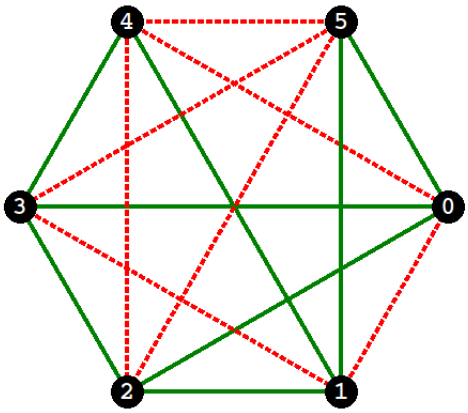
Ramsey graphs and combinatorics

You will see that a very simple problem, which concerns the relationships between only six people, will require a lot of calculations to be solved.



Lesson 1 (Ramsey's problem)**.**

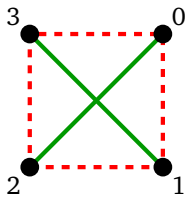
Proposition. In a group of 6 *people, there is always* 3 *friends (the three know each other) or* 3 *strangers (all three are strangers to each other).*

The purpose of this chapter is to have the computer demonstrate this statement. For that, we will model the problem by graphs, then we will check the statement for each of the 32 768 possible graphs.

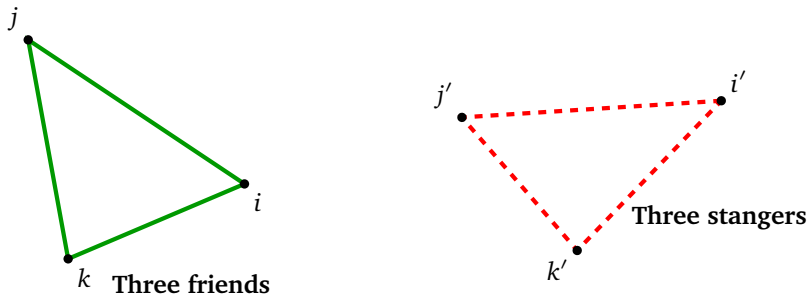
We consider n people. For two of them, either they know each other (they are friends) or they do not know each other (they are strangers to each other). We represent this using a graph:

- a person is represented by a vertex (numbered from 0 to $n-1$);
- if two people are friends, the corresponding vertices are connected by a green solid edge;
- otherwise (they are strangers), the corresponding vertices are connected by a red dotted edge.

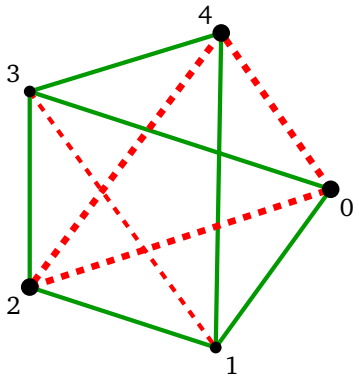
The graph below means that 0 is friend with 2; 1 is friend with 3. The other pairs don't know each other.



A graph checks the Ramsey problem, if there are among its vertices, either 3 friends, or either 3 strangers.



Here is an example of 5 peoples that verify the statement: there is group of 3 strangers (the 0, 2 and 4 vertices), even if there is no group of three friends.



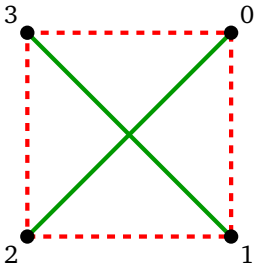
A graph with $n = 5$ that satifies Ramsey's statement

