6-2016

REALIZING TOURNAMENTS AS MODELS FOR K-MAJORITY VOTING

Gina Marie Cheney
California State University - San Bernardino

Realizing Tournaments as Models for k-Majority Voting

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Arts

in

Mathematics

---

by

Gina Marie Cheney

June 2016

Realizing Tournaments as Models for k-Majority Voting

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Gina Marie Cheney

June 2016

Approved by:

Dr. Jeremy Aikin, Committee Chair	Date
Dr. Joseph Chavez, Committee Member	
Dr. Belisario Ventura, Committee Member	
Dr. Charles Stanton, Chair, Department of Mathematics	Dr. Corey Dunn Graduate Coordinator, Department of Mathematics

Abstract

A k-majority tournament is a directed graph that models a k-majority voting scenario, which is realized by $2k-1$ rankings, called linear orderings, of the vertices in the tournament. Every k-majority voting scenario can be modeled by a tournament, but not every tournament is a model for a k-majority voting scenario. In this thesis we show that all acyclic tournaments can be realized as 2-majority tournaments. Further, we develop methods to realize certain quadratic residue tournaments as k-majority tournaments. Thus, each tournament within these classes of tournaments is a model for a k-majority voting scenario. We also explore important structures specifically pertaining to 2- and 3-majority tournaments and introduce the idea of pseudo-3-majority tournaments and inherited 2-majority tournaments.

Acknowledgements

First, I want to thank my family for always being there and helping me through all of my years of education. I couldn't have done any of this without their support.

Next, I would like to thank all of the faculty at CSUSB for the wonderful educational experience I have had there. I would especially like to thank Dr. Joseph Chavez and Dr. Belisario Ventura, who were my committee members for this thesis and my professors for many courses throughout my time at CSUSB.

Finally, I would like to thank Dr. Jeremy Aikin, my adviser. He introduced me to graph theory and guided me through this process. I am truly grateful for all of the time and work he dedicated to helping me get to this point.

Table of Contents

List of Figures

Chapter 1

Introduction

A k-majority voting scenario is one in which $2k-1$ voters rank n candidates, by their preference. Candidate A defeats Candidate B if A is ranked above B by at least k of the voters. The following is an example of rankings of five candidates, labeled 1 through 5, by five voters. Note that the vertices are separated by $<$, indicating that each vertex v is ranked above the next vertex, and every other vertex that follows v in the ranking.

$$\text{Voter } 1 : 1 < 2 < 3 < 4 < 5$$
$$\text{Voter } 2 : 2 < 3 < 1 < 5 < 4$$
$$\text{Voter } 3 : 4 < 5 < 2 < 1 < 3$$
$$\text{Voter } 4 : 4 < 1 < 5 < 2 < 3$$
$$\text{Voter } 5 : 5 < 3 < 1 < 2 < 4$$

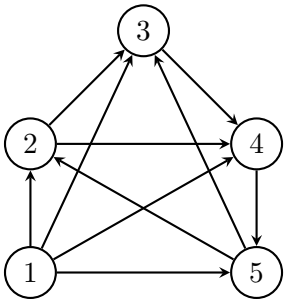


Figure 1.1: Diagram of a tournament

This example represents as a 3-majority voting scenario. Every k-majority voting scenario can be modeled by an object called a tournament. Figure 1.1 is a diagram of the tournament that models the 3-majority voting scenario shown above it.

Each candidate is represented by a vertex, or node, in the tournament. We can see, in the voter rankings, that Candidate 1 is ranked above Candidate 2 by three (a majority) of the voters, and thus Candidate 1 defeats Candidate 2. This relationship is represented in the tournament by an arrow, or arc, pointing from Vertex 1 to Vertex 2. The same is true for the remaining pairs of candidates/vertices in the 3-majority tournament.

Now consider the set of rankings given by Voter 1, Voter 4, and Voter 5.

$$\text{Voter } 1 : 1 < 2 < 3 < 4 < 5$$
$$\text{Voter } 4 : 4 < 1 < 5 < 2 < 3$$
$$\text{Voter } 5 : 5 < 3 < 1 < 2 < 4$$

This set of rankings can be modeled by the same tournament as the original set of five rankings. In both sets, we have: Vertex 1 defeats the vertices 2, 3, 4, and 5; Vertex 2 defeats the vertices 3 and 4; Vertex 3 defeats Vertex 4; Vertex 4 defeats Vertex 5; and Vertex 5 defeats the vertices 2 and 3. Thus, the tournament in Figure 1.1 is actually a model for a 2-majority voting scenario as well.

It is true that every k-majority voting scenario can be modeled by a tournament. However, not every tournament is a model for a k-majority voting scenario. So, when does a tournament model a k-majority voting scenario? In this thesis we will determine that certain classes, or subclasses, of tournaments are k-majority tournaments, for specific values of k.

If Vertex A defeats Vertex B in a k-majority tournament, then A is said to dominate B. A dominating set for a k-majority tournament is a collection of vertices that together dominate all of the vertices in the tournament. A minimum dominating set is a smallest such collection of vertices. In the example above, the vertex labeled 1 dominates every other vertex, and therefore $\{1\}$ is a minimum dominating set for the tournament. Thus, in this tournament, the candidate represented by Vertex 1 is the winner in the voting scenario, as it defeats all other candidates.

Now consider the tournament represented by the voter rankings and tournament diagram in Figure 1.2. This is another 2-majority tournament, but this tournament has

Voter 1 : $1 < 2 < 3 < 4 < 5$

Voter 2 : $3 < 1 < 2 < 5 < 4$

Voter 3 : $2 < 3 < 1 < 4 < 5$

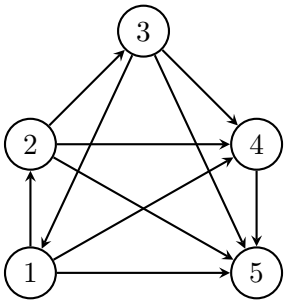


Figure 1.2: Another voting scenario

minimum dominating sets $\{1, 2\}$, $\{1, 3\}$, and $\{2, 3\}$. Thus, in this tournament there is no clear winner, but the dominating sets may be used to help determine a winner.

It has been proven, by Paul Erdös, that general tournaments can have minimum dominating sets that are arbitrarily large. In contrast, Noga Alon et al. proved that for the class of k-majority tournaments, with a fixed k, this is not the case. In fact, the size of the minimum dominating sets for k-majority tournaments is bounded above by a value that depends only on k. Thus, knowing that a class of tournaments is contained in the class of k-majority tournaments implies that those tournaments cannot have arbitrarily large minimum dominating sets.

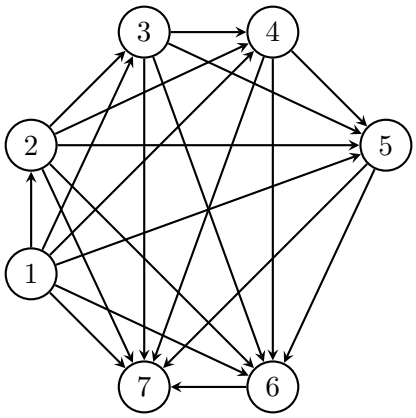
Chapter 2

Tournaments, k-Majority Voting, and Dominating Sets

2.1 Tournaments

A *graph* G is a finite nonempty set V of *vertices* together with a possibly empty set of *edges*, which are 2-element subsets of V [CLZ11]. The *order* of a graph G is the number of vertices in the vertex set V of G, and is denoted by n. A *digraph* D has a set V of vertices and a set A of *arcs*, which are ordered pairs of vertices in V. A *tournament* T is a digraph such that for all distinct pairs of vertices u and v in V, either uv is an arc in A or vu is an arc in A, but not both. If uv is an arc in a digraph D, then u and v are known as *adjacent vertices*, more specifically, u is *adjacent to* v and v is *adjacent from* u. The *out degree* of a vertex v in a digraph D, denoted $od(v)$, is the number of vertices that v is adjacent to , while the *in degree* of a vertex v, denoted $id(v)$, is the number of vertices that v is adjacent from. The *adjacency matrix* is an nxn matrix such that the a_{ij} entry is 1 if the vertex v_i is adjacent to the vertex v_j and 0 otherwise. Figure 2.1 shows a tournament and the adjacency matrix associated with it.

A *path* in a tournament T is a sequence of distinct vertices in V, $v_1v_2...v_k$, such that v_1v_2, v_2v_3, ..., $v_{k-1}v_k$ are in the set of arcs, A, in T. A *cycle* is a path such that the arc v_kv_1 is also in A. The *length* of a path or cycle is the number of arcs in the path or cycle. A cycle of length three is referred to as a *triangle*. The tournament in Figure 2.1 is an *acyclic* tournament, meaning that it contains no cycles. This tournament is also



$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2.1: A tournament and its adjacency matrix

transitive, meaning that whenever uv and vw are arcs in A of T, uw is also an arc in A. In fact, it is always the case that if a tournament is acyclic, it is also transitive [CLZ11].

Theorem 2.1. *A tournament is transitive if and only if it is acyclic.*

Proof. Let T be an acyclic tournament with vertex set V and arc set A. Let u, v, and w be vertices in V, with uv and vw arcs in A. Since T is acyclic, wu cannot be an arc in A, since it would form a cycle, so uw must be an arc in A. Thus T is also transitive.

Now suppose T is a transitive tournament, but T is not acyclic. So, there must be a cycle C in T such that, for some subset of vertices in V, $C : v_1v_2...v_kv_1$. Then, v_1v_2 and v_2v_3 are arcs in A. But, since T is transitive, v_1v_3 is also in A. Then, v_1v_3 and v_3v_4 are in A, and thus, v_1v4 must also be in A. Continuing in this fashion, v_1v_{k-1} and $v_{k-1}v_k$ are in A. By the definition of C, v_kv_1 is in A, but T is transitive, so v_1v_k must be in A, a contradiction. Thus, a tournament T is transitive if and only if it is acyclic. □

The tournament in Figure 2.1 also contains a *Hamiltonian path*, which is a path that contains every vertex in V of T. Actually, every tournament contains a Hamiltonian path [CLZ11].

Theorem 2.2. *Every tournament contains a Hamiltonian path.*

Proof. Let T be a tournament with vertex set V and arc set A, and suppose there is no Hamiltonian path in T. Consider a longest path P in T, such that $P : v_1v_2...v_k$. Since T contains no Hamiltonian path, there must be some vertex $v_j \in V$ that is not on the path P. In a tournament, for every pair of vertices u and v in V, either uv or vu is in A, so for each vertex v_i, with $i \in [1, k]$, either v_iv_j is in A or v_jv_i is in A. Either of the arcs v_jv_1 or v_kv_j will contradict the maximality of P. From this, it follows that there must exist $i \in [1, k]$ such that $v_1v_jv_{i+1}$ is a path. This again implies that P is not a longest path, which is a contradiction. Thus, every tournament contains a Hamiltonian path. □

2.2 k- Majority Tournaments

In a tournament T with vertex set V, for vertices $u, v \in V$, we say that u *dominates* v, denoted $u \to v$, if uv is an arc in A, or if $u = v$. A set X of vertices *dominates* a set Y of vertices if, for all $y \in Y$, there exists an $x \in X$ such that $x \to y$. A linear ordering, π_i, is a ranking of the vertices in the vertex set V of a tournament,

where the first vertex in the linear ordering is ranked highest and the last vertex is ranked lowest. Given a set $\mathscr{L} = \{\pi_1, \pi_2, ..., \pi_{2k-1}\}$ of $2k-1$ linear orderings of the elements of a finite set V, we can construct a tournament having vertex set V by imposing the rule that for all $u, v \in V$, $u \to v$ if and only if u lies above v in at least k of the linear orderings in $\mathscr{L}$. We call such a tournament a *k-majority tournament* T and say that T is *realized* by the set $\mathscr{L}$ of linear orderings of V. We will write $T \sim \mathscr{L}$ when T is realized by a set $\mathscr{L}$ of linear orderings.

The tournament in Figure 2.1 can be realized by the following set of linear orderings as a 2-majority tournament:

$$\pi_1 : 2 < 1 < 4 < 5 < 3 < 7 < 6$$
$$\pi_2 : 1 < 2 < 3 < 4 < 5 < 6 < 7$$
$$\pi_3 : 1 < 3 < 2 < 6 < 7 < 5 < 4$$

There are many other configurations of linear orderings that can be used to realize the same tournament, this is just one example.

Given a tournament T, that can be realized as a k-majority tournament, it is not always easy to find linear orderings that realize T. In fact, it can be very difficult to even determine an appropriate value of k if it is not given. However, with acyclic tournaments, there is a simple way to find linear orderings that realize the tournament. To show this, we must prove several results, and explain some other properties of tournaments.

A digraph D_1 is *isomorphic* to a digraph D_2, denoted $D_1 \cong D_2$, if there exists a bijective function $\phi : V(D_1) \to V(D_2)$ such that $(u, v) \in A(D_1)$ if and only if $(\phi(u), \phi(v)) \in A(D_2)$ [CLZ11].

Lemma 2.3. *If D is an acyclic digraph, then D contains a vertex u such that $od(u) = 0$ and a vertex v such that $id(v) = 0$.*

Proof. Suppose D is an acyclic digraph, but no such u or no such v exists. Consider the case in which there is no vertex u in D such that $od(u) = 0$, and there is a vertex v such that $id(v) = 0$. Then, every vertex in D has out degree at least one. We know we have a vertex v with in degree zero, and it must have out degree at least one. So, v must be adjacent to some vertex a in D. The vertex a must also have out degree at least one, so a must be adjacent to some vertex b in D. Now, this vertex b must have out degree at

least one, and it cannot be adjacent to v or a, so it must be adjacent to some vertex c in D.

This pattern will continue for c and every successive vertex. No vertex can be adjacent to v, since it has in degree zero, and no vertex can be adjacent to any previous vertex, since that would create a cycle. But, the order of D is finite, so some vertex x in D must be adjacent to v or to one of the previous vertices, which is a contradiction.

Now consider the case in which there is a vertex u such that $od(u) = 0$, but there is no vertex v such that $id(v) = 0$. Then, every vertex must have in degree at least one. We have a vertex u with out degree zero and in degree at least one, so there is some vertex a in D which is adjacent to u. But the vertex a has in degree at least one, so there must be some vertex b in D which is adjacent to a. Then there must be a vertex c in D that is adjacent to b, and similarly for c and every successive vertex in D. But, again, D has finite order, and so there must be some vertex x which is adjacent from some previous vertex in D, creating a cycle, another contradiction.

Finally, consider the case where there is no vertex u such that $od(u) = 0$ and no vertex v such that $id(v) = 0$. Then, every vertex in D has out degree at least one and in degree at least one. So, a vertex u is adjacent to some vertex a and adjacent from some vertex b. Then the vertex a is adjacent to some other vertex c, and so on. But, D is finite, so some vertex x in D must be adjacent to a previous vertex, thus creating a cycle, a contradiction.

Therefore, if D is an acyclic digraph, then D contains a vertex u such that $od(u) = 0$ and a vertex v such that $id(v) = 0$. □

In a digraph, a *source* is a vertex v such that $id(v) = 0$, and a *sink* is a vertex v such that $od(v) = 0$. So, every acyclic tournament has a source and a sink. Further, every acyclic tournament has a unique source and a unique sink.

Theorem 2.4. *An acyclic tournament has a unique source and a unique sink.*

Proof. Let T be an acyclic tournament with vertex set V and arc set A. Then, by Lemma 2.3, there is a vertex $u \in V$ such that $od(u) = 0$ and a vertex $v \in V$ such that $id(v) = 0$. Suppose there is another vertex $a \in V$ such that $od(a) = 0$. Since T is a tournament, either ua or au is an arc in A. Then, either $od(u) \neq 0$ or $od(a) \neq 0$. So, only one vertex in V of T can have out degree 0.

Now, suppose there is another vertex $b \in V$ such that $id(b) = 0$. Since T is a tournament, either vb or bv is in A. Then, either $id(b) \neq 0$ or $id(v) \neq 0$. So, only one vertex in V of T can have in degree 0. Thus, if T is an acyclic tournament, then T has a unique source and a unique sink. □

Theorem 2.5. *A tournament T is acyclic if and only if T can be realized as a k-majority tournament such that every linear ordering is identical.*

Proof. Let T be a k-majority tournament with vertex set V. Suppose that the tournament T can be defined such that every linear ordering is identical, but T is not acyclic. Then T must contain a cycle. But, each vertex in V can only be adjacent to a vertex below it in the linear orderings, and for T to contain a cycle there must be some vertex that is adjacent to some other vertex that is above it in the linear orderings, which is a contradiction. Thus, if T can be realized such that every linear ordering is identical, then T is acyclic.

