MATH 3310 – Final Experience

Fall 2017

David E Brown

Matt Bishop
Phil Brannon
Daniel Oliveros
Molly Robinson
Nicole Westfall

✠ Sub-Experience One: The Binary Address Graph.

Define the graph Q_n to be the graph whose vertices are all the binary strings of length n and vertices are adjacent if and only if they differ in exactly one entry of the list. The vertices could be thought of as vectors $\vec{v} = (v_1, v_2, \ldots, v_n)$, where $v_i \in \{0, 1\}$; then $\vec{u}\vec{v} \in E(Q_n)$ if and only if $v_i \neq u_i$ for exactly one i. Please prove the following about Q_n.

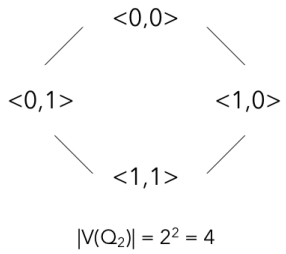
1. $|V(Q_n)| = 2^n$.
2. Q_n is an n-regular graph; that is, $\deg(\vec{v}) = n$ for each $\vec{v} \in V(Q_n)$.
3. Q_n is *bipartite*; that is, $V(Q_n)$ consists of two sets, say X and Y such that $X \cap Y = \emptyset$ and the only edges of Q_n have one end-vertex in X and the other in Y (so X and Y induce graphs with no edges).
4. Q_n is Hamiltonian for $n \geq 2$.

---

(1) We are asked to prove that the total number of adjacent vertices $|V(Q_n)|$, each identified with a binary string of length n for the purpose of determining adjacency, are equal to 2^n supposing that adjacency is defined by two vertices having binary strings differing at exactly one position out of n positions. To model the situation, we will hereafter refer to each vertex in graph Q_n as being identified with a vector possessing n binary components.

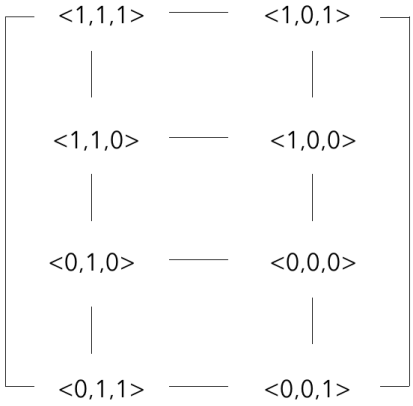
Each position in a set of vectors having n components contains one of two possible values, either 1 or 0. Thus, for each position there is only the possibly of a single adjacency. For example, the set of all possible vectors containing $n = 1$ components results in the vectors $< 0 >$ and $< 1 >$ and 2 vertices are related for every one edge.

fig. 1



$|V(Q_2)| = 2^2 = 4$

fig. 2



$|V(Q_2)| = 2^3 = 8$

This relation holds for all values of n. For $n = 2$, our set of possible vectors is: $\{< 0,0 >, < 1,0 >, < 0,1 >, < 1,1 >\}$ (as shown in fig. 1). Given $n = 3$, we have $\{< 0,0,0 >, < 1,0,0 >, < 0,1,0 >, < 1,1,0 >, < 0,1,1 >, < 0,1,1 >, < 1,1,1 >, < 1,0,1 >\}$ (as shown in fig. 2), so on and so forth. For every additional binary vector component, the total number of vertices grows by a factor of 2. Each individual component of the vector makes possible one adjacency relating 2 of the vectors (vertices) in the set, thus $|V(Q_n)| = 2^n$.

(2) As shown in (1) above, each vertex has a single adjacency for every binary component of the vector associated with it. All vertices therefore have a degree of n, or, $\deg(\vec{v}) = n$ for each $\vec{v} \in V(Q_n)$. Thus Q_n is a n-regular graph.

(3) We are asked to prove that Q_n is bipartite, that is to say that every cycle in Q_n must have an even length in accord with theorem 12.4.1. If every cycle in Q_n is shown to have an even length, then, by theorem 12.4.2, Q_n is bipartite.

We begin by observing that the number of vertices for a Q_n graph is even for any value of n. This is clear when examining the equality $|V(Q_n)| = 2^n$ proven in (1). For every number of n, the resulting number of vertices MUST be even as any integer multiplied by an even number, in this case 2, must be even.

Next, we note that two vertices are connected only if our comparison of their related vectors results in a single component being different; that is to say, in one position, one of the vertices in the pair must have a 0 and the other a 1. If we were to add all of the components of our vectors together we would note that in every case, connections are between vertices possessing identity vectors with odd sums to vertices possessing identity vectors with even sums. Thus, in accord with our definition of bipartite, we can formulate two distinct sets X and Y composed of "even sum" and "odd sum" vectors respectively, with the assurance that connections between vertices identified by vectors exclusively in one set or the other, cannot be connected to members of the same set as shown below.

fig. 3

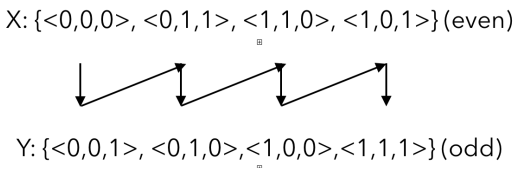
$\|V(Q_2)\| = 2^2 = 4$	$\|V(Q_2)\| = 2^3 = 8$
X: {<0,0>, <1,1>}(even)	X: {<0,0,0>, <0,1,1>, <1,1,0>, <1,0,1>}(even)
Y: {<0,1>, <1,0>}(odd)	Y: {<0,0,1>, <0,1,0>,<1,0,0>,<1,1,1>}(odd)

As a result, the completion of any cycle within Q_n requires an even number of connections to "close the loop", or return to the starting vertex. This demonstrates that we meet the condition set forth by theorem 12.4.2 and thus we have proven that Q_n is bipartite.

(4) We are asked to prove that Q_n is Hamiltonian, i.e., that there exists a spanning cycle of Q_n that includes every vertex of Q_n and is a subgraph of Q_n.

