCGVAWNAwVZo'	'GCGB@XNBh^Lg'	'GEWR?WfIx Fg'	'GEGBCWFNX NG'
'GDgFAWN@wVZo'	'GCGRBWNAwV\\o'	'GCWJ?XNFgnFW'	'GEGR@WfIx Lg'	'GEwBDwfbWZDw'
'GDWJBWN@wfdw'	'GCGRAWNEwV^O'	'GCGJ@XNFgnLW'	'GEGR?WfIx Mg'	'GEgFDwfbWZHw'
'GDGNBWN@wfhw'	'GCGRAWNAxV^_'	'GCGJ@XNBhnLg'	'GEGR?WfMx NG'	'GEgFCWvBWZIw'
'GDWJAWNDwfwFW'	'GCW^?WnAwjBw'	'GCWB?XnFgzFW'	'GEwZ?WVAwNEw'	'GEgFCwfwWZJW'
'GDGJAW^Dwfmw'	'GCGZ@W^AwjKw'	'GCGF?XnFgzJW'	'GEg^?WfEwnJW'	'GEgFCwfbXzJg'
'GDGNAWN@xfJg'	'GCWZ?WnEwjFW'	'GCGB@XnFgzLW'	'GEgZ?WVEwnMW'	'GEgBCwffXzNG'
'GDGJBWN@xfLg'	'GCGZ@WnEwjLW'	'GCGB?X^FgzMW'	'GEwZ?WFAxNFg'	'GEgBCwffWZ^O'
'GDGJAW^@xfMg'	'GCWZ?WnAxjFg'	'GCGB@XnBhzLg'	'GEg^?WFAxNJg'	'GEWJDwfbWjDw'
'GDGJAWNDxfNG'	'GCGZ@WnAwj\\o'	'GCGB?X^BhzMg'	'GEgZ?WFAxN^_'	'GEGJDwfwWjLW'
'GDWJAWN@wfvO'	'GCWVAWnAwrbw'	'GCGNBXNBgfHw'	'GEwVAWFAwVbw'	'GEWJCWfBXjFg'
'GDGJBWN@wfv\\o'	'GCWRBwnAwrdw'	'GCGNAX^BgfIw'	'GEWRBwFAwVDw'	'GEGJCWvBXjMg'
'GDGJAW^@wfv]o'	'GCGVAW^Awriw'	'GCWJAXNFgfw'	'GEgVBwFAwVhw'	'GEGJCwffXjNG'
'GDGJAWNDwfv^O'	'GCCRBW^AwrkW'	'GCGJAX^Fgfmw'	'GEgVAwFEwVJW'	'GEGJDwfbWj\\o'
'GDGJAWN@xf^_'	'GCWRAWnEwrfw'	'GCWJAXNBhffg'	'GEgRAWVEwVMW'	'GEGJCWfBXj^_'
'GDwB@Wn@wZDw'	'GCGVAWnEwrJW'	'GCGJAX^Bhfmg'	'GEgRBwFAxVLg'	'GEGNCWvFWjIW'
'GDgF@Wn@wZHw'	'GCWRAWnAxrFg'	'GCGJAXNFhfnG'	'GEgRAWVAxVMg'	'GEwB?XFBh^Fg'
'GDwB?W^@wZEw'	'GCCRBWnAxrLg'	'GCWJAXNBgfVo'	'GEgRAWFEwVNG'	'GEWJ?XFBhnFg'
'GDwB?WnDwZFW'	'GCCRAW^AxrMg'	'GCGJBXNBgf\\o'	'GEgVAWFAwVZo'	'GEWBAXFBhvFg'
'GDgF?WnDwZJW'	'GCCRAWnExrNG'	'GCGJAX^Bgf]o'	'GEgRBwFAwV\\o'	'GEGB?XvBhzMg'
'GDgB@WnDwZLW'	'GCGVAWnAwrZo'	'GCGJAXNFgf^O'	'GEgRAWFAxV^_'	'GEWB?XFJh Fg'
