CERC 2015: Presentation of solutions

University of Zagreb

A:	ASCII Addition	Easy
B:	Book Borders	Medium
C:	Cow Confinement	Very hard
D:	Digit Division	Easy
E:	Export Estimate	Medium
F:	Frightful Formula	Hard
G:	Greenhouse Growth	Hard
H:	Hovering Hornet	Easy
I:	Ice Igloos	Medium
J:	Juice Junctions	Hard
K:	Kernel Knights	Easy
L:	Looping Labyrinth	Hard

Problem A
ASCII Addition

Submits: 86
Accepted: at least 59

First solved by:
University of Warsaw 1
(Wojciech Nadara, Marcin Smulewicz, Marek Sokołowski)
00:14:47

Author: Luka Kalinovčić

```
....x.xxxxx.xxxxx.x...x.xxxxx.xxxxx.xxxxx.......xxxxx.xxxxx.xxxxx
....x.....x.....x.x...x.x.....x.........x...x...x...x.x...x.x...x
....x.....x.....x.x...x.x.....x.........x...x...x...x.x...x.x...x
....x.xxxxx.xxxxx.xxxxx.xxxxx.xxxxx.....x.xxxxx.xxxxx.xxxxx.x...x
....x.x.........x.....x.....x.x...x.....x...x...x...x.....x.x...x
....x.x.........x.....x.....x.x...x.....x...x...x...x.....x.x...x
....x.xxxxx.xxxxx.....x.xxxxx.xxxxx.....x.......xxxxx.xxxxx.xxxxx
```

Three obvious steps:

- Convert the input ASCII art into a string.
- Parse the operands from the string.
- Convert the sum of operands to output ASCII art.

Use sample test data to avoid typing in individual matrices.

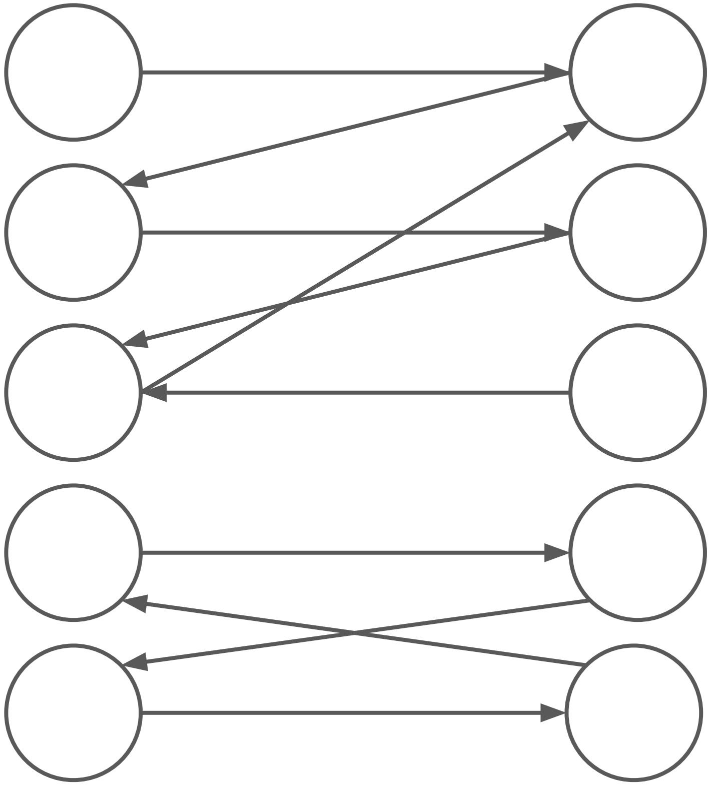
Problem K

Kernel Knights

Submits: 159
Accepted: at least 36

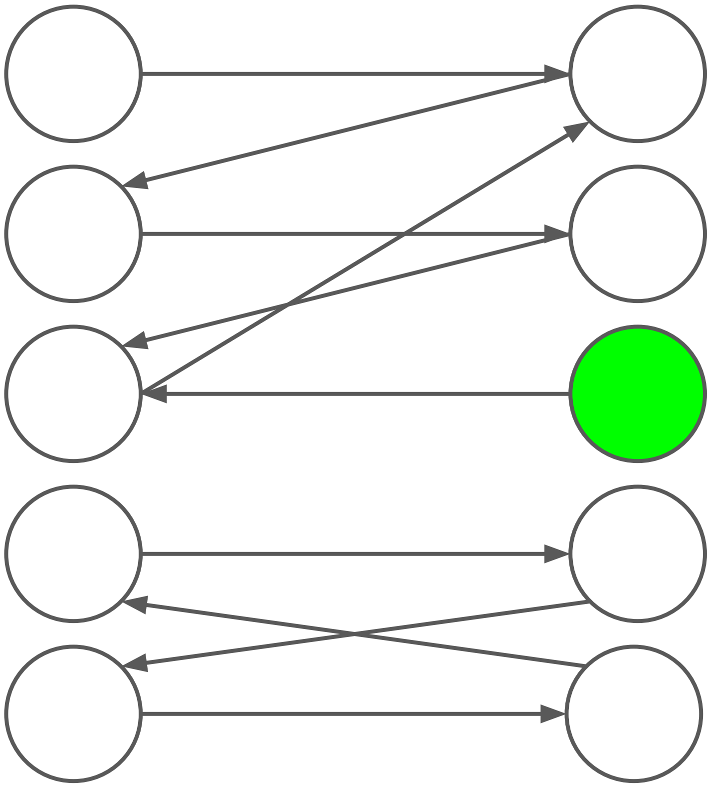
First solved by:
University of Warsaw 3
(Kamil Dębowski, Błażej Magnowski, Marek Sommer)
00:26:33

Author: Adrian Satja Kurdija

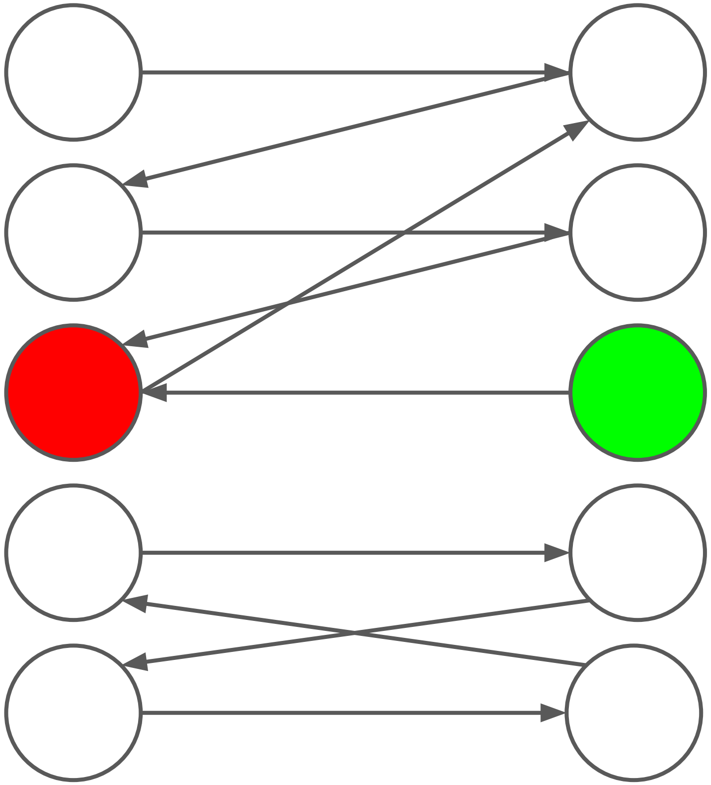


A *kernel* is defined as some subset *S* of knights with the following two properties:

- No knight in S was challenged by another knight in S.
- Every knight not in S was challenged by some knight in S.

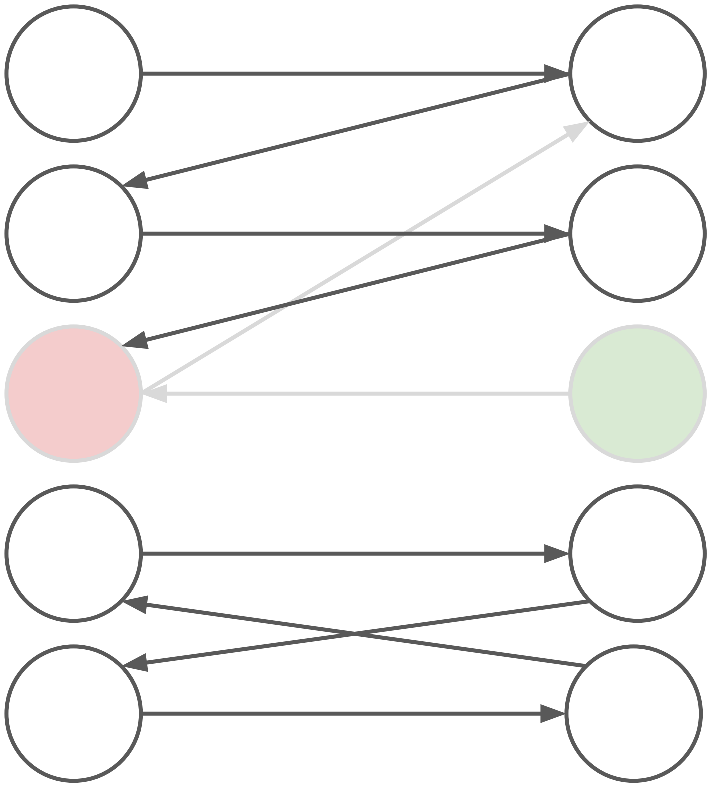


Knight A that nobody challenged must be in the kernel.



Knight A that nobody challenged must be in the kernel.

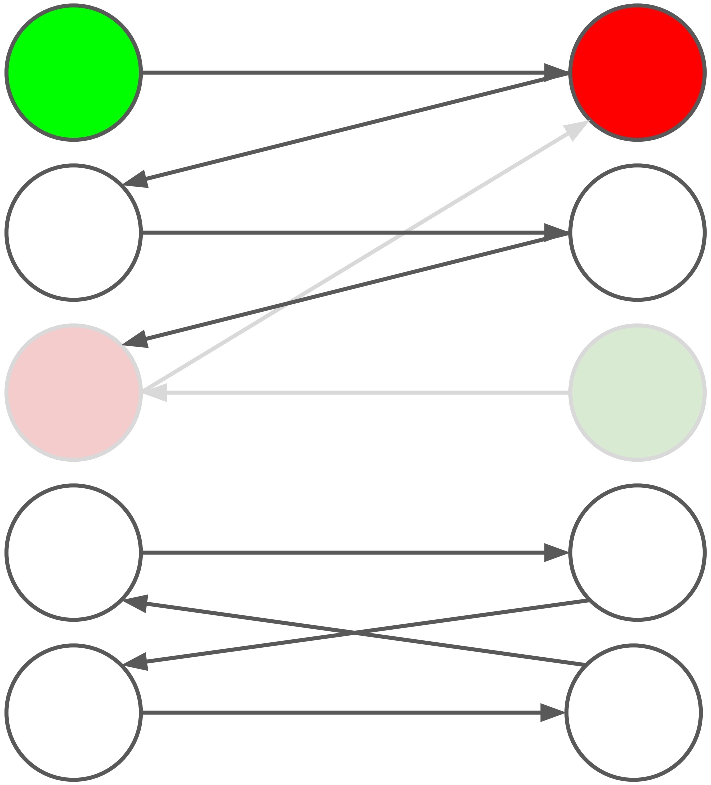
Knight B that A had challenged can’t be in the kernel.



Knight A that nobody challenged must be in the kernel.

Knight B that A had challenged can’t be in the kernel.

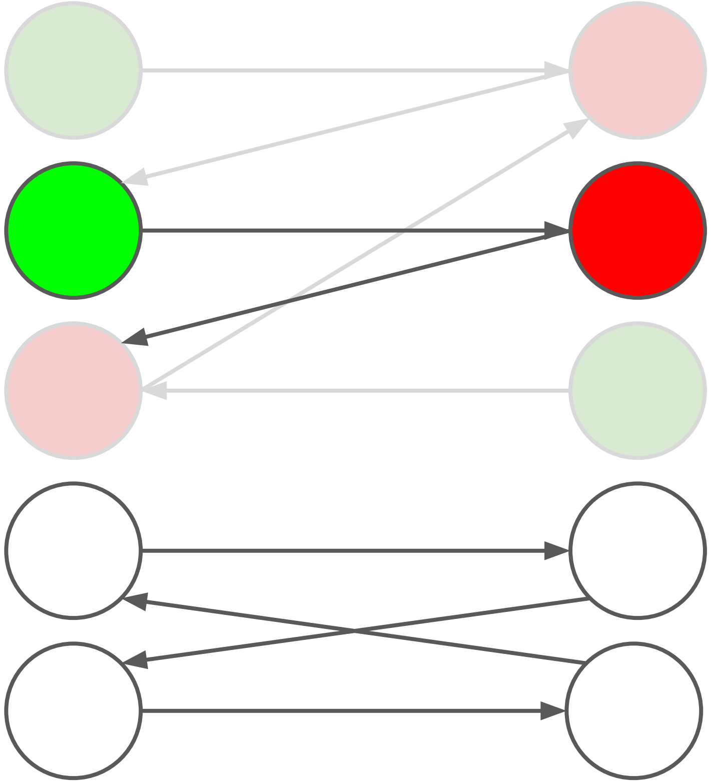
We no longer have to look at A or B.



Knight A that nobody challenged must be in the kernel.

Knight B that A had challenged can’t be in the kernel.

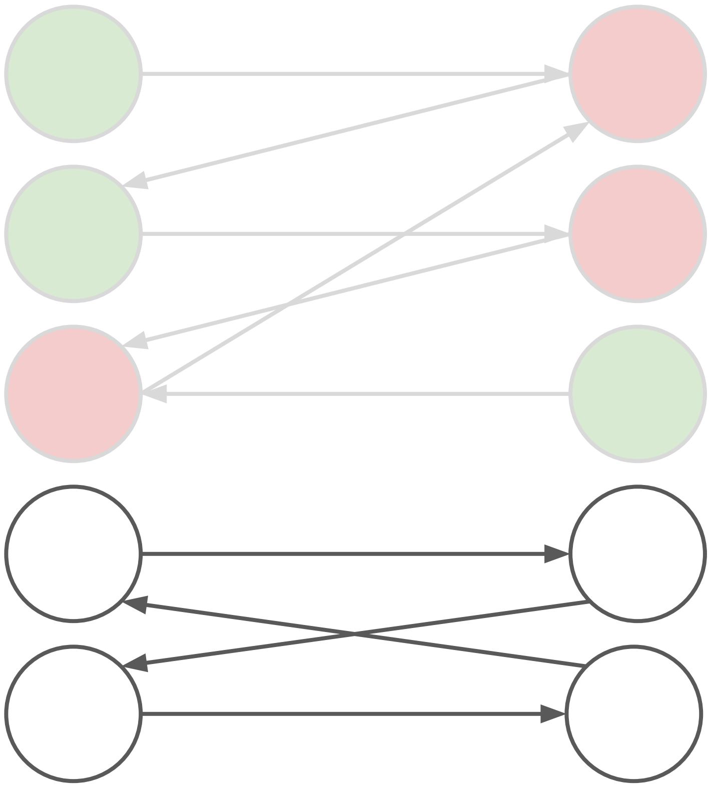
We no longer have to look at A or B.



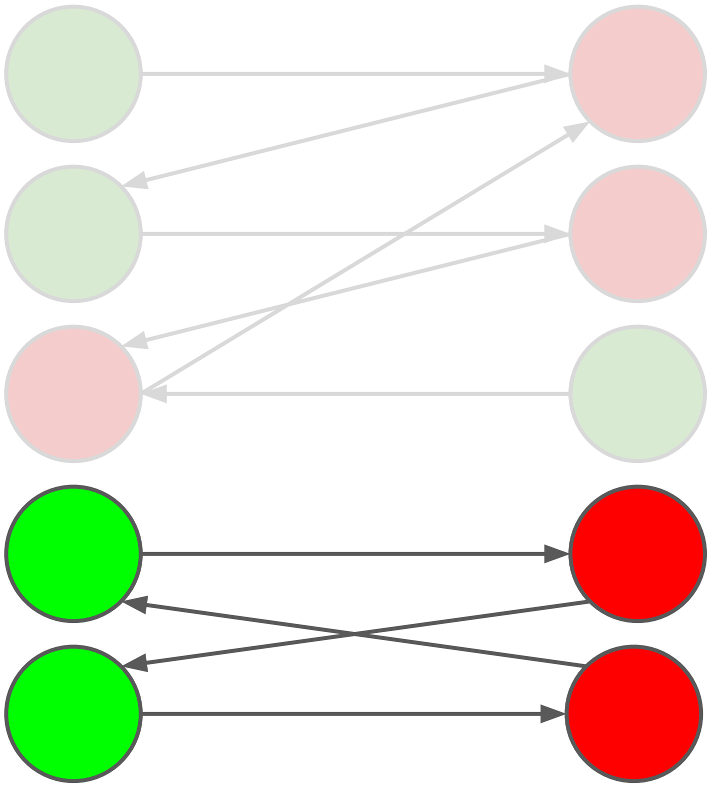
Knight A that nobody challenged must be in the kernel.

Knight B that A had challenged can’t be in the kernel.

We no longer have to look at A or B.

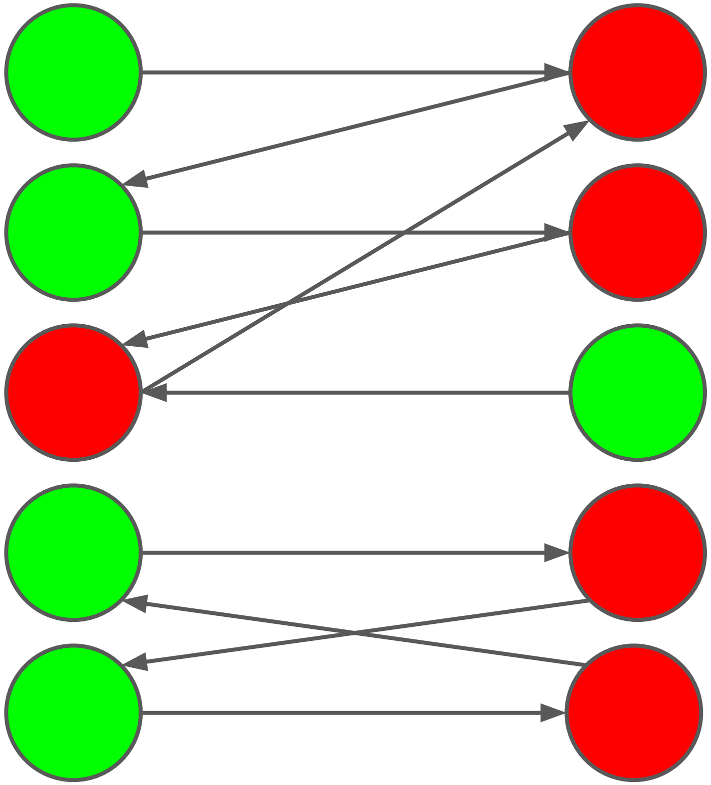


We are left with even-length cycles.



We are left with even-length cycles.

Simply select all knights from the same side.



We are left with even-length cycles.

Simply select all knights from the same side.

Done!

Problem D
Digit Division

Submits: 92
Accepted: at least 40

First solved by:
University of Warsaw 3
(Kamil Dębowski, Błażej Magnowski, Marek Sommer)
00:18:47

Author: Ivan Katanić

1 2 | 7 1 1 | 6 | 4 8

The problem requires a partition such that every group is a number divisible by m.

Key observation:
This is equivalent to the requirement that every prefix cut is a number divisible by m.

1 2 | 7 1 1 | 6 | 4 8

The problem requires a partition such that every group is a number divisible by m.

Key observation:
This is equivalent to the requirement that every prefix cut is a number divisible by m.

1 2 7 1 1 | 6 | 4 8

The problem requires a partition such that every group is a number divisible by m.

Key observation:
This is equivalent to the requirement that every prefix cut is a number divisible by m.

1 2 7 1 1 6 | 4 8

The problem requires a partition such that every group is a number divisible by m.

Key observation:
This is equivalent to the requirement that every prefix cut is a number divisible by m.

1 2 7 1 1 6 4 8

The problem requires a partition such that every group is a number divisible by m.

Key observation:
This is equivalent to the requirement that every prefix cut is a number divisible by m.

1 2 | 7 1 1 | 6 | 4 8

Take first two groups and concatenate them:

$$\text{concat}(A, B) = A * 10^{\text{num\_digits}(B)} + B$$

1 2 7 1 1 | 6 | 4 8

Take first two groups and concatenate them:

$\text{concat}(A, B) = A * 10^{\text{num\_digits}(B)} + B$

1 2 7 1 1 6 | 4 8

Take first two groups and concatenate them:

$\text{concat}(A, B) = A * 10^{\text{num\_digits}(B)} + B$

1 2 7 1 1 6 4 8

Take first two groups and concatenate them:

$$\text{concat}(A, B) = A * 10^{\text{num\_digits}(B)} + B$$

1 2 | 7 1 1 6 4 8

1 2 7 1 1 | 6 4 8

1 2 7 1 1 6 | 4 8

The algorithm:

Find all valid cut positions $\{p_1, p_2, \dots, p_n\}$.

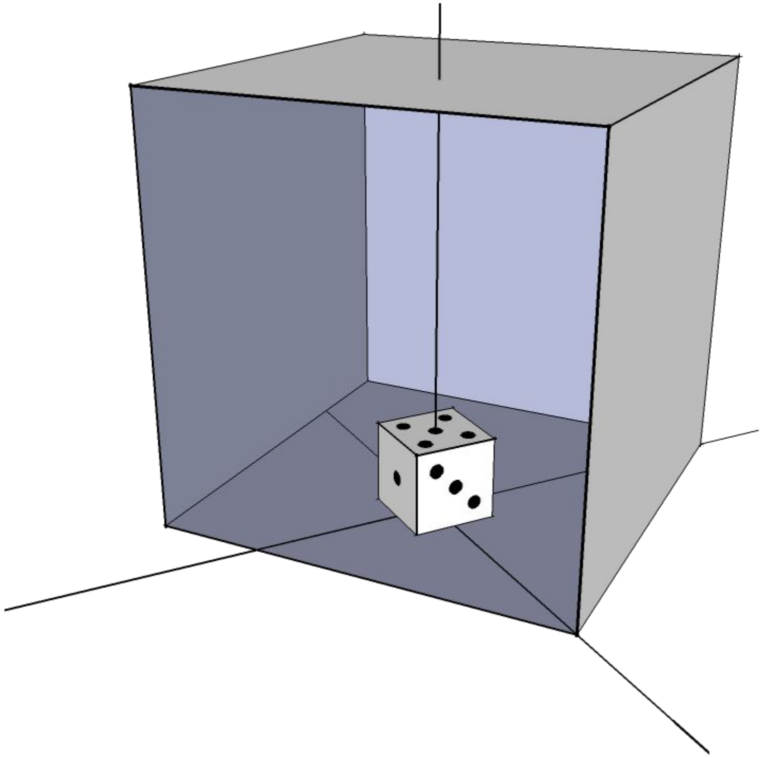
The result is 2^n (mod $10^9 + 7$).

Problem H
Hovering Hornet

Submits: 62
Accepted: at least 13

First solved by:
University of Warsaw 4
(Patryk Czajka, Karol Farbiś, Krzysztof Pszeniczny)
01:08:55

Author: Luka Kalinovčić

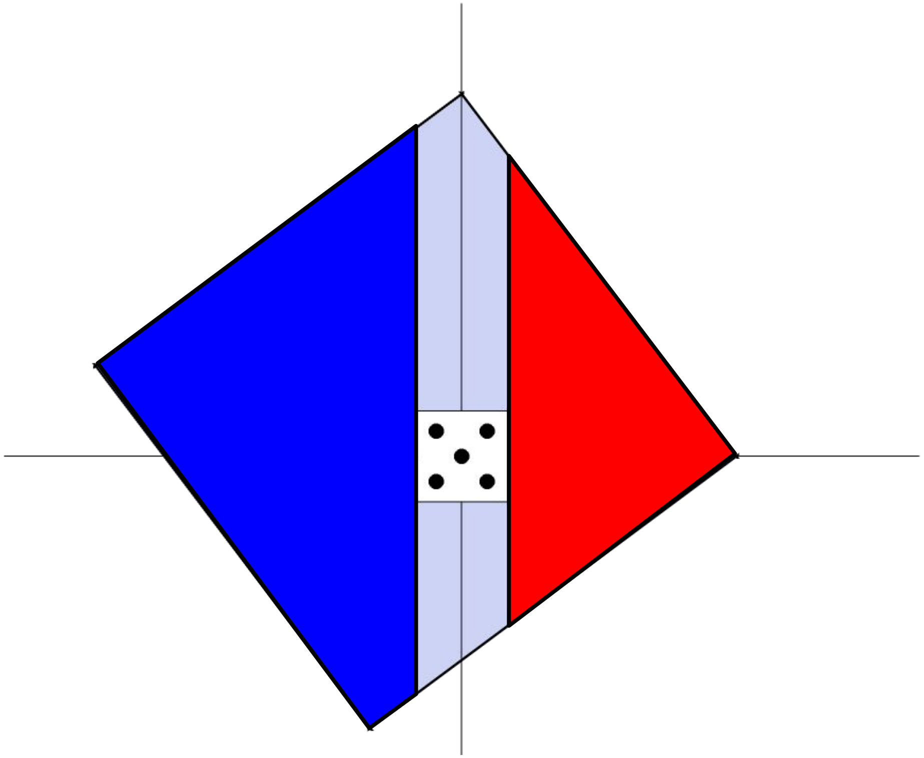


Expected value:

p(1) * 1 + p(2) * 2 + p(3) * 3 + p(4) * 4 + p(5) * 5 + p(6) * 6

p(2) = 0

p(5) = (4 * 5 * 5) / (5 * 5 * 5 - 1)

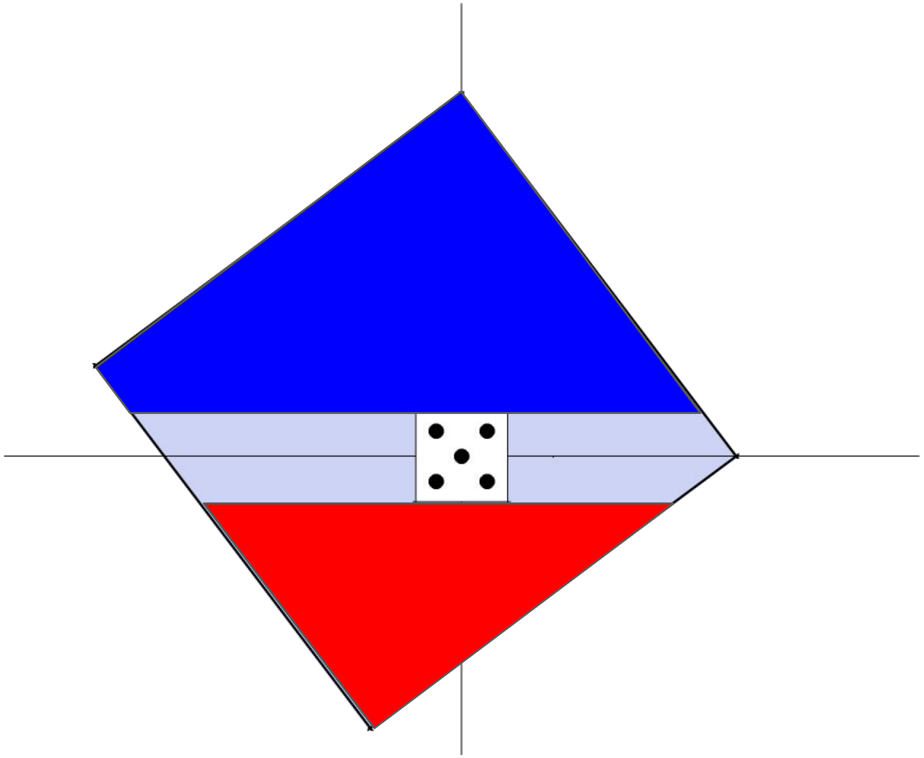


Expected value:

p(1) * 1 + p(2) * 2 + p(3) * 3 + p(4) * 4 + p(5) * 5 + p(6) * 6

$p(3) = (5 * a_3) / (5 * 5 * 5 - 1)$

$p(4) = (5 * a_4) / (5 * 5 * 5 - 1)$



Expected value:

$p(1) * 1 + p(2) * 2 + p(3) * 3 + p(4) * 4 + p(5) * 5 + p(6) * 6$

$p(1) = (5 * a_1) / (5 * 5 * 5 - 1)$

$p(6) = (5 * a_6) / (5 * 5 * 5 - 1)$

Problem B
Book Borders

Submits: 101
Accepted: at least 28

First solved by:
University of Zagreb 1
(Mislav Bradač, Dominik Gleich, Gustav Matula)
00:48:53

Author: Ivan Katanić

```
|its.a.long...|     |its.a.long.way|
|way.to.the...|     |to.the.top.if.|
|top.if.you...|     |you.wanna.rock|
|wanna.rock.n.|     |n.roll........|
|roll.........|
```

Start with a fixed maximum line length m.

