

# Commentaries on Problems

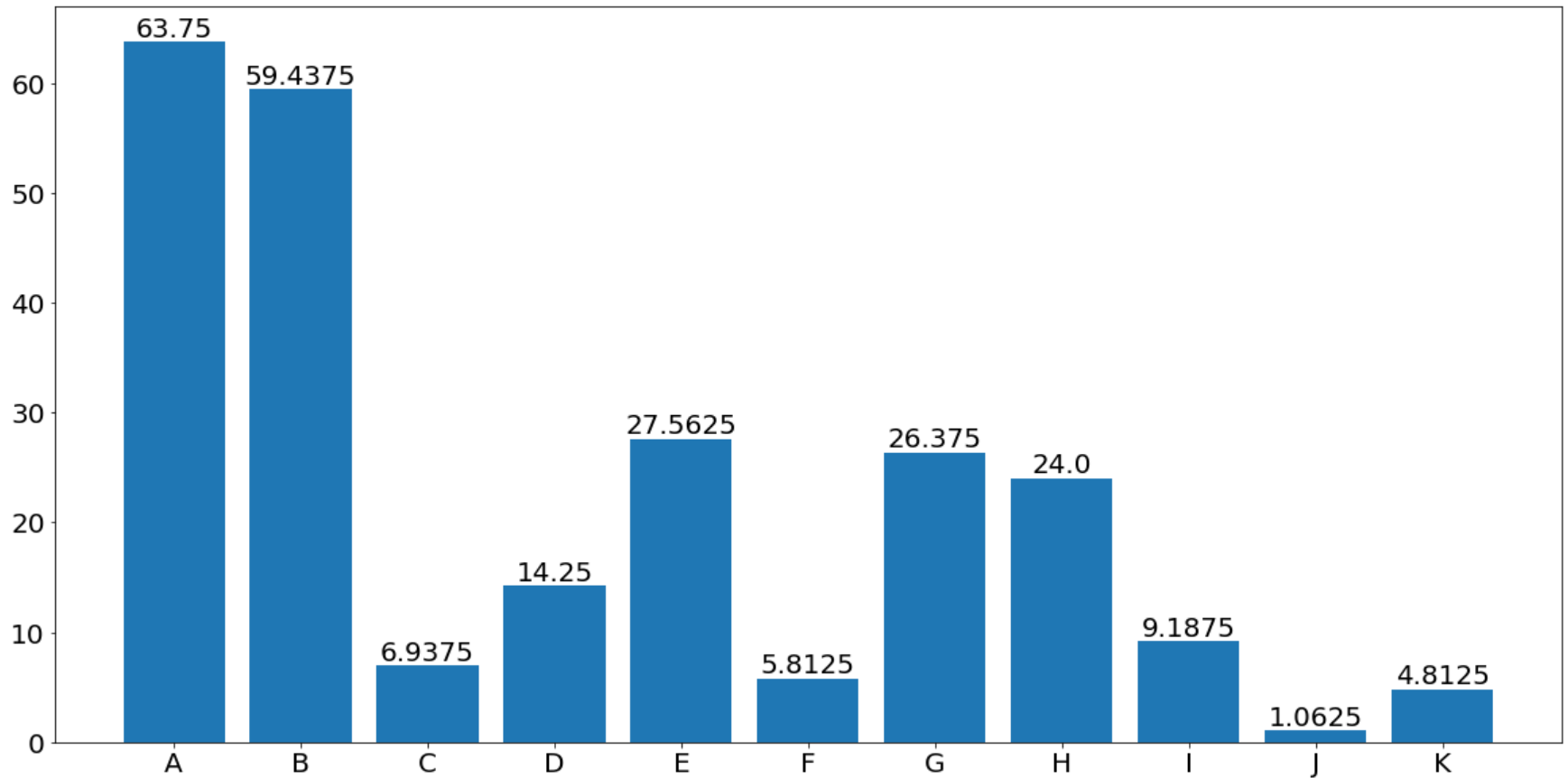
---

JUDGE TEAM

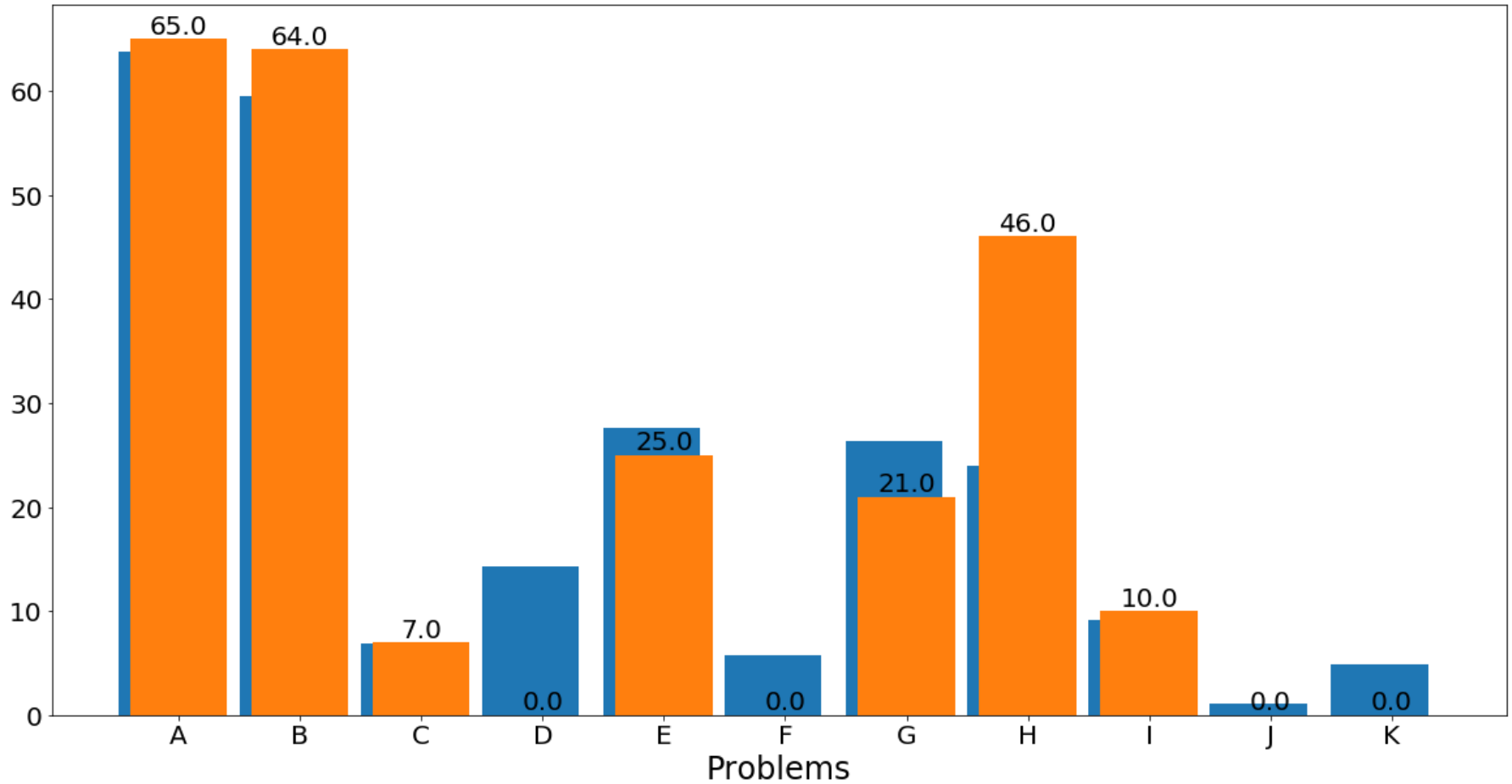
ICPC 2019 ASIA YOKOHAMA REGIONAL



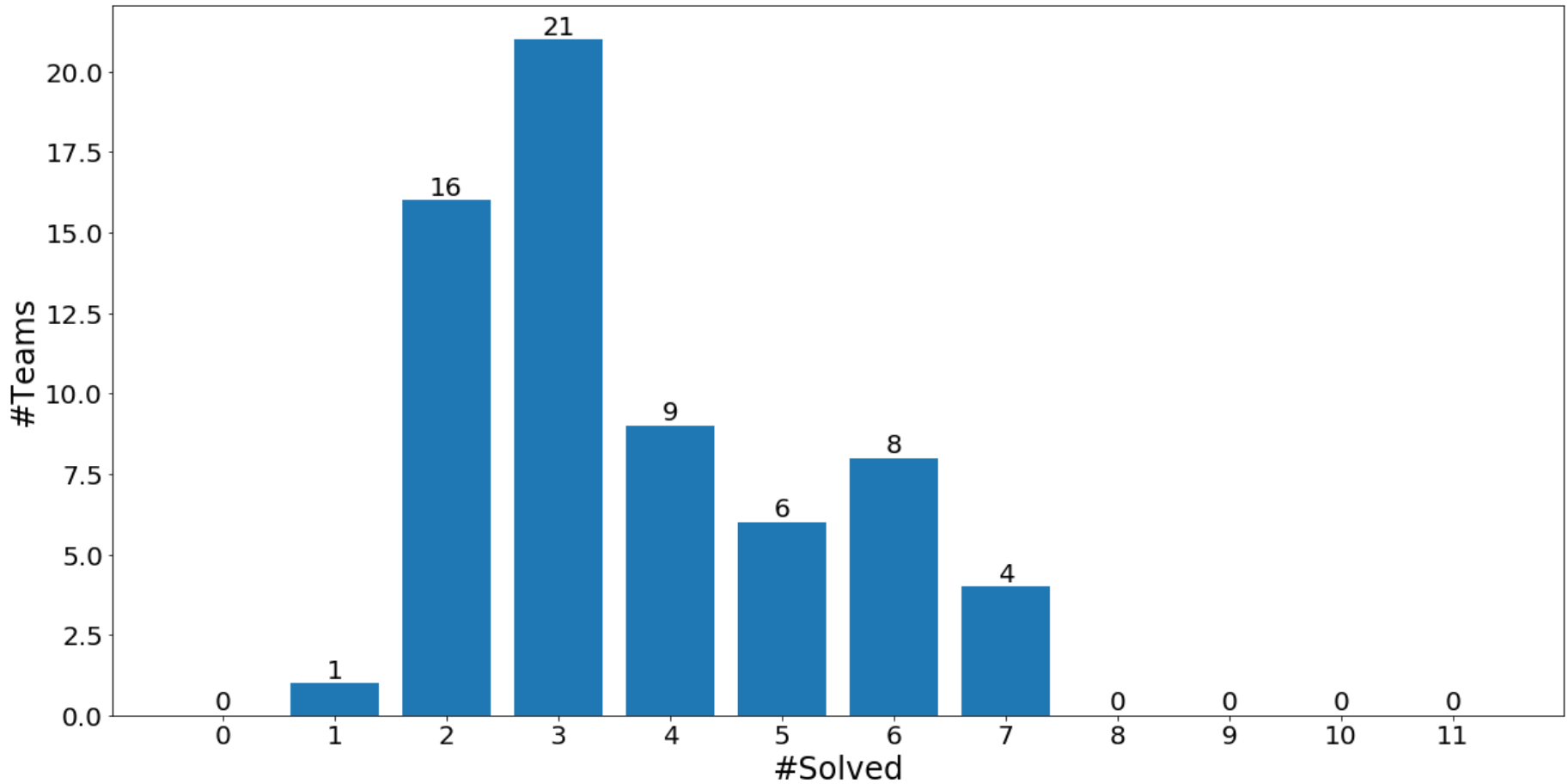
# Predicted # of Correct Answers



# Estimated vs. Solved @Freeze



# #Solved vs #Teams @Freeze



# Commentaries

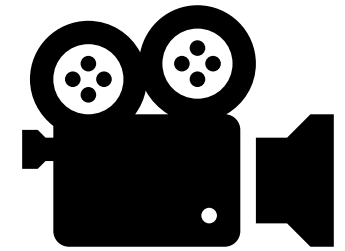
Decreasing order of #solved teams  
(tie-broken by judge's estimation)

A --> B --> H --> E --> G --> I --> C --> D --> F --> K --> J

# A:Fast Forwarding

---

# Story



3x

triples the video playing speed.

1/3x

reduces the speed to one-third.


Button state is checked each 1 second.

***How quick can you fast-forward the video to the start of your favorite scene?***

# Key Observations

No need to consider pressing  $[1/3x]$  before any  $[3x]$ .

$$1 + 3 + 3 + 1 + 3 + 1 = 1 + 3 + 3 + 3 + 1 + 1$$



No need to consider pressing  $[3x]$  later than sooner.

$$1 + 3 + 3 + 3 + 9 + 3 = 1 + 3 + 9 + 3 + 3 + 3$$





# Solution

No need to search all button pressing patterns!

Greedy press [3x] for a first few seconds  
(until it overruns the target scene.)



$$1 + 3 + 9 + 27 + (\dots + 27) + (\dots + 9) + (\dots + 3) + (\dots + 1)$$

Then the timing of [1/3x] easily follows from  
the remainder of the divisions by 3, 9, 27, ...

# B: Estimating the Flood Risk

---

# Story

Mr. Boat wants to know the average altitude of his land in order to estimate the flood risk.

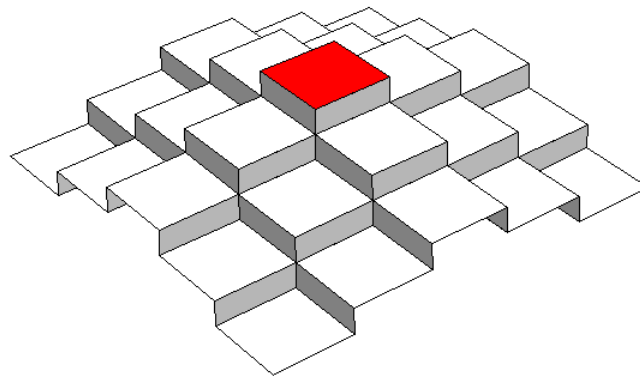
As no steep slopes are in his estate, he thought that it would be enough to measure the altitudes of only limited number of sites and approximate the altitudes of the rest based on them.

Your job is to write a program that computes total of altitudes of all the mesh-divided areas.

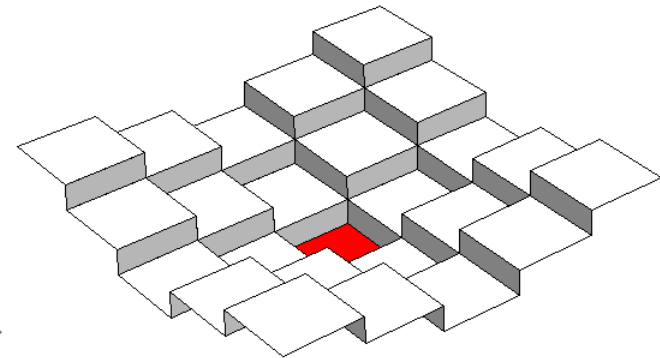
If there are multiple possibilities, among which one that gives the lowest average altitude should be considered.

# Solution

The range of the altitude of each area  $(x, y)$  is the intersection of  $[z_i - d, z_i + d]$  for all measured area  $(x_i, y_i)$  of altitude  $z_i$ , where  $d$  is the Manhattan distance between  $(x, y)$  and  $(x_i, y_i)$ .



lower bound



upper bound

If there exist one or more area in which the range of the altitude is empty, answer "**No**".

