

# CERC 2016: Presentation of solutions

University of Zagreb

A: Appearance Analysis	Easy
B: Bipartite Blanket	Hard
C: Convex Contour	Easy
D: Dancing Disks	Medium
E: Easy Equation	Hard
F: Free Figurines	Easy
G: Geohash Grid	Hard
H: Hangar Hurdles	Medium
I: Invisible Integers	Hard
J: Jazz Journey	Medium
K: Key Knocking	Easy
L: Lost Logic	Medium

# Problem A

## Appearance Analysis

Submits: 108

Accepted: at least 57

First solved by: Zagreb 1  
University of Zagreb  
(Balunović, Bradac, Tomić)  
00:18:40

Author: Adrian Satja Kurdija

```
#####  
#...+#####+...#...+.#...+#####+...#...+.#  
#...+.#...+.#...+.#...+.#...+.#...+.#...+.#...+.#  
#...+.#...+.#...+.#...+.#...+.#...+.#...+.#...+.#  
#####  
#...+.#...+.#...+.#...+.#...+.#...+.#...+.#...+.#  
#...+.#...+.#...+.#...+.#...+.#...+.#...+.#...+.#  
#...+#####+...#...+.#...+#####+...#...+.#  
#####
```

Detect windows size, locate windows, then either:

- a) Compare each window pair to get a graph of matching windows, and count the components in this graph.
- b) Find a canonical rotation for each window and insert the canonical rotation into a set / hash set.

# Problem K

## Key Knocking

Submits: 224

Accepted: at least 29

First solved by: Zagreb 6  
University of Zagreb  
(Babojelić, Barišić, Šego)  
00:34:55

Author: Luka Kalinovčić

100011111000000010

3·n digit binary number.

The weight of the number is the number of groups of consecutive zeroes and ones.

1|000|11111|0000000|1|0

weight = 6

3·n digit binary number.

The weight of the number is the number of groups of consecutive zeroes and ones.

100011111000000010

3·n digit binary number.

In one move we can flip two consecutive bits.



100011110100000010

3·n digit binary number.

In one move we can flip two consecutive bits.

1|000|1111|0|1|000000|1|0

weight = 8

3·n digit binary number.

In one move we can flip two consecutive bits.

The goal is to get a number with weight  $\geq 2 \cdot n$  in at most n moves.

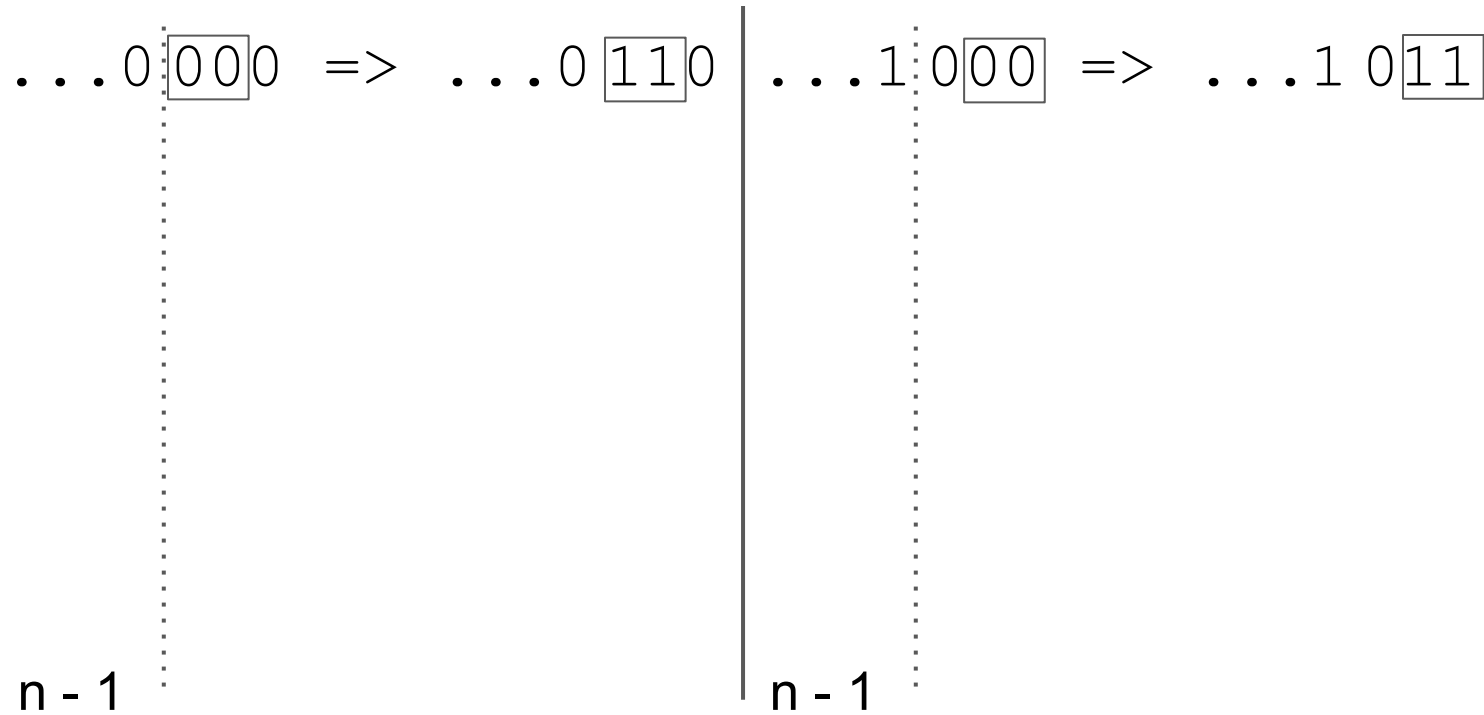
000 => 110

111 => 001

We start by analyzing the simple case where  $n = 1$ .

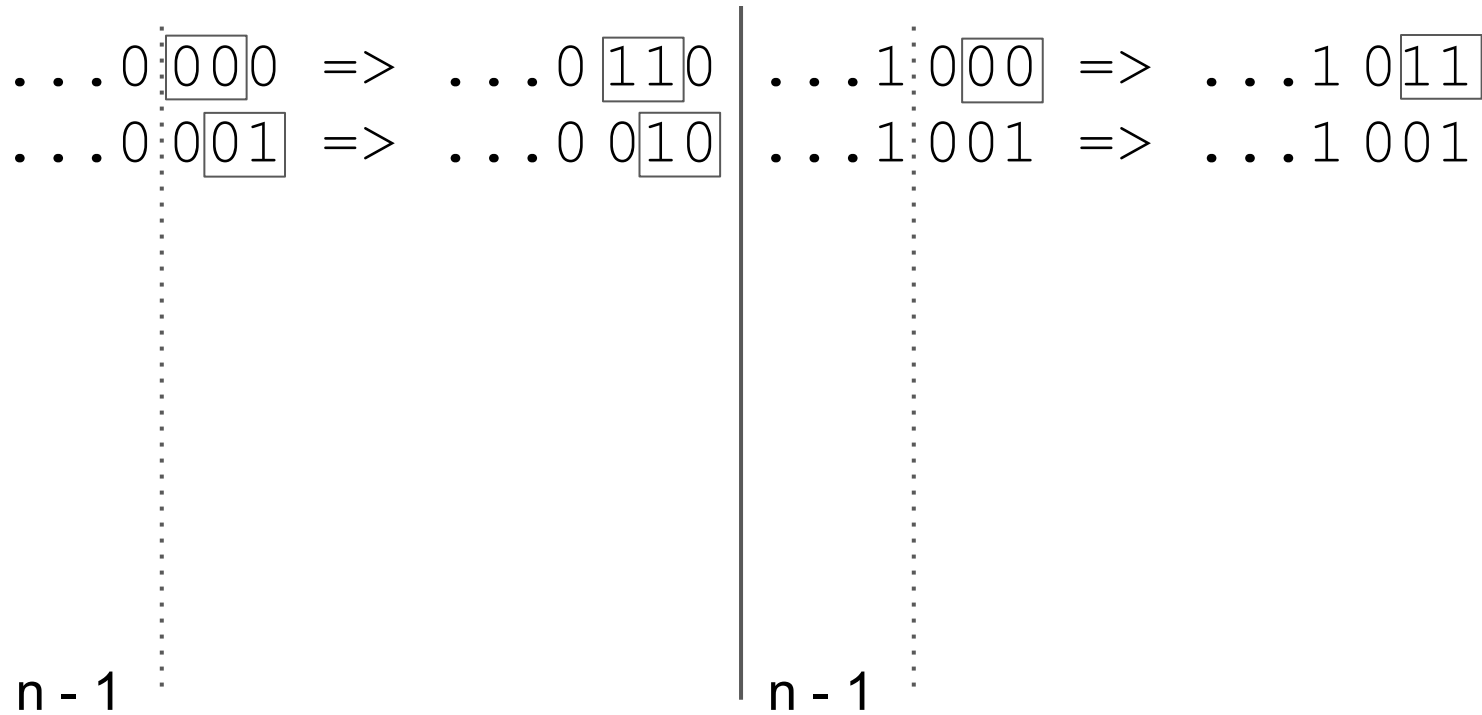
One move is always enough.

000 and 111 are the only cases with weight  $< 2$ , and any move will do.



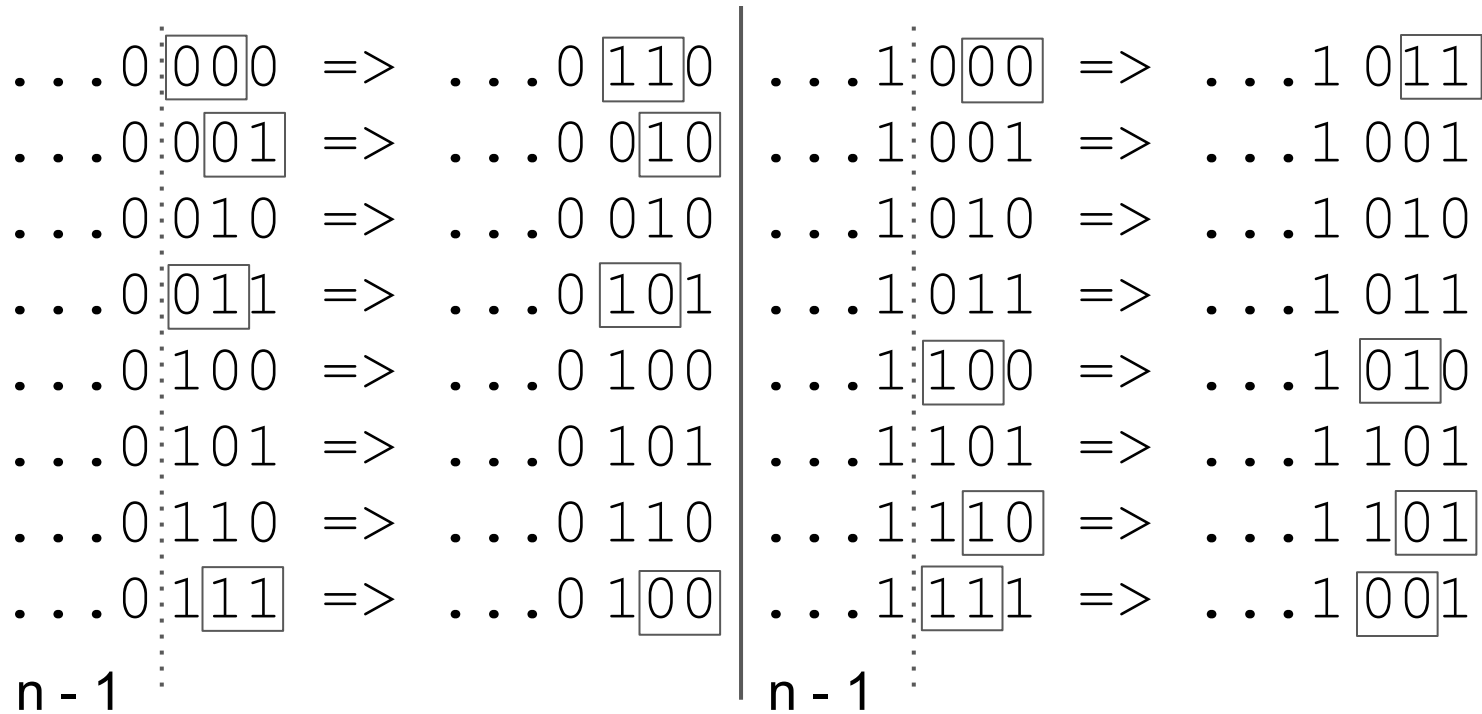
Assume we can solve any  $n - 1$  instance.

Use  $n - 1$  moves to solve the  $n - 1$  instance, then use the final move to increase the weight by at least 2 if necessary.



Assume we can solve any  $n - 1$  instance.

Use  $n - 1$  moves to solve the  $n - 1$  instance, then use the final move to increase the weight by at least 2 if necessary.



Assume we can solve any  $n - 1$  instance.

Use  $n - 1$  moves to solve the  $n - 1$  instance, then use the final move to increase the weight by at least 2 if necessary.

# Problem F

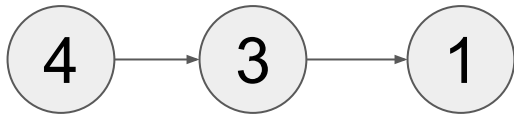
## Free Figurines

Submits: 157

Accepted: at least 50

First solved by: Jagiellonian 4  
Jagiellonian University in Krakow  
(Derbisz, Łabaj, Stokowacki)  
00:20:28

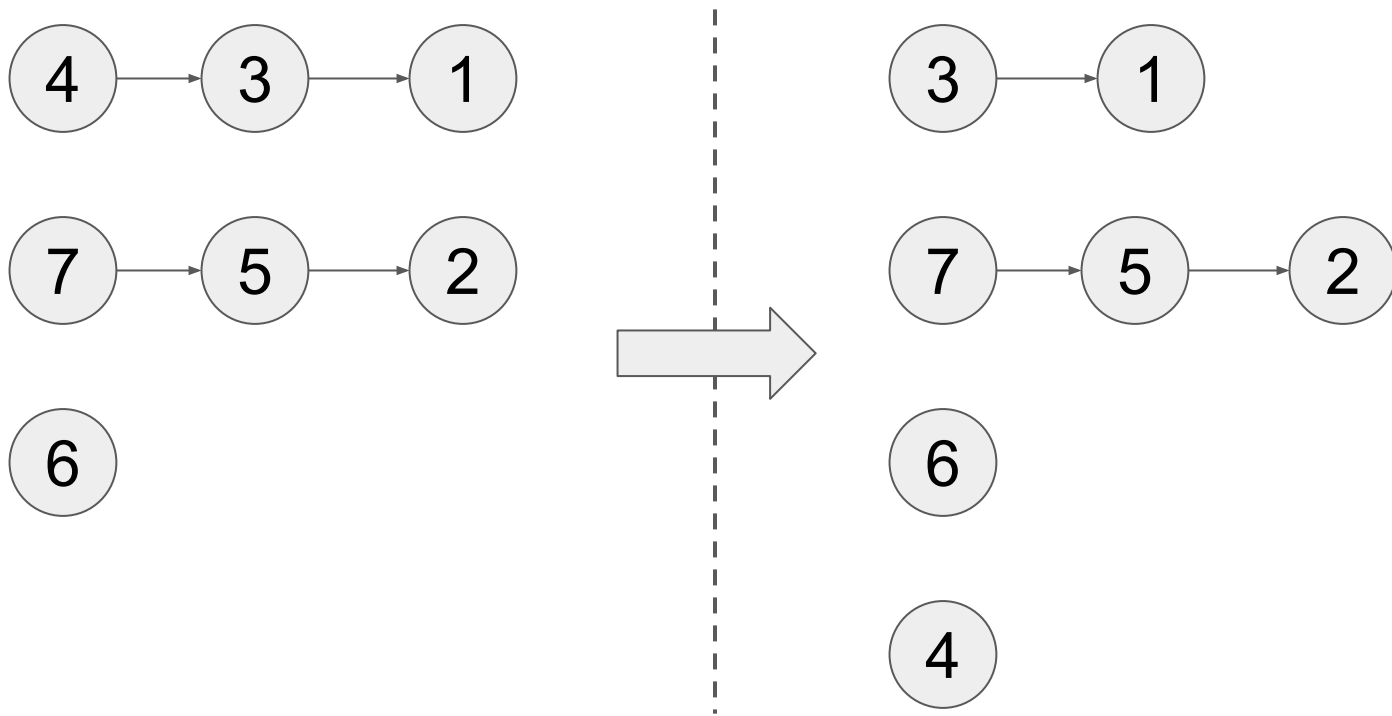
Author: Adrian Satja Kurdija



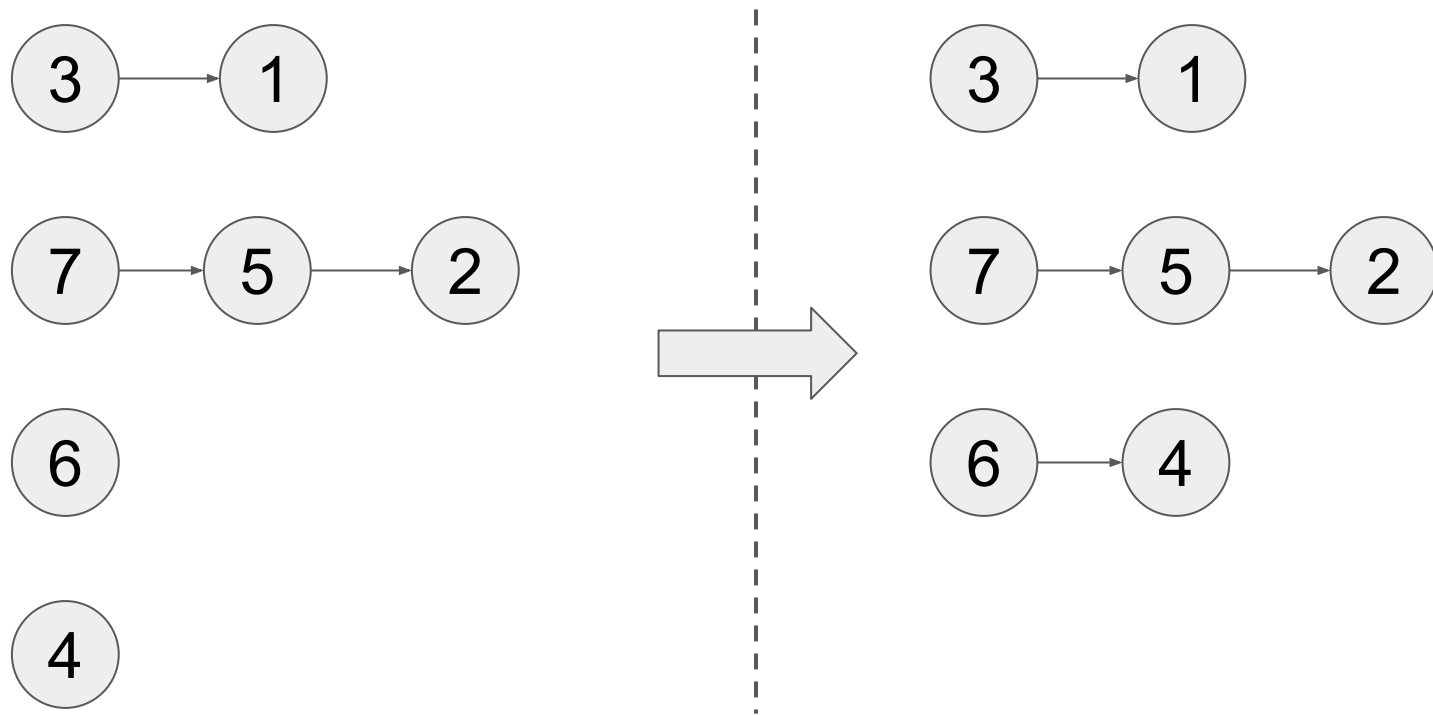
We can represent a configuration of matryoshka dolls as a set of linked lists.

A link from node  $x$  to node  $y$  indicates that doll  $x$  contains doll  $y$ .

