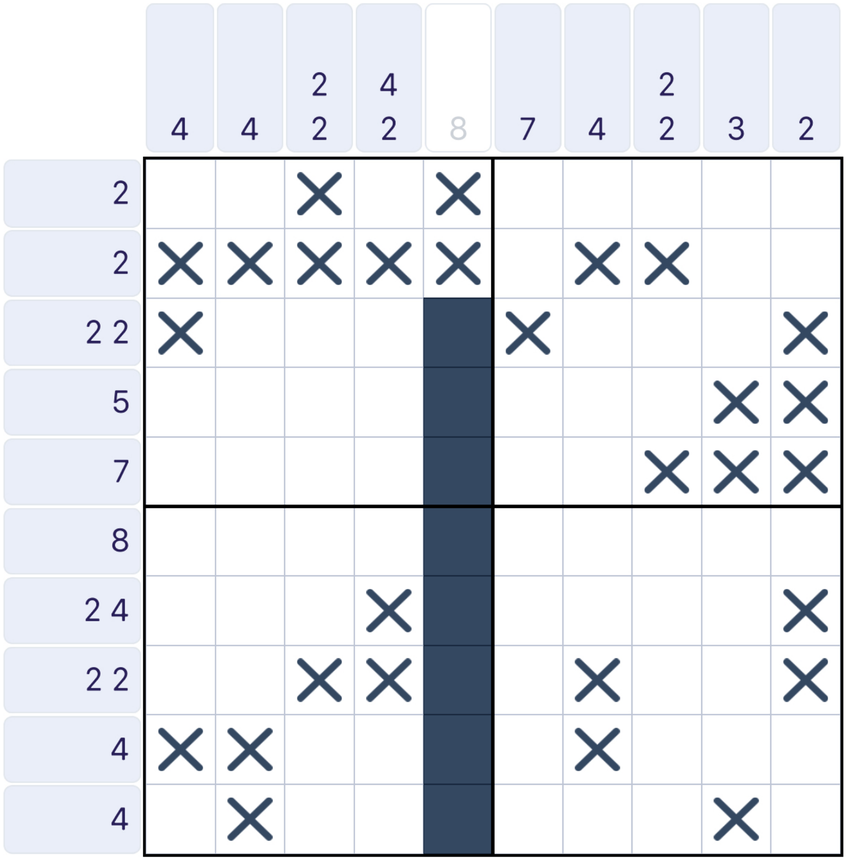
Project Nonogram

Problem Statement

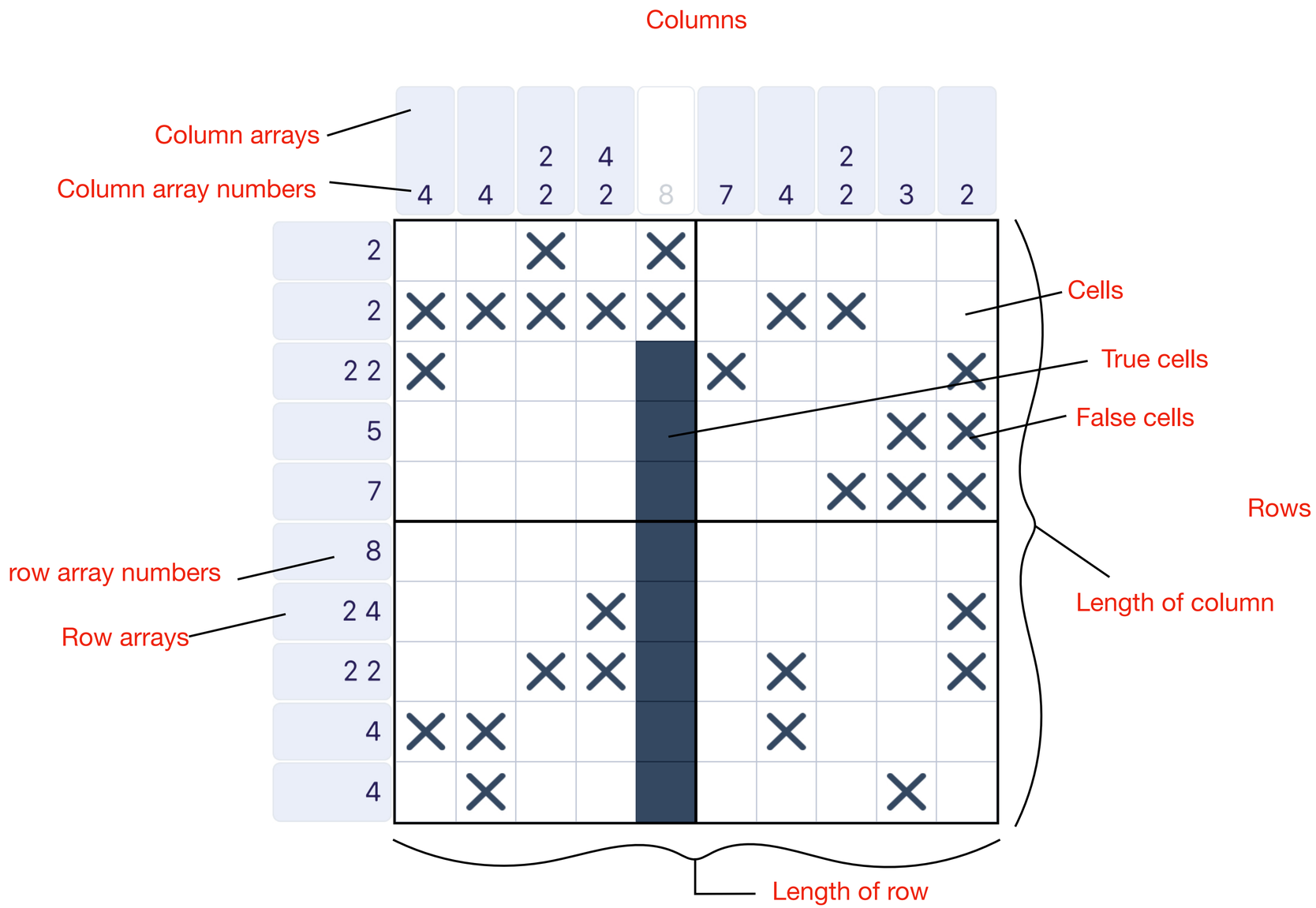
- Write a program to solve Nonogram Puzzle!
  - The number of true cells in a row/column is the sum of the numbers in that row/column array.
  - There should be one or more space (false) between true cells according to the array numbers.
  - This condition should be met for both in rows and columns.