# H:Parentheses Editor

---

# Problem

Given text editing commands

- Append (
- Append )
- Delete last character

Count the number of balanced substrings

Examples

(())                    answer = 2

()()                    answer = 3

((()))                  answer = 4

# Solution

- Count the difference of the answer after commands
- Separate with unmatched left parens
- Divide into minimal balanced strings. This is a stack

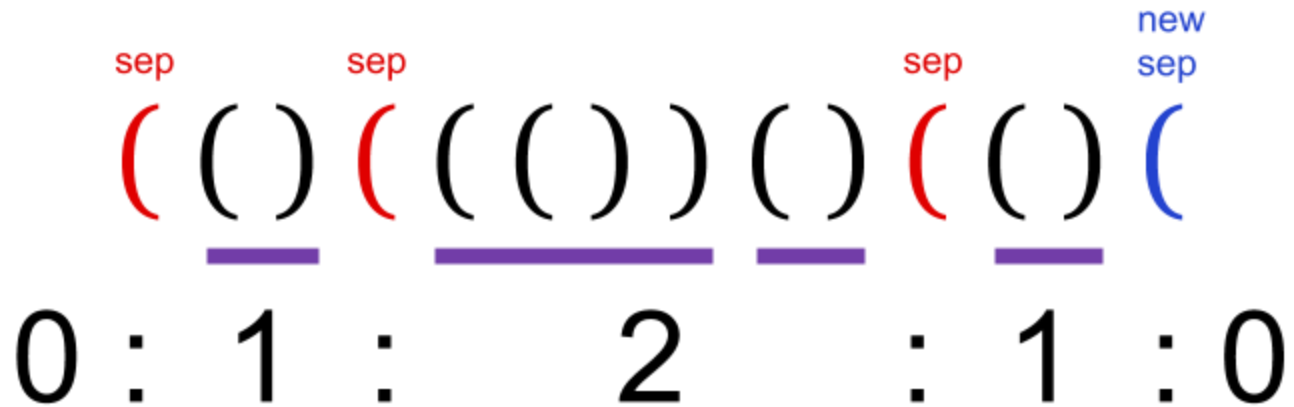
sep            sep            sep

( ( ) ( ( ( ) ) ( ) ( ( ) )

0 : 1 :            2 : 1

# Append left paren (

stack.Push(0)



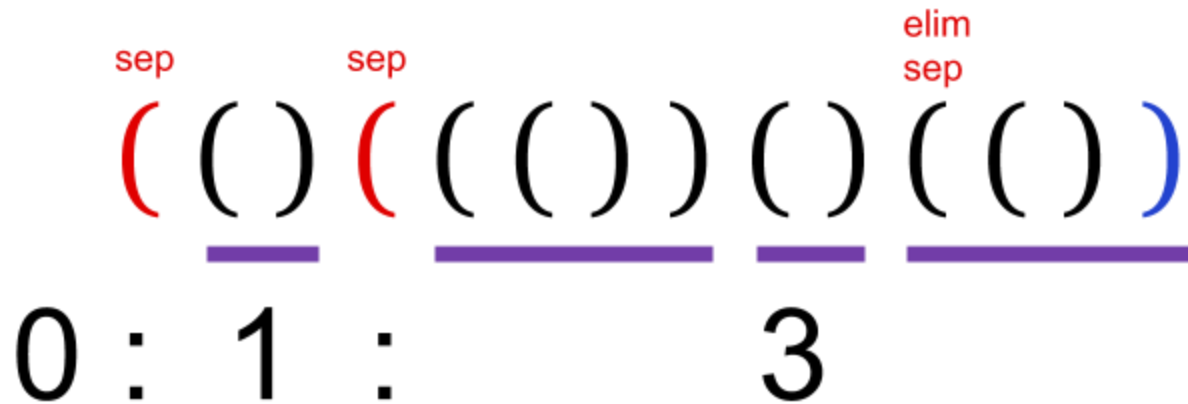


# Append right paren )

stack.Pop()

stack.Top() += 1

answer += stack.Top()



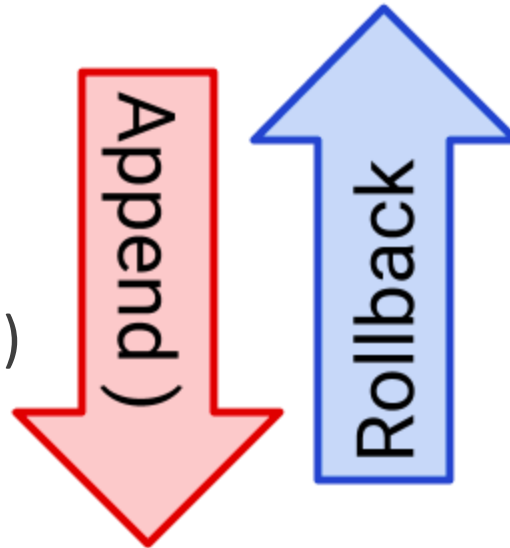
# Append right paren ), no sep

When there is no separator, the size of stack is one.  
`stack.Top() = 0`

# Delete Last Character

- Append commands run in  $O(1)$
- Rollback also runs in  $O(1)$

```
x = stack.Pop()  
stack.Top() += 1  
answer += stack.Top()
```

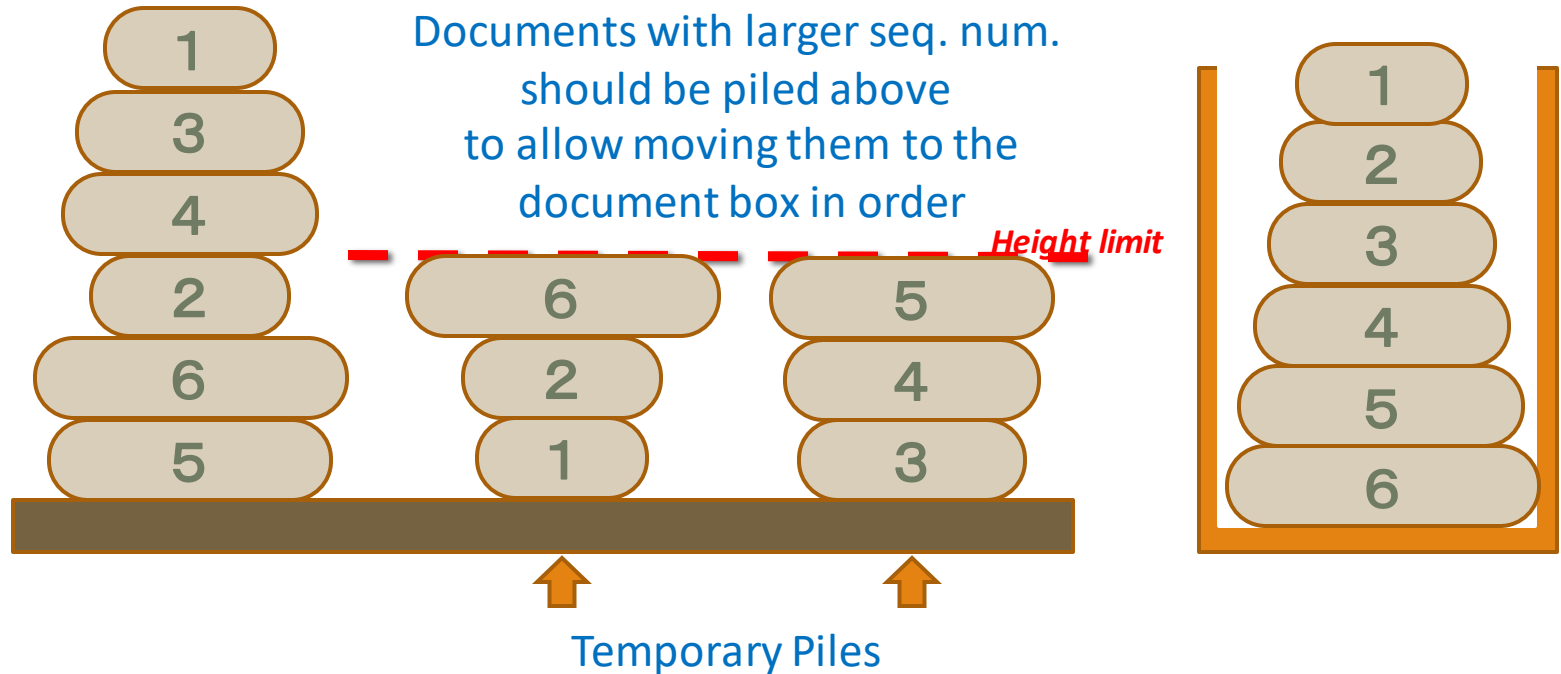


```
stack.Push(x)  
stack.Top() -= 1  
answer -= stack.Top()
```

# Problem E: Reordering the Documents



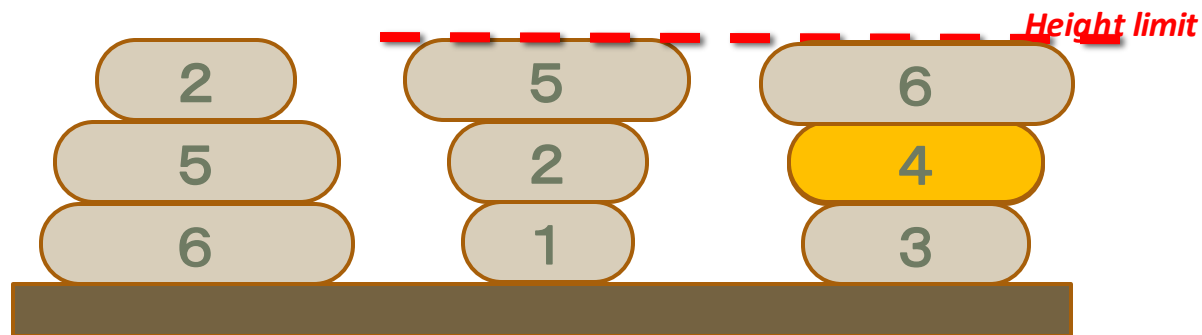
# Reorder documents via two extra piles



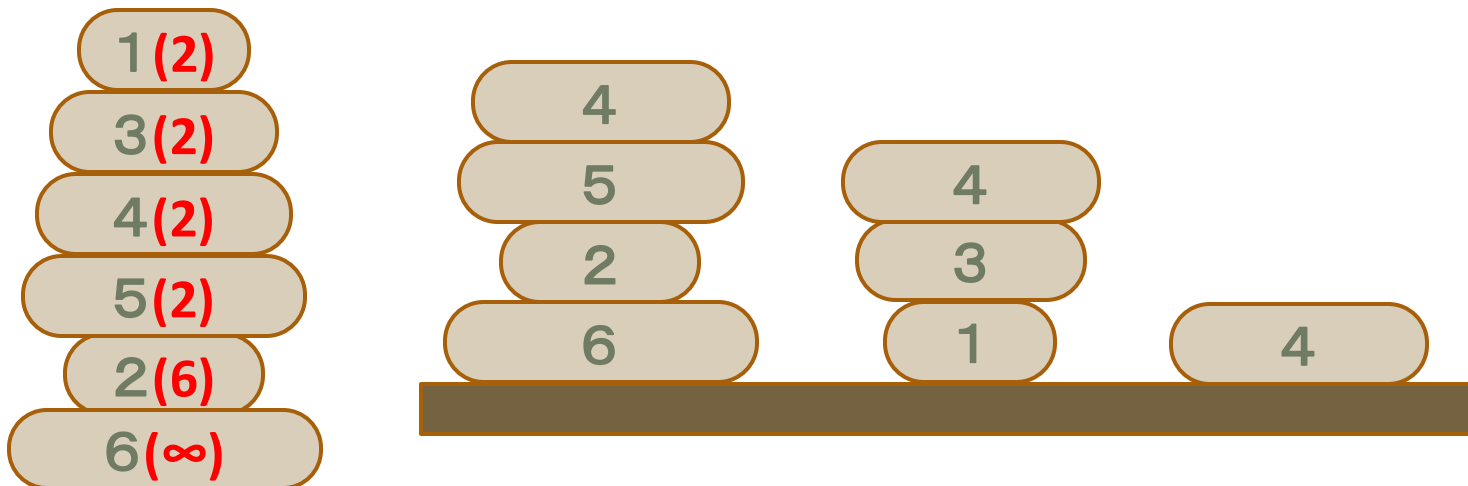
The document with the largest seq# moved so far (M) must be on top for the temporary piles to be properly ordered

- If the next document to move has # less than M, the document should not be placed upon M. No choice here.
- If the next document to move has # larger than M, the document can be moved to either of the temp. piles.

The height limit should be respected in both cases.



- If any remaining documents have # less than both tops of the temporary piles, they can never be handled properly.
- The min. seq. # below each doc can be computed beforehand with  $O(n)$ .
- With this info, moves leading to a problem in future can be avoided beforehand.



# Solution: $O(mn)$

- For docs on the pile, compute the min. doc # below:  $O(n)$
- Build a table  $t$  of the different possible ways to move docs, indexed by the height of the pile with the larger # doc on top. Initiate it as  $t[0] = 1$  and other elements 0:  $O(m)$
- For each doc, from the top, update the table:  $O(mn)$ 
  - If its # is less than the largest so far, no changes are needed
  - If it is larger, it may be placed on either of the temp. piles. Initiate a new table  $n$  with all its items 0  
For  $i = 1..m$ ,
    - $n[i] += t[i]$  ; move to the pile with smaller top
    - $n[i+1] += t[k-i]$  ; move to the pile with larger top,  $k$  is # docs so far
  - Copy  $n$  to  $t$

Pile height constraints should be checked during the operation



# G: Ambiguous Encoding

---

# Problem Description

Find the shortest length of **ambiguous binary sequences** made from the given code set.

Code Set	Characters to encode	Encoding of String to Binary Seq.
1	... 'A'	"AAB" -> 1 1 01
01	... 'B'	↑ ?
100	... 'C'	1101
110	... 'D'	↓ ?
		"DA" -> 110 1

Ambiguous!  
(You can't decode this correctly)

# Naïve Way to Find Amb. Seq.

Code Set

- 001
- 01
- 11010
- 0111

01

0111

- Start from a pair of codes s.t. one is a **prefix** of the other.

# Naïve Way to Find Amb. Seq.

Code Set

- 001
- 01
- 11010
- 0111

0111010

0111

- Start from a pair of codes s.t. one is a prefix of the other.
- Put a code **whose prefix is the remainder**, or

# Naïve Way to Find Amb. Seq.

Code Set

- 001
- 01
- 11010
- 0111

0111010

011101

- Start from a pair of codes s.t. one is a prefix of the other.
- Put a code whose prefix is the remainder, or  
Put a code that is **a prefix of the remainder**.

# Naïve Way to Find Amb. Seq.

Code Set

- 001
- 01
- 11010
- 0111

011101001

011101001

Found an ambiguous seq.

- Start from a pair of codes s.t. one is a prefix of the other.
- Put a code whose prefix is the remainder, or  
Put a code that is a prefix of the remainder.
- Done when the remainder is an empty sequence.

# Observation

01110101100

0110110101100

01110101100

0110110101100

- Given an intermediate state of the naïve process,  
the shortest tail to the even length depends only on the remainder.
- So, we can do DP on remainders
  - You need to remember only the shortest seq. to the remainder

# Solution: Shortest Path Search

- $V =$  Possible remainders  $:: |V| = O(2^{16})$
- $E =$  Transition by code  $:: |E| = O(16 * n + 16 * 2^{16})$ 
  - $(R1, R2) \in E \Leftrightarrow$  there exists a code  $C$  such that
$$C = R1 + R2 \quad (\text{at most } 16 \text{ for each code}), \text{ or}$$
$$R1 = C + R2 \quad (\text{at most } 16 \text{ for each remainder})$$
  - $\text{Length} = \min( |R1|, |C| )$
- $\text{Start} =$  Remainders of all possible pairs of the given codes
- $\text{Goal} =$  Empty remainder
- No path to the goal  $\Leftrightarrow$  no ambiguous sequence exists.
- So, you can use Dijkstra algorithm to solve this prob. in time



# I:One-Way Conveyors

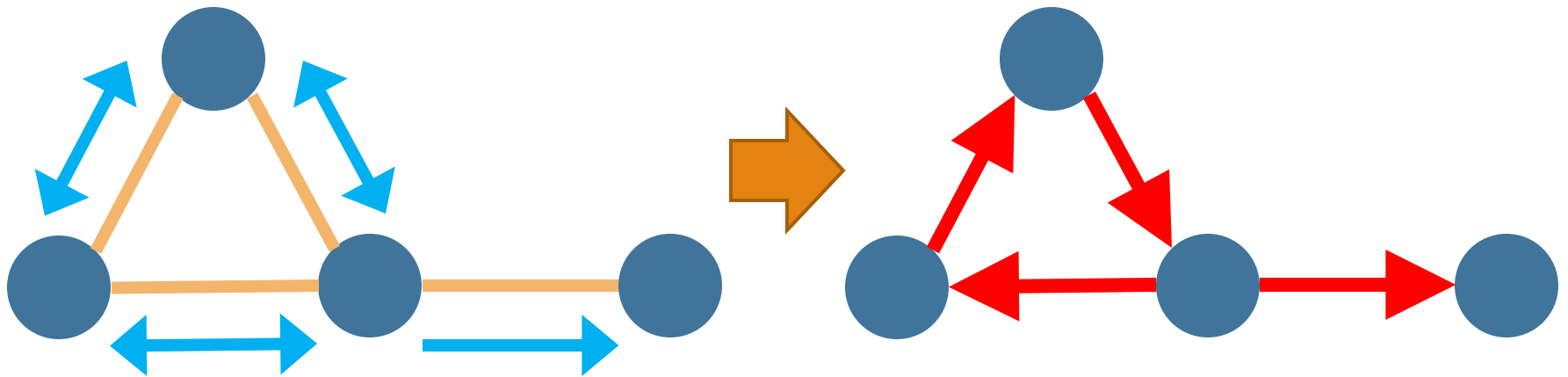
---

# Problem Summary

Given an undirected graph and necessary moves

Make the graph directed while keeping all the necessary moves

$$2 \leq n \leq 10000, 1 \leq m \leq 100000, 1 \leq k \leq 100000$$

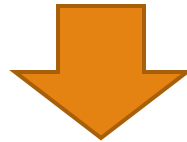


# Observation

If **an edge is always passed** in two moves with different directions, it's definitely impossible to make the graph directed



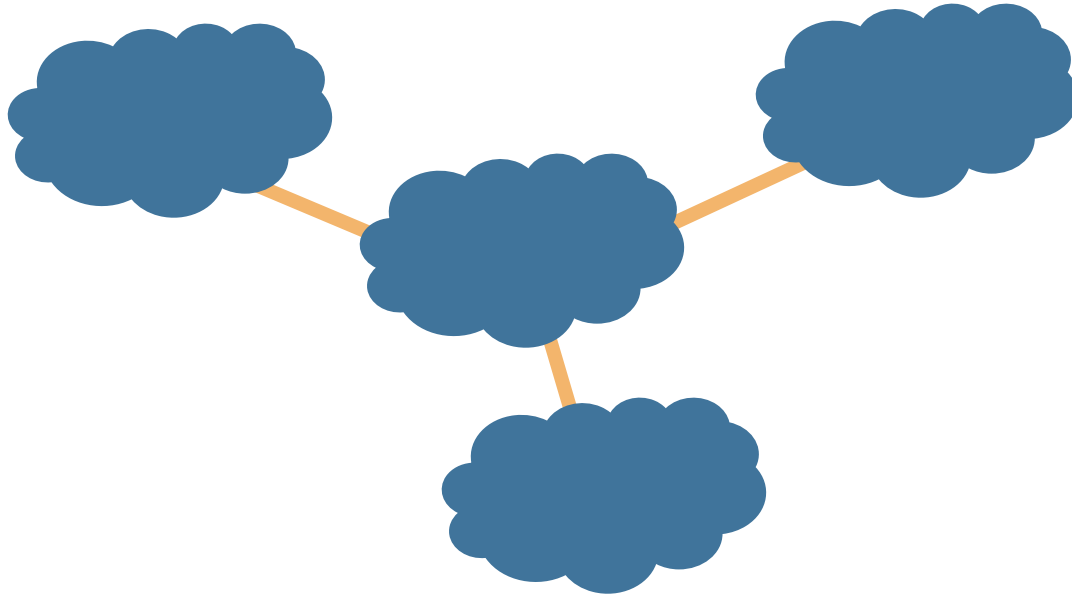
If **an edge is a bridge**, it's passed in any possible paths for the move



Directions of all bridges are determined uniquely (or not relevant)

# Decompose graph by bridges

Decompose the graph by bridges



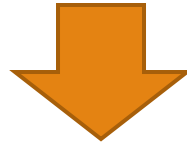
The graph is a **tree**

Every vertex is a **2-edge-connected graph**

Every edge is a **bridge**

# 2-edge connected graph

Is it possible to make a 2-edge connected graph directed while keeping all the moves?