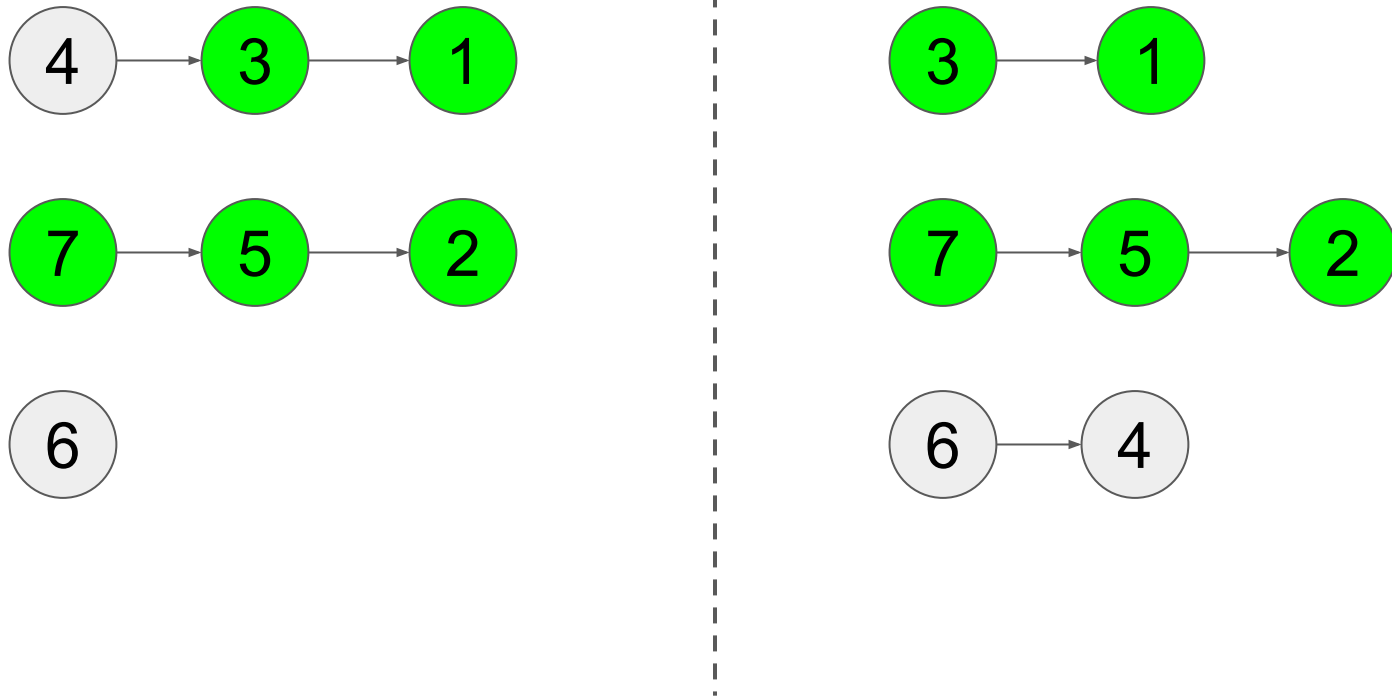




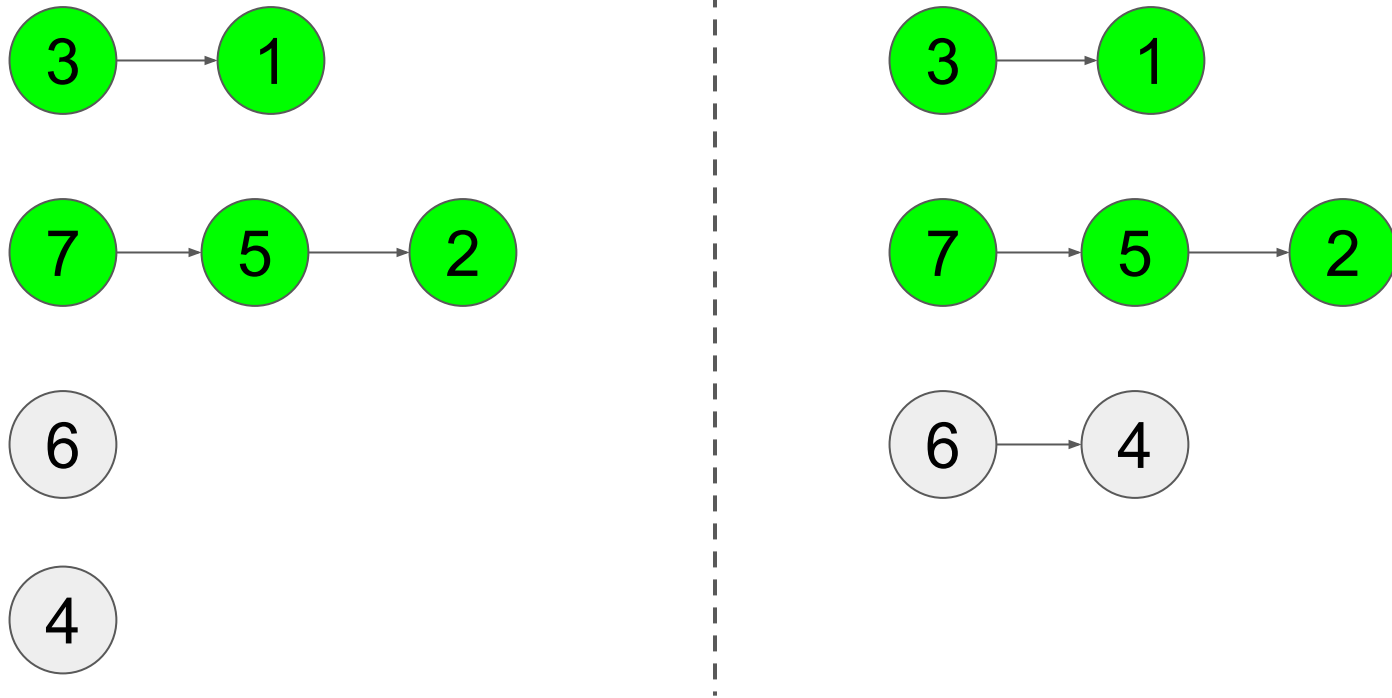
An operation of opening a non-empty free figurine and taking out the figurine inside corresponds to splitting the head of the list from the rest of the list.



An operation of opening an empty figurine and placing a free figurine inside corresponds to pushing a single node to the beginning of the list.



Key observation: In every list in the initial configuration, we can keep the longest tail that matches a tail in the target configuration. We have to split off everything else.



We are left with a set of single-element lists (free dolls) that we can simply join with the correct target list one by one.

In every move we lose one list, we can simply add the number of lists in the target configuration subtracted by the number of lists in the current configuration.

# Problem C

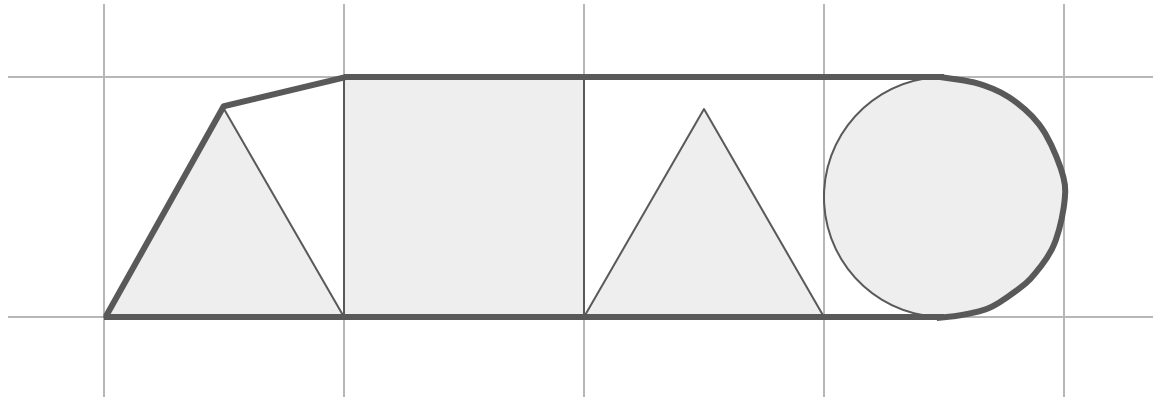
## Convex Contours

Submits: 139

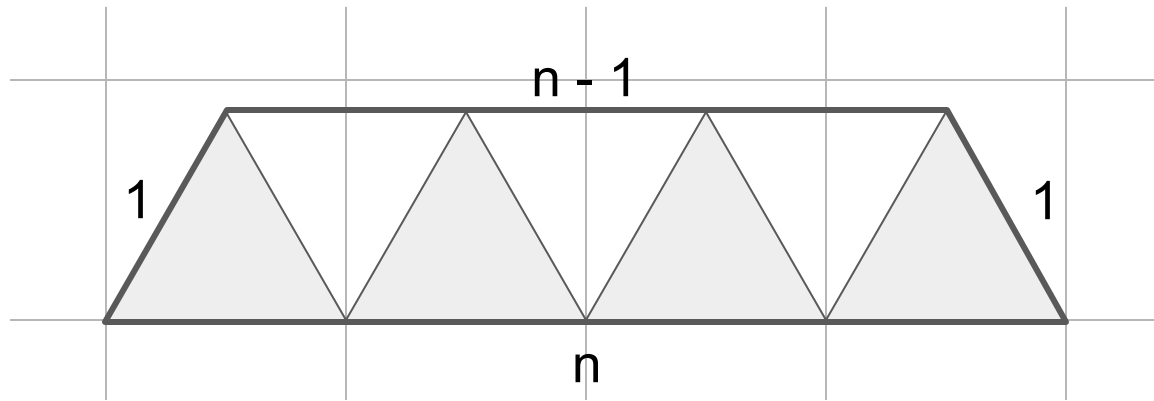
Accepted: at least 33

First solved by: UW2  
University of Warsaw  
(Dębowski, Radecki, Sommer)  
00:42:47

Author: Luka Kalinovič

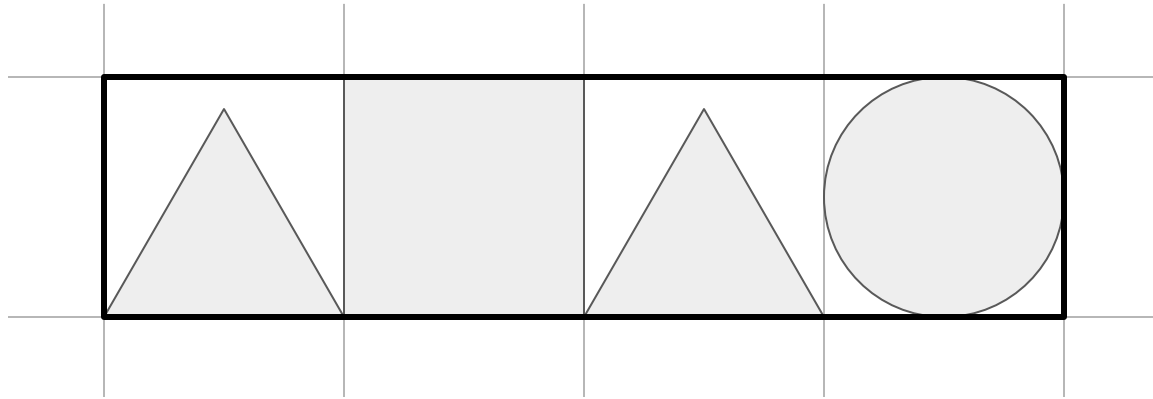


Given a row of  $n$  simple shapes, find the length of convex contour enclosing them.



Special case: Triangles only

Just output  $2 \cdot n + 1$

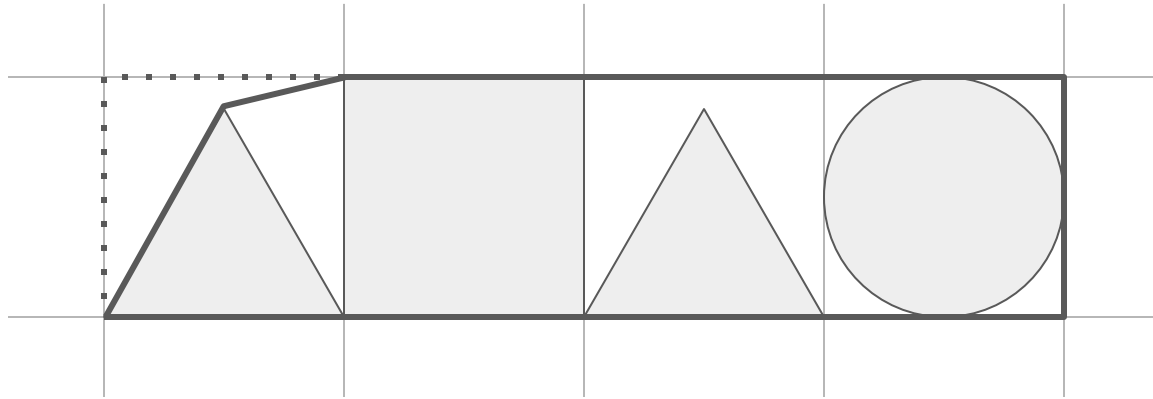


Otherwise, contour touches both upper and lower line.

Algorithm:

1) Start with  $n$  by 1 rectangle.

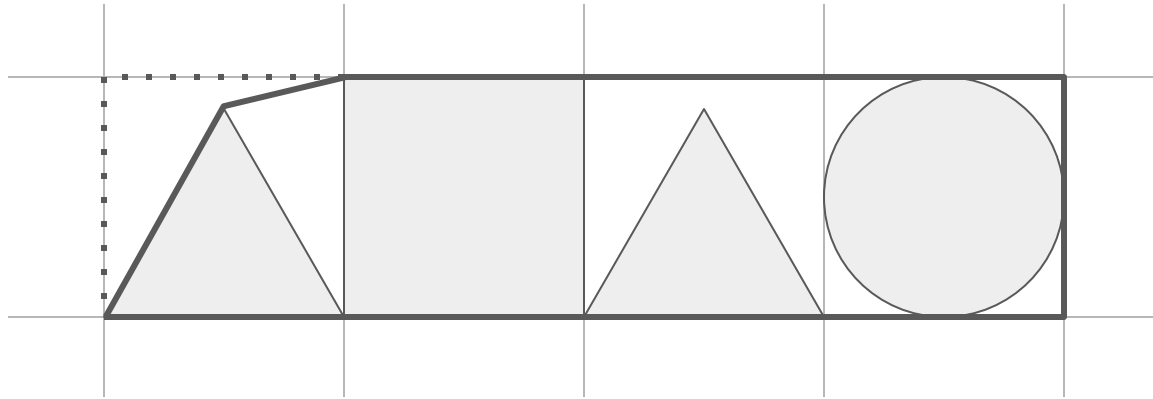




Otherwise, contour touches both upper and lower line.

Algorithm:

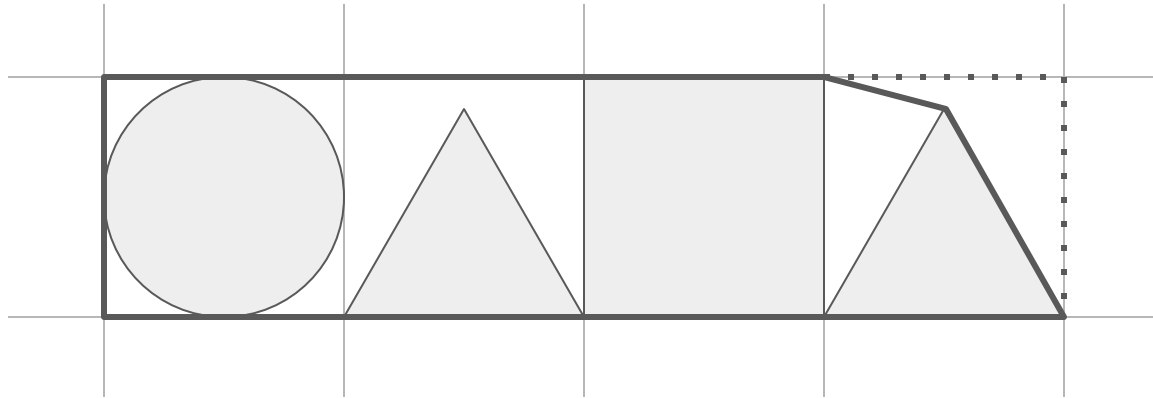
- 1) Start with  $n$  by 1 rectangle.
- 2) Adjust the left part of the contour.



Otherwise, contour touches both upper and lower line.

Algorithm:

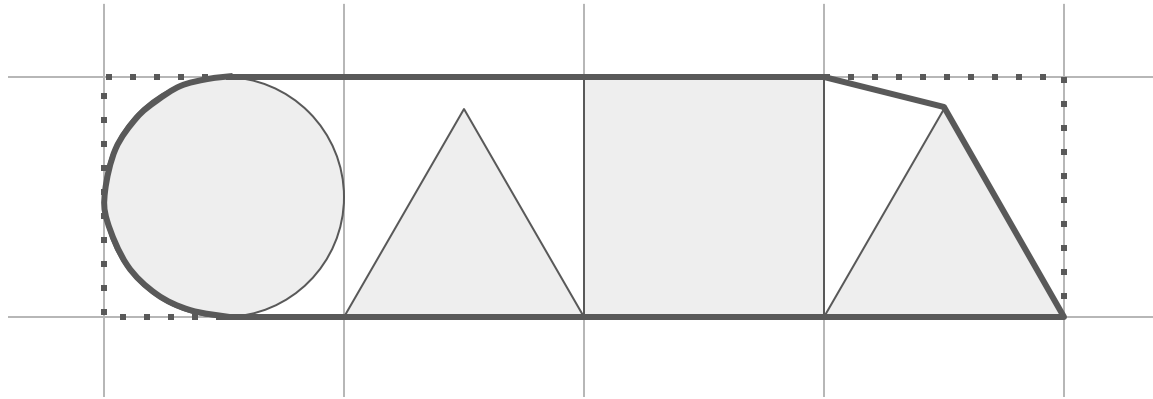
- 1) Start with  $n$  by 1 rectangle.
- 2) Adjust the left part of the contour.
- 3) Adjust the right part of the contour.



Otherwise, contour touches both upper and lower line.

Algorithm:

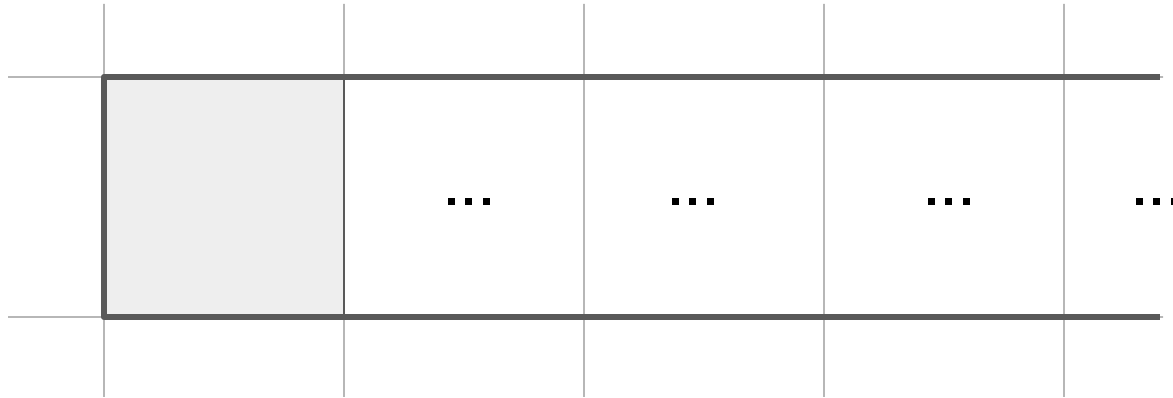
- 1) Start with  $n$  by 1 rectangle.
- 2) Adjust the left part of the contour.
- 3) Adjust the right part of the contour.
  - a) By reversing the sequence, and
  - b) Repeating 2)



Otherwise, contour touches both upper and lower line.