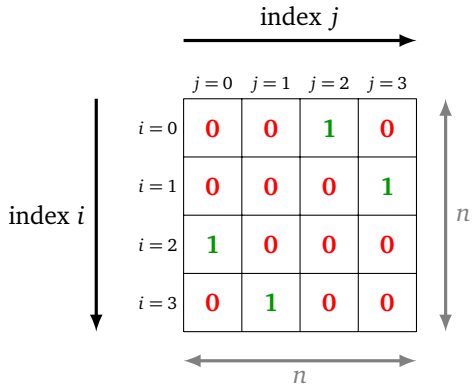
Lesson 2 (Model)**.**

We model a graph using an array, containing 0*'s and* 1*'s.*

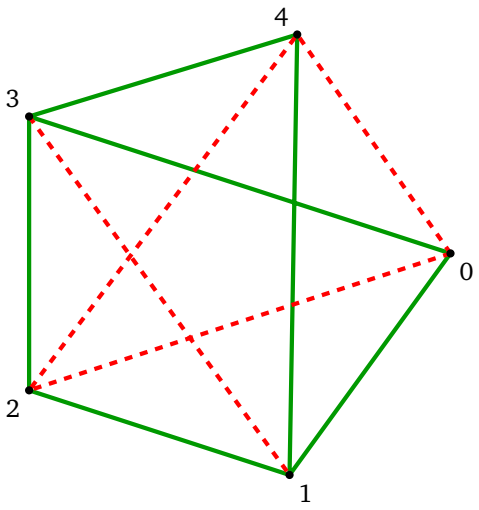
Let be a graph with n vertices, numbered from 0 to $n-1$. The ***array of the graph*** is a table of size $n \times n$ in which we place a 1 in position (i, j) if the vertices i and j are connected by an edge, otherwise we place a 0.

First example below: the vertices 0 and 2 are friends (because they are connected by a green edge) so the table contains a 1 in position $(0, 2)$ and also in $(2, 0)$. Similarly 1 and 3 are friends, so the table contains a 1 in positions $(1, 3)$ and $(3, 1)$. The rest of the table contains 0.





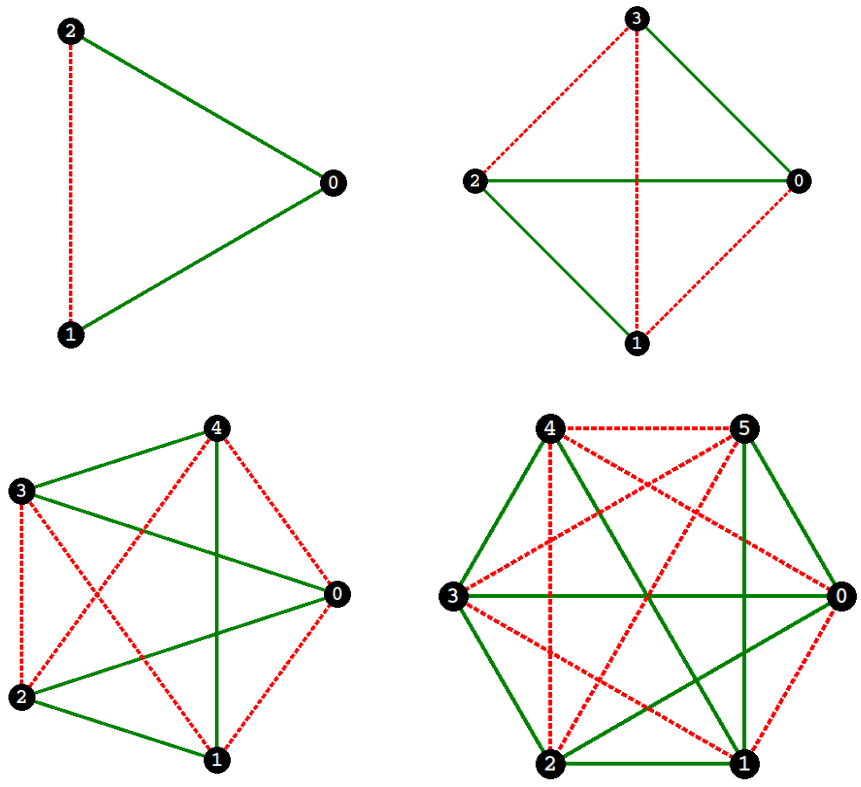
Here is a more complicated graph and its array:



	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$
$i=0$	0	1	0	1	0
$i=1$	1	0	1	0	1
$i=2$	0	1	0	1	0
$i=3$	1	0	1	0	1
$i=4$	0	1	0	1	0

Activity 1 (Build graphs)**.**

Goal: define graphs and test if three given vertices are friends.