Objectives

- Basic Goals
  1. Given an $r \times c$ problem, find a solution.
  2. Visualize the solution using GUI.
- Advanced Goals
  1. Create a random problem.
  2. Make a user-interactive game.
  3. Make colored version.

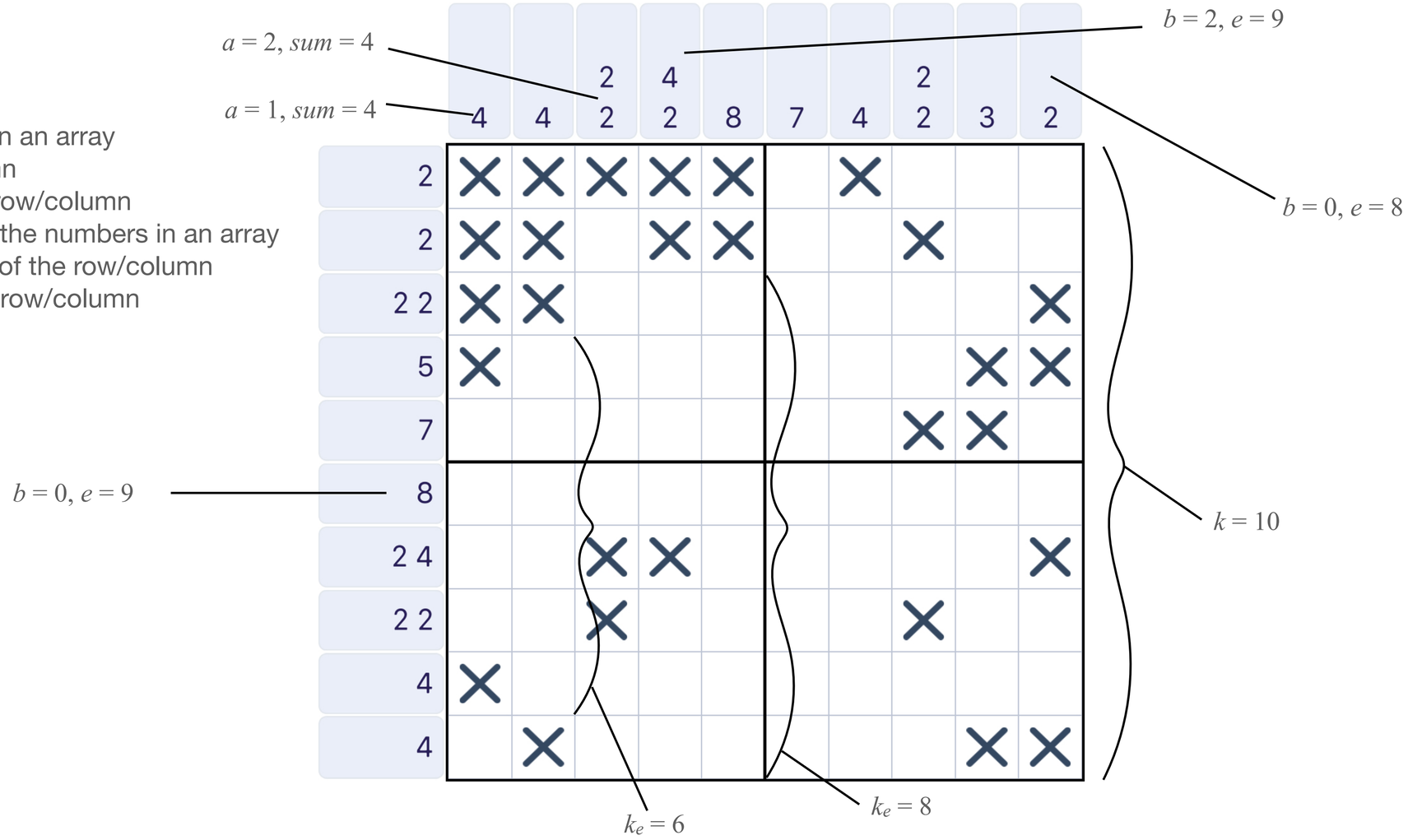


Nomenclature



Nomenclature

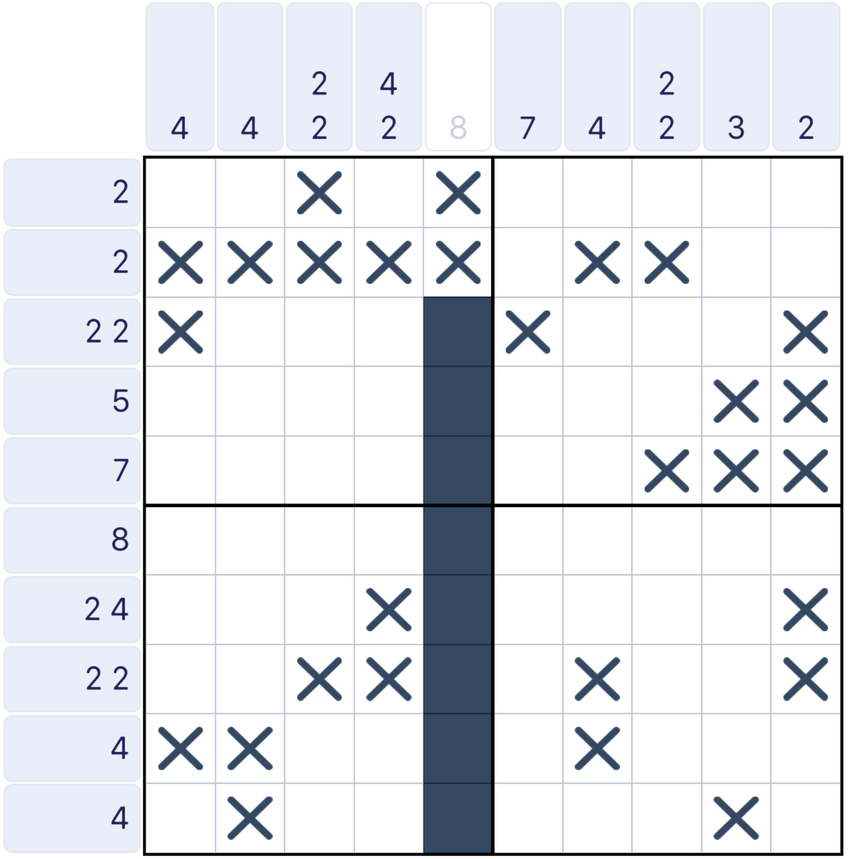
a: number of numbers in an array
k = length of row/column
k_e = effective length of row/column
sum = summation of all the numbers in an array