Now, suppose T is an acyclic tournament. Then, by Theorem 2.4, there exist unique vertices $s, t \in V$ such that $id(s) = 0$ and $od(t) = 0$. Then, $s \to V$ and $V \to t$ in T. This relationship can be represented by placing s at the top and t at the bottom of $2k-1$ linear orderings of V in T, to realize T as a k-majority tournament. If we delete s and t from T, we have an acyclic tournament T_1 with vertex set V_1, of cardinality $n-2$, since deleting vertices cannot create a cycle. Now, T_1 has unique vertices $s_1, t_1 \in V_1$ such that $id(s_1) = 0$ and $od(t_1) = 0$. Then, in the original tournament T, we have $s_1 \to V \backslash \{s\}$ and $V \backslash \{t\} \to t_1$. This relationship can be represented by placing s_1 below s, but above all other vertices in V, in every linear ordering and by placing t_1 above t, but below all other vertices in V, in every linear ordering. We can continue in this fashion until all the vertices in V are accounted for. If $|V|$ is odd, then the remaining vertex will be placed in the middle of each linear ordering. Thus, if T is an acyclic tournament, then T can be realized by a set of $2k-1$ linear orderings such that every linear ordering is identical.

□

Corollary 2.6. *Let T be a tournament. If T is acyclic, then T can be realized as a 2-majority tournament.*

Proof. If T is an acyclic tournament, then, by Theorem 2.5, T can be realized as a k-majority tournament such that every linear ordering is identical. If each linear ordering is identical, then we can choose any three of the linear orderings and these will define

a 2-majority tournament that is isomorphic to T. Thus, if T is an acyclic tournament, then T can be realized as a 2-majority tournament. □

Therefore, given an acyclic tournament T, there is a simple way to realize T as a 2-majority tournament. In fact, the linear orderings are formed using a Hamiltonian path in T, and when T is acyclic, T contains only one Hamiltonian path [CLZ11].

Theorem 2.7. *Every acyclic tournament contains exactly one Hamiltonian path.*

Proof. Let T be an acyclic tournament with vertex set V, and let $P : v_1v_2...v_n$ be a Hamiltonian path in T, with $v_i \in V$. Suppose that P' is also a Hamiltonian path in T. Since T is acyclic, there is a vertex v_a such that $id(v_a) = 0$, and a vertex v_b such that $od(v_b) = 0$. The vertex v_a is not adjacent from any other vertex, so v_a must be the vertex v_1 in P. But then it must also be the first vertex in P', otherwise there would be some other vertex in V that is adjacent to v_a. Similarly, the vertex v_b is not adjacent to any other vertex, so v_b must be the vertex v_n in P, and also the last vertex in P'.

Now, consider the tournament $T' = T \setminus \{v_1, v_n\}$. Deleting these two vertices in T cannot create a cycle, so T' is also acyclic. So, T' has vertices v_c and v_d such that $id(v_c) = 0$ and $od(v_d) = 0$. Thus, v_c must be the vertex v_2 in P, and v_d must be the vertex v_{n-1} in P, and similarly for P'.

Continuing in this fashion, we see that each vertex $v_i \in V$ must be in the same position in both P and P'. Thus $P = P'$, and therefore every acyclic tournament contains exactly one Hamiltonian path. □

So, any acyclic tournament T can be realized as a 2-majority tournament with three identical linear orderings that are determined by the Hamiltonian path contained in T. Actually, only two of the linear orderings need to be represented by the Hamiltonian path, and the order of the vertices in the third linear ordering is irrelevant, since T is a 2-majority tournament and the dominance of the vertices is established by the two identical linear orderings. The following are linear orderings that realize the tournament from Figure 2.1 in this way:

$$\pi_1 : 1 < 2 < 3 < 4 < 5 < 6 < 7$$
$$\pi_2 : 1 < 2 < 3 < 4 < 5 < 6 < 7$$
$$\pi_3 : 7 < 6 < 5 < 4 < 3 < 2 < 1$$

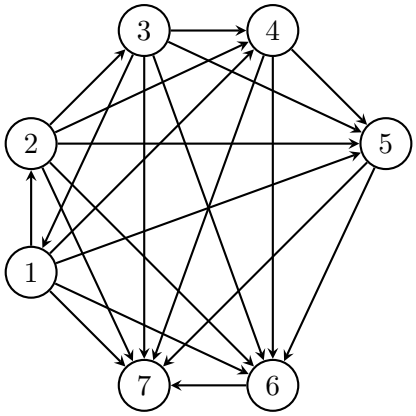
Notice, that the vertices could be in any order in π_3, and the tournament would not change.

2.3 Dominating Sets

General tournaments arise in political science when considering the theory of voting and voting paradoxes [cL97]. In the context of k-majority tournaments, each vertex represents a candidate and each linear ordering is a voter's preference ranking of the candidates. If Candidate A is ranked above Candidate B in a majority of the linear orderings, then Candidate A defeats Candidate B in the election. In some tournaments, the result is one candidate that clearly defeats all others, but, in many cases, there is no such candidate. In these scenarios, we can find a group of candidates, represented by a set of vertices, that together defeat the remaining candidates. This is called a *dominating set*. The dominating set can then be used to help determine a winner, known as the *tournament solution*.

A *dominating set* Γ of a k-majority tournament T is a nonempty set of vertices in V that dominates the entire vertex set V of T. A *minimum dominating set* is a dominating set Γ such that there is no smaller dominating set in T. For example, the tournament T in Figure 2.1 is a 2-majority tournament with vertex set $V = \{1, 2, 3, 4, 5, 6, 7\}$, and since it is acyclic, there is a vertex that has in degree 0, Vertex 1 in T, and therefore the tournament has a minimum dominating set of cardinality one, namely $\Gamma = \{1\}$. In this case, there is only one choice for the tournament solution, but that is not always what happens.

Figure 2.2 shows a 2-majority tournament with vertex set $V = \{1, 2, 3, 4, 5, 6, 7\}$. However, there is no single vertex in V that dominates all other vertices in V. A minimum dominating set for this tournament could be $\{1, 2\}$, $\{1, 3\}$, or $\{2, 3\}$. In this situation, there is no obvious solution to the tournament, so we must now use other methods to determine the tournament solution. One method, in the case of an election, could be to hold a runoff, in which only the candidates in a chosen minimum dominating set are considered. Another possible method could be to weight each of the vertices by the number of other vertices that it dominates. However, with this particular tournament, this method would not determine a solution, since each vertex in any of the minimum dominating sets would have the same weight.



$$\pi_1 : 1 < 2 < 3 < 4 < 5 < 6 < 7$$
$$\pi_2 : 3 < 1 < 2 < 4 < 5 < 6 < 7$$
$$\pi_3 : 2 < 3 < 1 < 7 < 6 < 4 < 5$$

Figure 2.2: A 2-majority tournament T and 3 linear orderings that realize T

Let $F(k)$ be the maximum size of a minimum dominating set over all k-majority tournaments. As shown in [Erd63], there are tournaments whose minimum dominating set is arbitrarily large. However, this is not the case for k-majority tournaments with a fixed value of k. Indeed, it has been shown that $F(k) \leq (80 + o(1))k \log k$, where the $o(1)$ term tends to zero as k tends to infinity [ABK$^+$06]. Also, for any k-majority tournament T with order n, any minimum dominating set of T has cardinality at most $\lceil \frac{1}{2}n \rceil$.

Theorem 2.8. *Any k-majority tournament T with order n has a minimum dominating set of cardinality at most $\lceil \frac{1}{2}n \rceil$.*

Proof. Let T be a k-majority tournament with vertex set V and order n. For every pair of vertices $u, v \in V$, either u dominates v or v dominates u, and every vertex dominates itself. If T has even order, then we can pair the vertices in V. For each pair, one of the vertices dominates the other. The set Γ of vertices that dominate their respective pair then dominates all of V in T, and this set Γ has cardinality $\frac{1}{2}n$. If T has odd order, then we can remove any one vertex x from V and proceed as if T were even, then include x in the set Γ. Then Γ dominates all of V in T. This gives us a dominating set of cardinality $\frac{1}{2}(n-1)+1$, which is the same result as $\lceil \frac{1}{2}n \rceil$. Thus, any k-majority tournament T with

order n has a minimum dominating set of cardinality at most $\lceil \frac{1}{2}n \rceil$. □

Further, it has been shown that $F(2) = 3$, and that $F(3) \geq 4$, but the exact value of $F(3)$ has not been established [ABK+06]. To show that $F(2) = 3$, Noga Alon et al. prove that three is an upper bound for minimum dominating sets over all 2-majority tournaments and that $F(2) > 2$. The following is the theorem and proof given by Noga Alon et al. to show that three is an upper bound for $F(2)$. In the proof, the linear orderings are labeled P_i and the symbols $<_i$, $>_i$, $\leq_i$, and $\geq_i$ denote the position of a vertex relative to another vertex in the linear ordering P_i. Where $v >_i u$ indicates that v lies above u in P_i and $v \leq_i u$ indicates that v is equal to or below u in P_i, and similarly for $<_i$ and $\geq_i$.

Theorem 2.9. *Every 2-majority tournament has a dominating set of size at most three. Moreover, if T does not have a dominating set of size one, then it has a dominating set of size three that induces a directed cycle.*

Proof. Consider a 2-majority tournament $T = (V, A)$ defined by the three linear orders $P_i = (V, >_i)$, $i \in [3]$. Choose the least vertex c in P_3 such that there exists a vertex $d \leq_3 c$ dominating the set $U = \{x \in V : x >_3 c\}$ of vertices strictly above c in P_3. If U is empty, i.e., c is the top element of P_3, then c is the only vertex dominating the set $\{c\}$, and so $c \to V$. Thus, we may assume that U is non-empty.

Let $D = \{x \in V \setminus U : x \to U\}$ be the (non-empty) set of vertices not in U that dominate U, and let $R = V \setminus (U \cup D \cup \{c\})$ be the set of remaining vertices. Let u_i be the maximum element of U in P_i, $i \in [2]$, and fix any $d \in D$. (See Figure 2.3)

No element of $D \setminus \{c\}$ can dominate c, since otherwise c's immediate predecessor in P_3 would have been preferred in the definition of c; hence $c \to D$. Any element $x \in V \setminus U$ satisfies $x <_3 u_1, u_2$. So, if x dominates both u_1 and u_2 then it satisfies $u_1, u_2 <_i x$ for both $i \in [2]$, and thus dominates all of U. It follows that $D = \{x \in V \setminus U : x >_1 u_1$ and $x >_2 u_2\} = \{x \in V \setminus U : x \to \{u_1, u_2\}\}$, and therefore $\{u_1, u_2\}$ dominates R. Thus $\{c, d, u_1, u_2\}$ dominates V, but we can do better. Let $R_i = \{x \in R : x <_i u_i\}$, $i \in [2]$. Then $R = R_1 \cup R_2$ and u_i dominates R_i, $i \in [2]$. Since $u_1, u_2 <_i d$ for both $i \in [2]$ and c dominates d, there exists $i \in [2]$ such that $u_i <_i c$; we may suppose $u_2 <_2 c$. Then R_2 is also dominated by c, since c is above R in P_3. It follows that $\{c, d, u_1\}$ is a dominating set for T.

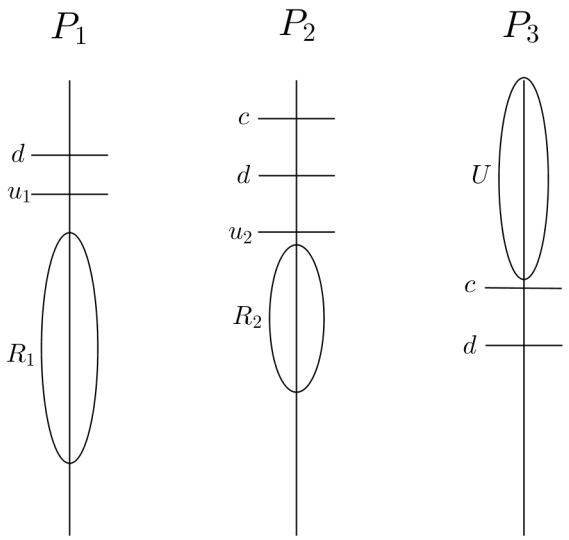


Figure 2.3: Structure of linear orders in a 2-majority tournament

Note also that if $c \in D$ then $c >_1 R_1$, and so $c \to V$. Otherwise, $c \to d \to u_1 \to c$. □

This proof is especially helpful, since it shows a certain structure that can be found in the linear orderings of k-majority tournaments. This proof shows that a 2-majority tournament has a minimum dominating set of cardinality at most three. To show the lower bound of two, Noga Alon et al. give an example of a 2-majority tournament that has property S_2. If a tournament has the property S_t, then every set U of t vertices is dominated by some vertex not in U, and a tournament T has no dominating set of cardinality t if and only if T has the property S_t [ABK$^+$06]. Similarly, the lower bound of $F(3) \geq 4$ is shown by proving that there exists a tournament with property S_3 [ABK$^+$06].

To better illustrate the structure of the linear orderings that realize k-majority tournaments shown in the proof of Theorem 2.9, consider the tournament in Figure 2.4. This tournament, T, is represented by its adjacency matrix, and is realized by three linear orderings of its vertex set $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. We could choose any of the three linear orderings to establish the structure, so we will look at the third linear ordering, π_3.

Adjacency Matrix of Tournament T

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Linear Orderings Realizing T

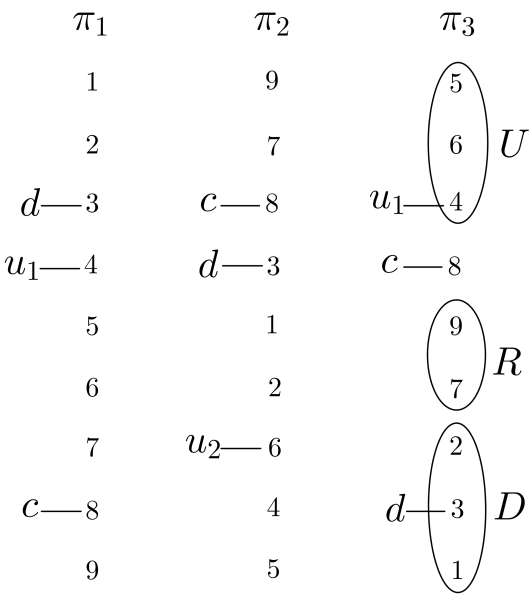


Figure 2.4: Structure example

In π_3, the vertex c, as described in the proof of Theorem 2.9, is the vertex 8. Notice, that the vertex 8 is the least vertex in π_3, such that every vertex above it is dominated by some other vertex, d, in V. Note, that there is more than one choice for d in this tournament, we will choose the vertex 3. The set $U = \{4, 5, 6\}$, is the set of vertices in V that are above 8 in π_3, and the set $D = \{1, 2, 3\}$ is the set of vertices in V such that each vertex in D dominates the set U in T. The vertex 4 is the top vertex of U in the linear ordering π_1, so 4 is the vertex u_1. The proof of Theorem 2.9 states that the set of vertices $\{c, d, u_1\}$ is a dominating set for the tournament, and the set of vertices $\{3, 4, 8\}$ is a dominating set in our example tournament T. In this particular example, the dominating set given by $\{c, d, u_1\}$ is a minimum dominating set for the tournament, but that is not the case for every tournament. Also, in this example, there are many other possible minimum dominating sets, a total of 54 of them actually, the set $\{3, 4, 8\}$ is only one of them.

2.4 Pseudo-3-Majority Tournaments

The key structural differences between 2- and 3-majority tournaments are hard to pin down. We know that if a tournament T is acyclic, then T can be realized as a 2-majority tournament. We know, from the proof of Theorem 2.9, that there is a certain structure within the linear orderings of 2-majority tournaments, and really all k-majority tournaments. We also know that if a tournament T can be realized by $2k - 1$ linear orderings, then it is a k-majority tournament. But, given linear orderings for a k-majority tournament T, how do we know that T could not be realized with a smaller value of k? In most cases, there is no way of knowing this without finding linear orderings to realize T so that it has a certain value for k.

Let T be a k-majority tournament such that T cannot be realized with a lower k value. We will call such a tournament T a *critical k-majority tournament*. For example, a 2-majority tournament is always critically 2-majority, since there is no smaller value for k that can be found. In finding $F(3)$ we are particularly interested in finding critical 3-majority tournaments. If a given 3-majority tournament T can be realized as a 2-majority tournament, we call T a *pseudo-3-majority tournament*.

Consider the tournaments in Figure 2.5. The tournaments T_1 and T_2 are 3-majority tournaments, each represented by five linear orderings that realize the respective

tournament. The tournament T_3 is a 2-majority tournament represented by three linear orderings that realize the tournament, and the tournament T_4 is represented as a graph. Actually, all of the tournaments in Figure 2.5 are just different representations of the same critical 2-majority tournament. Therefore, T_1 and T_2 are pseudo-3-majority tournaments. Further, notice in Figure 2.5, that the linear orderings π_1, π_2, and π_3 of T_2 are copies of the linear orderings of T_3, and the linear orderings π_4 and π_5 of T_2 are copies of π_1 and π_2.