Given that the problem asserts that our graph Q_n contains "all binary strings (vectors) of length n", and that a difference in exactly one position between the vectors of two vertices implies a connection between them, we know that every possible connection must be included in the configuration. As a result, by alternating between our bipartite sets X and Y, we will always be able to "weave" between vertices to complete a Hamiltonian cycle (see fig 4).

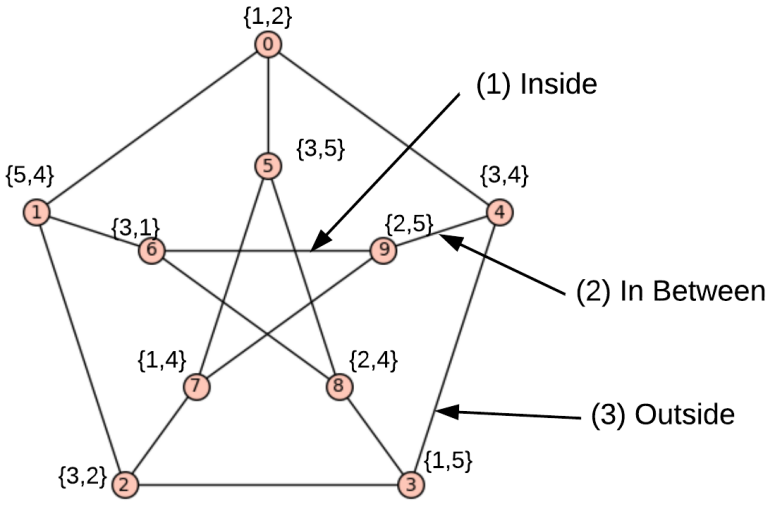
fig. 4



✠ **Sub-Experience Two: The Matrix Has You.**

(**No outside resources, please.**) Let P denote the graph with $V(P) = \{2\text{-sets of } [5]\}$ and vertices adjacent if and only if they are disjoint (as sets). The **complexity** of a labeled graph is the number of spanning trees contained in the graph. Let $\tau(G)$ denote the complexity of graph G.

Figure 1: Graph P



2.1 Please compute $\tau(P)$, the complexity of P.

The complexity of a graph is the number of spanning trees of that graph. The following code using Sage was used to compute the number of spanning trees of graph P.

```
P = graphs.PetersenGraph()
lenG = len(P.spanning_trees())
```

The variable lenG holds the number of spanning trees, or the complexity of the graph P. The value of lenG is 2000, so $\tau(P) = 2000$.

2.2 Let P' be P with one edge deleted. Compute $\tau(P')$ and argue that $\tau(P')$ does not depend on the edge deleted.

Considering the graph P is represented in figure 1, the edges of the graph can be classified into three different groups, an inside, outside, and in between edge. The adjacency matrix of the graph P can be manipulated to remove one of each classified edge. The below code using Sage will modify the adjacency matrix of P, deleting the edge between vertex 0 and 5, then count the number of spanning trees.

```
P = graphs.PetersenGraph()
M = Matrix(P)
# Deleting a (3) between edge, the adjacency
# of 0 and 5 in the graph P (Figure 1)
M[0,5] = 0
M[5,0] = 0
```

```
# Generate the modified graph
p1 = Graph(M)
lenP1 = len(p1.spanning_trees())
```

The variable lenP1 stores the number of spanning trees of a (2) between edge, which is the complexity of P'. This process can be repeated for each type of edge to compare the effect they have on the complexity of P'. The table belows shows the different class of edge and the complexity of the graph after it has been removed.

Edge Class	$\tau(P')$
1	800
2	800
3	800

Since the complexity of the graph doesn't change when each of the different types of edges are chosen, each of the nodes connections can be indistinguishable.

2.3 Let P'' be P with two edges deleted. Compute $\tau(P'')$, but note that this value will depend on which edges are deleted.Please try to determine all possible valued of $\tau(P'')$. (I hope you find a way to do this without computing all $\binom{15}{2} = 105$ possibilities.)

Complexity is independent of the specific edges a node is connected to, so when deleting two edges the distance between the deleted edges needs to be accounted for. The maximum distance between two edges in this graph is two, so only three cases need to be considered, which are having a distance of zero, one, and two between the two edges. The process of deleting edges is the same as explained in 2.2, but now four entries of the adjacency matrix of P will be changed.

```
P = graphs.PetersenGraph()
M = Matrix(P)
# Deleting an edge of distance 0
# for the graph P (Figure 1)

# Deleting the first edge
M[0,5] = 0
M[5,0] = 0
# Deleting the second edge
M[0,4] = 0
M[4,0] = 0

# Generate the modified graph
p1 = Graph(M)
lenP1 = len(p1.spanning_trees())
```

The table below contains the value of $\tau(P'')$ for deleting edges a distance of zero to two.

Distance	$\tau(P'')$
0	240
1	300
2	320

The **trace** of a square matrix is the sum of its diagonal entries.Let G be a graph and A its adjacency matrix.

2.4 Please show that the trace of A^3 is 6t, where t is the number of K_3s in G.

Consider two graphs G, H, whose adjacency matrices are defined as $A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$ and $B = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$ respectively.

Figure 2: Graph G

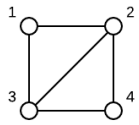
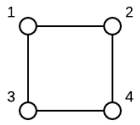


Figure 3: Graph H



Consider graph G, the number of K_3s in the graph is 2, so t in 6t should equal 2. To confirm this value, the trace of the adjacency matrix A^3 needs to be equal to 6(2).

$$A^3 == \begin{bmatrix} 2 & 5 & 5 & 2 \\ 5 & 4 & 5 & 5 \\ 5 & 5 & 4 & 5 \\ 2 & 5 & 5 & 2 \end{bmatrix}$$

The trace of A^3 is 12, which is equal to 6(2), so the trace of A^3 is equal to 6t. Now consider graph H. The number of K_3s is 0, so the trace of B^3 should also be 0.

$$B^3 = \begin{bmatrix} 0 & 4 & 4 & 0 \\ 4 & 0 & 0 & 4 \\ 4 & 0 & 0 & 4 \\ 0 & 4 & 4 & 0 \end{bmatrix}$$

Its trace is 0, and when set equal to 6t, $t = 0$. This is correct as there are no K_3s in graph H.

