CS 2130
Computational Structures

GROUP PROJECT

Each group will research and analyze a specific problem related to either games of chance or computational puzzles. The project requires collaboration among team members and will provide an opportunity to investigate a single topic in more detail.

This project requires that each group write a program that simulates, demonstrates, and validates the objectives suggested by the assignment description. Each group will be assigned a different task:

Team 1:

Research and investigate the Monty Hall problem, also known as the three-door problem. This problem examines the probability of winning a prize in a game made famous in the old "Let's Make a Deal" show hosted by Monty Hall.

Write a program that validates the central claim made by the solution to the problem. At a minimum, your code should perform 10 million simulations of the game to determine the probability of winning a prize.

Team 2:

Research and investigate the Tower of Hanoi puzzle, invented by the French mathematician Edouard Lucas in 1883.

Write a program that illustrates the solution to the puzzle. At a minimum, your code should provide a simulation showing how five disks stacked in decreasing size from bottom-to-top can be successfully moved from the source peg to the target peg.

Team 3:

Research and investigate Nim, a two-player game in which players alternately remove at least one object from a single heap. The game typically involves more than one heap, and during normal play, the person who takes the last object wins. Nim has been solved mathematically, which is another way to say that a winning strategy has been determined.

Write a program that illustrates the winning strategy for a normal play of Nim. At a minimum, your code should provide a winning strategy for a game that starts with 3 heap sizes of 5, 6, and 7, and the winning person makes the first move.

<u>Team 4:</u>

The following legend about the first-century historian Flavius Josephus is recounted:

In the Jewish revolt against Rome, Josephus and 39 of his comrades were holding out against the Romans in a cave. With defeat imminent, they resolved that, like the rebels at Masada, they would rather die than be slaves to the Romans. They decided to arrange themselves in a circle. One man was designated as number one, and they proceeded clockwise killing every seventh man.....Josephus (according to the story) was among other things an accomplished mathematician; so he instantly figured out where he ought to sit in order to be the last to go. But when the time came, instead of killing himself he joined the Roman side.

In some versions of the story, Josephus saves a friend by having the friend stand in a position so that the two of them are the last two people left alive.

Write a program that will determine both Josephus' position and his friend's position given *p* people and a skip amount denoted by *s*. At a minimum, your code should provide answers to the following given that *J(p,s)* represents a Josephus trial with *p* people and skip amount *s*.

- Josephus and friend position for *J(39,7)*
- Josephus and friend position for *J(15,3)*
- Josephus and friend position for *J(15,2)*
- Josephus and friend position for *J(32,2)*
- Josephus and friend position for *J(5,4)*

<u>Team 5:</u>

A technique named the **Sieve of Eratosthenes** can be used to find all primes not exceeding a specified positive integer. Write a program that employs this technique to find and display all the primes not exceeding 1000.

<u>Team 6:</u>

A young, newborn pair of rabbits (one of each sex) is placed on an island. A pair of rabbits does not breed until they are 2 months old. After they are 2 months old, each pair of rabbits produces another pair each subsequent month.

Assuming that no rabbits die, write a program that models the reproduction of these rabbits and that can output the number of pairs of rabbits on the island for each month between 1 and 50 inclusive.
With sufficient planning, each project can be easily completed during the in-class time allotted. The program can be written using Java, C/C++, or Prolog. ***All code submitted for this project should be original. Significant point deductions will occur for cases in which programs have been "borrowed" from external sources.***

Assessment criteria for the project include:

- Clarity and completeness of report.
- Correctness of computations, mathematics, and algorithms
- Style and neatness