Theorem 2.10. *If T is a 2-majority tournament that can be realized by the set of linear orderings $\mathscr{L} = \{\pi_1, \pi_2, \pi_3\}$, then T can be realized as a pseudo-3-majority tournament T', such that $\mathscr{L}' = \{\pi_1, \pi_2, \pi_3, \pi_i, \pi_j\}$, with $i \neq j$ and $i, j \in [3]$.*

Proof. Let T be a 2-majority tournament with vertex set V that can be realized by the set of linear orderings $\mathscr{L} = \{\pi_1, \pi_2, \pi_3\}$ of V. For any pair of vertices $u, v \in V$, if $u \to v$, then v can be above u in at most one of π_1, π_2, and π_3. Now let T' be the 3-majority tournament with the same vertex set V, as T, that can be realized by the set of linear orderings $\mathscr{L}' = \{\pi_1, \pi_2, \pi_3, \pi_i, \pi_j\}$ of V, such that $i \neq j$ and $i, j \in [3]$. Then, for the same $u, v \in V$, v can be above u in at most two of the linear orderings in $\mathscr{L}'$. Since u and v are arbitrary vertices in V, this is true for all vertices in V. Thus, T and T' are isomorphic. □

Thus, any 2-majority tournament can be realized by five linear orderings as a pseudo-3-majority tournament. It is relatively easy to find examples of critical 2-majority tournaments, since any set of three linear orderings represents some critical 2-majority tournament. Therefore, it is also easy to find examples of pseudo-3-majority tournaments. However, it is extremely difficult to find an example of a critical 3-majority tournament. In the article by Noga Alon et al., they find only one theoretical example of a 3-majority tournament, which they use to show that $F(3) \geq 4$ [ABK$^+$06].

There is no algorithm or easy way to find a critical 3-majority tournament, or to prove that a set of five linear orderings realizes a critical 3-majority tournament. One possible way stems from the proof of Theorem 2.9. In this proof, Noga Alon et al., give a structure to the linear orderings of a general 2-majority tournament, which can be extended to all k-majority tournaments. In the proof, Noga Alon et al. show that, for 2-majority tournaments, the set $\{c, d, u_1\}$ is always a dominating set for the tournament.

T_1

$\pi_1 : 1 < 2 < 6 < 3 < 0 < 5 < 4$

$\pi_2 : 2 < 4 < 6 < 1 < 3 < 5 < 0$

$\pi_3 : 3 < 0 < 4 < 1 < 5 < 6 < 2$

$\pi_4 : 4 < 3 < 5 < 0 < 2 < 6 < 1$

$\pi_5 : 5 < 6 < 0 < 1 < 2 < 3 < 4$

T_2

$\pi_1 : 0 < 1 < 2 < 3 < 4 < 5 < 6$

$\pi_2 : 5 < 2 < 6 < 3 < 0 < 4 < 1$

$\pi_3 : 4 < 6 < 1 < 3 < 5 < 0 < 2$

$\pi_4 : 0 < 1 < 2 < 3 < 4 < 5 < 6$

$\pi_5 : 5 < 2 < 6 < 3 < 0 < 4 < 1$

T_3

$\pi_1 : 0 < 1 < 2 < 3 < 4 < 5 < 6$

$\pi_2 : 5 < 2 < 6 < 3 < 0 < 4 < 1$

$\pi_3 : 4 < 6 < 1 < 3 < 5 < 0 < 2$

T_4

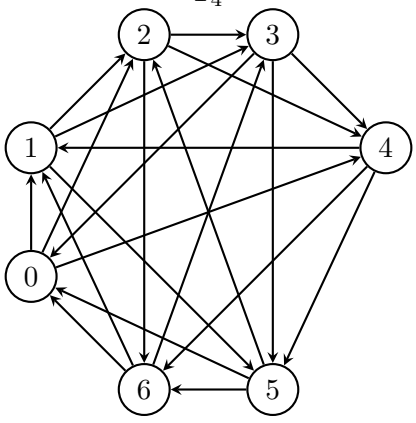


Figure 2.5: Representations of tournaments

Then, for pseudo-3-majority tournaments, this should hold as well. But, what if there were a 3-majority tournament realized by five linear orderings, such that, for at least one of the linear orderings, the set $\{c, d, u_1\}$ for that particular ordering is not a dominating set for T? Possibly, this tournament would be critically 3-majority. A set of five linear orderings, realizing a possibly critical 3-majority tournament, and the adjacency matrix of that tournament, with this property are given in Figure 2.6.

In the tournament, T, from Figure 2.6, the fifth linear ordering, π_5, has the property that the set $\{c, d, u_1\}$ is not a dominating set for T. In this ordering, $c = 1$, $U = \{3, 4\}$, and $d = 2$, and neither of the sets $\{1, 2, 3\}$ nor $\{1, 2, 4\}$ is a dominating set for T.

Thus far, we have not found a set of three linear orderings that can realize this tournament as a 2-majority tournament, however we have not proven it to be critically 3-majority either.

2.5 Inherited 2-Majority Tournaments

Every 3-majority tournament T, with vertex set V, can be realized by at least one set $\mathscr{L}$ of five linear orderings of V. For each set $\mathscr{L} = \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5\}$ that realizes T, there is a set $\mathscr{T} = \{T_1, T_2, ..., T_{10}\}$ of *inherited 2-majority tournaments* such that, for each $T_i \in \mathscr{T}$, $T_i = \{\pi_a, \pi_b, \pi_c\}$, with distinct $a, b, c \in [5]$, and for each pair of tournaments $T_i, T_j \in \mathscr{T}$, we have $T_i \neq T_j$.

For a general 3-majority tournament T, that is realized by the set of linear orderings $\mathscr{L} = \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5\}$, T has the following inherited 2-majority tournaments: $T_1 = \{\pi_1, \pi_2, \pi_3\}$, $T_2 = \{\pi_1, \pi_2, \pi_4\}$, $T_3 = \{\pi_1, \pi_2, \pi_5\}$, $T_4 = \{\pi_1, \pi_3, \pi_4\}$, $T_5 = \{\pi_1, \pi_3, \pi_5\}$, $T_6 = \{\pi_1, \pi_4, \pi_5\}$, $T_7 = \{\pi_2, \pi_3, \pi_4\}$, $T_8 = \{\pi_2, \pi_3, \pi_5\}$, $T_9 = \{\pi_2, \pi_4, \pi_5\}$, $T_{10} = \{\pi_3, \pi_4, \pi_5\}$. Notice, that each linear ordering, $\pi_i \in \mathscr{L}$, is contained in exactly six of the inherited 2-majority tournaments. Also, each pair of linear orderings, $\pi_i, \pi_j \in \mathscr{L}$, is contained in exactly three of the inherited 2-majority tournaments.

There are some relationships between 3-majority tournaments and their inherited 2-majority tournaments that reveal some information about the dominance among the vertices in the tournaments, and about the dominating sets for certain 3-majority tournaments.

$$\pi_1 : 7 < 8 < 5 < 3 < 6 < 1 < 2 < 9 < 10 < 4$$
$$\pi_2 : 4 < 9 < 10 < 8 < 5 < 6 < 1 < 2 < 3 < 7$$
$$\pi_3 : 9 < 10 < 2 < 3 < 6 < 7 < 1 < 4 < 8 < 5$$
$$\pi_4 : 5 < 6 < 7 < 8 < 1 < 2 < 4 < 3 < 9 < 10$$
$$\pi_5 : 4 < 3 < 1 < 2 < 9 < 10 < 5 < 6 < 7 < 8$$

$$\begin{bmatrix}
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0
\end{bmatrix}$$

Figure 2.6: Possible critical 3-majority tournament

Theorem 2.11. *Let T be a 3-majority tournament, with vertex set V, that is realized by a set $\mathscr{L}$ of five linear orderings of V, and let T_i and T_j be two of the inherited 2-majority tournaments of T that intersect in only one linear ordering. If T_i and T_j have the same dominating set of cardinality one, then T has that same dominating set of cardinality one.*

Proof. Let T be a 3-majority tournament, with vertex set V, that is realized by a set $\mathscr{L}$ of five linear orderings of V, and T has the set of inherited 2-majority tournaments $\mathscr{T} = \{T_1, T_2, ..., T_{10}\}$. Suppose $\{x\}$ is a dominating set for T_i and T_j, with $i, j \in [10]$ and $i \neq j$, such that T_i and T_j intersect in only one linear ordering. Then, $x \in V$ dominates all of V in T_i and in T_j. Let $a \in V$ be an arbitrary vertex. Since x dominates V in T_i, x is above a in at least two of the linear orderings in T_i. Similarly, x must be above a in at least two of the linear orderings in T_j. Since T_i and T_j intersect in only one linear ordering, there must be at least three linear orderings in $\mathscr{L}$ in which x dominates a and, since a is arbitrary, this is true for every vertex in V. Thus, x dominates every vertex in V of T, and therefore $\{x\}$ is a dominating set for T. □

Theorem 2.12. *Let T be a 3-majority tournament, with vertex set V, that is realized by a set $\mathscr{L}$ of five linear orderings of V, and let u and v be vertices in V, with $u \neq v$. Then, u dominates v in T if and only if u dominates v in at least four of the inherited 2-majority tournaments of T.*

Proof. Let T be a 3-majority tournament, with vertex set V, that is realized by a set $\mathscr{L} = \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5\}$ of five linear orderings of V, and let u and v be vertices in V such that $u \neq v$. Suppose that u dominates v in T. Then u lies above v in at least three of the linear orderings in $\mathscr{L}$. Without loss of generality, we can assume that u lies above v in π_1, π_2, and π_3. Let T_1, T_2, T_3, and T_4 be four of the inherited 2-majority tournaments of T, where $T_1 \sim \{\pi_1, \pi_2, \pi_3\}$, $T_2 \sim \{\pi_1, \pi_2, \pi_4\}$, $T_3 \sim \{\pi_1, \pi_2, \pi_5\}$, and $T_4 \sim \{\pi_1, \pi_3, \pi_4\}$. Then u dominates v in each of T_1, T_2, T_3, and T_4. Thus, if u dominates v in T then u dominates v in at least four of the inherited 2-majority tournaments of T.

Now, suppose that u dominates v in at least four of the inherited 2-majority tournaments of T. Let T_1 be one of these inherited 2-majority tournaments. If u dominates v in T_1, then u is above v in at least two of the linear orderings that realize T_1. If u is above v in three of the orderings, then we are done, so we will assume that u lies above v in exactly two of the linear orderings that realize T_1. There are two other inherited

2-majority tournaments of T that share those two linear orderings in which u lies above v, say T_2 and T_3. So, u dominates v in T_2 and T_3 as well. Let $\mathscr{L}'$ be the union of the linear orderings in $\mathscr{L}$ that realize tournaments T_1, T_2, and T_3. Now, any other inherited 2-majority tournament can only contain at most one of the two linear orderings in $\mathscr{L}'$ in which u lies above v. So, if u dominates v in a fourth inherited 2-majority tournament of T, say T_4, then u is above v in at least two of the linear orderings that realize T_4, only one of which can be in $\mathscr{L}'$. Therefore, if u dominates v in at least four of the inherited 2-majority tournaments of T, then u lies above v in at least three of the linear orderings in $\mathscr{L}$, and thus u dominates v in T. □

Corollary 2.13. *Let T be a 3-majority tournament, with vertex set V, that is realized by a set $\mathscr{L}$ of five linear orderings of V, and let u and v be vertices in V, with $u \neq v$. If u dominates v in at least four of the inherited 2-majority tournaments of T, then u dominates v in at least seven of the inherited 2-majority tournaments of T.*

Proof. Let T be a 3-majority tournament, with vertex set V, that is realized by a set $\mathscr{L}$ of five linear orderings of V, and let u and v be vertices in V such that $u \neq v$. Suppose that u dominates v in at least four of the inherited 2-majority tournaments of T, but in less than seven. Then, by Theorem 2.12, u dominates v in T. But, if u dominates v in less than seven of the inherited 2-majority tournaments, then v dominates u in at least four of the inherited 2-majority tournaments. Then, by Theorem 2.12, v dominates u in T, which is not possible. Thus, if u dominates v in at least four of the inherited 2-majority tournaments of T, then u dominates v in at least seven of the inherited 2-majority tournaments of T. □

Theorem 2.14. *Let T be a 3-majority tournament, with vertex set V, that is realized by a set $\mathscr{L}$ of five linear orderings of V. If a set X is a dominating set for every inherited 2-majority tournament of T, where $|X| \leq 3$, then X is a dominating set for T.*

Proof. Let T be a 3-majority tournament, with vertex set V, that is realized by a set $\mathscr{L}$ of five linear orderings of V. Then T has a set $\mathscr{T}$ of ten inherited 2-majority tournaments. Let the set X be a dominating set for every inherited 2-majority tournament in $\mathscr{T}$. If $|X| = 1$, then by Theorem 2.11, X is a dominating set for T. So, we will assume $|X| > 1$.

Suppose $X = \{x, y, z\}$, with $x, y, z \in V$, is a dominating set for every inherited 2-majority tournament in $\mathscr{T}$, where at least two of the vertices in X are distinct. So,

$|X| = 2$ or $|X| = 3$. Then, each vertex x, y, and z dominates some subset of V in each inherited 2-majority tournament. Let u be an arbitrary vertex in V. If any vertex in X dominates u in four or more of the inherited 2-majority tournaments in $\mathscr{T}$, then, by Theorem 2.12, that vertex dominates u in T. So, suppose each vertex in X dominates u in exactly three of the inherited 2-majority tournaments. This leaves one inherited 2-majority tournament in which no vertex in X dominates u. Since X is a dominating set for all of the inherited 2-majority tournaments, some vertex in X must dominate u in the remaining inherited 2-majority tournament. So, some vertex in X must dominate u in at least four of the inherited 2-majority tournaments, and thus, by Theorem 2.12, some vertex in X must dominate u in T. Therefore, X must dominate all of V in T. □

Note that although X is a dominating set for T, it may not be a minimum dominating set for T.

Theorem 2.15. *Let T be a 3-majority tournament, with vertex set V, that is realized by a set $\mathscr{L}$ of five linear orderings of V. For any pair of vertices $u, v \in V$, if $u \to v$ in T and u lies above v in at least four of the five linear orderings in $\mathscr{L}$, then u dominates v in every inherited 2-majority tournament of T.*

Proof. Let T be a 3-majority tournament, with vertex set V, that is realized by a set $\mathscr{L}$ of five linear orderings of V. Let u and v be vertices in V, with $u \to v$. Suppose u lies above v in four of the linear orderings of T, then v lies above u in one of the linear orderings. Every inherited 2-majority tournament of T contains three of the linear orderings in $\mathscr{L}$, and v can lie above u in at most one of those linear orderings for each inherited 2-majority tournament. So, u must lie above v in at least two of the linear orderings in each inherited 2-majority tournament. Thus, u dominates v in every inherited 2-majority tournament of T. □

Corollary 2.16. *Let T be a 3-majority tournament, with vertex set V, that is realized by a set $\mathscr{L}$ of five linear orderings of V. If for every pair of vertices $u, v \in V$, such that $u \to v$, the vertex u lies above v in at least four of the five linear orderings in $\mathscr{L}$, then T can be realized as one of its inherited 2-majority tournaments.*

Proof. Let T be a 3-majority tournament, with vertex set V, that is realized by a set $\mathscr{L}$ of five linear orderings of V. For every pair of vertices $u, v \in V$, where $u \to v$, let

u lie above v in four of the linear orderings of V that realize T. Then, any set of three linear orderings in $\mathscr{L}$ contain at least two linear orderings in which u lies above v. Thus, every $T_i \in \mathscr{T}$ is isomorphic to T. Therefore, T can be realized by any of its inherited 2-majority tournaments. □

So, we can see that, in some cases, the dominance among the vertices, and the dominating sets, of 3-majority tournaments and their inherited 2-majority tournaments are linked.

2.6 Computer Programs

Many of the computations and analysis done in mathematics are not practical to do by hand. Thus the aid of a computer is often necessary. Computer programs can substantially reduce the time needed to analyze or calculate information, and thus make research much easier to accomplish.

In researching k-majority tournaments, we need to examine many examples to find patterns and information that may help reveal the structure in the linear orderings of those tournaments and to determine if a given set of linear orderings realizes a certain tournament. Analyzing a single set of five linear orderings, with as few as ten vertices in the vertex set of the tournament, can take a while by hand, and there is a lot of room for error. This only increases in complexity as the number of vertices increases, or as the number of linear orderings increases, or both. Therefore, the implementation of a computer program to help analyze the tournaments and their linear orderings is crucial to the research process.

There are two common situations in which we may need to analyze a certain k-majority tournament. The first, is when we have the graph or adjacency matrix of a tournament and we want to know the minimum dominating sets for that tournament. The second, is when we have a set of linear orderings. In this situation we may need to know if this particular set of linear orderings realizes a certain tournament. We may also need to know the minimum dominating sets for the tournament that is realized by the set of linear orderings.

The particular program being used was written in the Python programming language, and the code for this program can be found in Appendix $A.1$. The program

first asks the user to input the order, n, of the tournament and the starting vertex, s. The starting vertex, s, is the lowest numerical value used to represent a vertex in the tournament. This program requires the vertex set V of the tournament T to be represented by integers, such that all integers from s to $n+s-1$, inclusive, are contained in V. The program then asks the user to input "1" for linear orderings or "2" for the adjacency matrix, depending on what information is to be input and analyzed.