✠ **Sub-Experience Three: Some Tournament Problems.**

Part One: Curling. Suppose 10 teams compete in a curling competition where each team plays every other team and no ties are allowed.Officials decide that there will be two elimination rounds the first of which will eliminate all teams except for those which beat at least 7 other teams. The second elimination round will eliminate all teams which do not beat a majority of the other teams. The final competition will be among the teams which remain after the second elimination round.

A. ***Determine with proof the maximum number of teams which may remain after the first elimination round.***

The first elimination round can be expressed using an Incidence Matrix. It works similarly to a tournament, where the result of each match must be either of the contestants winning, with all teams playing each other once.

In order to determine the maximum number of teams that can make it through this round, we can start filling out entries in our Matrix to represent teams beating 7 other teams, which is their required number of wins to make it to the second round. So, the Matrix showing the first team beating 7 other teams is:

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	0	0
2	0	0								
3	0		0							
4	0			0						
5	0				0					
6	0					0				
7	0						0			
8	0							0		
9	1								0	
10	1									0

From here, we can also represent the scenario where teams 2 and 3 are able to beat 7 teams.

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	0	0
2	0	0	1	1	1	1	1	1	1	0
3	0	0	0	1	1	1	1	1	1	1
4	0	0	0	0						
5	0	0	0		0					
6	0	0	0			0				
7	0	0	0				0			
8	0	0	0					0		
9	1	0	0						0	
10	1	1	0							0

Teams 4-8 have lost all their games vs 3 teams already, making it impossible for them to score 7 wins. Let's

move down to the teams on the bottom, these teams still have a chance of making it out of this round.

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	0	0
2	0	0	1	1	1	1	1	1	1	0
3	0	0	0	1	1	1	1	1	1	1
4	0	0	0	0					0	0
5	0	0	0		0				0	0
6	0	0	0			0			0	0
7	0	0	0				0		0	0
8	0	0	0					0	0	0
9	1	0	0	1	1	1	1	1	0	1
10	1	1	0	1	1	1	1	1	0	0

Teams 9 and 10 were able to get enough wins to make it to the next round. This left teams 4 through 8 behind.This is the most optimal way of distributing wins among teams in order to maximize the number of teams with 7 wins.

Based on this, we can determine the maximum number of teams that can make it through the first elimination round is 5

B. ***Determine with proof the maximum number of teams which may remain after the second elimination round.***

For the second elimination round, we will work with the 5 teams that made it past the last round. We need to maximize the number of teams that get though this round. Each team is playing 4 other teams, and they need to beat a majority of their contenders, so for a contestant to make it through they need to win 3 out of the 4 games they play. We will follow a similar approach to the one used in the last problem:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & & \\ 0 & 0 & & 0 & \\ 1 & 0 & & & 0 \end{pmatrix} \end{array}$$

Teams 3 and 4 are unable to get enough wins to make it through, so we shift our attention to team 5.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & & 0 \\ 0 & 0 & & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Regardless of what happens in their game against each other, neither team 3 or team 4 can make it through this tournament. We were still able to maximize the number of teams that make it through by ensuring each team that makes it through does so with the least number of wins required.

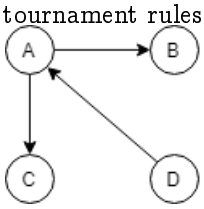
Based on this, we can determine that at most 3 teams can make it through the second elimination round.

Part Two: Strong Tournaments. Recall that a directed D graph is **strong** if between any pair of vertices x and y, there is an x, y-path, and a y, x-path.

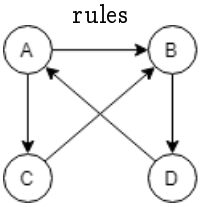
A. ***Prove that in any strong tournament*** T ***on at least*** 4 ***vertices, there exist two distinct vertices*** x ***and*** y ***such that*** $T - x$ ***and*** $T - y$ ***are strong. Prove or disprove whether the conclusion "*** $T - x - y$ ***is strong" holds as well.***

In order to prove the first part, we can make a general 4 vertex tournament and determine if we do indeed have these vertices. In a general 4 vertex tournament, we have two vertices that beat 2 vertices directly, and 2 that only beat one.

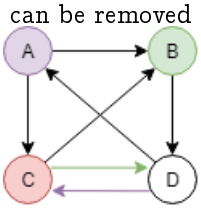
Start by picking any vertex to beat other two, this vertex must be beaten by the one it doesn't beat, as per tournament rules



One of the nodes beaten by A must beat the other, the losing one must beat D in order to follow tournament rules

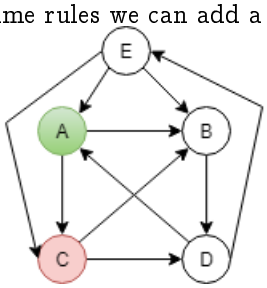


In this scenario, vertex C can be removed from the graph and the resulting graph would still be strong. Depending on the direction of the arrow between nodes C and D we will have a different second vertex that can be removed



If C beats D, vertex B is now removable since we have a 3-cycle in which it does not belong. Same happens to vertex A if D beats C.

We have proven how a 4-vertex tournament must have these vertices, in order to show how higher-count tournaments follow these same rules we can add a new vertex to this scenario.