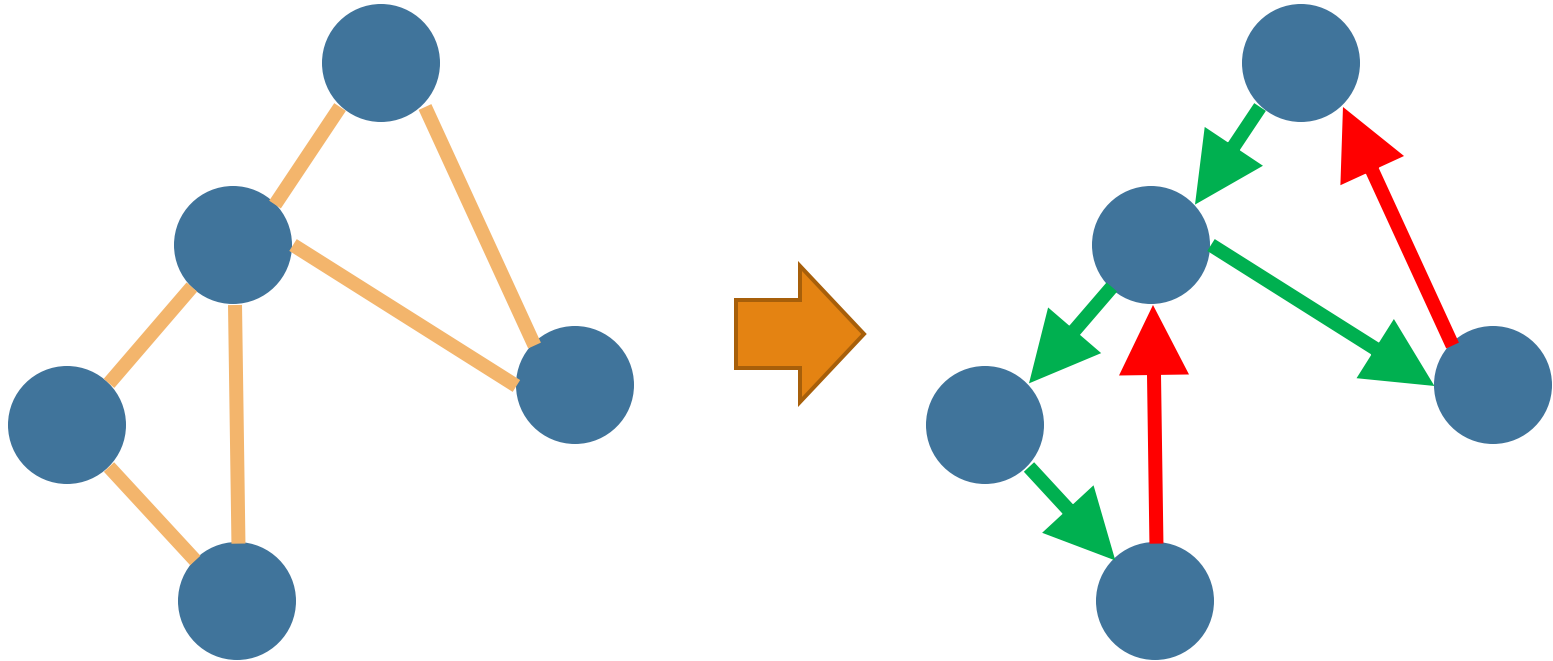
**Always Yes!!**

## **Robbins' theorem**

It's possible to make a 2-edge connected graph directed that has a path from every vertex to every other vertex

# 2-edge connected graph

One easy way to assign directions of edges



Perform a **depth-first search**

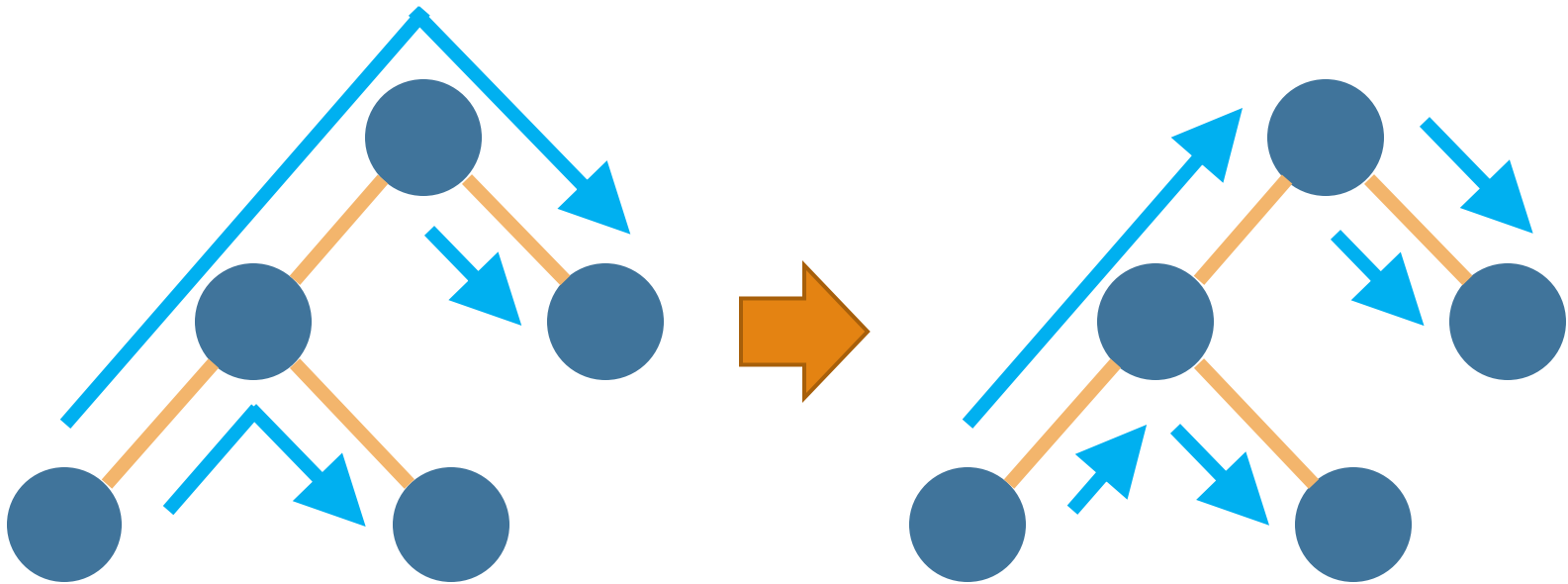
**Forward edges** are directed from the top to the bottom

**Back edges** are directed from the bottom to the top

# Directions of bridges

The graph is a **tree** now

Each move is divided to two moves by its **lowest common ancestor**

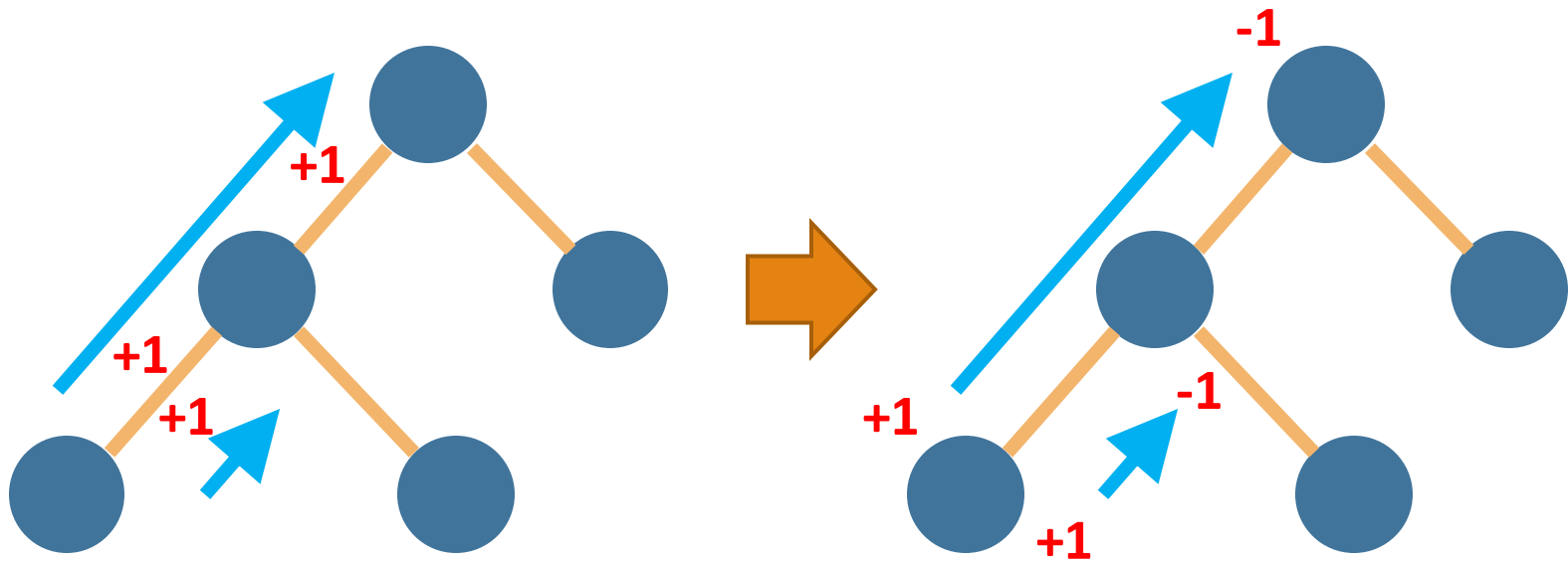


# Directions of bridges

For each upward (downward) move, add +1 to its edges

If an edge has a value, the edge should be upward (downward)

This can be efficiently done by adding +1/-1 to endpoints and traversing the tree with summing the values





# Summary

1. Decompose the graph by bridges
2. Assign directions of 2-edge connected graphs
3. Divide moves on the tree to upward/downward moves by LCA
4. Validate and assign directions of the tree

1 and 2 can be done together by a depth-first search in  $O(n + m)$

3 can be done in  $O((n + k) \log n)$

4 can be done in  $O(n)$

**The total time complexity is  $O(m + (n + k) \log n)$**

# Appendix

It's also interesting to consider how to validate answers efficiently

Given a directed graph and necessary moves

Check if all the moves are actually valid

# C:Wall Painting

---

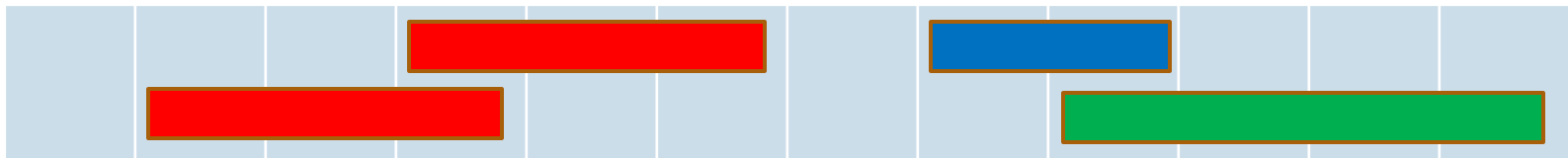
# Problem Description

$i$ -th robot paints  $j$  (in  $[l_i, r_i]$ )-th panel in color  $c_i$ , when activated.

The aesthetic value of each panel is defined as follows:

- Left unpainted: zero score
- Painted in 1 color: positive score ( $+x$ )
- Painted in 2+ colors: negative score ( $-y$ )

Maximize the sum of aesthetic values.



In case of  $x=7$  and  $y=2$ , the sum of values is 61.

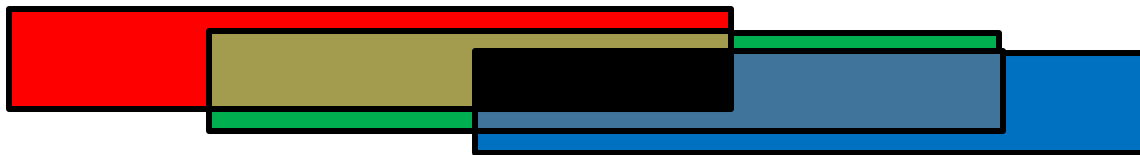
# Observation

The following cases are ignorable. (There is a redundant range)

- There is a range overlapped by another range.



- There is a panel which is painted 3+ times.



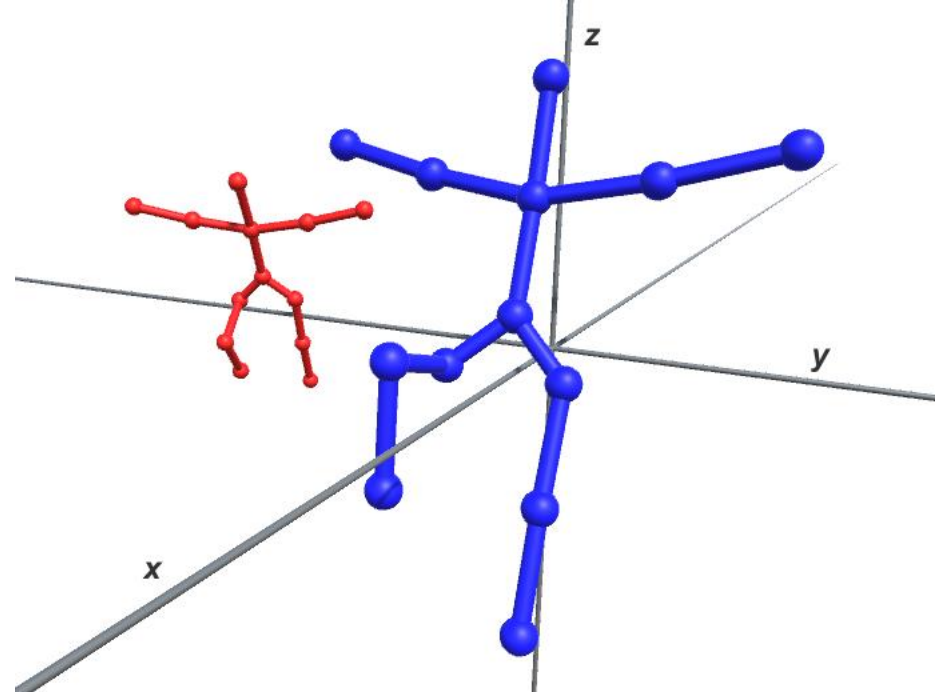
# Solution

First, sort all robots increasing based on  $r_i$ .

Let  $dp[i]$  be the maximum sum of aesthetic values when the right most activated robot is  $i$ -th robot.

$$dp[i] = \max\{\begin{aligned} & \max_{[j \text{ s.t. } j < i \text{ and } r_j < l_i]} dp[j] + (r_i - l_i) * x \\ & \max_{[j \text{ s.t. } j < i, l_j < l_i \leq r_j \text{ and } c_i = c_j]} dp[j] + (r_i - r_j) * x \\ & \max_{[j \text{ s.t. } j < i, l_j < l_i \leq r_j \text{ and } c_i \neq c_j]} dp[j] + (r_i - r_j) * x - (r_j - l_i) * y \end{aligned}\}$$





D: Twin Trees Bros.

---



How to check whether two trees are *twin* or not. →  
Translate

Find a leaf node of a tree.

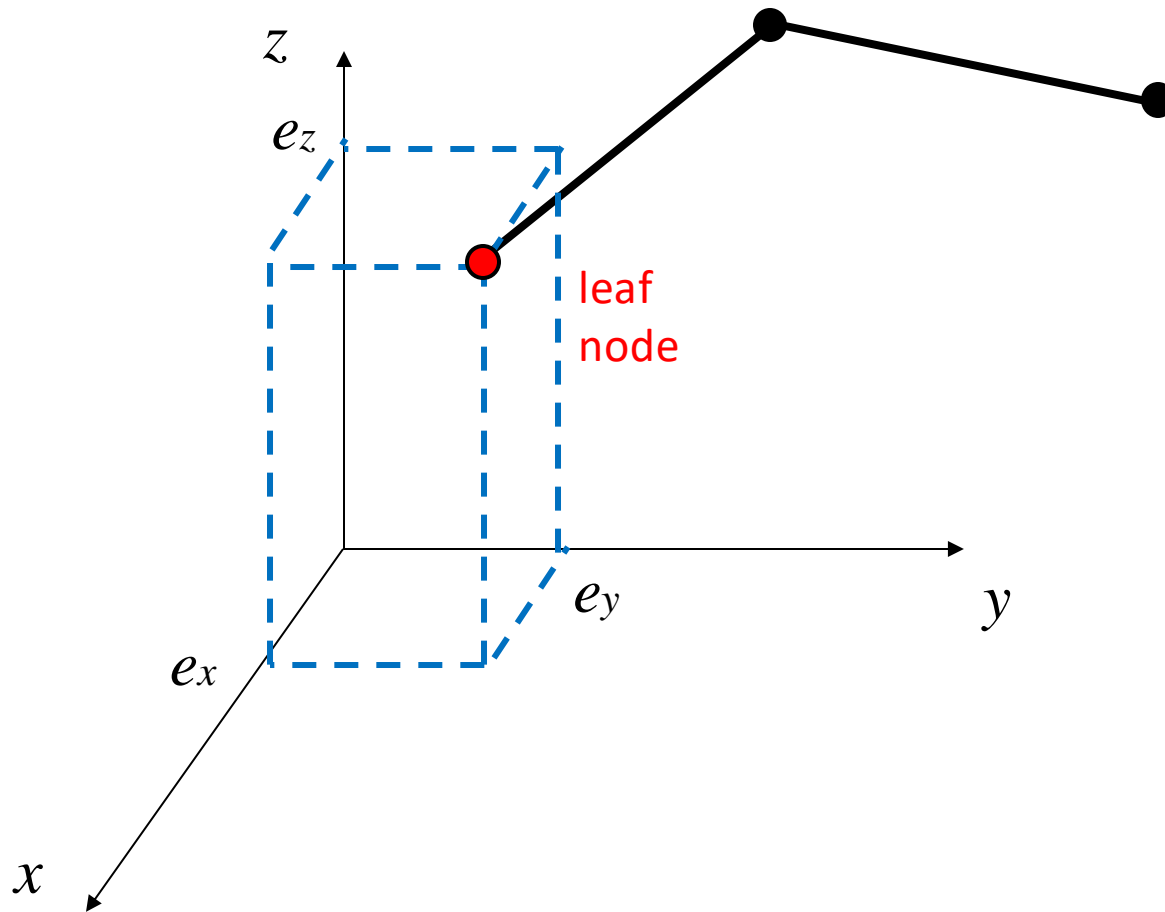
Translate the leaf node to the origin point  $(0,0,0)$ .