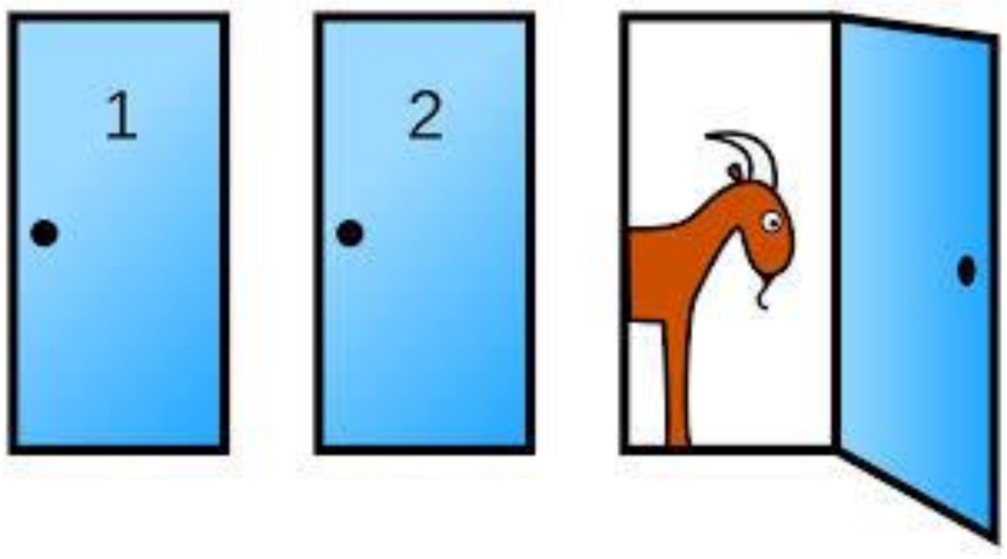
Your Design Group should prepare a written report containing the following sections:

- A cover page listing the group members and project name.
- An abstract summarizing the problem statement.
- A **Proposed Solution** section that includes (1) computational algorithms and strategies used to solve the programming problem, (2) any mathematical techniques employed to formulate the algorithms, and (3) a brief analysis of program output or results.
- A summary with concluding comments, including the group's reflections about researching the topic. Was the area of investigation difficult or easy? What roadblocks needed to be overcome? What was learned in the process generally about discrete mathematics research, and specifically about the chosen topic?

You are required to include a source code appendix as an upload to Canvas. This upload shall be a program listing in Java, C/C++, or Prolog that implements your solution for your problem. **Remember that code submitted for the group project should contain appropriate inline documentation and change log as described in the syllabus**.

Important: Make sure the problem that is being investigated is clearly and sufficiently explained. ***Assume your audience has no prior exposure to the topic being examined by your group.***

Team 1 - Monty Hall problem



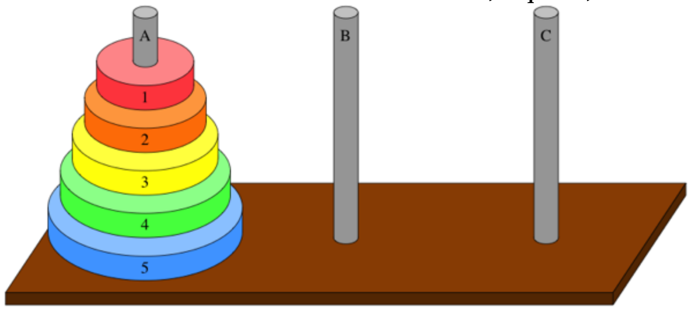
Run random simulations of the Monty Hall game. Show the effects of a strategy of the contestant always keeping his first guess so it can be contrasted with the strategy of the contestant always switching his guess.

> Suppose you're on a game show and you're given the choice of three doors. Behind one door is a car; behind the others, goats. The car and the goats were placed randomly behind the doors before the show. The rules of the game show are as follows: After you have chosen a door, the door remains closed for the time being. The game show host, Monty Hall, who knows what is behind the doors, now has to open one of the two remaining doors, and the door he opens must have a goat behind it. If both remaining doors have goats behind them, he chooses one randomly. After Monty Hall opens a door with a goat, he will ask you to decide whether you want to stay with your first choice or to switch to the last remaining door. Imagine that you chose Door 1 and the host opens Door 3, which has a goat. He then asks you "Do you want to switch to Door Number 2?" Is it to your advantage to change your choice?
> Stefan Krauss, X. T. Wang, "The psychology of the Monty Hall problem: Discovering psychological mechanisms for solving a tenacious brain teaser.", Journal of Experimental Psychology: General, Vol 132(1), Mar 2003, 3-22 DOI: 10.1037/0096-3445.132.1.3