If the user chooses "1", the program asks for the number of linear orderings and then prompts the user to input the linear orderings, one at a time. The program will then output a list of what each vertex dominates in the k-majority tournament T, the adjacency matrix for T, all of the minimum dominating sets of T, and a list of all triangles in T. For example consider the set of linear orderings for T_3 in Figure 2.5. The program output, and input, for this tournament is given below:

```
Input order of the tournament: 7
Input the starting vertex value: 0
Enter "1" for linear orderings or "2" for adjacency matrix: 1
Input number of linear orderings: 3
Linear ordering 1 :
: 0
: 1
: 2
: 3
: 4
: 5
: 6
Linear ordering 2 :
: 5
: 2
: 6
: 3
: 0
: 4
```

```
: 1
Linear ordering 3 :
: 4
: 6
: 1
: 3
: 5
: 0
: 2
What each vertex dominates:
0 -> [0, 1, 2, 4]
1 -> [1, 2, 3, 5]
2 -> [2, 3, 4, 6]
3 -> [0, 3, 4, 5]
4 -> [1, 4, 5, 6]
5 -> [0, 2, 5, 6]
6 -> [0, 1, 3, 6]
Adjacency Matrix:
[0, 1, 1, 0, 1, 0, 0]
[0, 0, 1, 1, 0, 1, 0]
[0, 0, 0, 1, 1, 0, 1]
[1, 0, 0, 0, 1, 1, 0]
[0, 1, 0, 0, 0, 1, 1]
[1, 0, 1, 0, 0, 0, 1]
[1, 1, 0, 1, 0, 0, 0]
Dominating Sets:
[0, 1, 2]
[0, 1, 4]
[0, 1, 5]
[0, 1, 6]
[0, 2, 3]
[0, 2, 4]
```

```
[0, 2, 5]
[0, 3, 4]
[0, 3, 5]
[0, 3, 6]
[0, 4, 6]
[0, 5, 6]
[1, 2, 3]
[1, 2, 5]
[1, 2, 6]
[1, 3, 4]
[1, 3, 5]
[1, 3, 6]
[1, 4, 5]
[1, 4, 6]
[2, 3, 4]
[2, 3, 6]
[2, 4, 5]
[2, 4, 6]
[2, 5, 6]
[3, 4, 5]
[3, 5, 6]
[4, 5, 6]
Triangles:
0 , 1 , 3
0 , 1 , 5
0 , 2 , 3
0 , 2 , 6
0 , 4 , 5
0 , 4 , 6
1 , 2 , 4
1 , 2 , 6
1 , 3 , 4
```

```
1 , 5 , 6
2 , 3 , 5
2 , 4 , 5
3 , 4 , 6
3 , 5 , 6
```

If the user chooses "2", the program prompts the user to input the adjacency matrix, one row at a time. The program then outputs all of the minimum dominating sets for the k-majority tournament T, and a list of all triangles in T. For example, consider the matrix for the tournament in Figure 2.6. The program output, and input, is given below:

```
Input order of the tournament: 10
Input the starting vertex value: 1
Enter "1" for linear orderings or "2" for adjacency matrix: 2
row 1 :
: 0
: 1
: 0
: 1
: 0
: 0
: 0
: 0
: 1
: 1
row 2 :
: 0
: 0
: 1
: 1
: 0
: 0
```

: 1
: 0
: 1
: 1
row 3 :
: 1
: 0
: 0
: 0
: 0
: 1
: 1
: 0
: 1
: 1
row 4 :
: 0
: 0
: 1
: 0
: 1
: 0
: 0
: 1
: 1
: 1
row 5 :
: 1
: 1
: 1
: 0
: 0

: 1
: 1
: 0
: 0
: 0
row 6 :
: 1
: 1
: 0
: 1
: 0
: 0
: 1
: 1
: 0
: 0
row 7 :
: 1
: 0
: 0
: 1
: 0
: 0
: 0
: 1
: 0
: 0
row 8 :
: 1
: 1
: 1
: 0

```
: 1
: 0
: 0
: 0
: 0
: 0
row 9 :
: 0
: 0
: 0
: 0
: 1
: 1
: 1
: 1
: 0
: 1
row 10 :
: 0
: 0
: 0
: 0
: 1
: 1
: 1
: 1
: 0
: 0
Dominating Sets:
[4, 5]
[4, 6]
Triangles:
```

1 , 2 , 3
1 , 2 , 7
1 , 4 , 3
1 , 4 , 5
1 , 4 , 8
1 , 9 , 5
1 , 10 , 5
1 , 9 , 6
1 , 10 , 6
1 , 9 , 7
1 , 10 , 7
1 , 9 , 8
1 , 10 , 8
2 , 3 , 6
2 , 4 , 5
2 , 4 , 8
2 , 9 , 5
2 , 10 , 5
2 , 9 , 6
2 , 10 , 6
2 , 7 , 8
2 , 9 , 8
2 , 10 , 8
3 , 6 , 4
3 , 7 , 4
3 , 9 , 5
3 , 10 , 5
3 , 6 , 8
3 , 7 , 8
3 , 9 , 8
3 , 10 , 8
4 , 5 , 6

```
4 , 5 , 7
4 , 9 , 6
4 , 10 , 6
4 , 9 , 7
4 , 10 , 7
5 , 6 , 8
5 , 7 , 8
```

Another instance in which a computer program would be useful is when we would like to analyze a 3-majority tournament and all of its inherited 2-majority tournaments. This could be done using the previous program, but that would require inputting information for 11 different tournaments. So, one program that could analyze all of these tournaments at once would be quite helpful. The code for this program can be found in Appendix *A*.2. This program outputs a lot of information, so we will try it out with a small 3-majority tournament. The output, and input, for this example is given below:

```
Input order of the tournament: 3
Input the starting vertex value: 1
Input linear ordering 1 :
: 1
: 2
: 3
Input linear ordering 2 :
: 3
: 1
: 2
Input linear ordering 3 :
: 2
: 3
: 1
Input linear ordering 4 :
: 1
: 2
```

```
: 3
Input linear ordering 5 :
: 3
: 1
: 2
Linear Orderings:
[1, 2, 3]
[3, 1, 2]
[2, 3, 1]
[1, 2, 3]
[3, 1, 2]
What each vertex dominates in the 3-majority tournament:
1 -> [1, 2]
2 -> [2, 3]
3 -> [1, 3]
Adjacency Matrix:
[0, 1, 0]
[0, 0, 1]
[1, 0, 0]
Dominating Sets:
[1, 2]
[1, 3]
[2, 3]
Triangles:
1 , 2 , 3
Inherited 2-majority tournaments:
T 1 : [[1, 2, 3], [3, 1, 2], [2, 3, 1]]
T 2 : [[1, 2, 3], [3, 1, 2], [1, 2, 3]]
T 3 : [[1, 2, 3], [3, 1, 2], [3, 1, 2]]
T 4 : [[1, 2, 3], [2, 3, 1], [1, 2, 3]]
T 5 : [[1, 2, 3], [2, 3, 1], [3, 1, 2]]
T 6 : [[1, 2, 3], [1, 2, 3], [3, 1, 2]]
```

```
T 7 :  [[3, 1, 2], [2, 3, 1], [1, 2, 3]]
T 8 :  [[3, 1, 2], [2, 3, 1], [3, 1, 2]]
T 9 :  [[3, 1, 2], [1, 2, 3], [3, 1, 2]]
T 10 :  [[2, 3, 1], [1, 2, 3], [3, 1, 2]]
What each vertex dominates in T 1 :
1 -> [1, 2]
2 -> [2, 3]
3 -> [1, 3]
What each vertex dominates in T 2 :
1 -> [1, 2, 3]
2 -> [2, 3]
3 -> [3]
What each vertex dominates in T 3 :
1 -> [1, 2]
2 -> [2]
3 -> [1, 2, 3]
What each vertex dominates in T 4 :
1 -> [1, 2, 3]
2 -> [2, 3]
3 -> [3]
What each vertex dominates in T 5 :
1 -> [1, 2]
2 -> [2, 3]
3 -> [1, 3]
What each vertex dominates in T 6 :
1 -> [1, 2, 3]
2 -> [2, 3]
3 -> [3]
What each vertex dominates in T 7 :
1 -> [1, 2]
2 -> [2, 3]
3 -> [1, 3]
```

```
What each vertex dominates in T 8 :
1 -> [1, 2]
2 -> [2]
3 -> [1, 2, 3]
What each vertex dominates in T 9 :
1 -> [1, 2]
2 -> [2]
3 -> [1, 2, 3]
What each vertex dominates in T 10 :
1 -> [1, 2]
2 -> [2, 3]
3 -> [1, 3]
Dominating sets for T 1 :
[1, 2]
[1, 3]
[2, 3]
Dominating sets for T 2 :
[1]
Dominating sets for T 3 :
[3]
Dominating sets for T 4 :
[1]
Dominating sets for T 5 :
[1, 2]
[1, 3]
[2, 3]
Dominating sets for T 6 :
[1]
Dominating sets for T 7 :
[1, 2]
[1, 3]
[2, 3]
```

```
Dominating sets for T 8 :
[3]
Dominating sets for T 9 :
[3]
Dominating sets for T 10 :
[1, 2]
[1, 3]
[2, 3]
```

Chapter 3

Quadratic Residue Tournaments

3.1 Quadratic Reciprocity

In this section, we provide results concerning the Law of Quadratic Reciprocity. These results can be found in most textbooks on elementary number theory. Our development of the theory parallels that found in [Bur11].

The Quadratic Reciprocity Law deals with the solvability of quadratic congruences of the form $ax^2 + bx + c \equiv 0 \ (mod\ p)$, where p is an odd prime and $gcd(a, p) = 1$. Finding the solutions for this quadratic congruence can be done by finding the solutions to $x^2 \equiv a \ (mod\ p)$. Basically, this involves finding perfect squares modulo p. So, an integer $a \in \mathbb{Z}_p$ is called a *quadratic residue of* p if a is congruent to a perfect square modulo p. If an integer $a \in \mathbb{Z}_p$ is not congruent to a perfect square modulo p, then a is called a *quadratic non-residue of* p. For our purposes, we will exclude 0 as a possible value for a. So, we will actually be working with the set of integers modulo p excluding 0, denoted $\mathbb{Z}_p*$.

As an example, consider $p = 11$. To find the quadratic residues of 11 we will examine the squares of the elements of $\mathbb{Z}_{11}*$.

$1^2 \equiv 10^2 \equiv 1 \quad (mod\ 11)$

$2^2 \equiv 9^2 \equiv 4 \quad (mod\ 11)$

$3^2 \equiv 8^2 \equiv 9 \quad (mod\ 11)$

$4^2 \equiv 7^2 \equiv 5 \quad (mod\ 11)$

$5^2 \equiv 6^2 \equiv 3 \quad (mod\ 11)$

Thus, $\{1, 3, 4, 5, 9\}$ is the set of quadratic residues of 11, and $\{2, 6, 7, 8, 10\}$ is the set of quadratic non-residues of 11.

There are several results involving quadratic residues that will be useful in our research. First, we need to define the *Legendre symbol*. Let p be an odd prime with $gcd(a, p) = 1$. The *Legendre symbol* (a/p) is defined such that $(a/p) = 1$ if a is a quadratic residue of p, and $(a/p) = -1$ if a is a quadratic non-residue of p. The following results are taken directly from the text *Elementary Number Theory* by David Burton [Bur11].

Theorem 3.1. *Let p be an odd prime and let a and b be integers that are relatively prime to p. Then the Legendre symbol has the following properties:*

(a) If $a \equiv b \pmod p$, then $(a/p) = (b/p)$

(b) $(a^2/p) = 1$

(c) $(a/p) \equiv a^{(p-1)/2} \pmod p$

(d) $(ab/p) = (a/p)(b/p)$

(e) $(1/p) = 1$ and $(-1/p) = (-1)^{(p-1)/2}$

Corollary 3.2. *If p is an odd prime, then*

$$(-1/p) = \begin{cases} 1 & \text{if } p \equiv 1 \pmod 4 \\ -1 & \text{if } p \equiv 3 \pmod 4 \end{cases}$$

Hence, the quadratic congruence $x^2 \equiv -1 \pmod p$ has a solution for an odd prime p if and only if p is of the form $4k + 1$.

Theorem 3.3. *If p is an odd prime, then*

$$\sum_{a=1}^{p-1} (a/p) = 0$$

Hence, there are precisely $(p - 1)/2$ quadratic residues and $(p - 1)/2$ quadratic non-residues of p.

Theorem 3.4 (Quadratic Reciprocity Law)**.** *If p and q are distinct odd primes, then*

$$(p/q)(q/p) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}$$

Corollary 3.5. *If p and q are distinct odd primes, then*

$$(p/q)(q/p) = \begin{cases} 1 & \text{if } p \equiv 1 \ (mod\ 4) \text{ or } q \equiv 1 \ (mod\ 4) \\ -1 & \text{if } p \equiv q \equiv 3 \ (mod\ 4) \end{cases}$$

Corollary 3.6. *If p and q are distinct odd primes, then*

$$(p/q) = \begin{cases} (q/p) & \text{if } p \equiv 1 \ (mod\ 4) \text{ or } q \equiv 1 \ (mod\ 4) \\ -(q/p) & \text{if } p \equiv q \equiv 3 \ (mod\ 4) \end{cases}$$

Theorem 3.7. *If p is an odd prime, then*

$$(2/p) = \begin{cases} 1 & \text{if } p \equiv 1 \ (mod\ 8) \text{ or } p \equiv 7 \ (mod\ 8) \\ -1 & \text{if } p \equiv 3 \ (mod\ 8) \text{ or } p \equiv 5 \ (mod\ 8) \end{cases}$$

3.2 Quadratic Residue Tournaments

To construct tournaments using quadratic residues, we will consider only odd primes p such that $p \equiv 3 \ (mod\ 4)$. So, let p be an odd prime such that $p \equiv 3 \ (mod\ 4)$, with the set of quadratic residues $Q = \{q_1, q_2, ..., q_{\frac{p-1}{2}}\}$, such that $q_j \equiv x^2 \ (mod\ p)$ for some $x \in \mathbb{Z}_p*$. Recall, from Theorem 3.3, that each odd prime p has $(p-1)/2$ quadratic residues and $(p-1)/2$ quadratic non-residues. Let the *quadratic residue tournament* QRT_p have vertex set $V = \{0, 1, ..., p-1\}$ and arc set $A = \{uv : u, v \in V \text{ and } u + q_j \equiv v \ (mod\ p)$, for some $j \in [1, (p-1)/2]\}$. Thus, a vertex $u \in V$ dominates another vertex $v \in V$ if and only if v is congruent, modulo p, to the sum of u and a quadratic residue $q_j \in Q$.

The reason for considering only odd primes p such that $p \equiv 3 \ (mod\ 4)$, and not $p \equiv 1 \ (mod\ 4)$, can be seen by reviewing Corollary 3.2. By Corollary 3.2, -1 is a quadratic residue of p if and only if $p \equiv 1 \ (mod\ 4)$. If -1 is a quadratic residue of p, then $p-1$ is a quadratic residue of p. Also, note that for any odd prime p, 1 is always a quadratic residue of p, since $1^2 \equiv 1 \ (mod\ p)$ for any odd prime p. If we considered odd primes p such that $p \equiv 1 \ (mod\ 4)$, we would have the set of quadratic residues $Q = \{1, ..., p-1\}$. Then, by our construction of QRT_p, we have $0 \to \{1, ..., p-1\}$ and $1 \to \{2, ..., 1+(p-1)\}$. But, $1 + (p-1) \equiv p \equiv 0 \ (mod\ p)$ and we cannot have $0 \to 1$ and $1 \to 0$ in a tournament. So, $p-1$ cannot be a quadratic residue of p, and thus we must have $p \equiv 3 \ (mod\ 4)$.

For an example of a quadratic residue tournament, consider $p = 7$. It is simple to find that 7 has the quadratic residue set $Q = \{1, 2, 4\}$. The quadratic residue tournament

QRT_7 then has vertex set $V = \{0, 1, 2, 3, 4, 5, 6\}$, such that $0 \to \{1, 2, 4\}$, $1 \to \{2, 3, 5\}$, $2 \to \{3, 4, 6\}$, $3 \to \{4, 5, 0\}$, $4 \to \{5, 6, 1\}$, $5 \to \{6, 0, 2\}$, and $6 \to \{0, 1, 3\}$. In addition, by definition of dominance in a tournament, each vertex dominates itself as well. The graph of QRT_7 is given in Figure 2.5.

By the way they are constructed, every quadratic residue tournament, QRT_p, with vertex set V, has the property that for each pair of vertices $u, v \in V$, $id(u) = id(v)$, $od(u) = od(v)$, and $id(u) = od(u)$. Specifically, for $u \in V$, $id(u) = od(u) = (p-1)/2$. The value $(p-1)/2$ is quite significant, as it defines many aspects of the quadratic residue tournaments. When p is an odd prime, and $p \equiv 3 \ (mod\ 4)$, the value of $(p-1)/2 \ (mod\ 4)$ is either congruent to 1 or 3.

