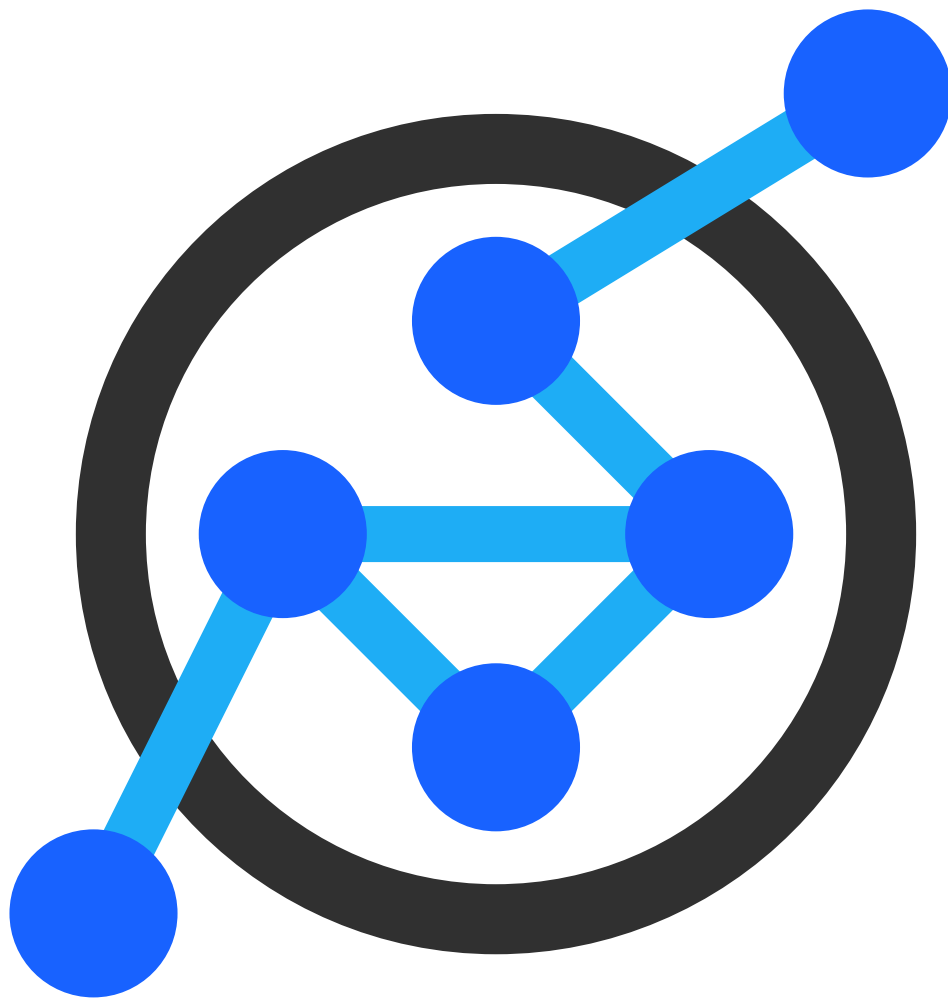


Round 1 SOI 2026/2027

Solution Booklet

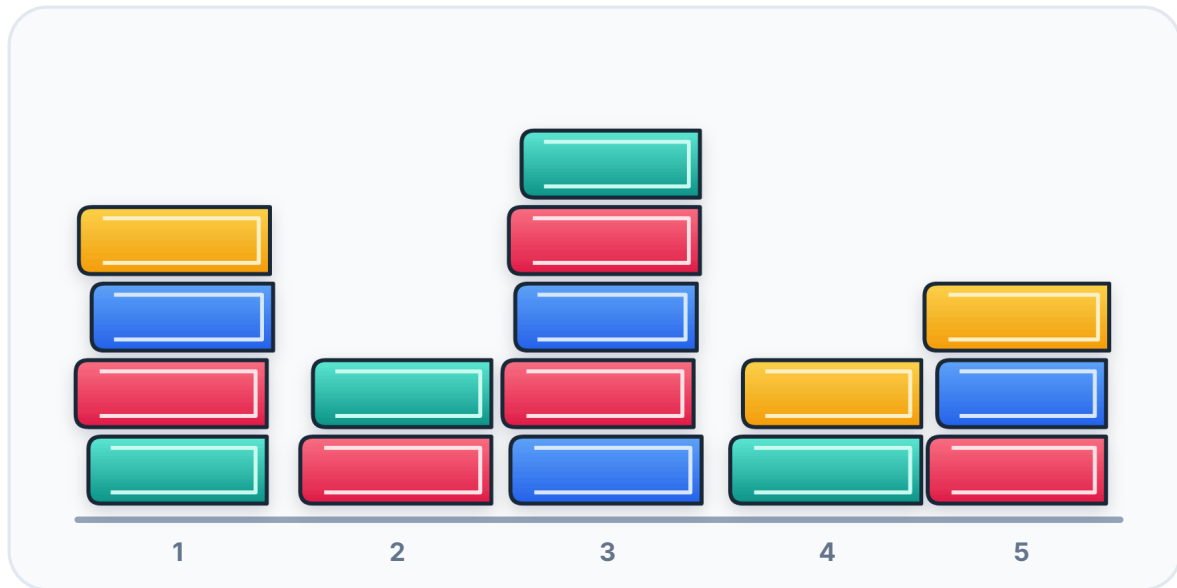


Swiss Olympiad in Informatics
01 September 2026 – 25 September 2026



Book Stacks

Task Preparation	Mark Neumann
Translation (de)	Benjamin Schmid
Translation (fr)	Cheng Zhong
Solution Author	Benjamin Schmid



Part 1: For his Matura exams, mouse Stofl needs to read 5 stacks of books, as depicted in the image. He noticed that the five stacks of books are of different heights, which he finds annoying. What is the minimum number of books that mouse Stofl would have to add to bring all the stacks to the same height? He can not remove any books, as he needs to read them all.

- ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10

Part 2: Mouse Stofl realized that he can also move books, one at a time, from one stack to any other stack. What is the minimum number of books you would have to move or add to bring all the stacks to the same height?

- ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6



Subtask 1: Adding Books

We need to add 9 books.

In this part, we can only add books. We can not remove or move any of the books. Therefore, a stack will never become lower than it initially was. Therefore, the best we can do is to find the tallest stack and add books to all other stacks to match the height of the tallest stack.

The tallest stack is stack 3 with 5 books. Stack 1 is missing 1 book, stack 2 is missing 3 books, stack 4 is missing 3 books, and stack 5 is missing 2 books. Hence, in total, we need to add 9 books.

Subtask 2: Moving Books

We have to perform 5 actions.

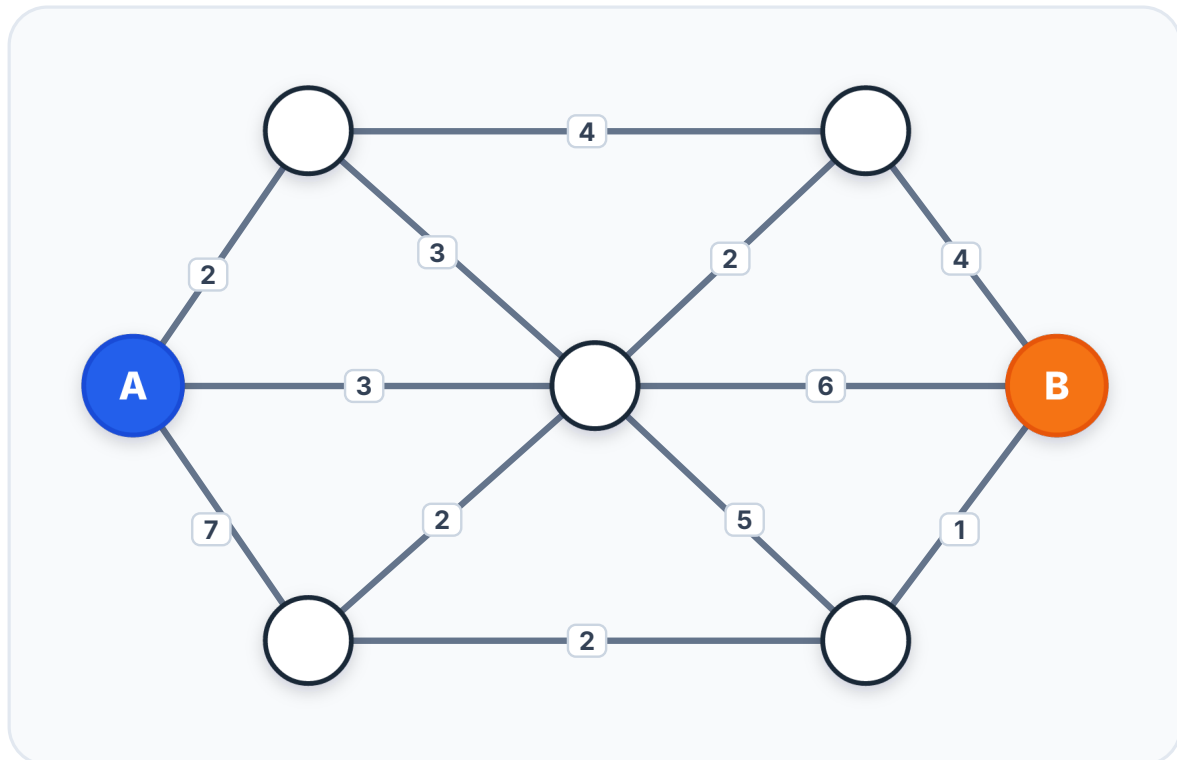
In the second part, we can also move books. As we saw in the first part, the number of books we need to add is determined by the tallest stack. We therefore want the tallest stack to be as small as possible. We achieve this by distributing the books evenly across the stacks. The tallest stack after distributing then tells us how tall each stack is going to be. Note that the tallest stack is at most one book taller than the lowest stack if we distribute the books evenly.