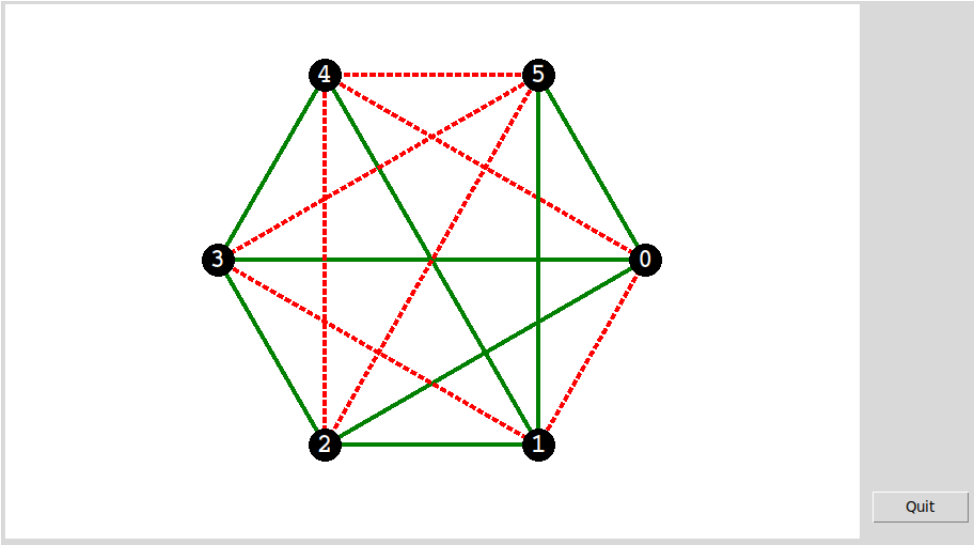
1. Define the graph array for the four examples above. You can start by initializing the array with
   `array = [[0 for j in range(n)] for i in range(n)]`
   Then add commands:
   `array[i][j] = 1` and `array[j][i] = 1`
   Don't forget that if vertex i is connected to the vertex j by an edge, then you have to put a 1 in position (i, j) but also in position (j, i).
2. Define a `print_graph(array)` function that allows you to display the table of a graph on the screen. Thus the third example above (with $n = 5$) should be displayed as follows:

```
0 0 1 1 0
0 0 1 0 1
1 1 0 0 0
1 0 0 0 1
0 1 0 1 0
```

3. We set the three vertices i, j, k of a graph. Write a `have_3_fix_friends(array,i,j,k)` function that tests if the vertices i, j, k are three friends (the function returns `True` or `False`). Do the same thing for a `have_3_fix_strangers(array,i,j,k)` function to find out if these vertices are strangers.
   Find on the fourth example (without computer), three friend or foreign vertices and check your answer using the functions you have just defined on these vertices.

Activity 2 (Draw nice graphs)**.**

Goal: draw a graph! Optional activity.

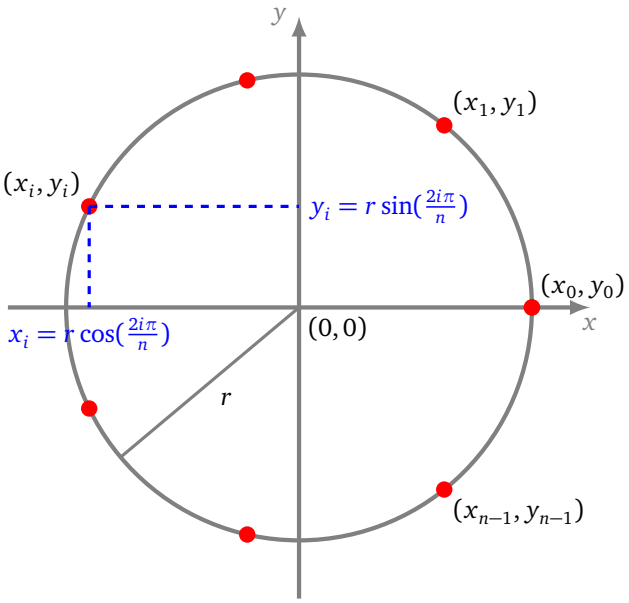


Program the graphical display of a graph by a `display_graph(array)` function.

Hints. This activity is not necessary for the next steps, it just helps to visualize the graphs. It is necessary to use the `tkinter` module and the `create_line()`, `create_oval()` and possibly `create_text()` functions.
The most delicate point is to obtain the coordinates of the vertices. You will need the sine and cosine functions (available in the `math` module). The coordinates (x_i, y_i) of the vertex number i of a graph with n elements can be calculated by using the formulas:

$$x_i = r\cos\left(\frac{2i\pi}{n}\right) \qquad \text{and} \qquad y_i = r\sin\left(\frac{2i\pi}{n}\right).$$

These vertices are located on the circle of radius r, centered at $(0,0)$. You will have to choose a large enough r (for example $r = 200$) and shift the circle to display it on the screen.