'GDwB?Wn@xZFg'	'GCCRAW^Awr]o'	'GCGJAXNBhf^_'	'GEWZBwFAwfdw'	'GEGB?XFNh NG'
'GDgB?W^@xZMg'	'GCCRAWnEwr^O'	'GCGN?X^BgjIw'	'GEG^AWFEwfJW'	'GEgFBXFBgVHw'
'GDwB?Wn@wZVo'	'GCCRAWnAxr^_'	'GCGJ@X^BgjKw'	'GEGZAWVEwfmw'	'GEwBAXVBgVEw'
'GDgB?WnDwZ^O'	'GCgBDWNFW^LW'	'GCWJ?XnFgjFW'	'GEWZAWFAxfFg'	'GEwBAXFFgVFW'
'GDgB?Wn@xZ^_'	'GCgBCW^FW^MW'	'GCGJ@XnFgjLW'	'GEWZAWFAwfvO'	'GEgFAXFFgVJW'
'GCGV?WNEw^JW'	'GCWJCWNFWnFW'	'GCGJ?X^FgjMW'	'GEGZAWFAwf]o'	'GEgFAXFBhVJg'
'GCGR@WNEw^LW'	'GCGJDWnFWnLW'	'GCGN?XnBhjJg'	'GEgV@WfAwZHw'	'GEgBAXFFhVNG'
'GCGR?W^Ew^MW'	'GCGJCW^FWnMW'	'GCGJ@XnBhjLg'	'GEGR@WvAwZKw'	'GEgFAXFBgVZo'
'GCGR@WNAx^Lg'	'GCGBFWNFwvLW'	'GCGJ?X^BhjMg'	'GEgV?WfEwZJW'	'GEWJBXFBgfDw'
'GCGR?W^Ax^Mg'	'GCGBEW^FWvMW'	'GCGJ@XnBhj\\o'	'GEGR@WfEwZLW'	'GEGNBXFBgfHw'
'GCG^?WNEwnJW'	'GCGBFWNBXvLg'	'GCGJ?X^Bgj]o'	'GEGR?WfExZLg'	'GEWJAXVBgfEw'
'GCGZ@WNEwnLW'	'GCwNCWNBWNBw'	'GCGJ?XnFgj^O'	'GEGR?WfExZNG'	'GEGNAXFFgfJW'
'GCWZ?WNAxnFg'	'GCGJDWnFWNLW'	'GCGJ?XnBhj^_'	'GEGR?WfEwZ^O'	'GEWJAXFBhfFg'

'GEGJAXVBhfMg' 'GDGjBWJ@xfLg' 'GCwr?WJMw\FW' 'GCgfAXJBhVJg' 'GFWj?WRHwLEw'
 'GEGJAXFFhfNG' 'GDGjAWZ@xfMg' 'GCgv?WJMw\JW' 'GCgbAXJFhVNG' 'GFGn?WBHx1Jg'
 'GEWJAXFBgfVo' 'GDGjAWJDxfNG' 'GCgv?WJIx\Jg' 'GCgfAXJBGVZo' 'GFGj@WBHx1Lg'
 'GEGJBXFBgf\o' 'GDWjAWJ@wfVo' 'GCgr@WJIx\Lg' 'GCgbAXJFgV^O' 'GFGj?WRHx1Mg'
 'GEGJAXVBgf]o' 'GDGjAWZ@wf]o' 'GCgr?WZIx\Mg' 'GCWjBXJBgfDw' 'GFWj?WBHw1Vo'
 'GEGJAXFFgf^O' 'GDGjAWJ@xf^_ ' 'GCgv?WJIw\Zo' 'GCCnAXZBgfIw' 'GFGn?WBHw1Zo'
 'GEGJAXFBhf^_ ' 'GDwb@Wj@wZDw' 'GCgr@WJIw\\\o' 'GCWjAXJFgfFW' 'GFGj?WBHx1^_ '
 'GEwZDWFANNDw' 'GDgf@Wj@wZHw' 'GCgr?WZIx\]o' 'GCWjAXJBhfFg' 'GFGf@WbHwxHw'