In total we have 16 books. With only 3 books per stack we can only have 15 books. Therefore, we need 4 books per stack (which would allow a total of 20 books). To achieve this, we can for example move one book from stack 3 to stack 5 and then add two books each on stack 2 and 4. This results in a total of 5 actions.



Shortest Path

Task Preparation	Mark Neumann
Translation (de)	Benjamin Schmid
Translation (fr)	Cheng Zhong
Solution Author	Benjamin Schmid



Part 1: Mouse Binna is currently living in city A and needs to get to city B as fast as possible. Some cities are connected with a road, each of which can be traversed in either direction. For each road we know how long it takes to traverse it. In the image, a map of the network of cities and roads (with the corresponding travel time) is displayed. What is the shortest possible travel time between city A and city B?

- ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐ 11

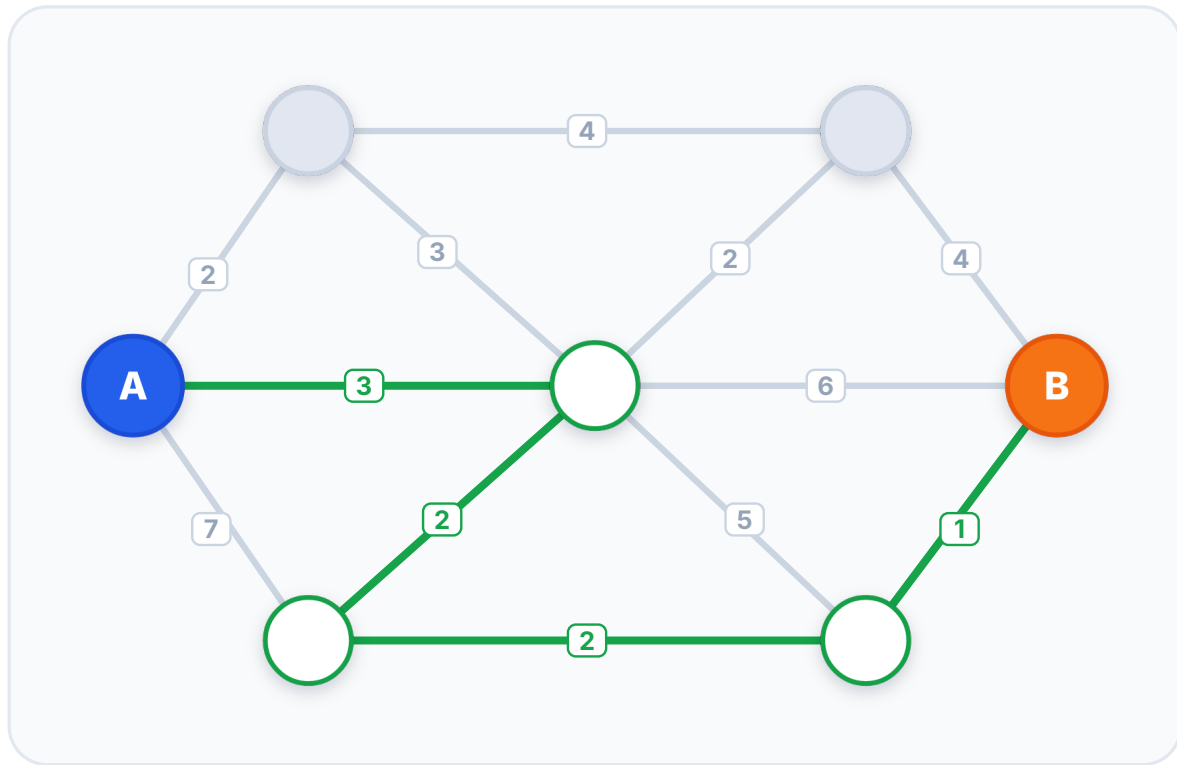
Part 2: Due to a storm, at most one of the roads may get shut down. What is the minimum amount of time mouse Binna should plan for such that, regardless of which road gets shut down, there will be a path for mouse Binna to get from city A to city B in that time? Note that Binna will know which road is closed before starting her trip.

- ☐ 8 ☐ 9 ☐ 10 ☐ 11 ☐ 12



Subtask 1: Direct Path

The shortest possible travel time is 8.



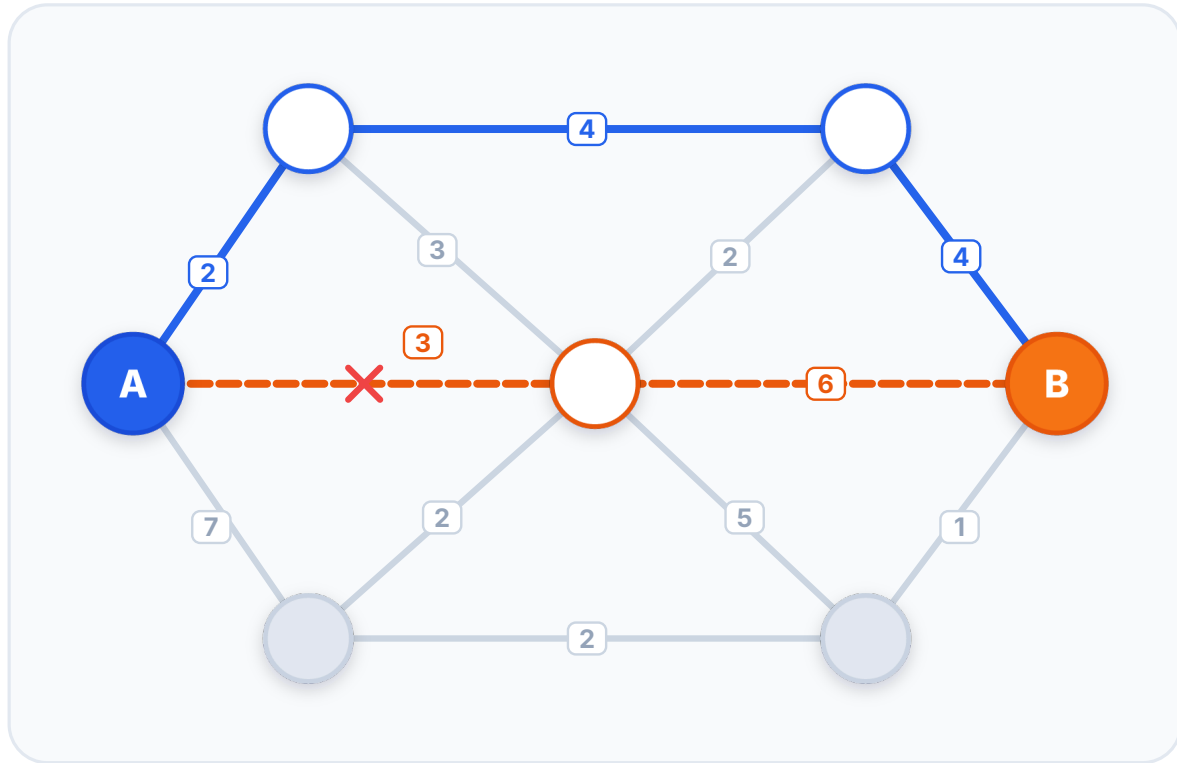
In computer science, this kind of network with nodes (= cities) and edges (= roads) is called a graph. A graph shows some entities (nodes; in our case cities) and their connections (edges; in our case roads). An edge can have a weight to denote some kind of “distance” this connection represents.

To find the shortest path in a graph we can use one of several algorithms solving this problem, the most famous one being Dijkstra’s algorithm.



Subtask 2: Blocked Roads

Mouse Binna needs to plan for a travel time of at least 10.



As mouse Binna knows the affected road beforehand, we can simulate different cases for which road might be affected. A first observation is that if the affected road is not on the shortest path from part 1, her travel time does not increase. Therefore, we only need to consider cases where one of the roads on the shortest path is unavailable. We can remove each edge of the shortest path one at a time and recalculate the shortest route. Out of all these routes, the solution is the maximum of what we found.

If we remove the edge of length 3 on the left (shown crossed out in the image above), the shortest path we find has length 10, taking always the topmost edge (blue path). If we remove any of the other edges of the original shortest path, we can always take the path straight ahead with a length of 9 (orange path).