b = effective beginning of the row/column
e = effective end of the row/column



Problem Design

Basic Rules

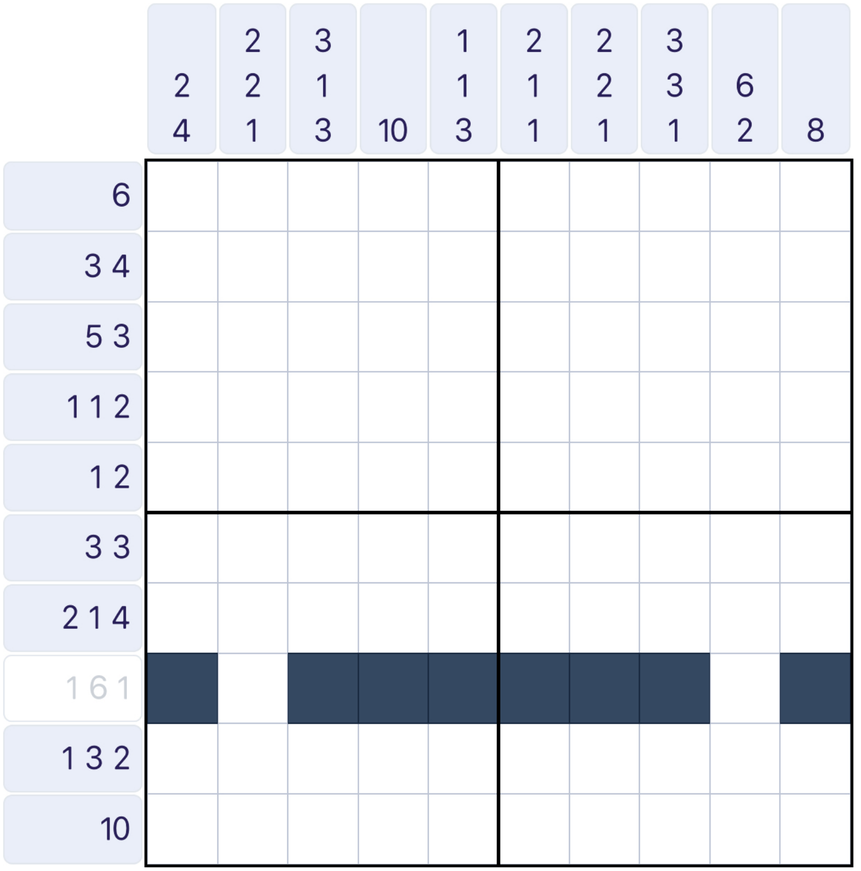
- For a row/column array of length k,
  - Numbers in an array
    - minimum: 1
    - maximum: k
  - The number of numbers in an array (a)
    - minimum: 1
    - maximum: $(k + 1) / 2$ (when $[1\ 1\ \ldots\ 1\ 1]$)
  - The sum of the numbers in an array + $(a$ - $1) <= k$



Solution Mechanisms

When the numbers are right, all true cells are found automatically.