Theorem 3.8. *Let p be an odd prime such that $p \equiv 3 \ (mod\ 4)$. Then, $(p-1)/2 \equiv 1 \ (mod\ 4)$ or $(p-1)/2 \equiv 3 \ (mod\ 4)$.*

Proof. Let p be an odd prime such that $p \equiv 3 \ (mod\ 4)$. Then, $p = 3 + 4m$, for $m \in \mathbb{Z}$. So, if m is even, then $p = 3 + 4(2k)$, and if m is odd, then $p = 3 + 4(2k+1)$, for $k \in \mathbb{Z}$.

If $p = 3 + 4(2k) = 3 + 8k$, then

$$\frac{p-1}{2} = \frac{(3+8k)-1}{2} = \frac{2+8k}{2} = 1 + 4k$$
$$\text{and } 1 + 4k \equiv 1 \ (mod\ 4)$$

Thus, if $p = 3 + 4(2k)$, then $(p-1)/2 \equiv 1 \ (mod\ 4)$.

If $p = 3 + 4(2k+1) = 7 + 8k$, then

$$\frac{p-1}{2} = \frac{(7+8k)-1}{2} = \frac{6+8k}{2} = 3 + 4k$$
$$\text{and } 3 + 4k \equiv 3 \ (mod\ 4).$$

Thus, if $p = 3 + 4(2k+1)$, then $(p-1)/2 \equiv 3 \ (mod\ 4)$. Therefore, $(p-1)/2 \equiv 1 \ (mod\ 4)$ or $(p-1)/2 \equiv 3 \ (mod\ 4)$. □

The quadratic residue tournament QRT_7 can be realized as a 2-majority tournament, as shown in Figure 2.5. The quadratic residue tournaments QRT_{11}, QRT_{19}, QRT_{23}, QRT_{31}, and QRT_{47} can also be realized as k-majority tournaments, for $k = (p+1)/4$. It may be possible to realize all quadratic residue tournaments, QRT_p, with p an odd prime such that $p \equiv 3 \ (mod\ 4)$, as k-majority tournaments, for $k = (p+1)/4$. However, the way the linear orderings are obtained varies depending on whether $(p-1)/2 \equiv 1 \ (mod\ 4)$ or $(p-1)/2 \equiv 3 \ (mod\ 4)$.

3.3 Realizing Quadratic Residue Tournaments with $(p-1)/2 \equiv 3 \ (mod\ 4)$

In constructing the linear orderings to realize a quadratic residue tournament, QRT_p, as a k-majority tournament, each quadratic residue $q_j \in Q$ of p will correspond to a different linear ordering, as there will be $(p-1)/2$ linear orderings of V in QRT_p. For every QRT_p, the first quadratic residue is $q_1 = 1$, thus the first linear ordering will be based on this value. Meaning, we will sequence the ordering of the vertices in the vertex set V by adding 1. We will start with the vertex 0, and continue adding 1, modulo p, until all the vertices in V are included in the ordering. Thus, we have $\pi_1 : 0 < 1 < 2 < ... < p-1$. We will follow this procedure for each $q_j \in Q$. Then we have $\pi_2 : 0 < q_2 < 2q_2 < ... < (p-1)q_2 \ (mod\ p)$, $\pi_3 : 0 < q_3 < 2q_3 < ... < (p-1)q_3 \ (mod\ p)$, and so on, up to $\pi_{\frac{p-1}{2}} : 0 < q_{\frac{p-1}{2}} < 2q_{\frac{p-1}{2}} < ... < (p-1)q_{\frac{p-1}{2}} \ (mod\ p)$.

This construction creates linear orderings such that each π_j represents a *Hamiltonian cycle* in QRT_p. By definition, for $u, v \in V$ of QRT_p, $u \to v$ if and only if $u + q_j \equiv v \ (mod\ p)$. Each π_j is constructed such that each vertex is congruent to the sum of q_j and the previous vertex in π_j modulo p, and thus each vertex dominates the following vertex in the linear ordering. The last vertex in the linear ordering, $(p-1)q_j \ (mod\ p)$, dominates the first vertex, 0, since $(p-1)q_j + q_j \equiv q_j(p-1+1) \equiv q_j(p) \equiv 0 \ (mod\ p)$. Thus, creating a cycle containing all of the vertices in V of QRT_p, which is a Hamiltonian cycle.

For every k-majority tournament T, the vertices in T have a certain position within each linear ordering in $\mathscr{L}$ of T. For each $v_i \in V$, let l_{i_j} be the position of v_i in π_j, where $l_{i_j} \in [0, n-1]$ such that the 0 position is first, and the $n-1$ position is last, in each linear ordering. The positions of the vertices in the linear orderings for QRT_p, thus far, are such that

$$l_{i_j} q_j \equiv v_i \ (mod\ p)$$
$$l_{i_j} \equiv \tfrac{v_i}{q_j} \ (mod\ p).$$

Next, in our construction, we want to shift the vertices in each linear ordering in QRT_p so that the $(p-1)/2$ vertex is in the $(p-1)/2$ position in each linear ordering. Currently, the position of the $(p-1)/2$ vertex is $l_{(\frac{p-1}{2})_j} \equiv \frac{(p-1)/2}{q_j} \ (mod\ p)$. We want

$(p-1)/2$ to have the position $l_{(\frac{p-1}{2})_j} \equiv (p-1)/2 \ (mod\ p)$. So, we need to shift the position of the vertices in each linear ordering up by s_j, where each s_j corresponds to a linear ordering π_j. The value of s_j will be determined by

$$\begin{gathered} s_j \equiv \tfrac{(p-1)/2}{q_j} - (p-1)/2 \ (mod\ p) \\ s_j q_j \equiv (p-1)/2 - ((p-1)/2)q_j \ (mod\ p) \\ s_j q_j \equiv ((p-1)/2)(1-q_j) \ (mod\ p). \end{gathered}$$

After this shift, the position of each vertex $v_i \in V$ is $l_{i_j} \equiv \frac{v_i}{q_j} - s_j \ (mod\ p)$, for each linear ordering π_j, with corresponding q_j and s_j. These linear orderings now realize QRT_p as a k-majority tournament, with $k = (p+1)/4$.

Also, note that each of these linear orderings still represents a Hamiltonian cycle in QRT_p, since the order of the vertices within the cycle was not changed, only the position of the vertices within the representation of the cycle was changed.

As an example of this algorithm, consider again QRT_7. We know that $Q = \{1, 2, 4\}$, so initially we have:

$$\begin{gathered} \pi_1 : 0 < 1 < 2 < 3 < 4 < 5 < 6 \\ \pi_2 : 0 < 2 < 4 < 6 < 1 < 3 < 5 \\ \pi_3 : 0 < 4 < 1 < 5 < 2 < 6 < 3 \end{gathered}$$

Next, we will perform the shift. For π_1, we will always have $s_1 = 0$, so π_1 will not change. For π_2, we have

$$\begin{gathered} (1-2)(\tfrac{7-1}{2}) \equiv s_2(2) \ (mod\ 7) \\ (-1)(3) \equiv 2s_2 \ (mod\ 7) \\ -3 \equiv 2s_2 \ (mod\ 7) \\ 4 \equiv 2s_2 \ (mod\ 7) \end{gathered}$$

and so, $s_2 = 2$. Then, the position of each vertex will move up 2 places in π_2. The vertex 0 is in the 0 position, so it will shift to the $(0-2)$ position, modulo p, and thus 0 will now be in the 5 position in π_2. So, after the shift, we have $\pi_2 : 4 < 6 < 1 < 3 < 5 < 0 < 2$.

Similarly, for π_3 we have

$$(1-4)(3) \equiv s_3(4) \ (mod\ 7)$$

$$-9 \equiv 4s_3 \ (mod\ 7)$$
$$5 \equiv 4s_3 \ (mod\ 7)$$
$$12 \equiv 4s_3 \ (mod\ 7)$$

and so, $s_3 = 3$. Then we have $\pi_3 : 5 < 2 < 6 < 3 < 0 < 4 < 1$. Notice, we now have the set of three linear orderings given for T_3 in Figure 2.5.

This algorithm for arranging the linear orderings of quadratic residue tournaments has been shown to work for odd primes p, such that $p \equiv 3 \ (mod\ 4)$ and $(p-1)/2 \equiv 3 \ (mod\ 4)$ up to $p = 607$, by computer. The computer program used to show this is discussed in Section 3.4.

The structure of the linear orderings for quadratic residue tournaments with odd prime p, such that $p \equiv (p-1)/2 \equiv 3 \ (mod\ 4)$, is given in Figure 3.1. The structure is shown in two different ways. The first is in *wrap-around form*, where we start in the center with $(p-1)/2$, and add q_j successively to obtain each vertex, moving downward in the linear orderings until reaching the last vertex in each linear ordering, and then continuing at the top with the first vertex in each linear ordering and moving downward until all vertices are included. The second is in *symmetric form*, again staring in the center with $(p-1)/2$, and then adding q_j successively to obtain each vertex below $(p-1)/2$ in the linear orderings, and subtracting q_j successively to obtain each vertex above $(p-1)/2$ in the linear orderings.

Let $t \cdot q_j$ be the amount added to $(p-1)/2$ to produce the linear orderings shown in Figure 3.1, where t is the position value in the linear orderings relative to $(p-1)/2$. Where we have $(p-1)/2 - t \cdot q_j$, t is a negative number. By Theorem 3.1(d), for an odd prime p, a quadratic residue of p multiplied by another quadratic residue of p results in a quadratic residue of p. Similarly, a quadratic residue of p multiplied by a quadratic non-residue of p results in a quadratic non-residue of p. We know that q_j is a quadratic residue of p, for $j \in [1, (p-1)/2]$. So, if t is a quadratic residue of p, then $t \cdot q_j$ is a quadratic residue of p. Take a vertex v in the vertex set V of QRT_p, with v represented as $(p-1)/2 + t \cdot q_j$ in the linear orderings, such that $t \cdot q_j$ is a quadratic residue of p. Then v is dominated by the vertex $(p-1)/2$ in QRT_p, since a vertex $u \in V$ dominates another vertex $v \in V$ if and only if $u + q_j \equiv v \ (mod\ p)$. Similarly, if $t \cdot q_j$ is a quadratic non-residue of p, then the vertex v dominates the vertex $(p-1)/2$ in QRT_p.

Given a k-majority tournament $T \sim \mathscr{L}$, each position value, $l_{i_j} \in [0, n-1]$,

Wrap-around Form

$\boldsymbol{\pi_1}$	$\boldsymbol{\pi_2}$		$\boldsymbol{\pi_{\frac{p-1}{2}}}$
$\frac{p-1}{2}+(\frac{p-1}{2}+1)q_1$	$\frac{p-1}{2}+(\frac{p-1}{2}+1)q_2$		$\frac{p-1}{2}+(\frac{p-1}{2}+1)q_{\frac{p-1}{2}}$
$\frac{p-1}{2}+(\frac{p-1}{2}+2)q_1$	$\frac{p-1}{2}+(\frac{p-1}{2}+2)q_2$		$\frac{p-1}{2}+(\frac{p-1}{2}+2)q_{\frac{p-1}{2}}$
$\vdots$	$\vdots$		$\vdots$
$\frac{p-1}{2}+(p-1)q_1$	$\frac{p-1}{2}+(p-1)q_2$		$\frac{p-1}{2}+(p-1)q_{\frac{p-1}{2}}$
$\frac{p-1}{2}$	$\frac{p-1}{2}$	$\cdots$	$\frac{p-1}{2}$
$\frac{p-1}{2}+q_1$	$\frac{p-1}{2}+q_2$		$\frac{p-1}{2}+q_{\frac{p-1}{2}}$
$\frac{p-1}{2}+2q_1$	$\frac{p-1}{2}+2q_2$		$\frac{p-1}{2}+2q_{\frac{p-1}{2}}$
$\vdots$	$\vdots$		$\vdots$
$\frac{p-1}{2}+(\frac{p-1}{2})q_1$	$\frac{p-1}{2}+(\frac{p-1}{2})q_2$		$\frac{p-1}{2}+(\frac{p-1}{2})q_{\frac{p-1}{2}}$

OR

Symmetric Form

$\boldsymbol{\pi_1}$	$\boldsymbol{\pi_2}$		$\boldsymbol{\pi_{\frac{p-1}{2}}}$
$\frac{p-1}{2}-(\frac{p-1}{2})q_1$	$\frac{p-1}{2}-(\frac{p-1}{2})q_2$		$\frac{p-1}{2}-(\frac{p-1}{2})q_{\frac{p-1}{2}}$
$\vdots$	$\vdots$		$\vdots$
$\frac{p-1}{2}-2q_1$	$\frac{p-1}{2}-2q_2$		$\frac{p-1}{2}-2q_{\frac{p-1}{2}}$
$\frac{p-1}{2}-q_1$	$\frac{p-1}{2}-q_2$		$\frac{p-1}{2}-q_{\frac{p-1}{2}}$
$\frac{p-1}{2}$	$\frac{p-1}{2}$	$\cdots$	$\frac{p-1}{2}$
$\frac{p-1}{2}+q_1$	$\frac{p-1}{2}+q_2$		$\frac{p-1}{2}+q_{\frac{p-1}{2}}$
$\frac{p-1}{2}+2q_1$	$\frac{p-1}{2}+2q_2$		$\frac{p-1}{2}+2q_{\frac{p-1}{2}}$
$\vdots$	$\vdots$		$\vdots$
$\frac{p-1}{2}+(\frac{p-1}{2})q_1$	$\frac{p-1}{2}+(\frac{p-1}{2})q_2$		$\frac{p-1}{2}+(\frac{p-1}{2})q_{\frac{p-1}{2}}$

Figure 3.1: Structure of linear orderings using algorithm

corresponds to a *row* within the linear orderings of T. Thus, there is a set of $2k-1$ vertices, which are not always distinct, contained in each row.

In a quadratic residue tournament, QRT_p, such that $p \equiv (p-1)/2 \equiv 3 \ (mod\ 4)$, that is realized by a set $\mathscr{L}$ of linear orderings found using our algorithm, each row corresponds to a value of t, as shown in Figure 3.1. For these tournaments, the vertices in each row are distinct, otherwise we would have $\frac{p-1}{2} + t \cdot q_i \equiv \frac{p-1}{2} + t \cdot q_j \ (mod\ p)$, for $q_i \not\equiv q_j \ (mod\ p)$, and this is not possible. Each row either corresponds to a t that is a quadratic residue of p or one that is a quadratic non residue of p. If the t for a row is a quadratic residue, then that row consists of all vertices that are dominated by the vertex $(p-1)/2$ in QRT_p. Similarly, if the t for a row is a quadratic non residue, then that row consists of all vertices that dominate the vertex $(p-1)/2$ in QRT_p.

It has been shown that for $p \equiv 3 \ (mod\ 4)$, there are more quadratic residues less than $(p-1)/2 \ (mod\ p)$ than there are greater than $(p-1)/2 \ (mod\ p)$ [Dav80]. Then there must be more rows below, than above, the $(p-1)/2$ row in which t is a quadratic residue, and more rows above, than below, the $(p-1)/2$ row in which t is a quadratic non residue. Thus, in the set $\mathscr{L}$ of linear orderings that realize QRT_p, for each $p \equiv (p-1)/2 \equiv 3 \ (mod\ 4)$, the vertex $(p-1)/2$ dominates every vertex $v \in V$ such that $(p-1)/2 + q_j \equiv v \ (mod\ p)$, for $q_j \in Q$, and every other vertex in V dominates the vertex $(p-1)/2$.

For any vertex v_i in V of QRT_p, we know that the position of v_i in a linear ordering π_j in $\mathscr{L}$ of QRT_p is $l_{i_j} \equiv \frac{v_i}{q_j} - s_j \ (mod\ p)$. Also, we know that $v_i \rightarrow v_h$ if and only if $v_h \equiv v_i + q_j \ (mod\ p)$. Then the position of the vertex $v_h \equiv v_i + q_j \ (mod\ p)$ in the linear ordering π_j is

$$\begin{aligned} l_{h_j} &\equiv \tfrac{v_h}{q_j} - s_j \ (mod\ p) \\ &\equiv \tfrac{v_i+q_j}{q_j} - s_j \ (mod\ p) \\ &\equiv \tfrac{v_i}{q_j} + \tfrac{q_j}{q_j} - s_j \ (mod\ p) \\ &\equiv \tfrac{v_i}{q_j} + 1 - s_j \ (mod\ p). \end{aligned}$$

So, $l_{h_j} \equiv l_{i_j} + 1 \ (mod\ p)$. Thus, v_i lies above v_h in π_j unless $l_{i_j} \equiv p-1 \ (mod\ p)$.

In any linear ordering π_m, such that $m \neq j$, $l_{i_m} \equiv \frac{v_i}{q_m} - s_m \ (mod\ p)$ and then

$$\begin{aligned} l_{h_m} &\equiv \tfrac{v_h}{q_m} - s_m \ (mod\ p) \\ &\equiv \tfrac{v_i+q_j}{q_m} - s_m \ (mod\ p) \end{aligned}$$

$$\equiv \frac{v_i}{q_m} + \frac{q_j}{q_m} - s_m \ (mod\ p)$$
$$\equiv \frac{v_i}{q_m} - s_m + \frac{q_j}{q_m} \ (mod\ p).$$

So, $l_{h_m} \equiv l_{i_m} + \frac{q_j}{q_m} \ (mod\ p)$. Now we must show that the value $\frac{q_j}{q_m} \ (mod\ p)$ is an integer, since l_{h_m} must be an integer.