 'GEg^CWFEWNJW' 'GDgf?Wz@wZIw' 'GCgr?WJMw\^O' 'CCGjAXZBhfMg' 'GFWb?WrHwxEw'
 'GEgZCWVEWNMW' 'GDgb@Wz@wZKw' 'GCgr?WJIx\^_ ' 'CCGjAXJFhfNG' 'GFWb?WbHxxFg'
 'GEgZDWFAXNLg' 'GDgb@WjDwZLW' 'CCGbFWJBXvLg' 'GCWjAXJBgfVo' 'GFGf?WbHxxJg'
 'GEgZCWVAXNMg' 'GDwb?Wj@xZFg' 'CCGbCWJNX\NG' 'CCGjAXJBhf^_ ' 'GFGb@WbHxxLg'
 'GEgZCWFAxN^_ ' 'GDgb?Wz@xZMg' 'GCwJDWJBWNDw' 'GCwb?XJNg\FW' 'GFGb@WbHwx\o'
 'GEw^DWFANWew' 'GDgb?Wj@xZ^_ ' 'CCGjDwJBXNLg' 'GCgb@XJNg\LW' 'GFGb?WbHxx^_ '
 'GEg^CWVEWNIW' 'GDGn?WjDwjJW' 'CCGjCWZBXNMg' 'GCgb?XZNg\MW' 'G@xB?gNDw^FW'
 'GEwZDWFAXNDg' 'GDGj@WjDwjLW' 'CCGjCWJFXNNG' 'GCwb?XJNh\Fg' 'G@hF?gNDw^JW'
 'GEgVBXFAgVHw' 'GDGj@Wj@xjLg' 'CCGjCWJFWN^O' 'GCgf?XJNh\Jg' 'G@xB?gN@x^Fg'
 'GEwRAXVAgVEw' 'GDGj?WjDxjNG' 'CCGjCWJBXN^_ ' 'GCgb@XJNh\Lg' 'G@HJ@gNDwnLW'
 'GEgVAXFAgVZo' 'GDGj@Wj@wj\o' 'CCWnCWjBwjBw' 'GCgb?XJNh\NG' 'G@HJ?g^DwnMW'
 'GEwVAXVAgVAw' 'GDGfBwj@wrHw' 'GCWjDwJBwjDw' 'GCgf?XJJg\Zo' 'G@HN?gN@xnJg'
 'GEwVAXFEgVBW' 'GDWbAwz@wrEw' 'CCGnCWzBwjIw' 'GCgb@XJJg\\\o' 'G@HJ@gN@xnLg'
 'GEgRBXFEhVLG' 'GDGfAwj@xrJg' 'CCGjDwzBwjKw' 'GCgb?XZJg\]o' 'G@HJ?g^@xnMg'
 'GEWZBXFAgfDw' 'GDGbAwjDxrNG' 'GCWjCWjFwjFW' 'GCgb?XJNg\^O' 'G@HNBgn@wfhw'
 'GEG^AXFEgfJW' 'GDGfAwj@wrZo' 'GCWjCWjBXjFg' 'GCgb?XJNh\^_ ' 'G@HNag^@wfiw'
 'GEWZAXFAgfVo' 'GDGbAwjDwr^O' 'CCGjCWzBXjMg' 'GDwjCWZ@WNEw' 'G@XJAgNDwffw'
 'GEGZAXVAgf]o' 'GCgr@WJAx^Lg' 'CCGjCWjFXjNG' 'GDwjCWJDWNFW' 'G@HJAg^Dwfmw'
 'GEw^AXFEgfBW' 'CCG^?WJAxnJg' 'CCGjDwJBwj\o' 'GDgnCWJDWNJW' 'G@HNAGn@xfJg'
 'GEG^BXFEgfHW' 'CCGrBWJAxvLg' 'CCGjCWzBwj]o' 'GDgjDwJ@XNLg' 'G@HJBgn@xfLg'