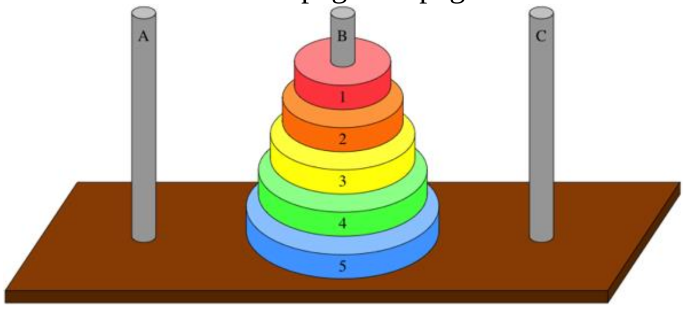
Note that the player may initially choose any of the three doors (not just Door 1), that the host opens a different door revealing a goat (not necessarily Door 3), and that he gives the player a second choice between the two remaining unopened doors.

Team 2 - Towers of Hanoi
from Kahn Academy

You are given a set of three pegs and n n nn disks, with each disk a different size. Let's name the pegs A, B, and C, and let's number the disks from 1, the smallest disk, to n n nn, the largest disk. At the outset, all n n nn disks are on peg A, in order of decreasing size from bottom to top, so that disk n n nn is on the bottom and disk 1 is on the top. Here's what the Towers of Hanoi looks like for n=5 n = 5 n=5n, equals, 5 disks:



The goal is to move all n n nn disks from peg A to peg B:



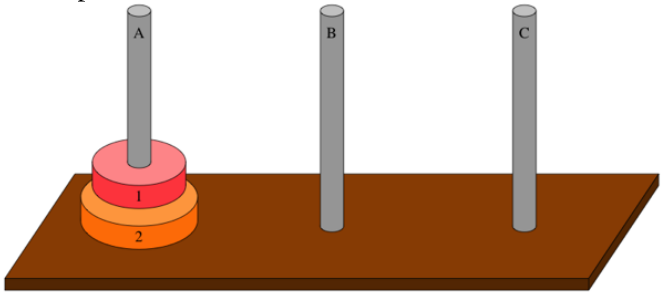
Sounds easy, right? It's not quite so simple, because you have to obey two rules:

1. You may move only one disk at a time.
2. No disk may ever rest atop a smaller disk. For example, if disk 3 is on a peg, then all disks below disk 3 must have numbers greater than 3.

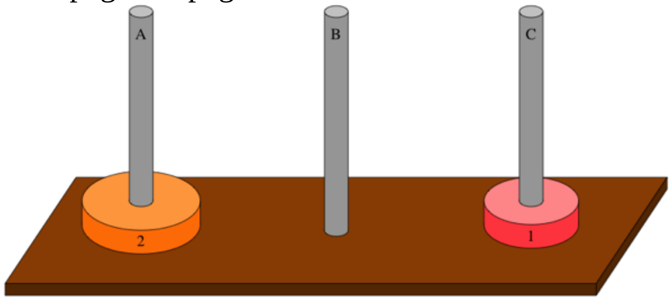
You might think that this problem is not terribly important. *Au contraire!* Legend has it that somewhere in Asia (Tibet, Vietnam, India—pick which legend on the Internet you like), monks are solving this problem with a set of 64 disks, and—so the story goes—the monks believe that once they finish moving all 64 disks from peg A to peg B according to the two rules, the world will end. If the monks are correct, should we be panicking in the streets?

First, let's see how to solve the problem recursively. We'll start with a really easy case: one disk, that is, n=1 n = 1 n=1n, equals, 1. The case of n=1 n = 1 n=1n, equals, 1 will be our base case. You can always move disk 1 from peg A to peg B, because you know that any disks below it must be larger. And there's nothing special about pegs A and B. You can move disk 1 from peg B to peg C if you like, or from peg C to peg A, or from any peg to any peg. Solving the Towers of Hanoi problem with one disk is trivial, and it requires moving only the one disk one time.