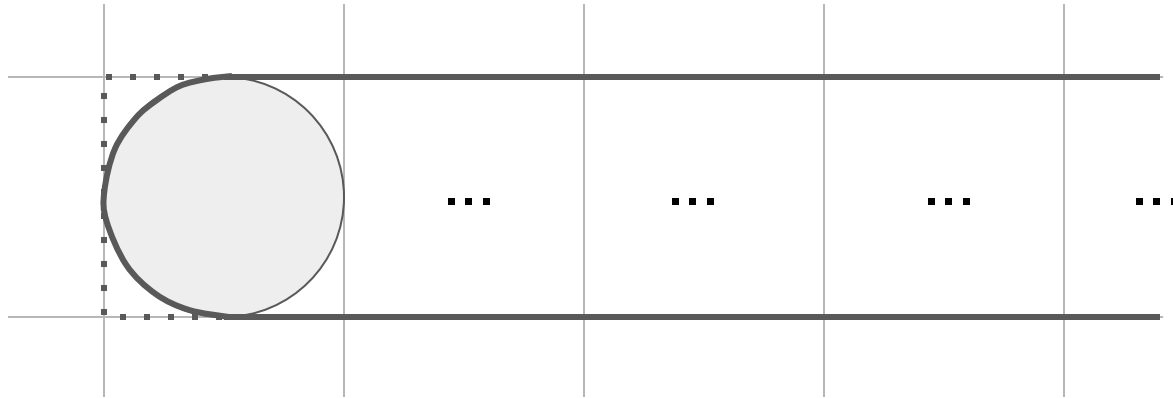
Algorithm:

- 1) Start with  $n$  by 1 rectangle.
- 2) Adjust the left part of the contour.
- 3) Adjust the right part of the contour.
  - a) By reversing the sequence, and
  - b) Repeating 2)



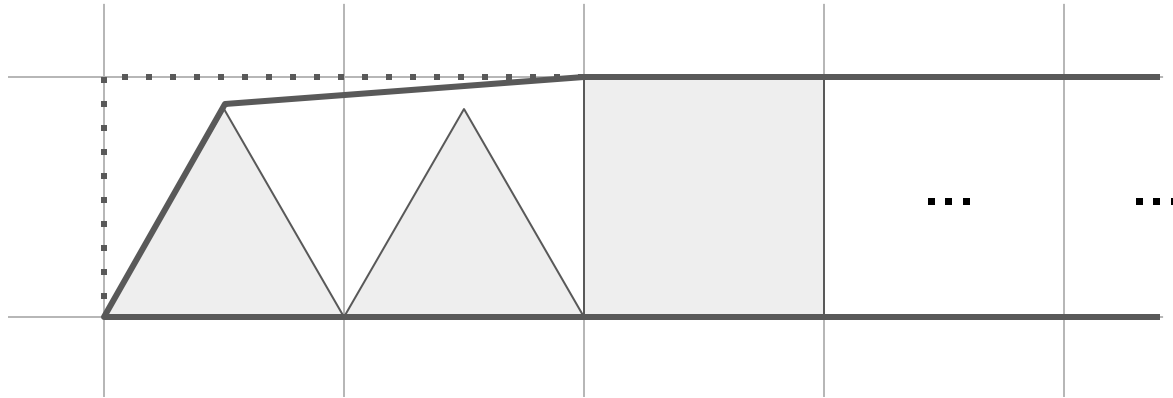
To adjust the left part of the contour, we analyze a few cases.

1) Sequence starts with a square.



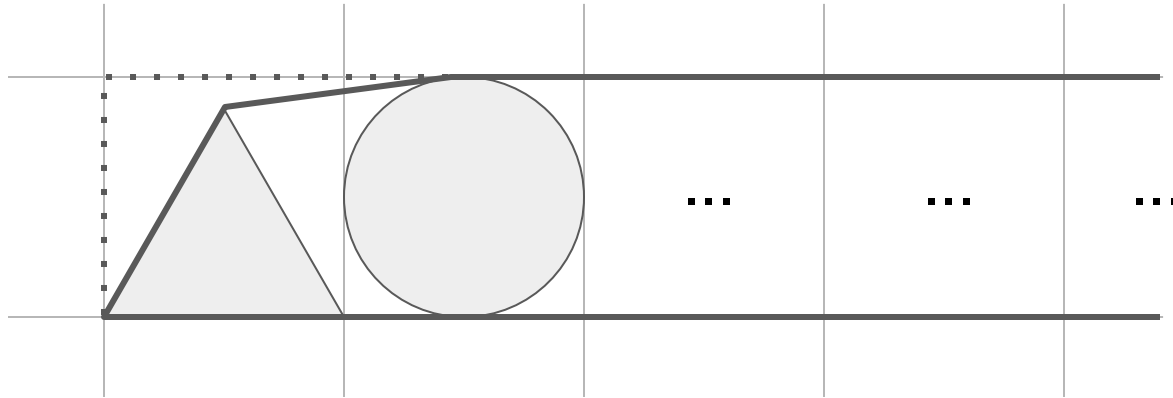
To adjust the left part of the contour, we analyze a few cases.

- 1) Sequence starts with a square.
- 2) Sequence starts with a circle.



To adjust the left part of the contour, we analyze a few cases.

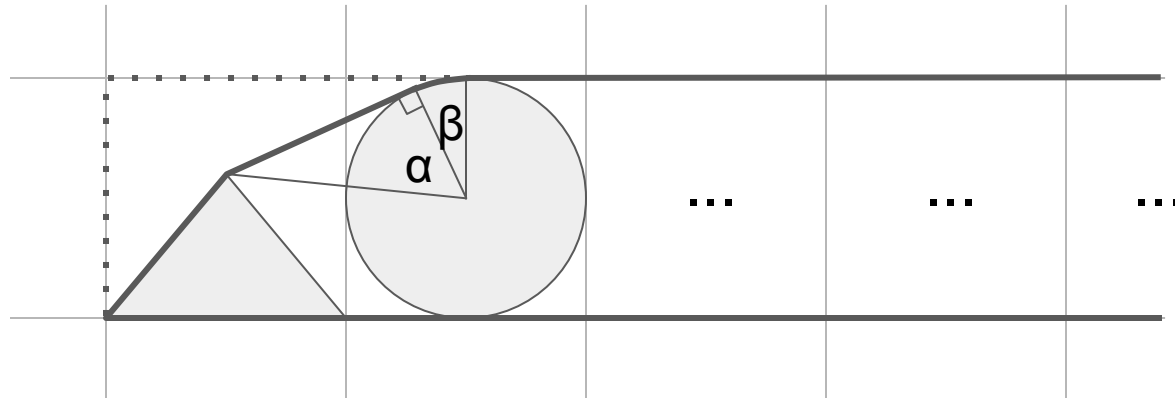
- 1) Sequence starts with a square.
- 2) Sequence starts with a circle.
- 3) Sequence starts with triangles followed by a square.



To adjust the left part of the contour, we analyze a few cases.

- 1) Sequence starts with a square.
- 2) Sequence starts with a circle.
- 3) Sequence starts with triangles followed by a square.
- 4) Sequence starts with triangles followed by a circle.





To adjust the left part of the contour, we analyze a few cases.

- 1) Sequence starts with a square.
- 2) Sequence starts with a circle.
- 3) Sequence starts with triangles followed by a square.
- 4) Sequence starts with triangles followed by a circle.

# Problem L

## Lost Logic

Submits: 26

Accepted: at least 3

First solved by: UW1

University of Warsaw

(Nadara, Smulewicz, Sokołowski)

02:27:14

Author: Ivan Katanić, Gustav Matula

x1	x2	x3	x4	x5	x6	x7	x8	x9
1	0	1	0	0	1	1	0	1
0	0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0	1

$x1 \rightarrow x2$   
 $!x3 \rightarrow x9$   
 $x1 \rightarrow !x3$   
 $\dots$

Given a set of boolean variables, we need to find a set of implications such that exactly three given assignments evaluate to true, and every other assignment evaluates to false.

x1	x2	x3	x4	x5	x6	x7	x8	x9
1	0	1	0	0	1	1	0	1
0	0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0	1

x2 -> !x2  
x4 -> !x4  
!x7 -> x7

**Step one: fix constants.**

x2 == 0

x4 == 0

x7 == 1

x1	x2	x3	x4	x5	x6	x7	x8	x9
1	0	1	0	0	1	1	0	1
0	0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0	1

$x2 \rightarrow !x2$   
 $x4 \rightarrow !x4$   
 $!x7 \rightarrow x7$   
 $x1 \rightarrow x9$   
 $x9 \rightarrow x1$   
 $x3 \rightarrow x6$   
 $x6 \rightarrow x3$

**Step two: find equivalent variables.**

$x1 == x9, x3 == x6$

We can fix two variables to always be equal using two implications.

x1	x2	x3	x4	x5	x6	x7	x8	x9
1	0	1	0	0	1	1	0	1
0	0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0	1

$$x2 \rightarrow !x2$$

$$x4 \rightarrow !x4$$

$$!x7 \rightarrow x7$$

$$x1 \rightarrow x9$$

$$x9 \rightarrow x1$$

$$x3 \rightarrow x6$$

$$x6 \rightarrow x3$$

$$x1 \rightarrow !x8$$

$$!x8 \rightarrow x1$$

$$x3 \rightarrow !x5$$

$$!x5 \rightarrow x3$$

Step two: find equivalent variables.

$$x1 == x9, \quad x3 == x6$$

We can fix two variables to always be equal using two implications.

We can do the same for two variables where one is a negation of the other.

$$x1 != x8, \quad x3 != x5$$

x1	x2	x3	x4	x5	x6	x7	x8	x9
1	0	1	0	0	1	1	0	1
0	0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0	1

We are left with two or three free variables, everything else is fixed once we assign them.

If there are two free variables, we have four valid assignments left. We just need to find the right last implication to get rid of the fourth assignment.

$x_2 \rightarrow !x_2$   
 $x_4 \rightarrow !x_4$   
 $!x_7 \rightarrow x_7$   
 $x_1 \rightarrow x_9$   
 $x_9 \rightarrow x_1$   
 $x_3 \rightarrow x_6$   
 $x_6 \rightarrow x_3$   
 $x_1 \rightarrow !x_8$   
 $!x_8 \rightarrow x_1$   
 $x_3 \rightarrow !x_5$   
 $!x_5 \rightarrow x_3$   
 $!x_1 \rightarrow x_3$

x1	x2	x3
1	1	1
0	0	1
1	0	0

If there are three free variables, we have eight valid assignments left.

However, it is impossible to find a set of implications that eliminates five out of eight, no matter what the configuration is.



# Problem J

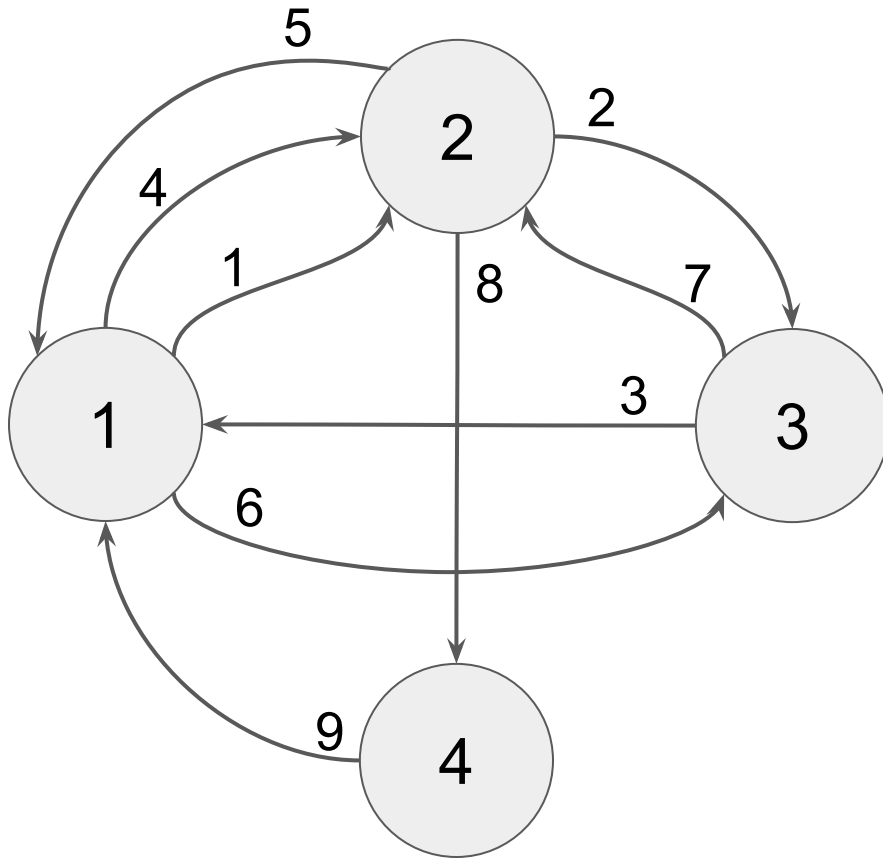
## Jazz Journey

Submits: 82

Accepted: at least 13

First solved by: UW2  
University of Warsaw  
(Dębowski, Radecki, Sommer)  
01:05:25

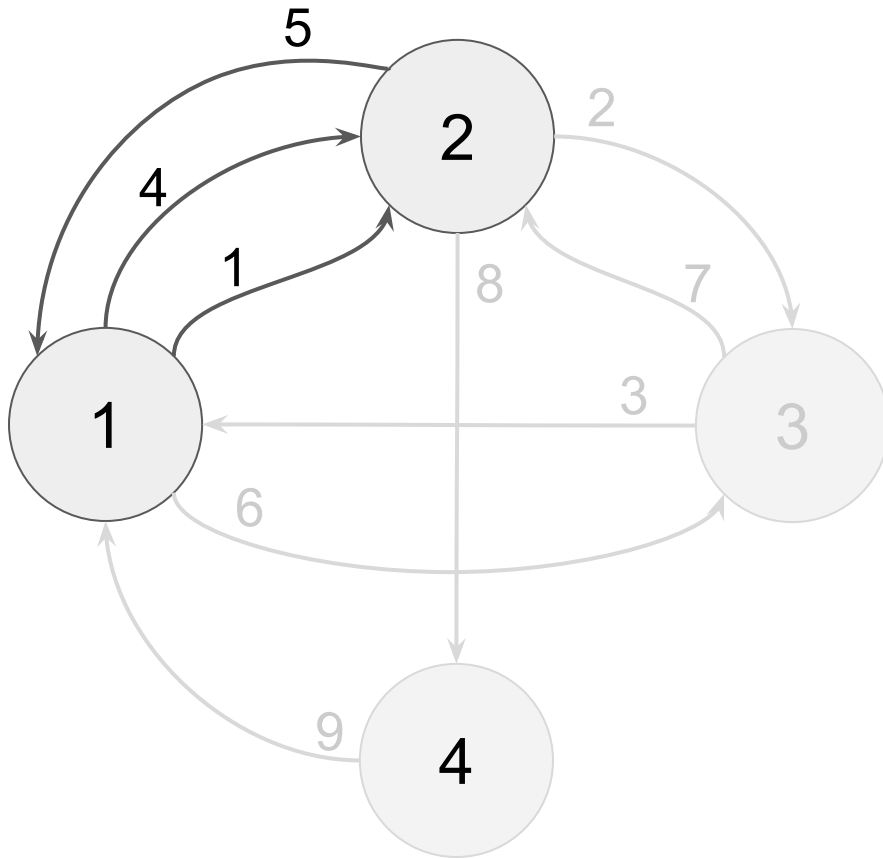
Author: Ivan Katanić



1	2	O	20
1	2	R	10
1	3	R	1
3	1	R	10
2	3	O	5
2	3	R	20
3	2	O	5
2	4	O	10
4	1	O	10

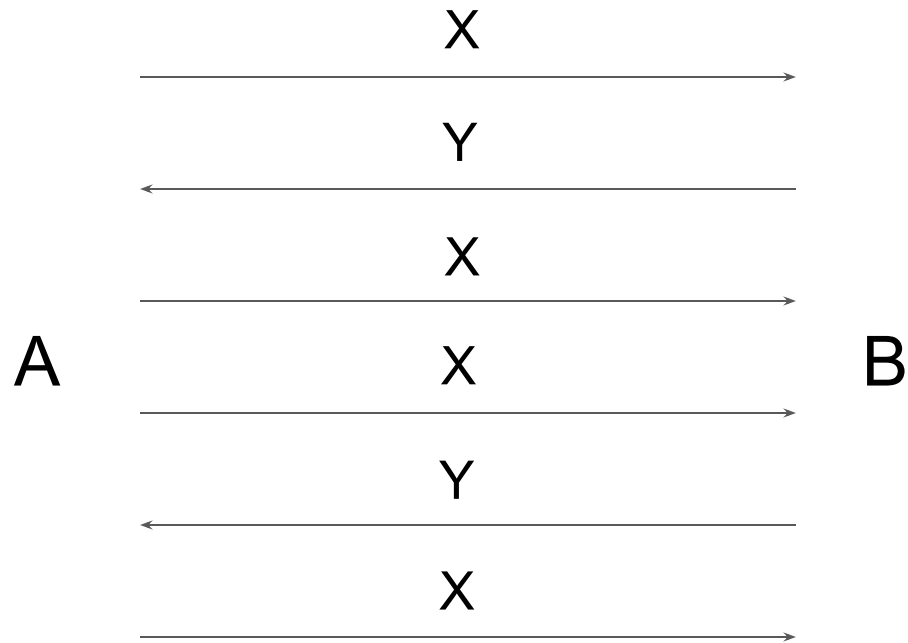
One-way ticket  $a b$ : can be used to fly from  $a$  to  $b$  once (but not in the opposite direction).

Round trip ticket  $a b$ : can be used to fly once from  $a$  to  $b$ , and once from  $b$  to  $a$ . The return segment (from  $b$  to  $a$ ) does not need to be used.



1	2	O	20
1	2	R	10
1	3	R	1
3	1	R	10
2	3	O	5
2	3	R	20
3	2	O	5
2	4	O	10
4	1	O	10

Observation: We can look at each pair of cities independently from one another.



We can encode flights between a pair of cities as a string of letters X and Y, where X represents a flight from A to B, and Y represents a flight from B to A.

XYXXYYXXXYYYXXXXYXYY

Let  $c_x$  be the cheapest flight to get from A to B.

Let  $c_y$  be the cheapest flight to get from B to A.

We can also use the first leg of a round trip here if it has a lower cost than one-way trip.

Let  $c_{xY}$  be the cheapest round trip from A to B to A.

Let  $c_{yX}$  be the cheapest round trip from B to A to B.

The problem of buying flight tickets now becomes a problem of eliminating characters from the string for a given cost:

$c_x$  to eliminate a single character X,  $c_{xY}$  to eliminate character X and character Y that appears anywhere after it, etc.

XYXXYYXXXYYYXXXXYXYY

We can choose A and B such that  $c_{XY} \leq c_{YX}$  (swap A and B if needed).

Now there are just three cases to consider:

1)  $c_X + c_Y \leq c_{XY} \leq c_{YX}$

The best strategy is to buy one-way tickets only, i.e. only eliminate single characters from the string for a cost of  $c_X$  or  $c_Y$  per character.

XYXXYYXXXYYYXXXXYXYY

$$2) c_{XY} \leq c_X + c_Y \leq c_{YX}$$

The best strategy is to eliminate as many XY pairs as possible before we start eliminating single characters.

We use a greedy algorithm to find the maximum number of XY pairs we can eliminate. Pick first X, and first Y that follows and eliminate for a cost of

$c_{XY}$ .

Eliminate what's left for a cost of  $c_X$  or  $c_Y$  per character.

XYXXYYXXXYYYXXXXYXYY

3)  $c_{XY} \leq c_{YX} \leq c_X + c_Y$

The best strategy is to eliminate as many XY pairs as possible, and then as many YX pairs as possible.

As before, we use a greedy algorithm to find the maximum number of XY pairs we can eliminate. We are left with a string that looks like YY...YXX..X. As long as there are both Y's and X's in the string, eliminate YX pairs for a cost of  $c_{YX}$ . Eliminate what's left for a cost of  $c_X$  or  $c_Y$  per character.