 'GEG^AXVEgfIW' 'CCGr@WjAxzLg' 'CCGjCWjBXj^_ ' 'GDgjCWZ@XNMg' 'G@HJAg^@xfMg'
 'GEGZBXVEgfKW' 'CCGr@WJIx|Lg' 'CCGnDwjFwjHW' 'GDgjCWJDWN^O' 'G@HJAgNDxfNG'
 'GEWZAXVAhfEg' 'GCwrBWJAwdw' 'GCwbDwJW\Dw' 'GDwnCWJDWNBW' 'G@XJAgN@wfvO'
 'GEG^AXFEhfJG' 'GCgvAWJEwVJW' 'GCwbCWJNW\FW' 'GDgnCWZ@XNIg' 'G@HJBgn@wfv\o'
 'GDwb?WJ@x^Fg' 'GCgrBWJAxvLg' 'CCGbDwJNW\LW' 'GDgjDwZ@XNKg' 'G@HJAgNDwf^O'
 'GDgb?WZ@x^Mg' 'GCgrAWZAxVMg' 'CCGbCWZNW\MW' 'GDwjCWJDXNFG' 'G@HJAgN@xf^_ '
 'GDGn?WJ@xnJg' 'GCgrAWJExVNG' 'CCGfCWJX\Jg' 'GDwnCWj@WJBw' 'G?hV?gNEw^JW'
 'GDGj@WJ@xnLg' 'GCgvAWJAwVZo' 'CCGbCWZJX\Mg' 'GDwnDwj@WJew' 'G?hR@gNEw^LW'
 'GDGbBWJ@xvLg' 'GCgrBWJAwV\o' 'CCGbCWJNX\NG' 'GDwnCWz@WJAw' 'G?hR?g^Ew^MW'
 'GDGf?Wj@xzJg' 'GCgrAWJEwV^O' 'CCGfCWJW\Zo' 'GDwjDwz@WJCw' 'G?H^?gNEwnJW'
 'GDGb@Wj@xzLg' 'GCgrAWJAxV^_ ' 'CCGbDwJW\\\o' 'GFwb?WB@x^Fg' 'G?HZ@gNEwnLW'
 'GDGf?WJHx|Jg' 'GCwrBWjAwrDw' 'CCGbCWZJW\]o' 'GFgb?WR@x^Mg' 'G?HZ?g^EwnMW'
 'GDGb@WJHx|Lg' 'CCGvAWzAwrIw' 'CCGbCWJNW\^O' 'GFgb?WBDx^NG' 'G?H^?gNAxnJg'
 'GDGb?WZHx|Mg' 'CCGrBwzAwrKw' 'CCGbCWJX\^_ ' 'GFGfAWB@xvJg' 'G?HZ?g^AxvMg'
 'GDGb?WJLx|NG' 'GCwrAWjEwrFW' 'CCGf?XJBh^Jg' 'GFGbAWBDxvNG' 'G?HRAg^AxvMg'
 'GDwj?WZ@wNEw' 'GCwrAWjAxrFg' 'CCGb@XJBh^Lg' 'GFGb@Wb@xzlG' 'G?HR@gNEwzLW'
 'GDgn?WJDWNJW' 'CCGrBWjAxrLg' 'CCGj@XJBhNLg' 'GFGb?WbDxzNG' 'G?HR@gNAxzLg'
 'GDgj?WZ@xNMg' 'CCGrAWzAxrMg' 'CCGf?XjBhzJg' 'GFWjBWB@wfdw' 'G?xRBgNAwVDw'
 'GDgj?WJDWN^O' 'CCGrAWjExrNG' 'CCGb@XjBhzLg' 'GFGnAWB@xfJg' 'G?hVAg^AwVIw'
 'GDwfAWJ@wVBw' 'CCGvAWjAwrZo' 'CCGb?XzBhzMg' 'GFGjBWB@xfLg' 'G?hVAgNEwVJW'
 'GDgfBWJ@wVHw' 'CCGrAWzAwr]o' 'CCGf?XJHh|Jg' 'GFGjAWR@xfMg' 'G?hRAg^EwVMW'