Rotate the edge between the leaf node and its parent around  $z$ -axis onto  $yz$ -plane.

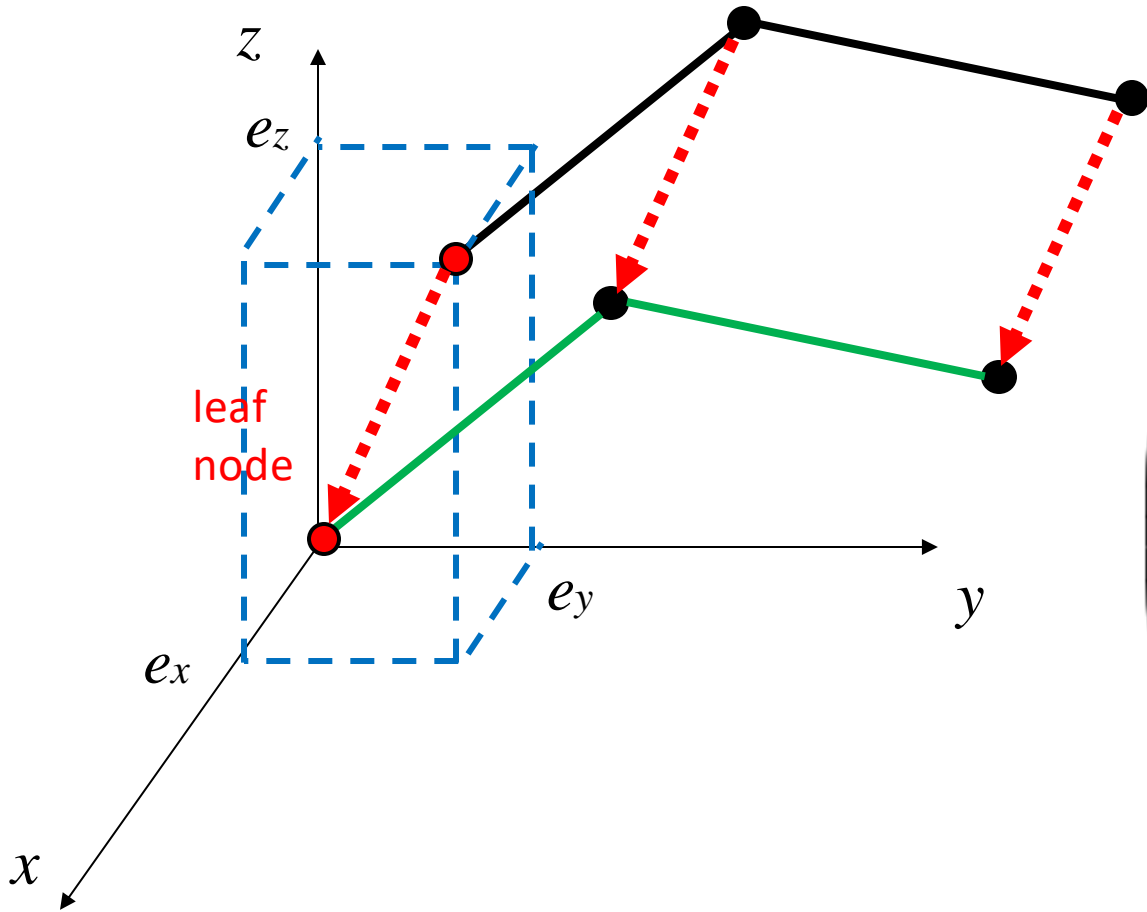
Rotate the parent node around  $x$ -axis onto  $z$ -axis.

(Optional) Rotate an ancestor node around  $z$ -axis onto  $yz$ -plane.

(1) Find a leaf node of the tree

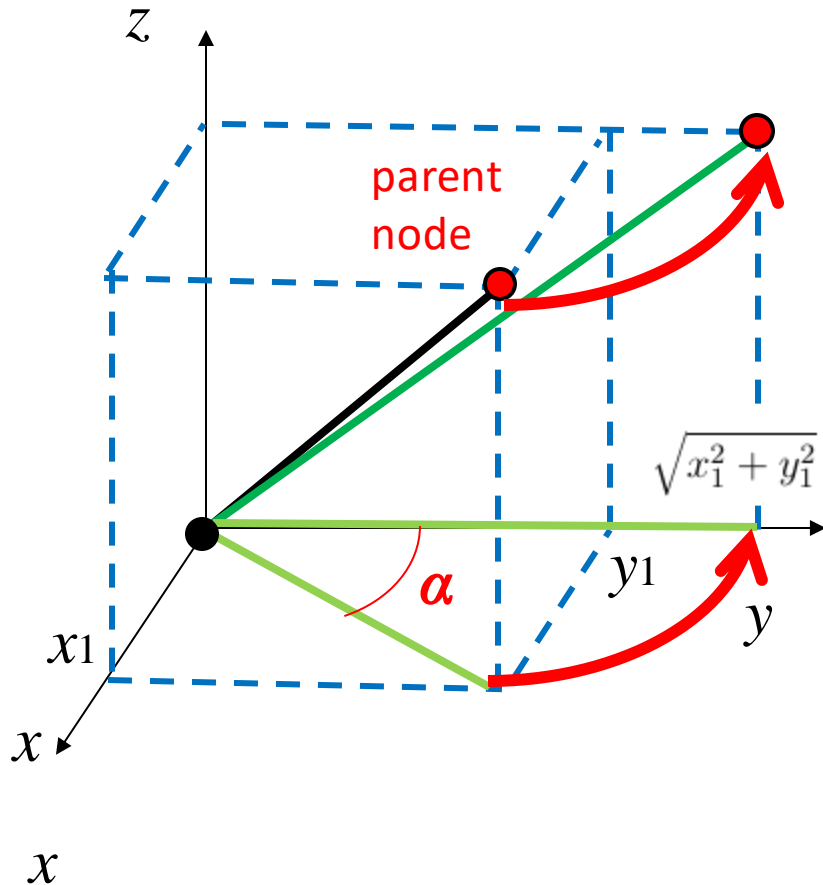


(2) Translate the tree so that the leaf node moves to the origin point.



$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix}$$

(3) Rotate the tree around  $z$ -axis so that the parent node moves onto the  $yz$ -plane.

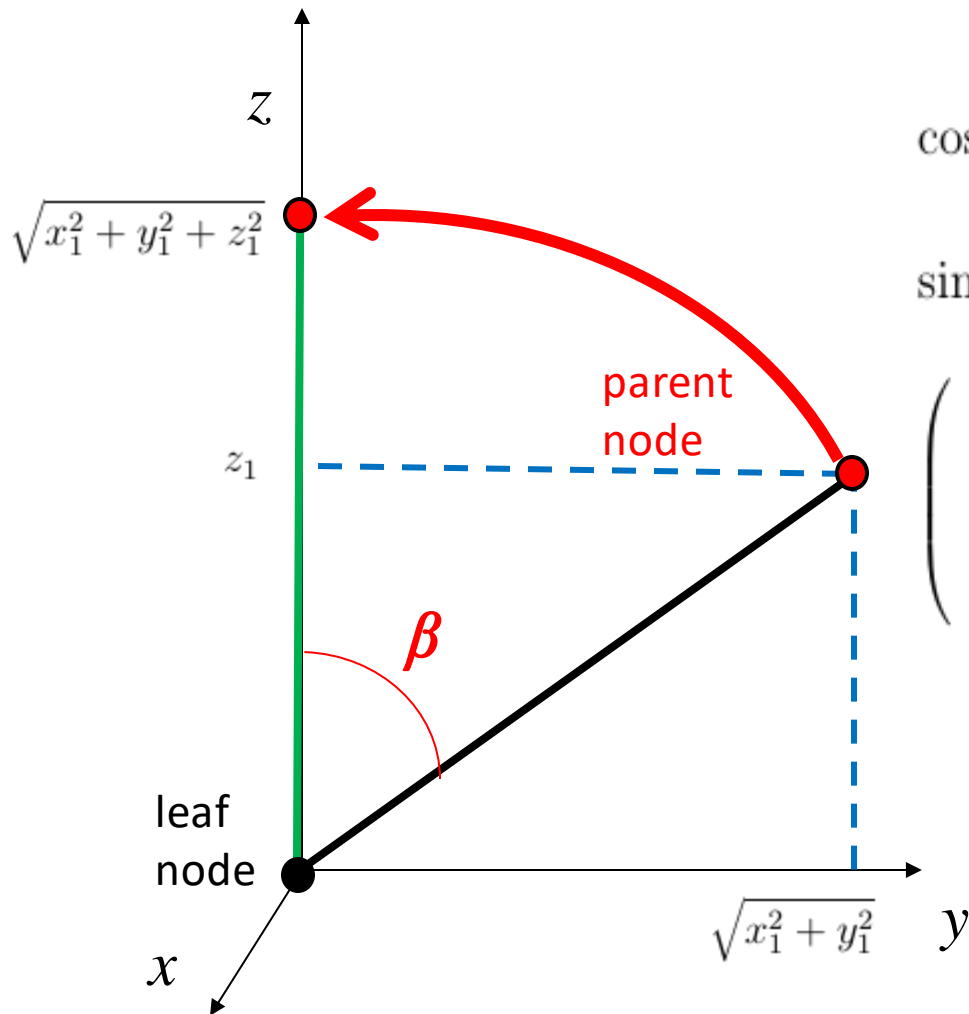


$$\cos \alpha = \frac{y_1}{\sqrt{x_1^2 + y_1^2}}$$

$$\sin \alpha = \frac{x_1}{\sqrt{x_1^2 + y_1^2}}$$

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix}$$

(4) Rotate the tree around  $x$ -axis so that the parent node moves onto the  $z$ -axis.

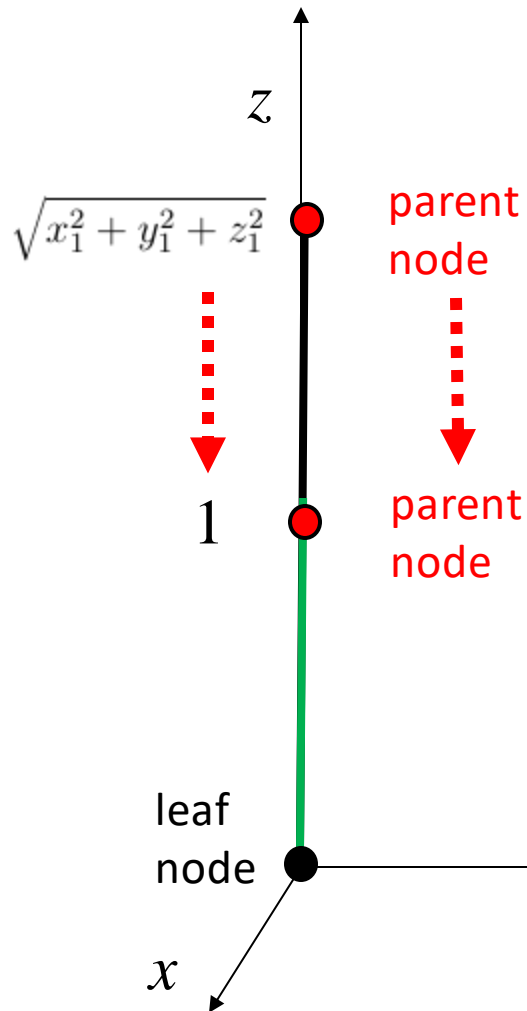


$$\cos \beta = \frac{z_1}{\sqrt{x_1^2 + y_1^2 + z_1^2}}$$

$$\sin \beta = \frac{\sqrt{x_1^2 + y_1^2}}{\sqrt{x_1^2 + y_1^2 + z_1^2}}$$

$$\begin{pmatrix} x_3 \\ y_3 \\ z_3 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \beta & -\sin \beta & 0 \\ 0 & \sin \beta & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix}$$

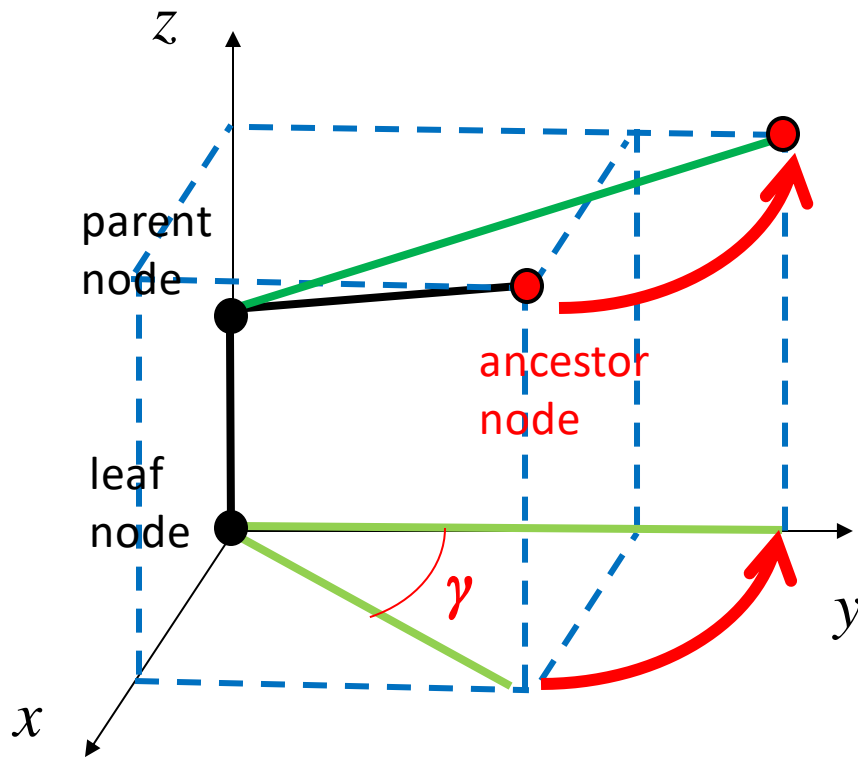
(5) Scale the tree so that the length of edge between the leaf and parent nodes is 1.



$$s = \frac{1}{\sqrt{x_1^2 + y_1^2 + z_1^2}}$$

$$\begin{pmatrix} x_4 \\ y_4 \\ z_4 \\ 1 \end{pmatrix} = \begin{pmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_3 \\ y_3 \\ z_3 \\ 1 \end{pmatrix}$$

(6) (Optional) Rotate the tree around  $z$ -axis an ancestor node moves onto  $yz$ -plane.



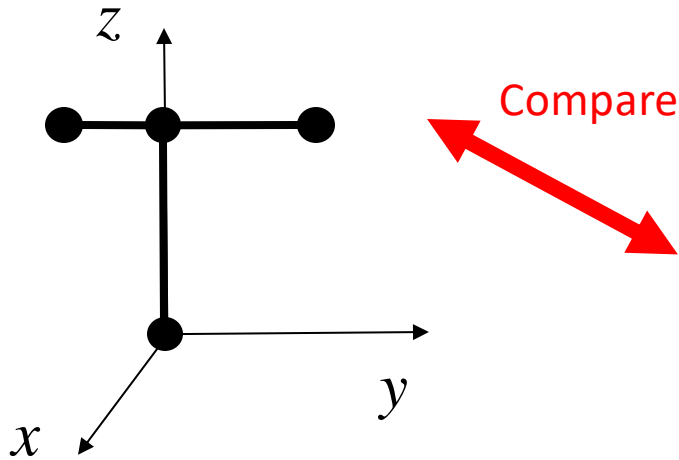
$$\cos \gamma = \frac{y_4}{\sqrt{x_4^2 + y_4^2}}$$

$$\sin \gamma = \frac{x_4}{\sqrt{x_4^2 + y_4^2}}$$

$$\begin{pmatrix} x_5 \\ y_5 \\ z_5 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \gamma & -\sin \gamma & 0 & 0 \\ \sin \gamma & \cos \gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_4 \\ y_4 \\ z_4 \\ 1 \end{pmatrix}$$

(7) For each leaf node of  $T$ , transform  $T$  and compare it with  $S$ .