Simulate typesetting algorithm, line-by-line.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
input_text(i)	i	t	s		a		l	o	n	g		w	a	y

Two helper functions:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
input_text(i)	i	t	s		a		l	o	n	g		w	a	y
word_length(i)	3	0	0	0	1	0	4	0	0	0	0	3	0	0

Two helper functions:

word_length(i) = the length of the word that starts at i-th position in the text.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
input_text(i)	i	t	s		a		l	o	n	g		w	a	y
word_length(i)	3	0	0	0	1	0	4	0	0	0	0	3	0	0
word_start(i)	0	0	0	4	4	6	6	6	6	6	11	11	11	11

Two helper functions:

word_start(i) =

- -1, if *i* exceeds the total length of the input text, or
- *i* + 1, if the character at position *i* is a space, or
- the position of the first character in the word that *i*-th character is a part of, otherwise.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
input_text(i)	i	t	s		a		l	o	n	g		w	a	y
word_length(i)	3	0	0	0	1	0	4	0	0	0	0	3	0	0
word_start(i)	0	0	0	4	4	6	6	6	6	6	11	11	11	11

word_start(p + m) gives us the position of the first word in the next line.

Example:

p = 0

m = 12

word_start(p + m) = 11

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
input_text(i)	i	t	s		a		l	o	n	g		w	a	y
word_length(i)	3	0	0	0	1	0	4	0	0	0	0	3	0	0
word_start(i)	0	0	0	4	4	6	6	6	6	6	11	11	11	11

word_start(p + m) gives us the position of the first word in the next line.

Example:

p = 0

m = 12

word_start(p + m) = 11

```
Solve(m):
    result = 0
    p = 0
    while p != -1:
        result += word_length(p) + 1
        p = word_start(p + m)
    return result - 1
```

```
|its.a.long...|
|way.to.the...|
|top.if.you...|
|wanna.rock.n.|
|roll.........|
```

Analysis:

Variable p advances by at least m positions in two iterations.

- Look at any two consecutive lines. The first word on the second line couldn't fit on the first line.
- O(z / m)

```
Solve():
    for m in [a, b]:
        output Solve(m)
```

Analysis:

$z / 1 + z / 2 + z / 3 + \ldots + z / z =$

$z * (1 / 1 + 1 / 2 + 1 / 3 + \ldots 1 / z) < z * (\ln z + 1)$

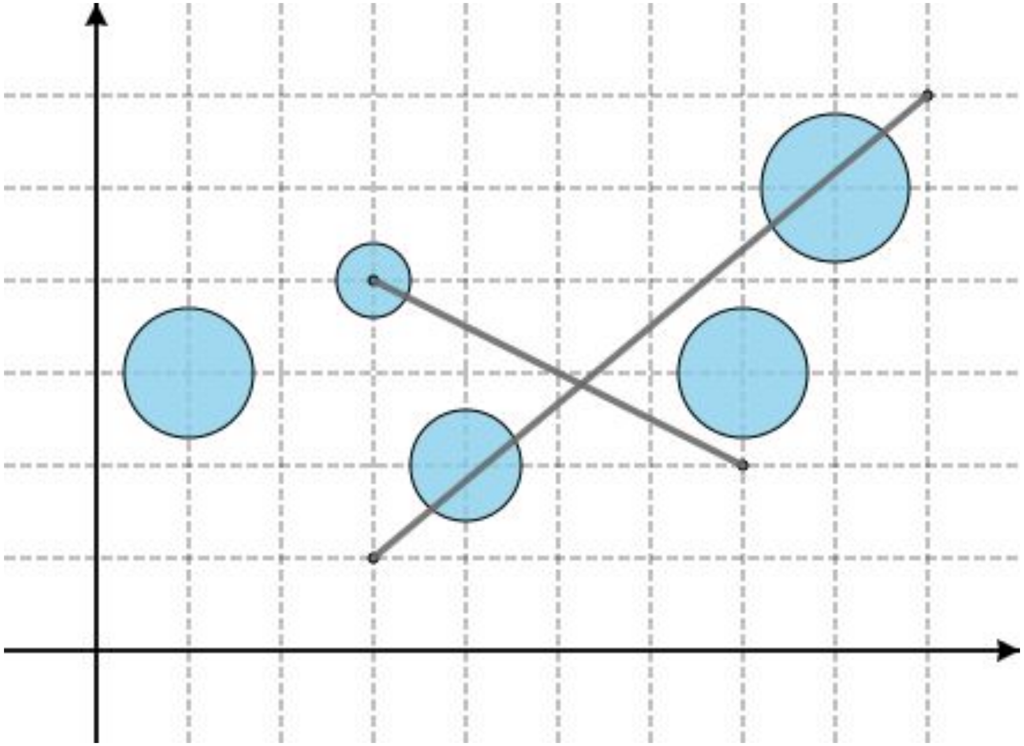
$O(z \log z)$

Problem I
Ice Igloos

Submits: 55
Accepted: at least 3

First solved by:
University of Warsaw 4
(Patryk Czajka, Karol Farbiś, Krzysztof Pszeniczny)
03:08:51

Author: Luka Kalinovčić

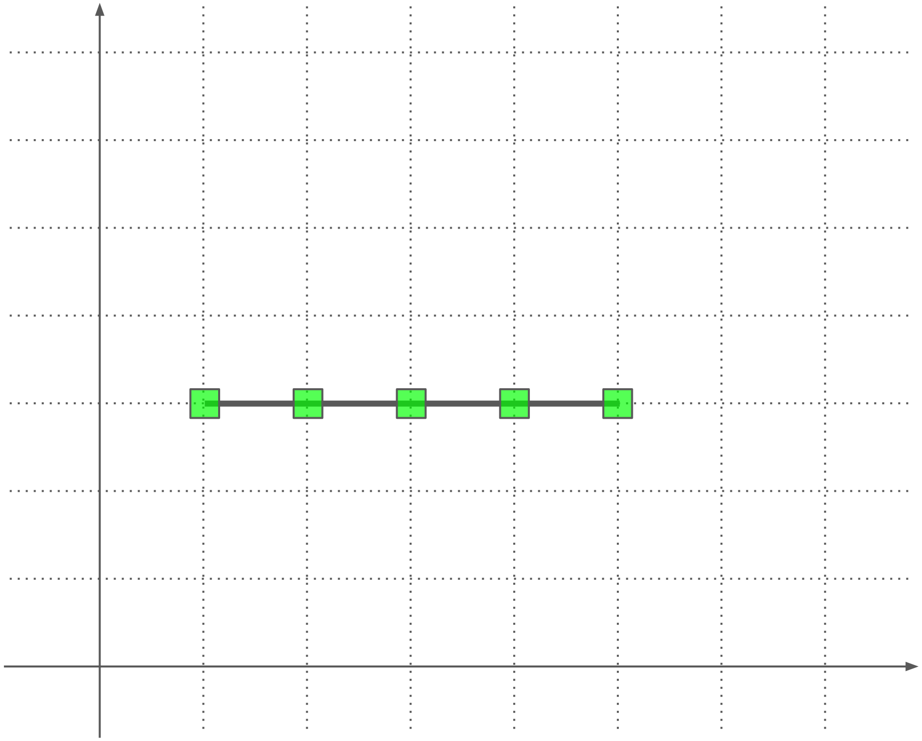


How to check whether segment intersects a circle?

distance(circle_center, segment) ≤ circle_radius

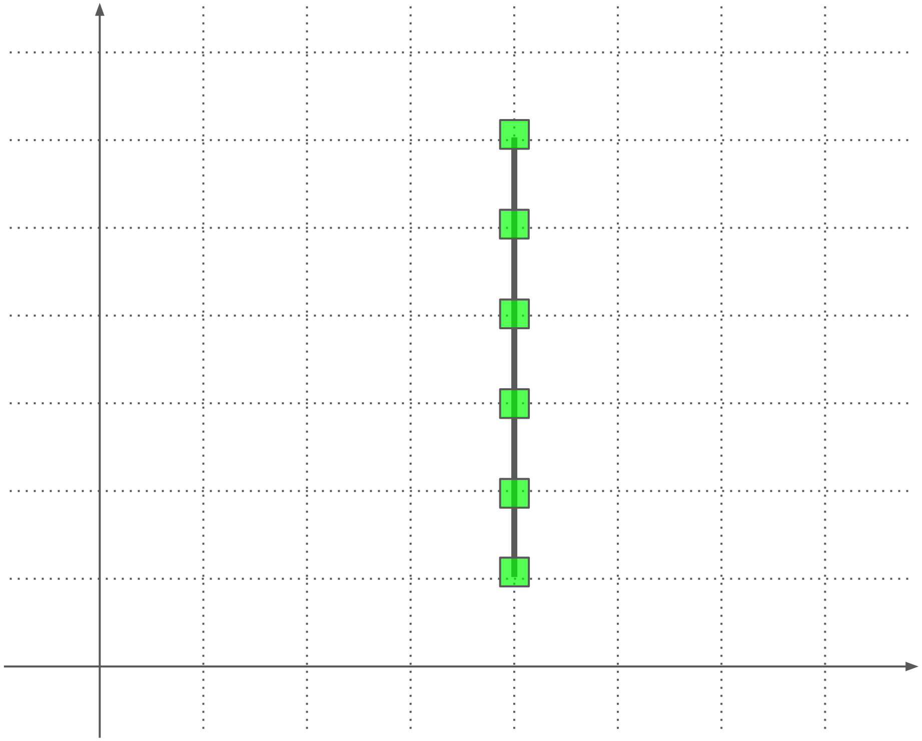
We can’t afford to check for every (circle, segment) pair.

Solution: Coordinates are small integers!



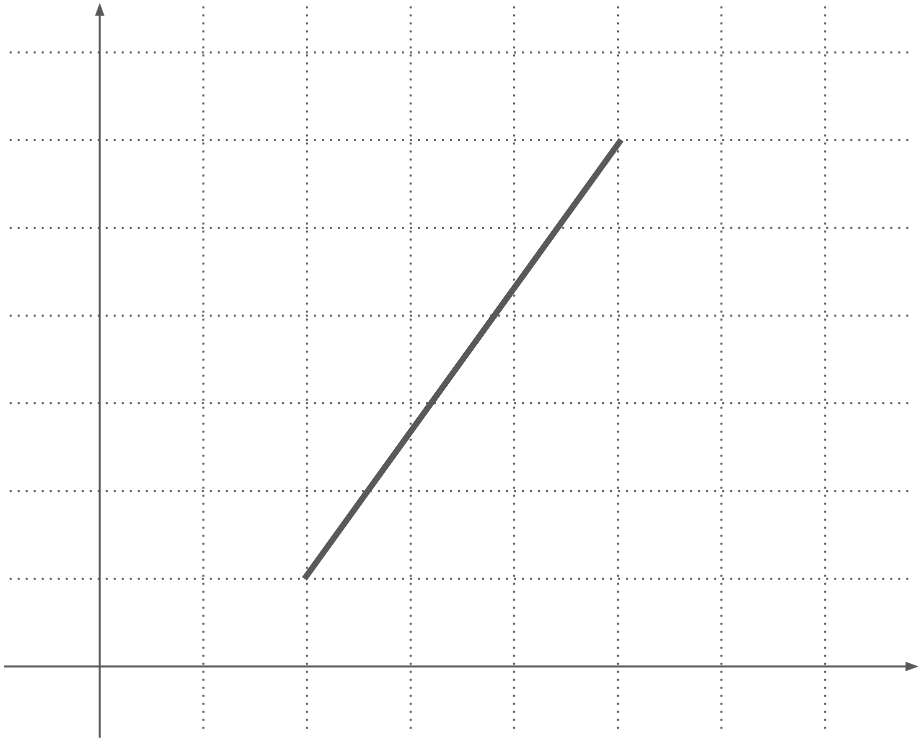
Horizontal segments are easy.

O(max_coords) igloo positions to consider.

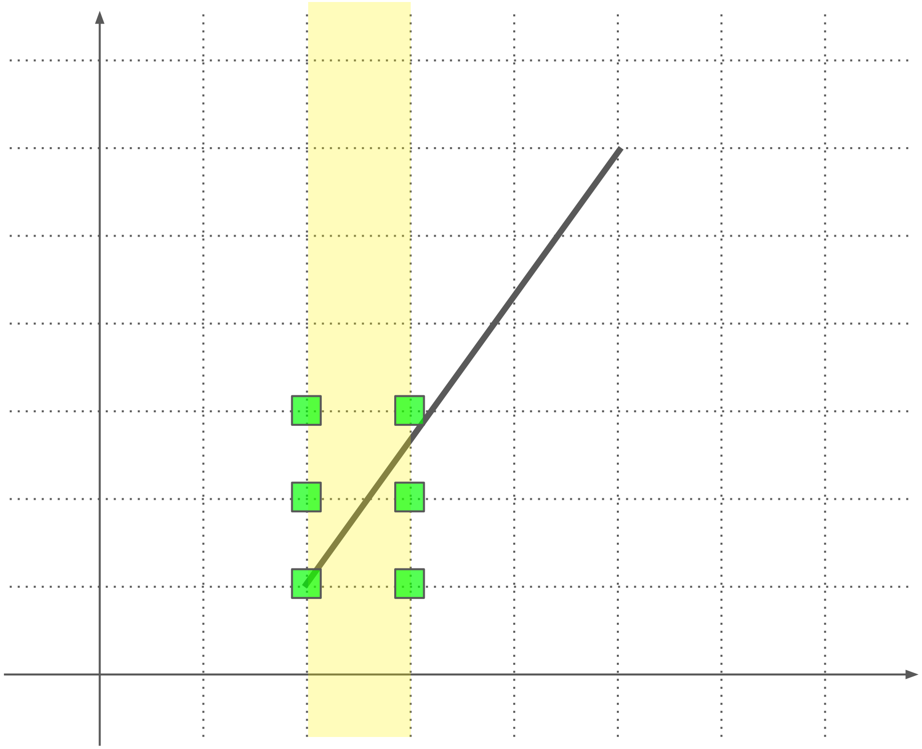


Vertical segments are easy.

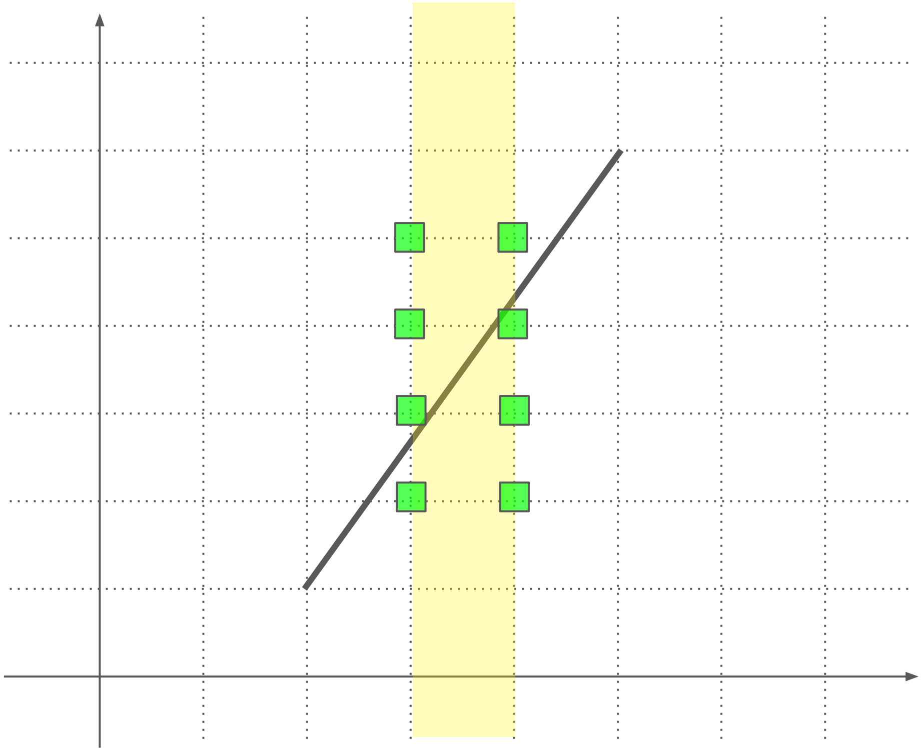
O(max_coords) igloo positions to consider.



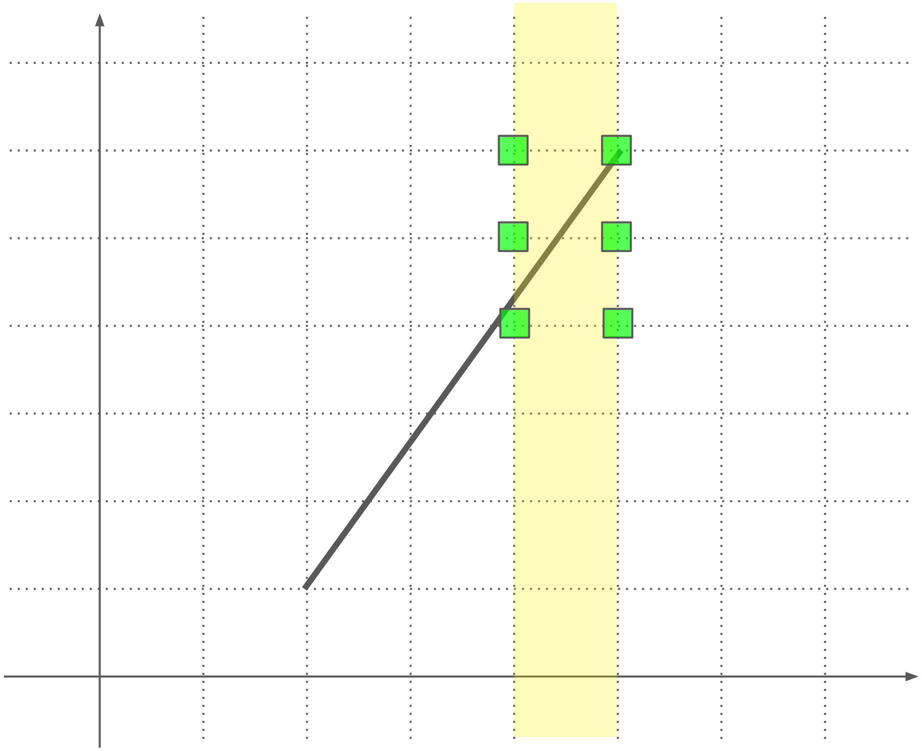
Process diagonal segments left-to-right to find relevant igloos. Think of it as a function y(x).



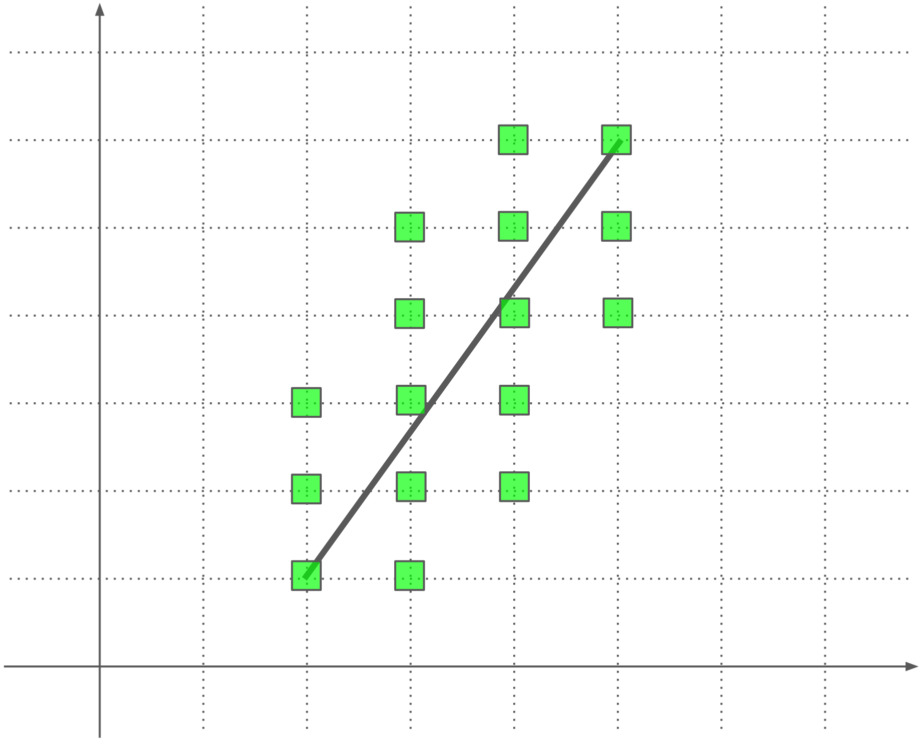
Process diagonal segments left-to-right to find relevant igloos. Think of it as a function y(x).



Process diagonal segments left-to-right to find relevant igloos. Think of it as a function y(x).



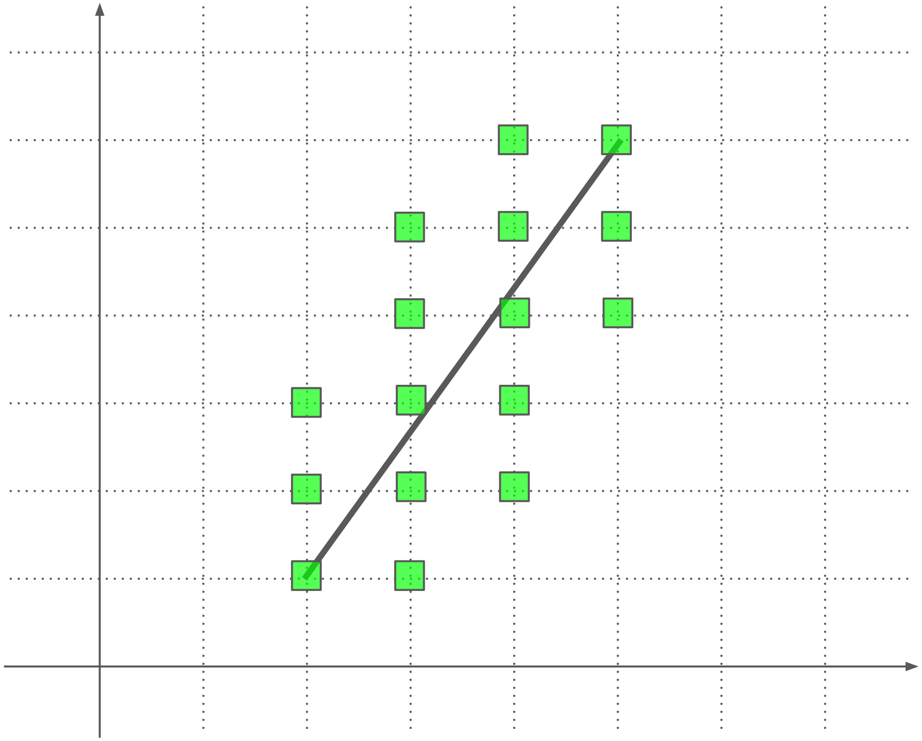
Process diagonal segments left-to-right to find relevant igloos. Think of it as a function y(x).



At a given coordinate x ($x_1 < x < x_2$), we consider igloos with y between floor(y(x - 1)) and ceil(y(x + 1)).

O(max_coords) igloo positions to consider.

Algorithm complexity: O(num_segments * max_coords)



Igloo position is within the (x_1, y_1) - (x_2, y_2) rectangle => distance(point, segment) == distance(point, line)

Avoid sqrt function by normalizing the line equation or squaring the inequality.

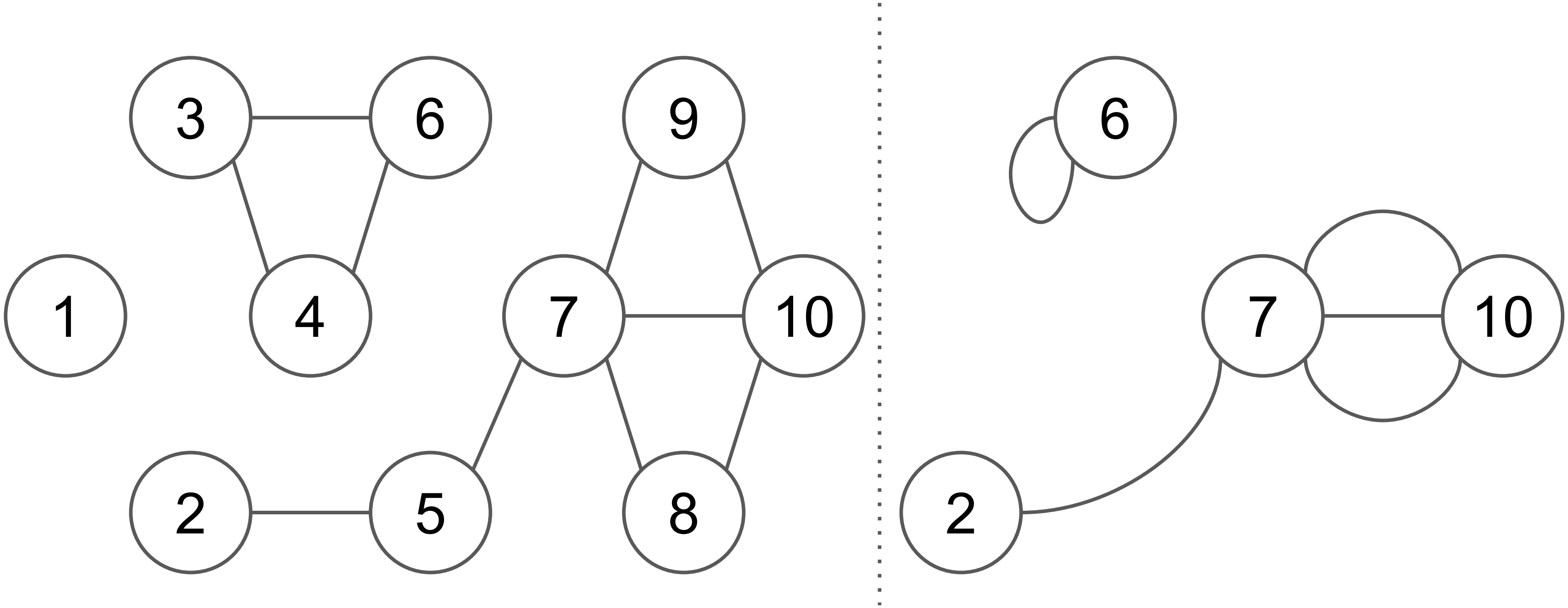
Problem E

Export Estimate

Submits: 26
Accepted: at least 3

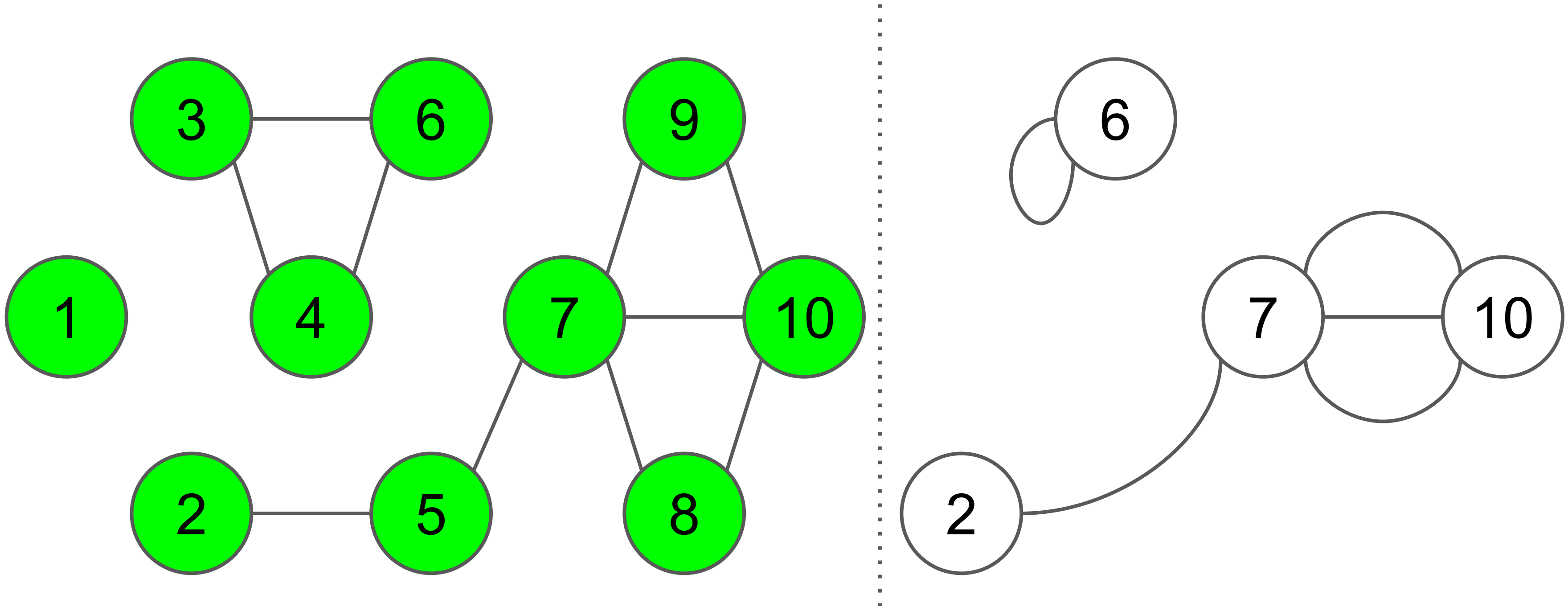
First solved by:
AGH University of Science and Technology 1
(Dawid Pawlak, Adam Szady, Jan Tułowiecki)
02:31:06

Author: Luka Kalinovčić



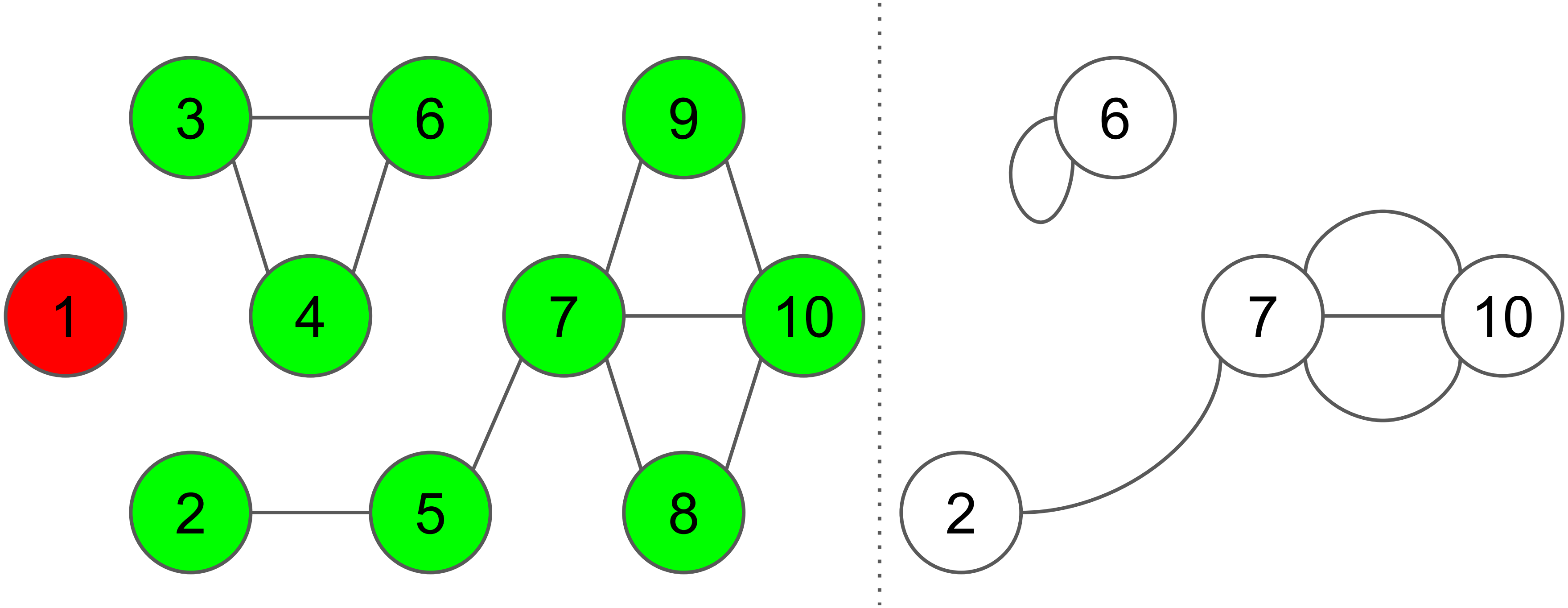
Assume there are no priorities yet.