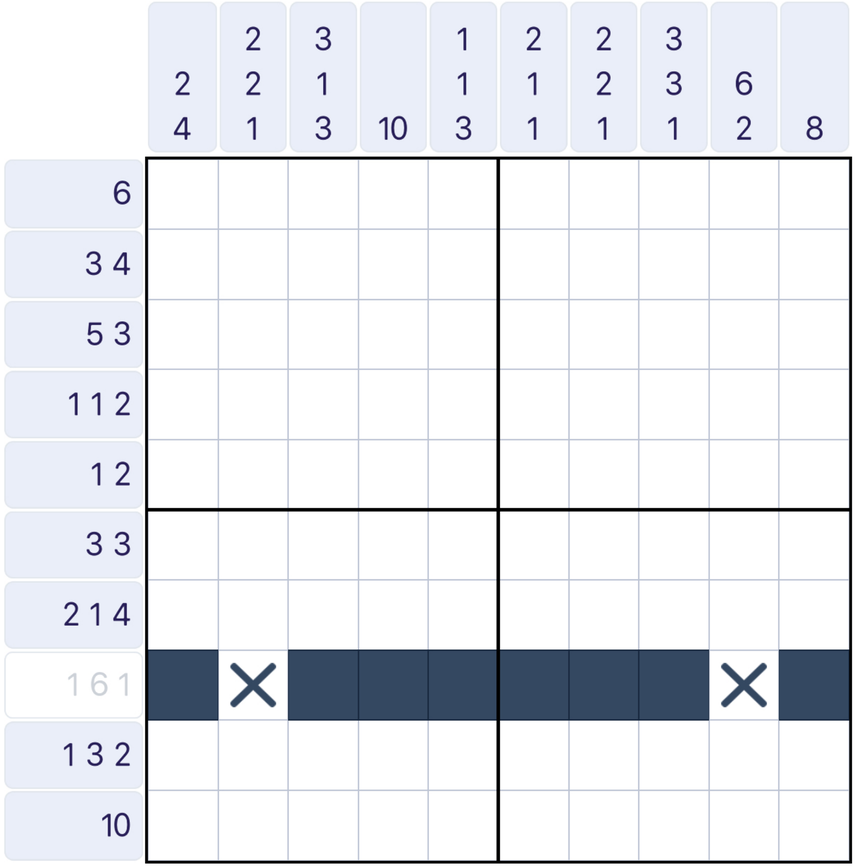
- If the sum of the numbers in an array + $(a$ - $1)$ is k, the solution is simply to fill up from the beginning. This is called *complete*.



Solution Mechanisms

When all true cells are found, everything else is false.

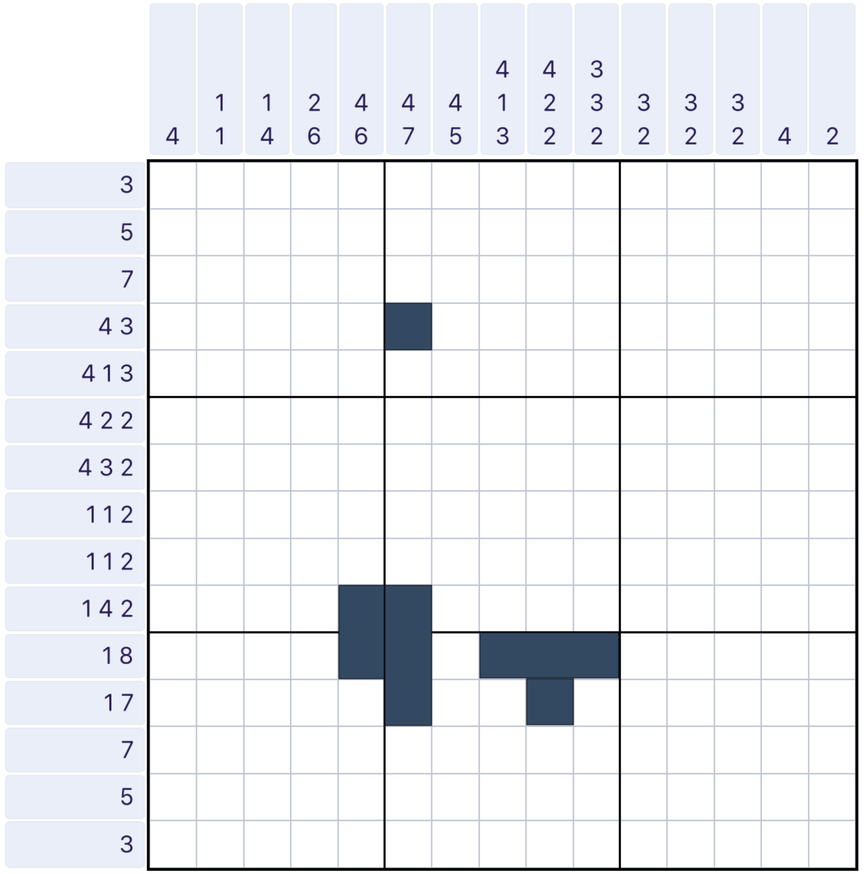
- If the number of true cells in a row/column reaches the sum of the numbers in its corresponding array, all others are false. This is called *solved*.



Solution Mechanisms

General rules on filling up the grid…

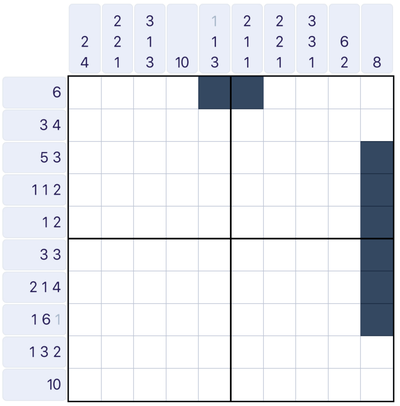
- For a number x_i in an array, you can fill up the following amount:
  - $toFill_i = 2x_i$ - (k_e - ((sum - x_i) + (a - 1))), $i = 0, 1, \ldots, a$ - 1
  - This means that it assumes all other numbers are at the very end so that we can consider the maximum range for x_i.
- If $(toFill_i > 0)$
  - Fill $[b + \sum_{n=0}^{i-1} x_n + i + x_i$ - $toFill_i$, $b + \sum_{n=0}^{i-1} x_n + i + x_i$ - 1], inclusive.
- Range of the section
  - $(p_i, q_i) = (b + \sum_{n=0}^{i-1} x_n + i$ - 1, e - ($\sum_{n=i+1}^{a-1} x_n + (a$ - (i + 1)))+ 1)
- This check needs to be done on every row/column.