Activity 3 (Binary notation with zero-padding)**.**

Goal: convert an integer to binary notation with possible leading zeros.

Program an `integer_to_binary(p,n)` function that displays an integer p in binary notation using n bits. The result is a list of 0's and 1's.

Example.

- The binary notation of $p = 37$ is 1.0.0.1.0.1. If you want its binary notation using $n = 8$ bits you have to add two 0's in front of it: 0.0.1.0.0.1.0.1.
- So the result of the command `integer_to_binary(37,8)` must be `[0, 0, 1, 0, 0, 1, 0, 1]`.
- The command `integer_to_binary(37,10)` returns 37 in binary notation using 10 bits: `[0, 0, 0, 0, 1, 0, 0, 1, 0, 1]`.

Hints.

- You can use the `bin(p)` command.
- The `list(mystring)` command returns the list of characters making up `mystring`.
- Attention! We want a list of integers 0 or 1, not of characters `'0'` or `'1'`. The command `int('0')` returns 0 and `int('1')` returns 1.
- `mylist = mylist + [element]` adds an item at the end of the list, while `mylist = [element] + mylist` adds the item at the beginning of the list.

Lesson 3 (Subsets)**.**
Let $E_n = \{0, 1, 2, \ldots, n-1\}$ be the set of all integers from 0 to $n-1$. The set E_n therefore contains n elements. For example $E_3 = \{0, 1, 2\}$, $E_4 = \{0, 1, 2, 3\}\ldots$

Subsets.
What are the subsets of E_n? For example there are 8 subsets of E_3, these are:

- the subset $\{0\}$ composed of the single element 0;

- the subset $\{1\}$ composed of the single element 1;
- the subset $\{2\}$ composed of the single element 2;
- the subset $\{0,1\}$ composed of the element 0 and the element 1;
- the subset $\{0,2\}$;
- the subset $\{1,2\}$;
- the subset $\{0,1,2\}$ composed of all elements;
- and the empty set $\varnothing$, which contains no elements!

Proposition. The set E_n contains 2^n subsets.

For example $E_4 = \{0,1,2,3\}$ has $2^4 = 16$ possible subsets. Have fun finding them all! For E_6 there are $2^6 = 64$ possible subsets.

Subsets of fixed cardinal.
We are only looking for subsets with a fixed number k of elements.
Examples:

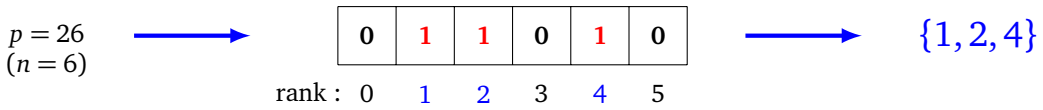
- For $n = 3$ and $k = 2$, the subsets having two elements and contained in $E_3 = \{0,1,2\}$ are the three pairs: $\{0,1\}$, $\{0,2\}$, $\{1,2\}$.
- For $n = 5$ and $k = 3$, the subsets having three elements and contained in $E_5 = \{0,1,2,3,4\}$ are the 10 triplets: $\{0,1,2\}$, $\{0,1,3\}$, $\{0,2,3\}$, $\{1,2,3\}$, $\{0,1,4\}$, $\{0,2,4\}$, $\{1,2,4\}$, $\{0,3,4\}$, $\{1,3,4\}$, $\{2,3,4\}$.

Activity 4 (Subsets)**.**

Goal: generate all subsets in order to test all triplets of vertices. For this we will use binary notation.

Here is how we associate a subset of $E_n = \{0,1,\ldots,n-1\}$ to each integer p verifying $0 \leqslant p < 2^n$.
Let's start with the example $n = 6$ and $p = 26$:

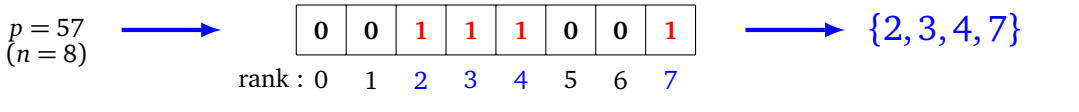
- the binary notation of $p = 26$ using $n = 6$ bits is `[0,1,1,0,1,0]`;
- there are 1's at indices 1, 2 and 4 (starting at rank 0 on the left);
- the associated subset is then $\{1,2,4\}$.