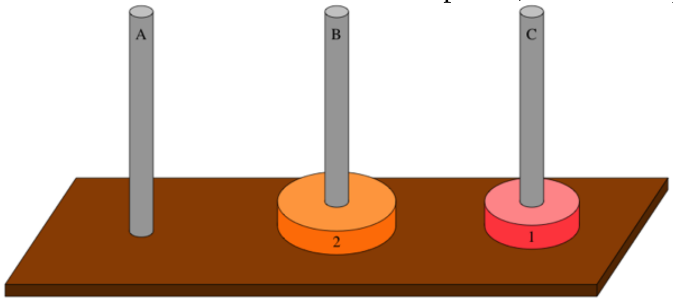
How about two disks? How do you solve the problem when n=2 n = 2 n=2n, equals, 2? You can do it in three steps. Here's what it looks like at the start:



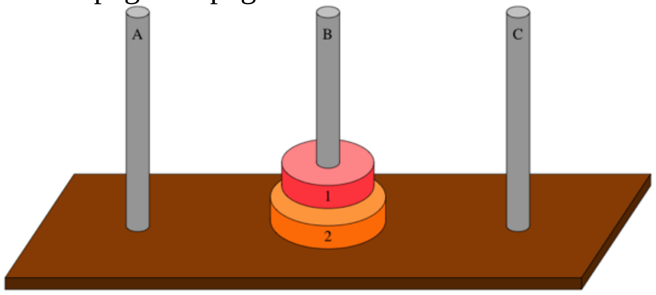
First, move disk 1 from peg A to peg C:



Notice that we're using peg C as a spare peg, a place to put disk 1 so that we can get at disk 2. Now that disk 2—the bottommost disk—is exposed, move it to peg B:



Finally, move disk 1 from peg C to peg B:



This solution takes three steps, and once again there's nothing special about moving the two disks from peg A to peg B. You can move them from peg B to peg C by using peg A as the spare peg: move disk 1 from peg B to peg A, then move disk 2 from peg B to peg C, and finish by moving disk 1 from peg A to peg C. Do you agree that you can move disks 1 and 2 from any peg to any peg in three steps? (Say "yes.")

Team 3 -Combinatorial Game Theory | Set 2 (Game of Nim)

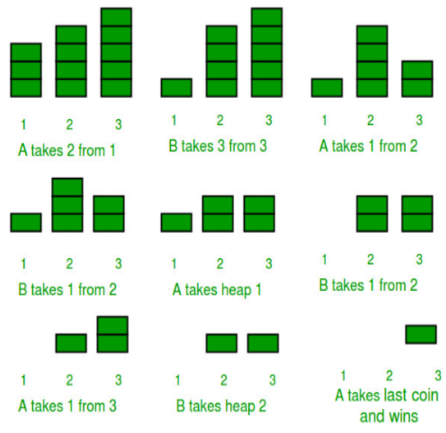
Taken from: https://www.geeksforgeeks.org/combinatorial-game-theory-set-2-game-nim/.

The Game of Nim is described by the following rules-

" *Given a number of piles in which each pile contains some numbers of stones/coins. In each turn, a player can choose only one pile and remove any number of stones (at least one) from that pile. The player who cannot move is considered to lose the game (i.e., one who take the last stone is the winner).* "

For example, consider that there are two players- **A** and **B**, and initially there are three piles of coins initially having **3, 4, 5** coins in each of them as shown below. We assume that first move is made by **A**. See the below figure for clear understanding of the whole game play.