Mouse Maze

Task Preparation

Mark Neumann

Translation (de)

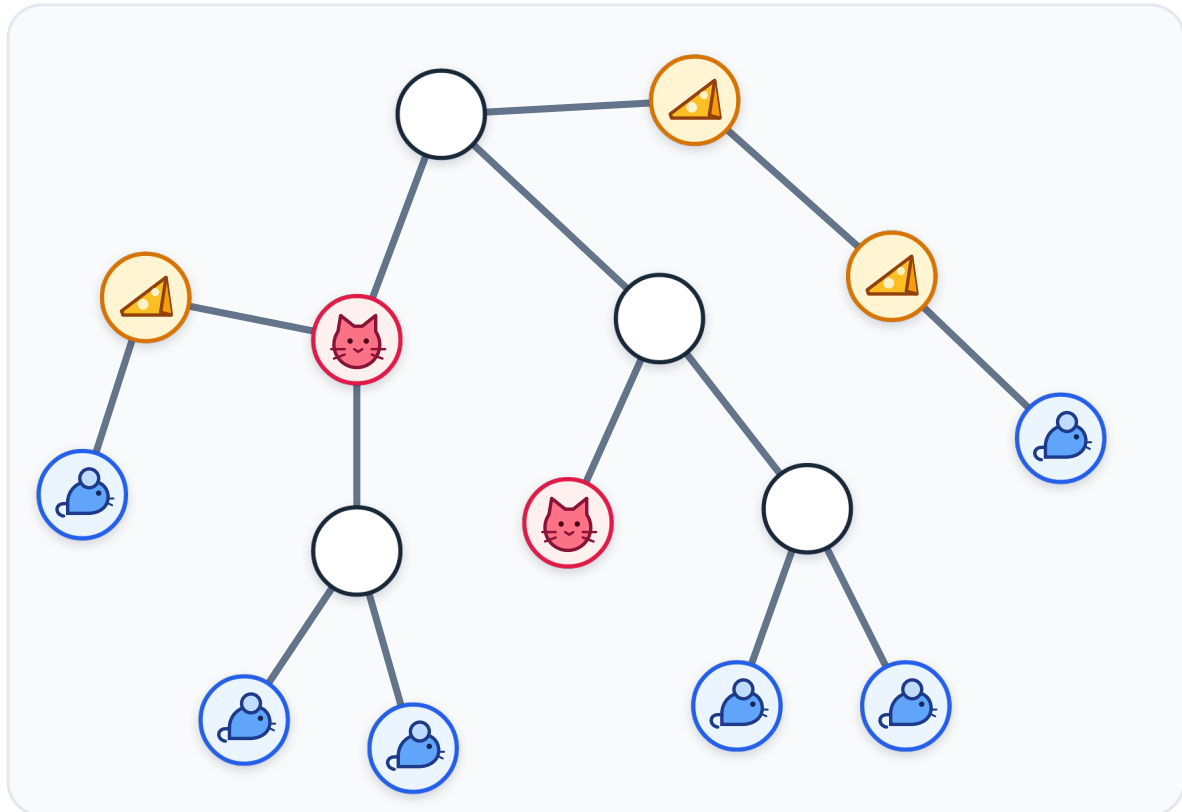
Benjamin Schmid

Translation (fr)

Cheng Zhong

Solution Author

Benjamin Schmid



Part 1: You have just built a mouse maze. The maze consists of various rooms connected by passages, as depicted in the image.

Some rooms in the maze contain mice, while others contain pieces of cheese. Some rooms are guarded by a cat, which the mice cannot pass. In the depicted setup, how many different (mouse, cheese) pairs exist where the mouse can successfully reach the cheese, without going through a room guarded by a cat? A single mouse can be paired with multiple pieces of cheese, and vice versa.

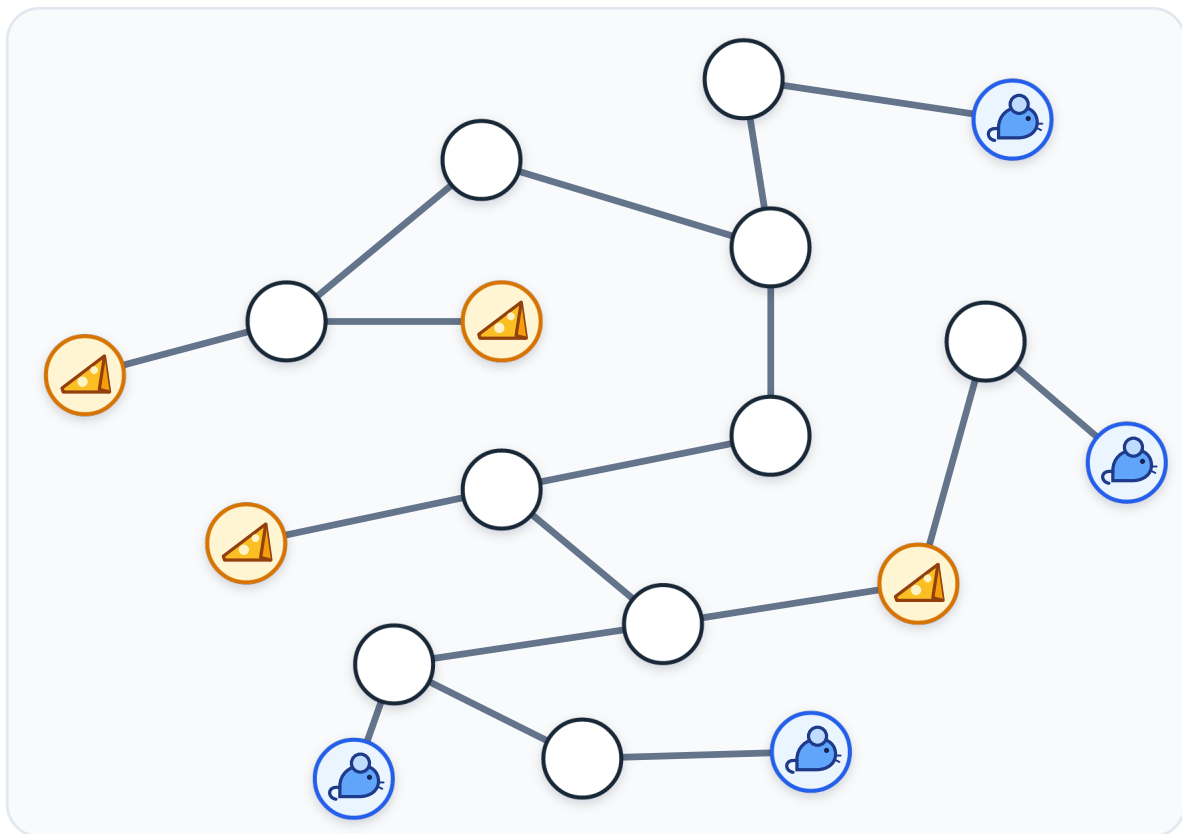
☐ 4

☐ 6

☐ 7

☐ 10

☐ 12



Part 2: Snack time is over, and now you want to prevent the mice from reaching the cheese. The maze now looks as shown in the image.

What is the minimum number of rooms where you must place a cat so that no mouse can reach any piece of cheese? You can not place a cat in a room with a mouse or with cheese.

- ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6



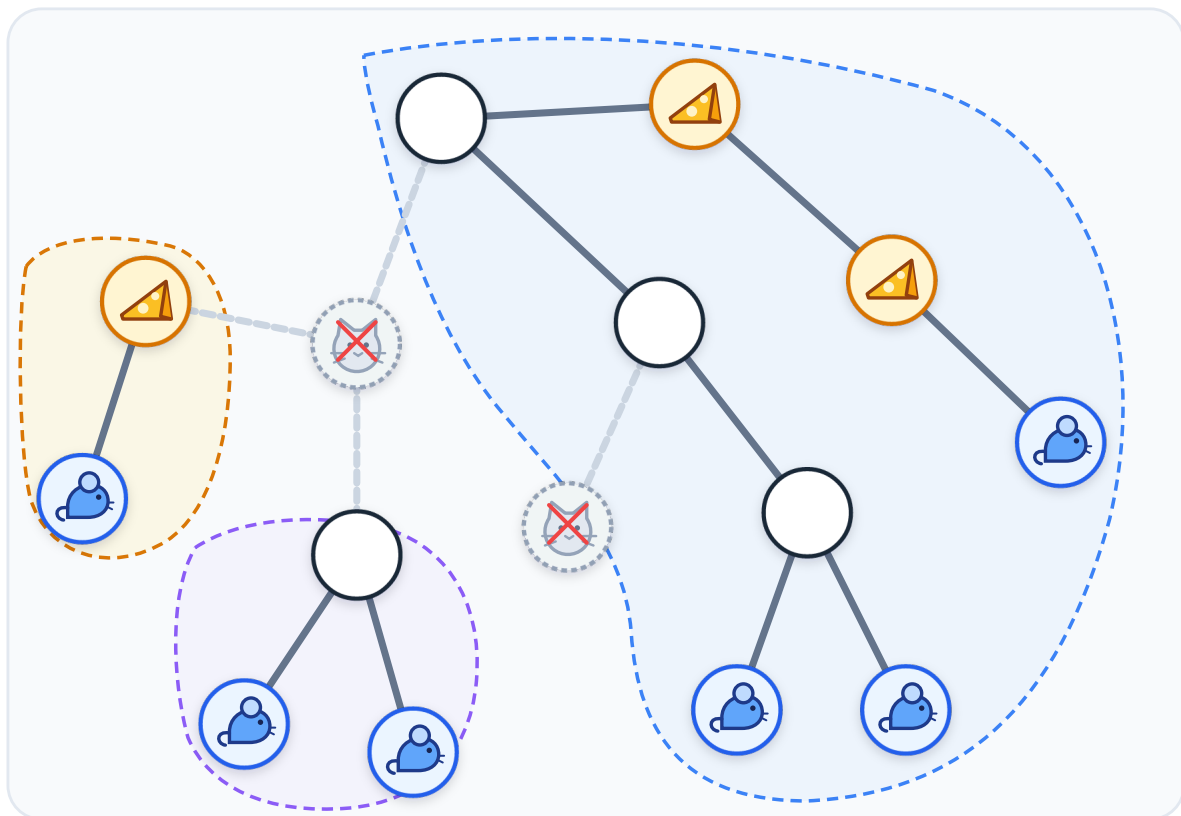
Subtask 1: Count the Pairs

There are 7 (mouse, cheese) pairs where the mouse can reach the cheese.

This representation of the maze with nodes (= rooms) and edges (= passages) is called a graph. In the first part we are interested in exploring components in the graph. A component is a set of some nodes of the graph which are connected by edges, either directly or indirectly.

Initially, there is only one component: ignoring the cats, we can reach every node from every other node. However, we know that a node with a cat on it can not be traversed, so we can remove it from the graph. This leads the graph to break up into several components. A mouse in one component can reach all the cheese in the same component, but it can not reach any cheese in another component. Therefore, we can look at each component separately and multiply the number of mice with the number of cheese in that component to get the number of pairs in that component. Then, we add all the components to get the total.