Initial integer p **Binary notation of** p **using** n **bits** **Subset associated to** p

Other examples.

- With $n = 8$ and $p = 57$ whose binary notation using 8 bits is `[0,0,1,1,1,0,0,1]`, the associated subset corresponds to the indices $2,3,4,7$, so the associated subset is $\{2,3,4,7\}$.



Initial integer p **Binary notation of** p **using** n **bits** **Subset associated to** p

- When $p = 0$, the binary notation is only formed of 0's, the associated subset is the empty set.

- When $p = 2^n - 1$, the binary notation is full of 1's, the associated subset is the whole set $E_n = \{0, 1, \ldots, n-1\}$ itself.

We model a set as a list of elements. For example:

- For us the set E_4 is the list `[0,1,2,3]`.
- `[1,3]` is an example of a subset of E_4.
- The empty set is represented by the empty list `[]`.

1. Program a `subsets(n)` function which returns the list of all the possible subsets of $E_n = \{0, 1, 2, \ldots, n-1\}$. For example, when $n = 3$, `subsets(n)` returns the list (which itself contains lists):

   ```
   [ [], [2], [1], [1, 2], [0], [0, 2], [0, 1], [0, 1, 2] ]
   ```

   That is to say the 8 subsets (starting with the empty set):

   $$\varnothing \quad \{2\} \quad \{1\} \quad \{1,2\} \quad \{0\} \quad \{0,2\} \quad \{0,1\} \quad \{0,1,2\}.$$

   Hint. To test your program, check that the returned list contains 2^n subsets.
2. Derive from it a `fix_subsets(n,k)` function that returns only the subsets of E_n that have exactly k elements.
   For example, for $n = 3$ and $k = 2$, `fix_subsets(n,k)` returns the list of pairs:

   ```
   [ [0, 1], [0, 2], [1, 2] ]
   ```

   Test your program:
   - For $n = 4$ and $k = 3$, the list returned by `fix_subsets(n,k)` contains 4 triplets.
   - For $n = 5$ and $k = 3$, there are 10 possible triplets.
   - For $n = 10$ and $k = 4$, there are 210 possible subsets!

From now on we will mostly use subsets with 3 elements. In particular, for $n = 6$, there are 20 triplets included in $\{0, 1, 2, 3, 4, 5\}$:

```
[[3, 4, 5], [2, 4, 5], [2, 3, 5], [2, 3, 4], [1, 4, 5],
 [1, 3, 5], [1, 3, 4], [1, 2, 5], [1, 2, 4], [1, 2, 3],
 [0, 4, 5], [0, 3, 5], [0, 3, 4], [0, 2, 5], [0, 2, 4],
 [0, 2, 3], [0, 1, 5], [0, 1, 4], [0, 1, 3], [0, 1, 2]]
```

Activity 5 (Ramsey's theorem for $n = 6$)**.**

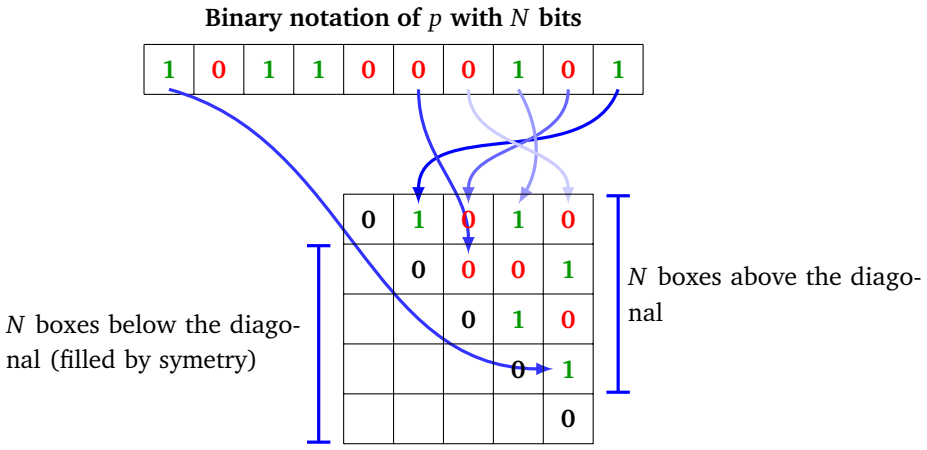
Goal: check that all graphs with 6 vertices contain three friends or three strangers.