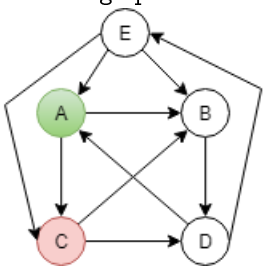


When adding a new vertex, the connections for the new Tournament formed may change one of the of the vertices that can be removed, but one of them will still remain as one that can be removed. Therefore, if we have a strong tournament T on at least 4 vertices, there will always be two vertices x and y such that $T - x$ and $T - y$ are strong

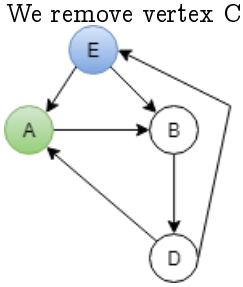
In order prove that a Strong tournament is still strong after removing both of these vertices, we can direct our attention to a statement about the behavior of these removable vertices made above. When we add a new node onto an already strong tournament a couple of things happen: the resulting tournament is still strong, one

of the removable vertices may change, and the other removable vertex will remain removable.
From these observations, we can work backwards and understand a couple of things that will happen once we remove one of these vertices. We would expect a couple of things to happen. For one, the resulting tournament would still be strong, meaning that (as long as it has at least 4 vertices) there are two vertices that can be removed from it and it will still be strong. One of these aforementioned vertices will be the other vertex we didn't remove. This means that if we were to remove both vertices from the start, the resulting graph would still be strong!
To show this, let's look at the following example:

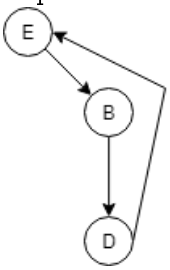
Start with the graph we saw earlier:



We remove vertex C



The resulting strong graph still has two removable nodes. Not only that, but one of them is one of the ones we could have removed in the previous step. After removing it, this is our resulting graph:



Based on this, we can determine that after removing both vertices identified earlier, the resulting graph is still strong.

Or in other words

$T - x - y$ is strong

B-1. ***Let*** T ***be a tournament on*** $n \geq 3$ ***vertices and let*** s ***be a vertex of*** T**.** ***Please prove the following statement:*** T **is strong if and only if for every vertex** $t \in V(T) \setminus \{s\}$ **there is an** s,t**-path and a** t,s**-path.** ***Is this statement true if*** T ***is a digraph (not necessarily a tournament).***

Since this is an if and only if statement, we have to prove both sides of the implication. First, let us begin by proving how the left side leads to the right one:

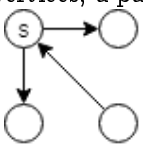
T is strong, therefore, for every vertex $t \in V(T) \setminus \{s\}$ there is an s,t-path and a t,s-path

Based on **Theorem 12.4.4**, which states that every vertex in a strong tournament belongs to a cycle of length k, where $3 \leq k \leq n$. So, based on this theorem, there is always bound to be an n-cycle in which every single vertex in the graph participates. Since this is the case, then, through this cycle we can get from s to any other vertex in T and back. This shows how T being a strong graph implies the existence of an s,t-path and a t,s-path.

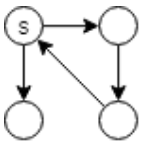
Now, to show how the right side implies the left one:

For every vertex $t \in V(T) \setminus \{s\}$ there is an s,t-path and a t,s-path, therefore T is strong

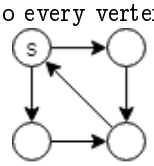
What the left side states, is that if we pick any vertex s in T we can find a path to and from any other vertex in this Tournament. This behavior implies that there must be a cycle involving at least 3 nodes between s and every other vertex. Each of these cycles is not separate from one another, which implies higher levels of connectivity between all of them. This principle entails the existence of cycles within our Tournament that go up to a length of n. To show this principle in action, let's look at a simple 4-vertex tournament:
Create a path from s to two other vertices, a path from the third one must go to s:



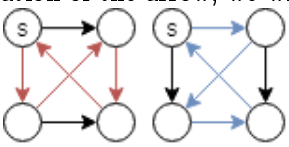
Now, create a path from s to the vertex it can't reach directly:



Now there's an s,t-path to every vertex. Draw the last t,s-path:

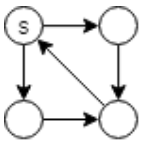


We have 2 three-cycles in this graph, but they are disconnected from one another. By connecting them, no matter the orientation of the arrow, we will form a 4-cycle:



This can be extrapolated to higher vertex counts, but the result is still the same. When connecting two n-cycles to one another tournament-style, the result is a $n+1$-cycle. So, when we link together the graphs formed by having s,t and t,s paths, we are bound to have cycles of size k where $3 \leq k \leq n$. A tournament with these paths must, based on **Theorem 12.4.4**, be strong.

This statement is not true when T is a digraph that is not a tournament. Based on this graph:



Even though there's clearly t, s and s, t paths for all vertices t, this graph is not strong due to its lack of a 4-cycle.

B-2. ***Suppose you have an algorithm $\mathcal{A}$ that determines whether there is a path from x to y, where x and y are vertices in a digraph. Suppose $\mathcal{A}$, when given a pair of vertices and a digraph D, performs this calculation with M operations in the worst case.How many operations, in the worst case, are needed to use $\mathcal{A}$ to determine whether D is strong using the standard definition of 'strong'?How many operations, in the worst case, are needed if the alternative definition of 'strong' provided in B-1?***

In this scenario, we have two very different ways of determining if a graph is strong. First, let's test the standard definition

T is **strong** if between any pair of vertices x and y, there is an x, y-path, and a y, x-path.

To determine if there is a path to and from each pair of vertices in the graph, we have to compute the algorithm twice for each distinct pair of vertices. The number of times this algorithm would need to be computed is:

$$n(n-1)$$

This number is equal to 2 times the number of edges in a tournament, which was stated in **Proposition 11.10.2**

So, since the worst-case runtime of the algorithm is M, then the total runtime to calculate if a graph with n vertices is strong is

$$O(n) = M(n)(n-1)$$

Now, let's look at how this differs from the method outlined in $B-1$:

T is strong if and only if for every vertex $t \in V(T) \setminus \{s\}$ there is an s, t-path and a t, s-path.

This method differs from the standard one in that instead of checking if there's a path to and from every single pair of vertices, we instead focus on seeing if there are paths to and from some vertex s and every other vertex in the tournament.

This results in a much smaller number of checks, $2(n-1)$ to be exact.

So, based on this, our worst-case runtime for this method is

$$O(n) = 2M(n-1)$$

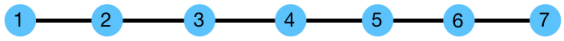
✠ Sub-Experience Four: An Optimization Problem

Let H denote an arbitrary graph. Recall that the distance between vertices in H is the length of the shortest path that has those vertices as its endpoints. Denote by $d_{max}(H)$ the maximum distance among all pairs of vertices of H.Recall that $\Delta(H)$ denotes the maximum degree among all vertices of H.