Theorem 3.9. *The ratio of any two quadratic residues is an integer.*

Proof. Let p be an odd prime, and let Q be the set of quadratic residues of p. Then $Q \subset \mathbb{Z}_p*$, where $\mathbb{Z}_p*$ is the set of integers modulo p excluding 0. Note that $\mathbb{Z}_p*$ is a group under multiplication.

Let $q_i, q_j \in Q$. Then, by definition, $q_i \equiv a^2 \ (mod\ p)$ and $q_j \equiv b^2 \ (mod\ p)$, for some $a, b \in \mathbb{Z}_p*$. Now consider the ratio $\frac{q_i}{q_j}$:

$$\frac{q_i}{q_j} \equiv \frac{a^2}{b^2} \ (mod\ p)$$
$$\equiv (\frac{a}{b})^2 \ (mod\ p)$$
$$\equiv (ab^{-1})^2 \ (mod\ p).$$

Since $\mathbb{Z}_p*$ is a group, $ab^{-1} \in \mathbb{Z}_p*$ and then $(ab^{-1})^2 \in \mathbb{Z}_p*$. Therefore, $\frac{q_i}{q_j} \in \mathbb{Z}_p*$, and thus the ratio of any two quadratic residues is an integer. □

We have shown that $\frac{q_j}{q_m} \ (mod\ p)$ is an integer. Further, from the proof of Theorem 3.9, we can conclude that the ratio of any two quadratic residues is a quadratic residue.

Theorem 3.10. *The ratio of any two quadratic residues is a quadratic residue.*

Proof. Let p be an odd prime, and let Q be the set of quadratic residues of p. Let $q_i, q_j \in Q$ and $a, b \in \mathbb{Z}_p*$. From the proof of Theorem 3.9, we know that $(ab^{-1}) \in \mathbb{Z}_p*$ and $\frac{q_i}{q_j} \equiv (ab^{-1})^2 \ (mod\ p)$. Then, by definition, $\frac{q_i}{q_j}$ is a quadratic residue of p. □

So, for vertices v_i and v_h in V of QRT_p, where $v_i + q_j \equiv v_h \ (modp)$, the value of $l_{h_m} - l_{i_m}$, in any linear ordering π_m, is congruent to a residue modulo p. Let $\frac{q_j}{q_m} \equiv q_r \ (mod\ p)$ and $l_{i_m} \equiv l \ (mod\ p)$, where $q_r, l \in \mathbb{Z}_p$. We can conclude that if $(p-1) - l \geq q_r$ for a linear ordering π_m, where $(p-1)$ is the position of the last vertex in π_m, then v_i is above v_h in π_m. Similarly, if $(p-1) - l < q_r$ for π_m, then v_h is above v_i in π_m. Therefore, to prove that our algorithm creates linear orderings that realize QRT_p, for $p \equiv (p-1)/2 \equiv 3 \ (mod\ 4)$, we must show that $(p-1) - l \geq q_r$ for a majority of the linear orderings.

3.4 Computer Programs

There are two different computer programs, with slightly different purposes, that can be used to analyze quadratic residue tournaments. Both of these programs are written in Python.

The first computer program prompts the user to input a prime congruent to 3 modulo 4, and this is the only input required for the program. The program then finds and prints the set of quadratic residues of that prime p. Next, the program uses the set of quadratic residues of p to determine what each vertex dominates in the tournament QRT_p, and prints this information. Then, the program uses the set of quadratic residues of p to find linear orderings that are based on those quadratic residues, and prints these linear orderings. These are the linear orderings before the shift in the algorithm, so they do not actually realize the tournament. The program then performs the shift on those linear orderings and prints the resulting linear orderings that do realize the tournament, if $(p-1)/2 \equiv 3 \ (mod\ 4)$. Then, the program analyzes these linear orderings to find what each vertex dominates, and compares this to what each vertex should dominate in QRT_p. The program queries if the linear orderings realize QRT_p, and prints "yes" if there is a match between what each vertex dominates in the linear orderings and what they should dominate in QRT_p, and prints "no" if there is not a match. The following is the output of the program when we input the prime 23.

```
Prime congruent to 3 modulo 4: 23
[1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18]
What each vertex dominates:
0 -> [0, 1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18]
1 -> [1, 2, 3, 4, 5, 7, 9, 10, 13, 14, 17, 19]
2 -> [2, 3, 4, 5, 6, 8, 10, 11, 14, 15, 18, 20]
3 -> [3, 4, 5, 6, 7, 9, 11, 12, 15, 16, 19, 21]
4 -> [4, 5, 6, 7, 8, 10, 12, 13, 16, 17, 20, 22]
5 -> [0, 5, 6, 7, 8, 9, 11, 13, 14, 17, 18, 21]
6 -> [1, 6, 7, 8, 9, 10, 12, 14, 15, 18, 19, 22]
7 -> [0, 2, 7, 8, 9, 10, 11, 13, 15, 16, 19, 20]
```

```
8 -> [1, 3, 8, 9, 10, 11, 12, 14, 16, 17, 20, 21]
9 -> [2, 4, 9, 10, 11, 12, 13, 15, 17, 18, 21, 22]
10 -> [0, 3, 5, 10, 11, 12, 13, 14, 16, 18, 19, 22]
11 -> [0, 1, 4, 6, 11, 12, 13, 14, 15, 17, 19, 20]
12 -> [1, 2, 5, 7, 12, 13, 14, 15, 16, 18, 20, 21]
13 -> [2, 3, 6, 8, 13, 14, 15, 16, 17, 19, 21, 22]
14 -> [0, 3, 4, 7, 9, 14, 15, 16, 17, 18, 20, 22]
15 -> [0, 1, 4, 5, 8, 10, 15, 16, 17, 18, 19, 21]
16 -> [1, 2, 5, 6, 9, 11, 16, 17, 18, 19, 20, 22]
17 -> [0, 2, 3, 6, 7, 10, 12, 17, 18, 19, 20, 21]
18 -> [1, 3, 4, 7, 8, 11, 13, 18, 19, 20, 21, 22]
19 -> [0, 2, 4, 5, 8, 9, 12, 14, 19, 20, 21, 22]
20 -> [0, 1, 3, 5, 6, 9, 10, 13, 15, 20, 21, 22]
21 -> [0, 1, 2, 4, 6, 7, 10, 11, 14, 16, 21, 22]
22 -> [0, 1, 2, 3, 5, 7, 8, 11, 12, 15, 17, 22]
Linear orderings based on quadratic residues:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
 16, 17, 18, 19, 20, 21, 22]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 1, 3, 5, 7,
 9, 11, 13, 15, 17, 19, 21]
[0, 3, 6, 9, 12, 15, 18, 21, 1, 4, 7, 10, 13, 16, 19, 22,
 2, 5, 8, 11, 14, 17, 20]
[0, 4, 8, 12, 16, 20, 1, 5, 9, 13, 17, 21, 2, 6, 10, 14,
 18, 22, 3, 7, 11, 15, 19]
[0, 6, 12, 18, 1, 7, 13, 19, 2, 8, 14, 20, 3, 9, 15, 21,
 4, 10, 16, 22, 5, 11, 17]
[0, 8, 16, 1, 9, 17, 2, 10, 18, 3, 11, 19, 4, 12, 20, 5,
 13, 21, 6, 14, 22, 7, 15]
[0, 9, 18, 4, 13, 22, 8, 17, 3, 12, 21, 7, 16, 2, 11, 20,
6, 15, 1, 10, 19, 5, 14]
[0, 12, 1, 13, 2, 14, 3, 15, 4, 16, 5, 17, 6, 18, 7, 19,
8, 20, 9, 21, 10, 22, 11]
```

```
[0, 13, 3, 16, 6, 19, 9, 22, 12, 2, 15, 5, 18, 8, 21, 11,
 1, 14, 4, 17, 7, 20, 10]
[0, 16, 9, 2, 18, 11, 4, 20, 13, 6, 22, 15, 8, 1, 17, 10,
 3, 19, 12, 5, 21, 14, 7]
[0, 18, 13, 8, 3, 21, 16, 11, 6, 1, 19, 14, 9, 4, 22, 17,
 12, 7, 2, 20, 15, 10, 5]
Linear orderings that realize QRT 23 :
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22]
[12, 14, 16, 18, 20, 22, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19,
 21, 0, 2, 4, 6, 8, 10]
[1, 4, 7, 10, 13, 16, 19, 22, 2, 5, 8, 11, 14, 17, 20, 0,
 3, 6, 9, 12, 15, 18, 21]
[13, 17, 21, 2, 6, 10, 14, 18, 22, 3, 7, 11, 15, 19, 0, 4,
 8, 12, 16, 20, 1, 5, 9]
[14, 20, 3, 9, 15, 21, 4, 10, 16, 22, 5, 11, 17, 0, 6, 12,
 18, 1, 7, 13, 19, 2, 8]
[15, 0, 8, 16, 1, 9, 17, 2, 10, 18, 3, 11, 19, 4, 12, 20,
 5, 13, 21, 6, 14, 22, 7]
[4, 13, 22, 8, 17, 3, 12, 21, 7, 16, 2, 11, 20, 6, 15, 1,
 10, 19, 5, 14, 0, 9, 18]
[17, 6, 18, 7, 19, 8, 20, 9, 21, 10, 22, 11, 0, 12, 1, 13,
 2, 14, 3, 15, 4, 16, 5]
[6, 19, 9, 22, 12, 2, 15, 5, 18, 8, 21, 11, 1, 14, 4, 17,
7, 20, 10, 0, 13, 3, 16]
[19, 12, 5, 21, 14, 7, 0, 16, 9, 2, 18, 11, 4, 20, 13, 6,
 22, 15, 8, 1, 17, 10, 3]
[20, 15, 10, 5, 0, 18, 13, 8, 3, 21, 16, 11, 6, 1, 19, 14,
 9, 4, 22, 17, 12, 7, 2]
Do these linear orderings realize QRT 23 :
Yes
```

The code for this program is given in Appendix *A*.3. This program is extremely

useful for primes $p \equiv 3 \ (mod\ 4)$, if $(p-1)/2 \equiv 3 \ (mod\ 4)$, as it finds the linear orderings that realize the tournament QRT_p and confirms that those linear orderings actually realize the tournament. The program does not produce linear orderings that realize QRT_p for $(p-1)/2 \equiv 1 \ (mod\ 4)$, but it does provide the set of quadratic residues of p, and a set of linear orderings that can then be used to try to realize QRT_p. A possible method for finding linear orderings to realized QRT_p with $(p-1)/2 \equiv 1 \ (mod\ 4)$ will be discussed in Section 3.5.

The second computer program is best to use only for primes p such that $p \equiv (p-1)/2 \equiv 3 \ (mod\ 4)$, and after confirming, using the first program, that the algorithm from Section 3.3 produces linear orderings that realize the tournament QRT_p. This program is actually like a hybrid of the previous program, and the program from Section 2.6 that analyzes the linear orderings of a k-majority tournament. This program prompts the user to input a prime congruent to 3 modulo 4, and, as in the first program in this section, this is the only input needed. The program then finds and prints the set of quadratic residues of p and what each vertex dominates in the tournament QRT_p based on those quadratic residues. The program then finds and prints the linear orderings based on the quadratic residues before and after the shift, from the algorithm in Section 3.3. Next, the program finds and prints the adjacency matrix for QRT_p and uses this to find and print the minimum dominating sets for QRT_p. The following is the output of the program for the prime 7.

```
Prime congruent to 3 modulo 4: 7
[1, 2, 4]
What each vertex dominaintes:
0 -> [0, 1, 2, 4]
1 -> [1, 2, 3, 5]
2 -> [2, 3, 4, 6]
3 -> [0, 3, 4, 5]
4 -> [1, 4, 5, 6]
5 -> [0, 2, 5, 6]
6 -> [0, 1, 3, 6]
Linear orderings based on quadratic residues:
```

```
[0, 1, 2, 3, 4, 5, 6]
[0, 2, 4, 6, 1, 3, 5]
[0, 4, 1, 5, 2, 6, 3]
Linear orderings that realize QRT 7 :
[0, 1, 2, 3, 4, 5, 6]
[4, 6, 1, 3, 5, 0, 2]
[5, 2, 6, 3, 0, 4, 1]
Adjacency Matrix:
[0, 1, 1, 0, 1, 0, 0]
[0, 0, 1, 1, 0, 1, 0]
[0, 0, 0, 1, 1, 0, 1]
[1, 0, 0, 0, 1, 1, 0]
[0, 1, 0, 0, 0, 1, 1]
[1, 0, 1, 0, 0, 0, 1]
[1, 1, 0, 1, 0, 0, 0]
Dominating Sets:
[]
[0, 1, 2]
[0, 1, 4]
[0, 1, 5]
[0, 1, 6]
[0, 2, 3]
[0, 2, 4]
[0, 2, 5]
[0, 3, 4]
[0, 3, 5]
[0, 3, 6]
[0, 4, 6]
[0, 5, 6]
[1, 2, 3]
[1, 2, 5]
[1, 2, 6]
```

```
[1, 3, 4]
[1, 3, 5]
[1, 3, 6]
[1, 4, 5]
[1, 4, 6]
[2, 3, 4]
[2, 3, 6]
[2, 4, 5]
[2, 4, 6]
[2, 5, 6]
[3, 4, 5]
[3, 5, 6]
[4, 5, 6]
```

The code for this program is given in Appendix *A*.4.

3.5 Realizing Quadratic Residue Tournaments with $(p-1)/2 \equiv 1 \ (mod\ 4)$

Unlike the quadratic residue tournaments QRT_p for $(p-1)/2 \equiv 3 \ (mod\ 4)$, there is, at this time, no algorithm to realize QRT_p for $(p-1)/2 \equiv 1 \ (mod\ 4)$. However, it can be shown that the quadratic residue tournaments QRT_{11} and QRT_{19} can be realized by $(p-1)/2$ linear orderings of their respective vertex sets. To do this, we will first determine the set of quadratic residues of p, and then find linear orderings as we did in the algorithm from Section 3.3. We can use the first computer program from Section 3.4 to accomplish this.

```
Prime congruent to 3 modulo 4: 11
[1, 3, 4, 5, 9]
What each vertex dominates:
0 -> [0, 1, 3, 4, 5, 9]
1 -> [1, 2, 4, 5, 6, 10]
2 -> [0, 2, 3, 5, 6, 7]
```

```
3 -> [1, 3, 4, 6, 7, 8]
4 -> [2, 4, 5, 7, 8, 9]
5 -> [3, 5, 6, 8, 9, 10]
6 -> [0, 4, 6, 7, 9, 10]
7 -> [0, 1, 5, 7, 8, 10]
8 -> [0, 1, 2, 6, 8, 9]
9 -> [1, 2, 3, 7, 9, 10]
10 -> [0, 2, 3, 4, 8, 10]
Linear orderings based on quadratic residues:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8]
[0, 4, 8, 1, 5, 9, 2, 6, 10, 3, 7]
[0, 5, 10, 4, 9, 3, 8, 2, 7, 1, 6]
[0, 9, 7, 5, 3, 1, 10, 8, 6, 4, 2]
Linear orderings that realize QRT 11 :
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 4, 7, 10, 2, 5, 8, 0, 3, 6, 9]
[7, 0, 4, 8, 1, 5, 9, 2, 6, 10, 3]
[2, 7, 1, 6, 0, 5, 10, 4, 9, 3, 8]
[4, 2, 0, 9, 7, 5, 3, 1, 10, 8, 6]
Do these linear orderings realize QRT 11 :
No


Prime congruent to 3 modulo 4: 19
[1, 4, 5, 6, 7, 9, 11, 16, 17]
What each vertex dominates:
0 -> [0, 1, 4, 5, 6, 7, 9, 11, 16, 17]
1 -> [1, 2, 5, 6, 7, 8, 10, 12, 17, 18]
2 -> [0, 2, 3, 6, 7, 8, 9, 11, 13, 18]
3 -> [0, 1, 3, 4, 7, 8, 9, 10, 12, 14]
4 -> [1, 2, 4, 5, 8, 9, 10, 11, 13, 15]
5 -> [2, 3, 5, 6, 9, 10, 11, 12, 14, 16]
```

```
6 -> [3, 4, 6, 7, 10, 11, 12, 13, 15, 17]
7 -> [4, 5, 7, 8, 11, 12, 13, 14, 16, 18]
8 -> [0, 5, 6, 8, 9, 12, 13, 14, 15, 17]
9 -> [1, 6, 7, 9, 10, 13, 14, 15, 16, 18]
10 -> [0, 2, 7, 8, 10, 11, 14, 15, 16, 17]
11 -> [1, 3, 8, 9, 11, 12, 15, 16, 17, 18]
12 -> [0, 2, 4, 9, 10, 12, 13, 16, 17, 18]
13 -> [0, 1, 3, 5, 10, 11, 13, 14, 17, 18]
14 -> [0, 1, 2, 4, 6, 11, 12, 14, 15, 18]
15 -> [0, 1, 2, 3, 5, 7, 12, 13, 15, 16]
16 -> [1, 2, 3, 4, 6, 8, 13, 14, 16, 17]
17 -> [2, 3, 4, 5, 7, 9, 14, 15, 17, 18]
18 -> [0, 3, 4, 5, 6, 8, 10, 15, 16, 18]
Linear orderings based on quadratic residues:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
 15, 16, 17, 18]
[0, 4, 8, 12, 16, 1, 5, 9, 13, 17, 2, 6, 10, 14, 18,
 3, 7, 11, 15]
[0, 5, 10, 15, 1, 6, 11, 16, 2, 7, 12, 17, 3, 8, 13,
 18, 4, 9, 14]
[0, 6, 12, 18, 5, 11, 17, 4, 10, 16, 3, 9, 15, 2, 8,
 14, 1, 7, 13]
[0, 7, 14, 2, 9, 16, 4, 11, 18, 6, 13, 1, 8, 15, 3,
 10, 17, 5, 12]
[0, 9, 18, 8, 17, 7, 16, 6, 15, 5, 14, 4, 13, 3, 12,
 2, 11, 1, 10]
[0, 11, 3, 14, 6, 17, 9, 1, 12, 4, 15, 7, 18, 10, 2,
 13, 5, 16, 8]
[0, 16, 13, 10, 7, 4, 1, 17, 14, 11, 8, 5, 2, 18, 15,
 12, 9, 6, 3]
[0, 17, 15, 13, 11, 9, 7, 5, 3, 1, 18, 16, 14, 12, 10,
 8, 6, 4, 2]
```

```
Linear orderings that realize QRT 19 :
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
 16, 17, 18]
[11, 15, 0, 4, 8, 12, 16, 1, 5, 9, 13, 17, 2, 6, 10,
 14, 18, 3, 7]
[2, 7, 12, 17, 3, 8, 13, 18, 4, 9, 14, 0, 5, 10, 15,
 1, 6, 11, 16]
[12, 18, 5, 11, 17, 4, 10, 16, 3, 9, 15, 2, 8, 14, 1,
 7, 13, 0, 6]
[3, 10, 17, 5, 12, 0, 7, 14, 2, 9, 16, 4, 11, 18, 6,
 13, 1, 8, 15]
[4, 13, 3, 12, 2, 11, 1, 10, 0, 9, 18, 8, 17, 7, 16,
 6, 15, 5, 14]
[5, 16, 8, 0, 11, 3, 14, 6, 17, 9, 1, 12, 4, 15, 7,
 18, 10, 2, 13]
[17, 14, 11, 8, 5, 2, 18, 15, 12, 9, 6, 3, 0, 16,
13, 10, 7, 4, 1]
[8, 6, 4, 2, 0, 17, 15, 13, 11, 9, 7, 5, 3, 1, 18,
16, 14, 12, 10]
Do these linear orderings realize QRT 19 :
No
```