What is the number of nodes and edges in the contracted graph?

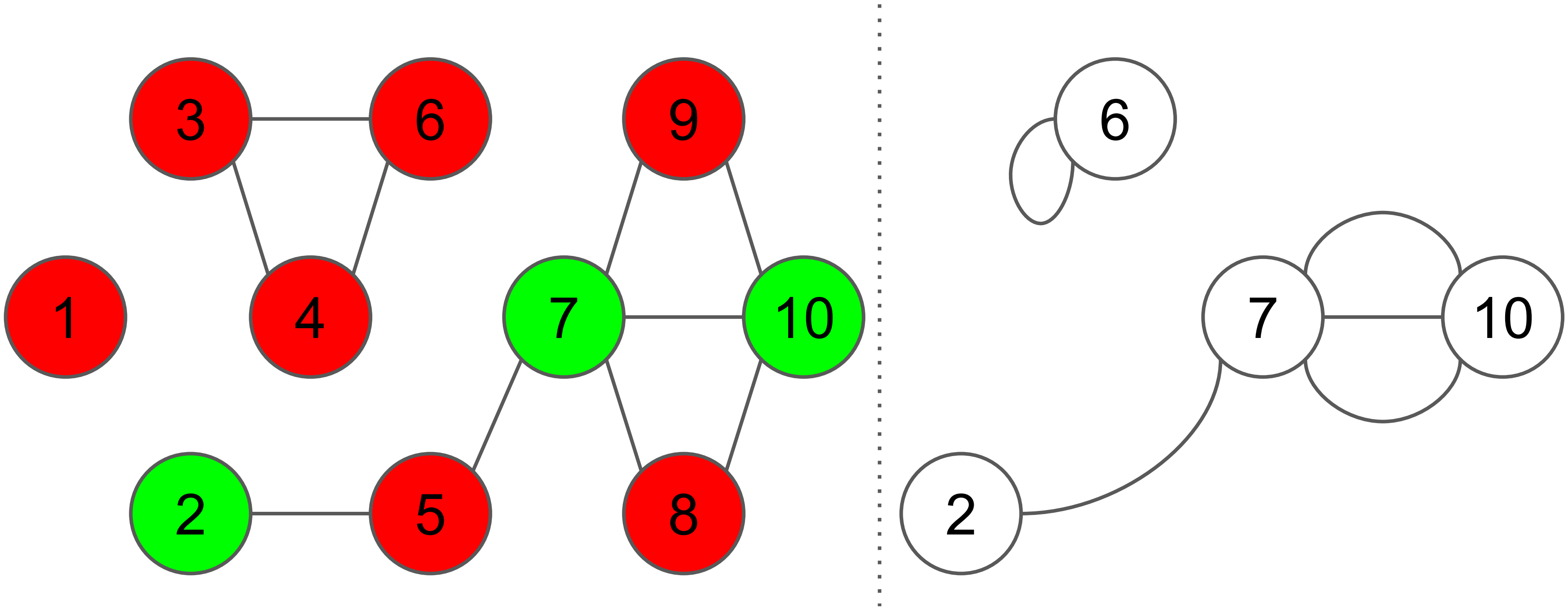


Nodes =

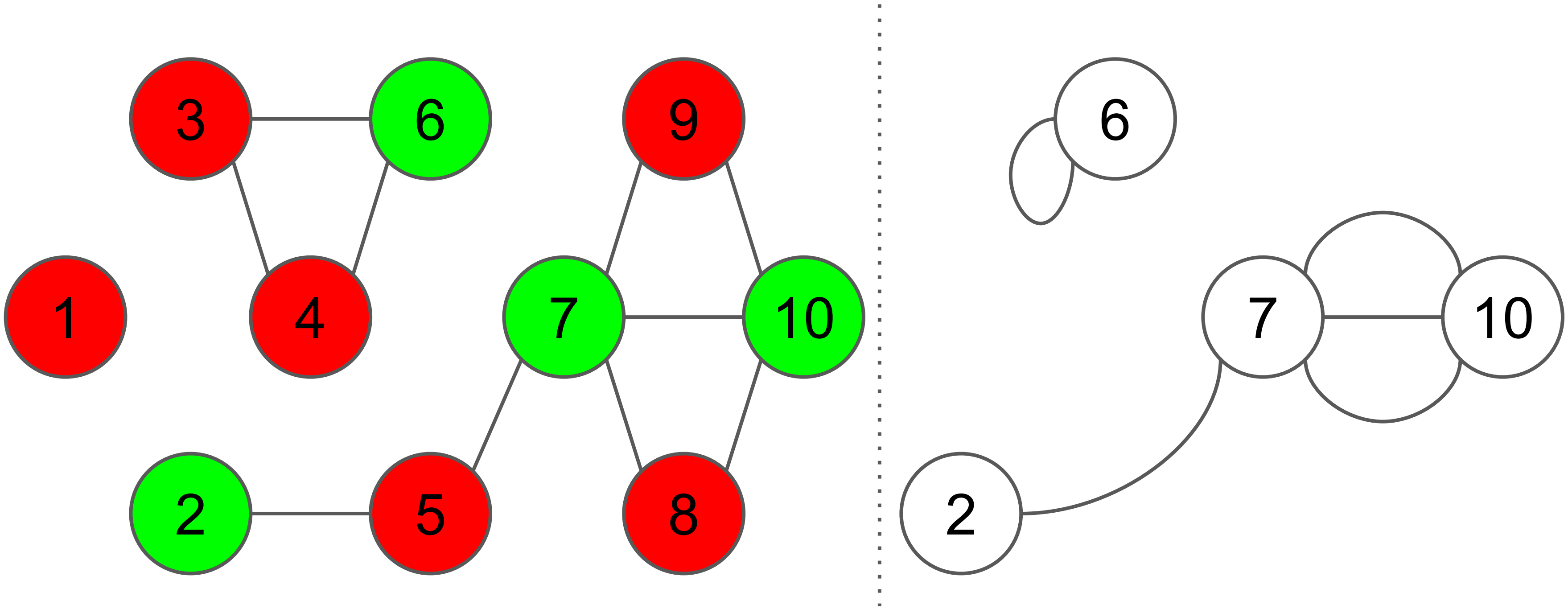
n?



Nodes =
 n - num_degree_0?



Nodes =
  n - num_degree_0 - num_degree_2?

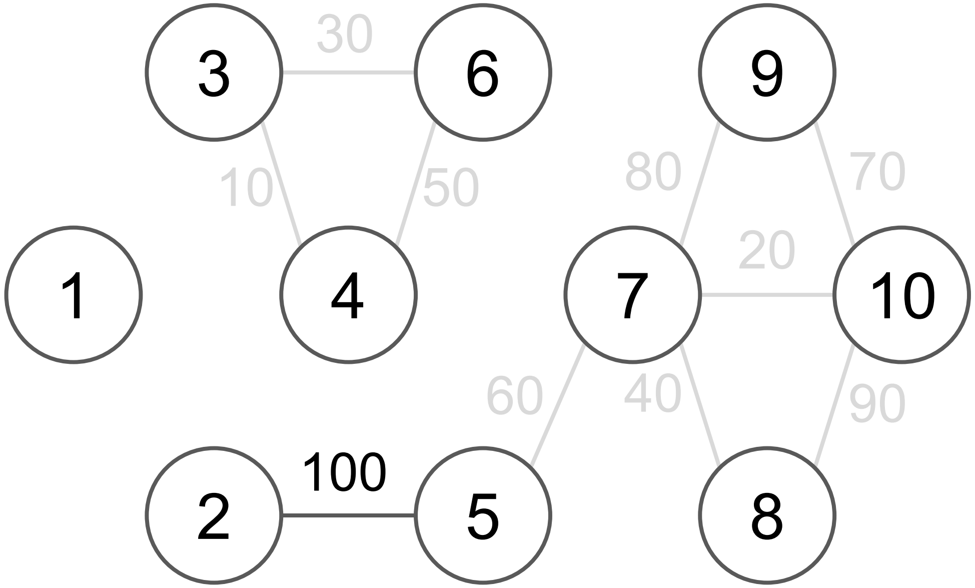


Nodes =

n - num_degree_0 - num_degree_2 + num_cycles

Edges =

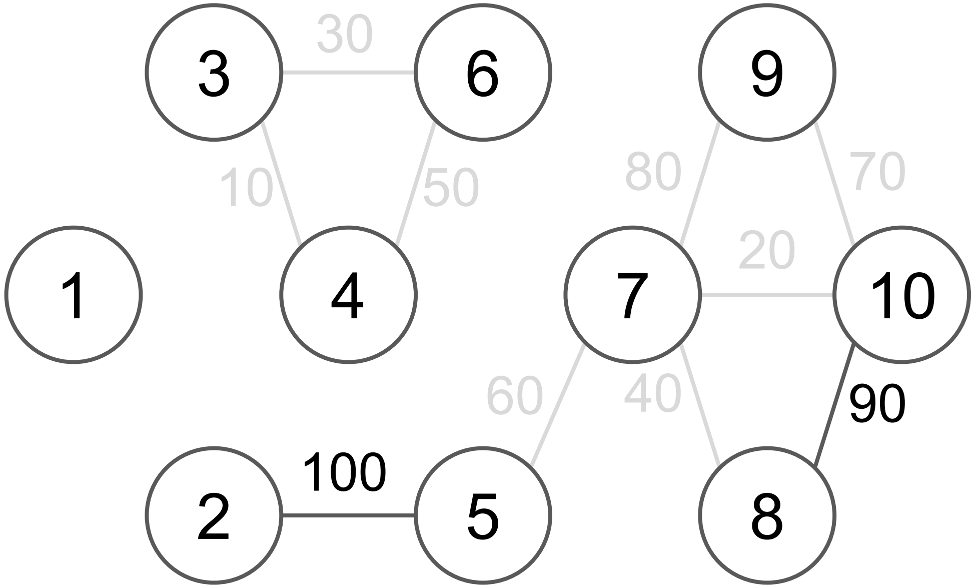
m - num_degree_2 + num_cycles



Reintroduce edge priorities.

We start with an empty graph, and add edges one-by-one ordered by decreasing priority.

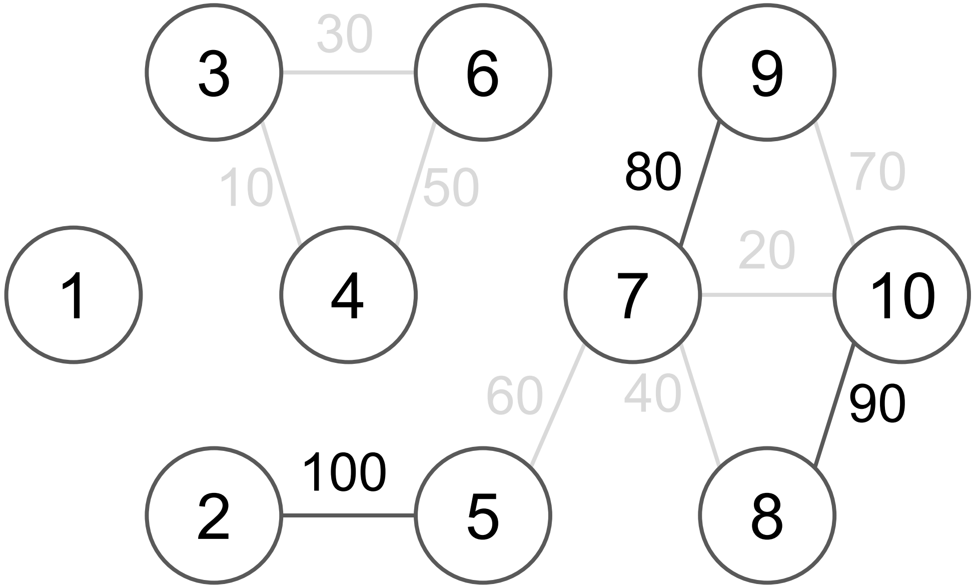
We answer export requests along the way.



Reintroduce edge priorities.

We start with an empty graph, and add edges one-by-one ordered by decreasing priority.

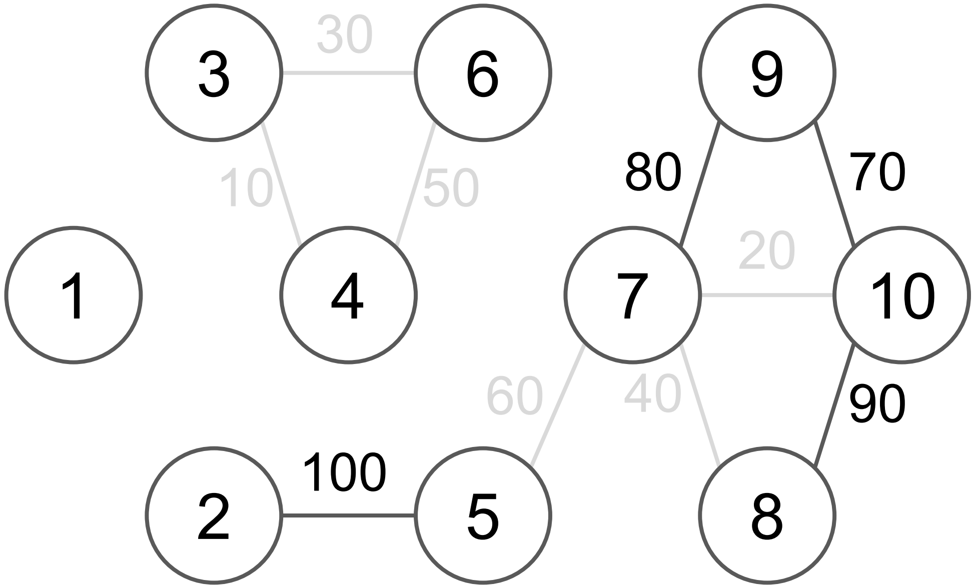
We answer export requests along the way.



Reintroduce edge priorities.

We start with an empty graph, and add edges one-by-one ordered by decreasing priority.

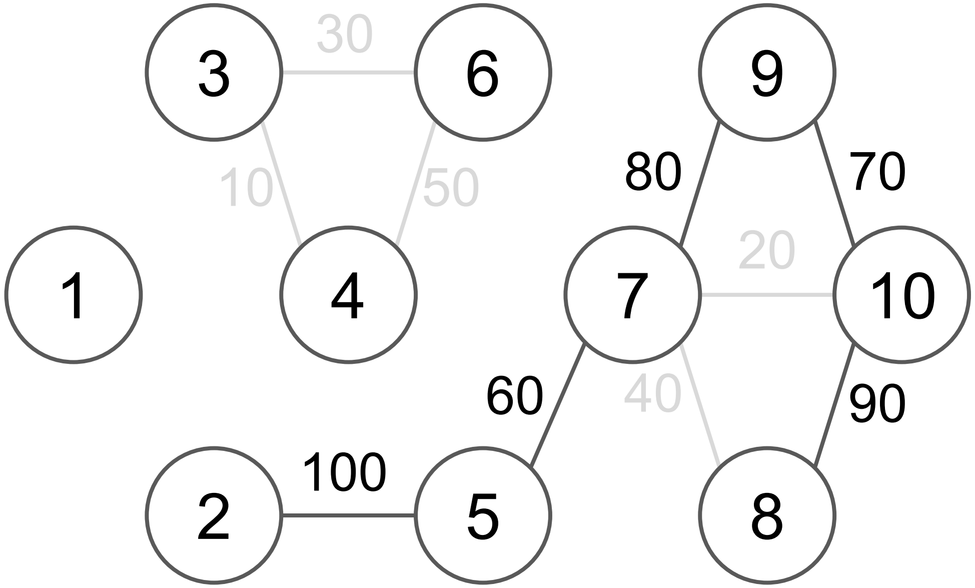
We answer export requests along the way.



Reintroduce edge priorities.

We start with an empty graph, and add edges one-by-one ordered by decreasing priority.

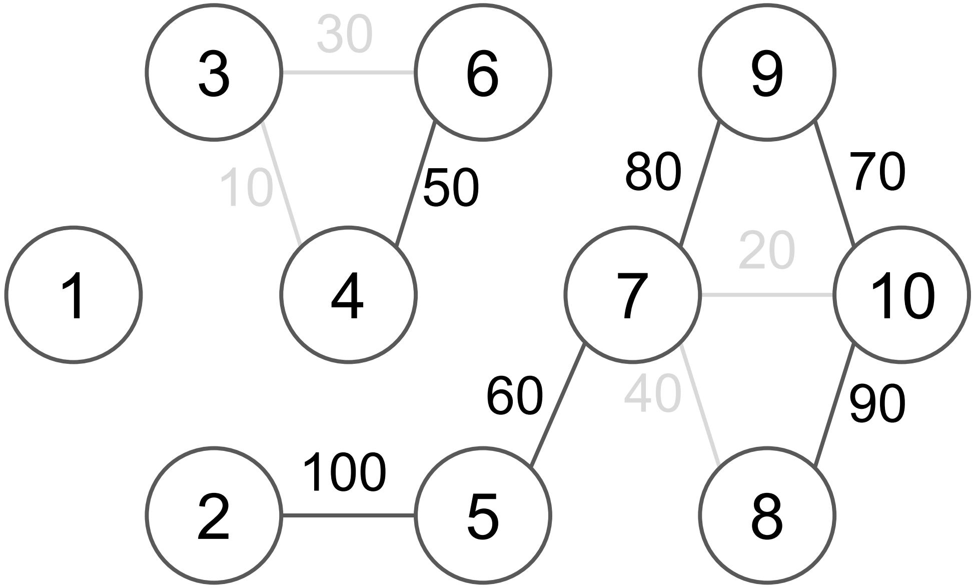
We answer export requests along the way.



Reintroduce edge priorities.

We start with an empty graph, and add edges one-by-one ordered by decreasing priority.

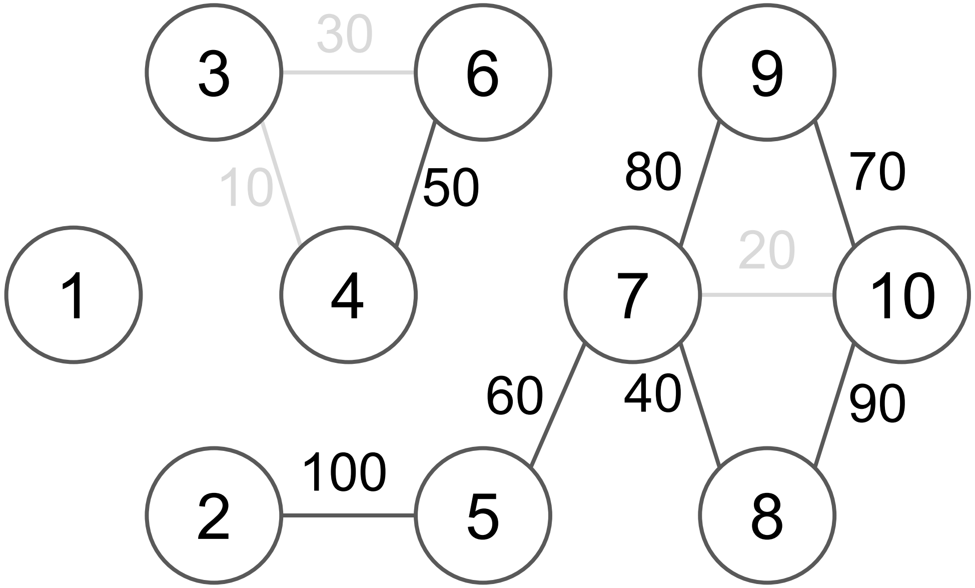
We answer export requests along the way.



Reintroduce edge priorities.

We start with an empty graph, and add edges one-by-one ordered by decreasing priority.

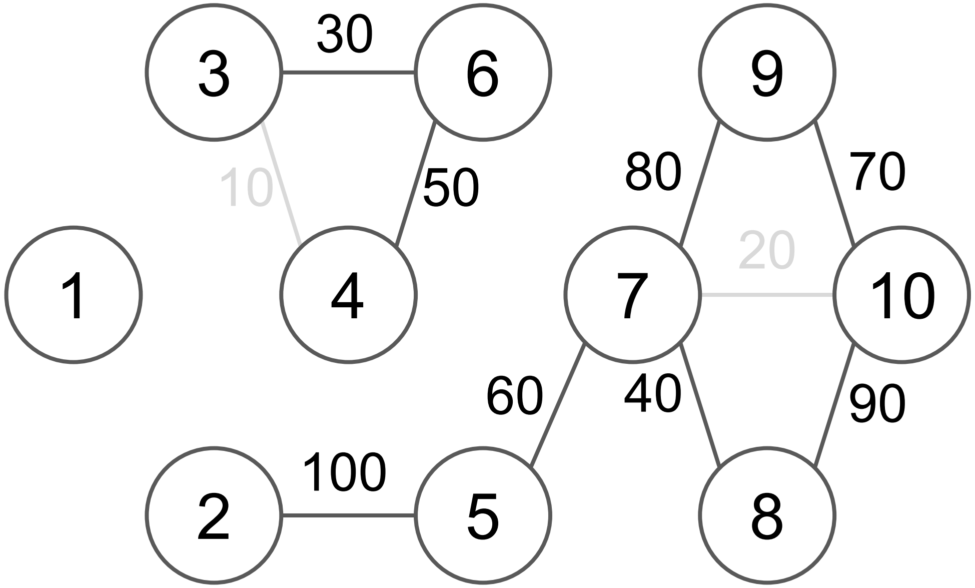
We answer export requests along the way.



Reintroduce edge priorities.

We start with an empty graph, and add edges one-by-one ordered by decreasing priority.

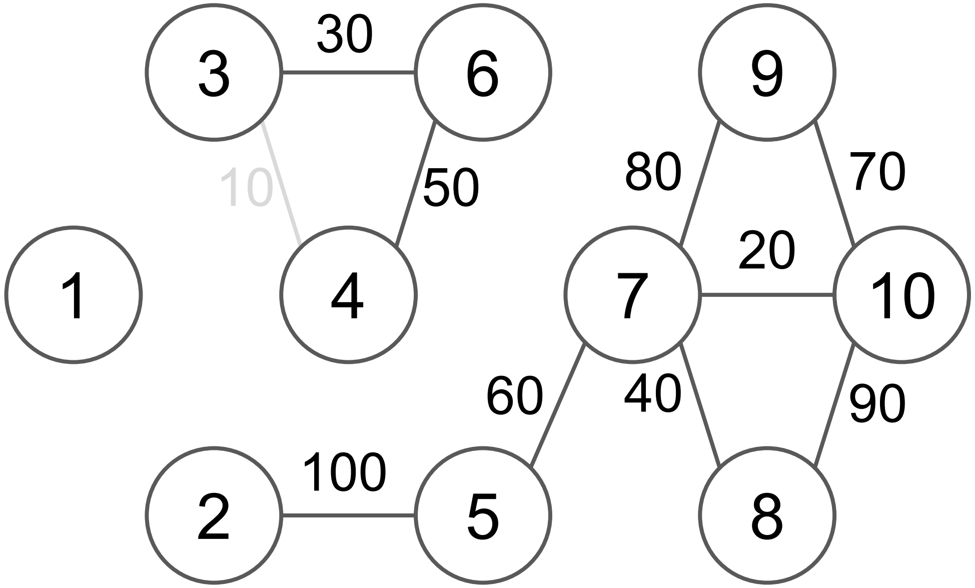
We answer export requests along the way.



Reintroduce edge priorities.

We start with an empty graph, and add edges one-by-one ordered by decreasing priority.

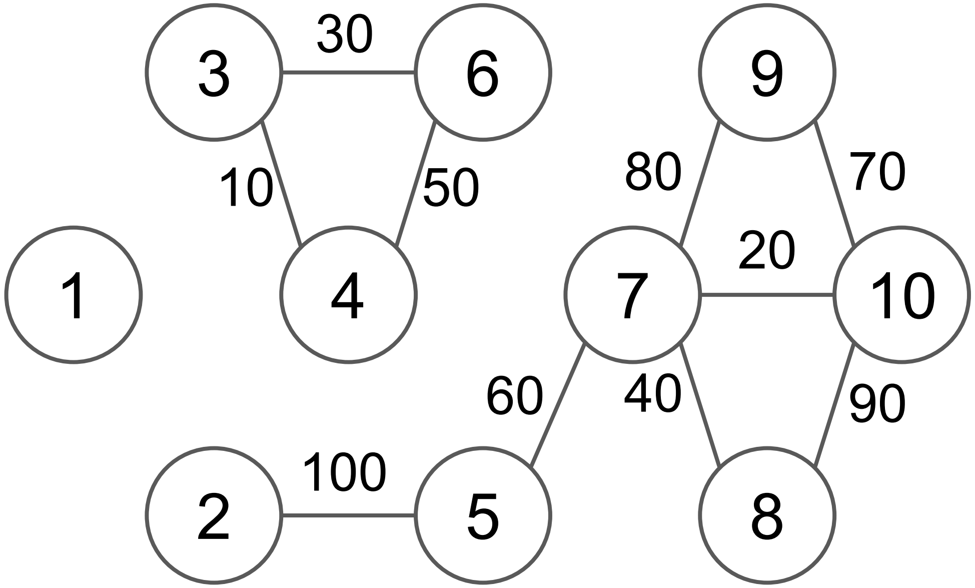
We answer export requests along the way.



Reintroduce edge priorities.

We start with an empty graph, and add edges one-by-one ordered by decreasing priority.

We answer export requests along the way.



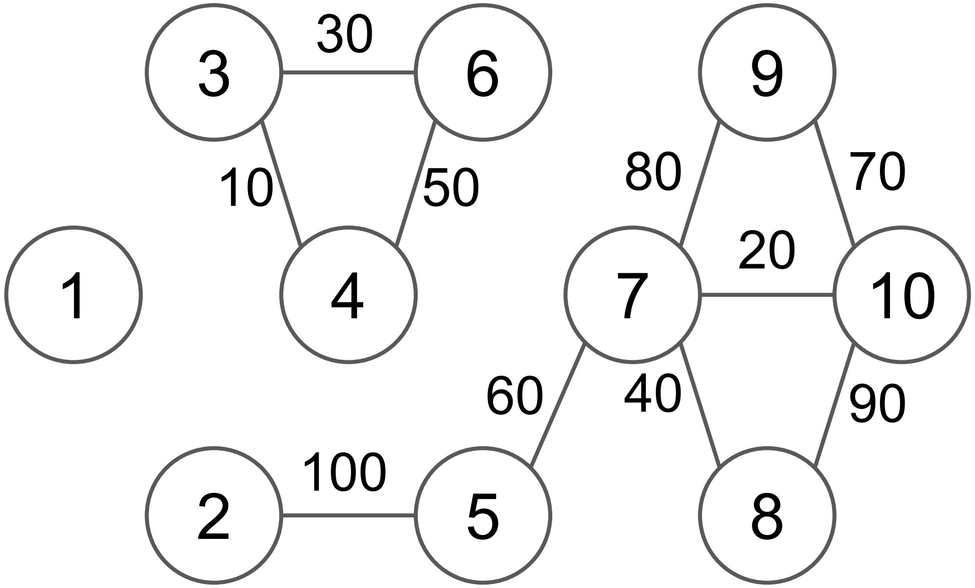
To answer requests we need to maintain:

m: increases by 1 as we add edges

num_degree_0: easy to maintain if we know degree[x]

num_degree_2: easy to maintain if we know degree[x]

num_cycles: tricky



num_cycles = number of graph components where every node is degree 2.