1. Program a `have_3(array)` function that tests if a graph contains 3 friends or 3 strangers. You must therefore call the functions `have_3_fix_friends(array,i,j,k)` and `have_3_fix_strangers(array,i,j,k)` from the first activity on all possible triplets of vertices (i, j, k).
   For the four examples from the first activity, only the fourth (with 6 summits) passes the test.
2. Program an `all_graphs(n)` function that computes all the possible graph arrays with n vertices. There are $N = \frac{(n-1)n}{2}$ possible arrays. You can generate them by using a method similar to the one for subsets:
   - for each integer p that satisfies $0 \leqslant p < 2^N$,
   - calculate the binary notation of p using N bits,
   - fill the array in element by element, with the 0's and 1's of the binary notation.

   Hints. To write a function to fill an array from a given binary notation of p named `binary_notation` (that is to say a list of 0's and 1's), you can use a double loop like:

```
for j in range(0,n):
    for i in range(j+1,n):
        b = binary_notation.pop()
        array[i][j] = b
        array[j][i] = b
```

Here is the principle of the loop that fills in the part above the diagonal in the grid (and also the part below it by symmetry). This loop takes the last bit of the list and places it in the first empty box above the diagonal; then the second last bit is placed in the second empty box...; the first bit of the list fills the last empty square.



Filling of an array

3. Convert the previous function into a `test_all_graphs(n)` function which tests the conjecture "there are three friends or three strangers" for all graphs with n vertices. You must find that:
   - for $n = 4$ and $n = 5$ the conjecture is false. Give a graph having 4 vertices (then 5 vertices) that has neither 3 friends, nor 3 strangers;
   - for $n = 6$ let the computer check that, for each of the $N = 2^{\frac{5\times 6}{2}} = 32\,768$ graphs with 6 vertices, either it has 3 friends or it has 3 strangers.

Activity 6 (To go further).

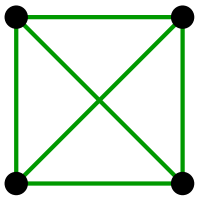
Goal: improve your program and prove other conjectures. Optional activity.

1. Improve your program so that it checks the conjecture for $n = 6$ in less than a second.
   Ideas.
   - The list of triplets must only be generated once at the beginning of the program (and not for each new graph).
   - It is not necessary to generate a list of all possible graphs, then test them in a second step. It is better to generate one and then test it before moving on to the next.
   - As soon as you find 3 friends (or 3 strangers) it's a win! Immediately stop the loop, even if it means using the `break` instruction, and move on to the next graph.
   - You can only test the graphs that correspond to p between 0 and $2^N/2$ (because for the rest of the p's it is like exchanging the green segments for red and vice versa).

   With these tips here are the calculation times you can expect:

Number of vertices	Number of graphs	Approximate calculation time
$n = 6$	32 768	$<$ 1 second
$n = 7$	2 097 152	$<$ 1 minute
$n = 8$	268 435 456	$<$ 1 hour
$n = 9$	68 719 476 476 736	$<$ 10 days

2. Here is a more difficult task. It is a question of finding out at what size n, a graph will always contain either 4 friends or 3 strangers. Being 4 friends means every pair is connected by a green segment, as below:



   (a) Find graphs with $n = 6$ (then $n = 7$) vertices that do not satisfy this statement.
   (b) By searching a little with your computer find graphs with 8 vertices that do not satisfy this statement.
   (c) Prove that any graph that has 9 vertices contains 4 friends or 3 strangers!
   Hints. It is necessary to test all the graphs corresponding to the integer p between 0 and $2^N = 2^{\frac{8\times 9}{2}} = 68\,719\,476\,736$. The total calculation time is about 20 days! You can split the calculations between several computers: one computer does the calculations for $0 \leqslant p \leqslant 1\,000\,000$, a second computer for $1\,000\,001 \leqslant p \leqslant 2\,000\,000$,...

3.
   - There are sets of arguments that prove with pencil and paper that for $n = 6$ there are always 3 friends or 3 strangers. Look for this kind of reasoning! With a little more effort, we can also prove that $n = 9$ answers the 4 friends/3 strangers problem.
   - We know how to prove that we need $n = 18$ to always have 4 friends or 4 strangers.
   - However, no one in the world knows what the smallest n is for the 5 friends/5 strangers problem!