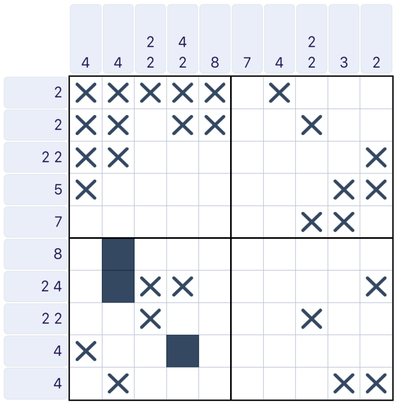


Solution Mechanisms

When the number is more than half the length…

- Suppose
  - x: current number in the array
  - p: The last cell index in the previous false cluster (1 or more consecutive false cells) (initial value : -1)
  - q: The first cell index in the next false cluster (initial value : k)
  - $k' = q - p - 1$: the number of empty cells between p and q
- If x is greater than $k' / 2$, the $(2x - k')$ cells in the middle (from $(p + 1) + k' - x$ to $(p + x)$) are true.

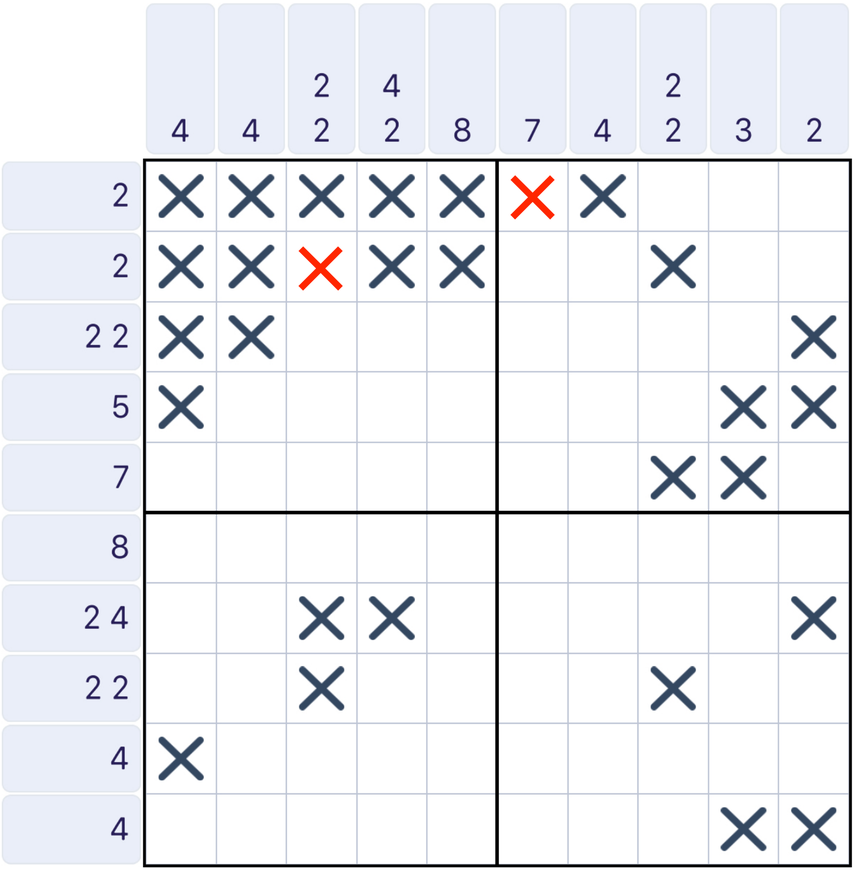




Solution Mechanisms

When there is less number of empty cells than the number…

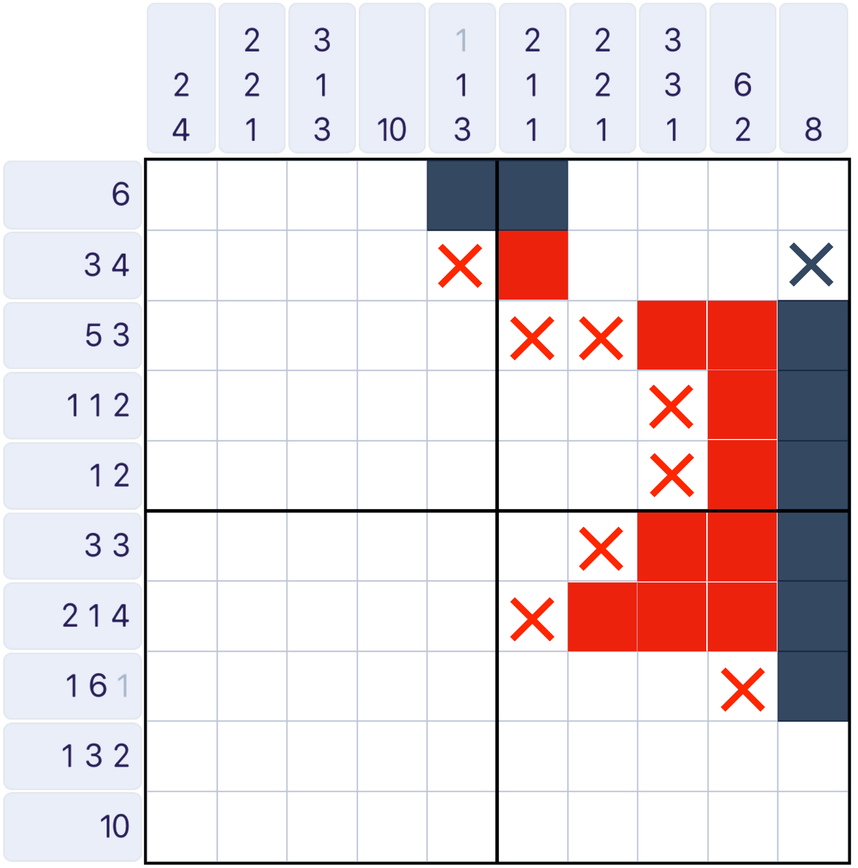
- Suppose
  - x: current number in the array
  - p: The last cell index in the previous false cluster (1 or more consecutive false cells)
  - q: The first cell index in the next false cluster
  - $k' = q - p - 1$: the number of empty cells between p and q
- If $k' < x$, all cells between p and q are false.