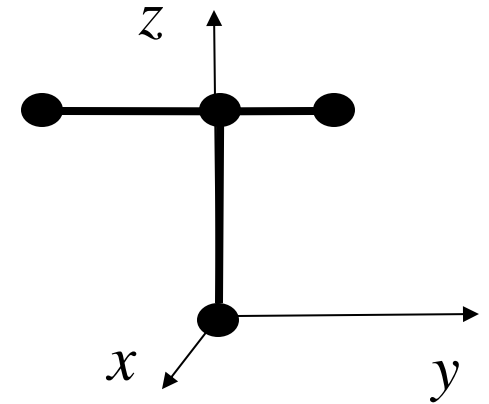
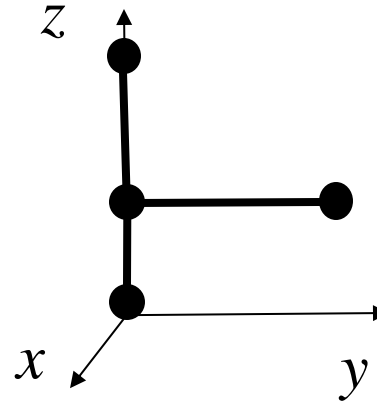
Tree  $S$



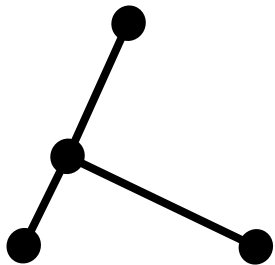
Compare



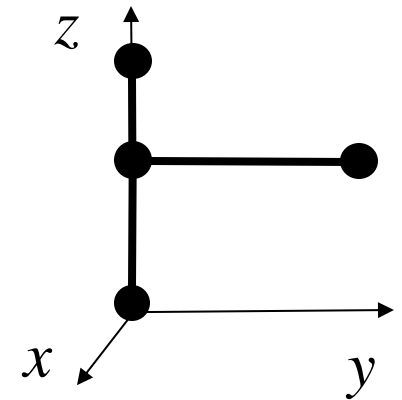
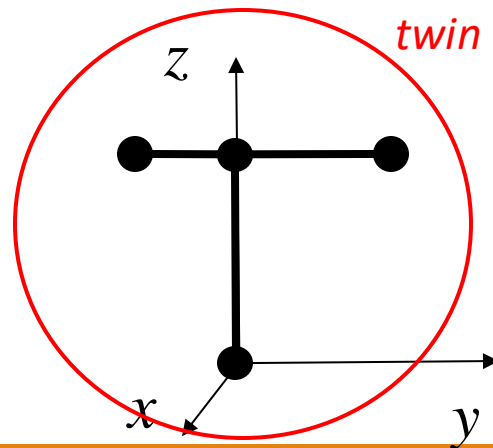
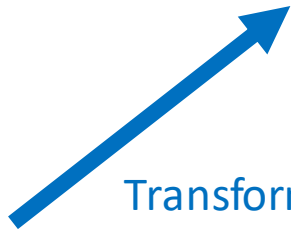
Transformed Tree  $T$



Tree  $T$

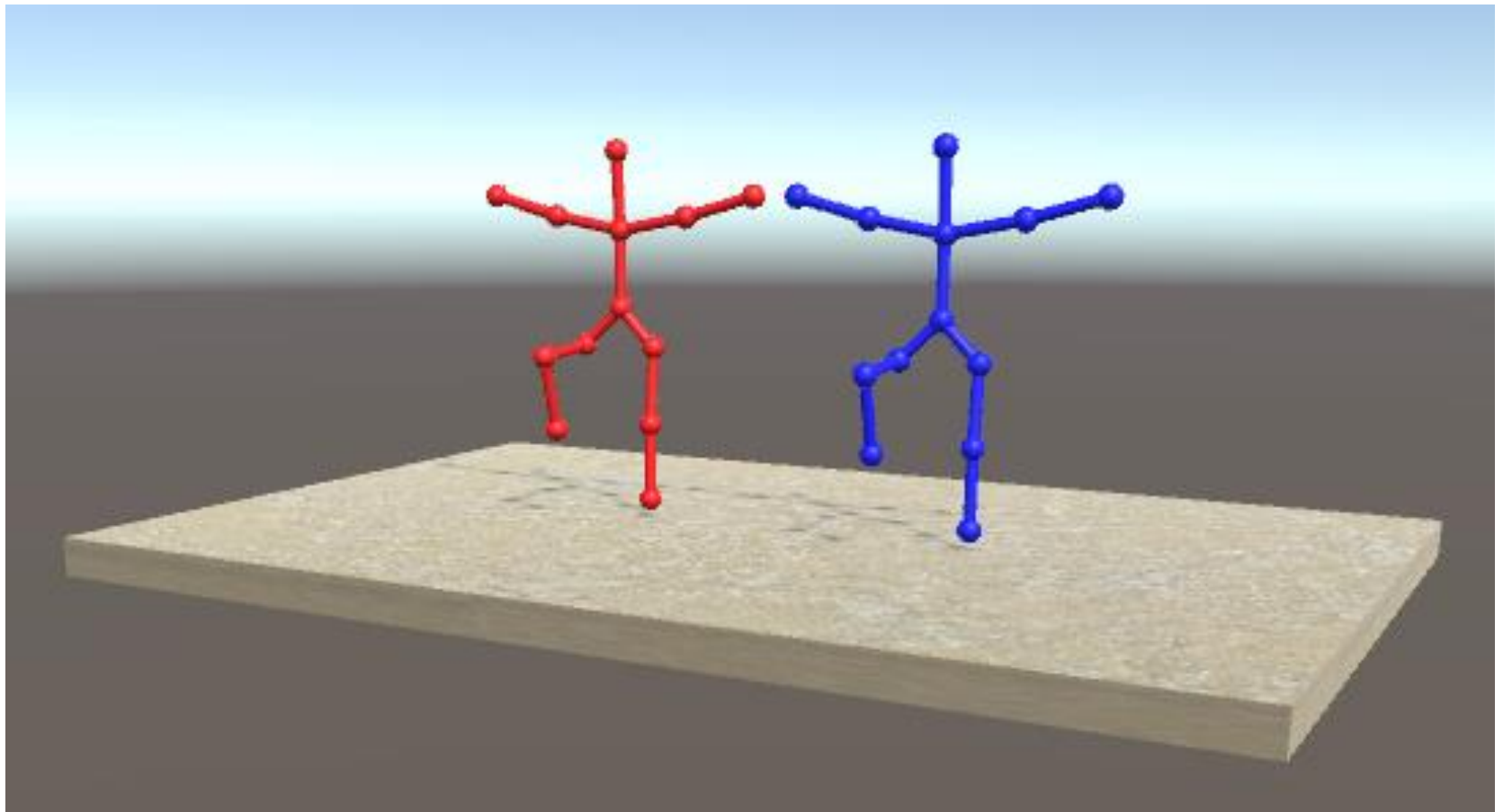


Transform





Thank you very much for your attention.



# F:Halting Problem

---

# Problem Summary

Solve the Halting problem of the following program.

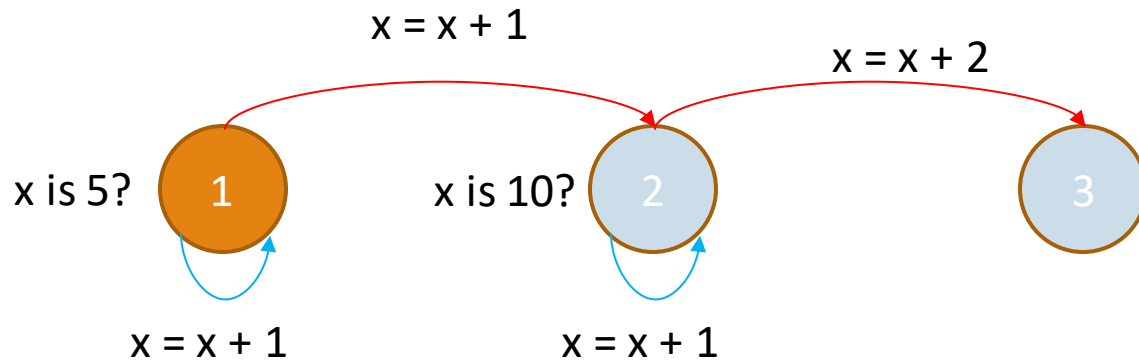
The program has two variables  $x$ ,  $i$ .

- In the initialization, set  $x = x_0$  and  $i = 1$
- (an execution step) When  $i < N + 1$ ,
  - if  $x$  is equal to  $a_i$ , then set  $x = x + b_i$  and  $i = c_i$ ,
  - otherwise, set  $x = x + d_i$  and  $i = e_i$ .
- When  $i$  becomes  $N + 1$ , the program terminates.
- For given  $x_0$ ,  $a$ ,  $b$ ,  $c$ ,  $d$ , determine whether the program halts or not.
  - If it halts, compute how many steps are executed.

Λ\_Λ ババババ  
(・ω・)=つ≡つ  
(つ≡つ=つ  
`/ )

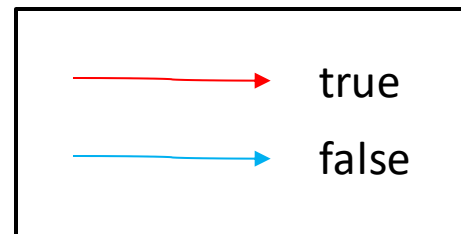
# Example

Sample 1 (N=2)



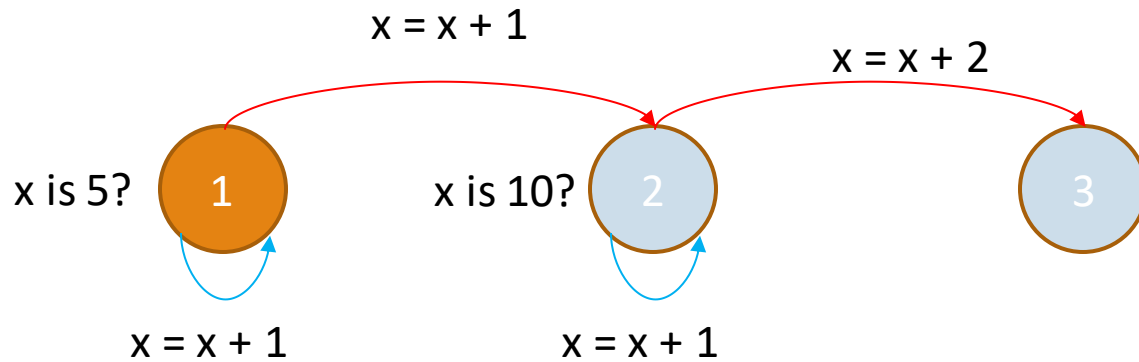
$x = 0$

$i = 1$



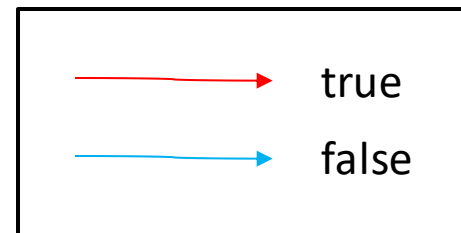
# Example

Sample 1 (N=2)



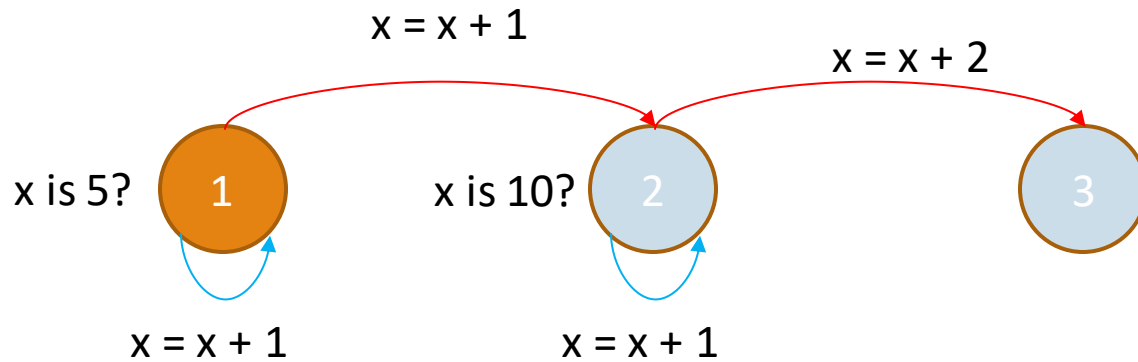
$x = 1$

$i = 1$



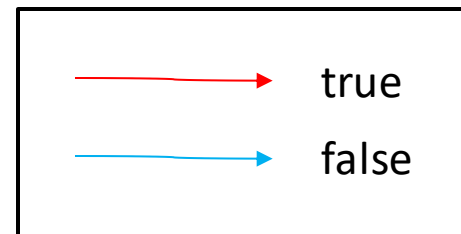
# Example

Sample 1 (N=2)



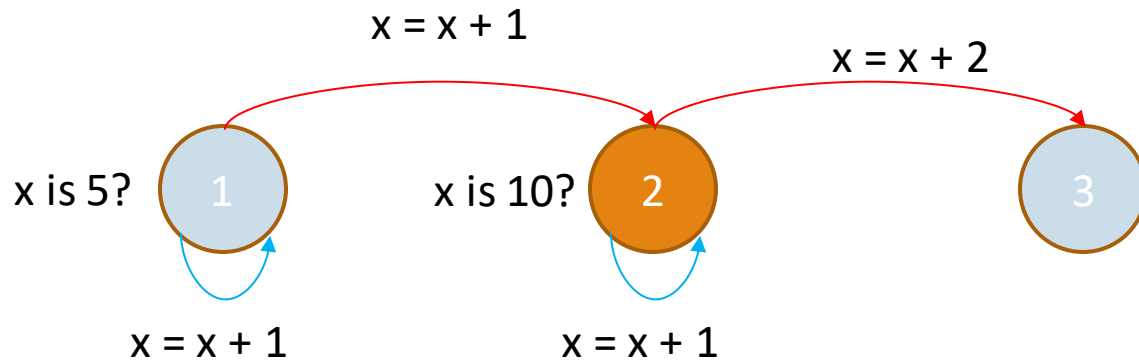
$x = 5$

$i = 1$



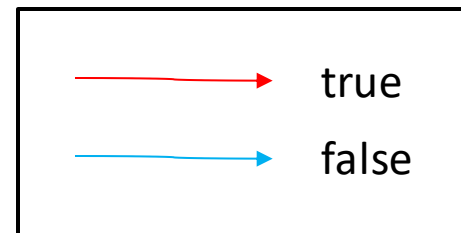
# Example

Sample 1 (N=2)



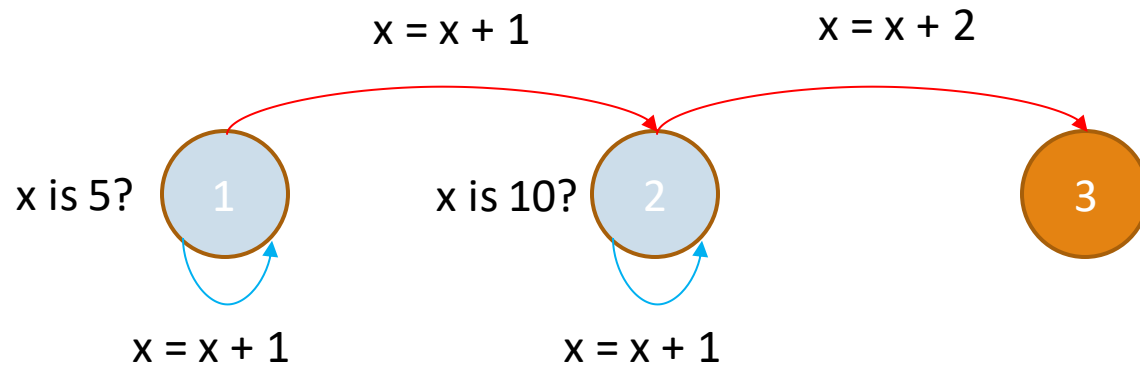
$x = 6$

$i = 2$



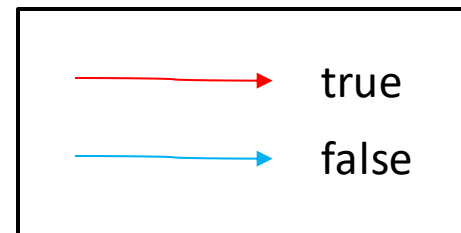
# Example

Sample 1 (N=2)



The program halts!  
The number of executed  
steps is 9.