Notice, how the program indicates, for both $p = 11$ and $p = 19$, that the linear orderings found using the algorithm do not realize their respective quadratic residue tournament. The program does, however, give us a lot of useful information about the tournaments QRT_{11} and QRT_{19}.

First, we will look at QRT_{11}. The program gave us the set of quadratic residues of 11, the list of what each vertex dominates in QRT_{11}, and a set of $(p-1)/2$ linear orderings. These linear orderings do not realize QRT_{11}, but we can use them to develop a set of linear orderings that do. We will consider the rows within the given set of linear orderings to be fixed, and move them around, as if the set of linear orderings were a matrix. The original linear orderings given by the program, labeled by row, and the altered linear orderings, which actually realize QRT_{11}, are given in Figure 3.2.

Linear orderings given by algorithm

	π_1	π_2	π_3	π_4	π_5
(0)	0	1	7	2	4
(1)	1	4	0	7	2
(2)	2	7	4	1	0
(3)	3	10	8	6	9
(4)	4	2	1	0	7
(5)	5	5	5	5	5
(6)	6	8	9	10	3
(7)	7	0	2	4	1
(8)	8	3	6	9	10
(9)	9	6	10	3	8
(10)	10	9	3	8	6

Linear orderings that realize QRT_{11}

	π_1	π_2	π_3	π_4	π_5
(3)	3	10	8	6	9
(6)	6	8	9	10	3
(4)	4	2	1	0	7
(2)	2	7	4	1	0
(7)	7	0	2	4	1
(5)	5	5	5	5	5
(8)	8	3	6	9	10
(0)	0	1	7	2	4
(9)	9	6	10	3	8
(1)	1	4	0	7	2
(10)	10	9	3	8	6

Figure 3.2: Linear orderings for QRT_{11}

π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9
15	14	1	7	13	6	18	10	16
1	15	7	18	10	13	16	14	6
2	0	12	5	17	3	8	11	4
6	16	13	10	7	1	14	18	15
3	4	17	11	5	12	0	8	2
10	13	14	15	16	18	1	6	7
0	11	2	12	3	4	5	17	8
4	8	3	17	12	2	11	5	0
5	12	8	4	0	11	3	2	17
9	9	9	9	9	9	9	9	9
7	1	18	16	14	10	6	15	13
11	17	0	2	4	8	12	3	5
8	5	4	3	2	0	17	12	11
14	10	15	1	6	16	7	13	18
12	2	5	8	11	17	4	0	3
16	18	6	13	1	15	10	7	14
13	6	10	14	18	7	15	16	1
17	3	11	0	8	5	2	4	12
18	7	16	6	15	14	13	1	10

Figure 3.3: Linear orderings that realize QRT_{19}

The linear orderings that realize QRT_{19} can be obtained using the same method. The linear orderings that actually realize QRT_{19} are given in Figure 3.3.

As stated before, there is no real algorithm for finding linear orderings that realize QRT_p for $(p-1)/2 \equiv 1 \ (mod\ 4)$, it is more of a guess and check method. However, it is possible to realize both QRT_{11} and QRT_{19} as k-majority tournaments, and it may be possible for other values of p as well. Also, it may be possible to produce a more polished method for realizing these quadratic residue tournaments.

Chapter 4

Conclusion

Realizing a k-majority tournament, with $2k-1$ linear orderings of the vertices in the vertex set of the tournament, can be difficult. However, there are certain types of tournaments for which we can more easily find linear orderings to realize them as k-majority tournaments. Any acyclic tournament T, with vertex set V, can be realized as a 2-majority tournament such that any two of the linear orderings of V are represented by the Hamiltonian path in T, and the third linear ordering can be any configuration of the vertices in V. Further, for acyclic tournaments, we know that the dominating set, Γ, consists of a single vertex.

For quadratic residue tournaments, QRT_p, where p is an odd prime such that $p \equiv (p-1)/2 \equiv 3 \ (mod\ 4)$, we have an algorithm for finding linear orderings that realize QRT_p and a computer program that can use the algorithm and verify that the linear orderings created actually realize the tournament. This is particularly important, since k-majority tournaments do not have arbitrarily large minimum dominating sets, they are bounded above such that $F(k) \leq (80 + o(1))k \log k$ [ABK$^+$06]. Thus, we can conclude that a subset of quadratic residue tournaments have minimum dominating sets that are bounded above based on the value of p, since $k = (p+1)/4$.

Also, for any 2-majority tournament T, that can be realized by the set of linear orderings $\mathscr{L}_1 = \{\pi_1, \pi_2, \pi_3\}$, T can be realized as a pseudo-3-majority tournament with the set of linear orderings $\mathscr{L}_2 = \{\pi_1, \pi_2, \pi_3, \pi_i, \pi_j\}$, for $i, j \in [3]$ and $i \neq j$. If we had $i = j$, in this situation, $\mathscr{L}_2$ would not realize T, but would actually represent an acyclic tournament, since k of the linear orderings would be identical.

It is rather simple to find an example of a pseudo-3-majority tournament, but very difficult to find an example of a critical 3-majority tournament. Although, using the structure in the linear orderings of 2-majority tournaments, as defined by Noga Alon et al. in Theorem 2.9, may lend a possible method for finding critical 3-majority tournaments.

Inherited 2-majority tournaments may also give some insight into the structural differences and relationship between 2- and 3-majority tournaments. They may allow us to determine when a 3-majority tournament can be realized as a 2-majority tournament. They may even be useful in determining dominating sets for a 3-majority tournament that is realized by a set of five linear orderings. However, more research is needed to develop and prove stronger relationships between 3-majority tournaments and their inherited 2-majority tournaments. Also, this concept may extend to k-majority tournaments for any value of k.

There is also more research needed to prove how the algorithm works for realizing quadratic residue tournaments, QRT_p, with odd primes p such that $p \equiv (p-1)/2 \equiv 3 \ (mod\ 4)$. It may also be possible to develop an algorithm, or more defined method, for realizing quadratic residue tournaments, QRT_p, with odd prime p where $p \equiv 3 \ (mod\ 4)$ and $(p-1)/2 \equiv 1 \ (mod\ 4)$.

Appendix A

Code for Computer Programs

A.1 k-Majority Tournaments

```
order = int(input('Input order of the tournament: '))

start = int(input('Input the starting vertex value: '))

Choice = int(input('Enter "1" for linear orderings or
                    "2" for adjacency matrix: '))

if Choice == 1:
    linearOrders = int(input('Input number of linear
                              orderings: '))
    Orderings = [[]]
    for i in range(linearOrders - 1):
        Orderings.append([])

    for i in range(linearOrders):
        print('Linear ordering', i + 1, ': ')
        for j in range(order):
            Orderings[i].insert(j, int(input(': ')))
```

```
HyperEdge = [[]]
for i in range(order - 1):
    HyperEdge.append([])

List = []

for j in range(order):
    for k in range(order):
        for i in range(linearOrders):
            if Orderings[i].index(j + start)
            <= Orderings[i].index(k + start):
                List.append(j)
            if len(List) >= ((linearOrders + 1) / 2):
                HyperEdge[j].append(k + start)
        del List[0: ]

HyperEdge1 = [[]]
for i in range(order - 1):
    HyperEdge1.append([])

for i in range(order):
    for j in range(order):
        if (j + start) in HyperEdge[i]:
            HyperEdge1[i].append(j + start)

print('What each vertex dominates: ')
for i in range(order):
    print(i + start, '->', HyperEdge1[i])

print('Adjacency Matrix: ')
Matrix = [[]]
```

```
    for i in range(order - 1):
        Matrix.append([])

    for i in range(order):
        for j in range(order):
            if j == i :
                Matrix[i].append(0)
            elif (j + start) in HyperEdge1[i]:
                Matrix[i].append(1)
            else:
                Matrix[i].append(0)

    for i in range(order):
        print(Matrix[i])

    Matrixa = [[]]
    for i in range(order - 1):
        Matrixa.append([])

    for i in range(order):
        for j in range(order):
            if j == i :
                Matrixa[i].append(1)
            elif (j + start) in HyperEdge1[i]:
                Matrixa[i].append(1)
            else:
                Matrixa[i].append(0)

elif Choice == 2:
    Matrix = [[]]
    for i in range(order - 1):
        Matrix.append([])
```

```
    for i in range(order):
        print('row', i + 1, ': ')
        for j in range(order):
            Matrix[i].insert(j, int(input(': ')))


    Matrixa = [[]]
    for i in range(order - 1):
        Matrixa.append([])

    for i in range(order):
        for j in range(order):
            if Matrix[i][j] == 1:
                Matrixa[i].append(1)
            elif i == j:
                Matrixa[i].append(1)
            else:
                Matrixa[i].append(0)

else:
    print('Invalid Entry')

Not = []
DominatingSets = [[]]
for i in range(order):
    if 0 in Matrixa[i]:
        Not.append(i)
    else:
        DominatingSets[0].append(i + start)

if DominatingSets == [[]]:
```

```
        Check = []
        for i in range(order):
            for j in range(i, order):
                for k in range(order):
                    Check.append(Matrixa[i][k] +
                                 Matrixa[j][k])

                if 0 in Check:
                    Not.append(i)
                else:
                    DominatingSets.append([])
                    DominatingSets[-1].append(i + start)
                    DominatingSets[-1].append(j + start)

                del Check[0: ]
                del Not[0: ]

    if DominatingSets == [[]]:
        for i in range(order):
            for j in range(i, order):
                for k in range(j, order):
                    for l in range(order):
                        Check.append(Matrixa[i][l] +
                                     Matrixa[j][l] +
                                     Matrixa[k][l])

                    if 0 in Check:
                        Not.append(i)
                    else:
                        DominatingSets.append([])
                        DominatingSets[-1].append(i + start)
                        DominatingSets[-1].append(j + start)
```

```
                    DominatingSets[-1].append(k + start)

                del Check[0: ]
                del Not[0: ]

if DominatingSets == [[]]:
    for i in range(order):
        for j in range(i, order):
            for k in range(j, order):
                for l in range(k, order):
                    for m in range(order):
                        Check.append(Matrixa[i][m] +
                                     Matrixa[j][m] +
                                     Matrixa[k][m] +
                                     Matrixa[l][m])

                    if 0 in Check:
                        Not.append(i)
                    else:
                        DominatingSets.append([])
                        DominatingSets[-1].append(i + start)
                        DominatingSets[-1].append(j + start)
                        DominatingSets[-1].append(k + start)
                        DominatingSets[-1].append(l + start)

                    del Check[0: ]
                    del Not[0: ]

if DominatingSets == [[]]:
    for i in range(order):
        for j in range(i, order):
            for k in range(j, order):
```

```
                for l in range(k, order):
                    for m in range(l, order):
                        for n in range(order):
                            Check.append(Matrixa[i][n] +
                                         Matrixa[j][n] +
                                         Matrixa[k][n] +
                                         Matrixa[l][n] +
                                         Matrixa[m][n])

                        if 0 in Check:
                            Not.append(i)
                        else:
                            DominatingSets.append([])
                            DominatingSets[-1].append(i + start)
                            DominatingSets[-1].append(j + start)
                            DominatingSets[-1].append(k + start)
                            DominatingSets[-1].append(l + start)
                            DominatingSets[-1].append(m + start)

                        del Check[0: ]
                        del Not[0: ]

print('Dominating Sets: ')
for i in range(len(DominatingSets)):
    if DominatingSets[i] != []:
        print(DominatingSets[i])

print('Triangles: ')
Test = [[], [], []]
Test2 = [[0, 0, 1], [1, 0, 0], [0, 1, 0]]
Test3 = [[0, 1, 0], [0, 0, 1], [1, 0, 0]]
for i in range(order):
```

```
    for j in range(i, order):
        for k in range(j, order):
            Test[0].append(Matrix[i][i])
            Test[0].append(Matrix[i][j])
            Test[0].append(Matrix[i][k])
            Test[1].append(Matrix[j][i])
            Test[1].append(Matrix[j][j])
            Test[1].append(Matrix[j][k])
            Test[2].append(Matrix[k][i])
            Test[2].append(Matrix[k][j])
            Test[2].append(Matrix[k][k])

            if Test == Test2:
                print(i + start, ', ',
                 k + start, ', ', j + start)
            elif Test == Test3:
                print(i + start, ', ',
                j + start, ', ', k + start)
            del Test[0][0: ]
            del Test[1][0: ]
            del Test[2][0: ]
```