Define the function $N(d,k)$ to be the maximum number of vertices among all graphs H with $d_{max}(H) = d$ and $\Delta(H) = k$.

One. ***Determine*** $N(n,2)$**.**

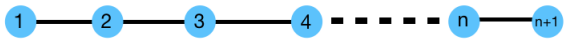
Consider a simple tree graph,



$$N(6,2) = 7$$

Each vertex has degree of 2, except for the 'end' vertices which are of degree 1. This implies that $k = 2$ for this example graph. And as you can see the max distance, d, is equal to 6. This is distance from the farthest left vertex to the farthest right vertex. And as you can see the number of vertices of this graph is 7, or $d+1$.

If we extend this concept to n vertices, the optimal graph will be of the following form:

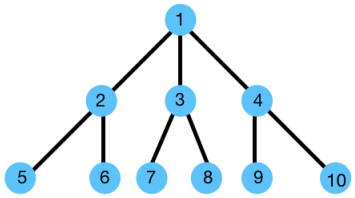


$$N(n,2) = n + 1$$

Therefore, $N(n,2) = n + 1$

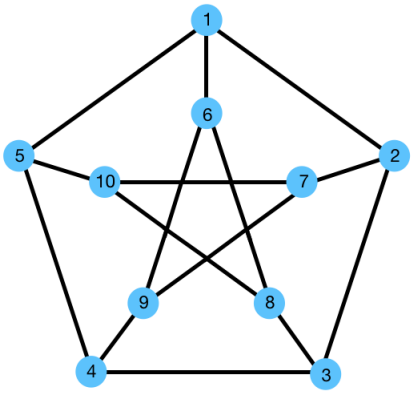
Two. ***Determine*** $\mathrm{N}(2,3)$.

Consider (without loss of generality) building a graph starting with 1 vertex and then maximizing that vertex's degree. From there, maximize the degree of the vertices generated from maximizing the degree of the first vertex. This process with give an upper bound for $\mathrm{N}(2,3)$. This can be seen in the following graph:



$N(2,3) \leq 10$

In this case, the maximum can actually be achieved. This can be seen in the 'Peterson Graph'.



$N(2,3) = 10$

Therefore, $\mathrm{N}(2,3) = 10$

Three. ***Verify that the graph*** G ***in Figure 4 has*** $d_{max}(G) = 2$.

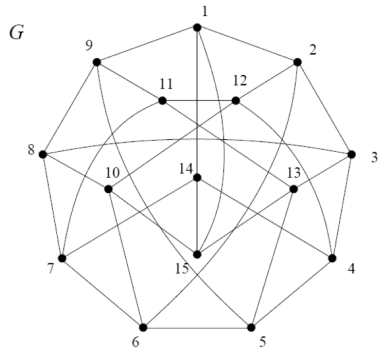


Figure 4: Potential 4-regular graph with diameter 2 on 15 vertices.

Note: **Theorem A. Suppose** G **is a graph and** $V(G) = \{v_1, v_2, ..., v_n\}$**. Then if A is the adjacency matrix of** G**, entry** (i, j) **in** A^k **is the number of walks of length** k **from** v_i **to** v_j**.**

Consider the adjacency matrix of G,

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{array} \begin{array}{c} \begin{array}{cccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

A 1 in the (i, j) entry of this matrix represents a path of length one between vertices v_i and v_j. And a 0 represents no path of length one between the vertices.

Thus upon squaring this adjacency matrix, we can determine the vertices with paths between them of length 2 by Theorem A.

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{array} \begin{array}{c} \begin{array}{ccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{array} \\ \begin{pmatrix} 4 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 4 & 0 & 2 & 1 & 0 & 1 & 1 & 1 & 2 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 4 & 0 & 2 & 1 & 1 & 0 & 1 & 1 & 1 & 2 & 0 & 1 & 1 \\ 1 & 2 & 0 & 4 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 2 & 0 & 1 \\ 1 & 1 & 2 & 0 & 4 & 0 & 1 & 1 & 0 & 1 & 2 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 4 & 0 & 2 & 1 & 0 & 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 4 & 0 & 2 & 2 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 2 & 0 & 4 & 0 & 0 & 2 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 2 & 0 & 4 & 1 & 0 & 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 1 & 1 & 0 & 2 & 0 & 1 & 4 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 2 & 1 & 0 & 2 & 0 & 1 & 4 & 0 & 0 & 1 & 1 \\ 1 & 0 & 2 & 0 & 1 & 2 & 1 & 1 & 1 & 0 & 0 & 4 & 1 & 1 & 1 \\ 1 & 1 & 0 & 2 & 0 & 1 & 1 & 1 & 2 & 1 & 0 & 1 & 4 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 4 \end{pmatrix} \end{array}$$

To show that every vertex that did not have a path length of 1 has a path of length 2, I will cross out the entries, in the squared adjacency matrix, that were 1′s in the adjacency matrix.