Thus, the result is $1 \cdot 1 + 2 \cdot 0 + 3 \cdot 2 = 1 + 0 + 6 = 7$



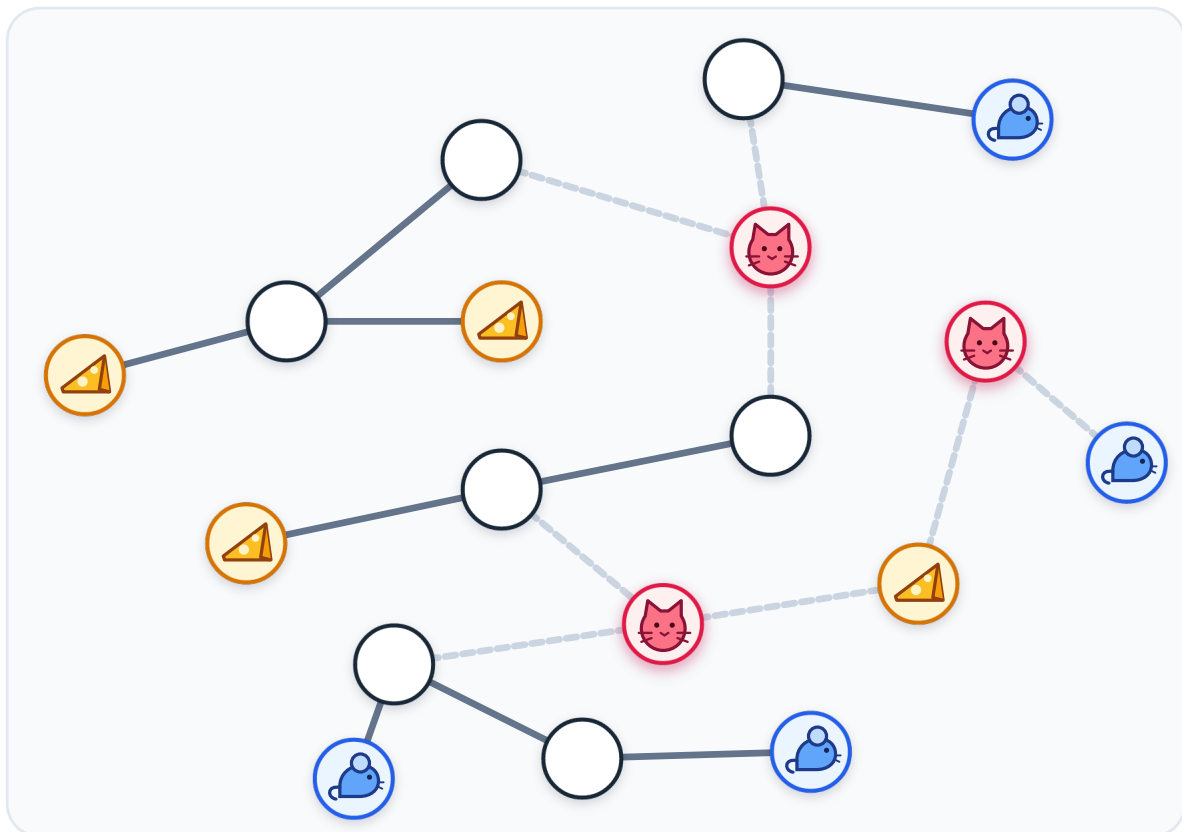
The three components are highlighted with different colors: the right component (blue) contains three mice and two cheese ($3 \cdot 2 = 6$ pairs), the left component (yellow) contains one mouse and one cheese ($1 \cdot 1 = 1$ pair), and the bottom component (purple) contains two mice but no cheese ($2 \cdot 0 = 0$ pairs).



Subtask 2: Snack Time is Over

We need at least 3 cats.

In the second part we need to choose nodes to place cats such that the resulting components all only have mice or cheese, but not both. There are multiple options, one of them is displayed below.



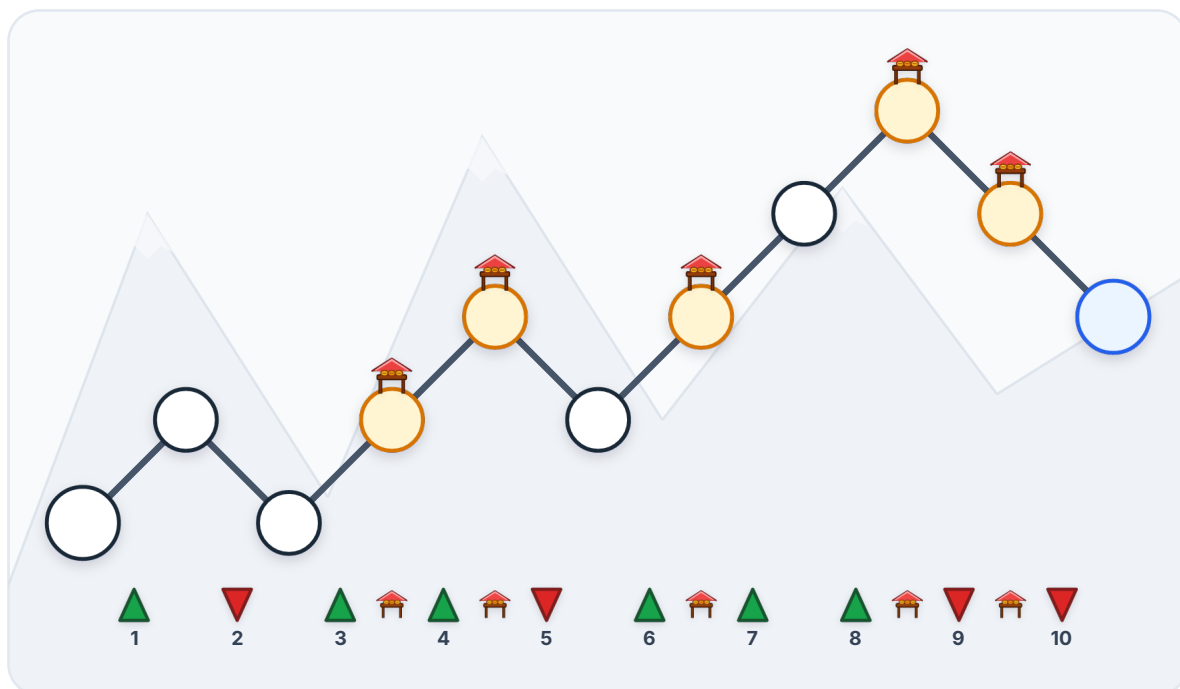
The three cats are placed in the rooms marked in red in the diagram. This splits the maze into six components, each containing either only mice or only cheese.

With such a small problem size, this can be done by hand by trying a few different nodes. A general solution could use the max-flow algorithm to find the min-cut in a specially crafted graph. The explanation of this is outside the scope of this solution booklet.



Hiking

Task Preparation	Mark Neumann, Benjamin Schmid
Translation (de)	Benjamin Schmid
Translation (fr)	Cheng Zhong
Solution Author	Benjamin Schmid



Part 1: You want to go for a mountain walk, bringing 3 sandwiches with you.

The walk consists of 10 sections and passes by some bread kiosks, as shown in the image. For each section, you will follow this two-step sequence before you start walking:

Look at your current starting point for the section.

- If there is a kiosk: You buy 2 sandwiches.
- If there is no kiosk: You buy 0 sandwiches.

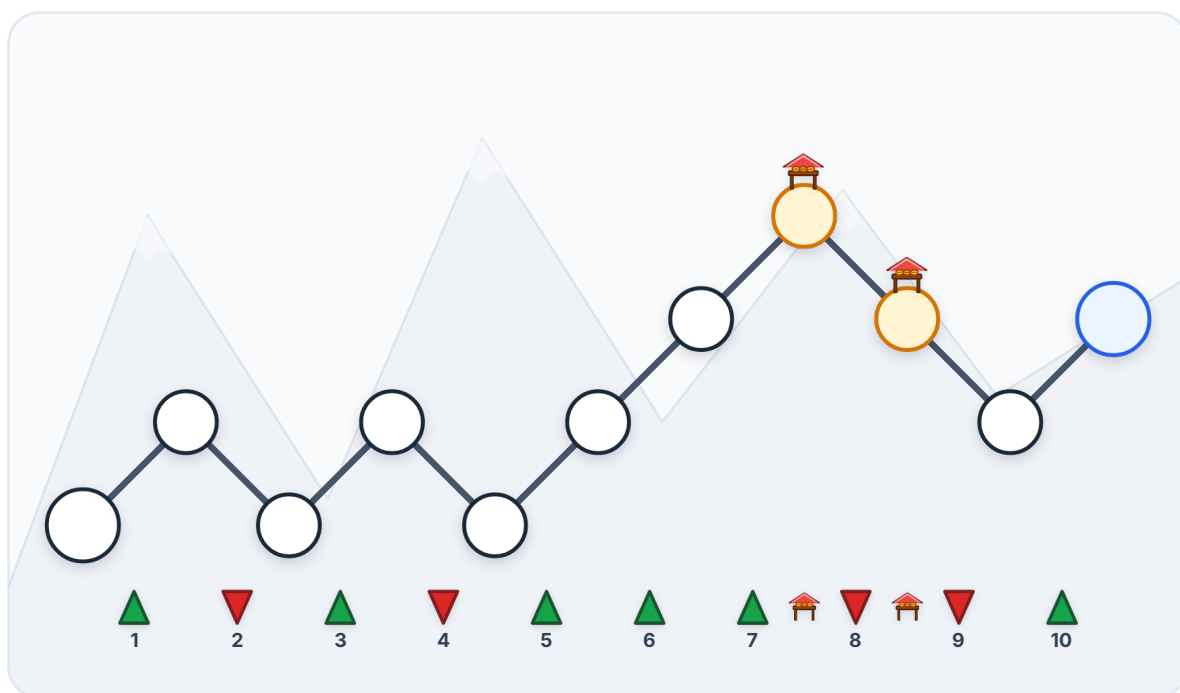
Look at the section you are about to walk.