We need to maintain graph components (union-find):

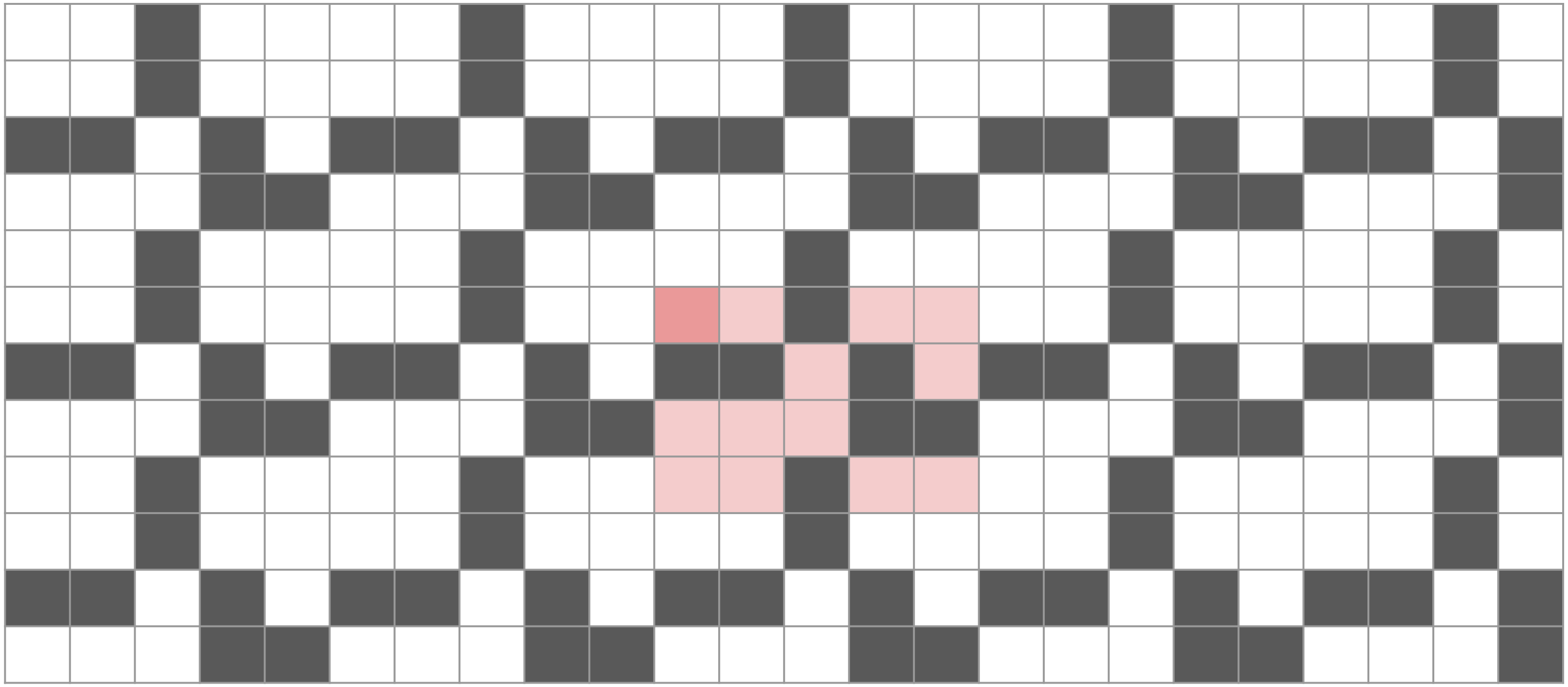
- num_nodes_in_component
- num_nodes_in_component_with_degree_2

Problem L
Looping Labyrinth

Submits: 22
Accepted: ???

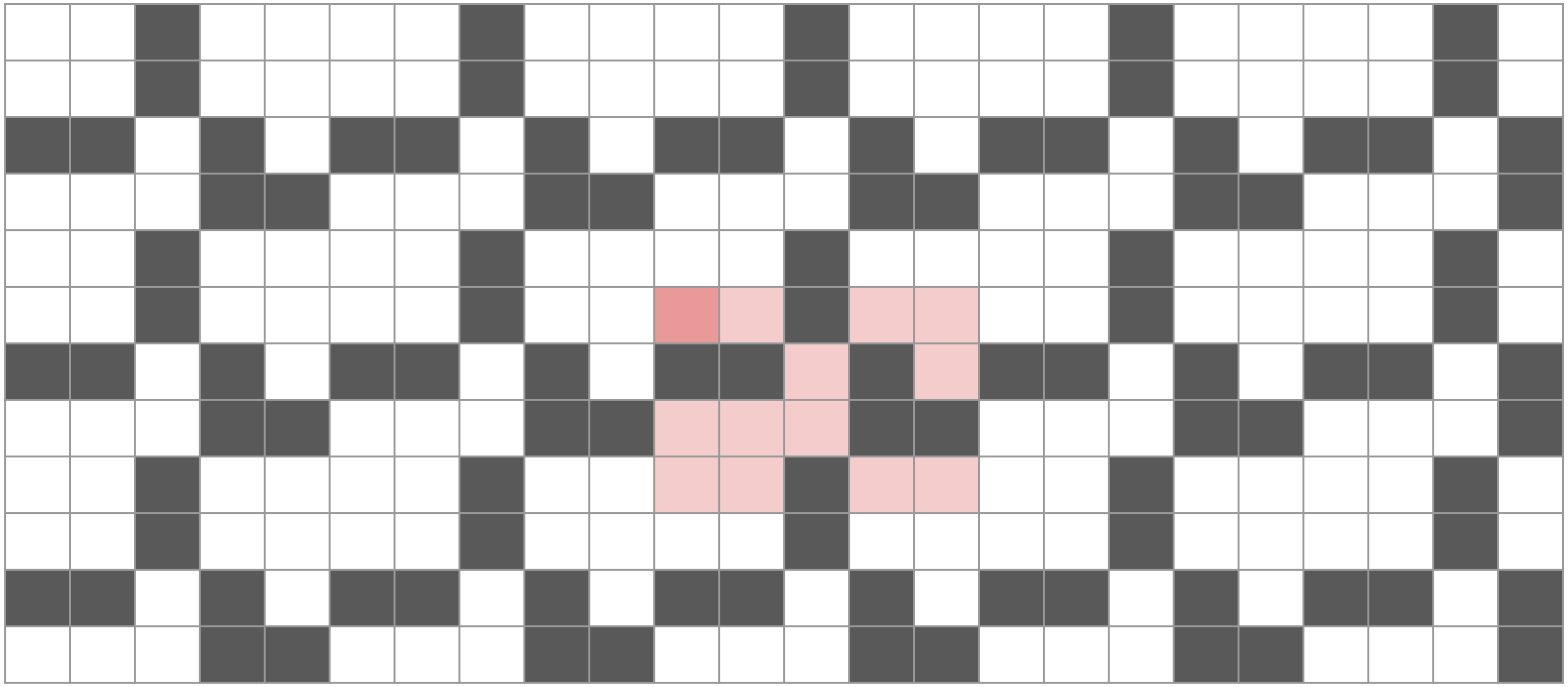
First solved by:
???

Author: Ante Đerek



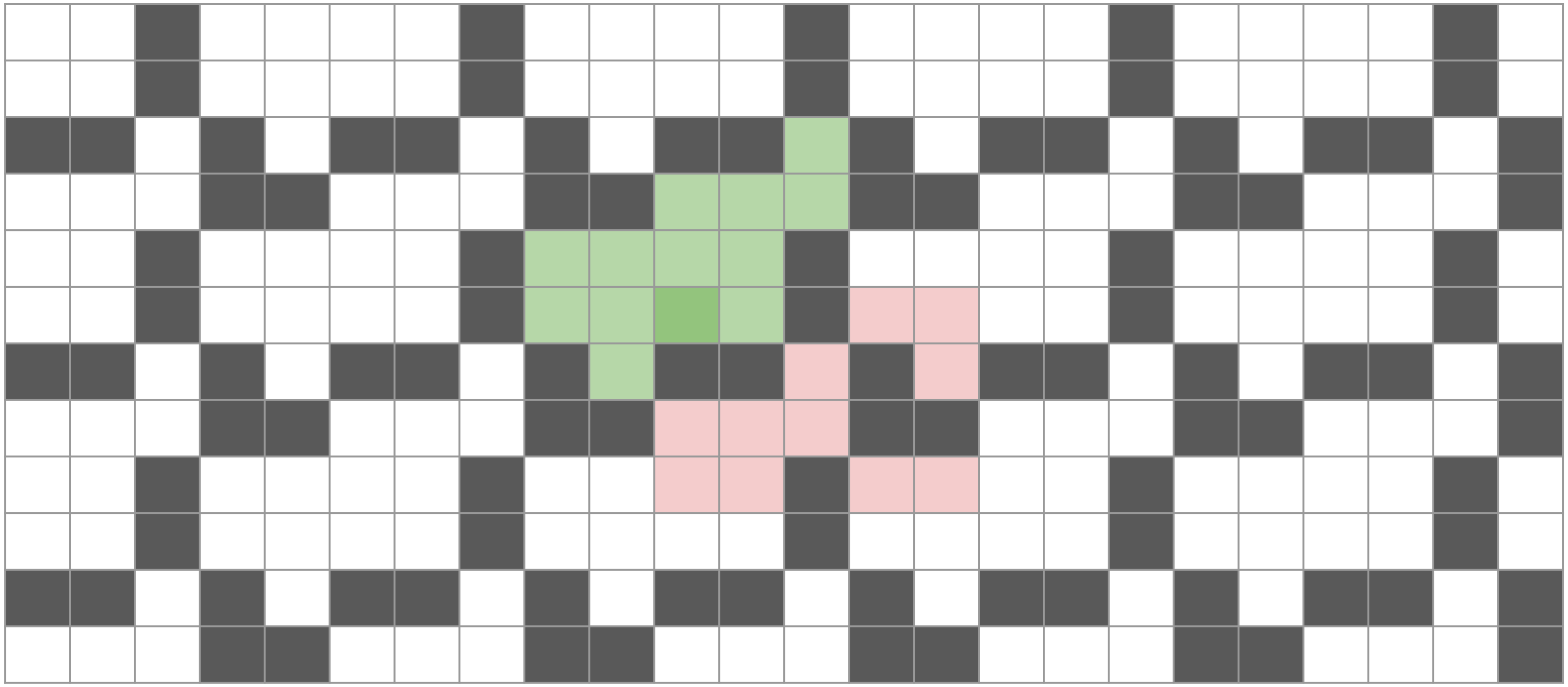
We start by running BFS from the exit (0, 0).

- The maze is infinite, so we limit the number of iterations to 1000000.



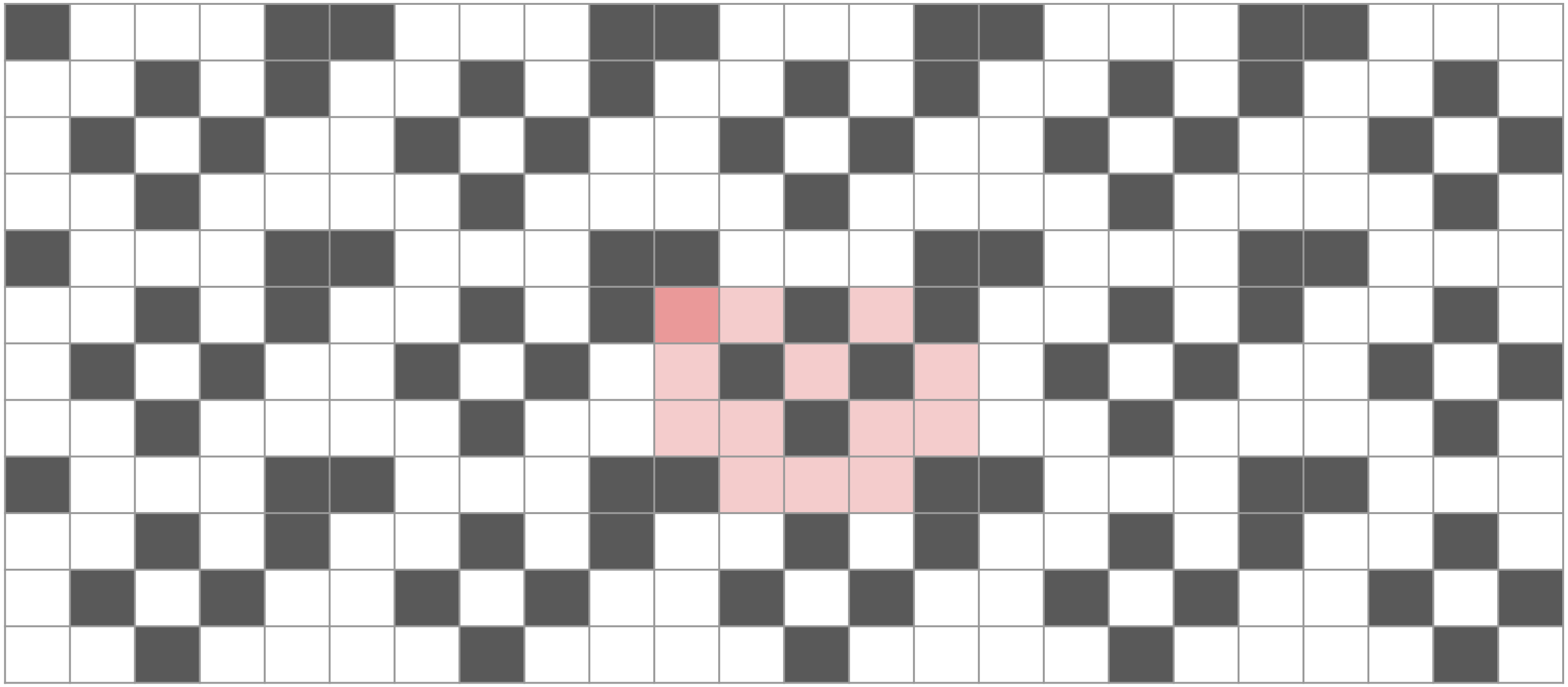
There are three possible outcomes:

- BFS terminates before reaching the limit.



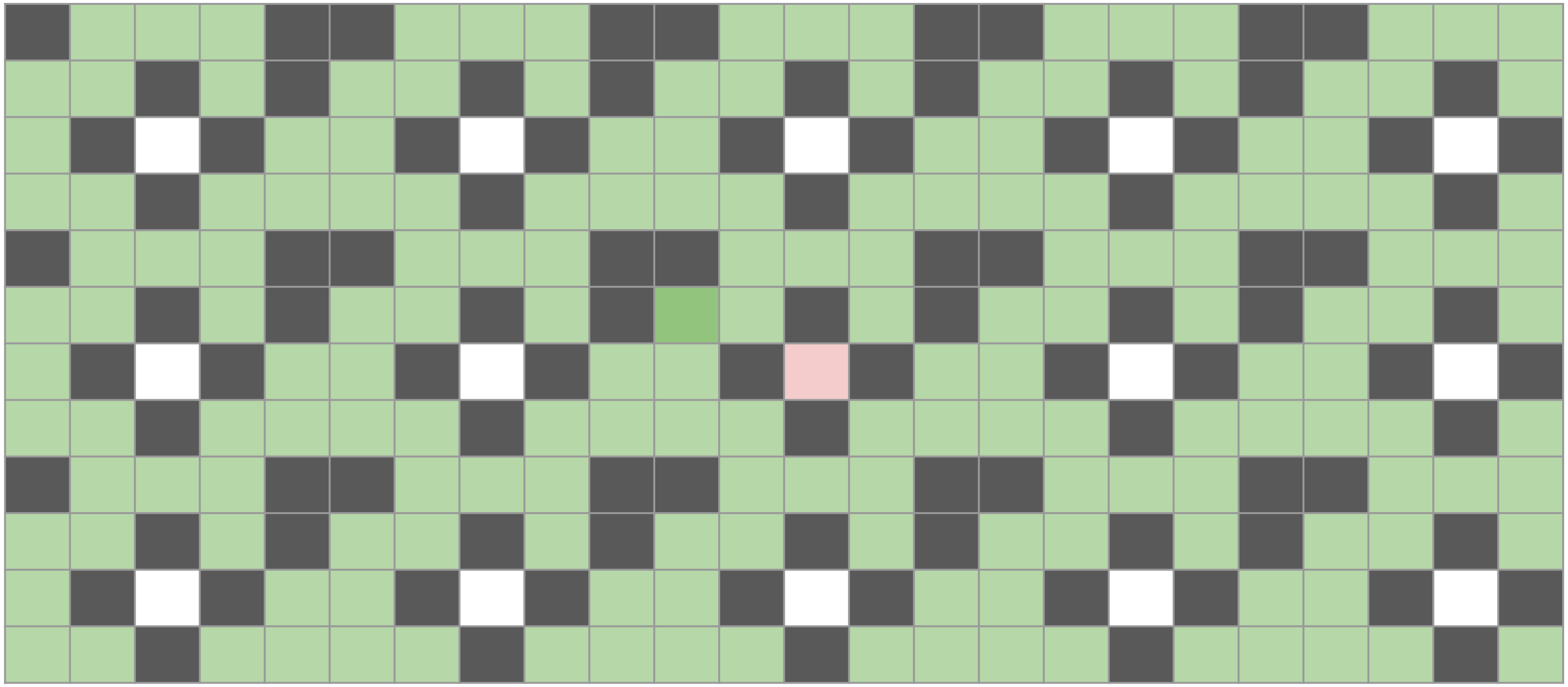
There are three possible outcomes:

- BFS terminates before reaching the limit.



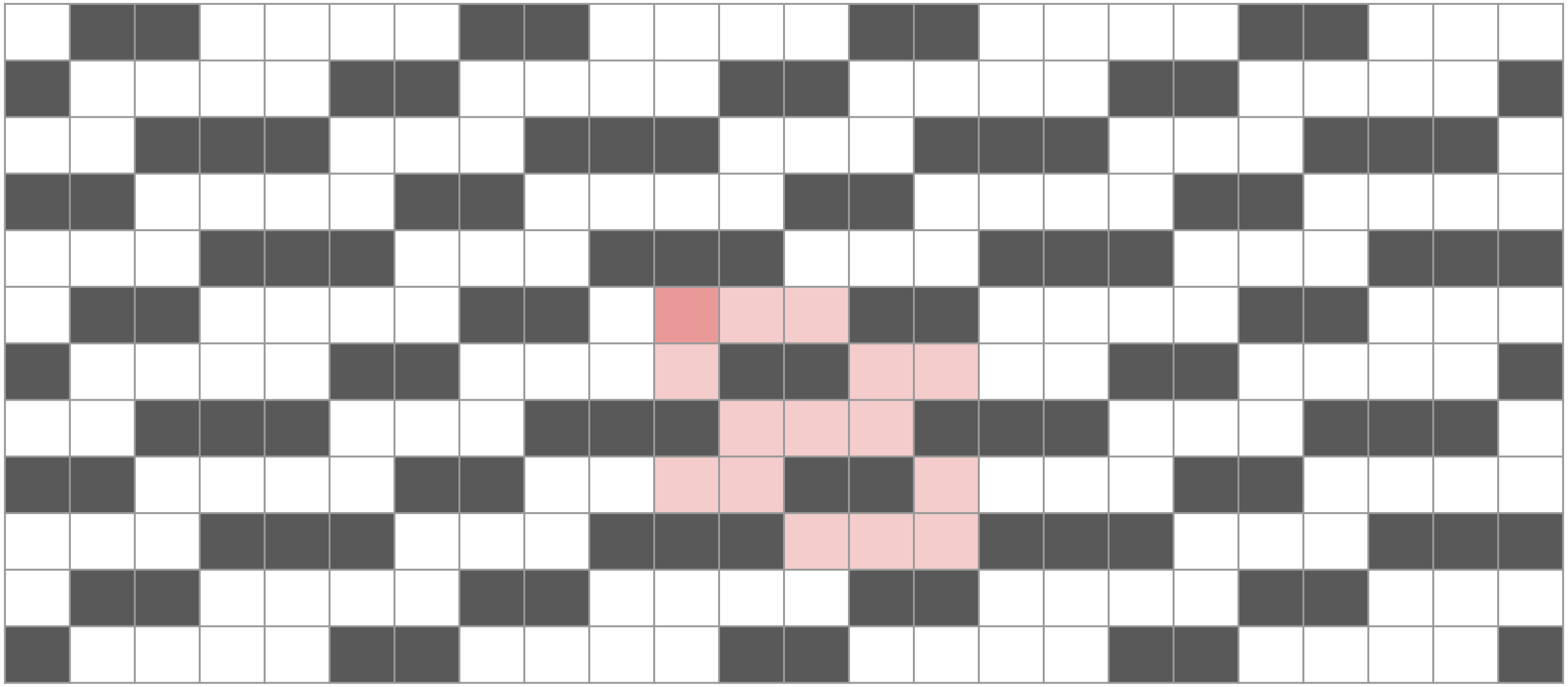
There are three possible outcomes:

- BFS terminates before reaching the limit.
- Every tile is reachable.



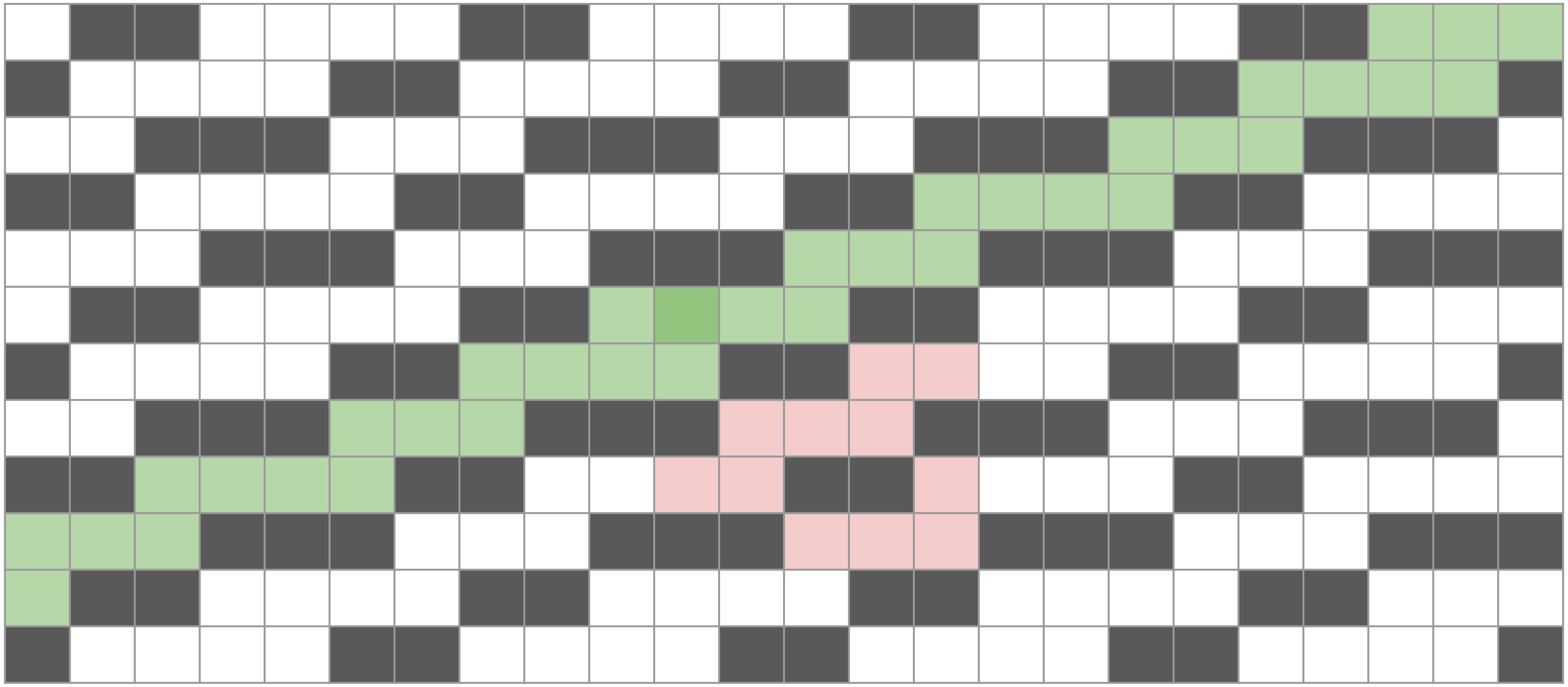
There are three possible outcomes:

- BFS terminates before reaching the limit.
- Every tile is reachable.



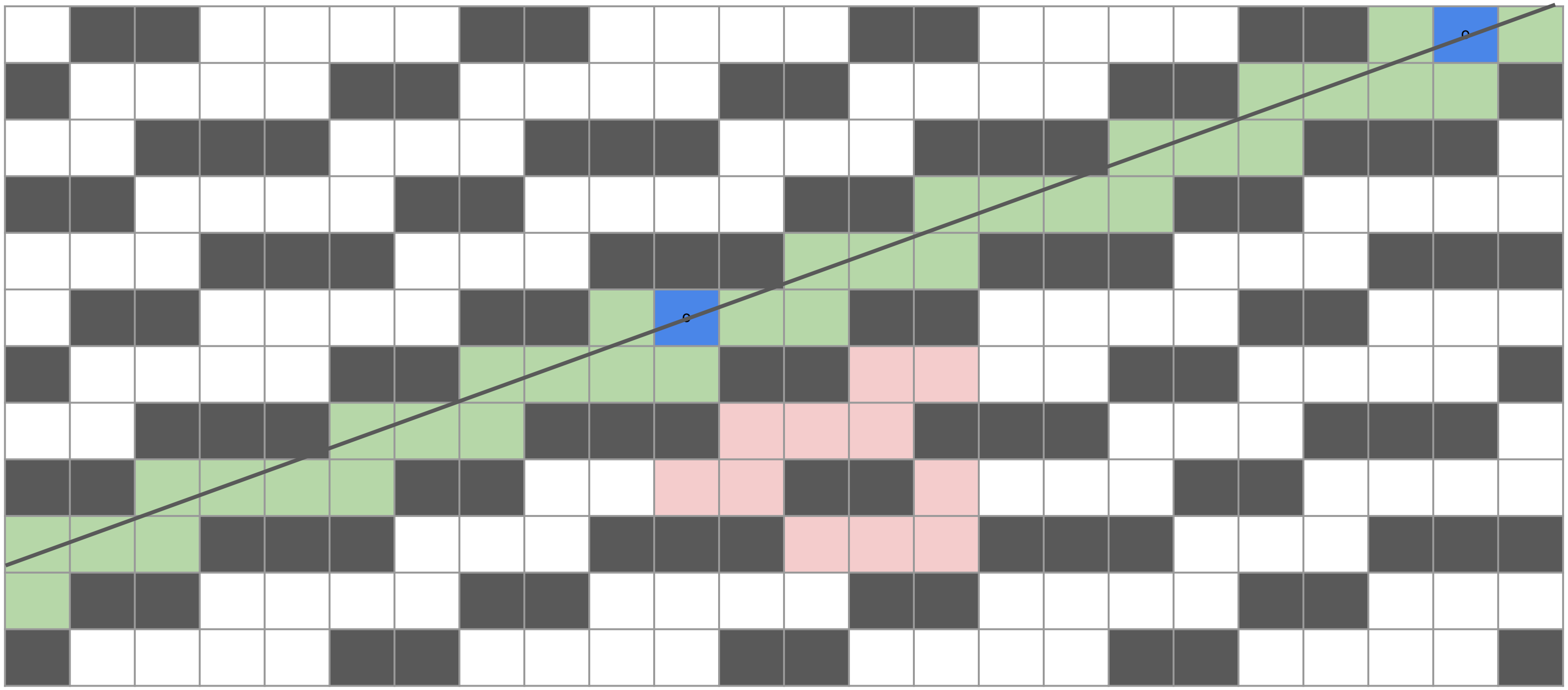
There are three possible outcomes:

- BFS terminates before reaching the limit.
- Every tile is reachable.
- Reachable cells repeat with an offset dr, dc. For every reached cell (r, c), cells (r + k * dr, c + k * dc) are reached as well.