 'GDgbAWZDwVMW' 'CCGrAWjEwr^O' 'CCGb@XJHh|Lg' 'GFGjAWBDxfNG' 'G?hRBgnAxvLg'
 'GDgbAWZ@xVMg' 'CCGrAWjAxr^_ ' 'CCGb?XJNh|NG' 'GFWjAWB@wfvO' 'G?hRAg^AxVMg'
 'GDgfAWJ@wVZo' 'CCGvBwzAwrGw' 'CCGfBXJBgVHw' 'GFGjAWR@wfv]o' 'G?hRAgNExVNG'
 'GDWjBWJ@wfdw' 'GCgv?WZIW\Iw' 'CCGbAXZfGVMW' 'GFGjAWB@xf^_ ' 'G?hVAgNAwVZo'
 'GDGnAWJ@xfJg' 'GCgr@WZIW\Kw' 'GCwbAXJBhVfG' 'GFWnBWB@wfvw' 'G?hRBgnAwv\o'

'G?hRag^AwV]o', 'GAhRagFAxV^_', 'GHGDAwnDwrJW', 'GJG@AwFLwt^O', 'GIGHFwFBXfLg',
 'G?hRagNEwV^O', 'GAhR@gvAwZKw', 'GHGDAwn@xrJg', 'GJG@AwFHxt^_', 'GIGHewVBWf]o',
 'G?hRagNAxV^_', 'GAhV?gfAxZJg', 'GHG@Aw~@xrMg', 'GIWX?wFAxnFg', 'GIGHewFBXf^_',
 'G?xR?g~AwZEw', 'GAhR@gfAxZLg', 'GHG@AwnDxrNG', 'GIG\?wFAxnJg', 'GIWHCwvBWjEw',
 'G?hV?gnEwZJW', 'GAhR?gfExZNG', 'GHW@Awn@wrVo', 'GIWPAwFAxvFg', 'GIGLCwvBWjIw',
 'G?hR@gnEwZLW', 'GAhR@gfAwZ\o', 'GHGDAwn@wrZo', 'GIGPAwFExvNG', 'GIGLCwffWjJW',
 'G?hV?gnAxZJg', 'GAhR?gfAxZ^_', 'GHG@Aw~@wr]o', 'GIWP?wFAxzFg', 'GIWHCwFBXjFg',
 'G?hR@gnAxZLg', 'GAXRBgfAwrdw', 'GHG@AwnDwr^O', 'GIGP?wvAxzMg', 'GIGHCwvBXjMg',
 'G?hR?gnExZNG', 'GAXRAgvAwrdw', 'GHG@Awn@xr^_', 'GIGP?wFExzNG', 'GIGHCwffXjNG',
 'G?hV?gnAwZZo', 'GAHVAgfEwrJW', 'GGwL?xNBgNBw', 'GIWP?wFIx|Fg', 'GIWHCwFBWjVo',
 'G?hR@gnAwZ\o', 'GAXRAgfAxrFg', 'GGwH@xNBgNDw', 'GIGP?wFMx|NG', 'GIGHDwFBWj\o',
 'G?hR?g~AwZ]o', 'GAHRAgfExrNG', 'GGgH@x^BgNKw', 'GIwX@wFAwNDw', 'GIGHCwvBWj]o',
 'G?hR?gnEwZ^O', 'GAXRAgfAwrdw', 'GGgH?x^FgNMW', 'GIgX@wVAwNKw', 'GIGHCwffWj^O',
 'G?hR?gnAxZ^_', 'GAHRAgfEwr^O', 'GGgL?xNBgNZo', 'GIg\?wFEwNJW', 'GIGHCwFBXj^_',