- If the path is uphill and you are at a kiosk: You eat 2 sandwiches.
- If the path is uphill and you are **not** at a kiosk: You eat 1 sandwich.
- If the path is downhill: You eat 0 sandwiches.

How many sandwiches will you have left at the end of the walk?

Note: If you do not have enough sandwiches you will have to abort your walk.

☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 8



Part 2: Once you arrive at the starting point, you realize you forgot your sandwiches! There is only one solution: hope that new kiosks open along the way. New kiosks can be added at the beginning of any section (including the first one) where there isn't already a kiosk.

What is the minimum number of new kiosks that must be added so that you can successfully complete the entire walk following the rules, even though you started with zero sandwiches?

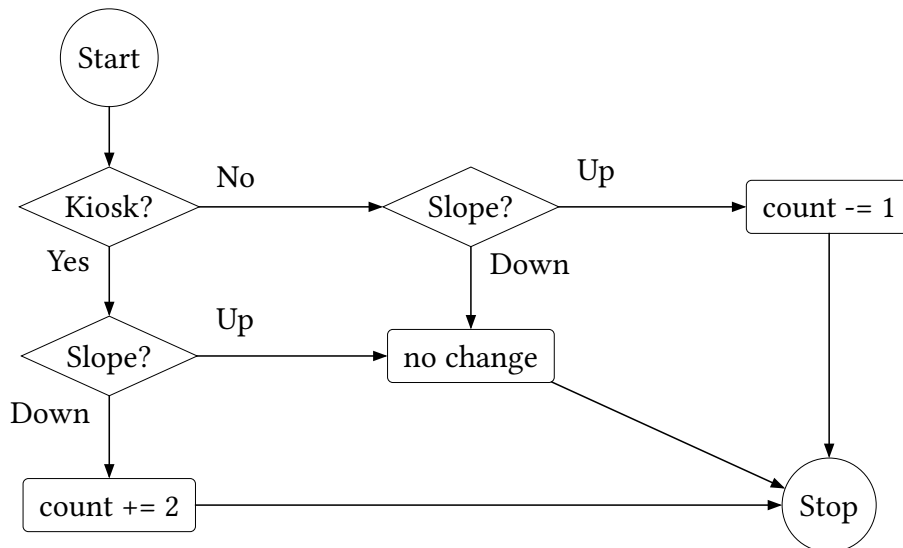
- ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4



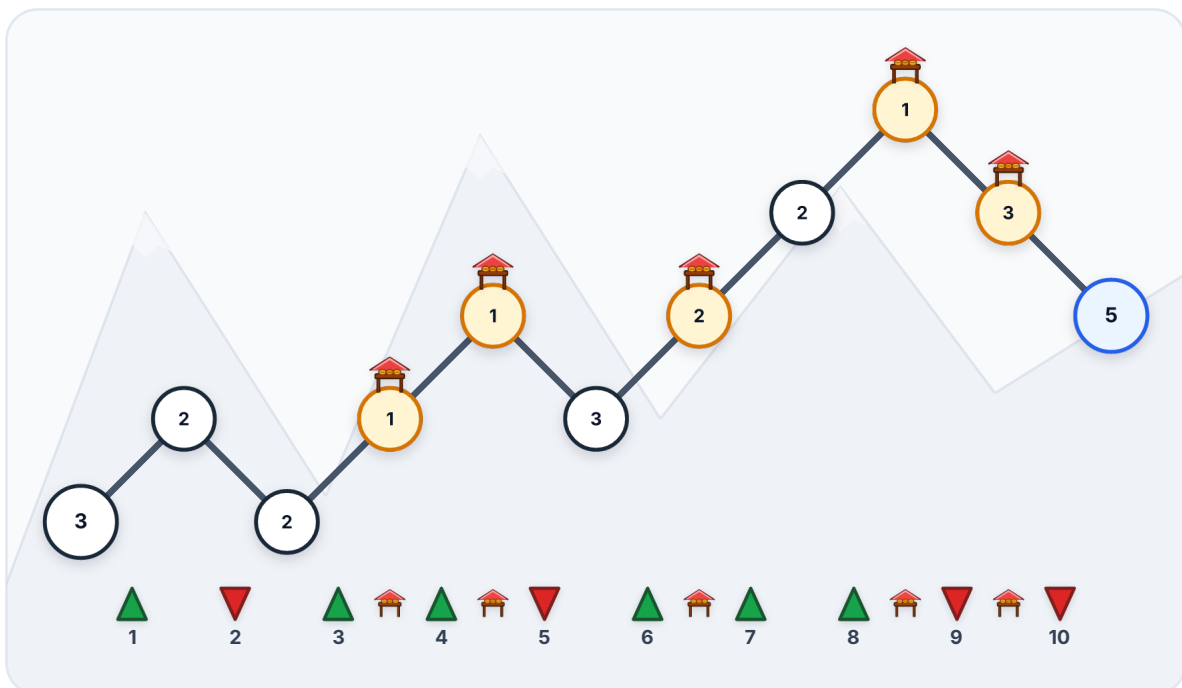
Subtask 1: Three Sandwiches

We end up with 5 sandwiches at the end of the walk.

This task is about executing a simple algorithm and figuring out the result. We can simplify the algorithm slightly by just noting what happens with the number of sandwiches depending on whether there is a kiosk or not and depending on the slope. We initialize the number of sandwiches (“count” below) to 3, then execute the following for each section:



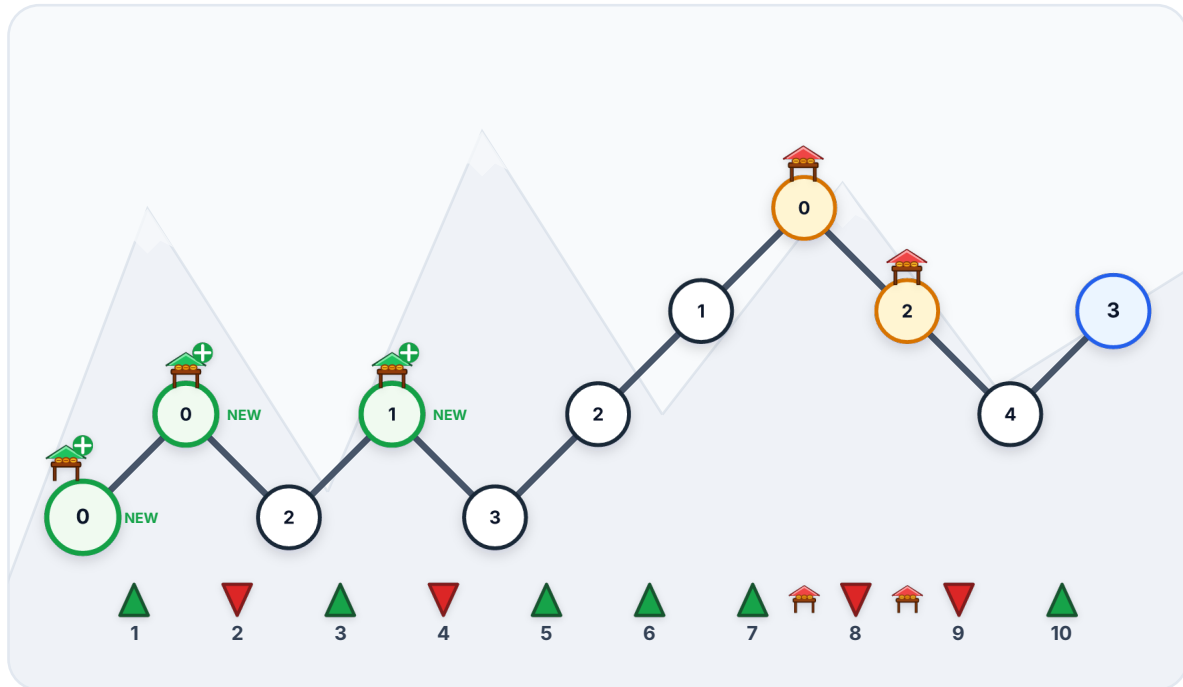
We end up with the following numbers of sandwiches on the different nodes:





Subtask 2: The Forgotten Sandwiches

We need to add at least 3 kiosks.