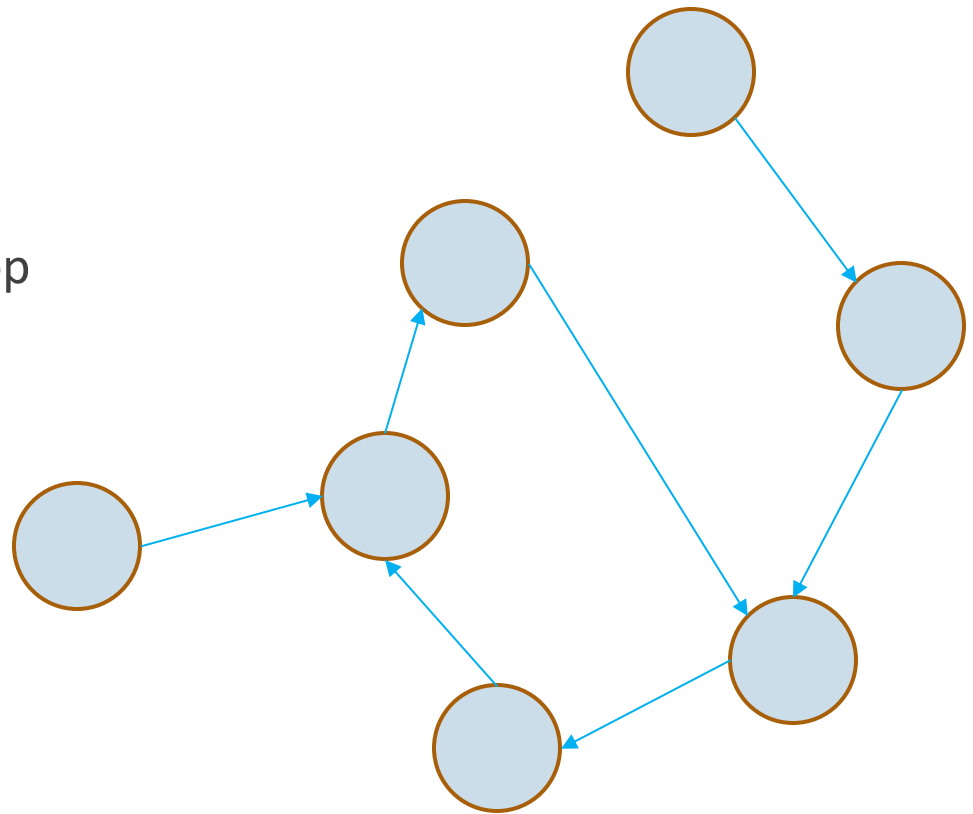
$x = 12$   
 $i = 3$





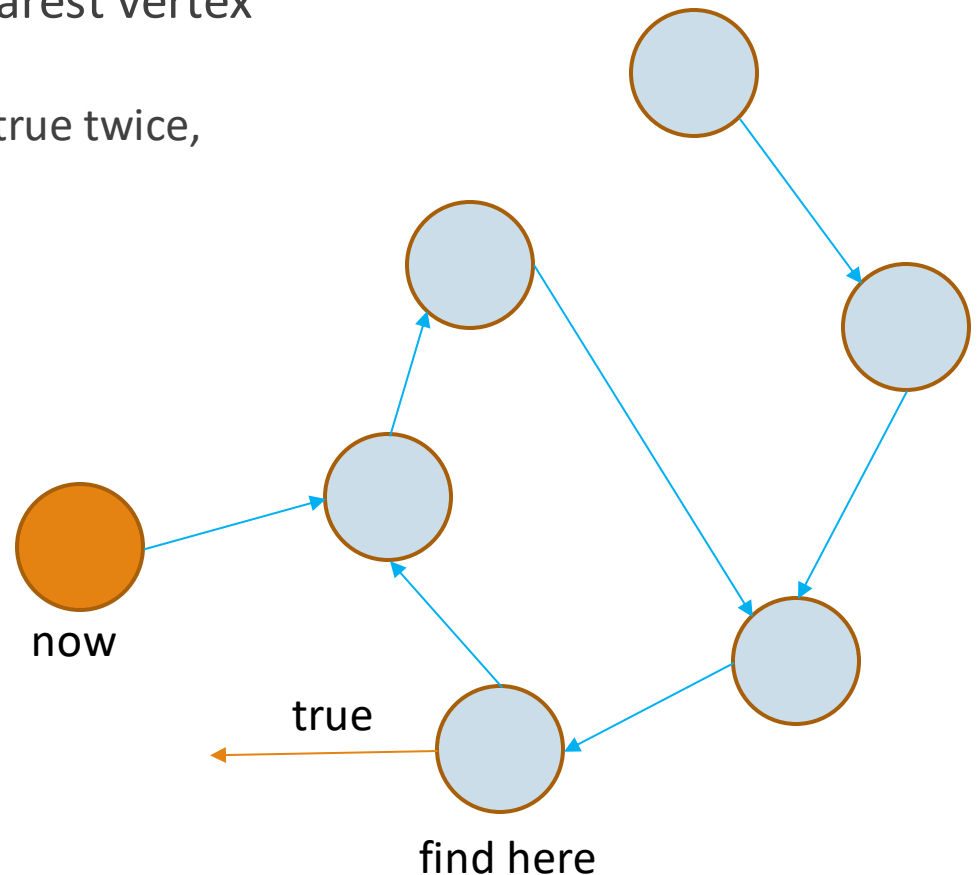
# Solution (Graph)

- Make a graph
  - vertex = state  $i$  (1, 2, 3, ...,  $N+1$ )
  - edge = false part
- This graph has at most one loop in each connected component.
  - outdegree = 1 for every vertices



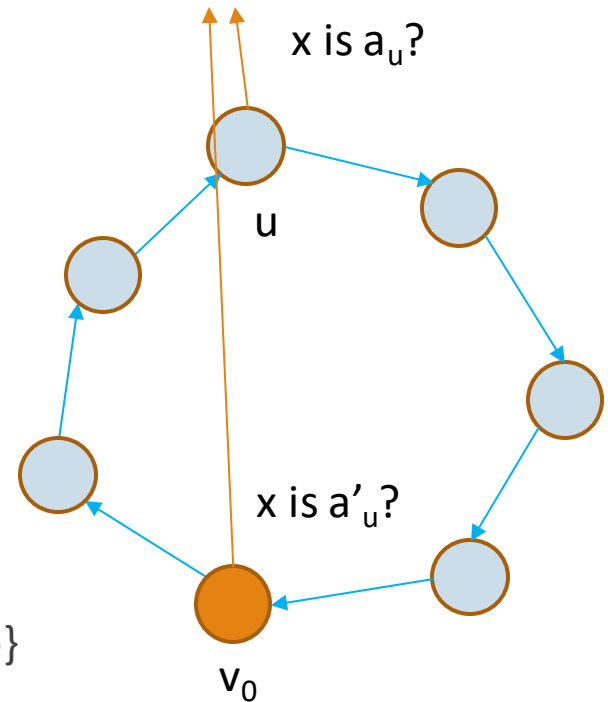
# Solution (query)

- (query) given  $(x, i)$ , find the nearest vertex where the condition is true.
  - If the same condition becomes true twice, the program does not halt.
- Remove one edge in the loop.
  - one tree + one loop
- Divide into two problems
  - in the tree
  - in the loop



# Solution (Loop part)

- Fix a vertex  $v_0$  (e.g. the root of the tree).
- $x$  become  $x + L$  after one loop.
- Reduction to  $v_0$ 
  - The condition  $x == a_u$  at  $u$
  - $\Leftrightarrow x == a'_u$  at  $v_0$
  - (in one loop)
- Make a table and do binary search here.
  - $\text{Table}[a'_u \bmod L] := \{a'_u : u \text{ is a vertex in the loop}\}$
  - $O(\log(N))$  for each query



# Solution (Tree part)

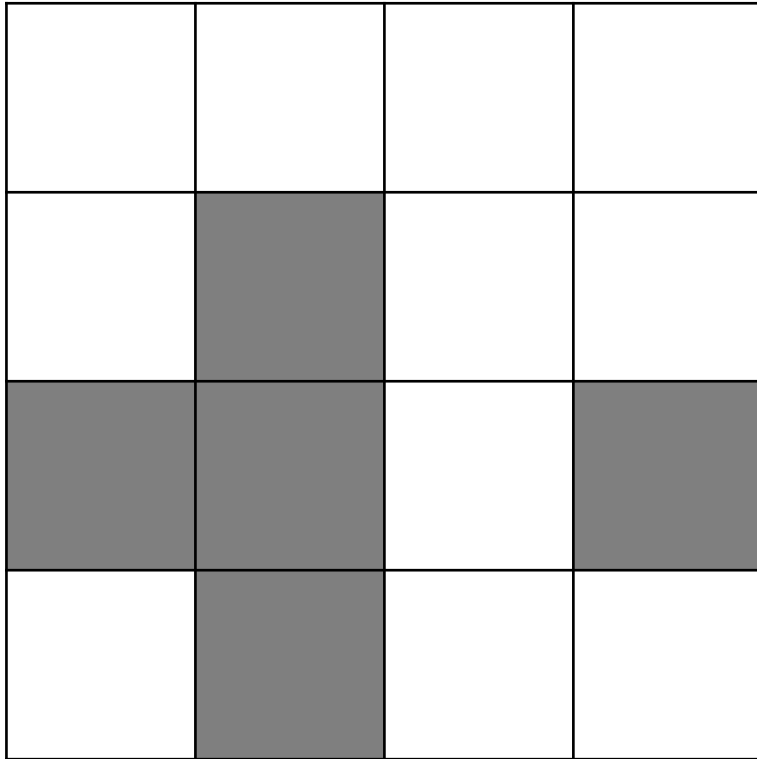
- The idea is the same as the loop part.
- There are two solutions of  $O(N\log(N))$ :
  1. Use persistent binary search tree
    - Not difficult because the insertion operation is static in this problem.
    - $O(\log(N))$  for each query
  2. Encode the tree to an array (like LCA  $\leftrightarrow$  RMQ)
    - Binary search on the array.
    - $O(\log(N))$  for each query