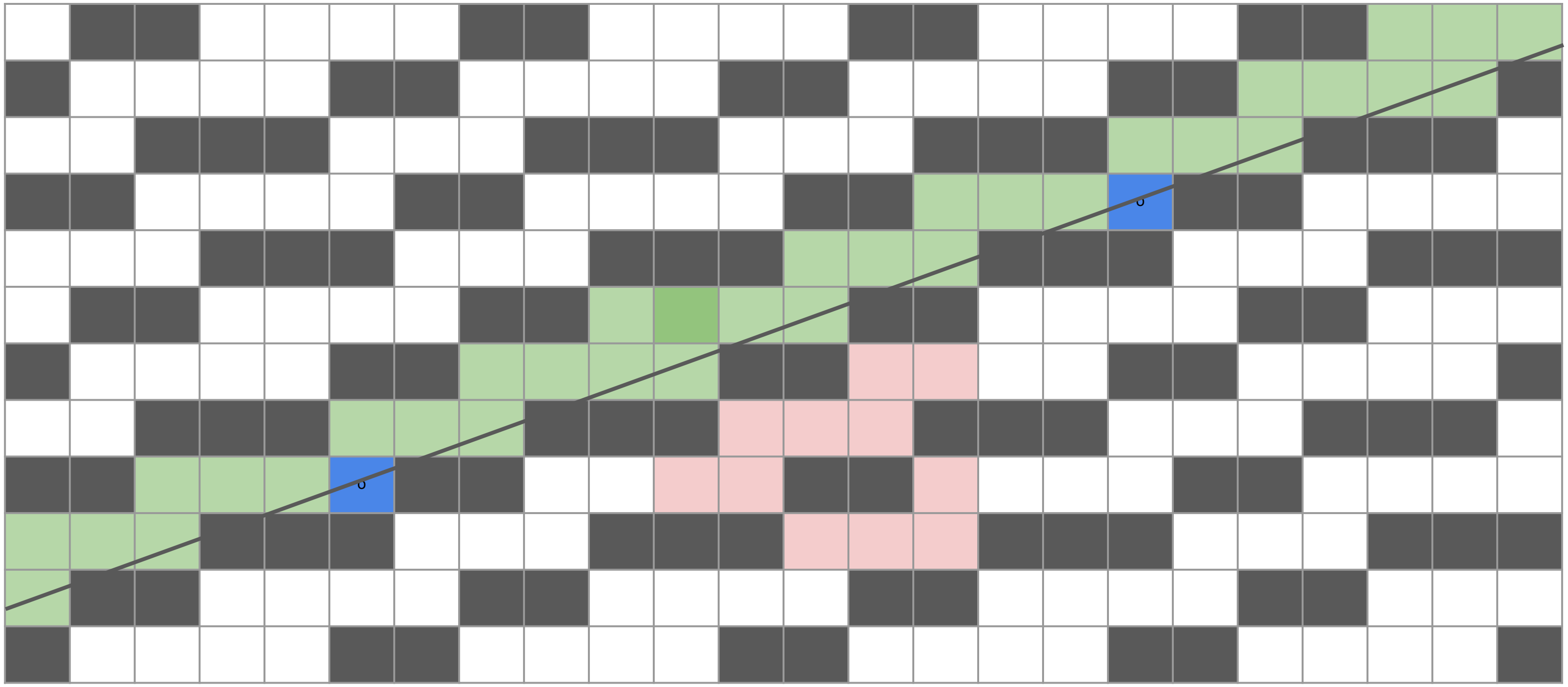
There are three possible outcomes:

- BFS terminates before reaching the limit.
- Every tile is reachable.
- Reachable cells repeat with an offset dr and dc. For every reached cell (r, c), cells (r + k * dr, c + k * dc) are reached as well.



There are three possible outcomes:

- BFS terminates before reaching the limit.
- Every tile is reachable.
- Cells repeat with an offset dr and dc. For every reached cell (r, c), cells (r + k * dr, c + k * dc) are reached as well.



There are three possible outcomes:

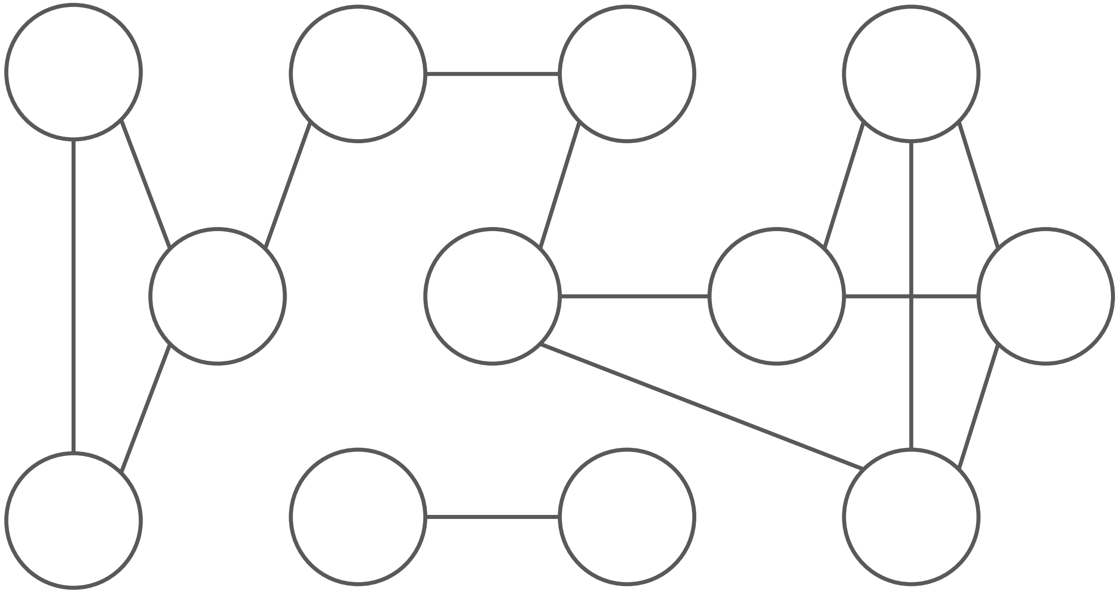
- BFS terminates before reaching the limit.
- Every tile is reachable.
- Cells repeat with an offset dr and dc. For every reached cell (r, c), cells (r + k * dr, c + k * dc) are reached as well.

Problem J
Juice Junctions

Submits: 12
Accepted: at least 1

First solved by:
University of Wroclaw 1
(Bartłomiej Dudek, Maciej Dulęba, Mateusz Gołębiewski)
02:50:23

Author: Luka Kalinovčić, Ivan Katanić



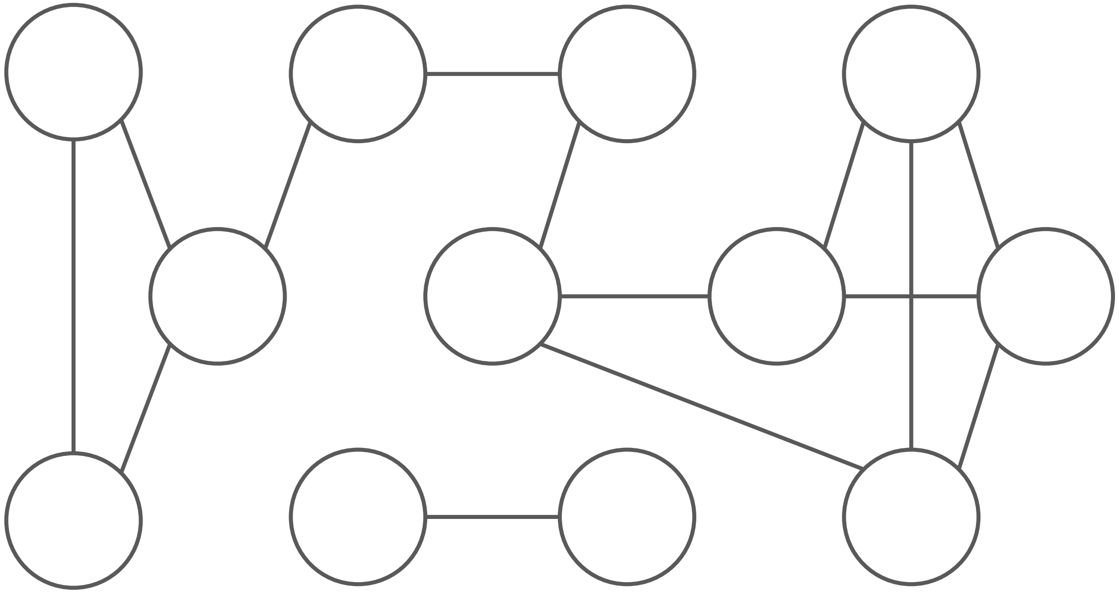
Max-flow == min-cut.

Key observation:

- The degree ≤ 3 => The min-cut is either 0, 1, 2 or 3.

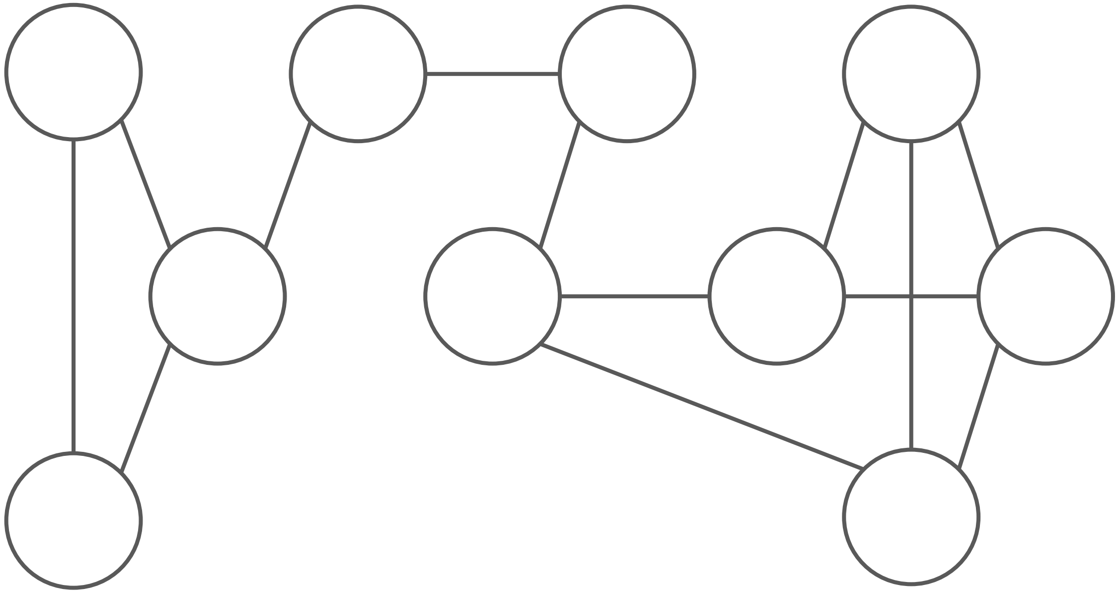
The standard max-flow algorithm is O(n).

However, if we run for every pair, it's $O(n^3)$ -- too slow.

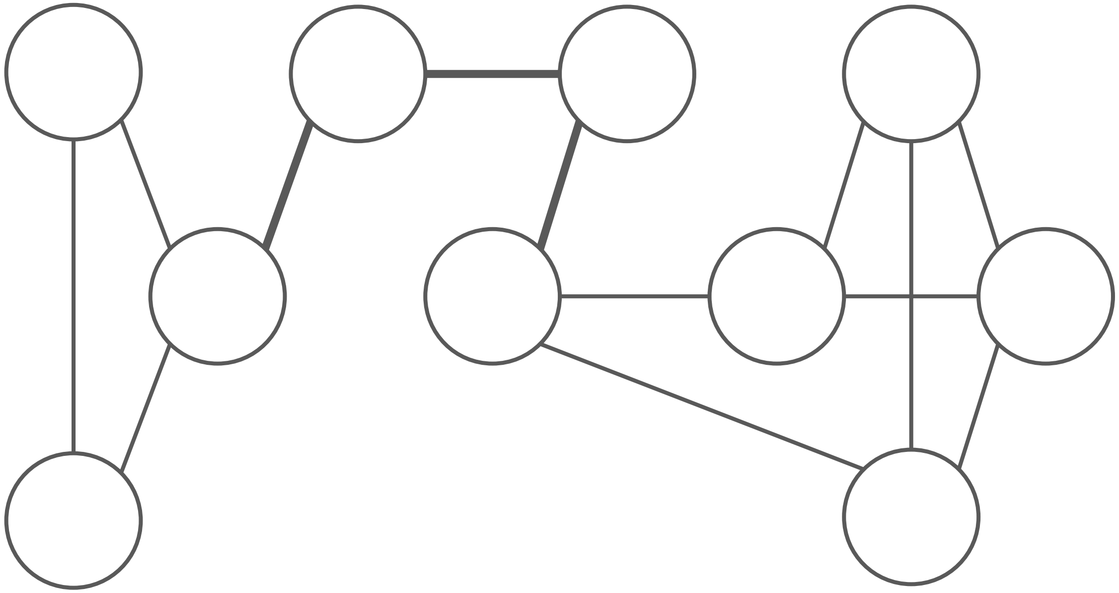


Nodes in different components are already disconnected, so the min-cut is 0.

Find components and handle each component individually.

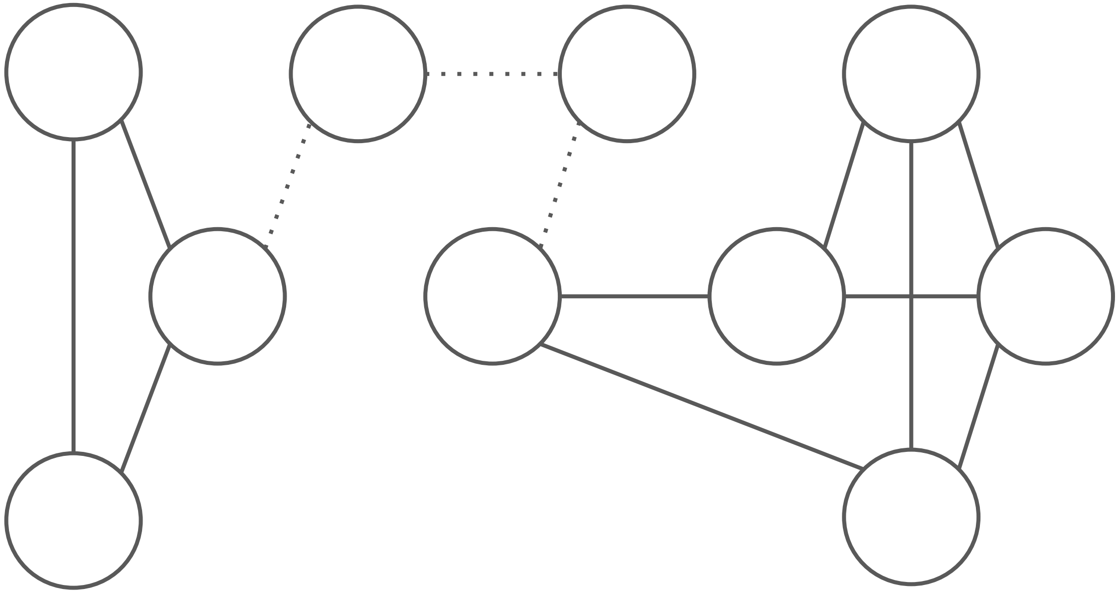


Within a single component, the min-cut is at least 1.



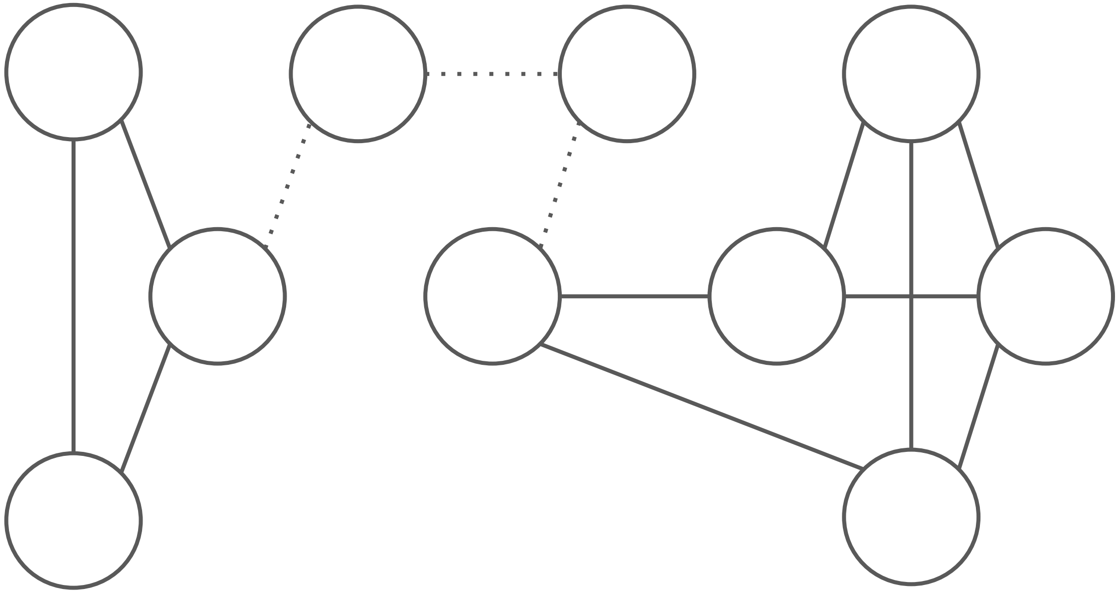
Within a single component, the min-cut is at least 1.

Find bridges and delete them.



Within a single component, the min-cut is at least 1.

Find bridges and delete them.

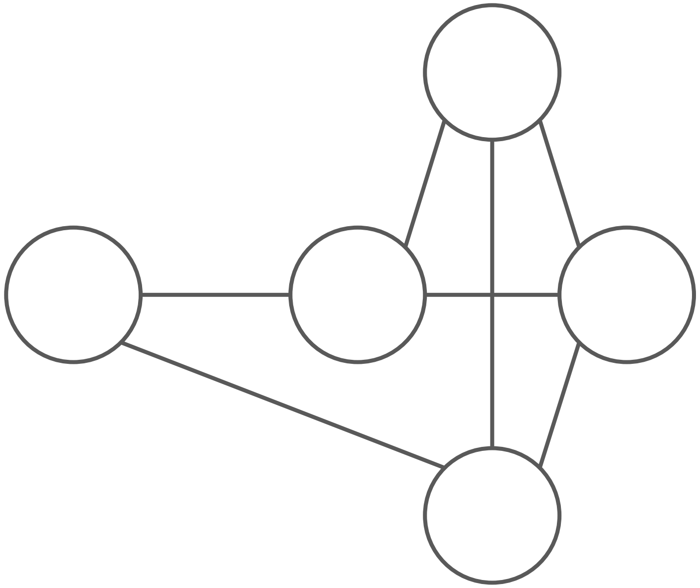


Within a single component, the min-cut is at least 1.

Find bridges and delete them.

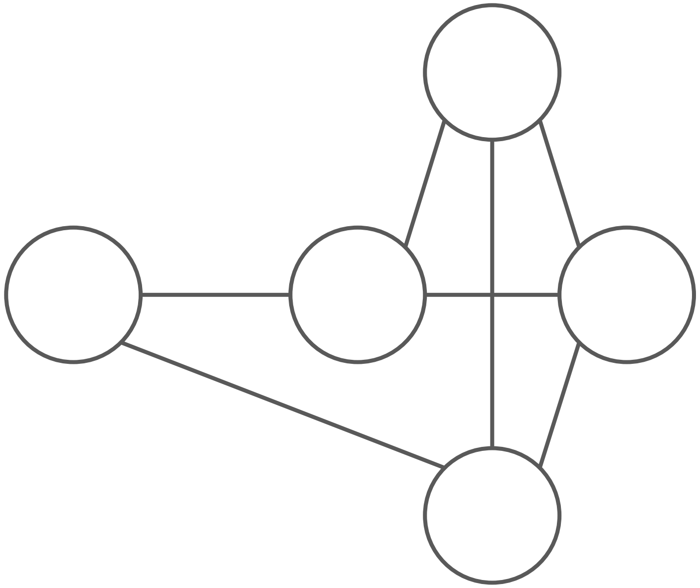
Min-cut for pairs of nodes that got disconnected is 1.

Find components and proceed with each component individually.



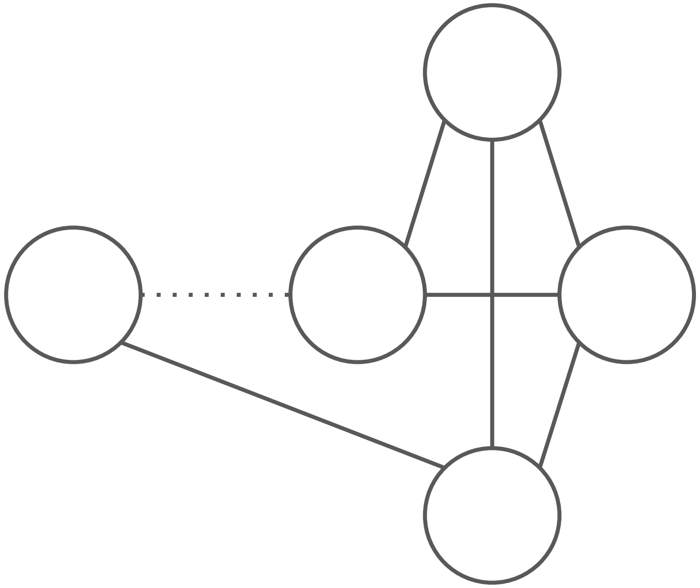
We now observe a single biconnected component.

The min-cut between a pair of nodes is either 2 or 3.

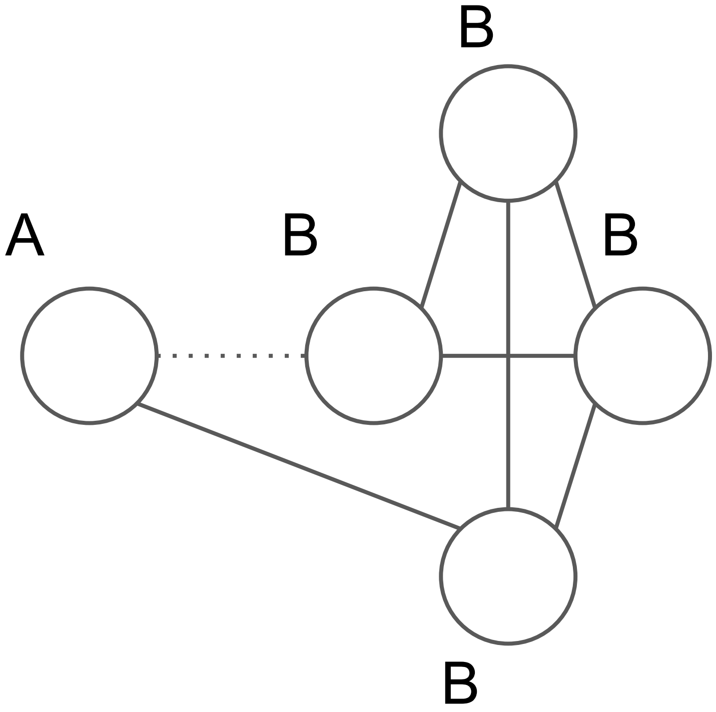


Key observation:

The min-cut between a pair of nodes is 2 iff there exists an edge whose removal causes the two nodes to move to different biconnected components. (i.e. iff there is a bridge between them when we remove one edge)

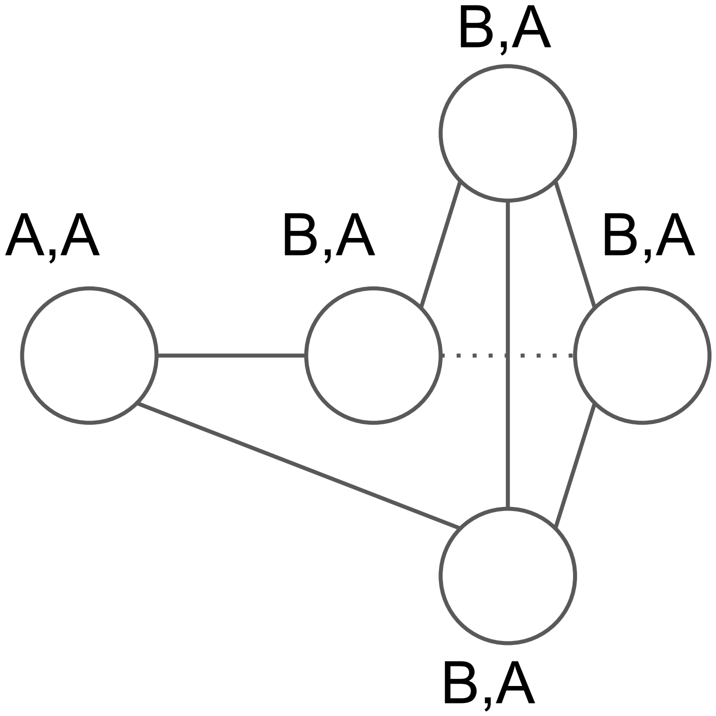


For each edge, we temporarily remove it, and find biconnected components.



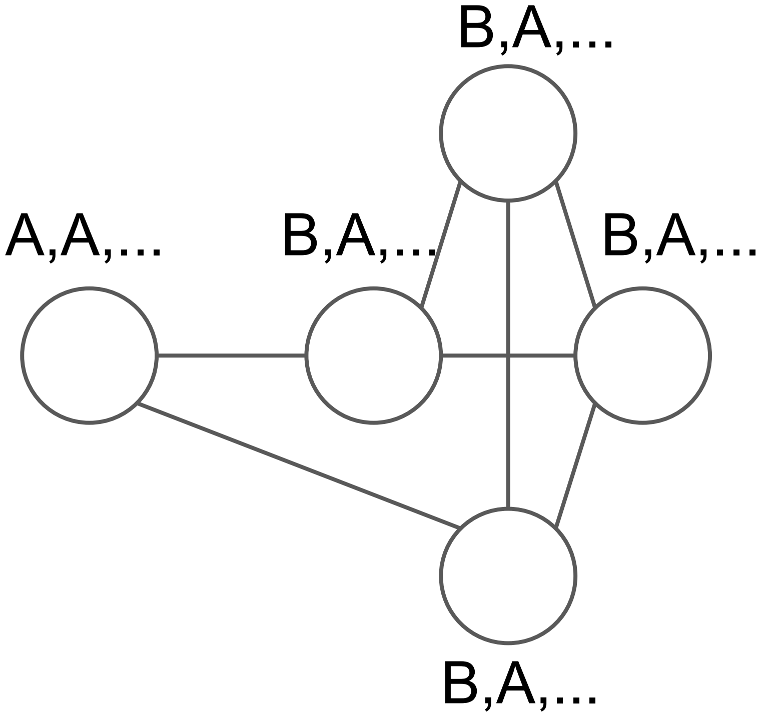
For each edge, we temporarily remove it, and find biconnected components.

We label the biconnected components, and append the label to the list of labels we store at each node.



For each edge, we temporarily remove it, and find biconnected components.

We label the biconnected components, and append the label to the list of labels we store at each node.



For each edge, we temporarily remove it, and find biconnected components.

We label the biconnected components, and append the label to the list of labels we store at each node.

The min-cut between the two nodes is 3 if the list of their labels matches exactly, or 2 otherwise. (hashing)

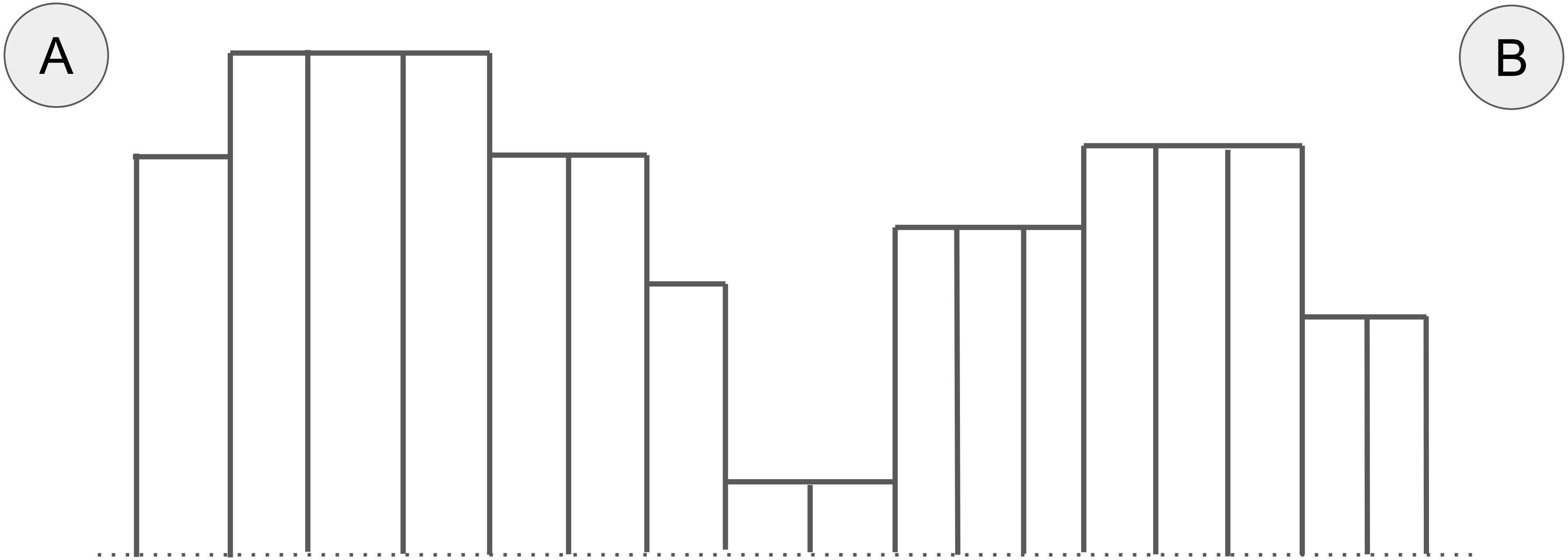
Problem G

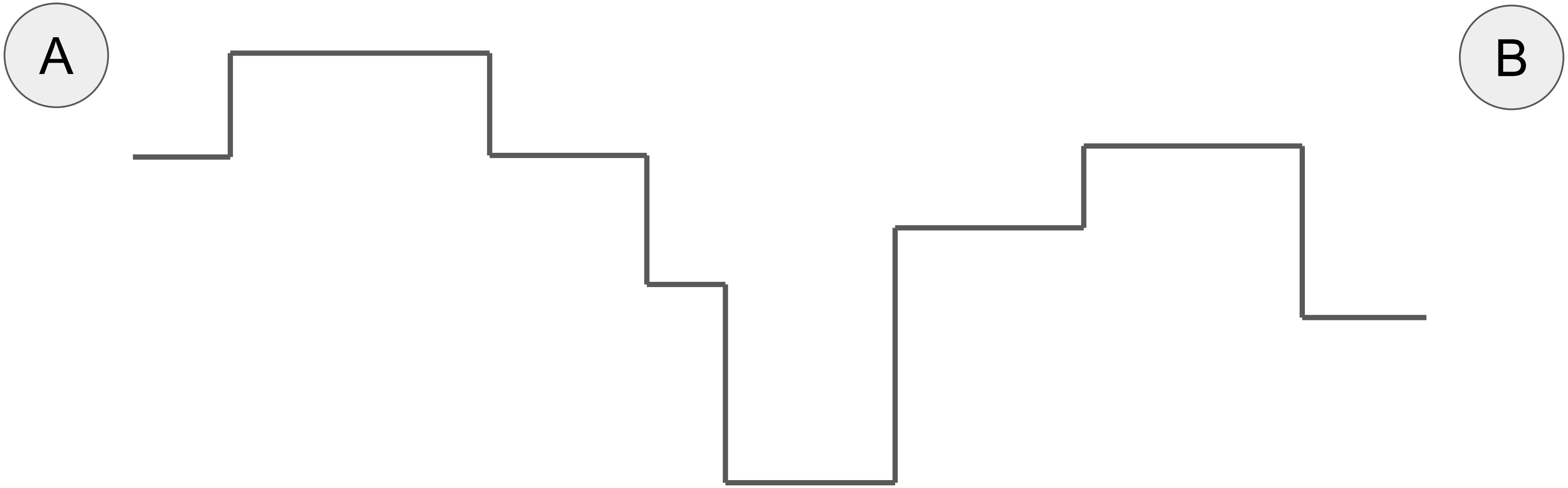
Greenhouse Growth

Submits: 27
Accepted: ???