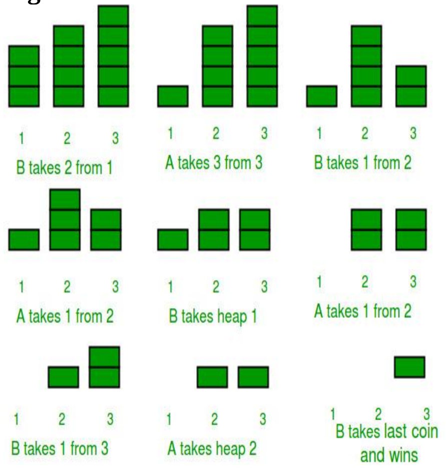
Figure 1: A won and A made the first move



So was **A** having a strong expertise in this game ? or he/she was having some edge over **B** by starting first ?

Let us now play again, with the same configuration of the piles as above but this time **B** starting first instead of **A**.

Figure 2: B won the match and moved first



By figure 2, it must be clear that the game depends on one important factor – Who starts the game first ?

So does the player who starts first will win every time ?

Let us again play the game, starting from A , and this time with a different initial configuration of piles. The piles have 1, 4, 5 coins initially.

Will A win again as he has started first ? Let us see.

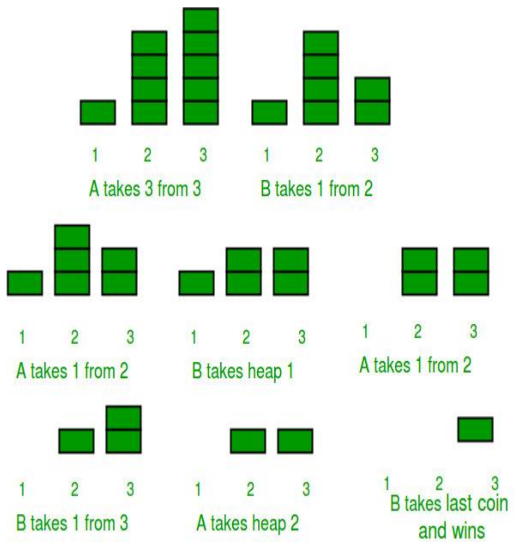


Figure 3: A moves first but loses

So, the result is clear. **A** has lost. But how? We know that this game depends heavily on which player starts first. Thus, there must be another factor which dominates the result of this simple-yet-interesting game. That factor is the initial configuration of the heaps/piles. This time the initial configuration was different from the previous one.

So, we can conclude that this game depends on two factors-

1. The player who starts first.
2. The initial configuration of the piles/heaps.

In fact, we can predict the winner of the game before even playing the game !

Nim-Sum : The cumulative XOR value of the number of coins/stones in each piles/heaps at any point of the game is called Nim-Sum at that point.

"If both A and B play optimally (i.e- they don't make any mistakes), then the player starting first is guaranteed to win if the Nim-Sum at the beginning of the game is non-zero. Otherwise, if the Nim-Sum evaluates to zero, then player A will lose definitely."

For the proof of the above theorem, see-
https://en.wikipedia.org/wiki/Nim#Proof_of_the_winning_formula

Let us apply the above theorem in the games played above. In the first game **A** started first and the Nim-Sum at the beginning of the game was, 3 XOR 4 XOR 5 = 2, which is a non-zero value, and hence **A** won. Whereas in the second game-play, when the initial configuration of the piles were 1, 4, and 5 and **A** started first, then **A** was destined to lose as the Nim-Sum at the beginning of the game wasv1 XOR 4 XOR 5 = 0 .