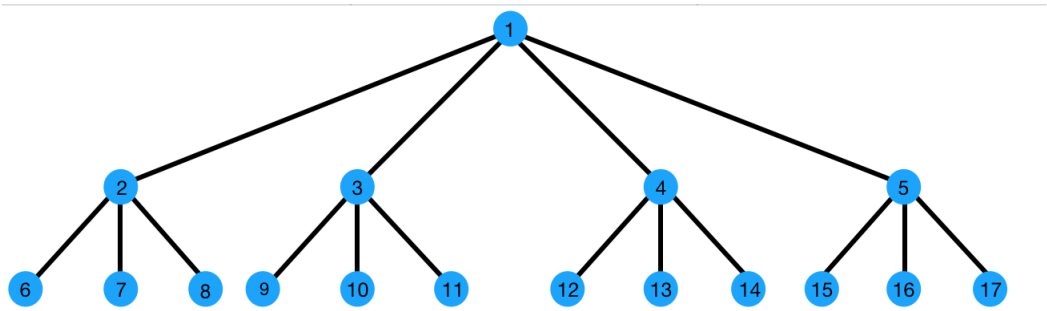
$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{array} \begin{array}{c} \begin{array}{ccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{array} \\ \begin{pmatrix} 4 & \times & 1 & 1 & 1 & 1 & 1 & 1 & \times & 1 & 1 & 1 & 1 & \times & \times \\ \times & 4 & \times & 2 & 1 & \times & 1 & 1 & 1 & 2 & 1 & \times & 1 & 1 & 1 \\ 1 & \times & 4 & \times & 2 & 1 & 1 & \times & 1 & 1 & 1 & 2 & \times & 1 & 1 \\ 1 & 2 & \times & 4 & \times & 1 & 1 & 1 & 1 & 1 & 1 & \times & 2 & \times & 1 \\ 1 & 1 & 2 & \times & 4 & \times & 1 & 1 & \times & 1 & 2 & 1 & \times & 1 & 1 \\ 1 & \times & 1 & 1 & \times & 4 & \times & 2 & 1 & \times & 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & \times & 4 & \times & 2 & 2 & \times & 1 & 1 & \times & 1 \\ 1 & 1 & \times & 1 & 1 & 2 & \times & 4 & \times & \times & 2 & 1 & 1 & 1 & 1 \\ \times & 1 & 1 & 1 & \times & 1 & 2 & \times & 4 & 1 & \times & 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 1 & 1 & \times & 2 & \times & 1 & 4 & 1 & \times & 1 & 1 & \times \\ 1 & 1 & 1 & 1 & 2 & 1 & \times & 2 & \times & 1 & 4 & \times & \times & 1 & 1 \\ 1 & \times & 2 & \times & 1 & 2 & 1 & 1 & 1 & \times & \times & 4 & 1 & 1 & 1 \\ 1 & 1 & \times & 2 & \times & 1 & 1 & 1 & 2 & 1 & \times & 1 & 4 & 1 & \times \\ \times & 1 & 1 & \times & 1 & 1 & \times & 1 & 1 & 1 & 1 & 1 & 1 & 4 & \times \\ \times & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \times & 1 & 1 & \times & \times & 4 \end{pmatrix} \end{array}$$

As you can see, there are no remaining zeros in the squared adjacency matrix. This means that every vertex is connected with maximum distance of two.

$$\text{Therefore, } d_{max}(G) = 2$$

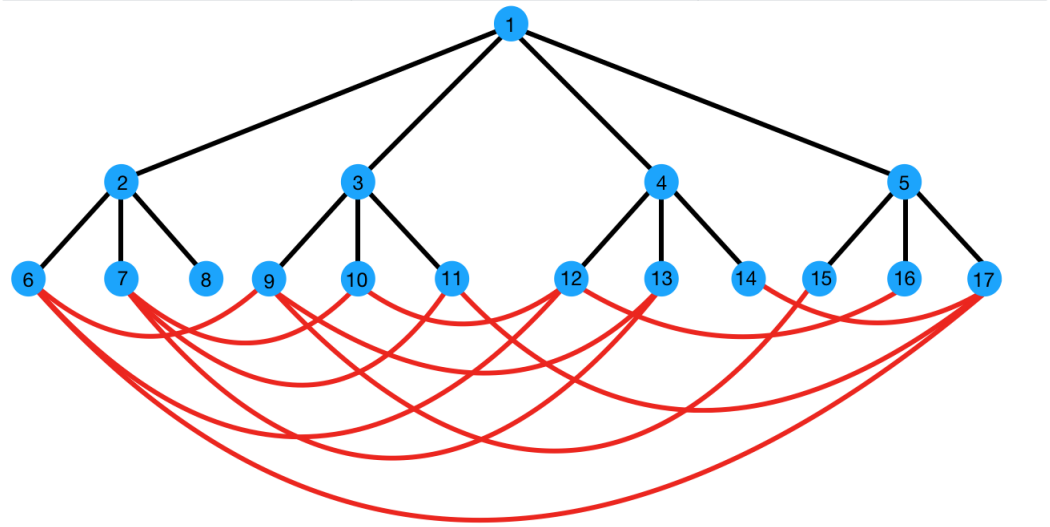
Four. ***Determine*** $\mathrm{N}(2,4)$.

Using the same process used in part 2 of this sub-experience, we can show that the upper bound of $\mathrm{N}(2,4)$ is 17 vertices. Consider the following graph:



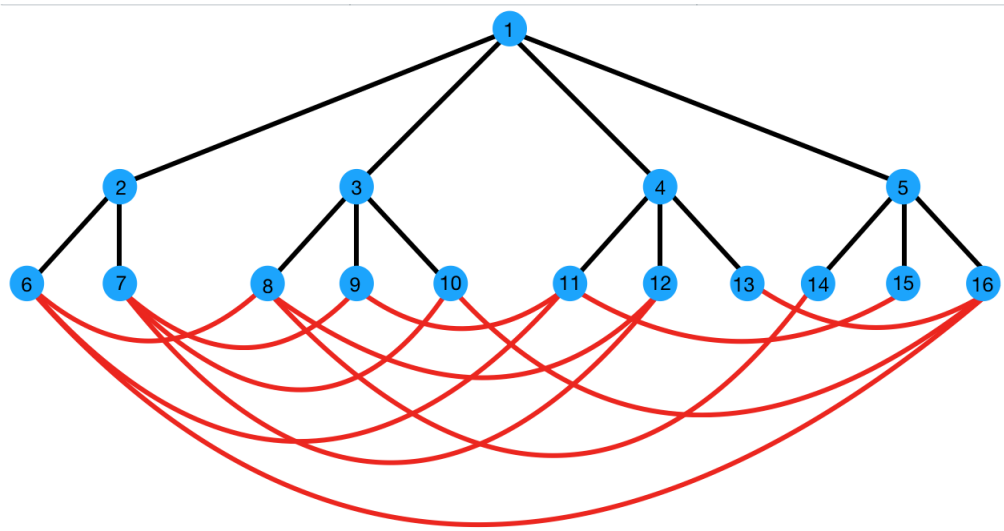
$$N(2,4) \leq 17$$

Without loss of generality, I will begin creating edges such that the maximum degree of each vertex is 4 and the maximum distance between vertices is 2. The following graph shows that although we can engineer vertex 6 to have max degree of 4 and maximum distance of 2 to every other vertex, it is impossible to do the same for vertex 7.