 'GBxB?gF@x^Fg', 'GHw?wNDw^FW', 'GGgH?x^BgN]o', 'GIgX@wFEwNLW', 'GIGLCwFNWlJW',
 'GBHN?gF@xNjg', 'GHw?wN@x^Fg', 'GJw@?wF@x^Fg', 'GIwX?wFAxNFg', 'GIGHCwvNWlMW',
 'GBHJ@gF@xNlg', 'GHg@wN@x^Lg', 'GJG@?wv@xzMg', 'GIg\?wFAxNJg', 'GIGHCwFNXlNG',
 'GBHJ?gV@xNmg', 'GHWH?wNDwNFw', 'GJWH?wv@wJew', 'GIgX?wFAxN^_', 'GIg\CwFEwNJW',
 'GBHF?gFHx|Jg', 'GHGH@wN@xNlg', 'GJGL?wv@wJiw', 'GIw\?wFEwNBW', 'GIgXCwFAXN^_',
 'GBxJ?gV@wNEw', 'GHGH?w~@xNmg', 'GJGH@wv@wJkw', 'GIwTawFAwVBw', 'GIw\CwFEwNBW',
 'GBhN?gV@wNIw', 'GHw@AwNDwVFW', 'GJGH@wv@wJlw', 'GIgTawFEwVJW', 'GIg\DwFEwNHw',
 'GBxJ?gFDwNFw', 'GHGDAwn@xvJg', 'GJGL?wf@xJjg', 'GIgPAwVEwVMW', 'GIg\CwVEwNIW',
 'GBhJ?gFDxNNG', 'GHG@Aw~@xvMg', 'GJGH@wv@xJlg', 'GIgPBwFAxVLg', 'GIwXDwFAXNDg',
 'GBhJ?gV@wN]o', 'GHw@?wNDwzFW', 'GJGH?wv@xJmg', 'GIgPAwVaxVMg', 'GIwTEwFAwVBw',
 'GBXNagF@wfbw', 'GHG@?w~@xzMg', 'GJGH?wfdxjNG', 'GIgPAwFExVNG', 'GIgTEwFEwVJW',
 'GBXJBgF@wfdw', 'GHw@BwN@wVDw', 'GJWH?wv@wJvo', 'GIgPAwVAwV]o', 'GIgPEwVEwVMW',
 'GBHNBgF@wfhw', 'GHgDAw~@wVIw', 'GJGH@wv@wJ\o', 'GIgPAwFEwV^O', 'GIgPFwFAXVLg',
 'GBHNBgV@wfiw', 'GHw@AwNDwVFW', 'GJGH?wv@wJ]o', 'GIGTawFMwtJW', 'GIgPEwFEwV^O',
 'GBXJagFDwffw', 'GHgDAwNDwVJW', 'GJGH?wfdwJ^O', 'GIGPBwFMwtLW', 'GIwTEwVAwVAw',
 'GBHNBgF@xfJg', 'GHg@Aw^DwVMW', 'GJGH?wv@xJ^_', 'GIGPAwVMwtMW', 'GIgPFwVAXVKg',
 'GBHJBgF@xfLg', 'GHw@AwN@xVFG', 'GJW@Bwv@wrDw', 'GIGPAwFMxtNG', 'GIgPFwFEXVLG',
 'GBHJagV@xfMg', 'GHg@BwN@xVLg', 'GJW@Awv@wrEw', 'GIw@CwFBX^Fg', 'GIwXFwFAwfdw',
 'GBHJagFDxfNG', 'GHw@AwN@wVVo', 'GJGDawv@wrIw', 'GIWHCwFBXnFg', 'GIwXewVAwFew',
 'GBXJagF@wfv', 'GHg@Aw~@wV]o', 'GJG@Bwv@wrKw', 'GIGHCwFFXnNG', 'GIG\EwFEwJW',