Solution Mechanisms

When the end cell is filled, finish filling up!

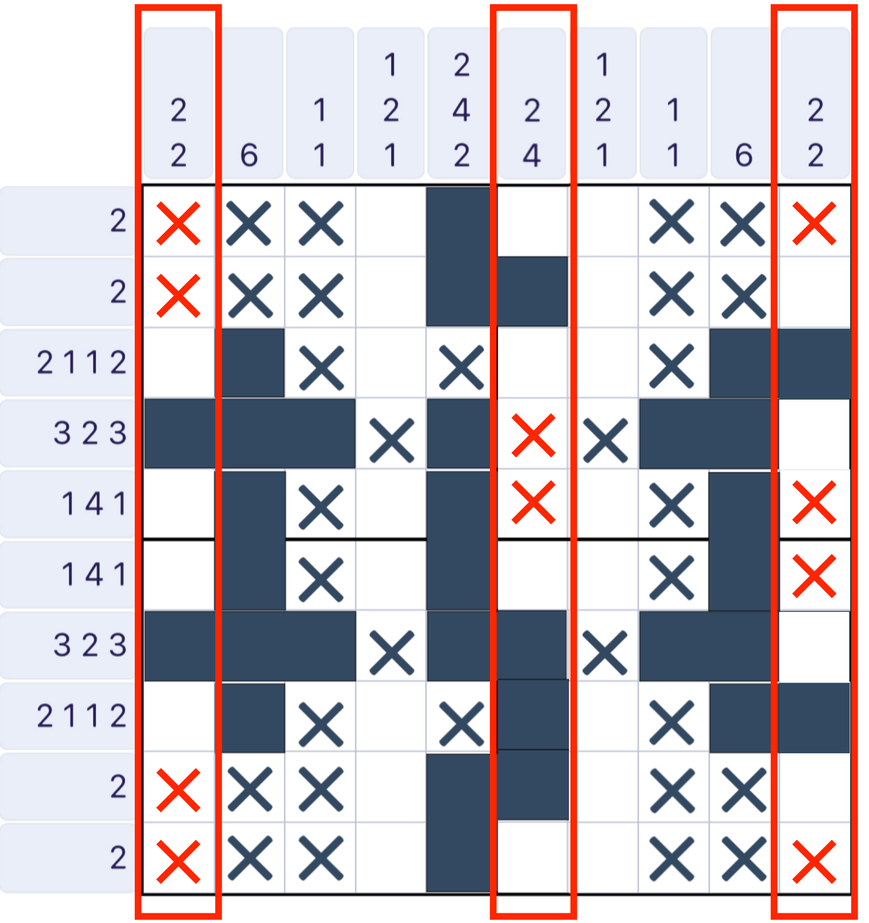
- If the first or the last cell is filled, finish filling up the cells to the corresponding number.
- Suppose
  - x: current number in the array
  - p: The last cell index in the previous false cluster (1 or more consecutive false cells)
  - q: The first cell index in the next false cluster
  - $k' = q - p - 1$: the number of empty cells between p and q
- If $p + 1$ is filled,
  - fill up from $p + 2$ to $p + x$.
  - mark $p + x + 1$ as False.
- If $q - 1$ is filled,
  - fill up from $q - x$ to $q - 2$.
  - mark $q - x - 1$ as False.



Solution Mechanisms

When number of true clusters == number of numbers

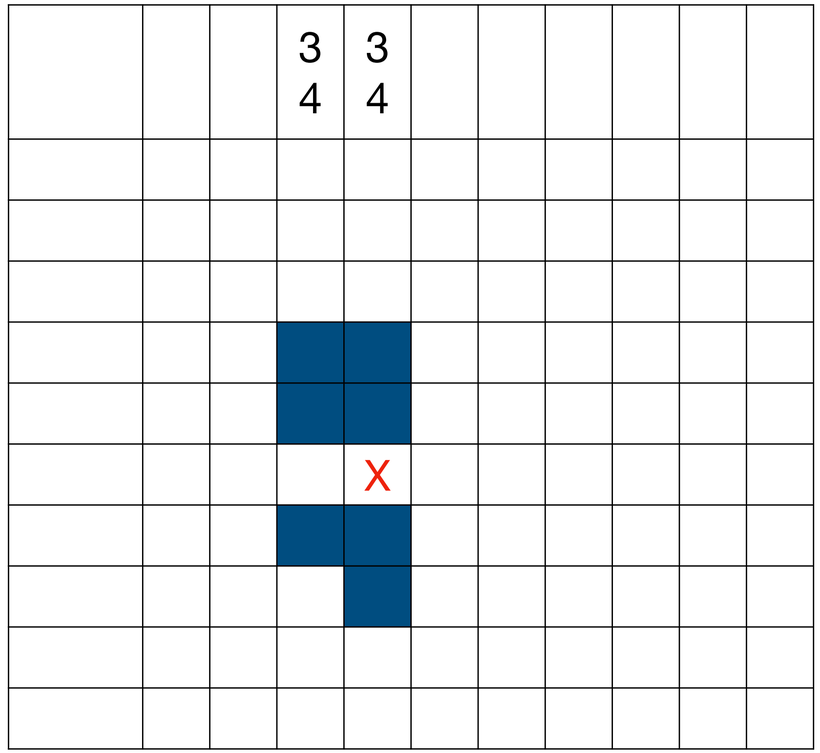
- If the number of distinct true clusters is the same as the number of numbers, make far cells false.
- Suppose
  - x: current number in the array
  - $begIdx$: The first index of the true cluster
  - $endIdx$: The last index of the true cluster
  - $remainingT$: x - length of true cluster
- For each x, make false out side of $[begIdx - remainingT, endIdx + remainingT]$ between true clusters.