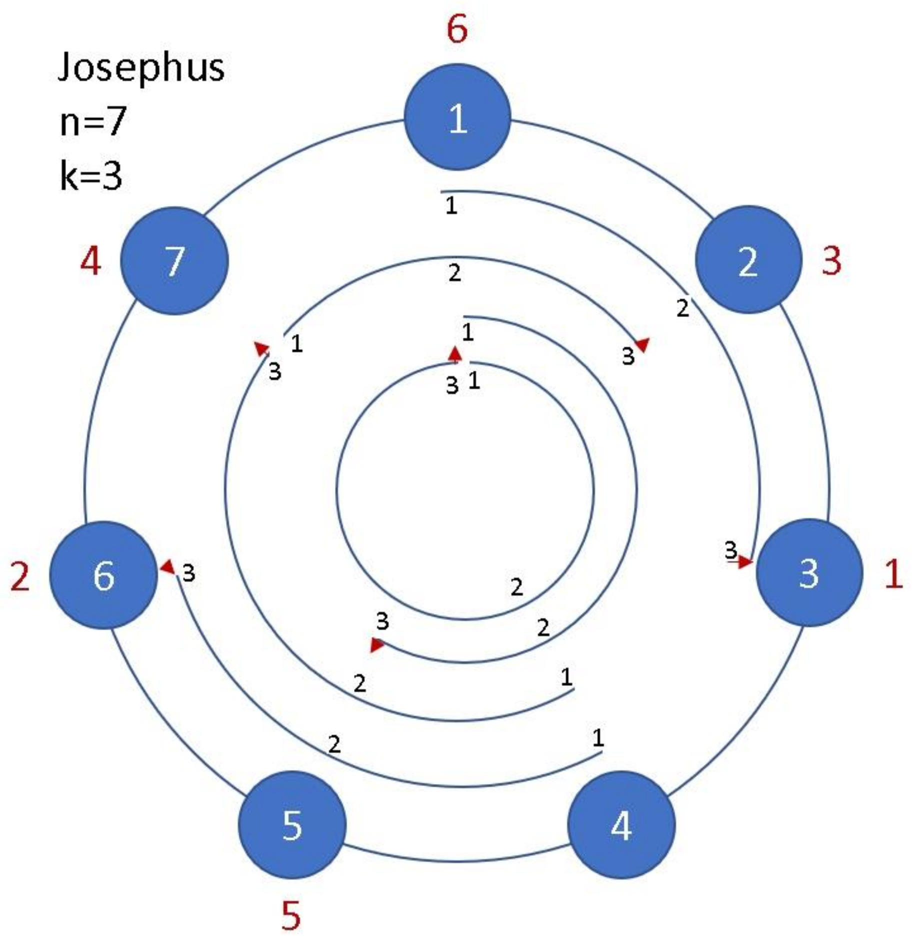
Team 4 - Josephus problem

In computer science and mathematics, the Josephus Problem (or Josephus permutation) is a theoretical problem. Following is the problem statement:

There are n people standing in a circle waiting to be executed. The counting out begins at some point in the circle and proceeds around the circle in a fixed direction. In each step, a certain number of people are skipped and the next person is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last person remains, who is given freedom. Given the total number of persons n and a number k which indicates that k-1 persons are skipped and kth person is killed in circle. The task is to choose the place in the initial circle so that you are the last one remaining and so survive.