# Problem H

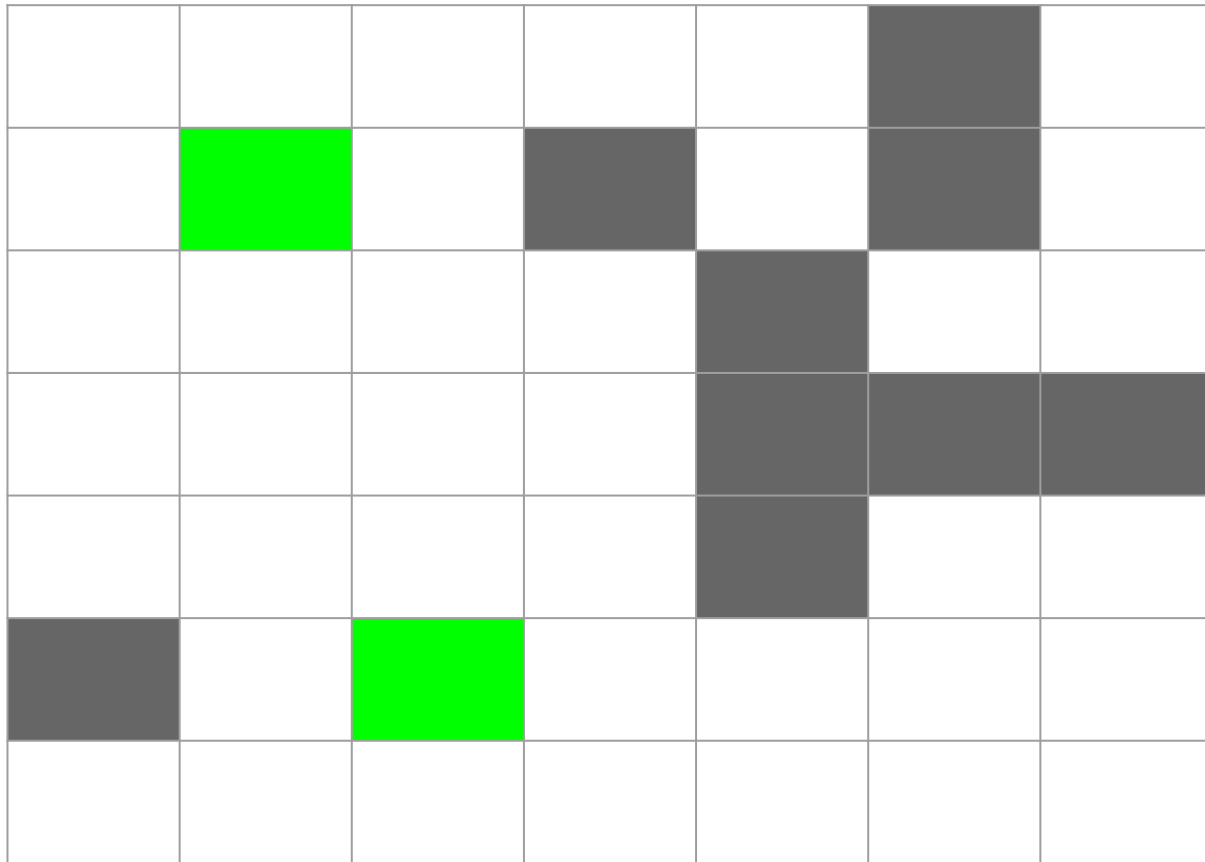
## Hangar Hurdles

Submits: 68

Accepted: at least 17

First solved by: UWr1  
University of Wroclaw  
(Łowicki, Michalak, Syposz)  
01:36:45

Author: Luka Kalinovčić



Given a bunch of cell pairs, how big of a crate can we move freely from one cell to the other?

1	1	1	1	1	0	1
1	3	1	0	1	0	1
1	3	1	1	0	1	1
1	3	3	1	0	0	0
1	1	3	1	0	1	1
0	1	3	1	1	1	1
1	1	1	1	1	1	1

First step: For each cell, find the size of the biggest crate that can be centered at that cell.

Dynamic programming or flood-fill starting from blocked cells.

50	5	25	0
20	0	60	70
90	15	30	0
80	0	65	45

We can generalize the problem:

In a matrix of numbers find a path from one cell to the other such that the smallest number on the path is as high as possible.

50	5	25	0
20	0	60	70
90	15	30	0
80	0	65	45

We sort cells by numbers decreasingly, and we build the tree by adding nodes in order, and connecting subtrees of neighbouring cells.

50	5	25	0
20	0	60	70
90	15	30	0
80	0	65	45

We sort cells by numbers decreasingly, and we build the tree by adding nodes in order, and connecting subtrees of neighbouring cells.

50	5	25	0
20	0	60	70
90	15	30	0
80	0	65	45

We sort cells by numbers decreasingly, and we build the tree by adding nodes in order, and connecting subtrees of neighbouring cells.

50	5	25	0
20	0	60	70
90	15	30	0
80	0	65	45

We sort cells by numbers decreasingly, and we build the tree by adding nodes in order, and connecting subtrees of neighbouring cells.



50	5	25	0
20	0	60	70
90	15	30	0
80	0	65	45

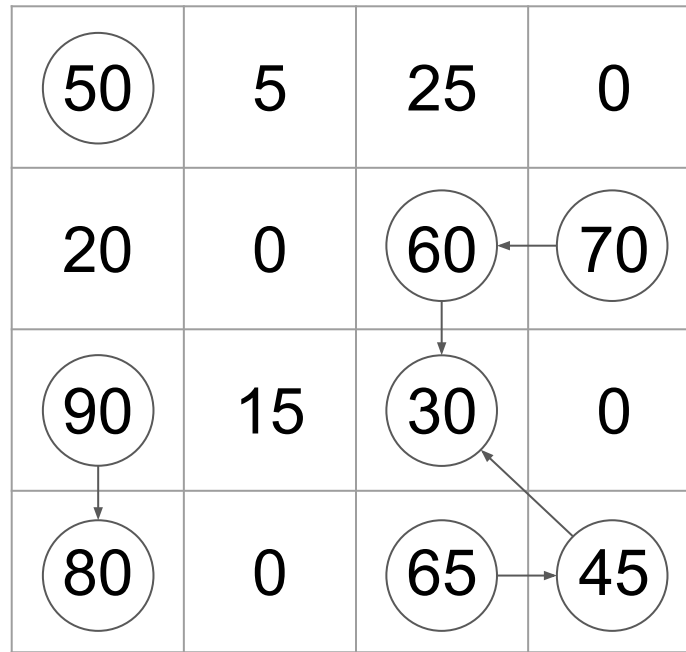
We sort cells by numbers decreasingly, and we build the tree by adding nodes in order, and connecting subtrees of neighbouring cells.

50	5	25	0
20	0	60	70
90	15	30	0
80	0	65	45

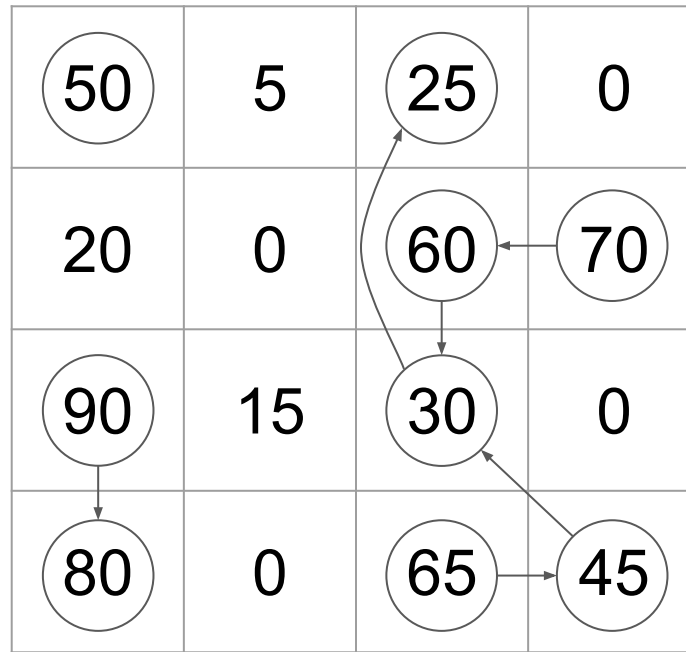
We sort cells by numbers decreasingly, and we build the tree by adding nodes in order, and connecting subtrees of neighbouring cells.

50	5	25	0
20	0	60	70
90	15	30	0
80	0	65	45

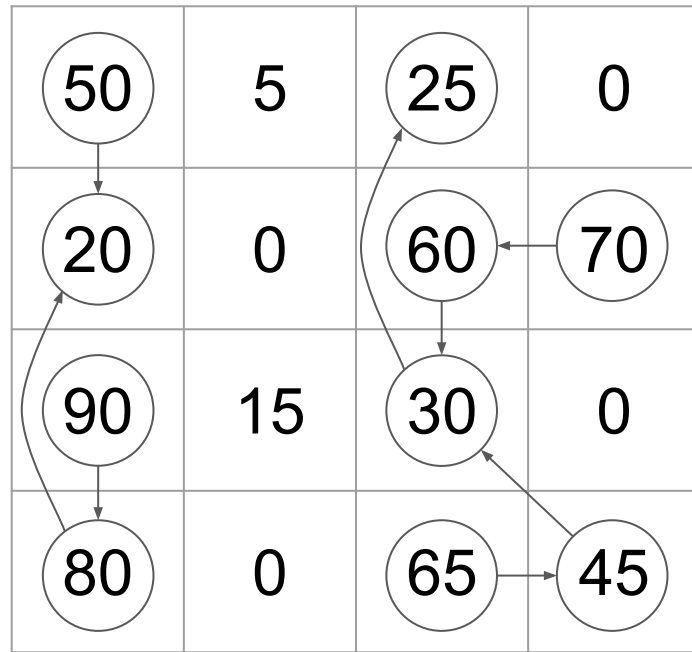
We sort cells by numbers decreasingly, and we build the tree by adding nodes in order, and connecting subtrees of neighbouring cells.



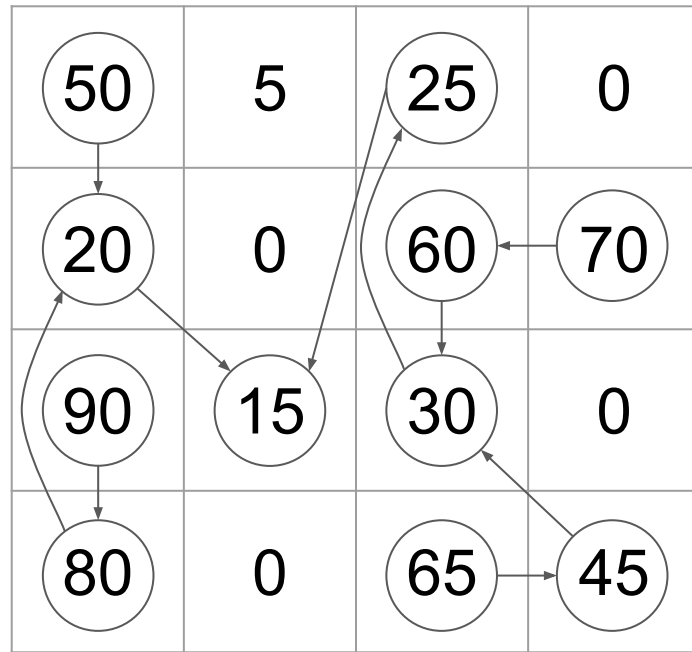
We sort cells by numbers decreasingly, and we build the tree by adding nodes in order, and connecting subtrees of neighbouring cells.



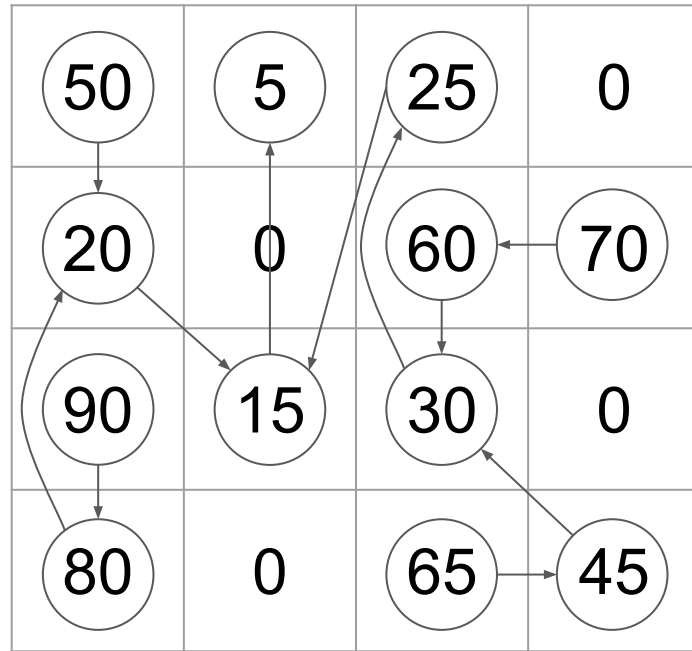
We sort cells by numbers decreasingly, and we build the tree by adding nodes in order, and connecting subtrees of neighbouring cells.



We sort cells by numbers decreasingly, and we build the tree by adding nodes in order, and connecting subtrees of neighbouring cells.

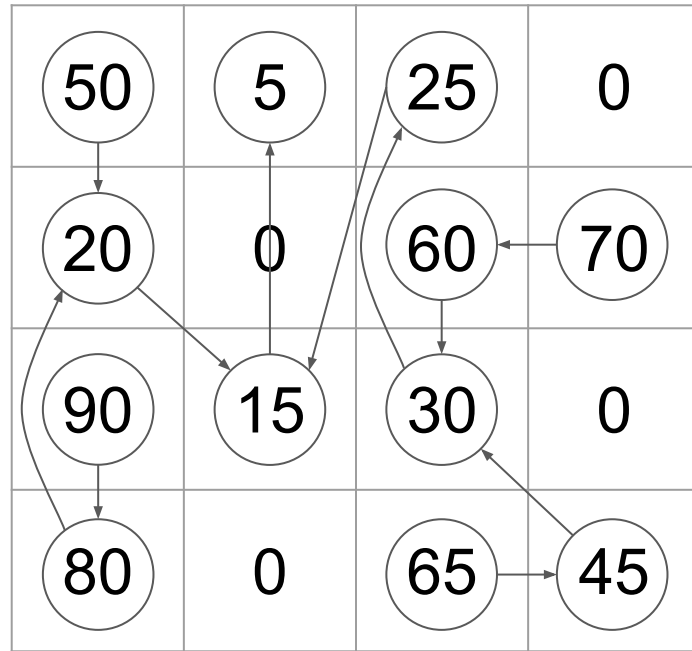


We sort cells by numbers decreasingly, and we build the tree by adding nodes in order, and connecting subtrees of neighbouring cells.



We sort cells by numbers decreasingly, and we build the tree by adding nodes in order, and connecting subtrees of neighbouring cells.





A unique path between two cells in our tree is the optimal path.

The minimum number on the path is found at the lowest common ancestor of two nodes in the tree.

# Problem D

## Dancing Disks

Submits: 5

Accepted: at least 1

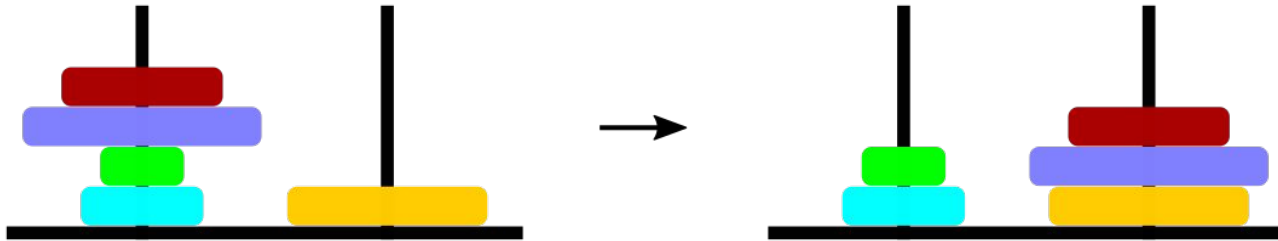
First solved by: UW1

University of Warsaw

(Nadara, Smulewicz, Sokołowski)

03:46:41

Author: Luka Kalinović

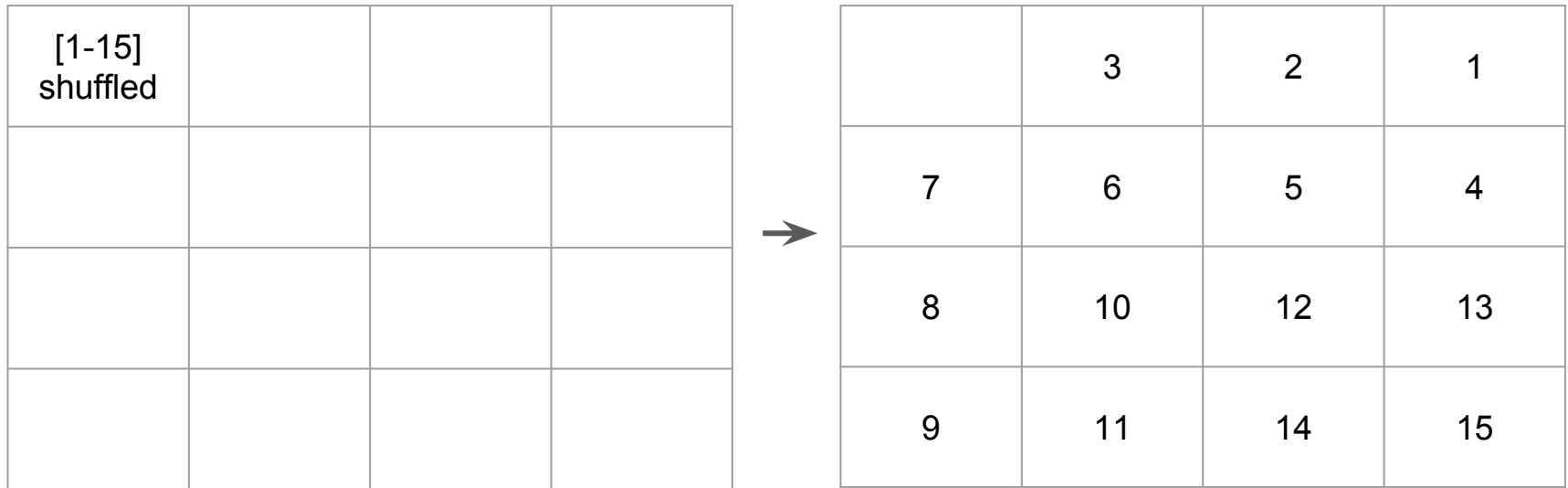


36 rods organized in 6 rows and 6 columns.

The puzzle starts with  $n$  disks stacked in arbitrary order on the rod in the upper-left corner.

In each step, a player can pick up a stack of one or more disks and transfer them to the top of the rod immediately below or to the right.

Find any sequence of moves that moves disks to the rod in the lower-right corner, such that disks are sorted by size.



Let's describe a simple solution for a  $R \times C$  matrix of rods that can sort  $R \cdot C - 1$  disks.

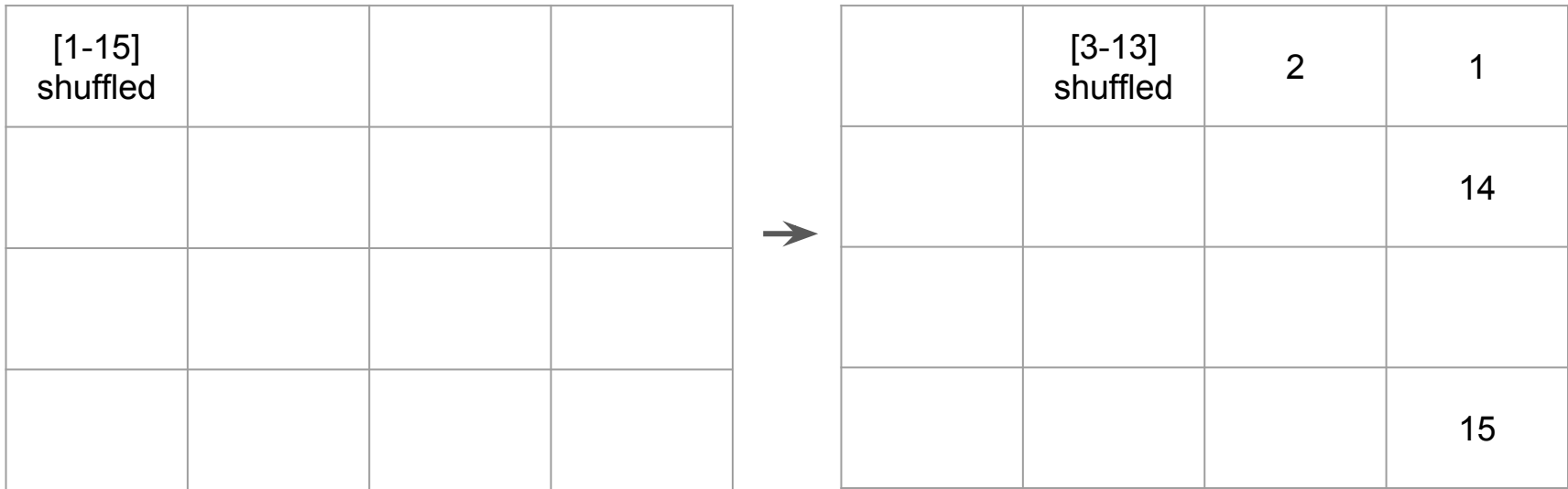
First, we go through disks one by one, and move each disk to a separate rod, with the largest disk at the target rod in the lower-right corner.