Solution Mechanisms

When two true clusters are one cell apart…

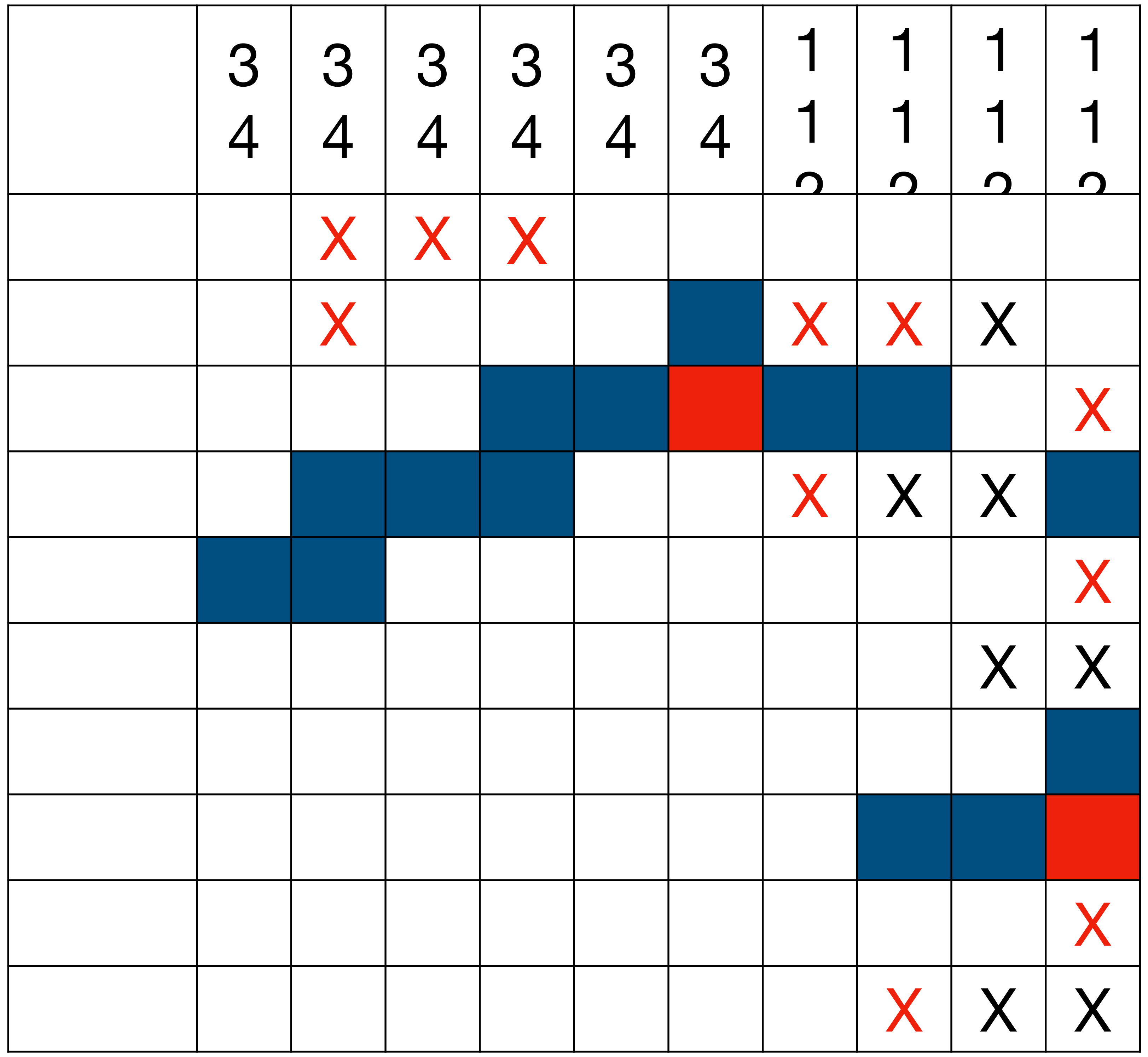
- If two true clusters are one empty cell apart and the combined length of both true clusters are larger than the maximum number in the array, make the empty cell false.



Solution Mechanisms

When the first true cluster is close to the front..