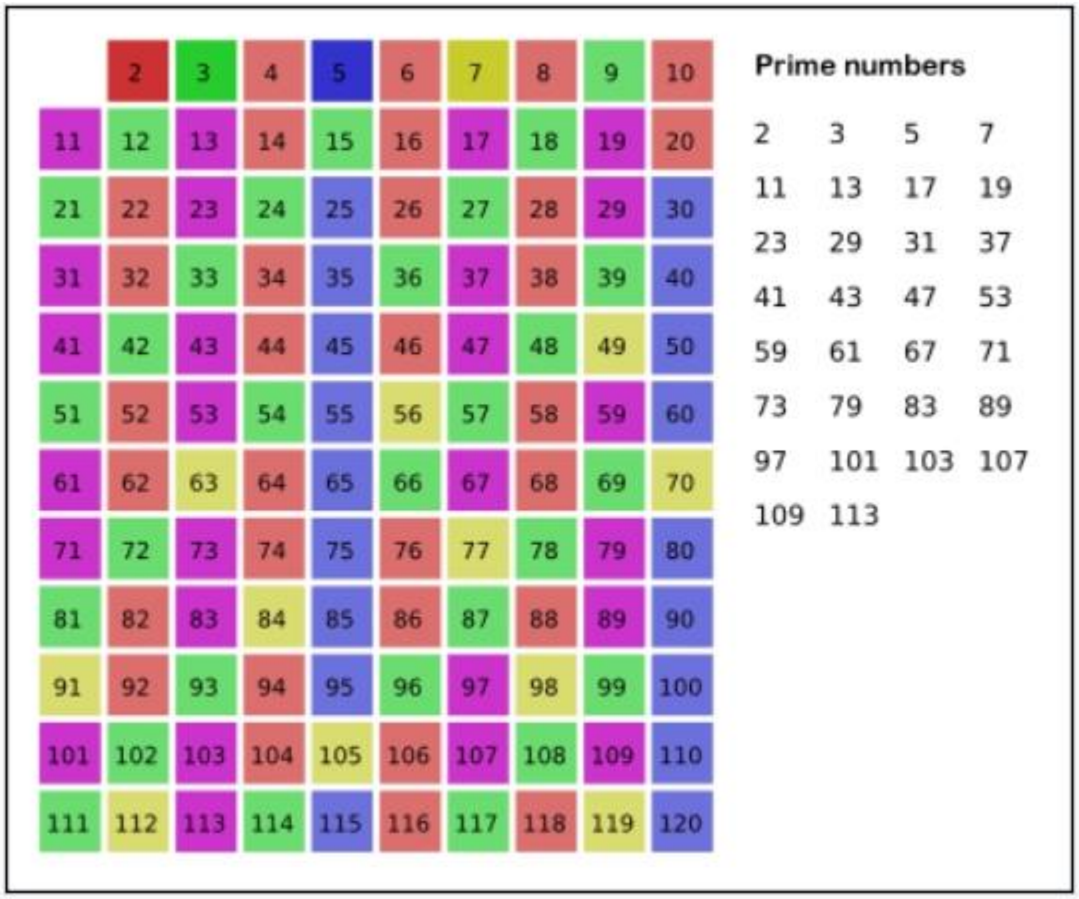
For example, if n = 5 and k = 2, then the safe position is 3. Firstly, the person at position 2 is killed, then person at position 4 is killed, then person at position 1 is killed. Finally, the person at position 5 is killed. So the person at position 3 survives.
If n = 7 and k = 3, then the safe position is 4. The persons at positions 3, 6, 2, 7, 5, 1 are killed in order, and person at position 4 survives.

And another example:



Team 5 - Sieve of Eratosthenes

From Wikipedia, the free encyclopedia



In mathematics, the **sieve of Eratosthenes** is a simple, ancient algorithm for finding all prime numbers up to any given limit.

It does so by iteratively marking as composite (i.e., not prime) the multiples of each prime, starting with the first prime number, 2. The multiples of a given prime are generated as a sequence of numbers starting from that prime, with constant difference between them that is equal to that prime.[1] This is the sieve's key distinction from using trial division to sequentially test each candidate number for divisibility by each prime.[2]

The earliest known reference to the sieve (Ancient Greek: κόσκινον Ἐρατοσθένους, *kóskinon Eratosthénous*) is in Nicomachus of Gerasa's *Introduction to Arithmetic*,[3] which describes it and attributes it to Eratosthenes of Cyrene, a Greek mathematician.

One of a number of prime number sieves, it is one of the most efficient ways to find all of the smaller primes. It may be used to find primes in arithmetic progressions.[4]

A prime number is a natural number that has exactly two distinct natural number divisors: 1 and itself.

To find all the prime numbers less than or equal to a given integer *n* by Eratosthenes' method:

1. Create a list of consecutive integers from 2 through *n*: (2, 3, 4, ..., *n*).
2. Initially, let *p* equal 2, the smallest prime number.
3. Enumerate the multiples of *p* by counting in increments of *p* from 2*p* to *n*, and mark them in the list (these will be 2*p*, 3*p*, 4*p*, ...; the *p* itself should not be marked).
4. Find the first number greater than *p* in the list that is not marked. If there was no such number, stop. Otherwise, let *p* now equal this new number (which is the next prime), and repeat from step 3.
5. When the algorithm terminates, the numbers remaining not marked in the list are all the primes below *n*.