In this part, we need to figure out how many kiosks we need to place and where to place them. We can make a few observations:

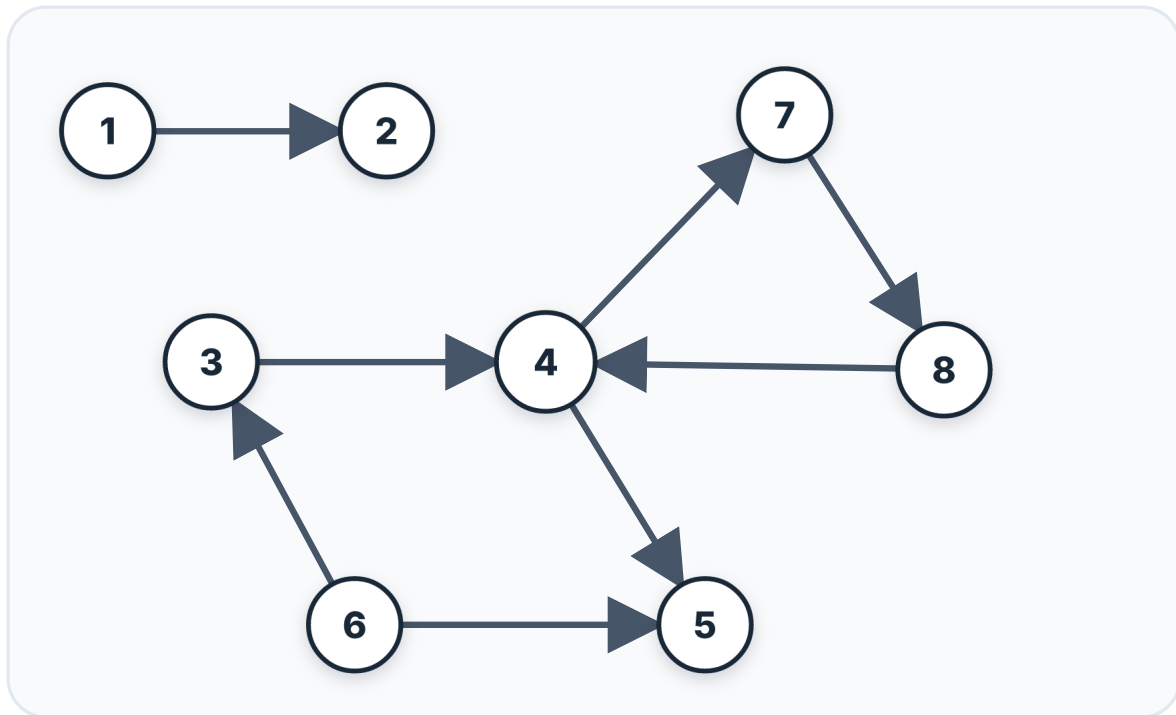
- If we place a kiosk before an uphill section, this allows us to traverse the next section, but we'll have the same number of sandwiches as before. Hence, this kiosk allows us to climb one additional uphill section (the one right after the kiosk).
- If we place a kiosk before a downhill section, we will have two more sandwiches after the next section, which we can use to climb two additional uphill sections which do not have a kiosk before them.
- Placing a kiosk before a downhill section is better than before an uphill section, as the first allows us to climb two uphill sections, while the latter only allows us to climb one.
- Therefore, we should only place a kiosk before an uphill section if absolutely necessary.

Using these observations, we can determine the optimal sections to add a kiosk. We need a kiosk before the first section, as we don't have any sandwiches and we start with an uphill section. Before the second section we check whether we can already reach the end without adding any further kiosks, but this is not the case. Therefore, we add a kiosk here as well, resulting in two sandwiches after the second section. The third section is uphill again, but we have sandwiches, so we don't add a kiosk. Before the fourth section we again check whether we can reach the end without any additional kiosks, but this is still not the case. Therefore, we add a kiosk, ending up with three sandwiches after the fourth section. The next few sections are uphill, but we have just enough sandwiches for all of them, so we don't need to add any other kiosks. Once we reach the summit, there are two downhill kiosks, which then is enough to reach the end without any additional kiosks. Hence, we need only three additional kiosks.



Solaria

Task Preparation	Mark Neumann
Translation (de)	Benjamin Schmid
Translation (fr)	Cheng Zhong
Solution Author	Benjamin Schmid

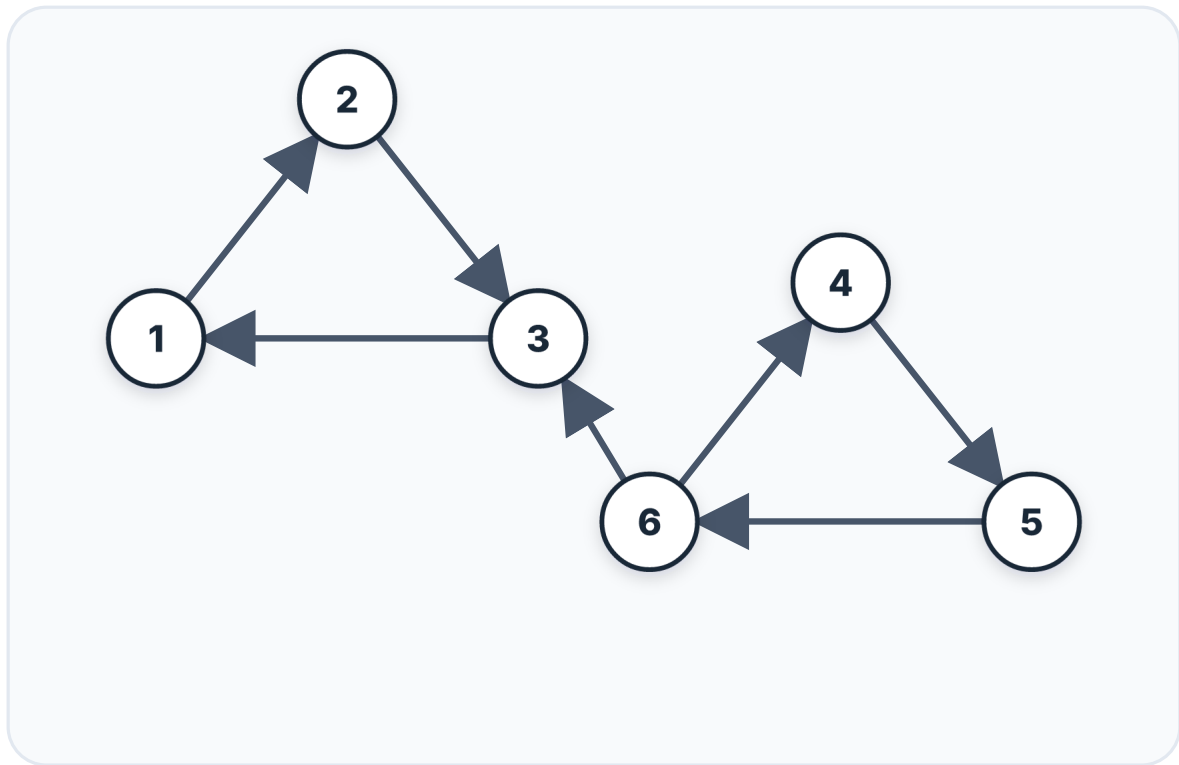


Part 1: The mountain city of Solaria relies on a network of narrow one-way bridges to connect its districts. But the city planners made a terrible oversight: because the bridges only run in one direction, drivers are getting trapped! There are parts of the city where, once you arrive, there is no route that leads back home. The mayor wants to fix the network immediately, but the city budget can only afford a few new bridges.

The map of the bridge network in the image indicates in which direction each bridge may be crossed. What is the minimum number of new one-way bridges the city needs to build such that a driver can start at any district and successfully travel to every other district?

Note: It is allowed to have two bridges run between the same two districts in opposite directions.

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

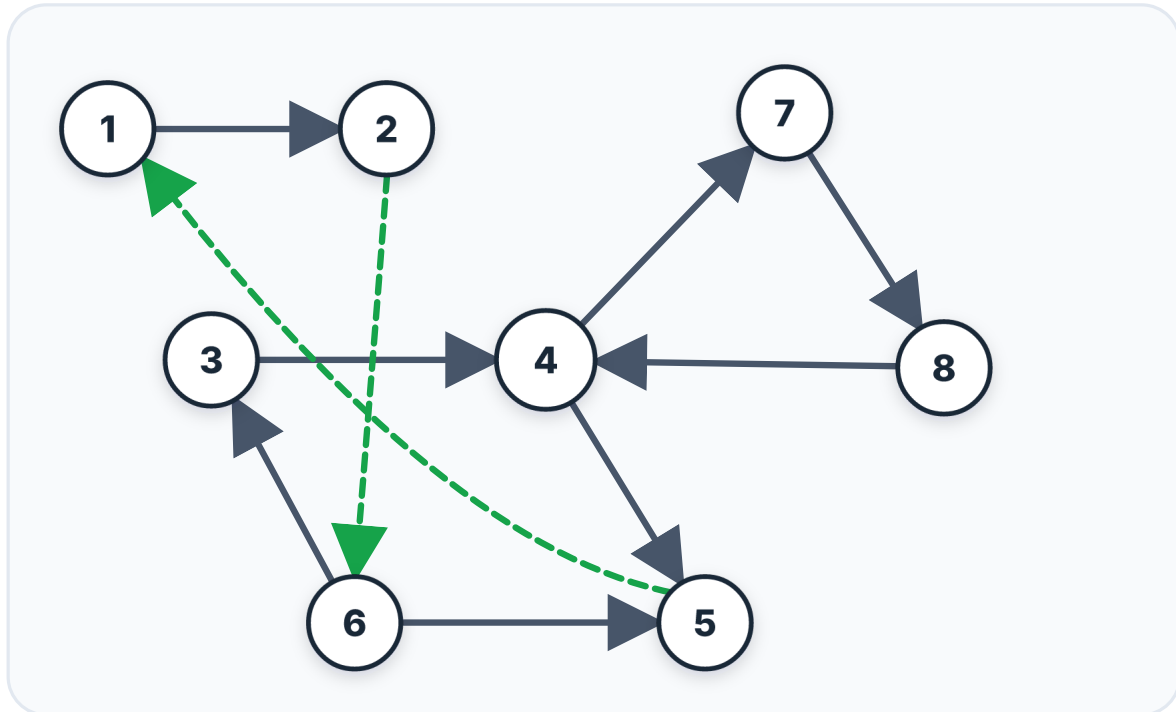


Part 2: Consider the new map. There are several ways to add exactly one new one-way bridge to make it such that a driver can start at any district and successfully travel to every other district. In how many ways can this be done?

- ☐ 3 ☐ 5 ☐ 6 ☐ 8 ☐ 9

Subtask 1: New Bridges

We can achieve this by adding just two edges: $5 \rightarrow 1$ and $2 \rightarrow 6$.



The representation of the bridge network using nodes (= districts) and edges (= bridges) is called a graph. In this case, the edges can only be used in one direction, so we call this a directed graph, which has directed edges.

For this task the concept of a “strongly connected component”, short SCC, is helpful. An SCC is a set of nodes such that we can reach each node in it from each other node in the set. Initially, there is an SCC with the nodes 4, 7, and 8. The goal of the task is to create one big SCC with all the nodes, while adding the fewest number of new edges.