	3	2	1
7	6	5	4
8	10	12	13
9	11	14	15



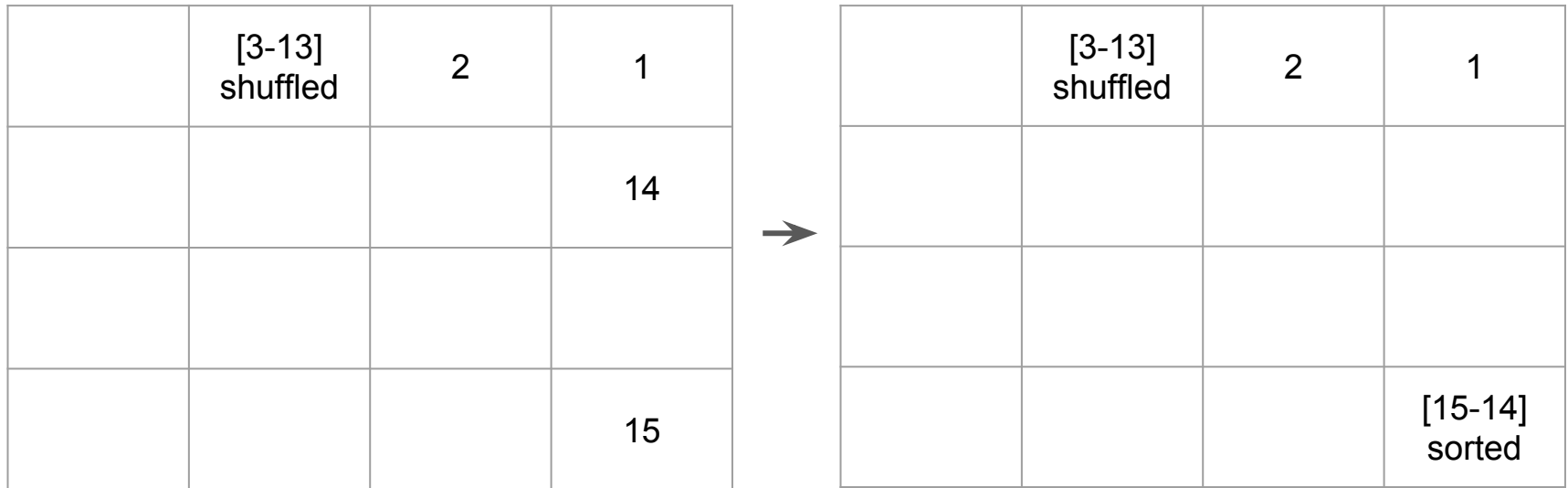
			[15-1] sorted

Second, we move disks one by one to the target rod, in order of decreasing size.

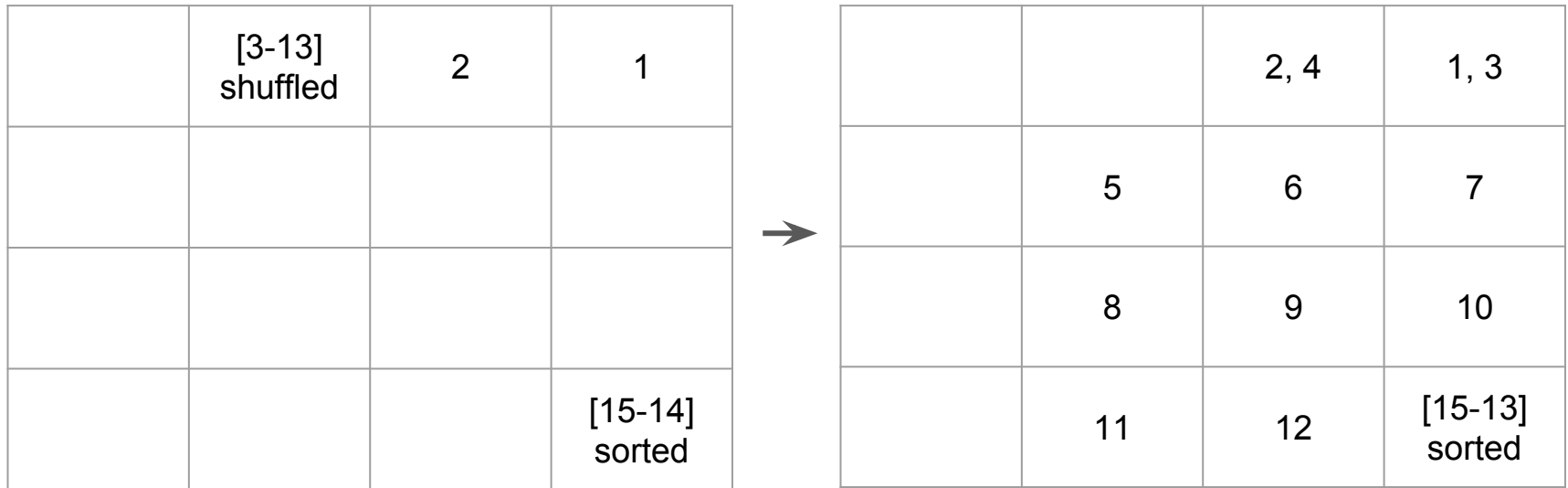


The algorithm is wasteful because it only moves single disks to each rod.

For example, our algorithm is able to sort  $3 \cdot 4 - 1 = 11$  disks using the 3 x 4 matrix of rods. So we could have as well assigned disks to rods as above.

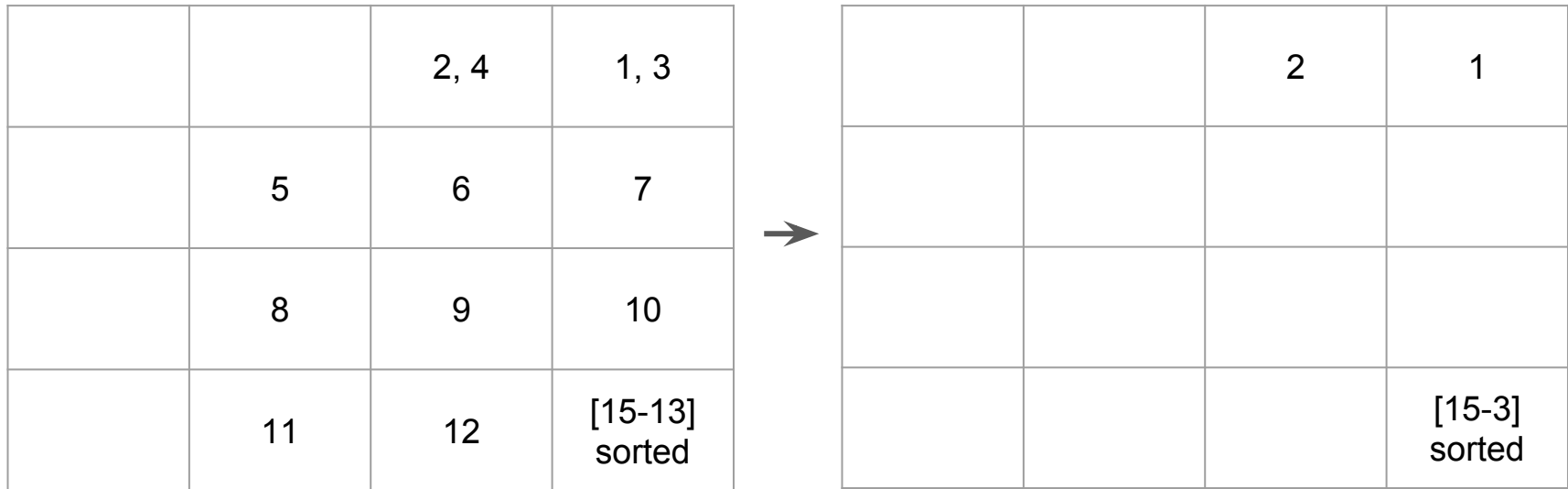


In the second step, we recursively use the same algorithm to move a sequence of consecutive unsorted disks to the final rod.



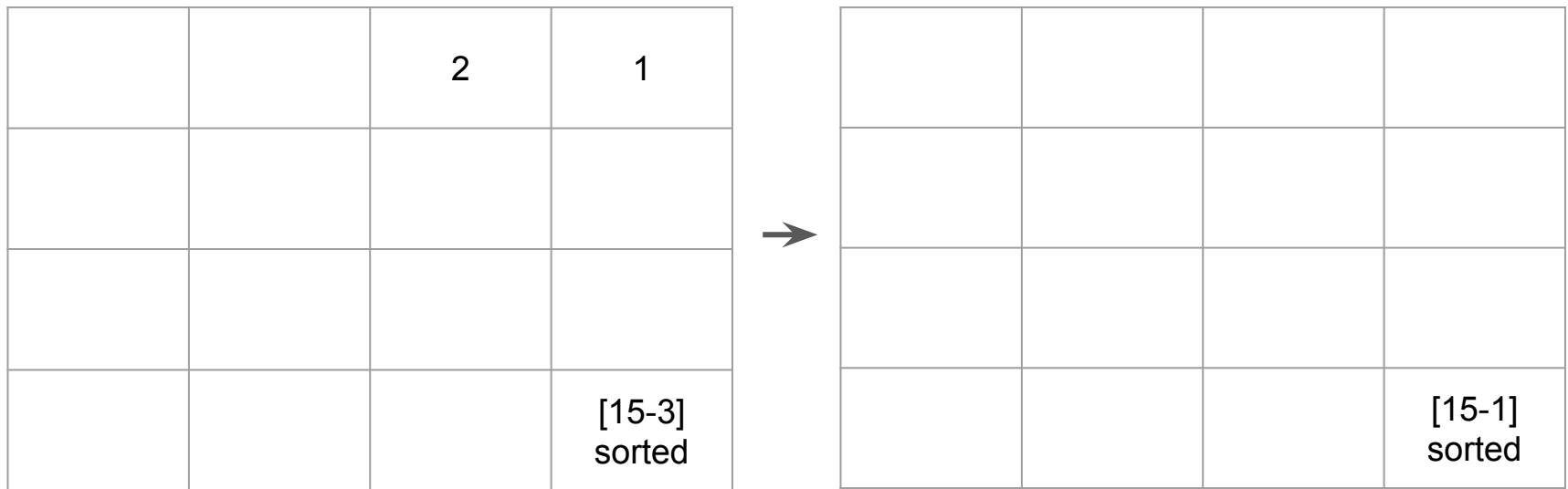
In the second step, we recursively use the same algorithm to move a sequence of consecutive unsorted disks to the final rod.





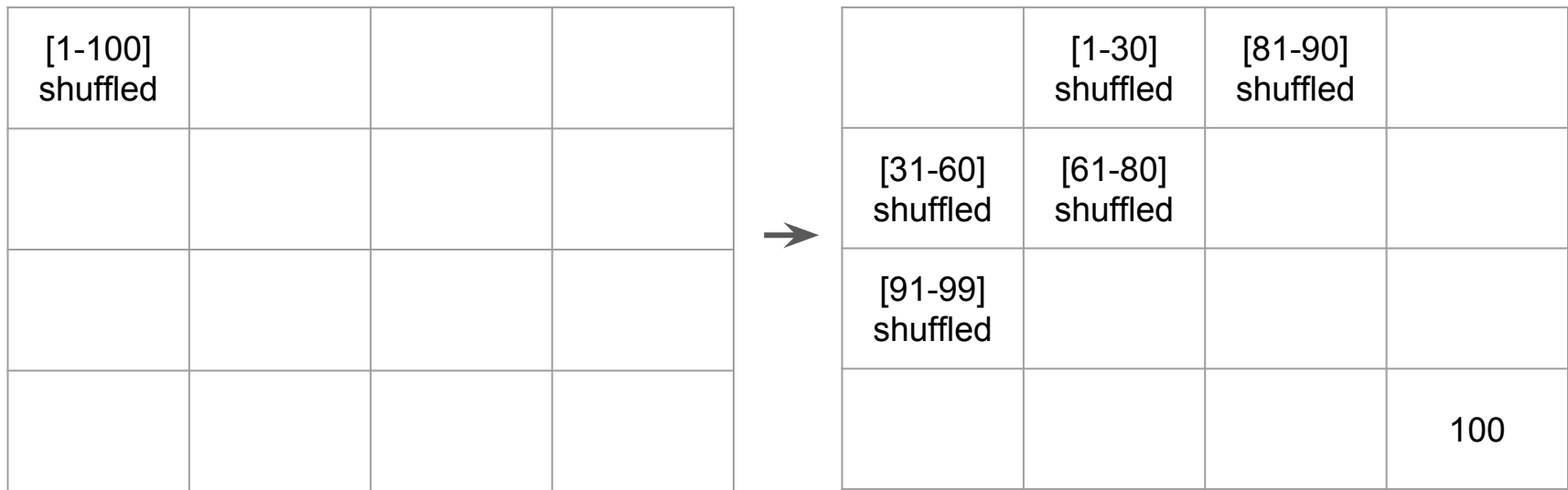
In the second step, we recursively use the same algorithm to move a sequence of consecutive unsorted disks to the final rod.

Note that we don't mind if there are already disks at rods we use in the recursive call. We won't touch them.



In the second step, we recursively use the same algorithm to move a sequence of consecutive unsorted disks to the final rod.

Note that we don't mind if there are already disks at rods we use in the recursive call. We won't touch them.



We come up with a revised algorithm where we go one by one through the first rod, and move disks to other rods such that disks on each rod have consecutive sizes (but may be shuffled).

We'll move as many disks to a rod as this algorithm can successfully handle.

But how much is that?

Let  $k(R, C)$  be the maximum number of disks we can sort with a matrix or  $R$  by  $C$  rods, using the described algorithm.

$$k(1, 1) = 1$$

$$k(R, C) = \sum(k(r, c), 1 \leq r \leq R, 1 \leq c \leq C, r \neq R \text{ or } c \neq C)$$

1	1	2	4	8	16
1	3	8	20	48	112
2	8	26	76	208	544
4	20	76	252	768	2208
8	48	208	768	2568	8016
16	112	544	2208	8016	26928

$$k(1, 1) = 1$$

$$k(R, C) = \sum(k(r, c), 1 \leq r \leq R, 1 \leq c \leq C, r \neq R \text{ or } c \neq C)$$

We can only sort 26928 disks using this algorithm.

1	2	3	6	12	24
2	5	13	32	76	176
3	13	42	122	323	864
6	32	122	404	1228	3520
12	76	332	1228	4104	12792
24	176	864	3520	12792	42960

Luckily, we can just add custom logic to handle 2 disks on 2 x 1 matrix of rods.

$$k(1, 1) = 1, \quad k(1, 2) = 2, \quad k(2, 1) = 2$$

$$k(R, C) = \sum(k(r, c), 1 \leq r \leq R, 1 \leq c \leq C, r \neq R \text{ or } c \neq C)$$

$$k(6, 6) = 42960 \quad \checkmark$$

# Problem E

## Easy Equation

Submits: 13

Accepted: at least 3

First solved by: UW3

University of Warsaw

(Hołubowicz, Paluszek, Tabaszewski)

01:40:17

Author: Luka Kalinovič

$$a^2 + b^2 + c^2 = k \cdot (a \cdot b + b \cdot c + c \cdot a) + 1$$

Given  $k \geq 2$ , find  $n$  solutions to the equation such that:

- $a$ ,  $b$  and  $c$  are positive integers, and
- we only use a number once across all  $n$  solutions.



$$a^2 + b^2 + c^2 = k \cdot (a \cdot b + b \cdot c + c \cdot a) + 1$$

Let's find any solution. Try  $a = 0$ ,  $b = 1$ :

$$0^2 + 1^2 + c^2 = k (0 \cdot 1 + 1 \cdot c + c \cdot 0) + 1$$

$$c^2 = k \cdot c$$

$$c = k$$

$(0, 1, k)$  is always a solution.

$$a^2 + b^2 + c^2 = k \cdot (a \cdot b + b \cdot c + c \cdot a) + 1$$

Key idea: Is there another solution if we keep two out of three numbers? We fix  $b$  and  $c$ , and solve for  $a$ .

$$a^2 - a \cdot (k \cdot b + k \cdot c) + b^2 + c^2 - k \cdot b \cdot c - 1 = 0$$

We can compute the derivative to find the vertex of this parabola at  $(k \cdot b + k \cdot c) / 2$ .

We can also find the vertex as arithmetic mean of two solutions.  $(a' + a'') / 2 = (k \cdot b + k \cdot c) / 2$

$$a'' = k \cdot b + k \cdot c - a'$$

$$a^2 + b^2 + c^2 = k \cdot (a \cdot b + b \cdot c + c \cdot a) + 1$$

Given a solution  $(a, b, c)$  we can find another three solutions:

$$(k \cdot b + k \cdot c - a, b, c)$$

$$(a, k \cdot a + k \cdot c - b, c)$$

$$(a, b, k \cdot a + k \cdot b - c)$$

We use BFS to explore this infinite graph of solutions starting from  $(0, 1, k)$ . Only output solutions with unique positive integers we haven't used so far.

# Problem B

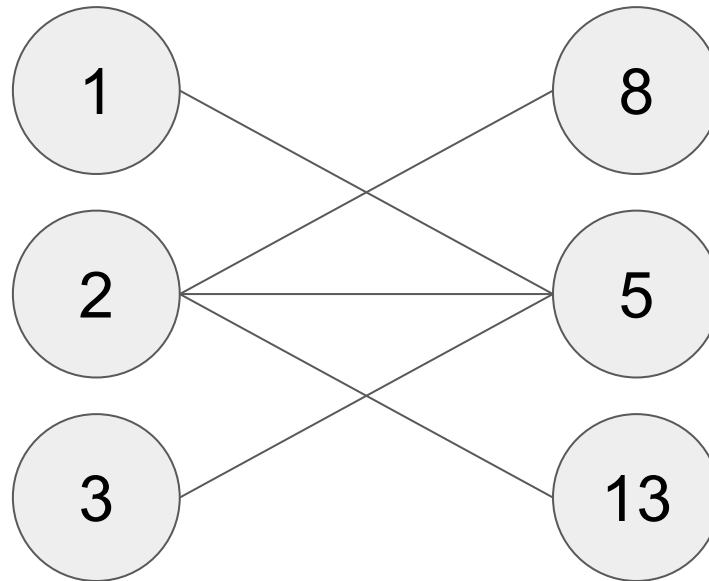
## Bipartite Blanket

Submits: 11

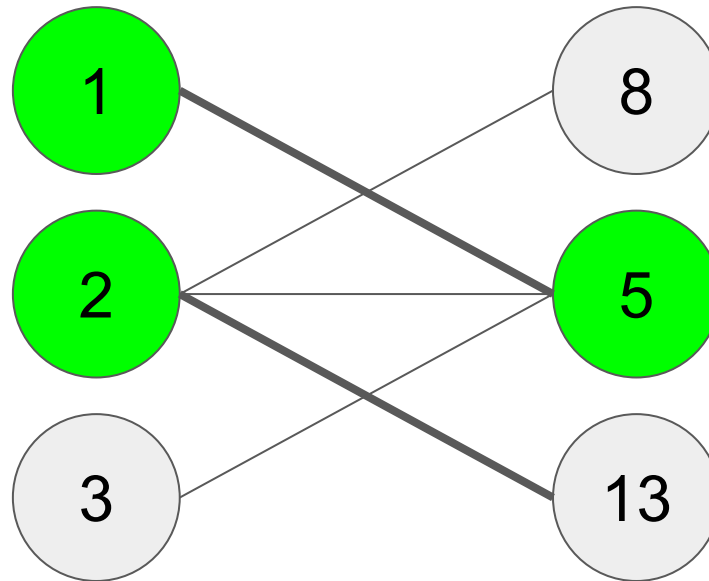
Accepted: at least 5

First solved by: UW2  
University of Warsaw  
(Dębowski, Radecki, Sommer)  
02:28:59

Author: Gustav Matula

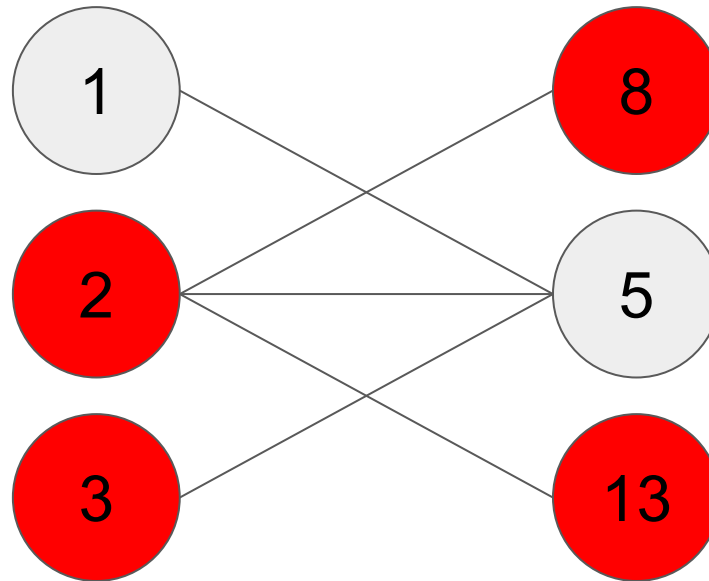


A set of nodes  $V$  is valid if there exists a matching  $M$  such that every node is an endpoint of at least one edge in  $M$ .