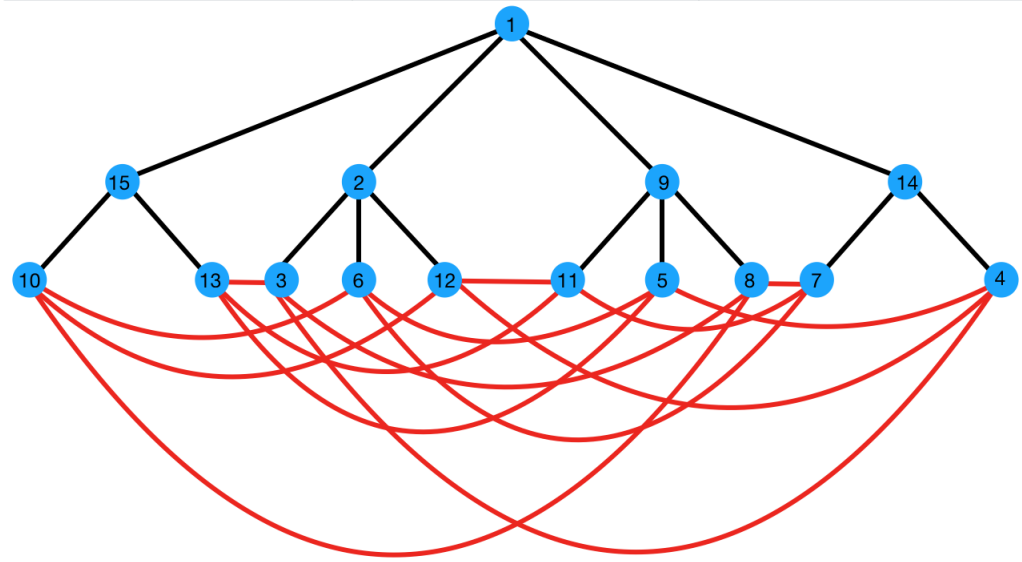
$$N(2,4) \neq 17$$

The same argument can be said for a graph with 16 vertices. I will use the same graph as on 17 vertices and then delete one vertex. This can be seen in the following graph:



$$N(2,4) \neq 16$$

But a graph on 15 vertices such that the maximum degree of the vertices is 4 and the maximum distance of 2 between all vertices is achievable. This can be seen in the following graph:



$$N(2,4) = 15$$

Therefore, $\mathrm{N}(2,4) = 15$

✠ Sub-Experience Five – A Matter of Life and Death

You are among a group of $n-1$ of your friends (pretend you have $n-1$ friends even if you don't, where n is possibly very large)and are captured by a horde of theater students who will force you and your $n-1$ friends to enact the play *Cats* with them over and over and over and ... . Your friends choose death over this fate and decide to form a circle and have every other person commit suicide (someone has a pistol and n bullets,which is quite reasonable to assume here at USU) until only one person survives,who will supposedly kill themselves. You want no part of this suicide madness since, you figure, the theater people will tire eventually and you can make your escape by taking Jennyanydots hostage with the pistol and remaining bullet and escape.

Number you and your friends 1 to n and assume you all form a vicious circle to facilitate the suicide.

Question: *In which position (call it* $S(n)$*) must you be in order to survive?*

Examples: $S(4) = 1$, $S(7) = 7$.

Consider the following image. The colored X's represent who dies in each round of suicides, or each time the gun gets completely passed around in the circle. The white circles are the position in which you must be in order to survive.

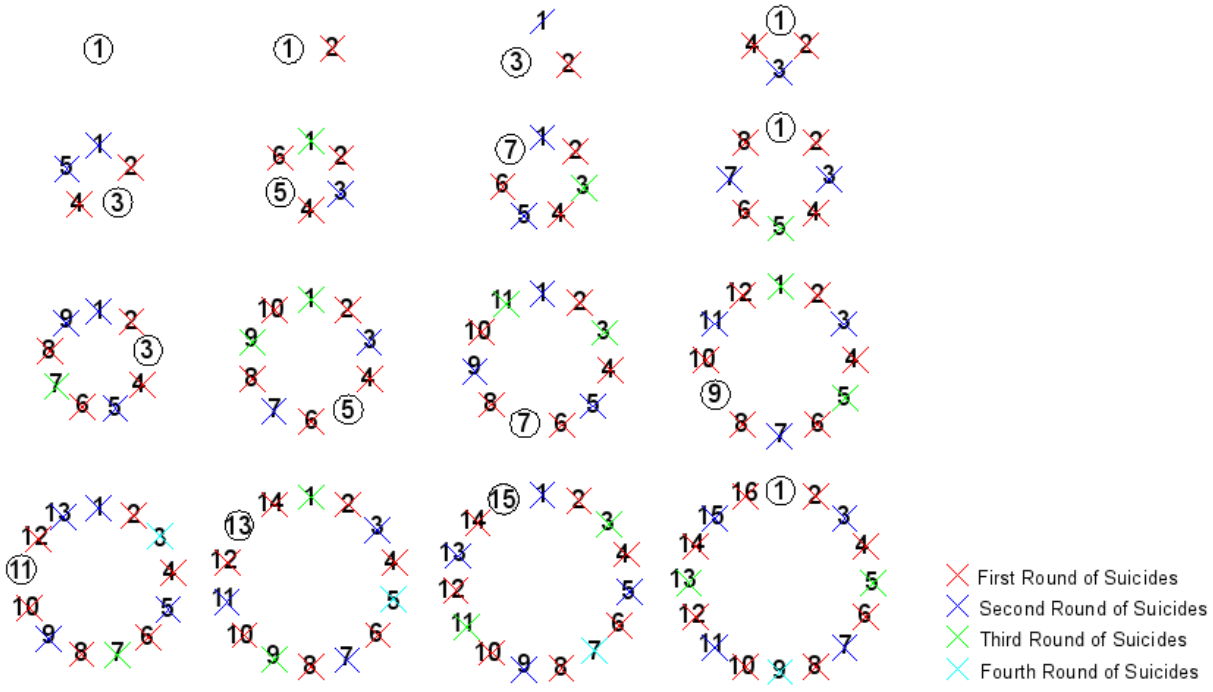


Figure 5.1

From the above figure, we are able to observe a pattern, which has been put in the table below where n is the number of people and $s(n)$ is the spot in which you must be to survive.