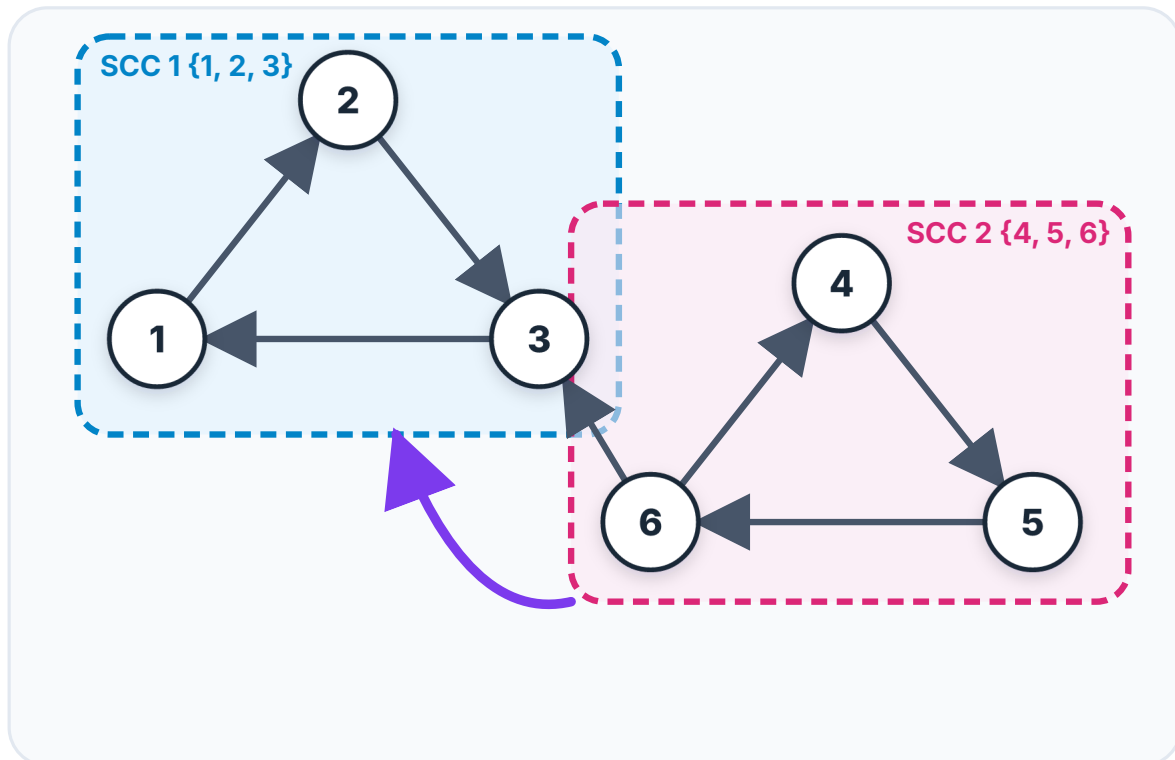
We need to add an outgoing edge to each node which does not yet have an outgoing edge, and we need to add an incoming edge to each node which does not yet have an incoming edge. The nodes without an outgoing edge are initially the nodes 2 and 5. The nodes without an incoming edge are initially the nodes 1 and 6. Pairing them up such that the little “island” of 1 and 2 is connected to the rest, we can create one big SCC.

Note that this does not hold in general. There are examples where all nodes have incoming and outgoing edges but the graph is still not one big SCC. The graph in part 2 is such a counter example.

Subtask 2: One New Bridge

There are 9 possible ways we can add one edge to fulfil the condition.

As in the first part, we want to create one big SCC. Currently, the graph has two small SCCs: one with the nodes 1, 2, and 3, and one with the nodes 4, 5, and 6. As we can reach every other node within an SCC, we can almost consider an SCC a node and replace all the nodes in the SCC by a single node. The graph would then become very simple, with just two nodes, one for the first SCC and one for the second SCC, with an edge from the second SCC to the first SCC.



From a node in the second SCC, we can already reach every other node in the graph. However, it is not possible to get from a node in the first SCC to a node in the second SCC. This is also clearly visible in our simplified graph with just two nodes.

Therefore, we need to add an edge from the first SCC to the second SCC. As we can reach each other node within an SCC, it does not matter which node we pick within an SCC. Hence, we can add an edge from any node in the first SCC to any node in the second SCC. There are three nodes to pick from in the first SCC, and three nodes in the second SCC. Thence, we have $3 \cdot 3 = 9$ possible edges we can add, which combine the graph into one big SCC.



Atlantis

Task Preparation

Mark Neumann

Translation (de)

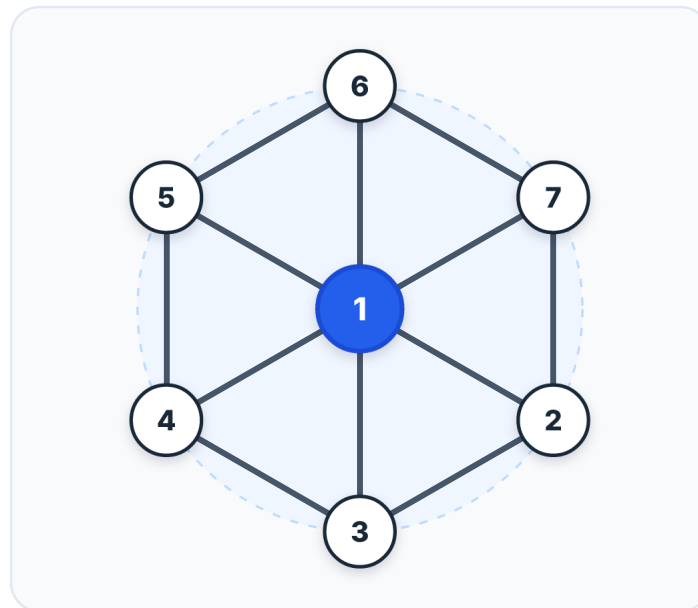
Benjamin Schmid

Translation (fr)

Cheng Zhong

Solution Author

Benjamin Schmid



Part 1: The mythical country of Atlantis has a peculiar train network connecting its n cities. In particular, it consists of a central stop, which is connected by a direct rail line to every other city. The remaining $n - 1$ cities form a ring of rail lines around the central city. An example for $n = 7$ is displayed. Formally, there is a direct rail line between city 1 and every other city. There is also a direct rail line from city x to $x + 1$ for all $2 \leq x \leq n - 1$ and a direct rail line from city 2 to n . Each rail line may be traversed in either direction.

Mouse Stofl would like to tour Atlantis by starting at some station and traveling along the rail network. However, to keep things interesting, he is willing to take each rail line at most once. What is the highest number of rail lines that mouse Stofl can travel across during his tour of Atlantis in a consecutive order without using any rail line twice, for $n = 7$?

An example of a valid tour that Stofl can take is $1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 5$. This path consists of 4 rail lines.

- ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐ 11

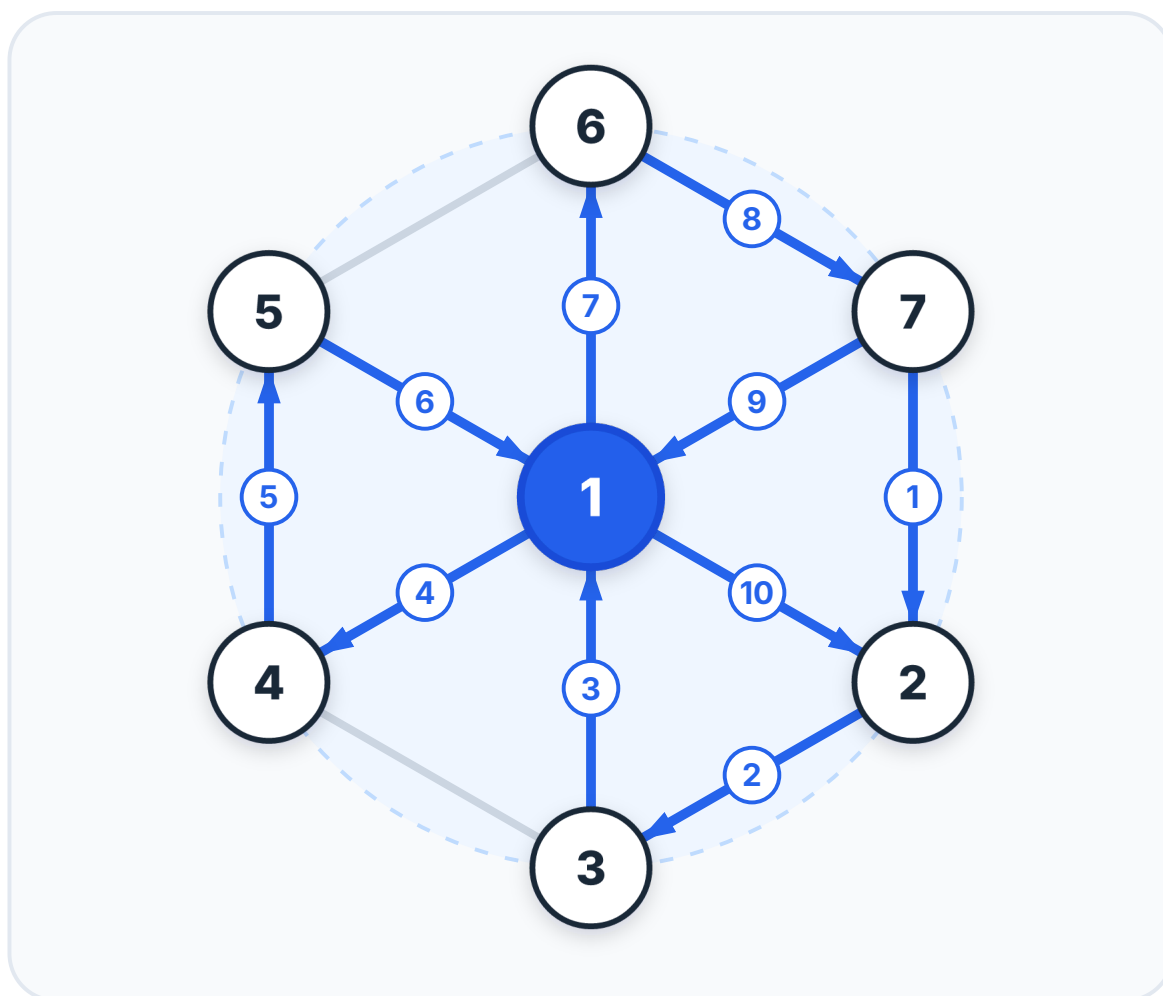
Part 2: Now assume Atlantis is much bigger and $n = 2027$. What is the longest tour that Stofl can take now?