 'GBHJBgF@wfv\o', 'GHg@AwNDwV^O', 'GJW@AwfDwrFW', 'GIGDEwFBXvJg', 'GIGXewVEwFMW',
 'GBHJagFDwfv^O', 'GHg@AwN@xV^_', 'GJGDawf@xrJg', 'GIG@FwFBXvLg', 'GIGXFwFAXfLg',
 'GBHJagF@xf^_', 'GHWH@wN@wJdw', 'GJG@Awv@xrMg', 'GIW@CwFBXzFg', 'GIGXewVAwF]o',
 'GBxJ?gFHw\Bw', 'GHGL?w~@wJiw', 'GJG@AwfDxrNG', 'GIGDCwFBXzJg', 'GIGXewFAXf^_',
 'GBHF@gFHw\Hw', 'GHGH@w~@wJkw', 'GJW@Awf@wrVo', 'GIG@CwvBXzMg', 'GIwXFwVAwfcw',
 'GBxB?gFLw\FW', 'GHWH?wNDwJFW', 'GJGDawf@wrZo', 'GIG@CwffXzNG', 'GIwXewVEwFEW',
 'GBhB?gFLx\NG', 'GHGH@wNDwJLW', 'GJG@Awv@wr]o', 'GIW@CwFJX|Fg', 'GIG\EwVEwFIW',
 'GAH^?gFAxnJg', 'GHGL?wN@xJjg', 'GJG@AwfDwr^O', 'GIG@CwFNX|NG', 'GIwXFwFAXfdg',
 'GAHZ?gVAxnMg', 'GHGH@wN@xJlg', 'GJG@Awf@xr^_', 'GIgDFwFBwVHw', 'GIG\FwFAXfHg',
 'GAHRagVAxvMg', 'GHGH?w~@xJmg', 'GJW@BwFhwtDw', 'GIgDEwVBwVlw', 'GIwXewVAXfeg',
 'GAHR@gfAxzLg', 'GHGH?wNDxjNG', 'GJGDawVhwtlw', 'GIw@EwFFwVFW', 'GIg\EwFEwFJW',
 'GAHR?gFMx|NG', 'GHWH?wN@wJvo', 'GJW@AwFLwtFW', 'GIgDEwFBXvJg', 'GIw\EwFEwFBW',
 'GAhVBgFAwVHw', 'GHGH@wN@wJ\o', 'GJG@BwFLwtLW', 'GIg@EwFFwV^O', 'GIg\FwFEwFHW',
 'GAhVagVAwVlw', 'GHGH?w~@wJ]o', 'GJW@AwFHxtFg', 'GIw@FwFFwVdW', 'GIg\EwVEwFIW',
 'GAhRBgFAxVLg', 'GHGH?wNDwJ^O', 'GJGDawFHxtJg', 'GIWHFwFBwfdw', 'GIWHEwVBwFew',
 'GAhRagVAxVMg', 'GHGH?wN@xJ^_', 'GJG@AwVhxtMg', 'GIWHEwVBwFew', 'GIGLEwFFwFJW',
 'GAhRagFExVNG', 'GHGDAw~@wrIw', 'GJW@AwFHwtVo', 'GIGLEwFFwFJW', 'GIGHewVFWfMW',
 'GAhVagFAwVzo', 'GHG@Bw~@wrKw', 'GJGDawFHwtZo', 'GIGHewVFWfMW', 'GIGHewVFWfMW',
 'GAhRagVAwV]o', 'GHw@AwnDwrFW', 'GJG@AwVhwt]o', 'GIGLEwFBXfJg', 'GIGLEwFBXfJg'