A set of nodes  $V$  is valid if there exists a matching  $M$  such that every node is an endpoint of at least one edge in  $M$ .

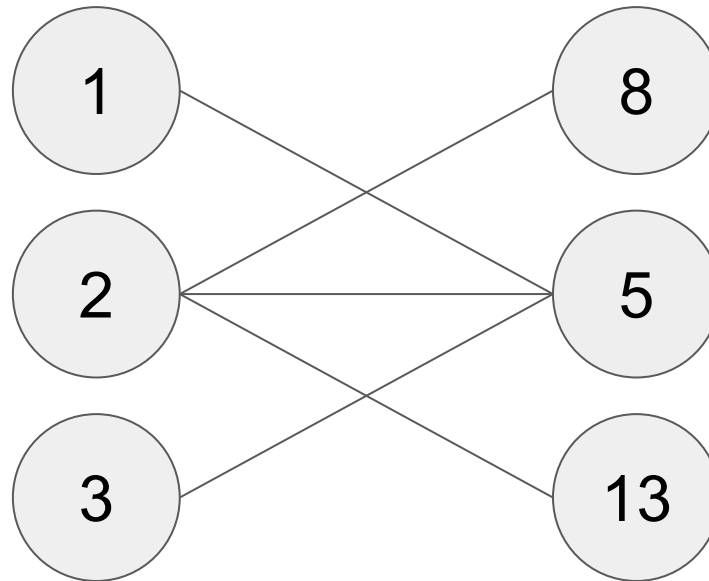
Set  $\{1, 2, 5\}$  is valid.



A set of nodes  $V$  is valid if there exists a matching  $M$  such that every node is an endpoint of at least one edge in  $M$ .

Set  $\{1, 2, 5\}$  is valid.

Set  $\{2, 3, 8, 13\}$  is not valid.



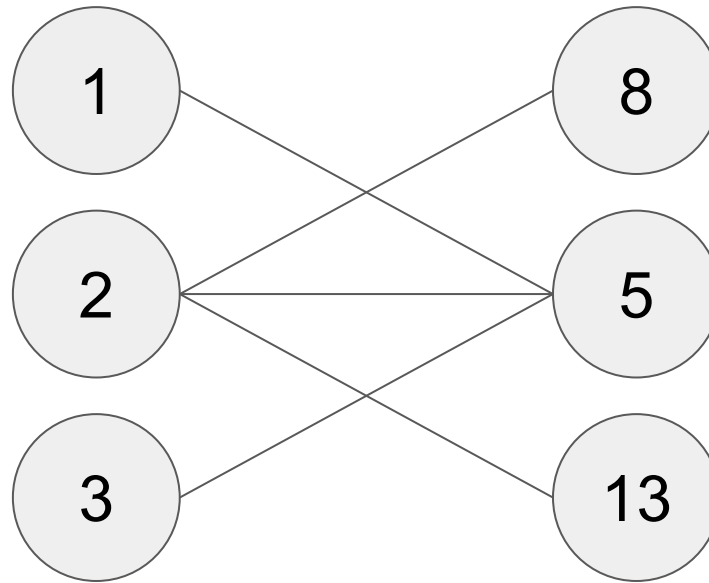
A set of nodes  $V$  is valid if there exists a matching  $M$  such that every node is an endpoint of at least one edge in  $M$ .

Set  $\{1, 2, 5\}$  is valid.

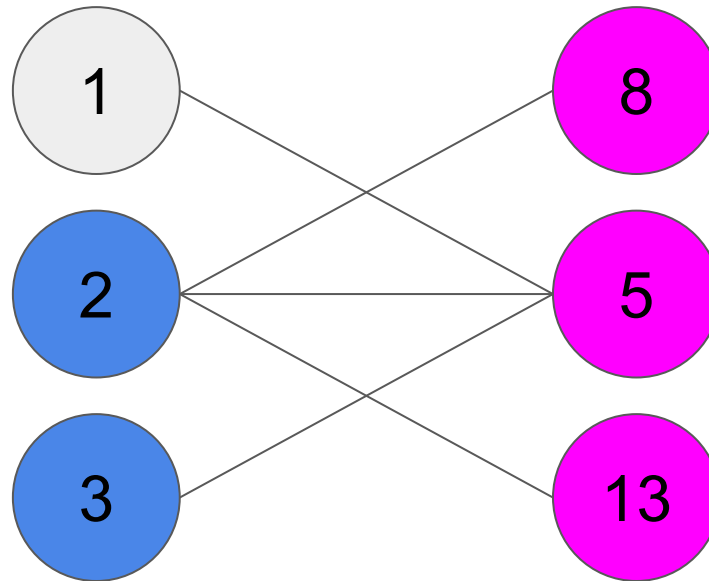
Set  $\{2, 3, 8, 13\}$  is not valid.

We need to count the number of valid sets with sum of node weights greater than a given threshold  $t$ .



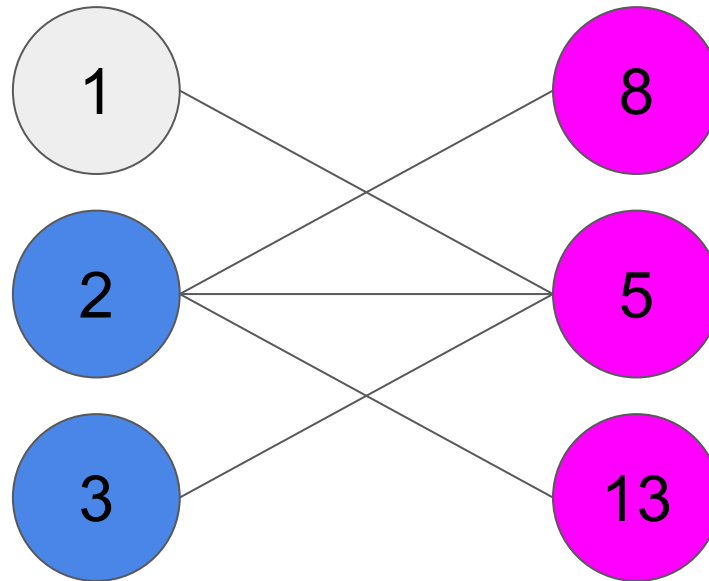


Hall's marriage theorem gives a necessary and sufficient condition for existence of a matching that covers a set of nodes on one side of the graph.

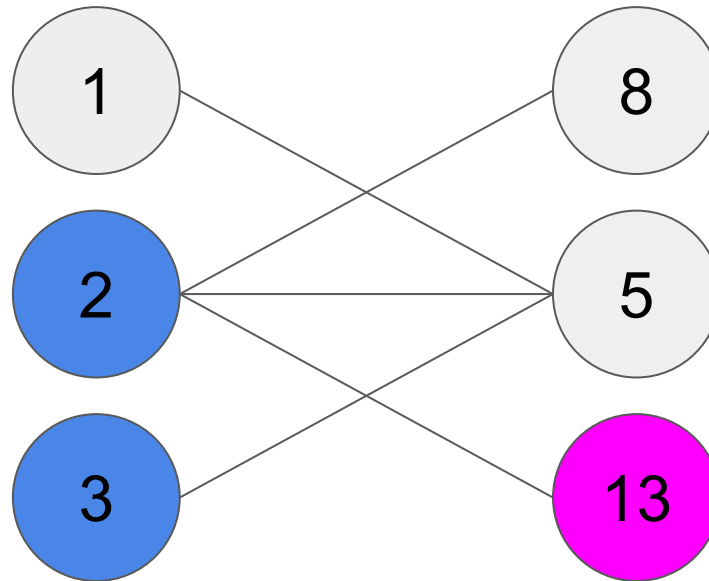


For a set of vertices  $A$  on one side of the graph, let  $N(A)$  denote the neighborhood of  $A$  in  $G$ , i.e. the set of all vertices on the other side of the graph adjacent to some element of  $A$ .

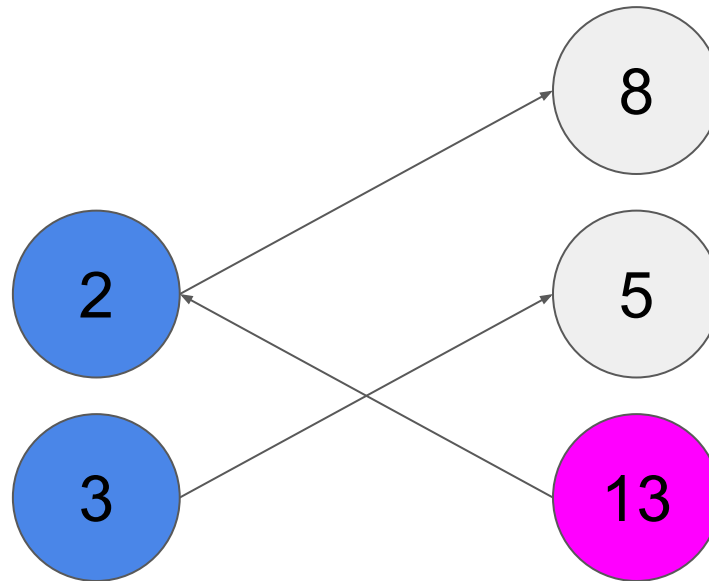
There is a matching that blankets  $A$  iff  $|x| \leq |N(x)|$ , for every subset  $x$  of  $A$ .



Iterating through all subsets of nodes on one side and applying the Hall's theorem, we can find which ones are valid in  $O(n \cdot 2^n)$ .

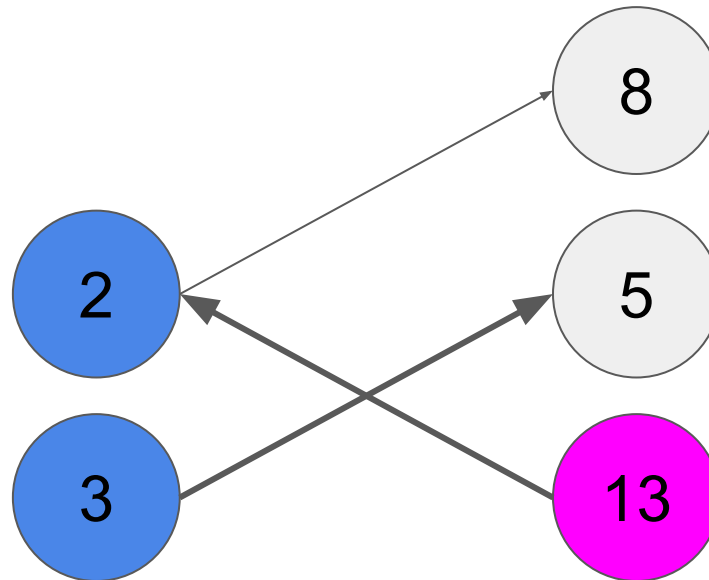


Key observation: Given a valid set  $A$  of nodes on one side, and a valid set  $B$  of nodes on the other side of the graph, set  $A \cup B$  is valid as well.



Proof sketch: We take a minimal matching  $M_A$  that covers **A**, and a minimal matching  $M_B$  that covers **B**, and we build a directed graph  $D$ , where we:

- add a directed edge from  $(a, b)$  if edge  $(a, b)$  is in  $M_A$ .
- add a directed edge from  $(b, a)$  if edge  $(b, a)$  is in  $M_B$ .




Every node in  $D$  has an out-degree of 1 if node is in  $A$  or  $B$  or 0 otherwise, so it consists of paths and/or cycles. We create a matching covering both  $A$  and  $B$  by taking every second edge on each cycle and path. The matching covers every node of  $D$ , except for nodes at the end of even-length paths. However, a node at the end of a path has a degree of 0, so it is not in  $A$  or  $B$ .

$$X = \{5, 10, 15, 20, 30\}$$

$$Y = \{10, 20, 25, 40\}$$

$$t = 45$$

The problem of counting valid subsets with a weight greater than  $t$ , is reduced to a standard problem of counting the number of ways to select two numbers  $x$  and  $y$ , one from set  $X$ , other from set  $Y$ , such that  $x + y > t$ .

  
 $X = \{5, 10, 15, 20, 30\}$

$Y = \{10, 20, 25, 40\}$

$t = 45$   


We can solve that by using two pointers. The first one runs on  $X$  in increasing order, while the second runs on  $Y$  in decreasing order.

If the sum of the two numbers is greater than the threshold, we move the pointer in  $Y$ , otherwise we move the pointer in  $X$ .



↓  
 $X = \{5, 10, 15, 20, 30\}$

$Y = \{10, 20, 25, 40\}$

↑  
 $t = 45$

We can solve that by using two pointers. The first one runs on  $X$  in increasing order, while the second runs on  $Y$  in decreasing order.

If the sum of the two numbers is greater than the threshold, we move the pointer in  $Y$ , otherwise we move the pointer in  $X$ .

$X = \{5, 10, 15, 20, 30\}$

$Y = \{10, 20, 25, 40\}$

$t = 45$

We can solve that by using two pointers. The first one runs on  $X$  in increasing order, while the second runs on  $Y$  in decreasing order.

If the sum of the two numbers is greater than the threshold, we move the pointer in  $Y$ , otherwise we move the pointer in  $X$ .

$X = \{5, 10, 15, 20, 30\}$

$Y = \{10, 20, 25, 40\}$

$t = 45$

We can solve that by using two pointers. The first one runs on  $X$  in increasing order, while the second runs on  $Y$  in decreasing order.

If the sum of the two numbers is greater than the threshold, we move the pointer in  $Y$ , otherwise we move the pointer in  $X$ .


$X = \{5, 10, 15, 20, 30\}$

$Y = \{10, 20, 25, 40\}$


$t = 45$

We can solve that by using two pointers. The first one runs on  $X$  in increasing order, while the second runs on  $Y$  in decreasing order.

If the sum of the two numbers is greater than the threshold, we move the pointer in  $Y$ , otherwise we move the pointer in  $X$ .

$$X = \{5, 10, 15, 20, 30\}$$


$$Y = \{10, 20, 25, 40\}$$

$$t = 45$$


We can solve that by using two pointers. The first one runs on  $X$  in increasing order, while the second runs on  $Y$  in decreasing order.

If the sum of the two numbers is greater than the threshold, we move the pointer in  $Y$ , otherwise we move the pointer in  $X$ .

$X = \{5, 10, 15, 20, 30\}$

$Y = \{10, 20, 25, 40\}$

$t = 45$

We can solve that by using two pointers. The first one runs on  $X$  in increasing order, while the second runs on  $Y$  in decreasing order.

If the sum of the two numbers is greater than the threshold, we move the pointer in  $Y$ , otherwise we move the pointer in  $X$ .

$X = \{5, 10, 15, 20, 30\}$

$Y = \{10, 20, 25, 40\}$

$t = 45$

We can solve that by using two pointers. The first one runs on  $X$  in increasing order, while the second runs on  $Y$  in decreasing order.

If the sum of the two numbers is greater than the threshold, we move the pointer in  $Y$ , otherwise we move the pointer in  $X$ .

# Problem G

## Geohash Grid

Submits: 3  
Accepted: ?

Author: Ante Đerek, Luka Kalinovčić



21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42

Given an axis-parallel polygon in a grid of size  $2^n$  by  $2^n$ , find the best  $k$ -approximation using  $k$  intervals of numbers that completely cover the polygon.

21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42

The optimal 1-approximation: 3–37.

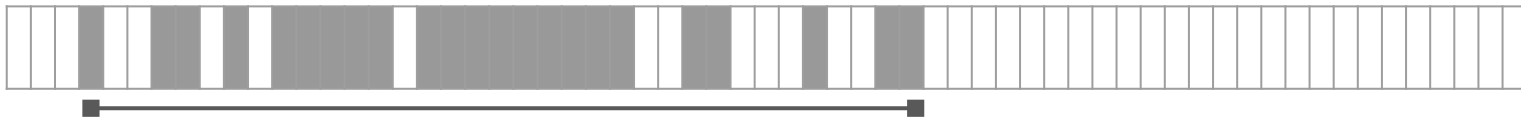
21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42

The optimal 2-approximation: 3–29, 33–37.

21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42

The optimal 3-approximation: 3–25, 28–29, 33–37.

21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42



If we lay down numbers on a single line and mark numbers within the polyline, the optimal 1-approximation is simply the interval that covers every marked number.

21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42

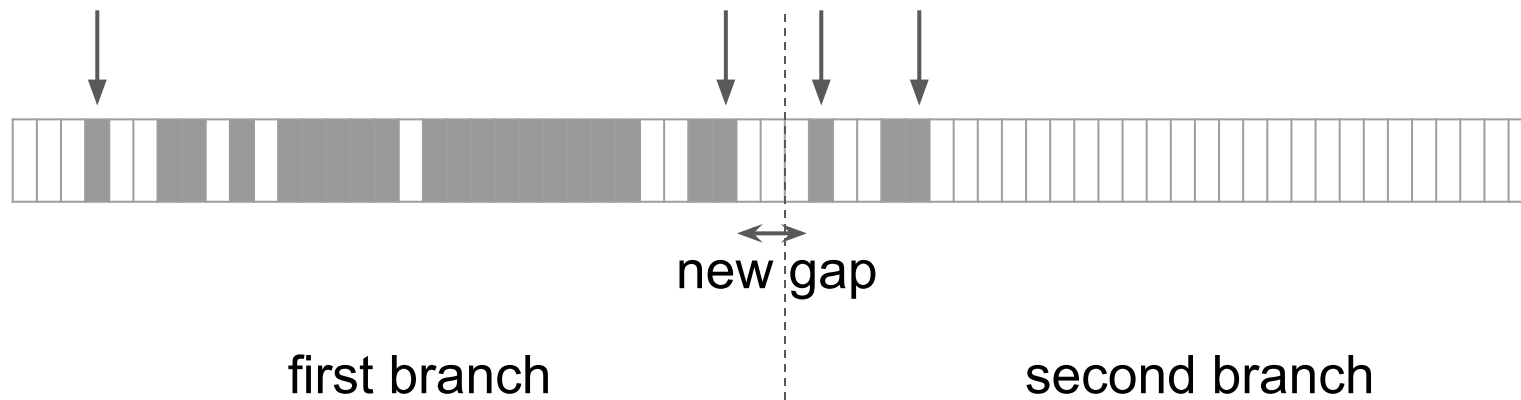


We obtain the optimal 2-approximation by removing the largest gap.

21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42



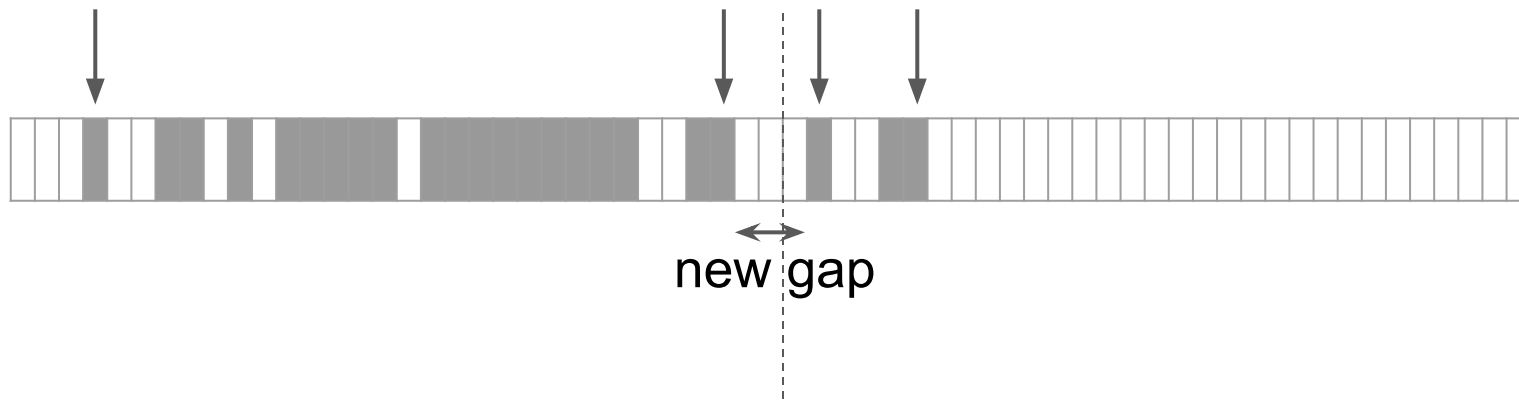
We obtain the optimal  $k$ -approximation by removing  $(k - 1)$  largest gaps.