.....



Subtask 1: Small Country

The longest tour we can find has length 10. There are multiple such tours, one example is $7 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 6 \rightarrow 7 \rightarrow 1 \rightarrow 2$.



The representation of the rail network using nodes (= cities) and edges (= rail lines) is called a graph.

There are several different ways we can “walk around” in a graph. The most basic one is called a walk, where we just “walk” from one node to one of its neighbors connected by an edge. We can use each node and each edge multiple times. If we restrict the walk to only visit a node at most once, we call this a path. In our case, the nodes are not restricted, but we can only use each edge at most once. This is called a trail.

The task is to find the longest possible trail in a graph. In general, there is no efficient algorithm to find the longest trail, but we can use the special form of the graph to come up with a solution. In this case, as the graph is small, we can just manually try out different trails or patterns for how we form the trail.



Subtask 2: Large Country

The longest tour has length 3040.

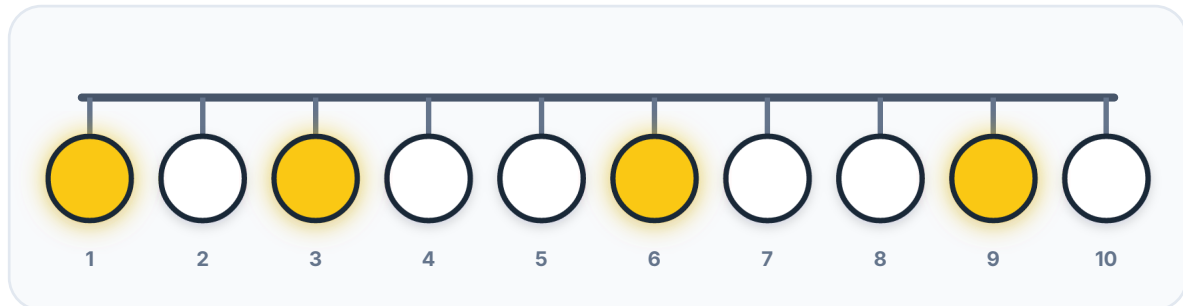
As the graph is too large to find such a tour by hand, we have to figure out a pattern. Looking at the solution from part 1, we can see that the tour is mostly from the center to an outer node, to a neighbor, and then back to the center. There is some change to the pattern at the start and end of the tour. Without the special case at the start and end, our tour would be just the triangles formed by the pattern.

How many such triangles can we fit without any edges overlapping? Each triangle needs one outer edge and we can use every second outer edge for a triangle. A triangle adds 3 to our tour length. With n nodes, we have $n - 1$ outer edges. Therefore, the normal pattern produces a tour of length $\frac{n-1}{2} \cdot 3$.

With the special handling of the start and end we can fit one additional edge into our tour, so our final length is $\frac{n-1}{2} \cdot 3 + 1$. For $n = 2027$ we have therefore a tour of length $\frac{2027-1}{2} \cdot 3 + 1 = 1013 \cdot 3 + 1 = 3040$.

LEDs

Task Preparation	Mark Neumann
Translation (de)	Benjamin Schmid
Translation (fr)	Cheng Zhong
Solution Author	Benjamin Schmid



Part 1: Stofl is a stage manager setting up a strip of 10 LED spotlights for a concert. Currently, some lights are shining yellow (turned ON), and some are white (turned OFF).

He wants to turn all 10 spotlights ON, but there is a catch: the control panel is wired strangely. He can only push buttons that control two adjacent lights at the exact same time. Whenever he pushes a button for two adjacent lights, both of their states flip. (If a light was ON, it turns OFF; if it was OFF, it turns ON).

You are given the starting sequence of 10 lights as shown in the image.

What is the minimum number of button presses Stofl needs to make so that all 10 lights are turned ON?

☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9

Part 2: Stofl realizes that because of the weird way the control panel is wired, there might be some starting patterns where it is impossible to turn all 10 lights ON, no matter how many buttons he presses, but he is not sure.

Each of the 10 lights can initially be either ON or OFF, meaning there are many different possible starting patterns.

Out of all the possible starting patterns for these 10 lights, how many of them can successfully be turned completely ON using Stofl's adjacent-light buttons, after some, possibly zero, operations?

.....



Subtask 1: Turn the Lights ON

We need to press at least 6 buttons.

We can make a few observations:

- Flipping OFF, ON results in ON, OFF, and vice versa.
- Pressing a button twice does not make sense, as the second button press reverses the first press. This is also true if we pressed some other buttons in-between.
- The final button press needs to be on OFF, OFF, otherwise we will always turn OFF a light and we are not done.

To figure out the minimum number of button presses, we can now just go through left to right. We need to flip light 2, for which we have two options: flip 1 and 2, or flip 2 and 3. Flipping 1 and 2 does not work, as then light 1 is OFF but there is no button we can press to turn it ON again. Hence, we need to flip 2 and 3. Next, we need to turn on light 3 again. With the same argument, there is only one option remaining: flipping 3 and 4.

Going through all the lights, following this procedure, we can determine exactly which buttons we need to press: 2 and 3, 3 and 4, 5 and 6, 6 and 7, 8 and 9, 9 and 10.

Subtask 2: Impossible Patterns

There are 512 patterns where we can successfully turn ON all lights.

In total, there are $2^{10} = 1024$ possible patterns. Pressing a button always flips exactly two lights. Therefore, the total number of lights turned ON either stays the same, increases by two, or decreases by two. Notably, the parity of the total number of lights does not change. Hence, if we start with an odd number of lights turned ON initially, we will never be able to turn ON all the lights. On the other hand, if we start with an even number of lights turned ON, it is always possible to turn ON all of them, using the procedure from part 1.

Using this observation, we need to count the number of patterns with an even number of lights turned ON. This is exactly half of all the patterns: 512.

Cogros

Task Preparation	Mark Neumann
Translation (de)	Benjamin Schmid
Translation (fr)	Cheng Zhong
Solution Author	Benjamin Schmid

Part 1: In the new Cogros shopping mall, there are 256 automatic doors. They are all currently turned OFF, and each door has its own dedicated switch.

You received an anonymous tip that exactly one of these 256 switches is completely broken, but you don't know which one. Your job is to find it!

Every minute, you can choose to do exactly one of the following two actions:

- Flip a switch: If it's the broken switch, nothing happens. If it's a working switch, it changes the door's state (from OFF to ON, or from ON to OFF). However, flipping the switch doesn't let you see the door, so you won't immediately know its new state.
- Inspect a door: You walk over to one specific door to look at it and see whether it is currently ON or OFF.

What is the minimum number of minutes you need to guarantee that you find the broken switch, even in the worst-case scenario (if you are extremely unlucky)?

☐ 384 ☐ 255 ☐ 256 ☐ 510 ☐ 512

Part 2: After thinking about it a bit more, you realize there is a third action you can choose to do. Just like the others, this action also takes exactly one minute:

- Check the power meter: You can measure the total electricity consumption of the entire Cogros mall. This instantly tells you exactly how many total doors are currently turned ON in the entire building.

Using this new action, along with the two original actions, what is the new minimum number of minutes you need to guarantee you find the broken switch?

Note: if you press multiple switches before doing a check, each switch still takes a full minute.

.....



Subtask 1: Find the Broken Door

We need at least 510 minutes.

There is no way to check multiple doors or flip multiple switches at the same time. Therefore, checking whether a given switch is broken takes exactly two minutes: one minute to flip the switch and one minute to check whether the door is ON.

A possible strategy is to just check all the switches, taking $2 \cdot 256 = 512$ minutes. There is one small optimization we can do: once we checked all the switches except for one, we do not need to check the last switch because we know that exactly one is broken. This saves us two minutes, leaving 510 minutes. This is the best solution.

Subtask 2: Electricity Consumption

We need at least 263 minutes.

The new operation enables us to check multiple doors at a time. This allows for a strategy called binary search. First, we switch exactly half of the switches. Then, we read the power meter. If the number of doors turned ON matches the number of switches we flipped, we know that all of the switches worked correctly. Thus, the broken switch needs to be in the other half. On the other hand, if the number of doors turned ON does not match, we know that one of the switches we flipped was broken. We can therefore eliminate the other half.

After one of these steps, we were able to eliminate one half of the switches. We can then recursively apply the same strategy on the half where we determined the broken switch was located. Once we are left with only two possible switches, we check one of them, and then have identified the broken switch.

We can then calculate the required number of minutes using the following table.

Remaining Switches	Switches to Check	Required Minutes
256	128	129
128	64	65
64	32	33
32	16	17
16	8	9
8	4	5
4	2	3
2	1	2

Summing up the last column, we get $129 + 65 + 33 + 17 + 9 + 5 + 3 + 2 = 263$ minutes.