A.2 Inherited 2-Majority Tournaments

```
order = int(input('Input order of the tournament: '))

start = int(input('Input the starting vertex value: '))

LO = [[], [], [], [], []]

for i in range(5):
    print('Input linear ordering', i + 1, ': ')
```

```
    for j in range(order):
        LO[i].append(int(input(': ')))

print('Linear Orderings: ')
for i in range(5):
    print(LO[i])

HyperEdge= [[]]
for i in range(order - 1):
    HyperEdge.append([])

Check = []

for i in range(order):
    for j in range(order):
        for k in range(5):
            if LO[k].index(i + start) <= LO[k].index(j + start):
                Check.append(i)
        if len(Check) >= 3:
            HyperEdge[i].append(j + start)

        del Check[0: ]

print('What each vertex dominates
                in the 3-majority tournament: ')
for i in range(order):
    print(i + 1, '->', HyperEdge[i])

print('Adjacency Matrix: ')
Matrix = [[]]
for i in range(order - 1):
    Matrix.append([])
```

```
for i in range(order):
    for j in range(order):
        if j == i :
            Matrix[i].append(0)
        elif (j + start) in HyperEdge[i]:
            Matrix[i].append(1)
        else:
            Matrix[i].append(0)

for i in range(order):
    print(Matrix[i])

Matrixa = [[]]
for i in range(order - 1):
    Matrixa.append([])

for i in range(order):
    for j in range(order):
        if j == i :
            Matrixa[i].append(1)
        elif (j + start) in HyperEdge[i]:
            Matrixa[i].append(1)
        else:
            Matrixa[i].append(0)

Not = []
DominatingSets = [[]]
for i in range(order):
    if 0 in Matrixa[i]:
        Not.append(i)
    else:
```

```
            DominatingSets[0].append(i + start)

    if DominatingSets == [[]]:
        Check = []
        for i in range(order):
            for j in range(i, order):
                for k in range(order):
                    Check.append(Matrixa[i][k] +
                                 Matrixa[j][k])

                if 0 in Check:
                    Not.append(i)
                else:
                    DominatingSets.append([])
                    DominatingSets[-1].append(i + start)
                    DominatingSets[-1].append(j + start)

                del Check[0: ]
                del Not[0: ]

    if DominatingSets == [[]]:
        for i in range(order):
            for j in range(i, order):
                for k in range(j, order):
                    for l in range(order):
                        Check.append(Matrixa[i][l] +
                                     Matrixa[j][l] +
                                     Matrixa[k][l])

                    if 0 in Check:
                        Not.append(i)
                    else:
```

```
                    DominatingSets.append([])
                    DominatingSets[-1].append(i + start)
                    DominatingSets[-1].append(j + start)
                    DominatingSets[-1].append(k + start)

                del Check[0: ]
                del Not[0: ]


print('Dominating Sets: ')
for i in range(len(DominatingSets)):
    if DominatingSets[i] != []:
        print(DominatingSets[i])

print('Triangles: ')
Test = [[], [], []]
Test2 = [[0, 0, 1], [1, 0, 0], [0, 1, 0]]
Test3 = [[0, 1, 0], [0, 0, 1], [1, 0, 0]]
for i in range(order):
    for j in range(i, order):
        for k in range(j, order):
            Test[0].append(Matrix[i][i])
            Test[0].append(Matrix[i][j])
            Test[0].append(Matrix[i][k])
            Test[1].append(Matrix[j][i])
            Test[1].append(Matrix[j][j])
            Test[1].append(Matrix[j][k])
            Test[2].append(Matrix[k][i])
            Test[2].append(Matrix[k][j])
            Test[2].append(Matrix[k][k])

            if Test == Test2:
```

```
                print(i + start, ', ', k +
                                    start, ', ', j + start)
            elif Test == Test3:
                print(i + start, ', ', j +
                                    start, ', ', k + start)
            del Test[0][0: ]
            del Test[1][0: ]
            del Test[2][0: ]

Inherited = [[LO[0], LO[1], LO[2]], [LO[0], LO[1], LO[3]],
             [LO[0], LO[1], LO[4]], [LO[0], LO[2], LO[3]],
             [LO[0], LO[2], LO[4]], [LO[0], LO[3], LO[4]],
             [LO[1], LO[2], LO[3]], [LO[1], LO[2], LO[4]],
             [LO[1], LO[3], LO[4]], [LO[2], LO[3], LO[4]]]

print('Inherited 2-majority tournaments: ')
for i in range(10):
    print('T', i + 1 , ': ', Inherited[i])



HyperEdge1= [[[]]]
for i in range(9):
    HyperEdge1.append([[]])

for i in range(10):
    for j in range(order - 1):
        HyperEdge1[i].append([])

Check = []

for h in range(10):
```

```
    for i in range(order):
        for j in range(order):
            for k in range(3):
                if Inherited[h][k].index(i + start) <=
                        Inherited[h][k].index(j + start):
                    Check.append(i)
            if len(Check) >= 2:
                HyperEdge1[h][i].append(j + start)

            del Check[0: ]


for i in range(10):
    print('What each vertex dominates in T', i + 1, ': ')
    for j in range(order):
        print(j + 1, '->', HyperEdge1[i][j])


Matrix1 = [[[]]]
for i in range(9):
    Matrix1.append([[]])

for i in range(10):
    for j in range(order - 1):
        Matrix1[i].append([])

for h in range(10):
    for i in range(order):
        for j in range(order):
            if j == i :
                Matrix1[h][i].append(0)
            elif (j + start) in HyperEdge1[h][i]:
```

```
                Matrix1[h][i].append(1)
            else:
                Matrix1[h][i].append(0)

Matrixa1 = [[[]]]
for i in range(9):
    Matrixa1.append([[]])

for i in range(10):
    for j in range(order - 1):
        Matrixa1[i].append([])

for h in range(10):
    for i in range(order):
        for j in range(order):
            if j == i :
                Matrixa1[h][i].append(1)
            elif (j + start) in HyperEdge1[h][i]:
                Matrixa1[h][i].append(1)
            else:
                Matrixa1[h][i].append(0)



Not = []
DominatingSets1 = [[[]]]
for i in range(9):
    DominatingSets1.append([[]])

for h in range(10):
    for i in range(order):
        if 0 in Matrixa1[h][i]:
            Not.append(i)
```

```
            else:
                DominatingSets1[h][0].append(i + start)

for h in range(10):
    Check = []
    if DominatingSets1[h] == [[]]:
        for i in range(order):
            for j in range(i, order):
                for k in range(order):
                    Check.append(Matrixa1[h][i][k] +
                                 Matrixa1[h][j][k])

                if 0 in Check:
                    Not.append(i)
                else:
                    DominatingSets1[h].append([])
                    DominatingSets1[h][-1].append(i + start)
                    DominatingSets1[h][-1].append(j + start)

                del Check[0: ]
                del Not[0: ]


for h in range(10):
    if DominatingSets1[h] == [[]]:
        for i in range(order):
            for j in range(i, order):
                for k in range(j, order):
                    for l in range(order):
                        Check.append(Matrixa1[h][i][l] +
                                     Matrixa1[h][j][l] +
                                     Matrixa1[h][k][l])
```

```
                    if 0 in Check:
                        Not.append(i)
                    else:
                        DominatingSets1[h].append([])
                        DominatingSets1[h][-1].append(i + start)
                        DominatingSets1[h][-1].append(j + start)
                        DominatingSets1[h][-1].append(k + start)

                    del Check[0: ]
                    del Not[0: ]


for h in range(10):
    print('Dominating sets for T', h + 1, ': ')
    for i in range(len(DominatingSets1[h])):
        if DominatingSets1[h][i] != []:
            print(DominatingSets1[h][i])
```

A.3 Quadratic Residue Tournaments Check

```
Prime = (int(input("Prime congruent to 3 modulo 4: ")))

Class = []
for i in range(1, Prime):
    Class.append(i**2)

for i in range(len(Class)):
    while Class[i] >= Prime:
        Class[i] = (Class[i] - Prime)

Class.sort()
```

```
Class1 = []
for i in range(Prime):
    if i in Class:
        Class1.append(i)

print(Class1)

print('What each vertex dominates: ')

HyperEdge = []
for i in range(Prime):
    HyperEdge.append([])

for i in range(Prime):
    HyperEdge[i].append(i)
    for j in range(len(Class1)):
        HyperEdge[i].append(i + Class1[j])

for i in range(Prime):
    for j in range(len(HyperEdge[i])):
        while HyperEdge[i][j] >= Prime:
            HyperEdge[i][j] = (HyperEdge[i][j] - Prime)

for i in range(Prime):
    HyperEdge[i].sort()

for i in range(Prime):
    print(i,'->', HyperEdge[i])

print('Linear orderings based on quadratic residues: ')
LinearOrders = [[]]
for i in range(1, len(Class1)):
```

```
    LinearOrders.append([])

for i in range(len(Class1)):
    LinearOrders[i].append(0)
    j = 0
    while j < Prime * Class1[i] - Class1[i]:
        LinearOrders[i].append(j + Class1[i])
        j = j + Class1[i]

for i in range(len(Class1)):
    for j in range(len(LinearOrders[i])):
        while LinearOrders[i][j] >= Prime:
            LinearOrders[i][j] = (LinearOrders[i][j] - Prime)

for i in range(len(Class1)):
    print(LinearOrders[i])

mid = (Prime + 1)/2

change = []

for i in range(1, len(Class1)):
    diff = int(mid - LinearOrders[i].index(mid))
    change.append(diff)

LinearOrders1 = [[]]
for i in range(1, len(Class1)):
    LinearOrders1.append([])

LinearOrders1[0] = LinearOrders[0]

for i in range(1, len(Class1)):
```

```
    j = 0
    while j < Prime * Class1[i]:
        LinearOrders1[i].append(j + Class1[i]*change[i-1])
        j = j + Class1[i]

for i in range(1, len(Class1)):
    for j in range(len(LinearOrders1[i])):
        while LinearOrders1[i][j] >= Prime:
            LinearOrders1[i][j] = (LinearOrders1[i][j] - Prime)

for i in range(1, len(Class1)):
    for j in range(len(LinearOrders1[i])):
        while LinearOrders1[i][j] < 0:
            LinearOrders1[i][j] = (LinearOrders1[i][j] + Prime)

print('Linear orderings that realize QRT', Prime, ': ')
for i in range(len(Class1)):
    print(LinearOrders1[i])

order = Prime

linearOrders = len(Class1)

Orderings = LinearOrders1

HyperEdge2 = [[]]
for i in range(order - 1):
    HyperEdge2.append([])

List = []

for j in range(order):
```

```
    for k in range(order):
        for i in range(linearOrders):
            if Orderings[i].index(j) <= Orderings[i].index(k):
                List.append(j)
            if len(List) >= ((linearOrders + 1) / 2):
                HyperEdge2[j].append(k)
        del List[0: ]

HyperEdge1 = [[]]
for i in range(order - 1):
    HyperEdge1.append([])

for i in range(order):
    for j in range(order):
        if j in HyperEdge2[i]:
            HyperEdge1[i].append(j)

print('Do these linear orderings realize QRT', Prime, ':')

if HyperEdge == HyperEdge1:
    print('Yes')
else:
    print('No')
```

A.4 Quadratic Residue Tournaments with Dominating Sets

```
Prime = (int(input("Prime congruent to 3 modulo 4: ")))

Class = []
for i in range(1, Prime):
    Class.append(i**2)
```

```
for i in range(len(Class)):
    while Class[i] >= Prime:
        Class[i] = (Class[i] - Prime)

Class.sort()
Class1 = []
for i in range(Prime):
    if i in Class:
        Class1.append(i)

print(Class1)

print('What each vertex dominaintes: ')

HyperEdge = []
for i in range(Prime):
    HyperEdge.append([])

for i in range(Prime):
    HyperEdge[i].append(i)
    for j in range(len(Class1)):
        HyperEdge[i].append(i + Class1[j])

for i in range(Prime):
    for j in range(len(HyperEdge[i])):
        while HyperEdge[i][j] >= Prime:
            HyperEdge[i][j] = (HyperEdge[i][j] - Prime)

for i in range(Prime):
    HyperEdge[i].sort()

for i in range(Prime):
```

```
    print(i,'->', HyperEdge[i])

print('Linear orderings based on quadratic residues: ')
LinearOrders = [[]]
for i in range(1, len(Class1)):
    LinearOrders.append([])

for i in range(len(Class1)):
    LinearOrders[i].append(0)
    j = 0
    while j < Prime * Class1[i] - Class1[i]:
        LinearOrders[i].append(j + Class1[i])
        j = j + Class1[i]

for i in range(len(Class1)):
    for j in range(len(LinearOrders[i])):
        while LinearOrders[i][j] >= Prime:
            LinearOrders[i][j] = (LinearOrders[i][j] - Prime)

for i in range(len(Class1)):
    print(LinearOrders[i])

mid = (Prime + 1)/2

change = []

for i in range(1, len(Class1)):
    diff = int(mid - LinearOrders[i].index(mid))
    change.append(diff)

LinearOrders1 = [[]]
for i in range(1, len(Class1)):
```

```
    LinearOrders1.append([])

LinearOrders1[0] = LinearOrders[0]

for i in range(1, len(Class1)):
    j = 0
    while j < Prime * Class1[i]:
        LinearOrders1[i].append(j + Class1[i]*change[i-1])
        j = j + Class1[i]

for i in range(1, len(Class1)):
    for j in range(len(LinearOrders1[i])):
        while LinearOrders1[i][j] >= Prime:
            LinearOrders1[i][j] = (LinearOrders1[i][j] - Prime)

for i in range(1, len(Class1)):
    for j in range(len(LinearOrders1[i])):
        while LinearOrders1[i][j] < 0:
            LinearOrders1[i][j] = (LinearOrders1[i][j] + Prime)

print('Linear orderings that realize QRT', Prime, ': ')
for i in range(len(Class1)):
    print(LinearOrders1[i])

order = Prime

print('Adjacency Matrix: ')
Matrix = [[]]
for i in range(order - 1):
    Matrix.append([])

for i in range(order):
```

```
    for j in range(order):
        if j == i :
            Matrix[i].append(0)
        elif j in HyperEdge[i]:
            Matrix[i].append(1)
        else:
            Matrix[i].append(0)

for i in range(order):
    print(Matrix[i])

Matrixa = [[]]
for i in range(order - 1):
    Matrixa.append([])

for i in range(order):
    for j in range(order):
        if j == i :
            Matrixa[i].append(1)
        elif j in HyperEdge[i]:
            Matrixa[i].append(1)
        else:
            Matrixa[i].append(0)

Not = []
DominatingSets = [[]]
for i in range(order):
    if 0 in Matrixa[i]:
        Not.append(i)
    else:
        DominatingSets[0].append(i)
```

```
if DominatingSets == [[]]:
    Check = []
    for i in range(order):
        for j in range(i, order):
            for k in range(order):
                Check.append(Matrixa[i][k] +
                             Matrixa[j][k])

            if 0 in Check:
                Not.append(i)
            else:
                DominatingSets.append([])
                DominatingSets[-1].append(i)
                DominatingSets[-1].append(j)

            del Check[0: ]
            del Not[0: ]

if DominatingSets == [[]]:
    for i in range(order):
        for j in range(i, order):
            for k in range(j, order):
                for l in range(order):
                    Check.append(Matrixa[i][l] +
                                 Matrixa[j][l] +
                                 Matrixa[k][l])

                if 0 in Check:
                    Not.append(i)
                else:
                    DominatingSets.append([])
                    DominatingSets[-1].append(i)
```

```
                    DominatingSets[-1].append(j)
                    DominatingSets[-1].append(k)

                del Check[0: ]
                del Not[0: ]

if DominatingSets == [[]]:
    for i in range(order):
        for j in range(i, order):
            for k in range(j, order):
                for l in range(k, order):
                    for m in range(order):
                        Check.append(Matrixa[i][m] +
                                     Matrixa[j][m] +
                                     Matrixa[k][m] +
                                     Matrixa[l][m])

                    if 0 in Check:
                        Not.append(i)
                    else:
                        DominatingSets.append([])
                        DominatingSets[-1].append(i)
                        DominatingSets[-1].append(j)
                        DominatingSets[-1].append(k)
                        DominatingSets[-1].append(l)

                    del Check[0: ]
                    del Not[0: ]

if DominatingSets == [[]]:
    for i in range(order):
        for j in range(i, order):
```

```
            for k in range(j, order):
                for l in range(k, order):
                    for m in range(l, order):
                        for n in range(order):
                            Check.append(Matrixa[i][n] +
                                         Matrixa[j][n] +
                                         Matrixa[k][n] +
                                         Matrixa[l][n] +
                                         Matrixa[m][n])

                        if 0 in Check:
                            Not.append(i)
                        else:
                            DominatingSets.append([])
                            DominatingSets[-1].append(i)
                            DominatingSets[-1].append(j)
                            DominatingSets[-1].append(k)
                            DominatingSets[-1].append(l)
                            DominatingSets[-1].append(m)

                        del Check[0:]
                        del Not[0:]

if DominatingSets == [[]]:
    for i in range(order):
        for j in range(i, order):
            for k in range(j, order):
                for l in range(k, order):
                    for m in range(l, order):
                        for n in range(m, order):
                            for o in range(order):
                                Check.append(Matrixa[i][o] +
```

```
                                        Matrixa[j][o] +
                                        Matrixa[k][o] +
                                        Matrixa[l][o] +
                                        Matrixa[m][o] +
                                        Matrixa[n][o])

                                if 0 in Check:
                                    Not.append(i)
                                else:
                                    DominatingSets.append([])
                                    DominatingSets[-1].append(i)
                                    DominatingSets[-1].append(j)
                                    DominatingSets[-1].append(k)
                                    DominatingSets[-1].append(l)
                                    DominatingSets[-1].append(m)
                                    DominatingSets[-1].append(n)

                                del Check[0: ]
                                del Not[0: ]

print('Dominating Sets: ')
for i in range(len(DominatingSets)):
    print(DominatingSets[i])
```

Bibliography

[ABK+06] Noga Alon, Graham Brightwell, H.A. Kierstead, A.V. Kostochka, and Peter Winkler. Dominating sets in k-majority tournaments. *Journal of Combinatorial Theory Series B*, pages 374–387, 2006.

[Bur11] David M. Burton. *Elementary Number Theory.* McGraw-Hill, 7th edition, 2011.

[cL97] Jean-François Laslier. *Tournament Solutions and Majority Voting.* Studies in Economic Theory 7. Springer, 1997.

[CLZ11] Gary Chartrand, Linda Lesniak, and Ping Zhang. *Graphs and Digraphs.* CRC Press, 6th edition, 2011.

[Dav80] Harold Davenport. *Multiplicative Number Theory.* Springer-Verlag, 2nd edition, 1980.

[Erd63] Paul Erdös. On a problem of graph theory. *The Mathematical Gazette*, pages 220–223, 1963.