n	s(n)	n	s(n)	n	s(n)	n	s(n)	n	s(n)	n	s(n)
1	1	7	7	13	11	19	11	25	7	31	19
2	1	8	1	14	13	20	13	26	9	32	1
3	3	9	3	15	15	21	15	27	11	33	3
4	1	10	5	16	1	22	1	28	13	34	5
5	3	11	7	17	3	23	3	29	15	35	7
6	5	12	9	18	5	24	5	30	17	36	9

Table 5.1

We observe from the table above a pattern of incrementing odds that reset to one every 2^n. From this pattern, we can also observe a recurrence relation for the odd amount of people in a circle as well as for an even amount. In order to survive with an even number of people, so $2n$ people, we again observe from the table that the safe spot, $s(n)$, for any $2n$ is two times the safe spot of n minus one. The recurrence is as follows:

$$s(2n) = 2s(n) - 1$$

Similarly, to survive in a group with an odd number of individuals, which we will denote by $s(2n+1)$, and for every $s(2n+1)$, the safe spot will be two times $s(n)$ plus one, thus the recurrence is as follows:

$$s(2n+1) = 2s(n) + 1$$

To create a function $s(n)$, when again we refer to the table, we can see that every 2^n, the value resets to one. We also know that a property of $\log_2(x)$ is every power of two, it increases by one. So, with that, we concluded that the function for our set of data is $s(n) = 2(n - 2^{\lfloor \log_2(n) \rfloor}) + 1$. Since we have both the recurrence for an even and odd number of people and the function for $s(n)$, we can prove by mathematical induction that $s(2n) = 2s(n) - 1 = s(n) = 2(n - 2^{\lfloor \log_2(n) \rfloor}) + 1$ as well as $s(2n+1) = 2s(n) + 1 = s(n) = 2(n - 2^{\lfloor \log_2(n) \rfloor}) + 1$.

Proof by Mathematical Induction

Even:
We want $s(N) = 2(N - 2^{\lfloor \log_2(N) \rfloor}) + 1$

BASE CASE: When $n = 2$, $s(2) = 2(2 - 2^{\lfloor \log_2(2) \rfloor}) + 1 = 1$

To prove that $s(2n) = s(n)$, let $N = 2n$. If $s(2n) = 2s(n) - 1$, then:

$$s(2n) = 2s(n) - 1 \Rightarrow s(N) = 2s(n) - 1$$

If we substitute in $s(n) = 2(n - 2^{\lfloor \log_2(n) \rfloor}) + 1$ then:

$$s(N) = 2s(n) - 1 \Rightarrow s(N) = 2(2(n - 2^{\lfloor \log_2(n) \rfloor}) + 1) - 1$$

Distributing the 2:

$$\Rightarrow 2(2n - 2 \cdot 2^{\lfloor \log_2(n) \rfloor} + 1) - 1$$

Using properties of logarithms, we move the "2" into the exponent and substitute in $N = 2n$:

$$\Rightarrow 2(N - 2^{\lfloor \log_2(n) \rfloor + 1} + 1) - 1$$

Now, by properties of floor functions, we know $\lfloor x + n \rfloor = \lfloor x \rfloor + n, \forall\ n \epsilon \mathbb{N}$, so:

$$\Rightarrow 2(N - 2^{\lfloor \log_2(n) + \log_2(2) \rfloor}) + 2 - 1 \Rightarrow 2(N - 2^{\lfloor \log_2(2n) \rfloor}) + 1 = 2(N - 2^{\lfloor \log_2(N) \rfloor}) + 1$$

Thus, $2s(n) - 1 = 2(N - 2^{\lfloor \log_2(N) \rfloor}) + 1$

Odd:
We want $s(N) = 2(N - 2^{\lfloor \log_2(N) \rfloor}) + 1$

BASE CASE: When $n = 1$, $s(1) = 2(1 - 2^{\lfloor \log_2(1) \rfloor}) + 1 = 1$

To prove that $s(2n+1) = s(n)$, let $N = 2n + 1$. If $s(2n+1) = 2s(n) + 1$, then:

$$s(2n+1) = 2s(n) + 1 \Rightarrow s(N) = 2s(n) + 1$$

Substitute in the $\log_2$ function to get:

$$s(N) = 2(2(n - 2^{\lfloor \log_2(n) \rfloor}) + 1) + 1$$

Now, if we solve for n in $N = 2n + 1$ and substitute it into our equation, we get:

$$s(N) = 2(2[(\frac{N-1}{2}) - 2^{\lfloor \log_2(\frac{N-1}{2}) \rfloor}] + 1) + 1$$

Distribute the 2:

$$\Rightarrow 2(2(\frac{N-1}{2}) - 2^{\lfloor \log_2(\frac{N-1}{2}) \rfloor + \log_2(2)}] + 1) + 1 \Rightarrow s(N) = 2((N-1) + 1 - 2^{\lfloor \log_2(N-1) \rfloor}) + 1$$

Because the $\lfloor \log_2(N-1) \rfloor$ is a floor function, each odd n will be the same as the last even n, so we can neglect the -1 and the equation is equivalent to:

$$\Rightarrow 2(N - 2^{\lfloor \log_2(N) \rfloor}) + 1$$

And thus, both even and odd recursive formulas are equivalent to $s(n)$, and therefore have proved:

$$s(2n) = s(n) \text{ and } s(2n+1) = s(n).$$

In order to be in the spot escape and survive of n people, you must be in the $s(n) = 2(n - 2^{\lfloor \log_2(n) \rfloor}) + 1$ position.