First solved by:
???

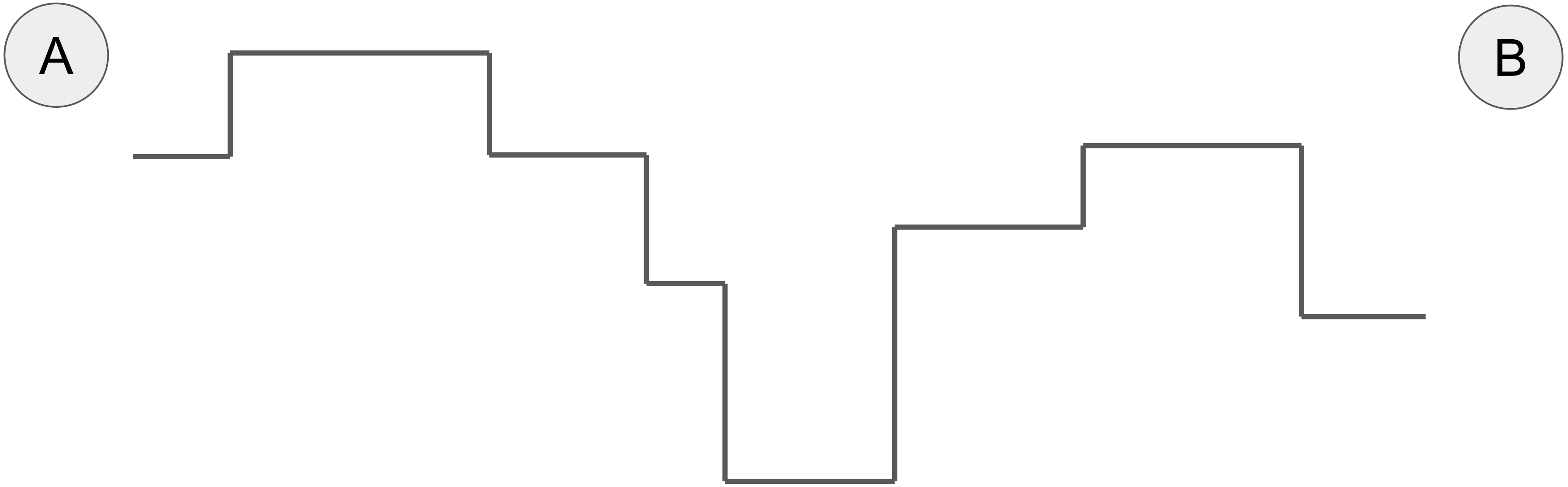
Author: Luka Kalinovčić


A
B



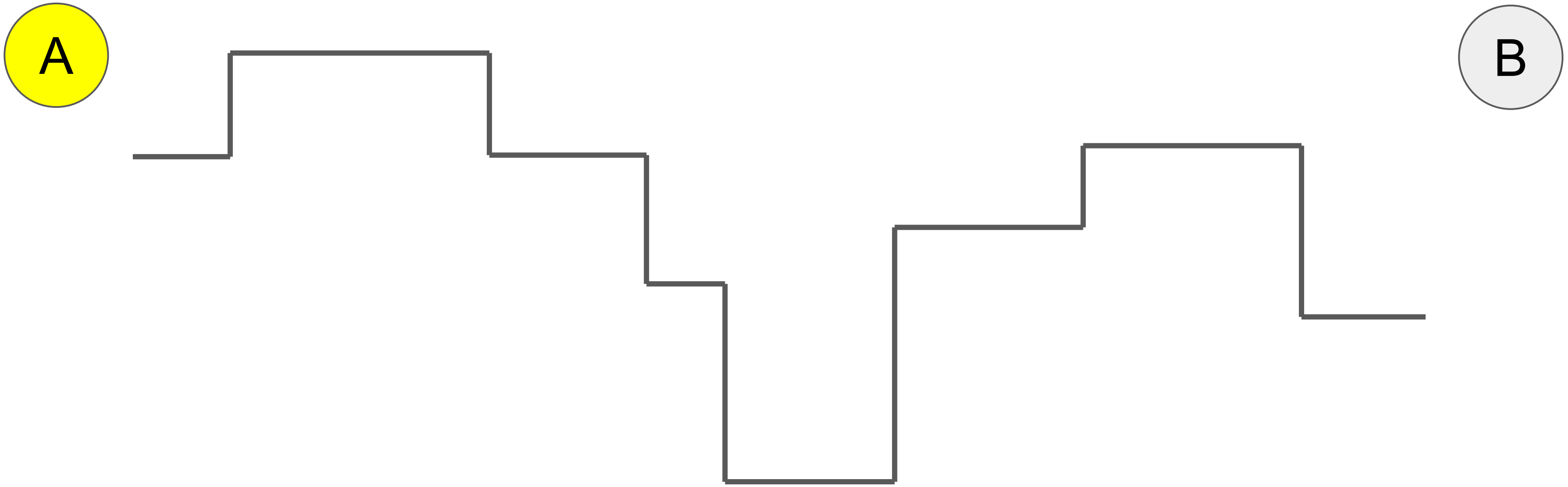
We maintain the “skyline” of the greenhouse as a linked list of horizontal and vertical segments.

As sunflowers grow, some vertical segments disappear and we merge horizontal segments at the same height. Problem: We can’t afford to store the height of each segment explicitly.



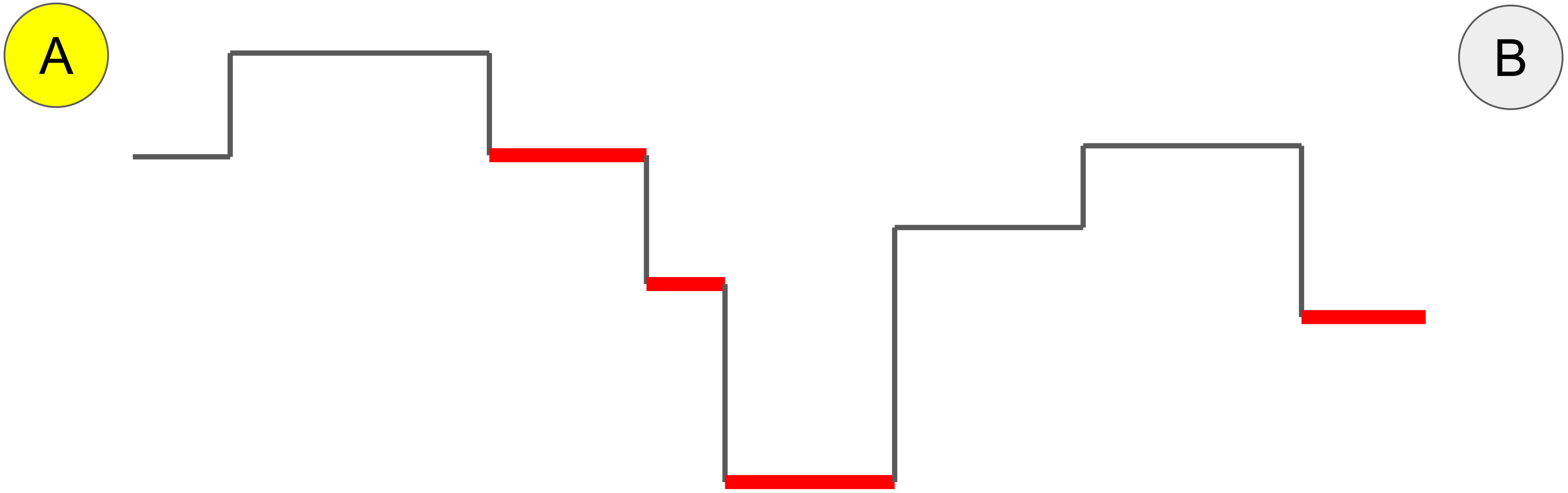
Instead, for a horizontal segment we store:

- h_0 and t_0 -- at time t_0 the height was h_0.



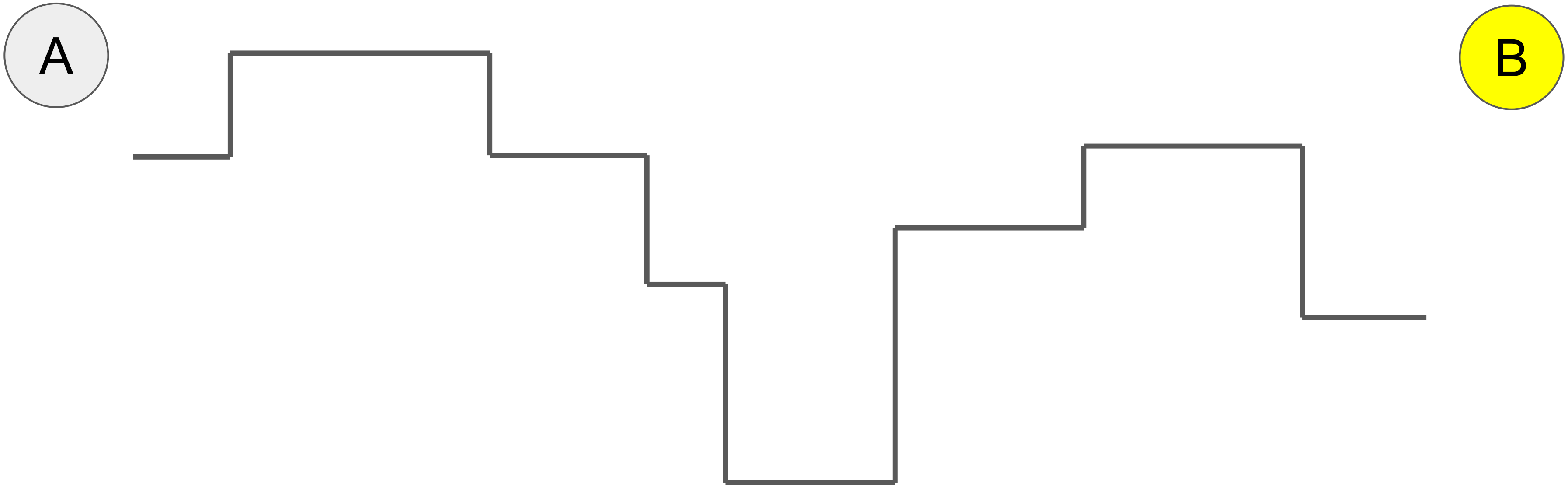
Instead, for a horizontal segment we store:

- h_0 and t_0 -- at time t_0 the height was h_0.
- grows_A -- whether it grows when lamp A is on.



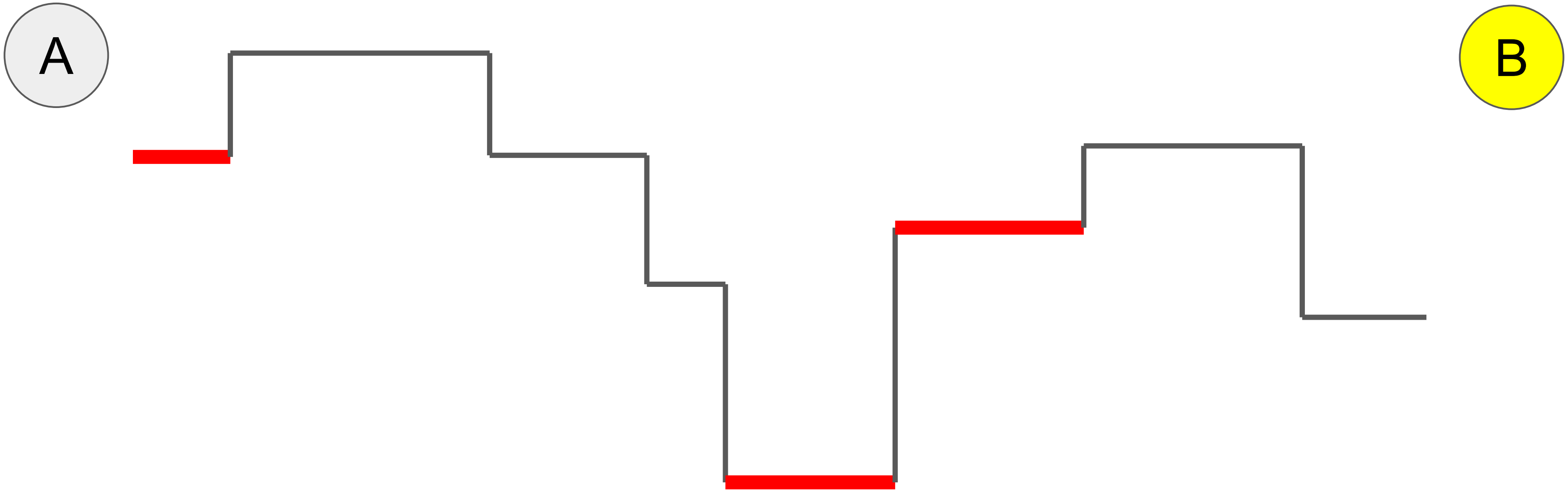
Instead, for a horizontal segment we store:

- h_0 and t_0 -- at time t_0 the height was h_0.
- grows_A -- whether it grows when lamp A is on.



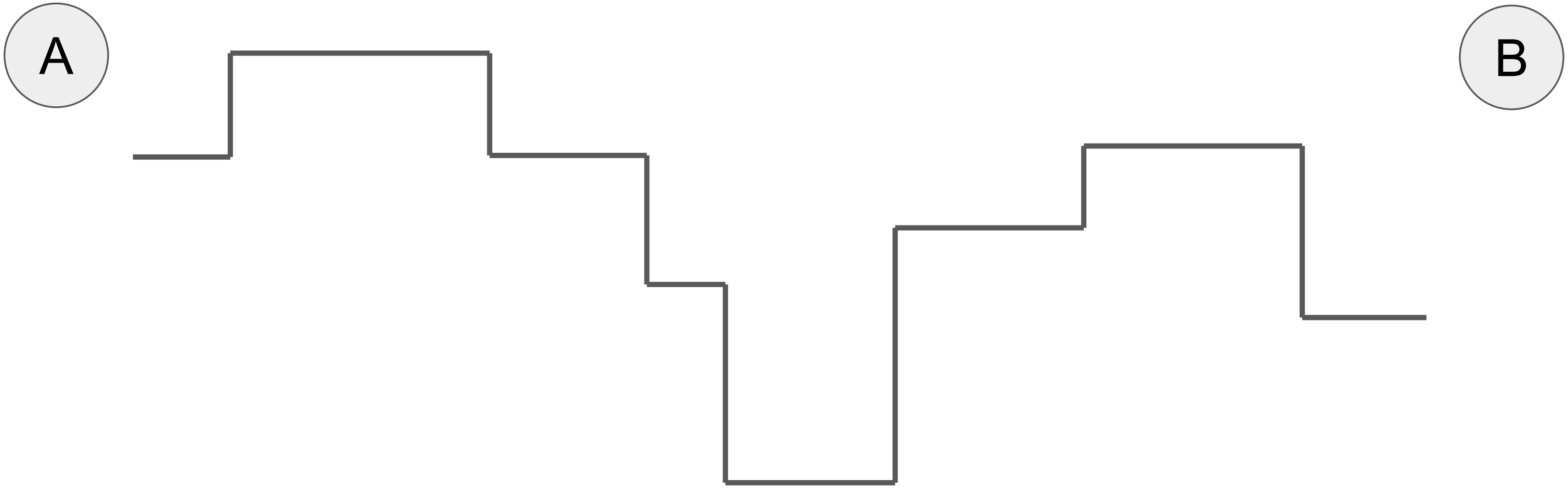
Instead, for a horizontal segment we store:

- h_0 and t_0 -- at time t_0 the height was h_0.
- grows_A -- whether it grows when lamp A is on.
- grows_B -- whether it grows when lamp B is on.



Instead, for a horizontal segment we store:

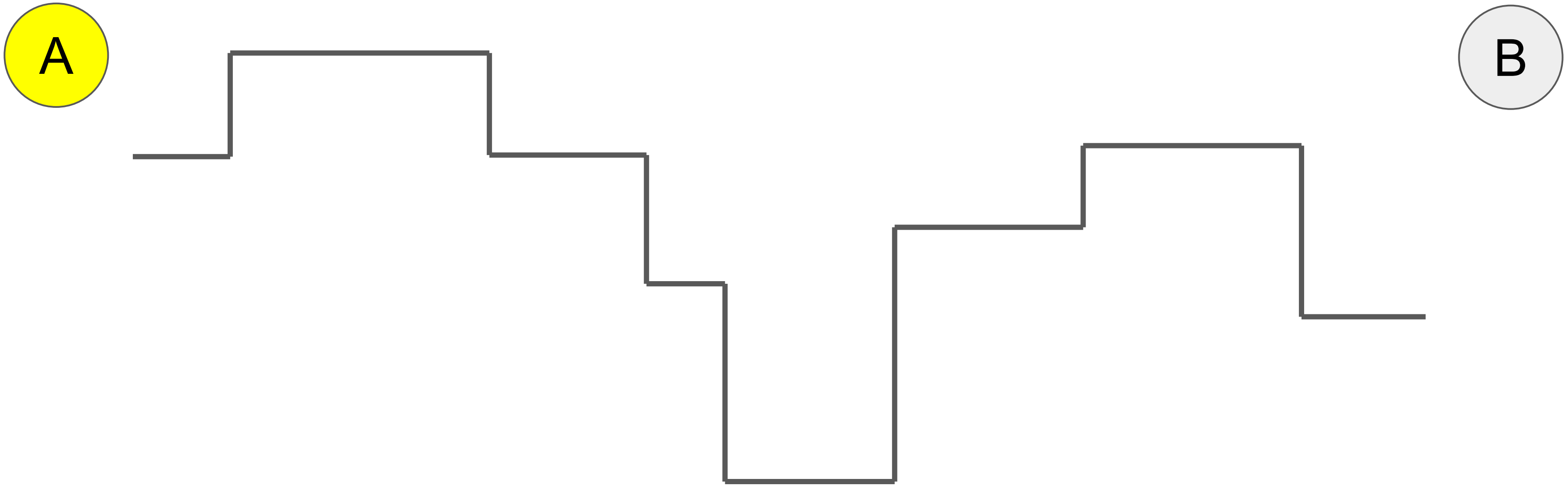
- h_0 and t_0 -- at time t_0 the height was h_0.
- grows_A -- whether it grows when lamp A is on.
- grows_B -- whether it grows when lamp B is on.



$$h(t_{now}) = h_0 + num\_A(t_0, t_{now}) * grows\_A + num\_B(t_0, t_{now}) * grows\_B.$$

… as long as grows_A and grows_B don’t change.

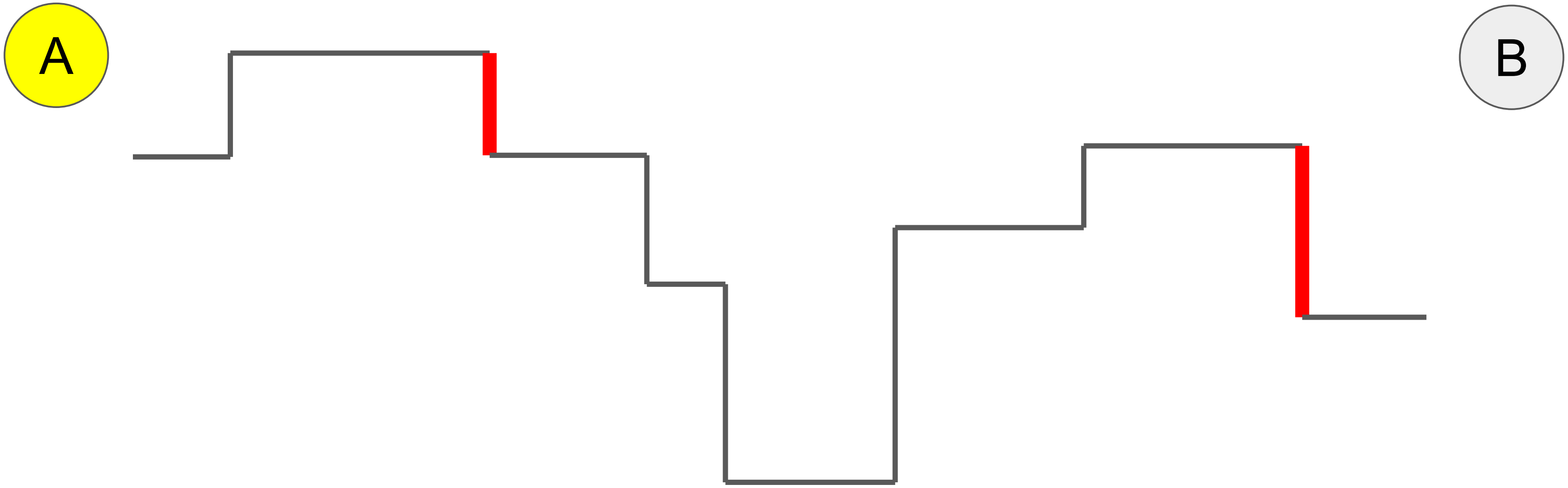
It only changes when a vertical segment disappears. But then we delete two horizontal segments and create a new merged segment.



When do vertical segments disappear?

For every vertical segment we store:

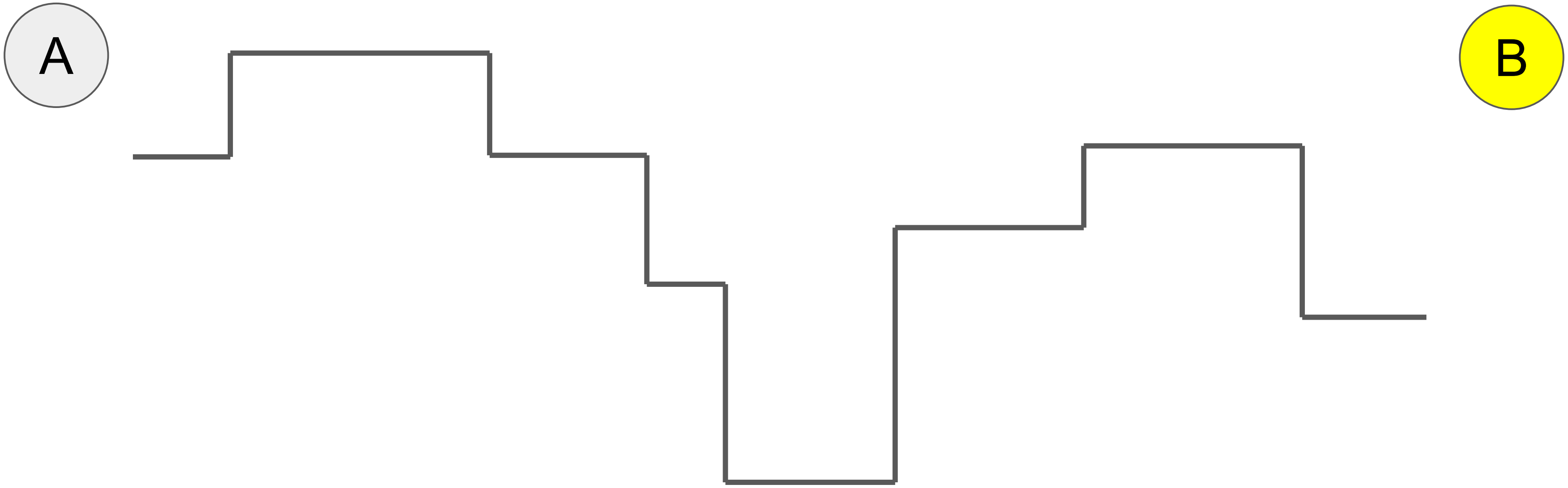
- shrinks_A -- whether it shrinks when lamp is on.



When do vertical segments disappear?

For every vertical segment we store:

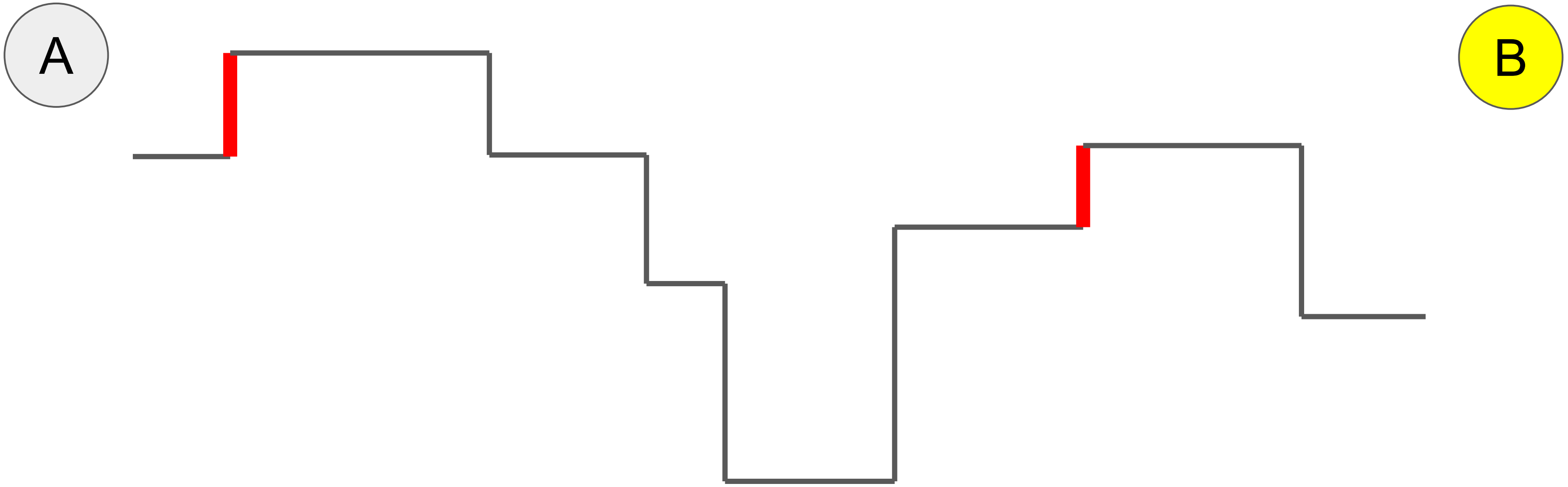
- shrinks_A -- whether it shrinks when lamp A is on.



When do vertical segments disappear?