We can find all gaps recursively if we return the first and the last marked number in the viewport.

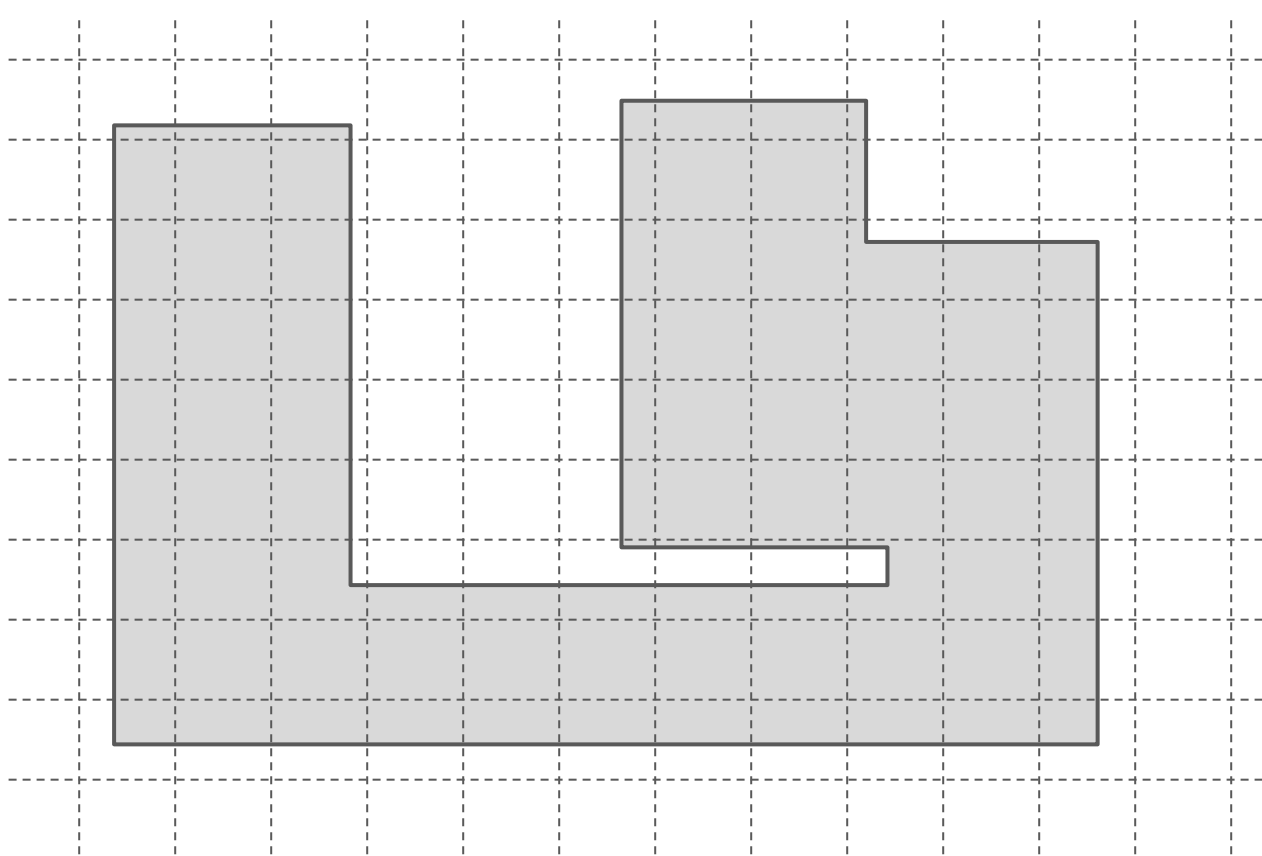
At every stage, there is a potential gap between the last marked number returned by the first branch, and the first marked number returned by the second branch.



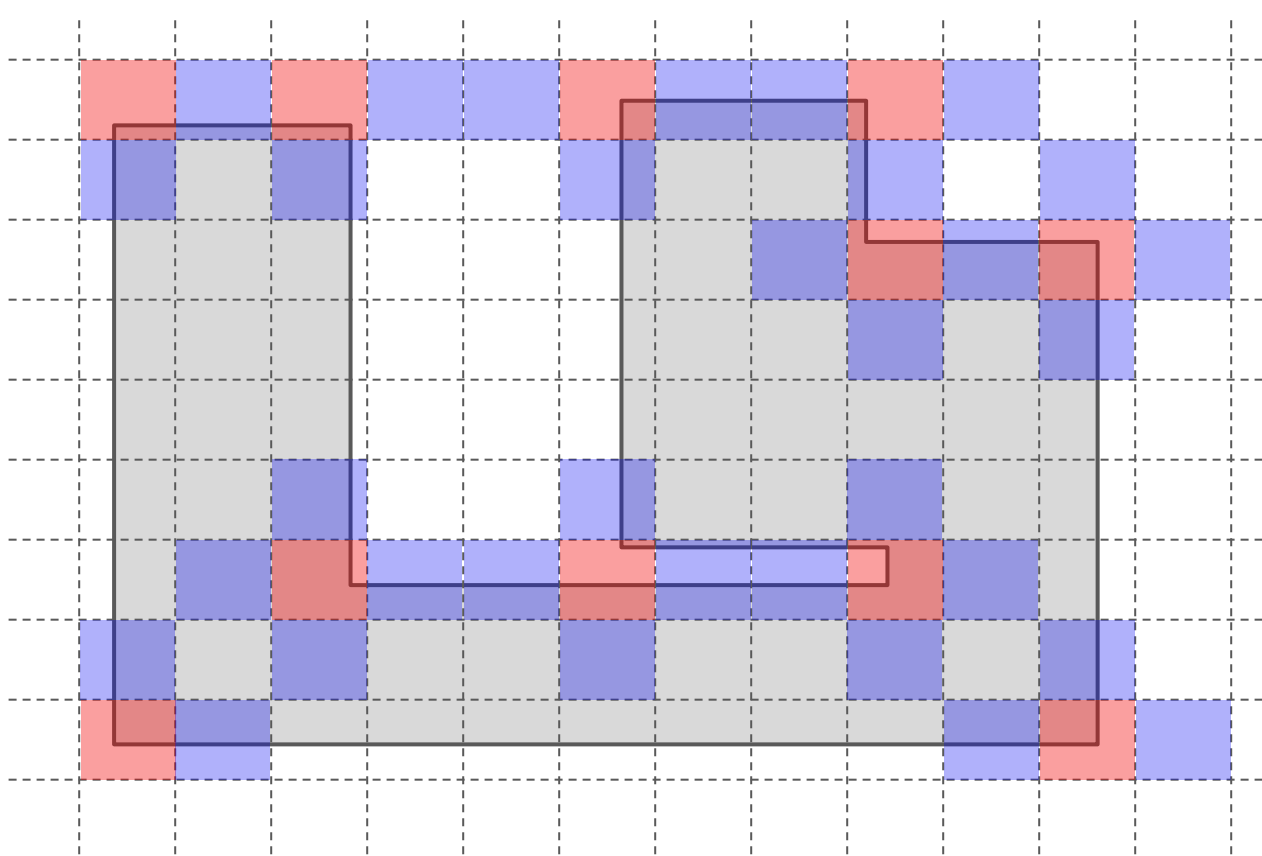


There can be billions of gaps, so we can't find them all one by one.

However, there is only a limited number of gap sizes that can occur, i.e. the histogram of gaps would contain a small number of very tall columns.



Key observation: Given a polygon with  $m$  vertices and an arbitrary rectangular tiling of a plane, there are at most  $O(m)$  different tiles with respect to intersection with the polygon.

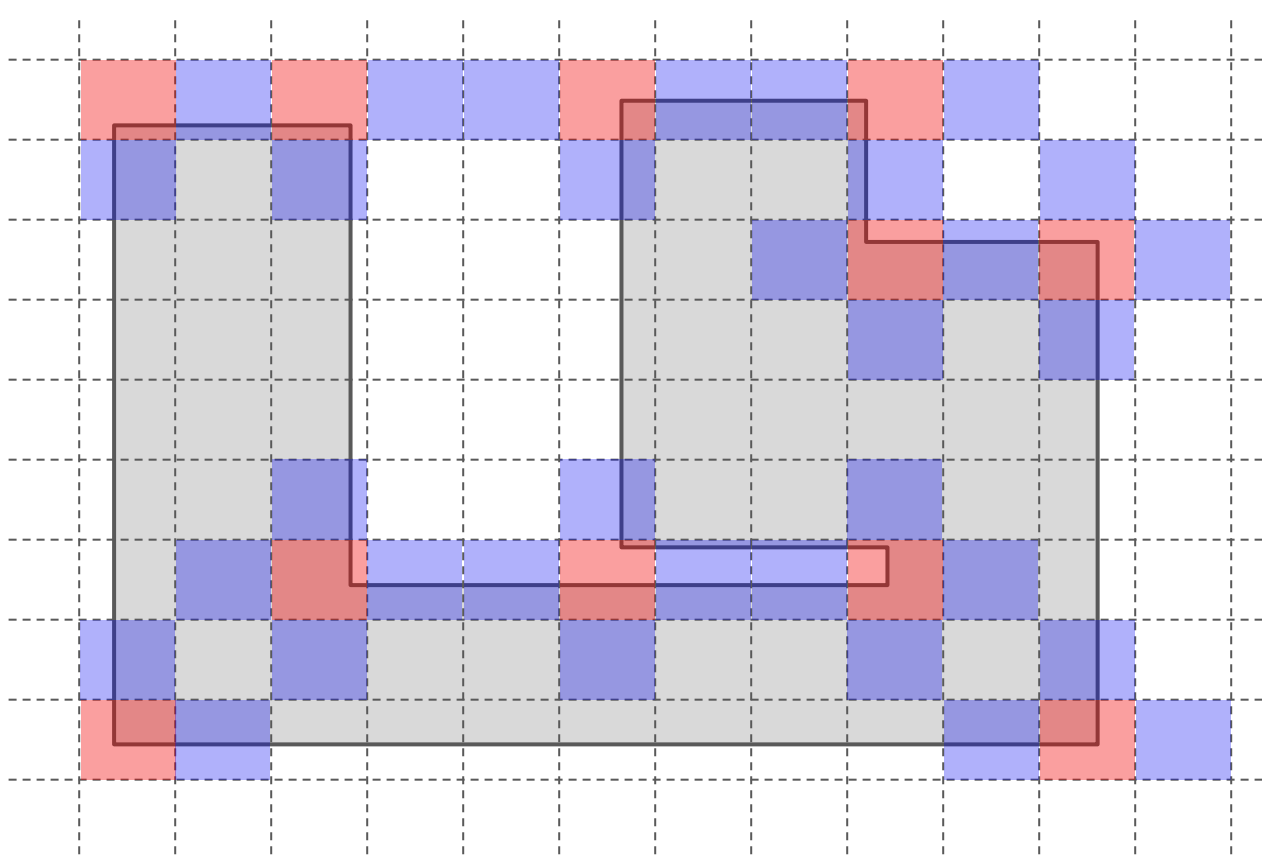


Proof sketch:

Color every tile containing a vertex **red**.

Color every tile adjacent to a red tile **blue**.

Any other tile looks like one of the blue tiles.



In our recursive algorithm, we effectively tile the plane  $2 \cdot n$  times with different rectangle sizes. Because a viewport of a given size can intersect with polygon in  $O(m)$  different ways, we can only get  $O(n \cdot m)$  different gap sizes.

Finally, we'll revise our gap-finding algorithm, not to visit both children when the split would result in two tiles that look the same with respect to intersecting with the polygon. Instead we visit only one children, but create twice as many gaps in that branch.

It can be shown that this way we'll only visit  $O(n \cdot m^2)$  states. We have to look at the polygon through the viewport at every state to determine whether to split, so we have a total complexity of  $O(n \cdot m^3)$  to find all gaps.

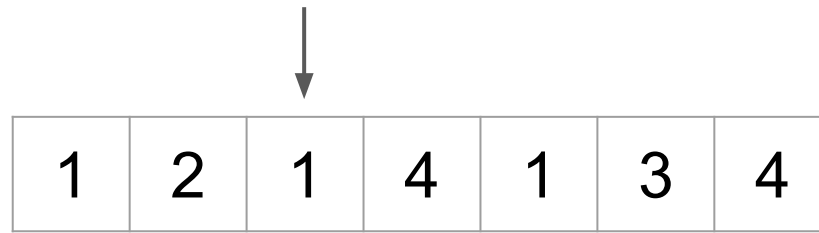
We can then find  $(k - 1)$  largest gaps in  $O(\log n \cdot m)$ .

# Problem I

## Invisible Integers

Submits: 2  
Accepted: ?

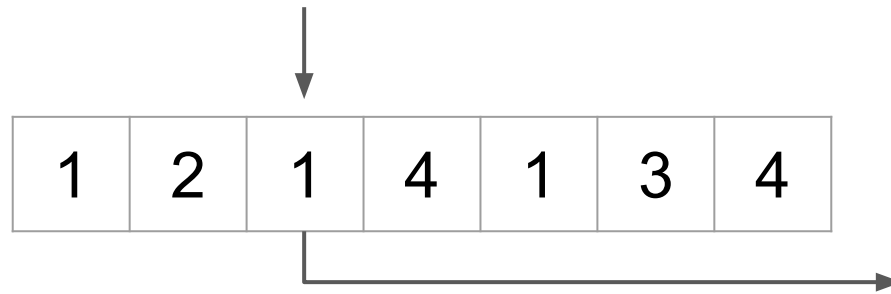
Author: Luka Kalinovčić



Hidden sequence of integers 1 through 9.

A hint is a sequence obtained as follows:

- An arbitrary starting position is chosen in the hidden sequence.

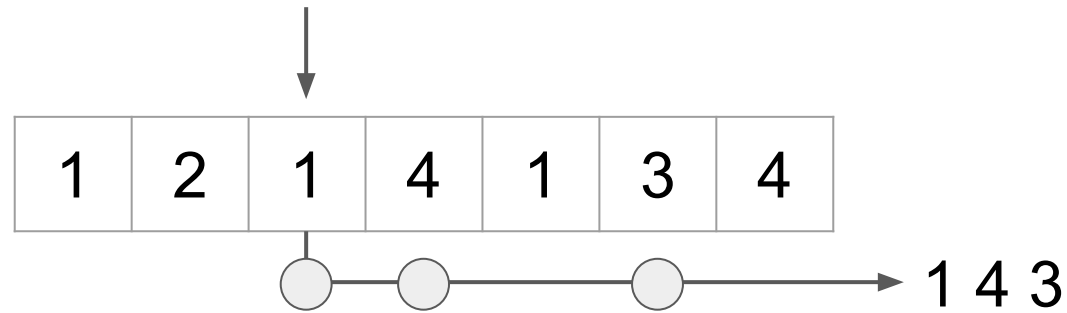


Hidden sequence of integers 1 through 9.

A hint is a sequence obtained as follows:

- An arbitrary starting position is chosen in the hidden sequence.
- An arbitrary direction is chosen — either left or right.

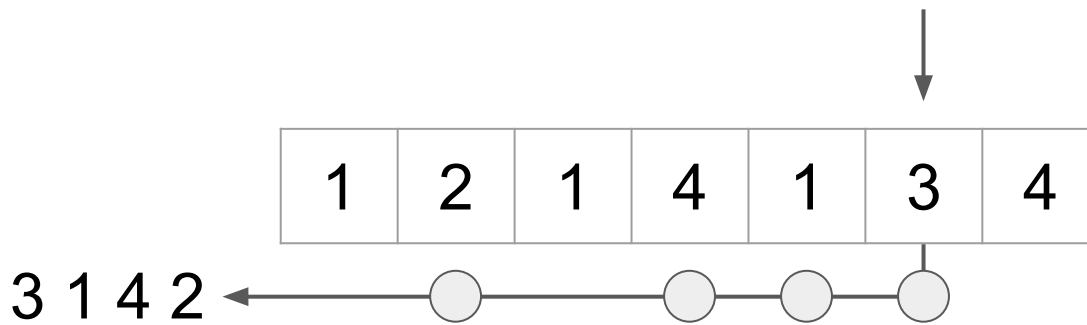




Hidden sequence of integers 1 through 9.

A hint is a sequence obtained as follows:

- An arbitrary starting position is chosen in the hidden sequence.
- An arbitrary direction is chosen — either left or right.
- Traverse integers of the hidden sequence in the chosen direction, appending an integer only the first time we see it.



Hidden sequence of integers 1 through 9.

A hint is a sequence obtained as follows:

- An arbitrary starting position is chosen in the hidden sequence.
- An arbitrary direction is chosen — either left or right.
- Traverse integers of the hidden sequence in the chosen direction, appending an integer only the first time we see it.

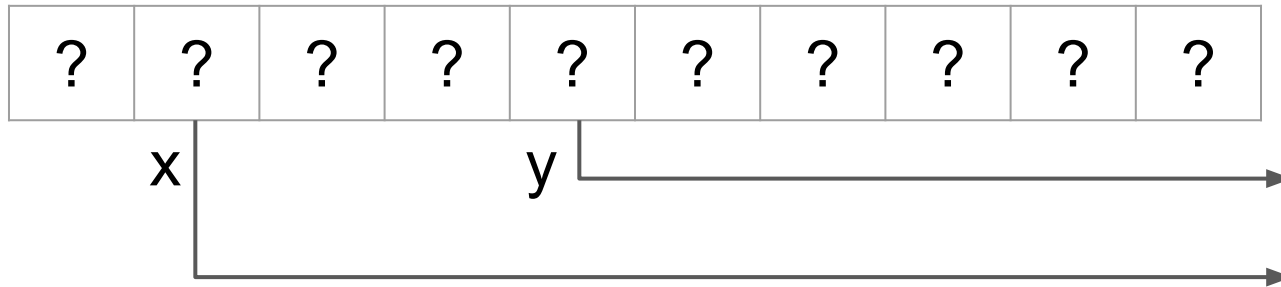
1	2	1	4	1	3	4
---	---	---	---	---	---	---

Hidden sequence of integers 1 through 9.

A hint is a sequence obtained as follows:

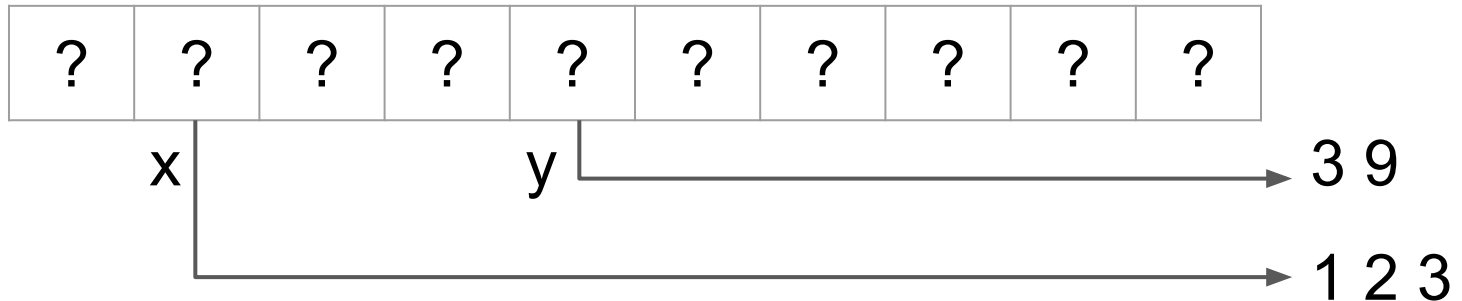
- An arbitrary starting position is chosen in the hidden sequence.
- An arbitrary direction is chosen — either left or right.
- Traverse integers of the hidden sequence in the chosen direction, appending an integer only the first time we see it.

Given n hints, find the shortest hidden sequence!



For now, we assume all hints go right.