K: Draw in  
Straight Lines

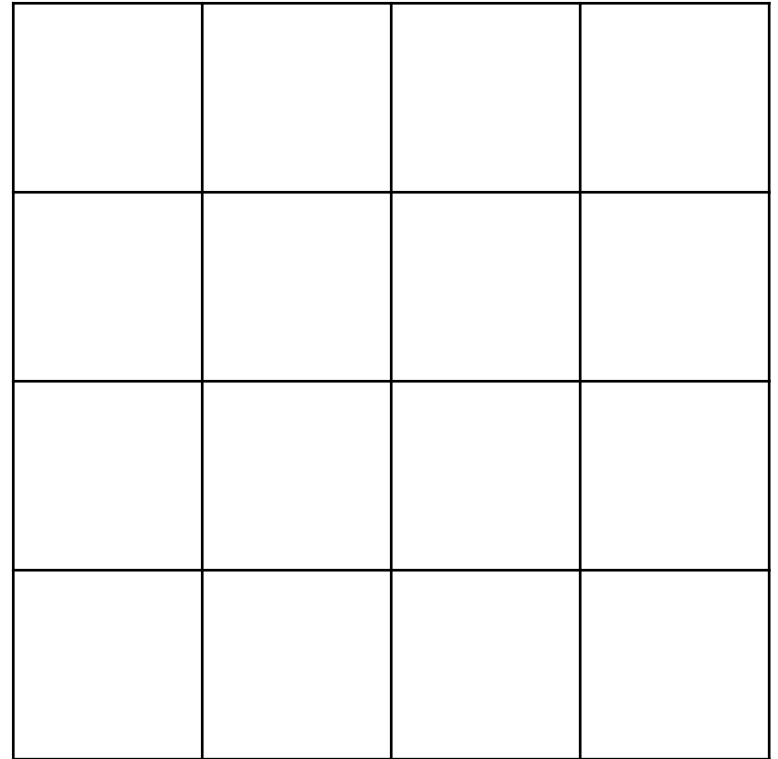
---

# Draw the specified image

Input

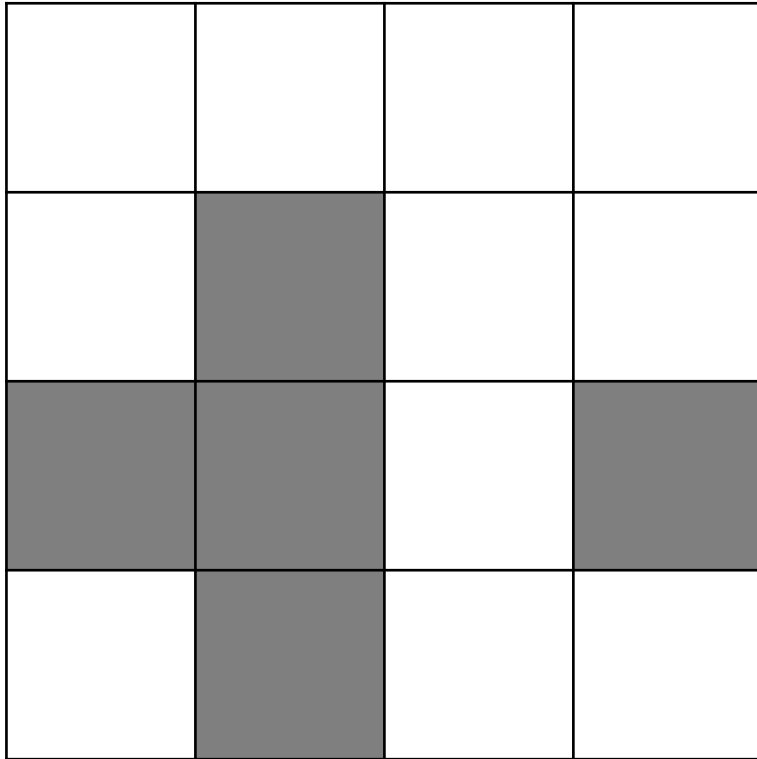


Solution

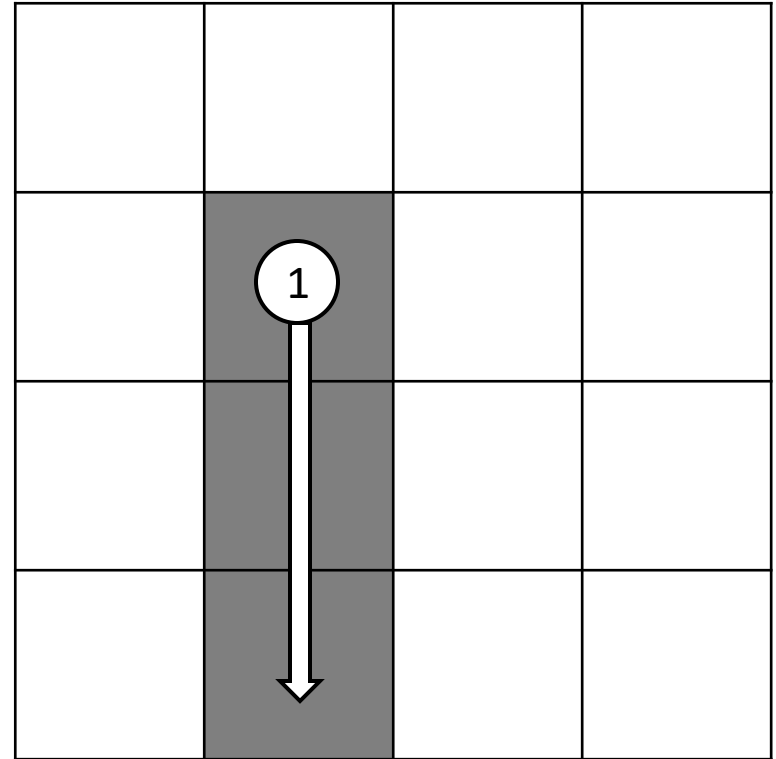


# Draw the specified image

Input

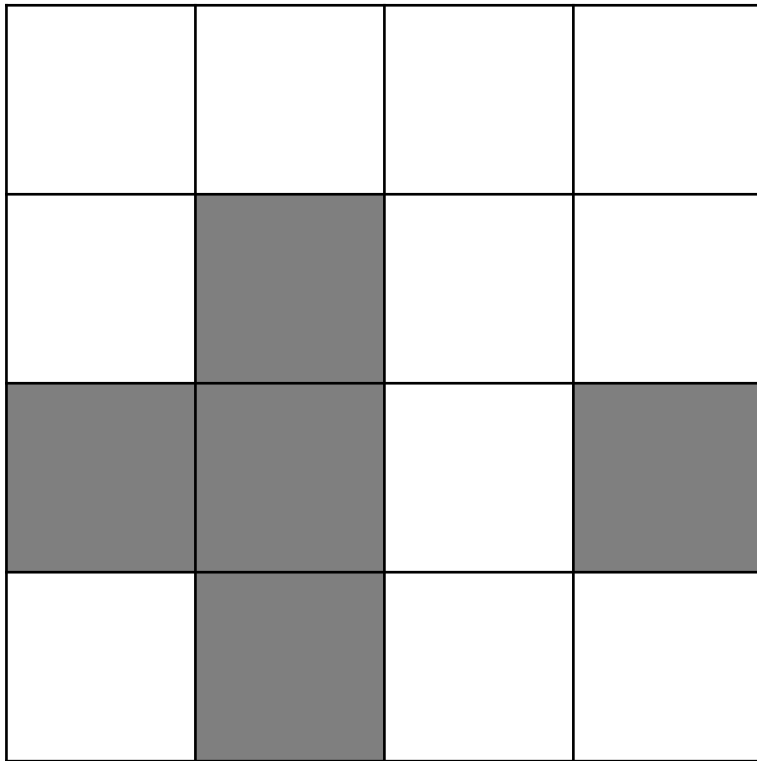


Solution

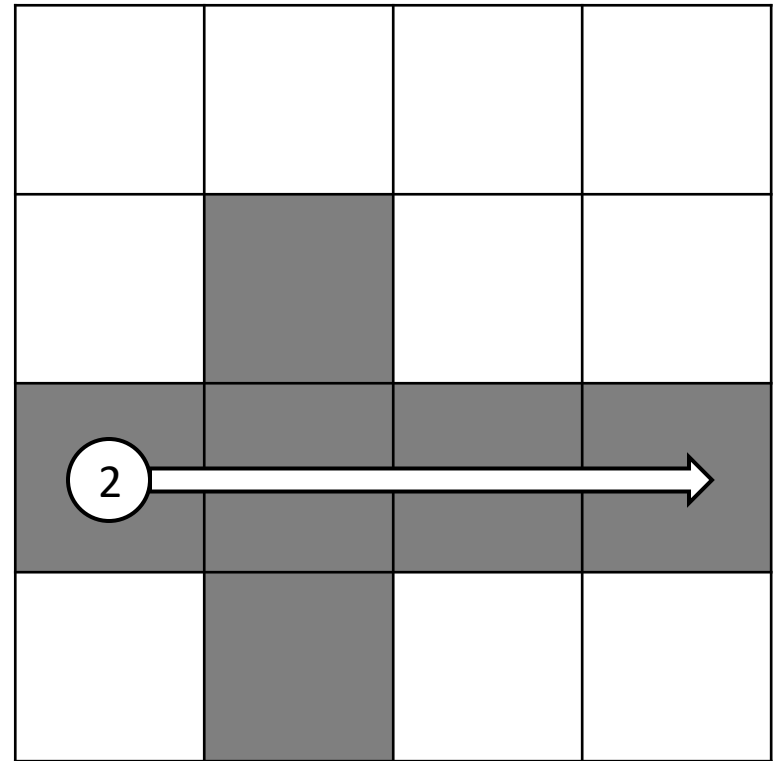


# Draw the specified image

Input



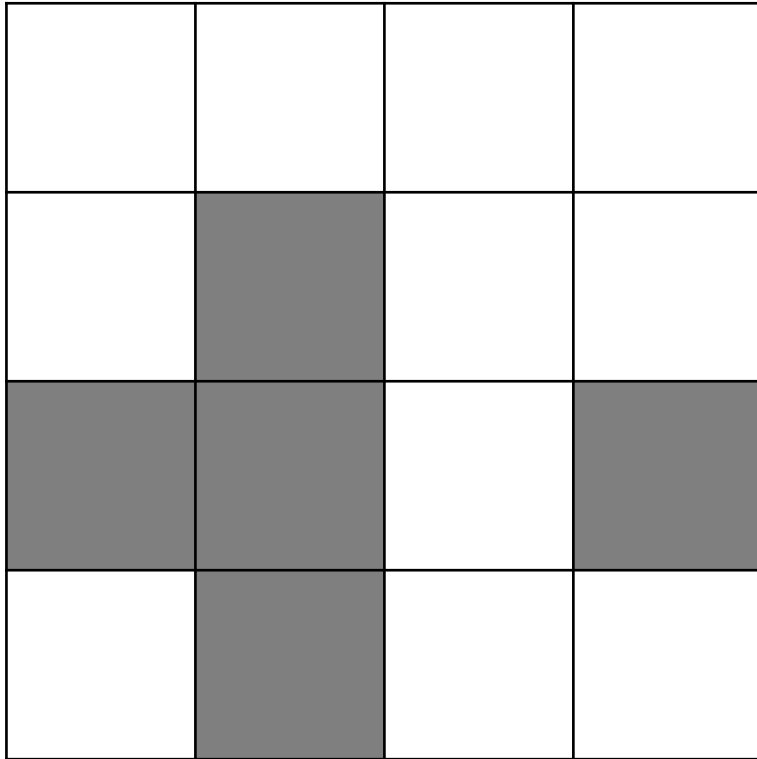
Solution



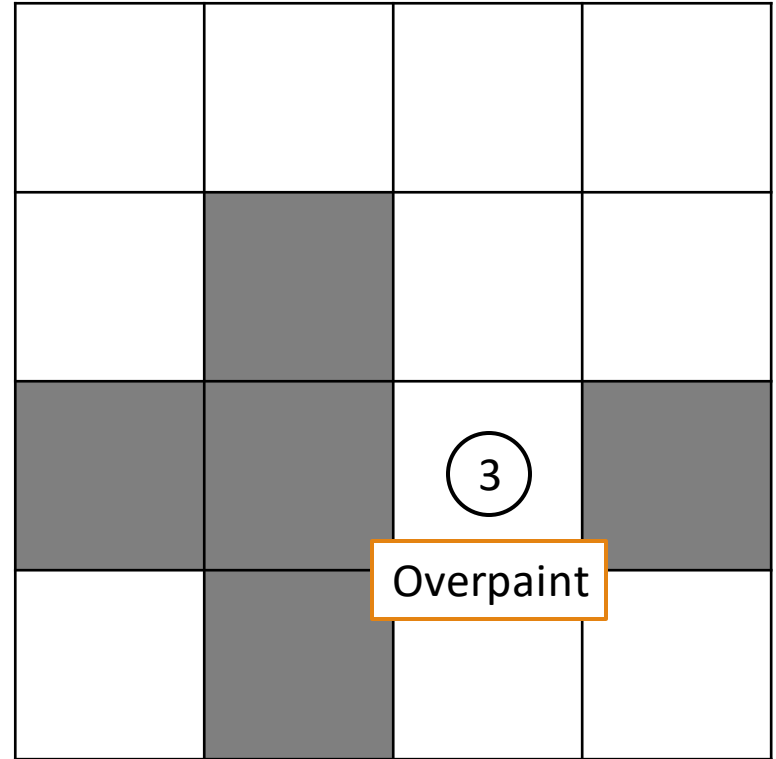


# Draw the specified image

Input

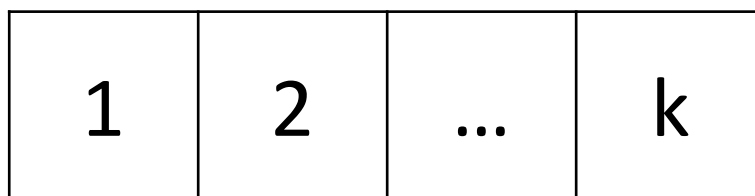


Solution



**Line-paint operation:** Paint  $1 \times k$  or  $k \times 1$  pixels either black or white.

$$\text{Cost} = ak + b$$

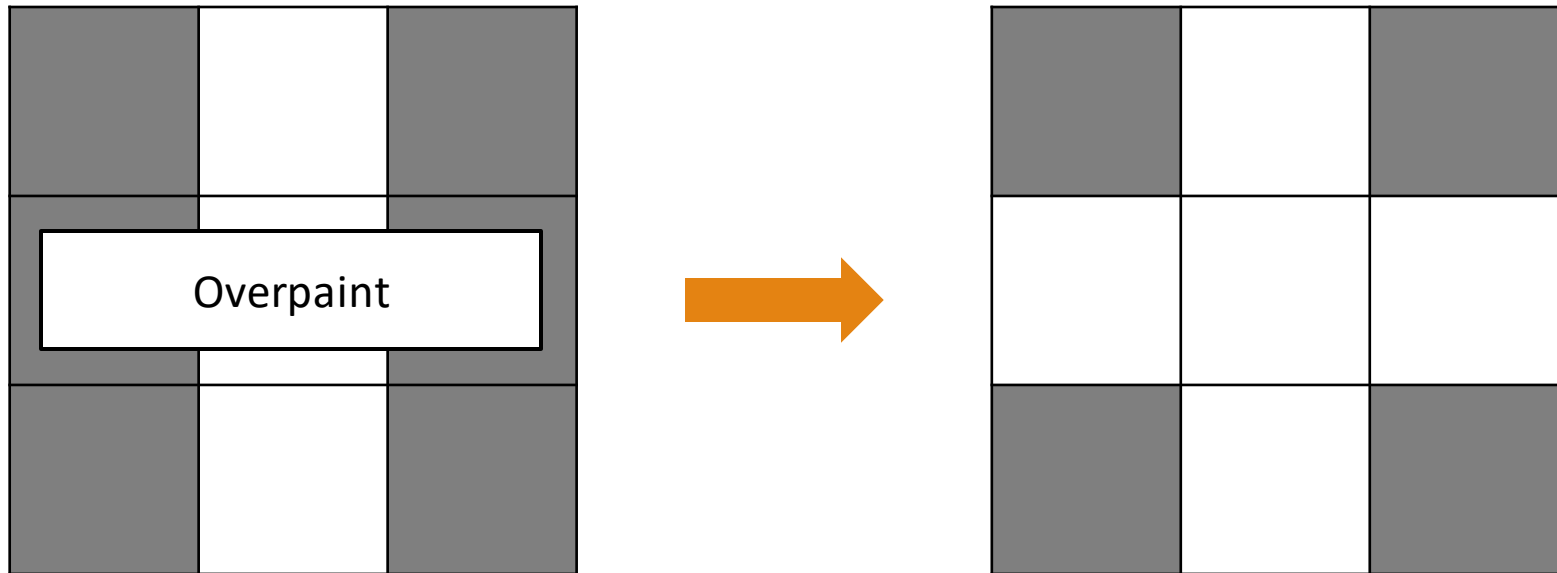


**Single-paint operation:** Paint a single pixel either black or white.

$$\text{Cost} = c$$

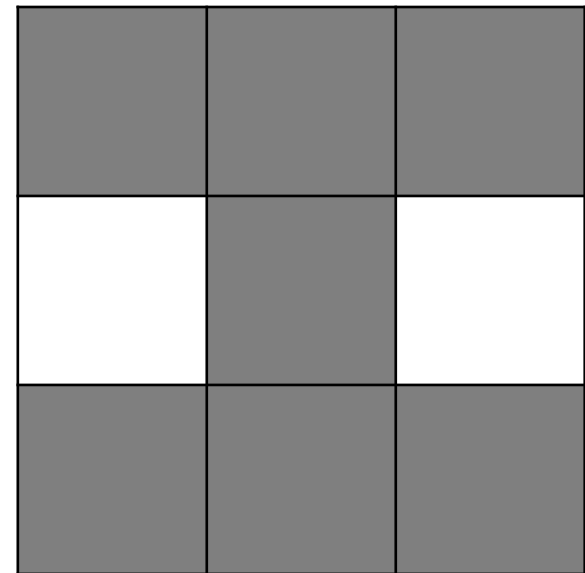
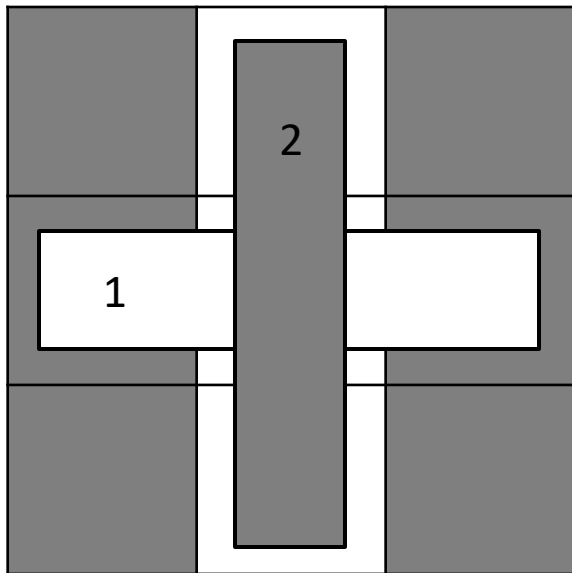
# Overpainting

You can overpaint black  $\Rightarrow$  white



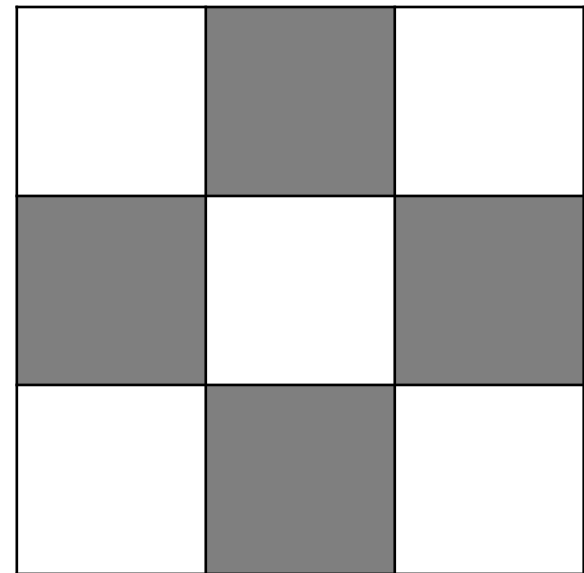
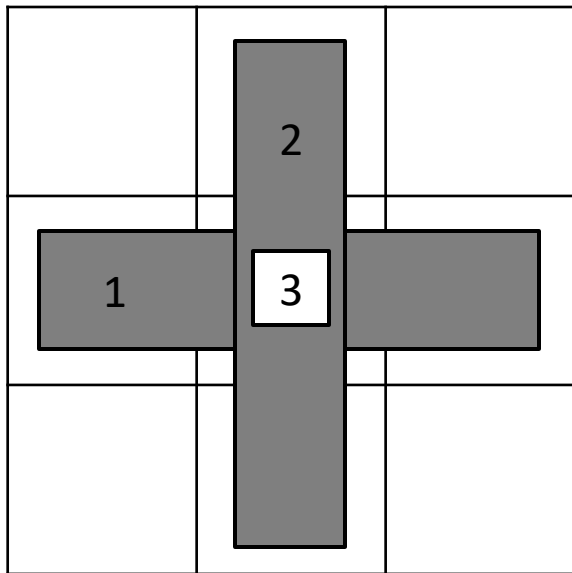
# Overpainting

You *cannot* overpaint white  $\Rightarrow$  black



# Overpainting

You can overpaint a pixel at most once



Solution

**Minimum Cut !!!!!**

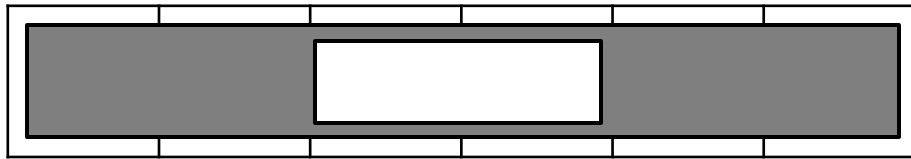
# Key observation 1

We can assume the following order of operations:

1. Black line-paint
2. White line-paint
3. Black/white single-paint (uniquely decided)

# Key observation 2

Once we have painted a pixel horizontally, we don't have to paint it horizontally anymore.



$$8a + 2b$$

∨



$$4a + 2b$$



## Variables

- $bh_{i,j} := [\text{paint}(i,j) \text{ black horizontally}]$
- $bv_{i,j} := 1 - [\text{paint}(i,j) \text{ black vertically}]$
- $wh_{i,j} := 1 - [\text{paint}(i,j) \text{ white horizontally}]$
- $wv_{i,j} := [\text{paint}(i,j) \text{ white vertically}]$

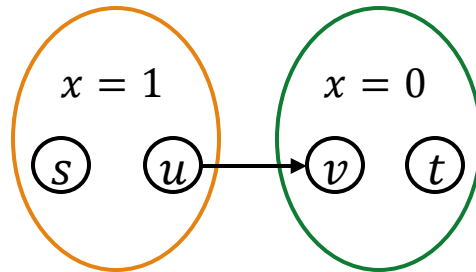
$$[\text{condition}] := \begin{cases} 1 & (\text{if the condition holds}) \\ 0 & (\text{if the condition does not hold}) \end{cases}$$

## Total cost of black-horizontal-line-paints

$$a \times \sum_{i,j} bh_{i,j} + b \times \sum_{i,j} bh_{i,j} \overline{bh_{i,j+1}}$$

$$\bar{x} := 1 - x$$

$\sum_{uv} c_{uv} x_u \bar{x}_v$  can be minimized by a reduction to s-t min-cut 😊



## Cost of painting $(i, j)$ black

$$\frac{c \times bv_{i,j} \overline{bh_{i,j}}}{\text{blue line}} + \frac{\infty \times (\overline{wh_{i,j}} + wv_{i,j})}{\text{orange line}}$$

No black-line  
⇒ black-single

cannot overpaint black over white

## Cost of painting $(i, j)$ white

$$\frac{c \times (bh_{i,j}\overline{wv_{i,j}} + wh_{i,j}\overline{bv_{i,j}})}{\quad} + \frac{\infty \times bh_{i,j}\overline{bv_{i,j}}}{\quad}$$

↑  
black-line & no white-line  
⇒ white-single

↑  
cannot overpaint  
a pixel twice

Thanks to the observation 2, we can skip other cases ( $bh_{i,j}wh_{i,j}$  and  $\overline{bv_{i,j}}\overline{wv_{i,j}}$ )



# J:Fun Region

---

Image source: [https://en.wikipedia.org/wiki/Island#/media/File:Fernando\\_noronha.jpg](https://en.wikipedia.org/wiki/Island#/media/File:Fernando_noronha.jpg)

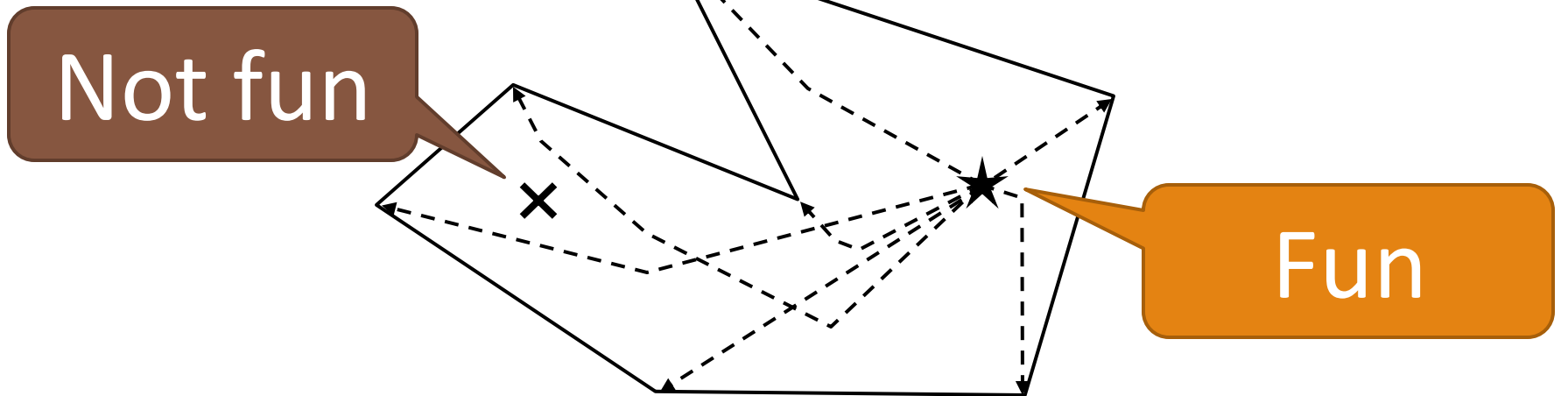
# Problem

You are given a polygon.