For every vertical segment we store:

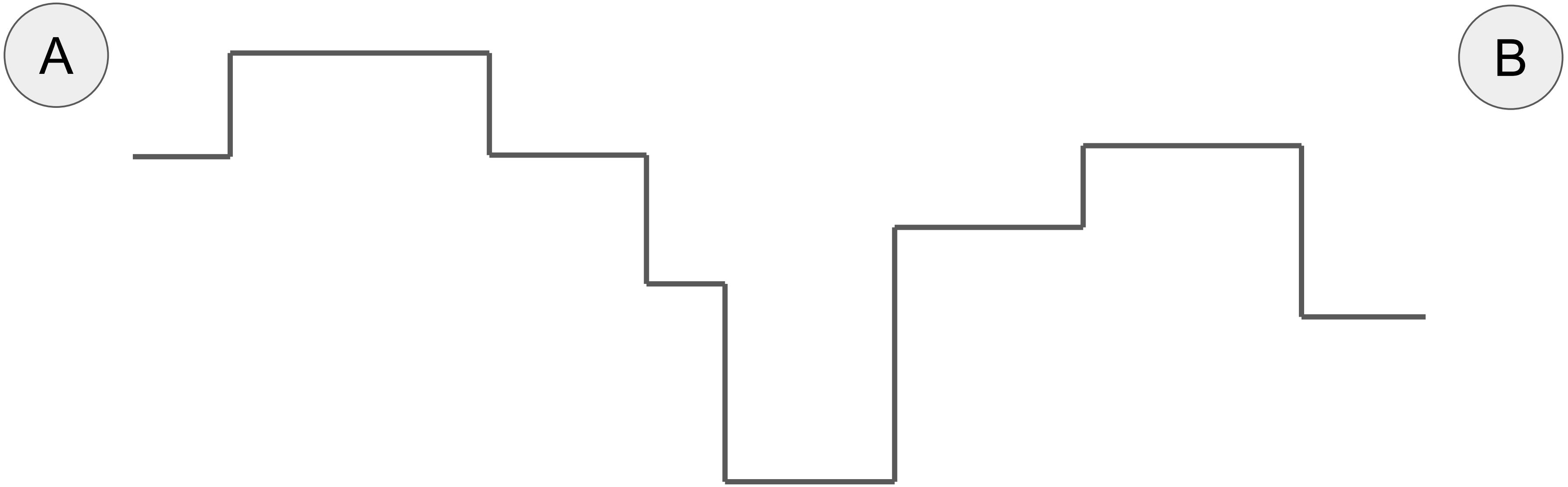
- shrinks_A -- whether it shrinks when lamp A is on.
- shrinks_B -- whether it shrinks when lamp B is on.



When do vertical segments disappear?

For every vertical segment we store:

- shrinks_A -- whether it shrinks when lamp A is on.
- shrinks_B -- whether it shrinks when lamp B is on.



Let L be the length of the vertical segment.

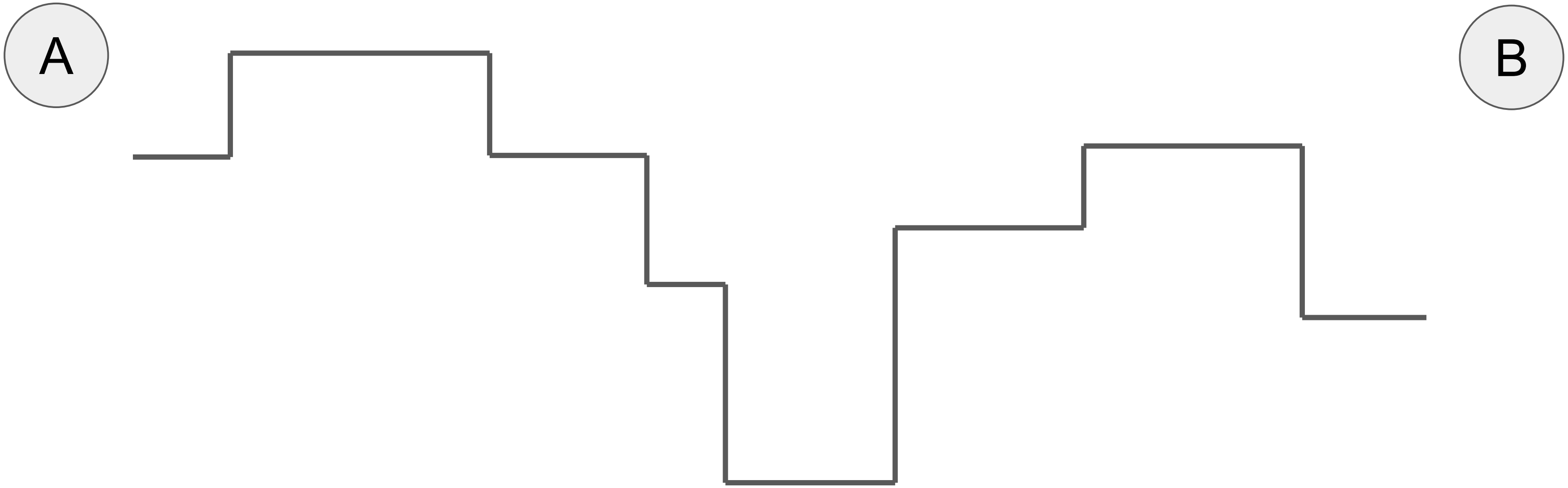
Segments set alarms to check if they disappeared:

If shrinks_A: wake me up after lamp A has been turned on L times.

If shrinks_B: wake me up after lamp B has been turned on L times.

If shrinks_A and shrinks_B: wake me up in L days.

Also, wake me up if my neighbours die -- get merged with some other segment. We need to reevaluate shrinks_A and shrinks_B and reset alarms.



Simulate turning lamps on, day-by-day:

- Waking up vertical segments whenever their alarms set off.
- Deleting vertical segments when they disappear and merging horizontal segments.

The total time complexity of O(n + m).

Problem F
Frightful Formula

Submits: 52
Accepted: at least 7

First solved by:
University of Zagreb 5
(Matej Gradiček, Zvonimir Jurelinec, Borna Vukorepa)
01:44:53

Authors: Adrian Satja Kurdija, Ivan Katanić

Start with a simpler formula where we don’t add c:

$F[i, j] = a \cdot F[i, j-1] + b \cdot F[i-1, j]$.

0	0	x	0	0
0	0	$b \cdot x$	$a \cdot b \cdot x$	$a^2 \cdot b \cdot x$
0	0	$b^2 \cdot x$	$2 \cdot a \cdot b^2 \cdot x$	$3 \cdot a^2 \cdot b^2 \cdot x$
0	0	$b^3 \cdot x$	$3 \cdot a \cdot b^3 \cdot x$	$6 \cdot a^2 \cdot b^3 \cdot x$
0	0	$b^4 \cdot x$	$4 \cdot a \cdot b^4 \cdot x$	$10 \cdot a^2 \cdot b^4 \cdot x$

Start with a simpler formula where we don’t add c:

F[i, j] = a · F[i, j-1] + b · F[i-1, j]

The contribution of a single number x at position (1, j):

choose(n - j, 2·n - j - 2) · a^{n-j} · b^{n-1} · x

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
x	$a \cdot x$	$a^2 \cdot x$	$a^3 \cdot x$	$a^4 \cdot x$
0	$a \cdot b \cdot x$	$2 \cdot a^2 \cdot b \cdot x$	$3 \cdot a^3 \cdot b \cdot x$	$4 \cdot a^4 \cdot b \cdot x$

Start with a simpler formula where we don't add c:

$F[i, j] = a \cdot F[i, j-1] + b \cdot F[i-1, j]$

The contribution of a single number x at position (1, j):

$\text{choose}(n - j, 2 \cdot n - j - 2) \cdot a^{n - j} \cdot b^{n - 1} \cdot x$

The contribution of a single number x at position (i, 1):

$\text{choose}(n - i, 2 \cdot n - i - 2) \cdot a^{n - 1} \cdot b^{n - i} \cdot x$

Because we have a prime module, we can compute choose(k, n) = n! / k! / (n - k)! by precomputing modular inverse of factorials.

We can then compute the contribution of all numbers in the first row and column in O(n).

0	0	0	0	0
0	0	0	0	0
0	c	$a \cdot c$	$a^2 \cdot c$	$a^3 \cdot b \cdot c$
0	$b \cdot c$	$2 \cdot a \cdot b \cdot c$	$3 \cdot a^2 \cdot b \cdot c$	$4 \cdot a^3 \cdot b \cdot c$
0	$b^2 \cdot c$	$3 \cdot a \cdot b^2 \cdot c$	$6 \cdot a^2 \cdot b^2 \cdot c$	$10 \cdot a^3 \cdot b^2 \cdot c$

Let’s reintroduce “plus c” but only at a single cell.

The contribution of a single number c at position (i, j):

$$\text{choose}(n - i, 2 \cdot n - i - j) \cdot a^{n - j} \cdot b^{n - i} \cdot c$$

However, we have (n - 1) · (n - 1) positions where we have to add c -- too many to evaluate the expression for every position (i, j).

$$\sum_{i=2}^{n}\sum_{j=2}^{n}(\frac{(2n-i-j)!)}{(n-i)!(n-j)!})a^{n-j}b^{n-i}c$$

$$c\sum_{i=2}^{n}\sum_{j=2}^{n}(2n-i-j)!(\frac{a^{n-j}}{(n-j)!})(\frac{b^{n-i}}{(n-i)!})$$

$$c\sum_{i=2}^{n}\sum_{j=2}^{n}f(i+j)g(i)h(j)$$

$$c\sum_{k=4}^{2n}f(k)(g*h(k))$$

A little bit of math.

Convolution can be done in O(n log n) with Fast Fourier Transform.

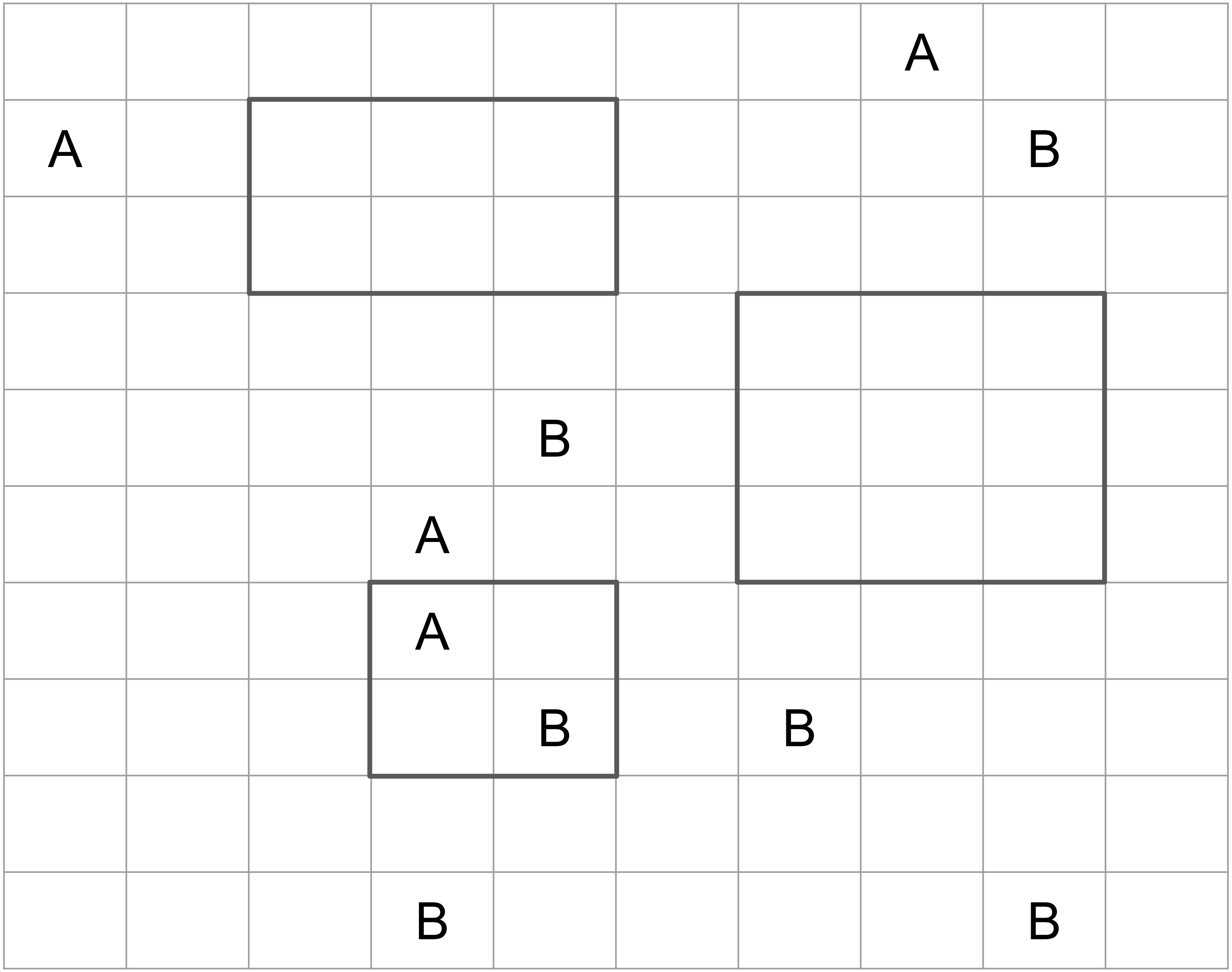
We also allowed Karatsuba’s O($n^{1.585}$) polynomial multiplication algorithm.