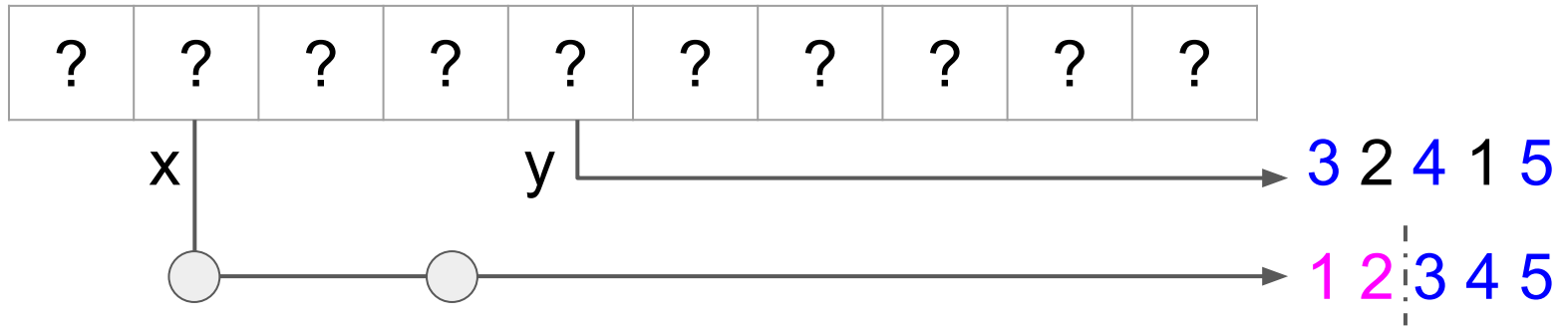
Given two hints  $x$  and  $y$ , we say that  $y$  *follows*  $x$ , if  $y$ 's starting position is after  $x$ 's starting position.



For now, we assume all hints go right.

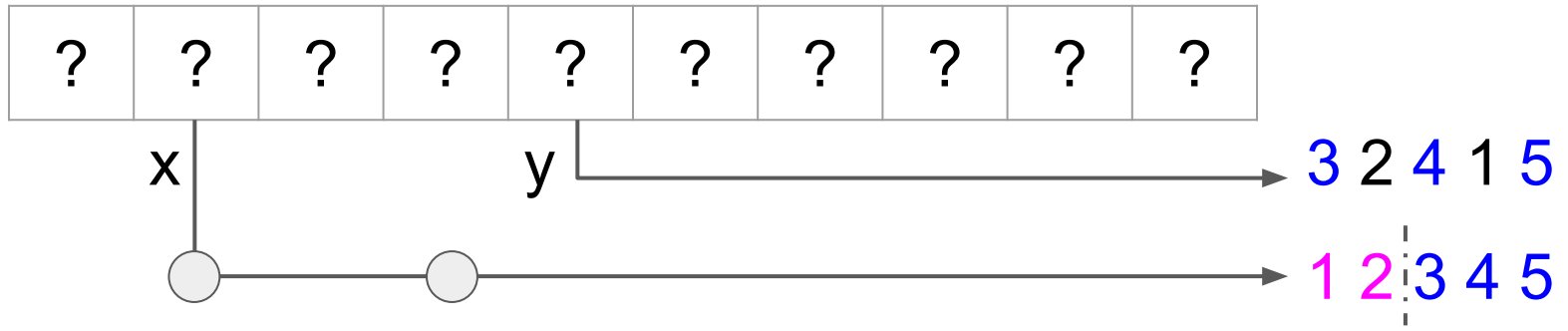
Given two hints  $x$  and  $y$ , we say that  $y$  *follows*  $x$ , if  $y$ 's starting position is after  $x$ 's starting position.

Hint  $y$  cannot follow hint  $x$  iff  $y$  contains a digit not contained in  $x$ .



We say hint  $x$  is able to *transition into*  $y$  at index  $i$ , if:

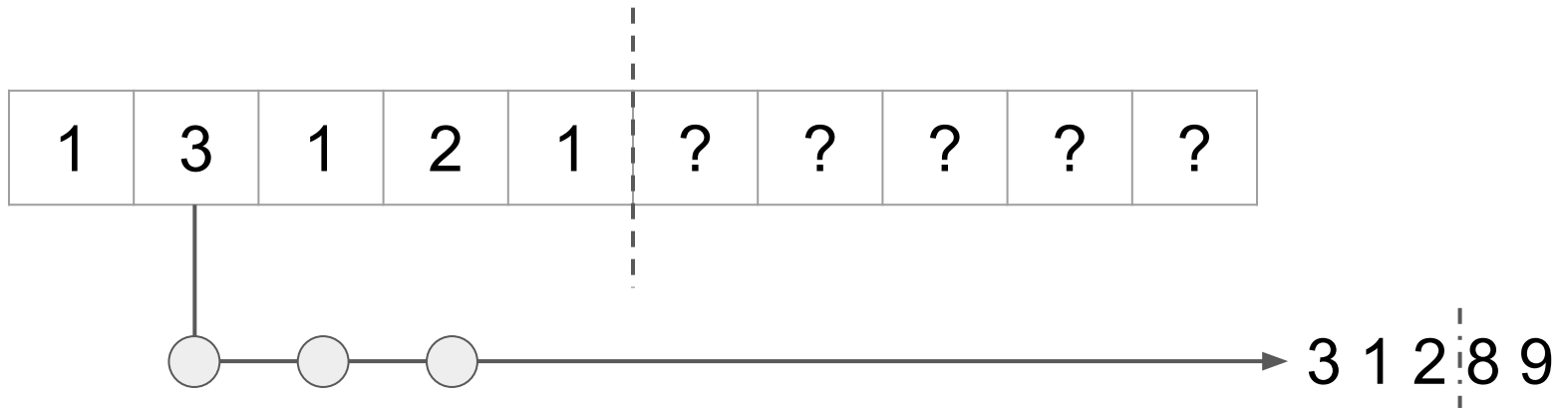
- $x$  has found  $i$  digits **up to  $y$ 's starting position**,
- $y$  can follow  $x$ , and
- digits found by hint  $x$  **after  $y$ 's starting position** appear in the same relative order in hint  $y$ .



We say hint  $x$  is able to *transition into*  $y$  at index  $i$ , if:

- $x$  has found  $i$  digits **up to  $y$ 's starting position**,
- $y$  can follow  $x$ , and
- digits found by hint  $x$  **after  $y$ 's starting position** appear in the same relative order in hint  $y$ .

Key observation: When  $x$  transitions into  $y$ , no matter what integer sequence we find that generates a given hint  $y$ , it will also complete hint  $x$ .



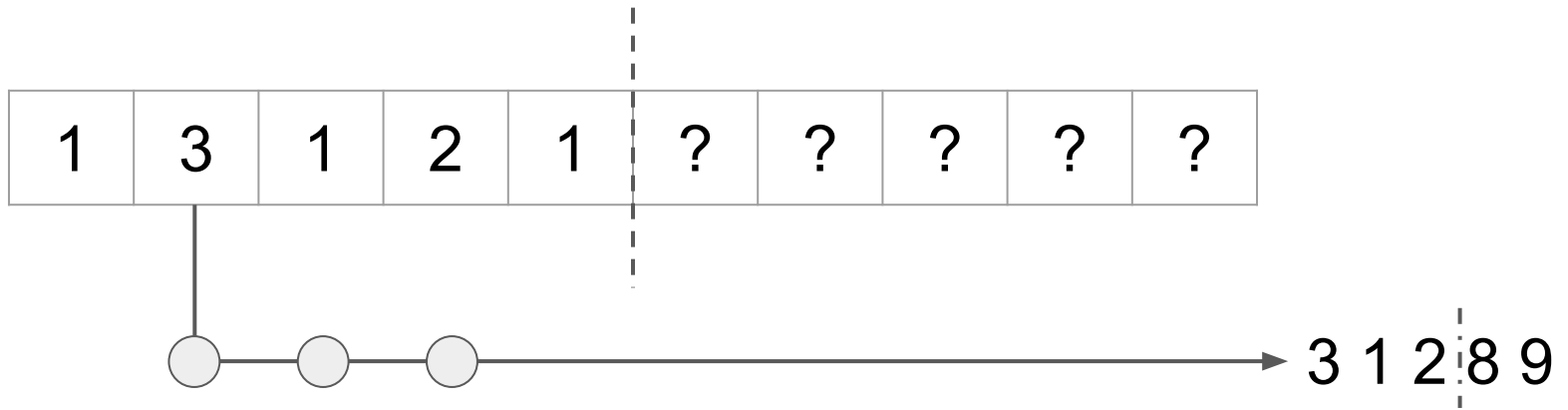
This leads to a dynamic programming solution:

We build the sequence left to right.

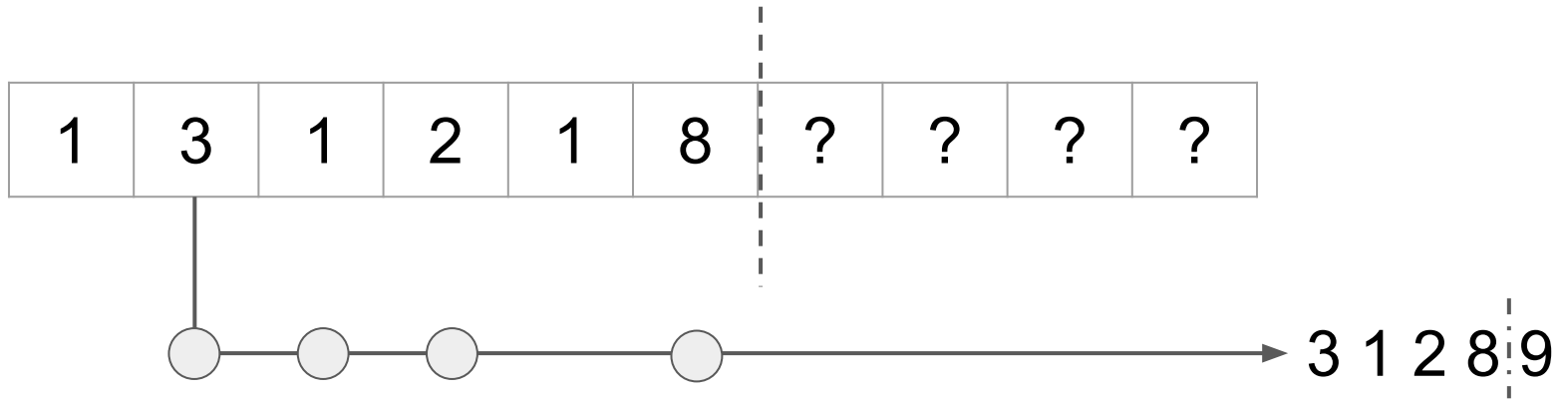
At a given state we only care about:

- Current hint - the last hint we've transitioned into.
- The index in the current hint - how many digits in that hint we've found already.
- Unused - bitmask of hints we have yet to transition into.

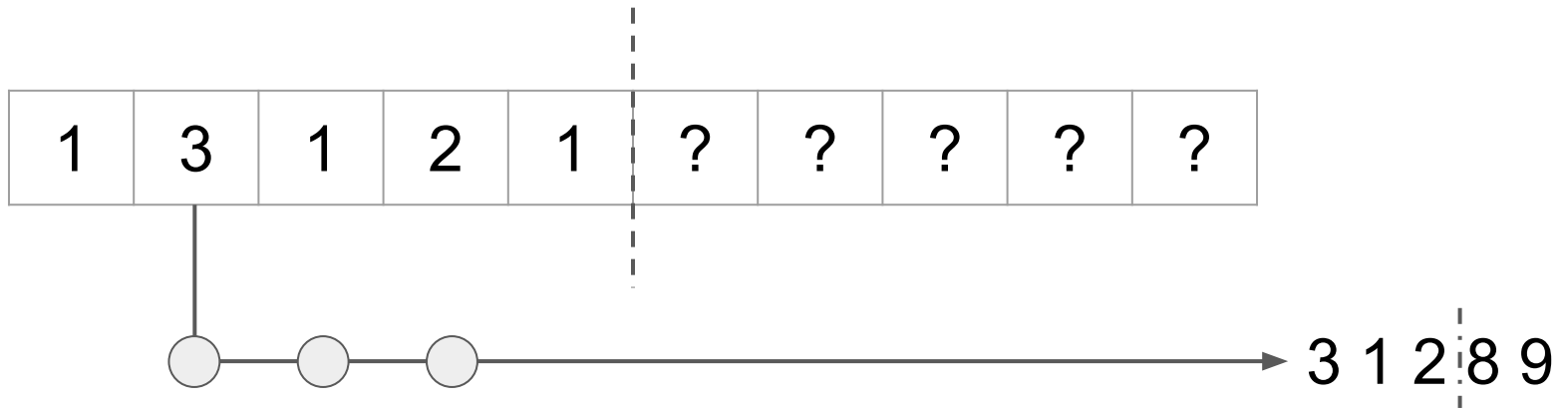




We try every digit that doesn't contradict the current hint. 1, 2, 3 and 8 are valid digits to continue the current hint.

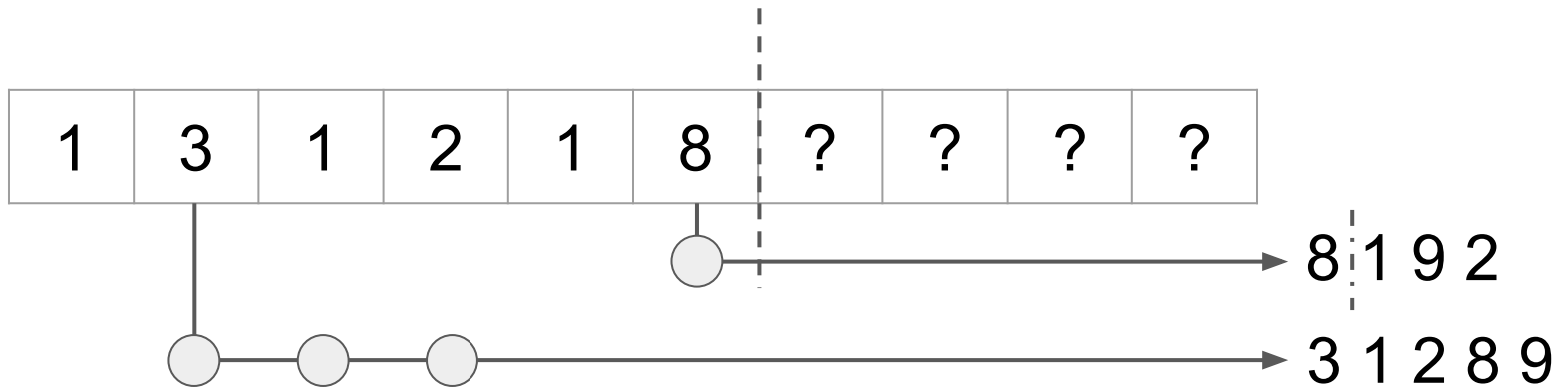


We try every digit that doesn't contradict the current hint. 1, 2, 3 and 8 are valid digits to continue the current hint.



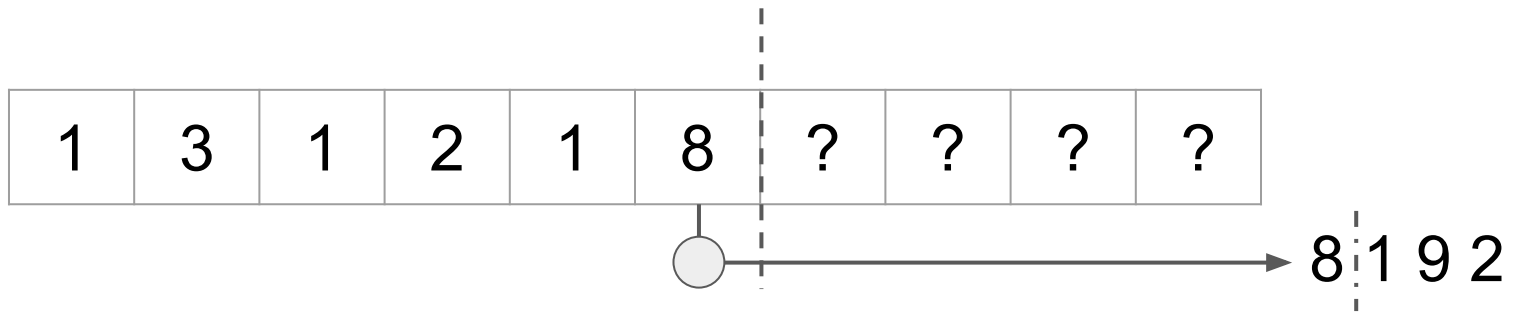
We try every digit that doesn't contradict the current hint. 1, 2, 3 and 8 are valid digits to continue the current hint.

We also need to try to transition into hints we have yet to start.



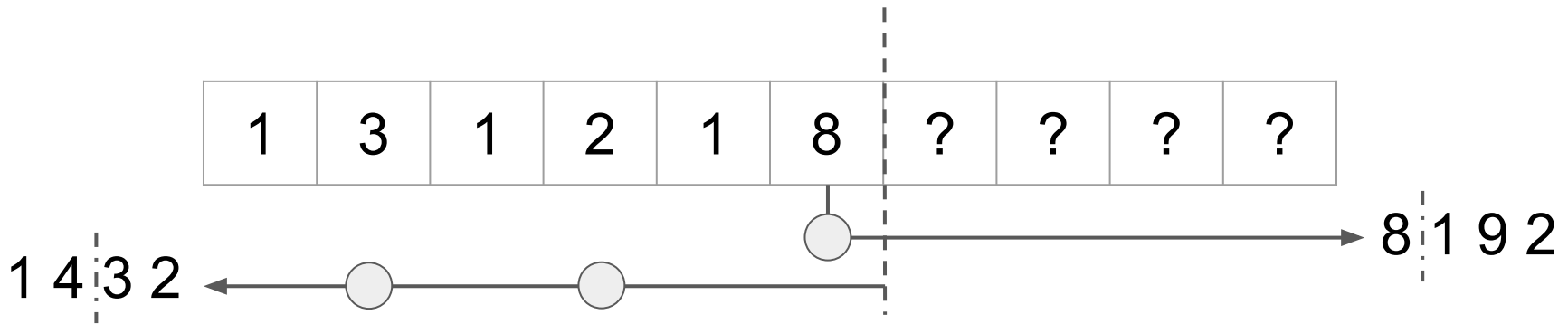
We try every digit that doesn't contradict the current hint. 1, 2, 3 and 8 are valid digits to continue the current hint.

We also need to try to transition into hints we have yet to start.



We try every digit that doesn't contradict the current hint. 1, 2, 3 and 8 are valid digits to continue the current hint.

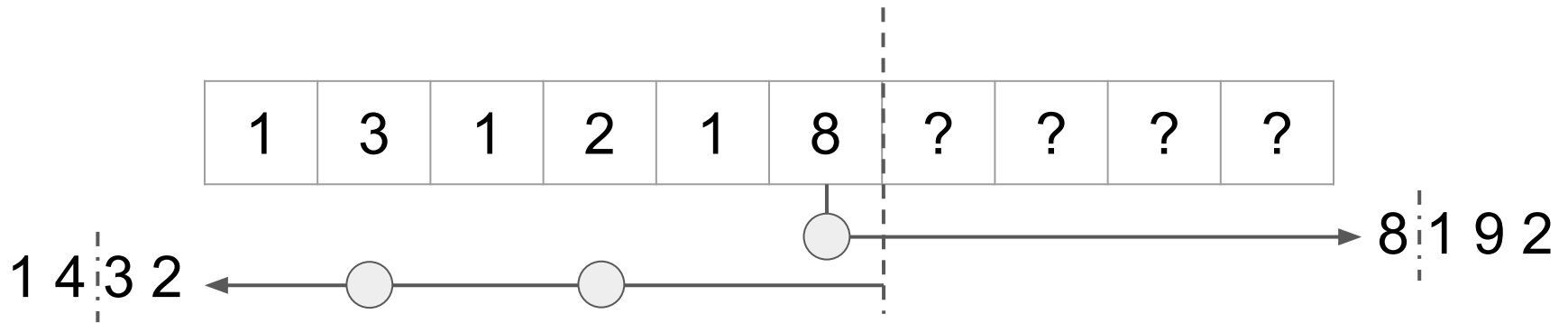
We also need to try to transition into hints we have yet to start. We can forget about the old hint, it will be completed by our new hint.



Finally, we reintroduce hints running left. Similar ideas apply, but we need to think in reverse.

We build a DP solution that keeps track:

- Right hint - the current hint running right.
- The index in the right hint - how many leading digits in that hint we've found already.
- Left hint - the current hint running left.
- The index in the left hint - how many trailing digits in that hint will be found in the part of the sequence that we've already built.
- Unused - bitmask of unused hints.



Details left as an exercise :)

Complexity:  $O(2^n \cdot n^2 \cdot \max\_digit^2 \cdot (n + \max\_digit))$

**Thanks!**