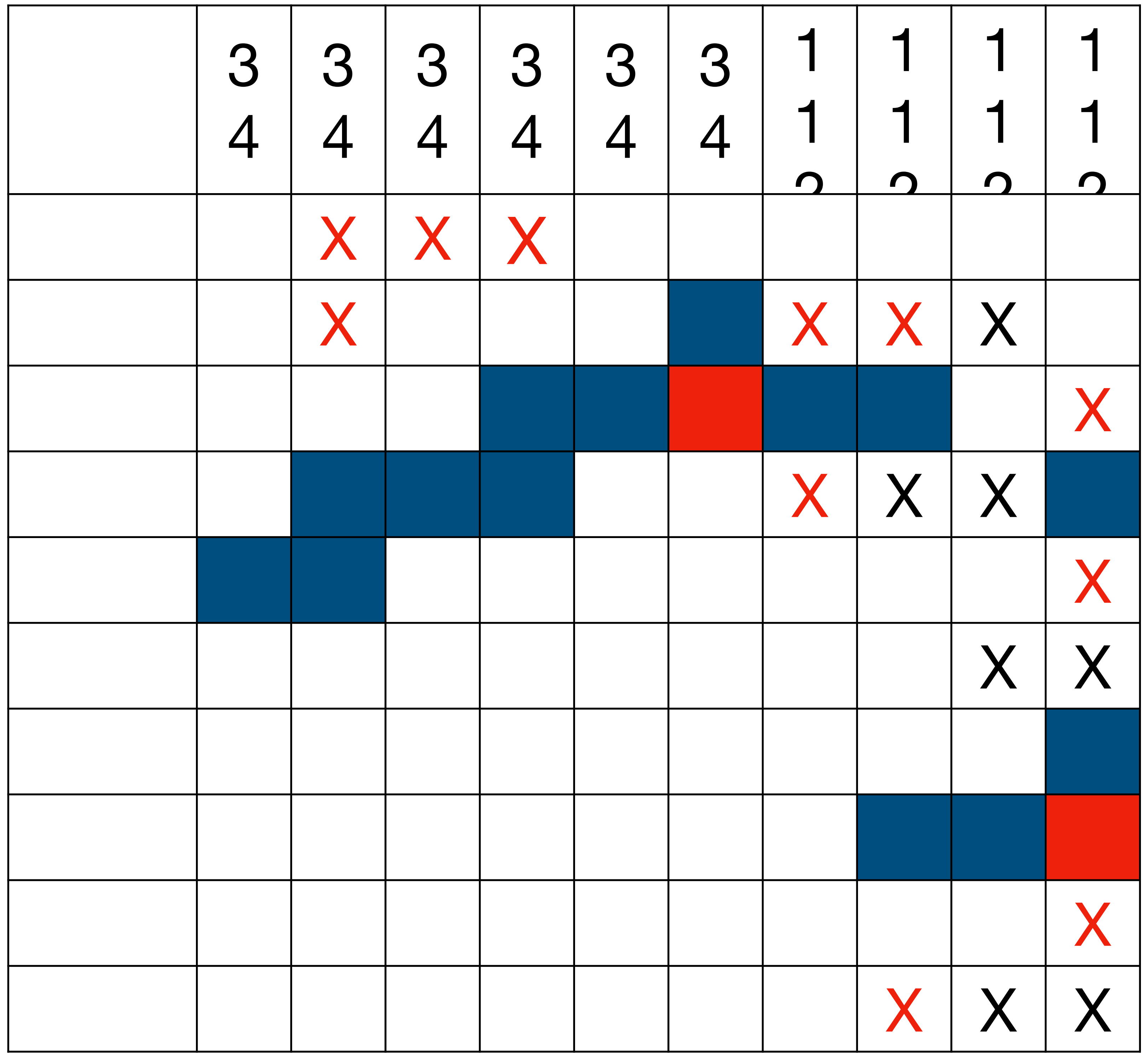
- If the first true cluster ends not farther than the first number, fill up to that length.
  - 1) the first number
  - Suppose
    - x_0: first number in the array
    - *begIdx*: The first index of the first true cluster
    - *endIdx*: The last index of the first true cluster
  - If (*endIdx* - *b*) < (x_0 - 1), fill up [*endIdx* + 1, *b* + (x_0- 1)]
- 2) the last number
- Suppose
  - x_e: last number in the array
  - *begIdx*: The first index of the last true cluster
  - *endIdx*: The last index of the last true cluster
- If (*e*-*begIdx*) < (x_e - 1), fill up [*e* - (x_0- 1), *begIdx* - 1]



Solution Mechanisms

When the first true cluster is close to the front..

- If the true cluster is just away from the beginning, make the first few false.
  - 1) the first number
  - Suppose
    - x_0: first number in the array
    - *begIdx*: The first index of the first true cluster
    - *endIdx*: The last index of the first true cluster
    - *remainingT* = x_0 - *length of the first true cluster*
  - If *begIdx* ≤ ($b + x_0$), fill up [*b*, *begIdx* - *remainingT* - 1)]
- 2) the last number
- Suppose
  - x_e: last number in the array
  - *begIdx*: The first index of the last true cluster
  - *endIdx*: The last index of the last true cluster
  - *remainingT* = x_0 - (*endIdx* - *begIdx* + 1)
- If *endIdx* ≥ (e - x_e), fill up [*endIdx* + *remainingT* + 1, *e*]



Algorithm

Method `findArraySolution(int[] curArray, int k)`

// Determine true or false of cells of the current array

`//` `k:` *the length of* `curArray`

`a =` *the number of numbers in* `curArray`

`solved = true` *if* `curArray` *is solved*

// Assertion: numbers in `curArray >= 1 && <= k`

// Assertion: `a >= 1 && <= (k+1) / 2`

// Assertion: The sum of the numbers in curArray `+ (a-1) <= k`

`if (!solved)`

`{`

`if (`*sum of the numbers in curArray* `+ (a-1) == k)` *// Initial condition automatically solves*

`{`

Fill up cells according to the curArray numbers, beginning at index `0`*.*

Make false at remaining empty cells.

`solved = true`

`}`

`else if (`*sum of the numbers in* `curArray ==` *number of* `True` *cells*`)` *// solved the array after trials*

`{`

Make false at remaining empty cells.

`solved = true`

`}`

`else if (`*sum of the numbers in curArray* `+ (a-1) == k-1)`

`{`

For each number `x` *in curArray, fill up the middle* `x-1` *cells in* `x+1` *space.*

`}`

`else`

`{`

`p =` *The last cell index in the previous false cluster or* `-1`

`q =` *The first cell index in the next false cluster or* `k`

`kp = q - p - 1` *// the number of empty cells between* `p` *and* `q`

For each number `x` *in curArray,*

`if (x > kp/2)`

The `(2x - kp)` *cells in the middle* (*from* `(p + 1) + kp - x` *to* `(p + x)`) *are* `true`*.*

`if (kp < x)`

All cells between `p` *and* `q` *are* `false`*.*

`if (p+1` *is filled*`)`

Fill up from `p + 2` *to* `p + x.`

Mark `p + x + 1` *as* `False.`

`if (q-1` *is filled*`)`

Fill up from `q - x` *to* `q - 2.`

Mark `q - x - 1` *as* `False.`

`}`

`}`