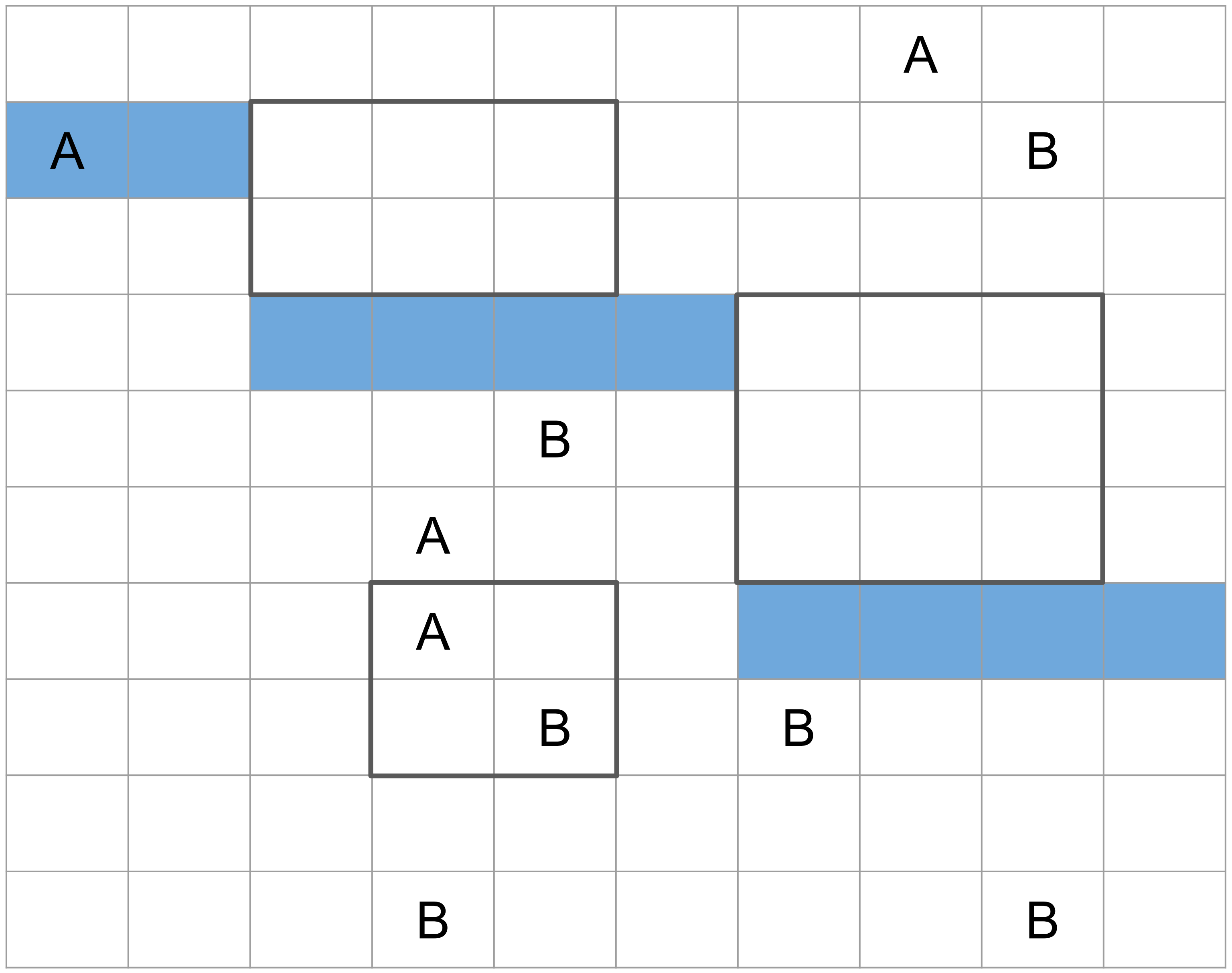
Problem C

Cow Confinement

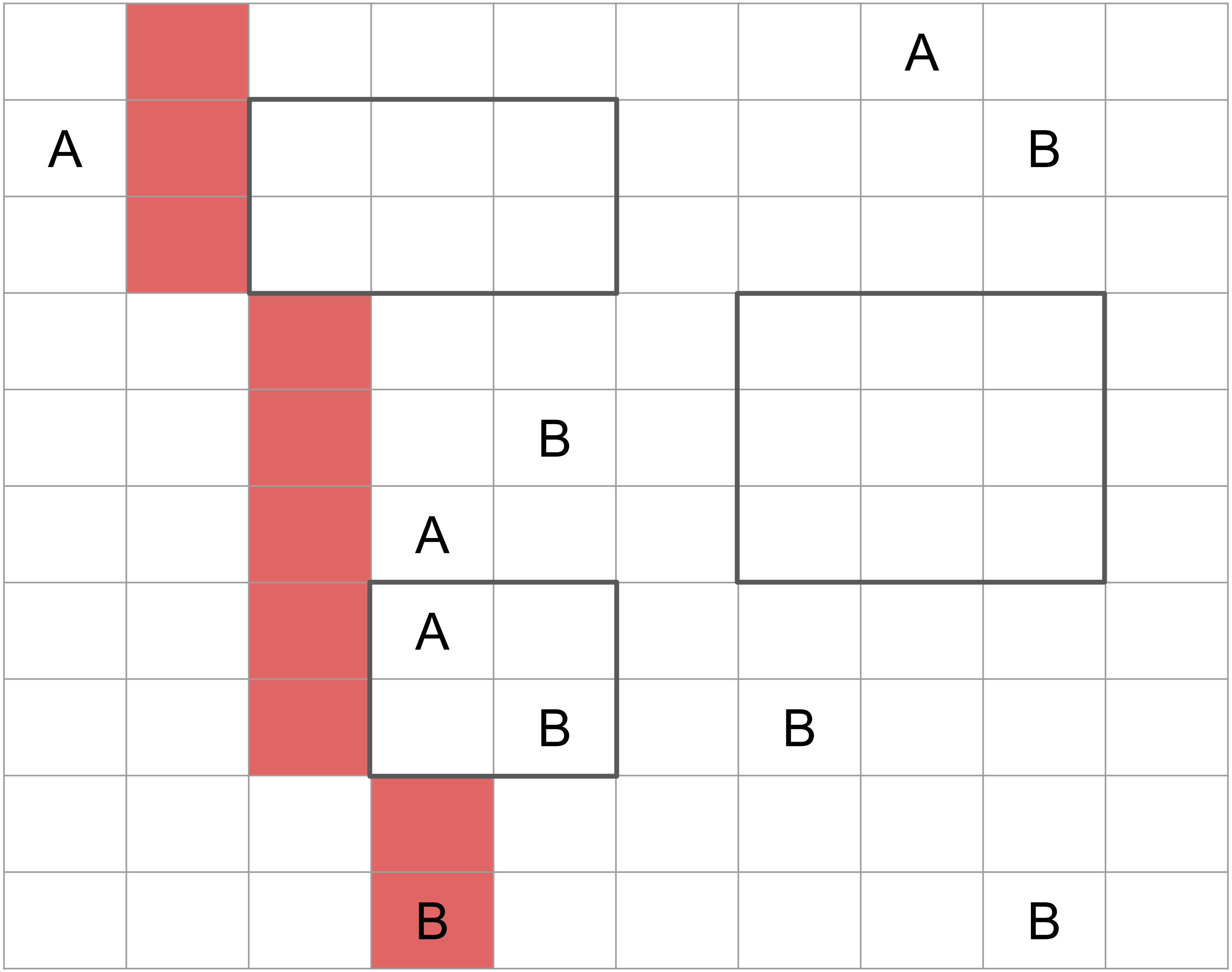
Submits: 3
Accepted: 0

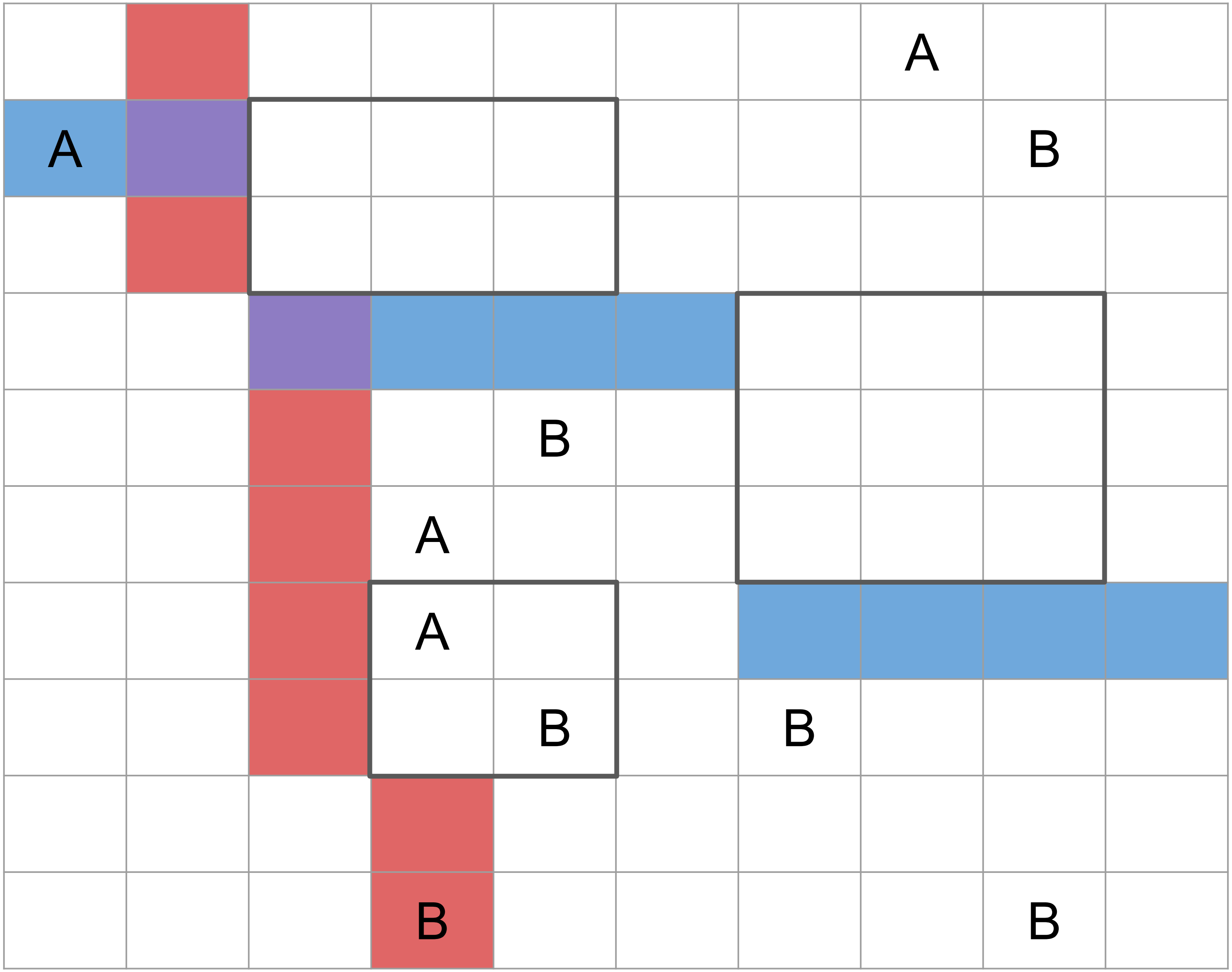
Authors: Luka Kalinovčić

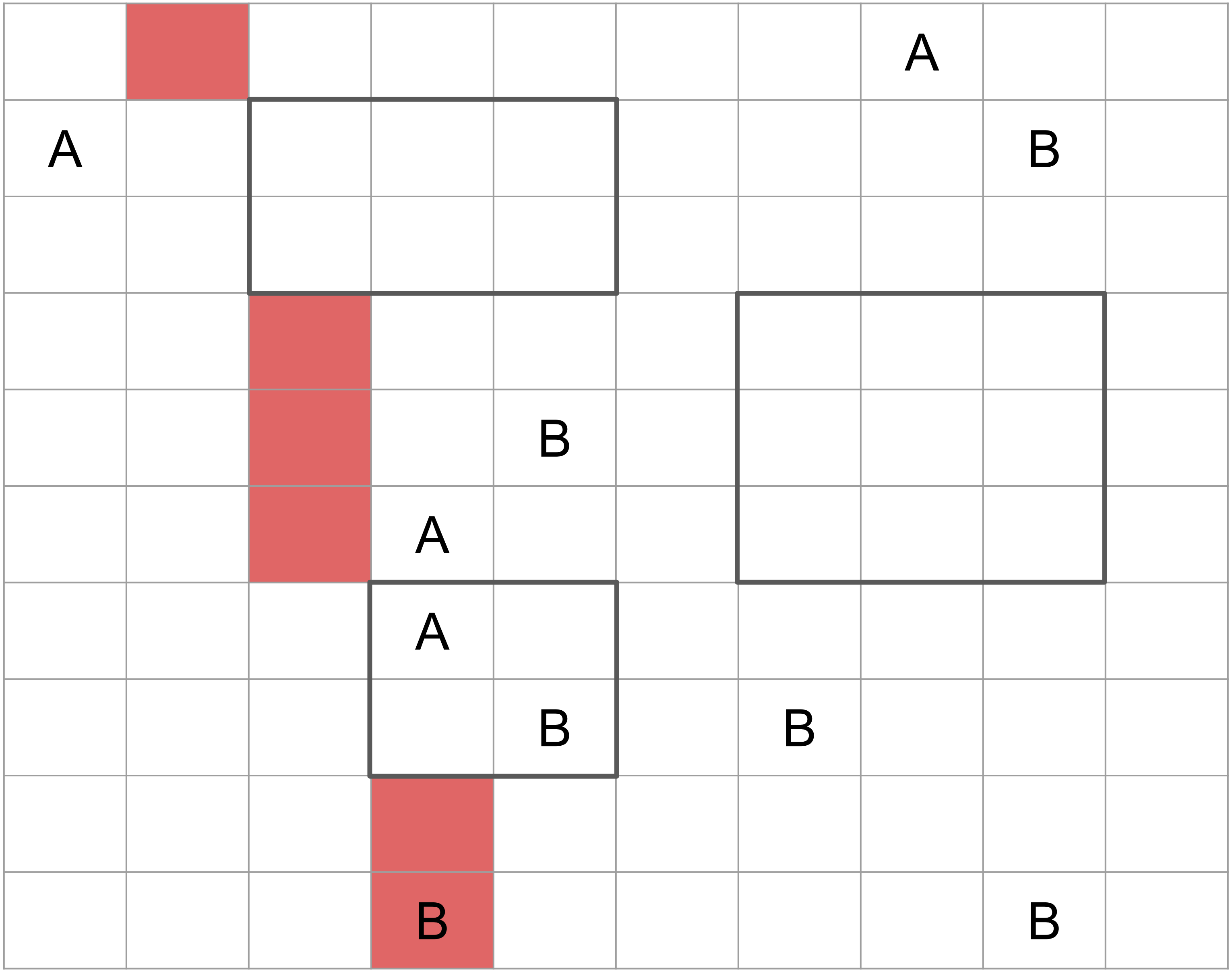


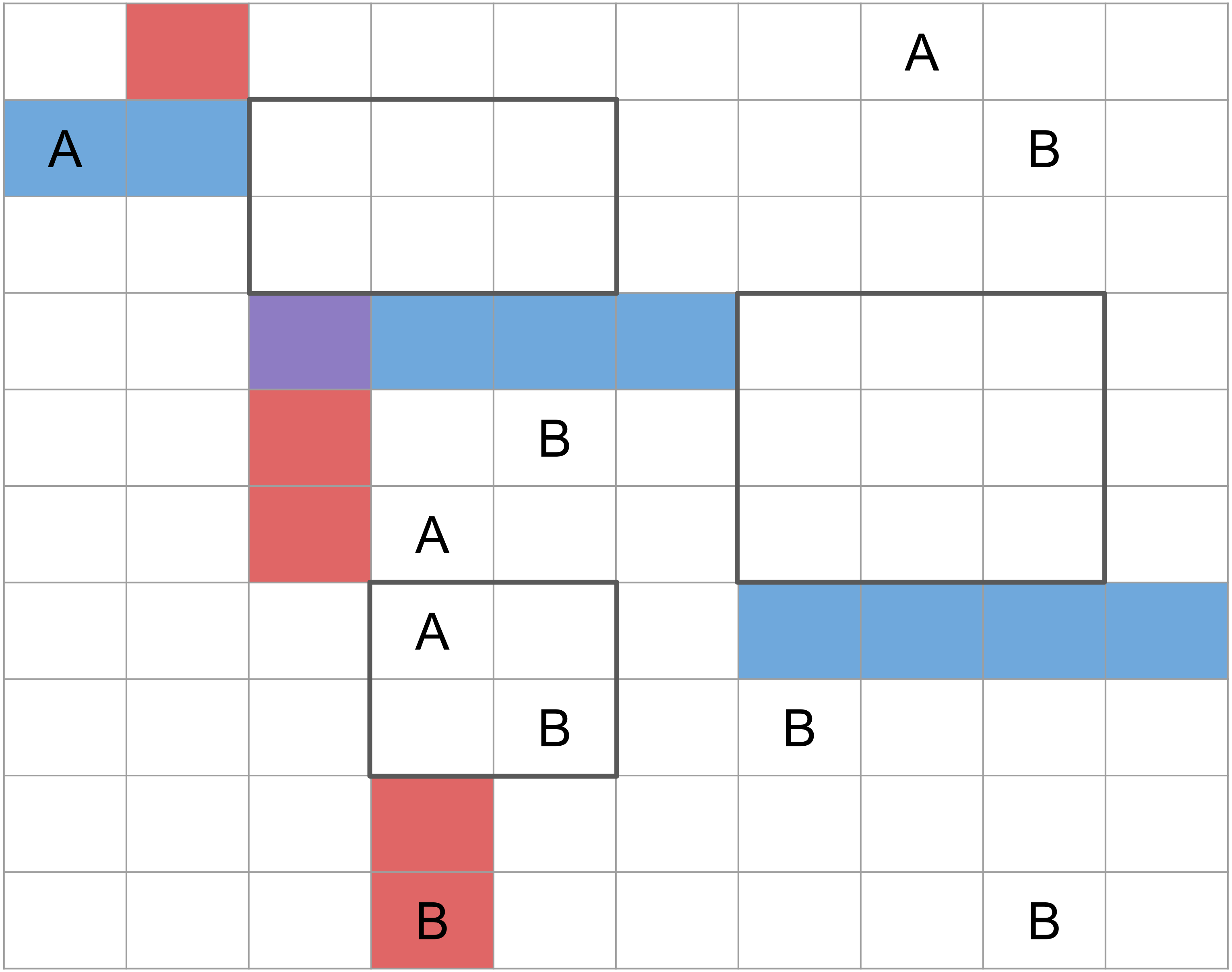


							A		
A								B	
				B					
			A						
			A						
				B		B			
			B					B	

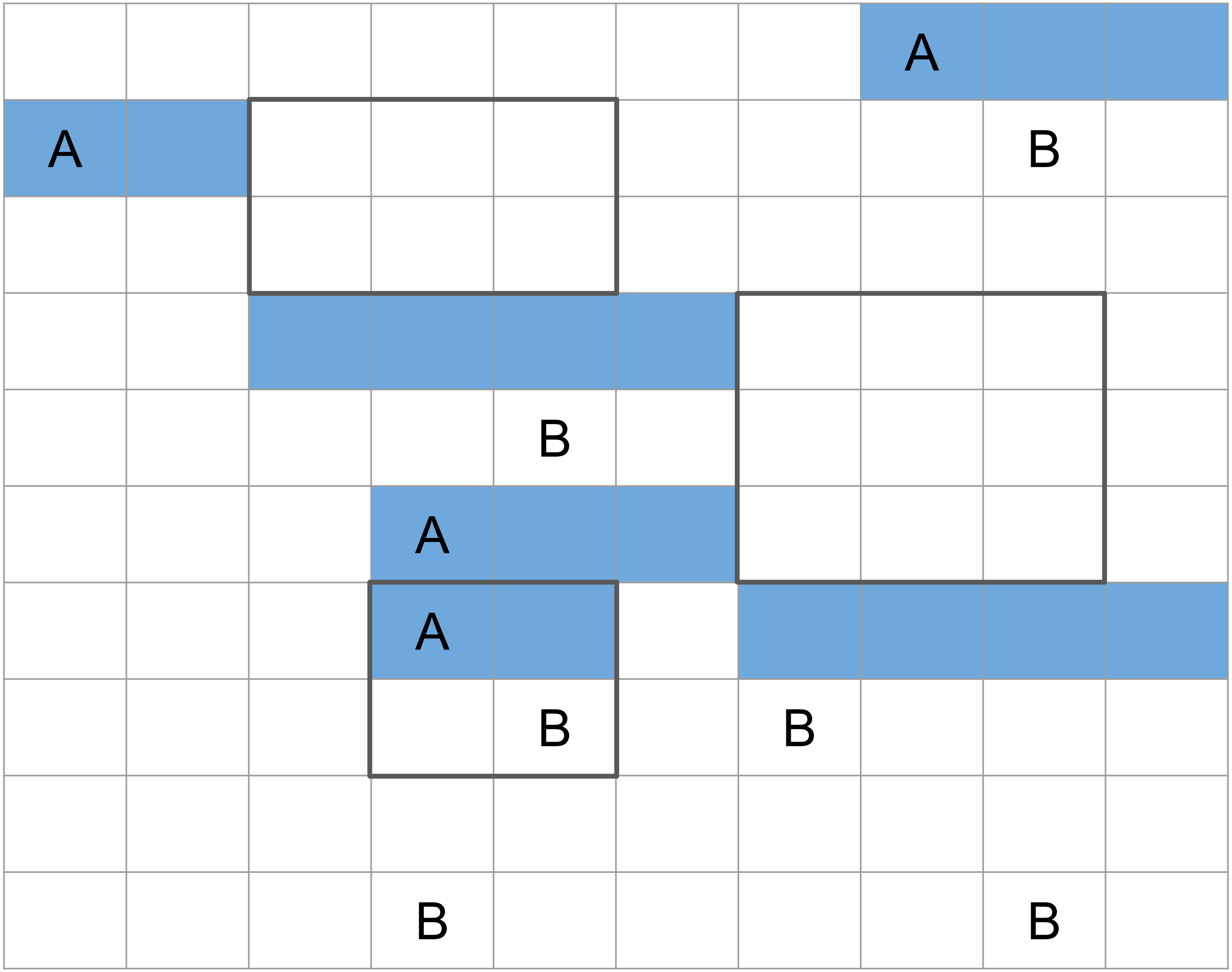

A
A
B
B
A
A
B
B
B
B

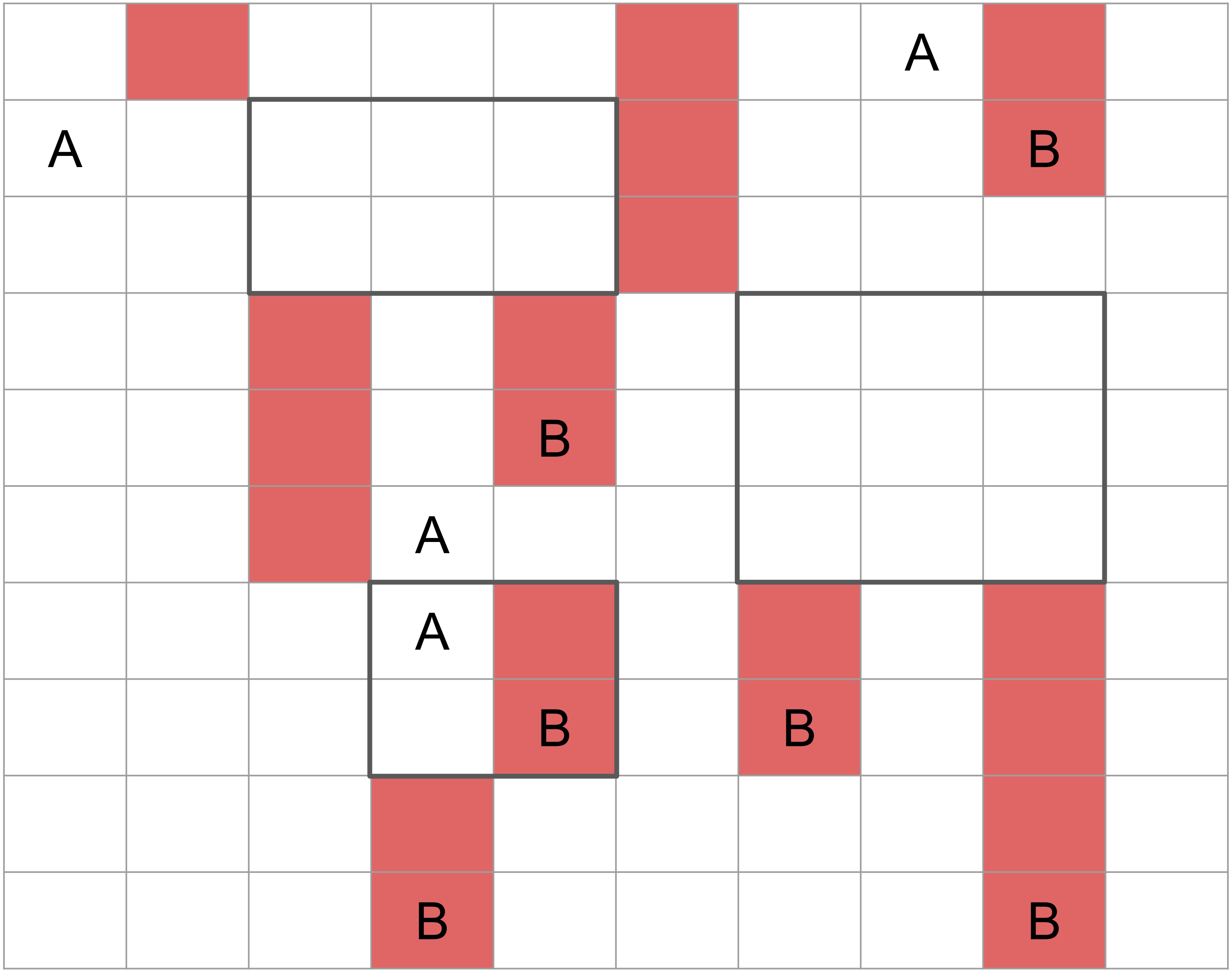

A
A
B
B
A
A
B
B
B
B

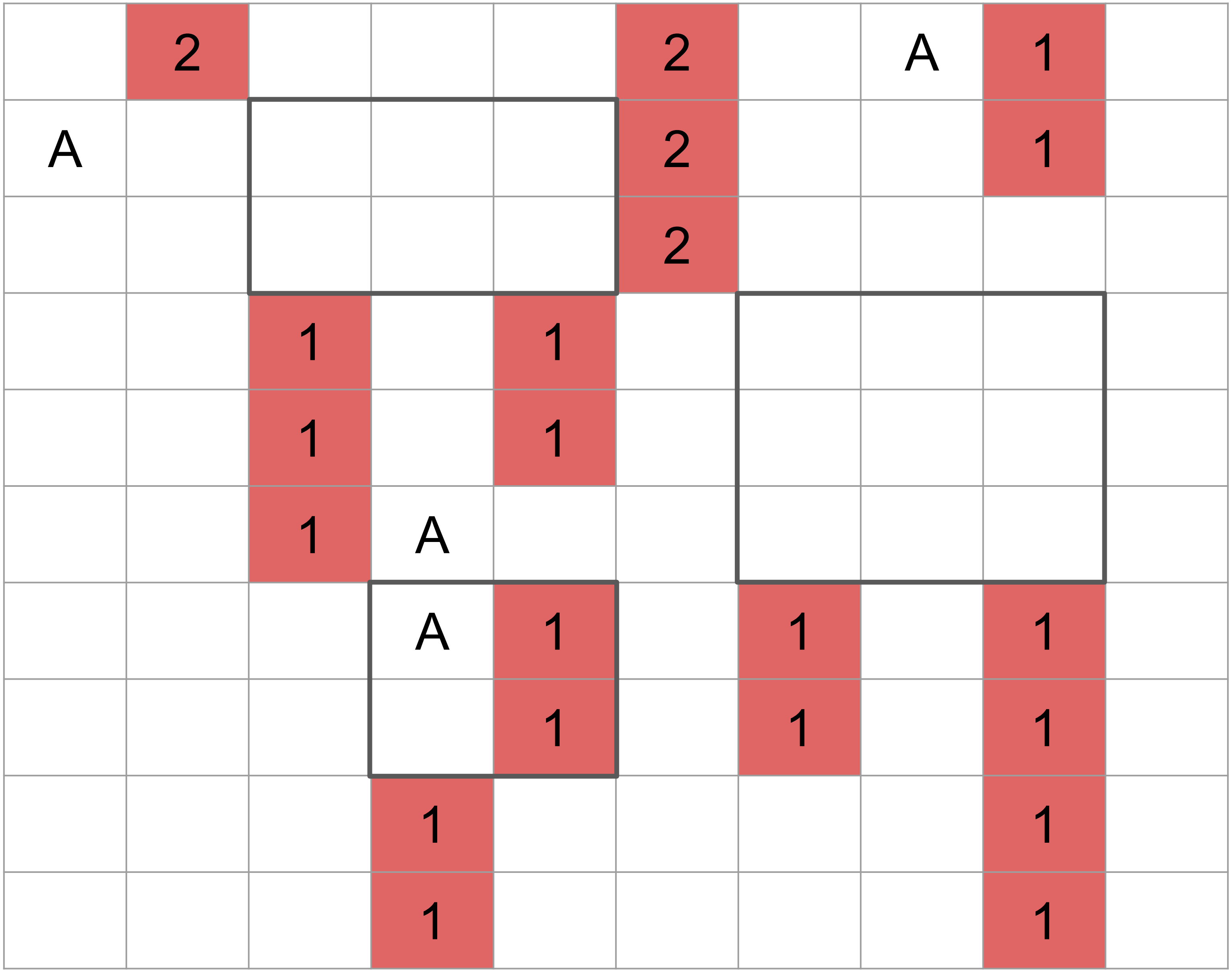

A
A
B
B
A
A
B
B
B
B



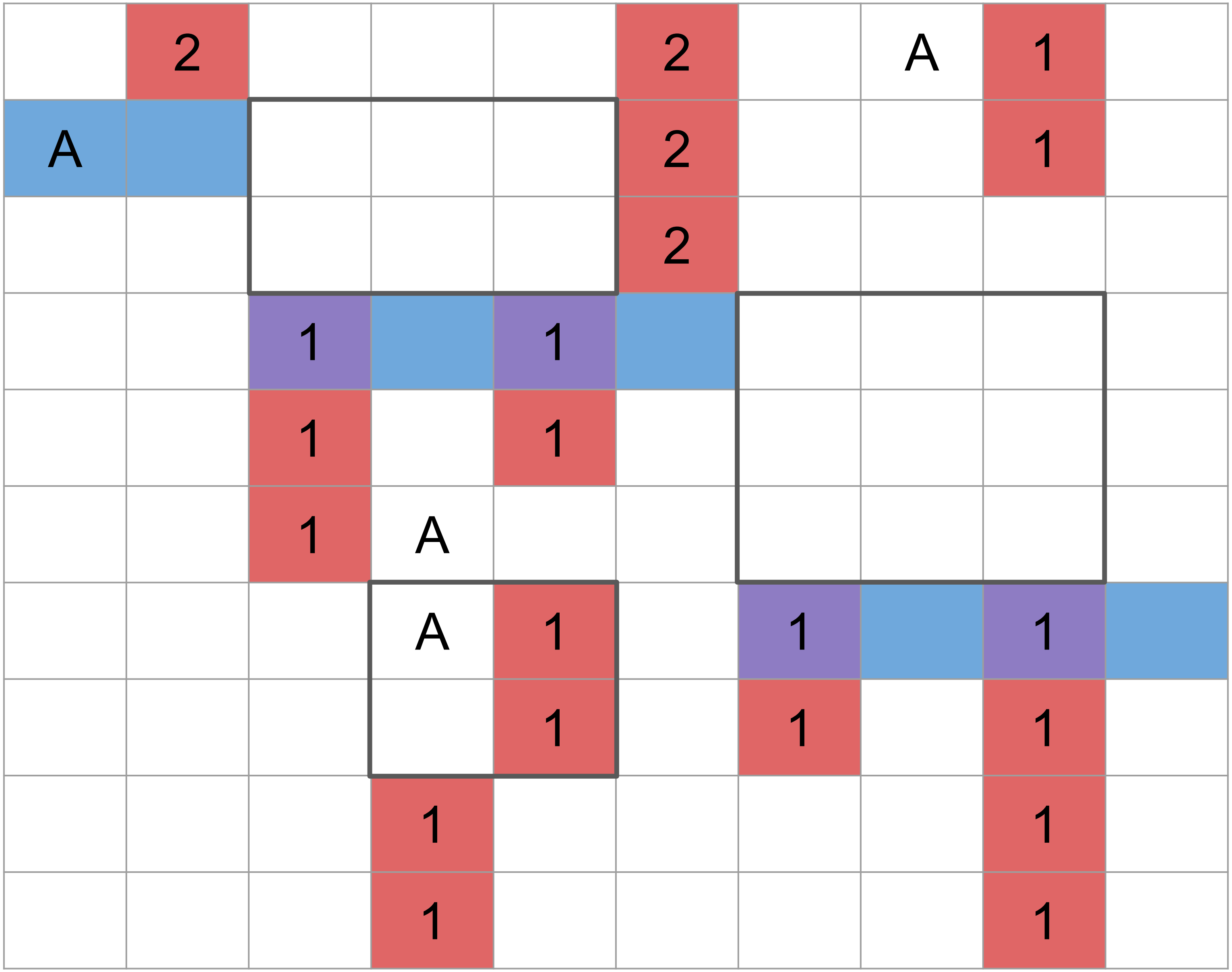
	🟥						A		
🟦 A	🟦							B	
		🟪	🟦	🟦	🟦				
		🟥		B					
		🟥	A						
			A			🟦	🟦	🟦	🟦
				B		B			
			🟥						
			🟥 B					B	


A
A
B
B
A
A
B
B
B
B

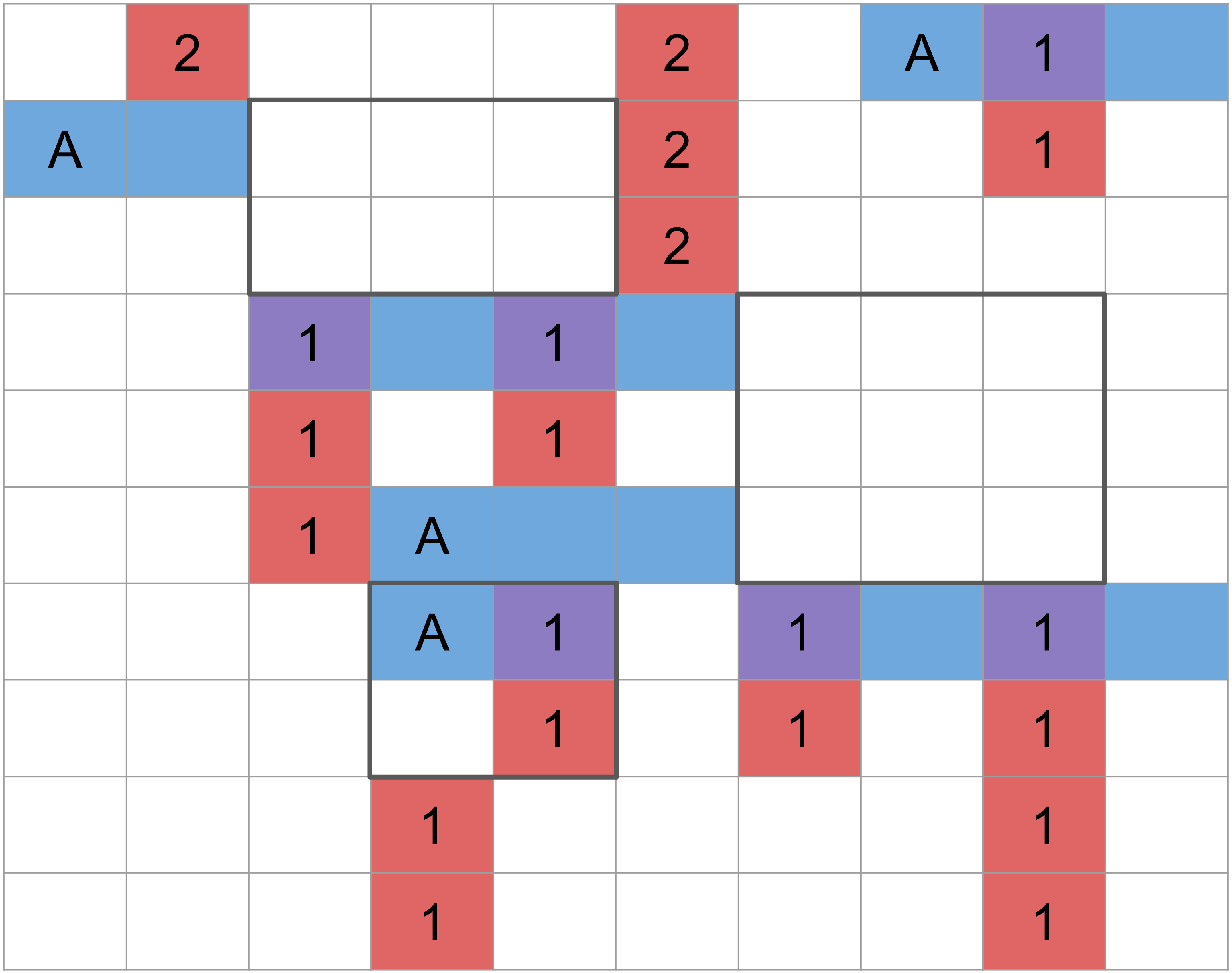

A
A
B
B
A
A
B
B
B
B



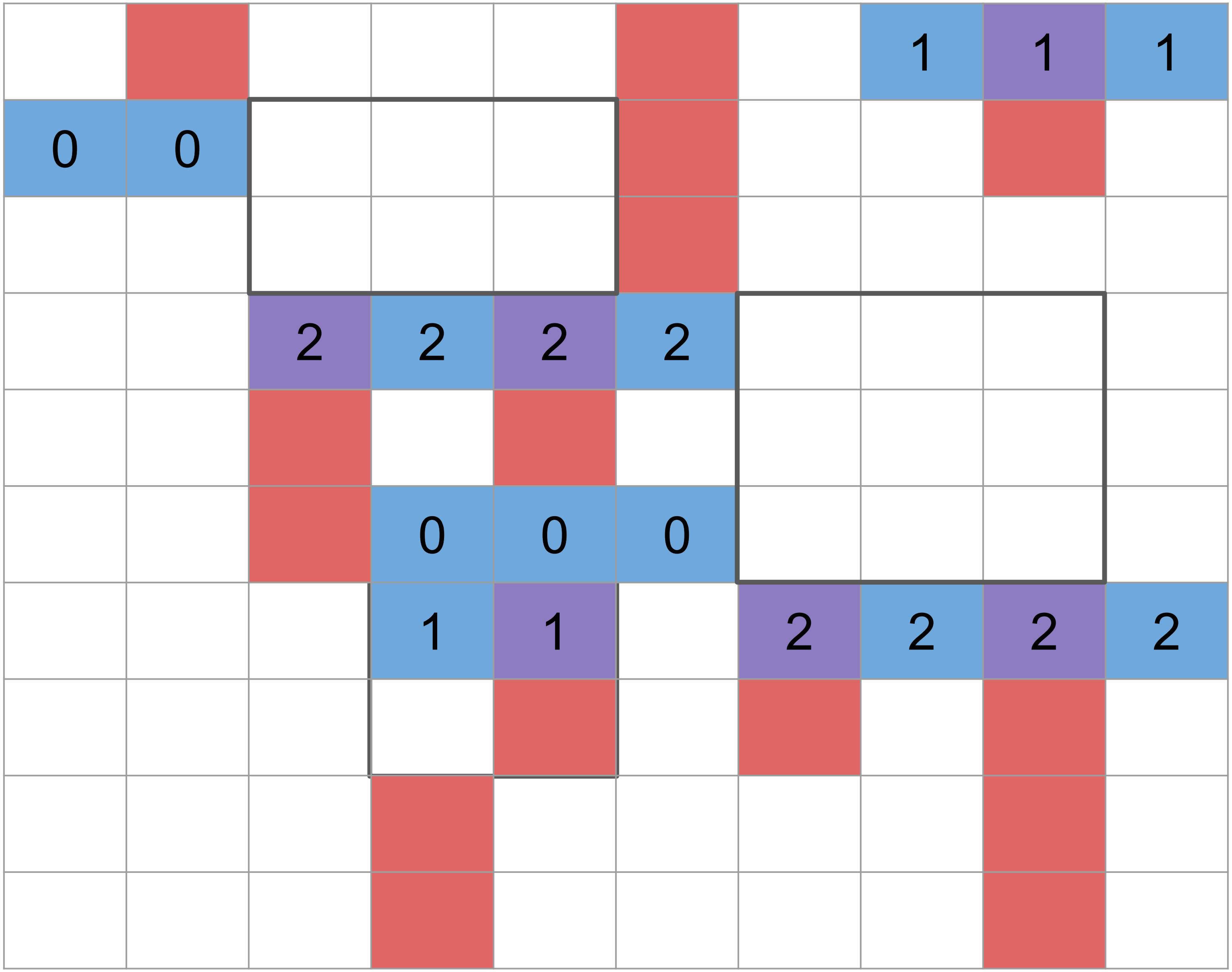
	2				2		A	1	
A					2			1	
					2				
		1		1					
		1		1					
		1	A						
			A	1		1		1	
				1		1		1	
			1					1	
			1					1	



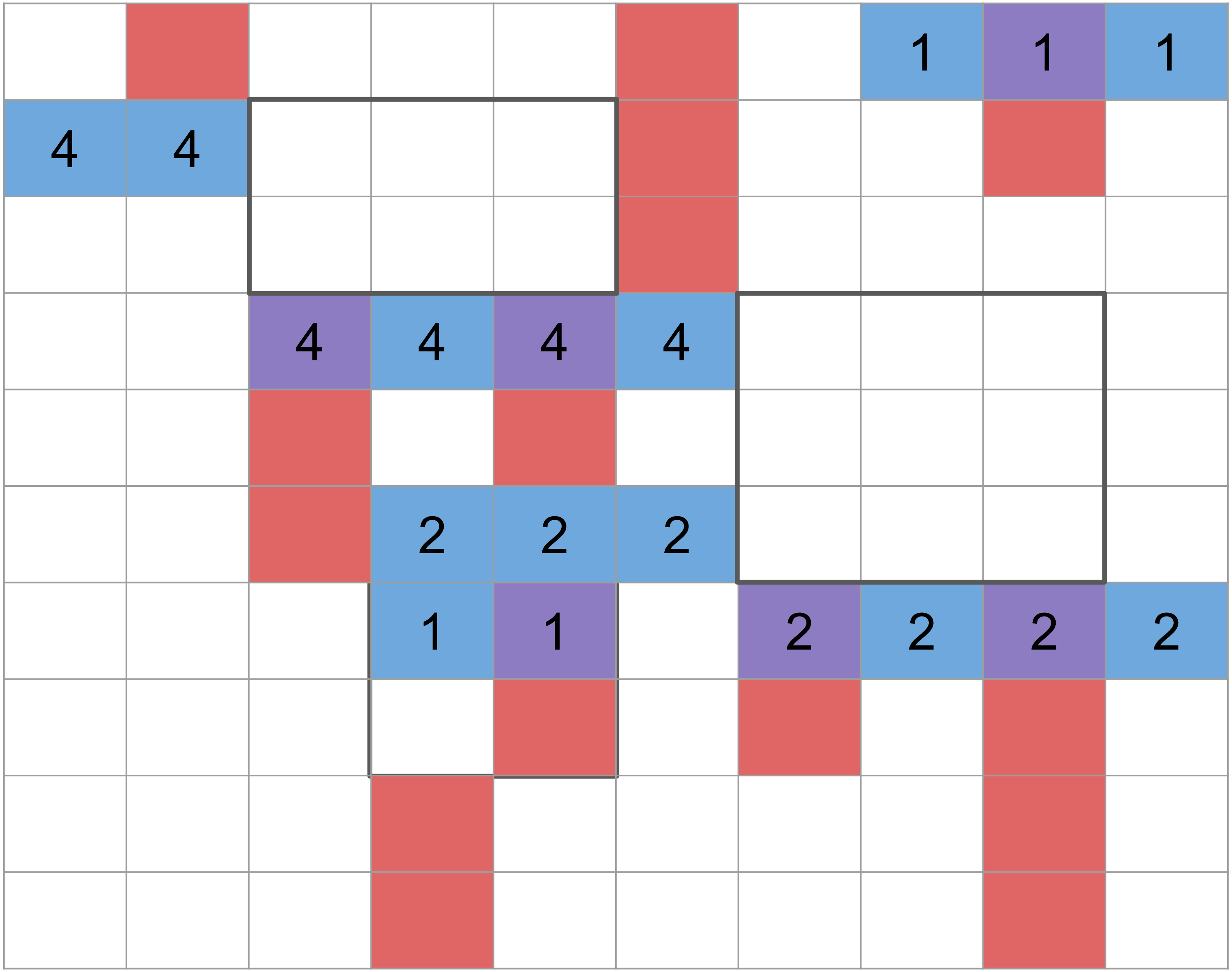
	2				2		A	1	
A	🟦				2			1	
					2				
		1	🟦	1	🟦				
		1		1					
		1	A						
			A	1		1	🟦	1	🟦
				1		1		1	
			1					1	
			1					1	



	2				2		A	1	
A					2			1	
					2				
		1		1					
		1		1					
		1	A						
			A	1		1		1	
				1		1		1	
			1					1	
			1					1	



	🟥				🟥		🟦 1	🟪 1	🟦 1
🟦 0	🟦 0				🟥			🟥	
					🟥				
		🟪 2	🟦 2	🟪 2	🟦 2				
		🟥		🟥					
		🟥	🟦 0	🟦 0	🟦 0				
			🟦 1	🟪 1		🟪 2	🟦 2	🟪 2	🟦 2
				🟥		🟥		🟥	
			🟥					🟥	
			🟥					🟥	



	🟥				🟥		🟦 1	🟪 1	🟦 1
🟦 4	🟦 4				🟥			🟥	
					🟥				
		🟪 4	🟦 4	🟪 4	🟦 4				
		🟥		🟥					
		🟥	🟦 2	🟦 2	🟦 2				
			🟦 1	🟪 1		🟪 2	🟦 2	🟪 2	🟦 2
				🟥		🟥		🟥	
			🟥					🟥	
			🟥					🟥	

Thanks!