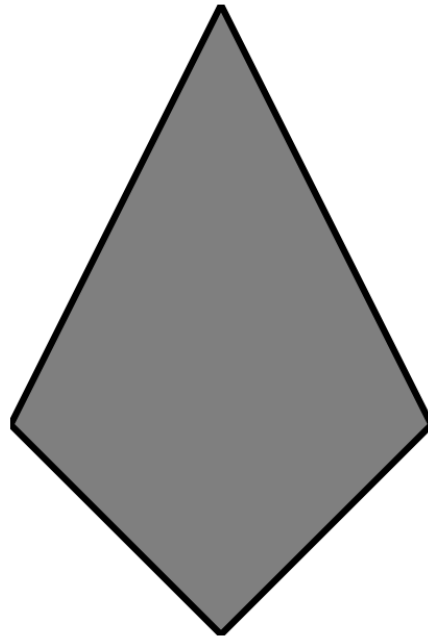
An inside point is *fun* if you can reach every vertex by some spiral path.

*Fun region* := All fun points.

Calculate the area of a fun region.

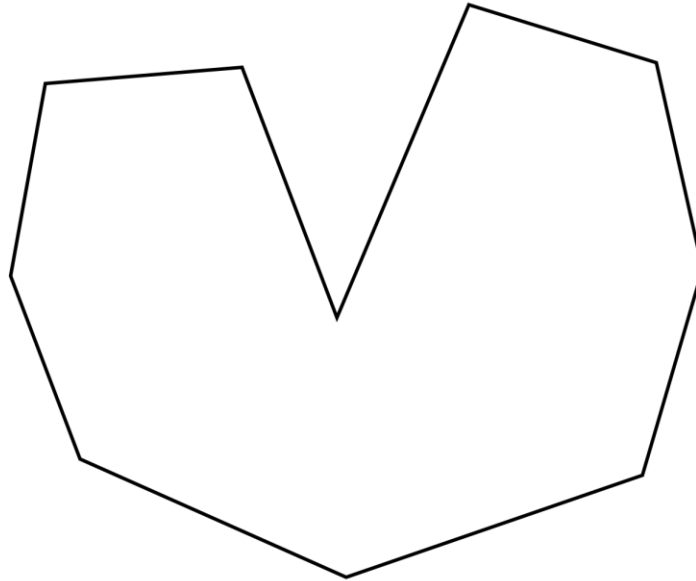


# Trivial Case



Convex  $\rightarrow$  Fun region is entire parts

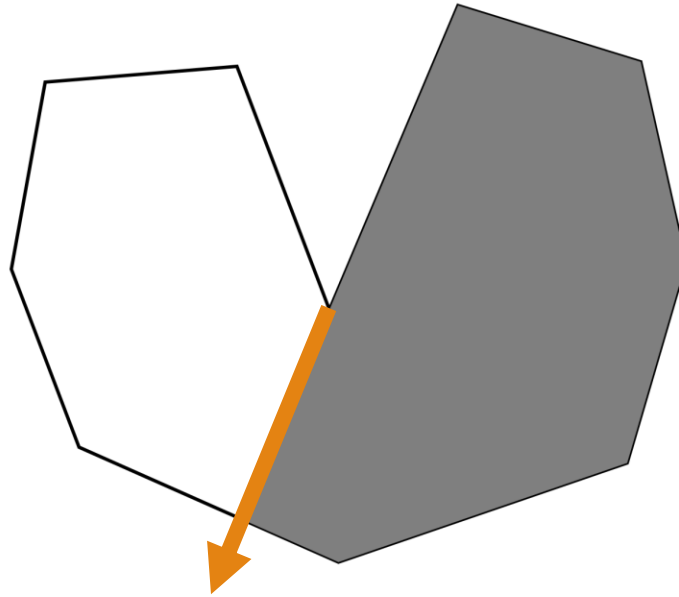
Concave?



Only one concave vertex  $\rightarrow$  ?

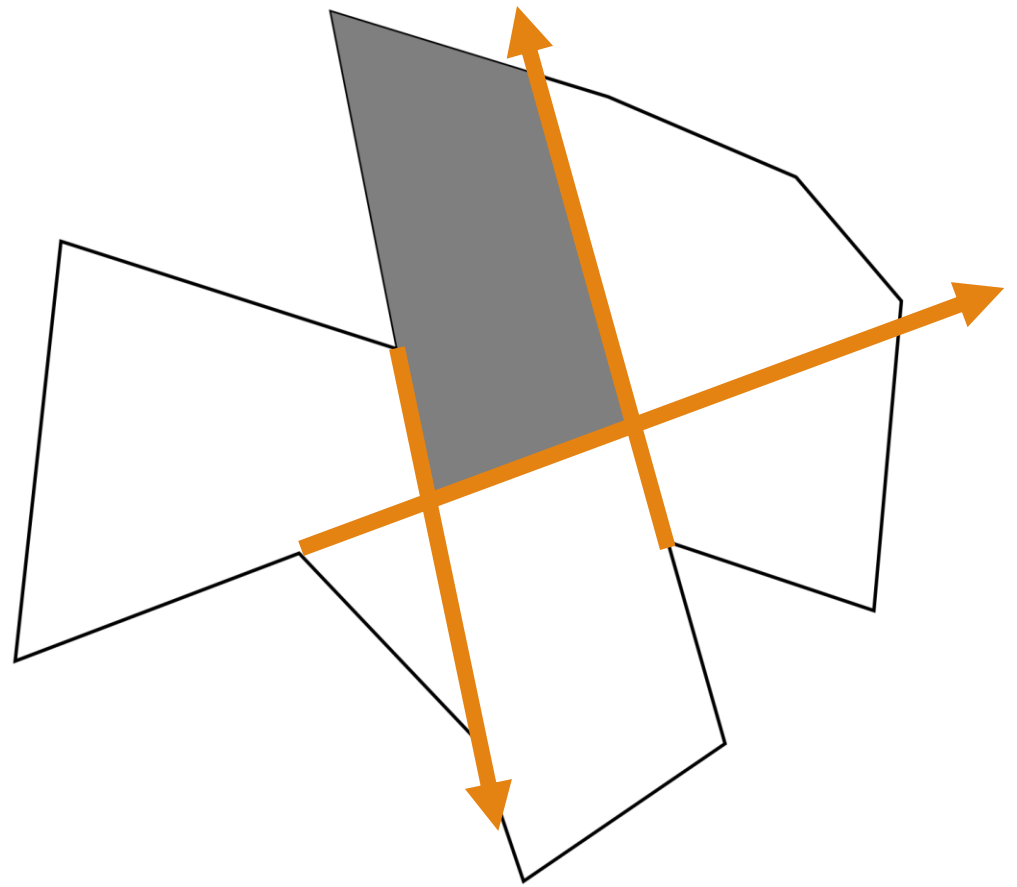


# Concave?



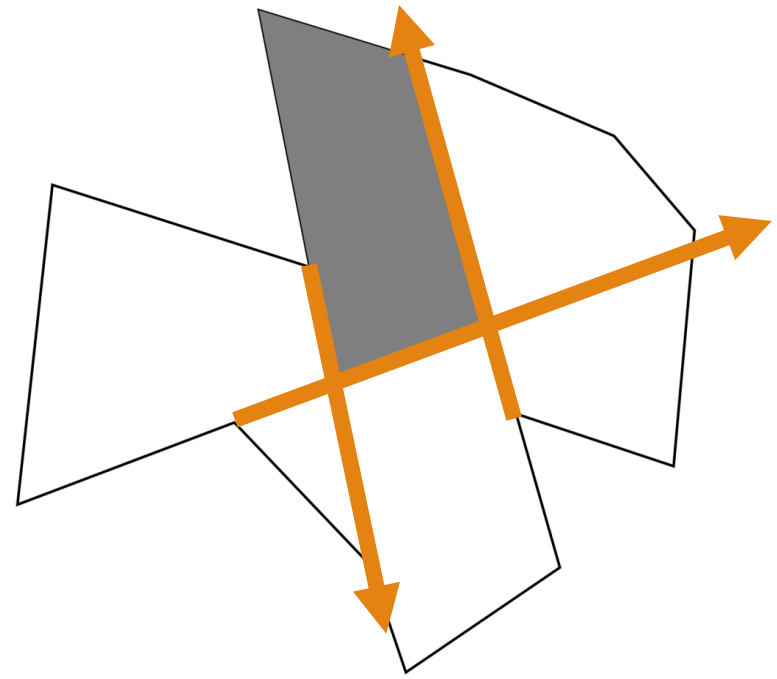
Fun region can be obtained by cutting the polygon with a segment from the concave vertex.

In General



For every concave point, perform cuts.  
The remained polygon is the answer.

# Solution



Perform non-convex cuts from each concave point.

Each cut can be done in  $O(n)$  time.

Total complexity is  $O(n^2)$ .

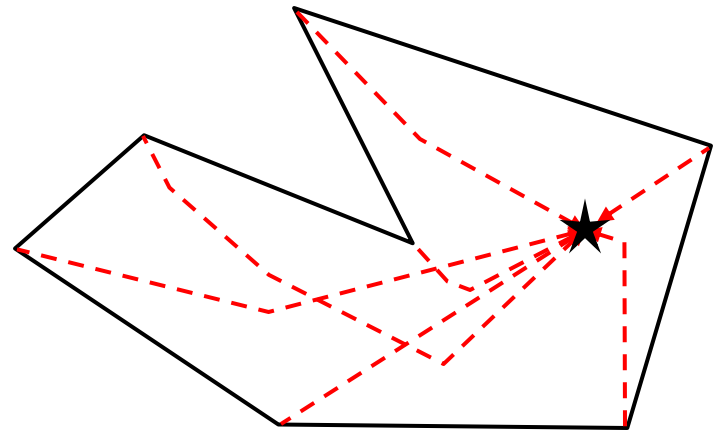
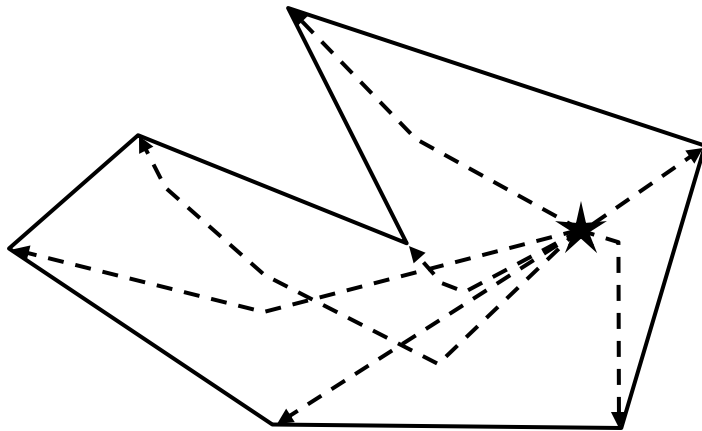
Correctness?

# Correctness (1/7)

Point  $p$  is fun

$\Leftrightarrow$  Every vertex is reachable from  $p$  by a spiral path

$\Leftrightarrow p$  is reachable from every vertex by a ccw-spiral path



# Correctness (2/7)

Point  $p$  is fun

$\Leftrightarrow$  Every vertex is reachable from  $p$  by a spiral path

$\Leftrightarrow p$  is reachable from every vertex by a ccw-spiral path

Let  $S(v) := \{p \mid p \text{ is reachable from } v \text{ by a ccw-spiral path}\}$ .

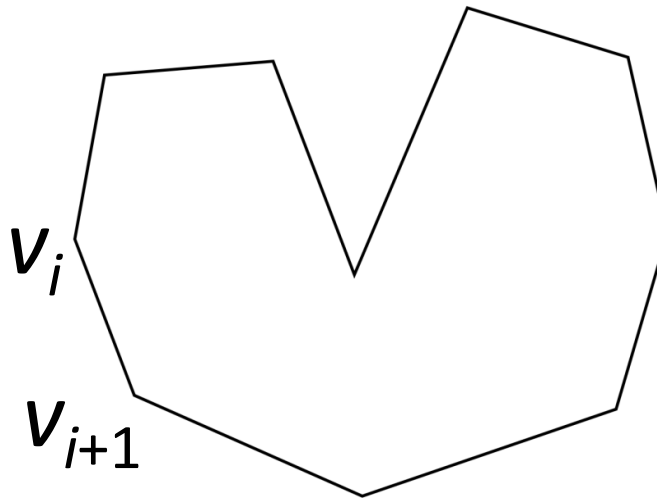
**Fun region** =  $\bigcap_{v:\text{vertex}} S(v)$ .

# Correctness (3/7)

When  $v_{i+1}$  is non-concave,  $S(v_{i+1}) \subseteq S(v_i)$  holds. So,

$$I := \{i \mid v_{i+1} \text{ is concave}\},$$

$$\text{Fun region} = \bigcap_{i \in I} S(v_i).$$

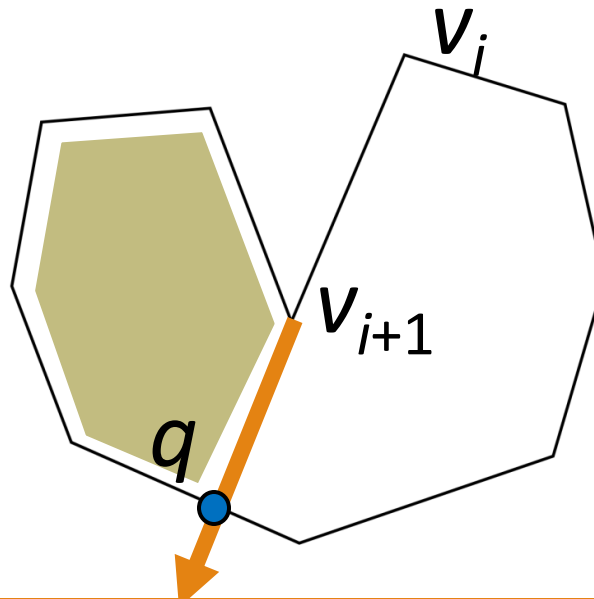


# Correctness (4/7)

Assume that  $v_{i+1}$  is concave.

Take a line that passes  $v_i$  and  $v_{i+1}$  and let  $q$  be a first contact point.

We can see that  $S(v_i)$  does not include any points in a region enclosed by  $v_{i+1}q$  and the polygon. Why?

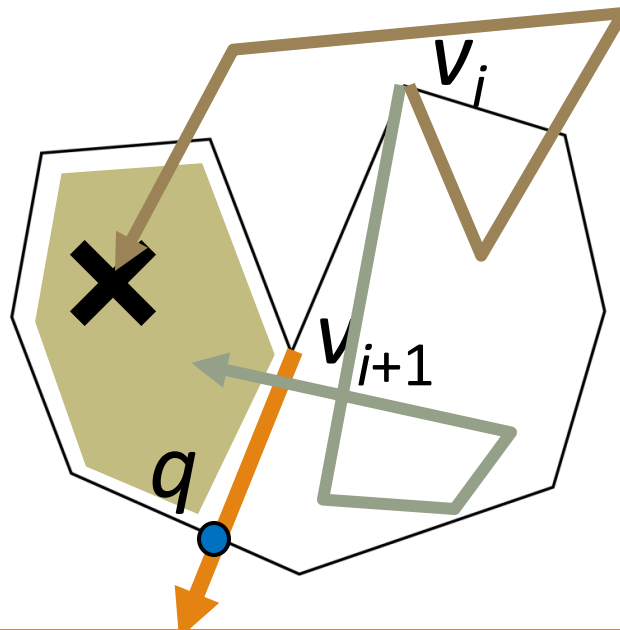




# Correctness (5/7)

Suppose that there exists some ccw-spiral path from  $v_i$  to  $X$ .

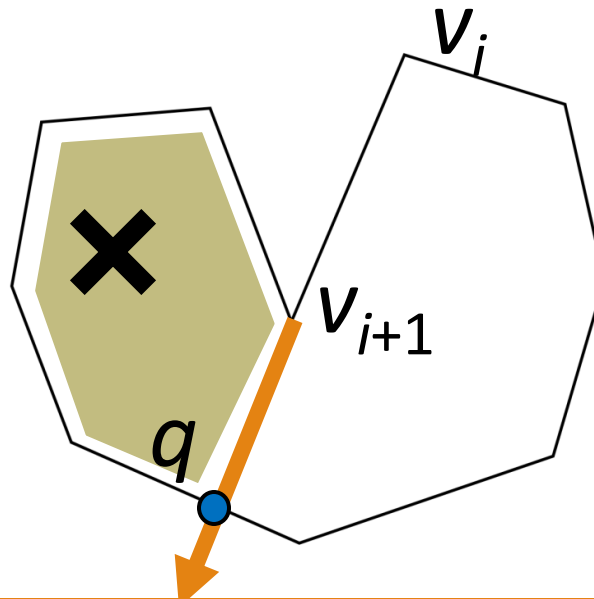
- The ccw-spiral path cannot cross segments  $v_{i+1}q$ .
- If the path can reach  $X$  without crossing  $v_{i+1}q$ , it implies there is a hole in the input. Contradiction.



# Correctness (6/7)

This implies that (remaining region by cuts)  $\supseteq$  (fun region).

Lastly, we will prove that " $\supseteq$ " is actually " $=$ ".



# Correctness (7/7)

“(remaining region by cuts) = (fun region)”

Suppose that (remaining region by cuts) - (fun region) is nonempty. This means there exists a point  $X$  and vertex  $v_i$  such that  $X$  won't be cut and  $X$  is not in  $S(v_i)$ . The boundary of  $S(v_i)$  is a polygon such that each edge is either (i) some edge of original polygon or (ii) half line passing from  $v_j$  to  $v_k$  where  $v_k$  is concave vertex. When  $X$  is in a region enclosed by the polygon and half line  $v_i v_k$ ,  $X$  should have been cut by a half line from  $v_k$ . This is contradiction.