The main idea here is that every value given to *p* will be prime, because if it were composite it would be marked as a multiple of some other, smaller prime. Note that some of the numbers may be marked more than once (e.g., 15 will be marked both for 3 and 5).

As a refinement, it is sufficient to mark the numbers in step 3 starting from p^2, as all the smaller multiples of *p* will have already been marked at that point. This means that the algorithm is allowed to terminate in step 4 when p^2 is greater than *n*.[1]

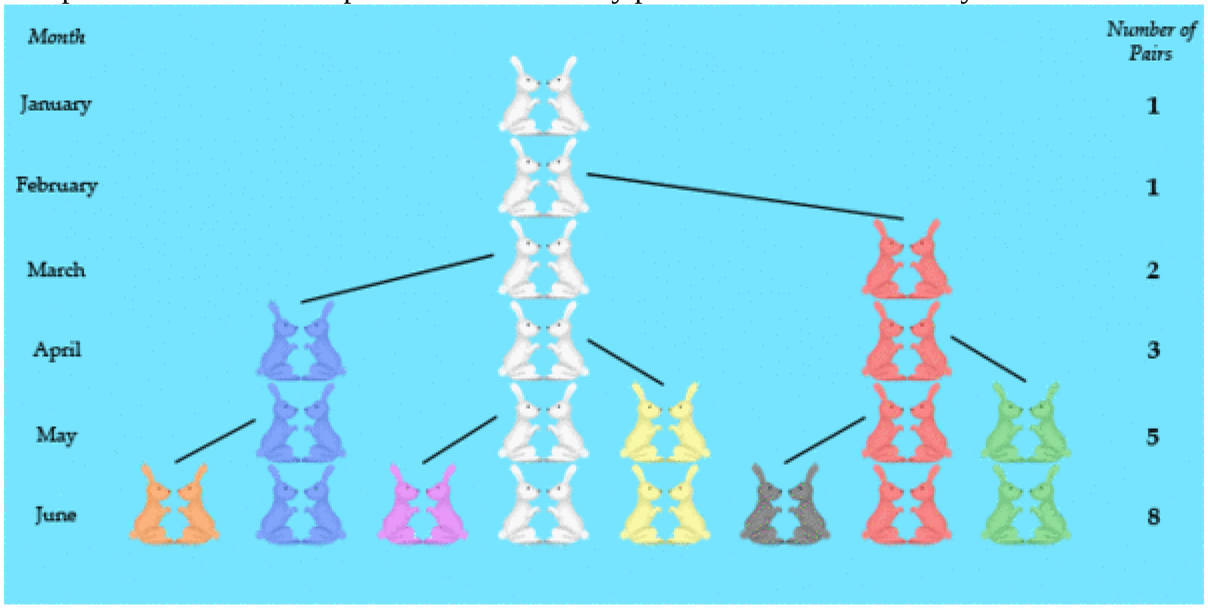
Another refinement is to initially list odd numbers only, (3, 5, ..., *n*), and count in increments of 2*p* from p^2 in step 3, thus marking only odd multiples of *p*. This actually appears in the original algorithm.[1] This can be generalized with wheel factorization, forming the initial list only from numbers coprime with the first few primes and not just from odds (i.e., numbers coprime with 2), and counting in the correspondingly adjusted increments so that only such multiples of *p* are generated that are coprime with those small primes, in the first place.[6]

Team 6 - Fibonacci's Rabbits

In the West, the Fibonacci sequence first appears in the book *Liber Abaci* (1202) by Leonardo of Pisa, known as Fibonacci. Fibonacci considers the growth of an idealized (biologically unrealistic) rabbit population, assuming that:

1. a single newly born pair of rabbits (one male, one female) are put in a field;
2. rabbits are able to mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits;
3. rabbits never die and a mating pair always produces one new pair (one male, one female) every month from the second month on.

The puzzle that Fibonacci posed was: how many pairs will there be in one year?


Month
Number of Pairs
January
1
February
1
March
2
April
3
May
5
June
8