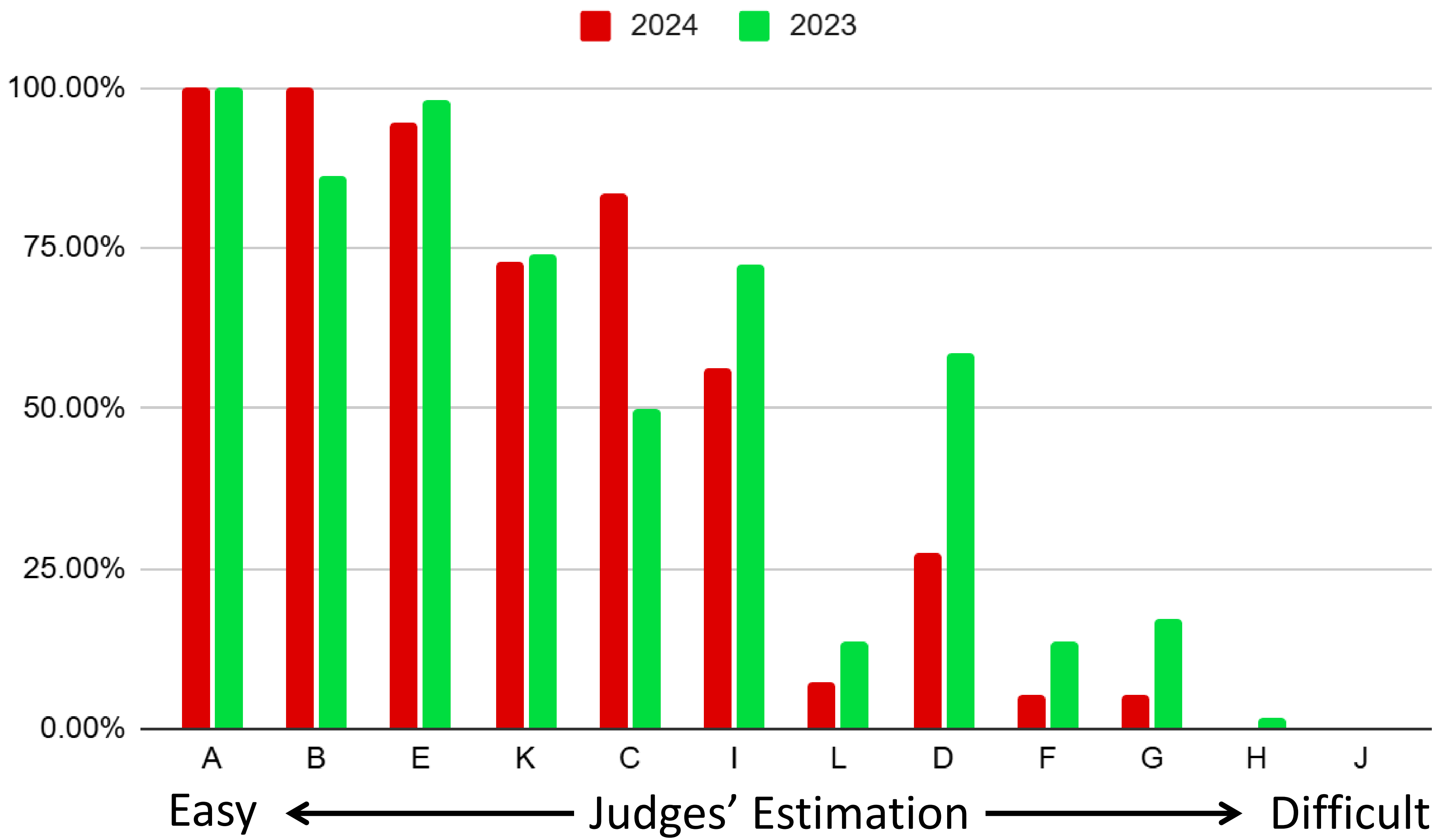
Commentaries on Problems

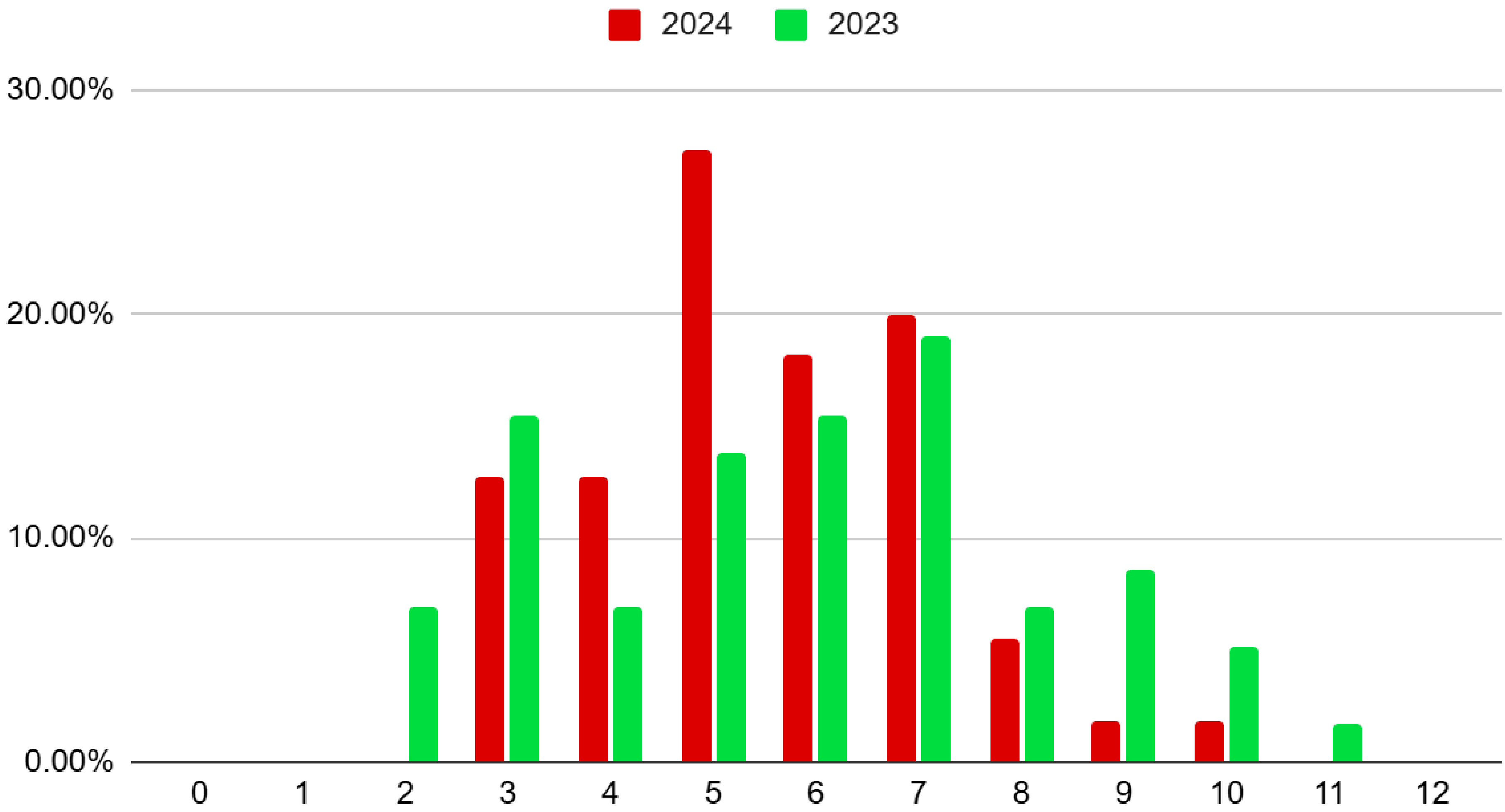
ICPC 2024 ASIA YOKOHAMA REGIONAL JUDGE TEAM
(CHIEF: YUTARO YAMAGUCHI)

#ACs vs. Problems



ALL teams get ACs for Problems A and B!! Congratulations!!

#Teams vs. #ACs

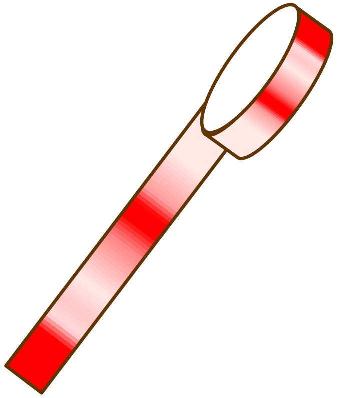


ALL teams get at least 3 ACs!!! Congratulations!!!

A: Ribbon on the Christmas Present

PROPOSER: KAZUHIRO INABA
AUTHOR: TOMOHARU UGAWA

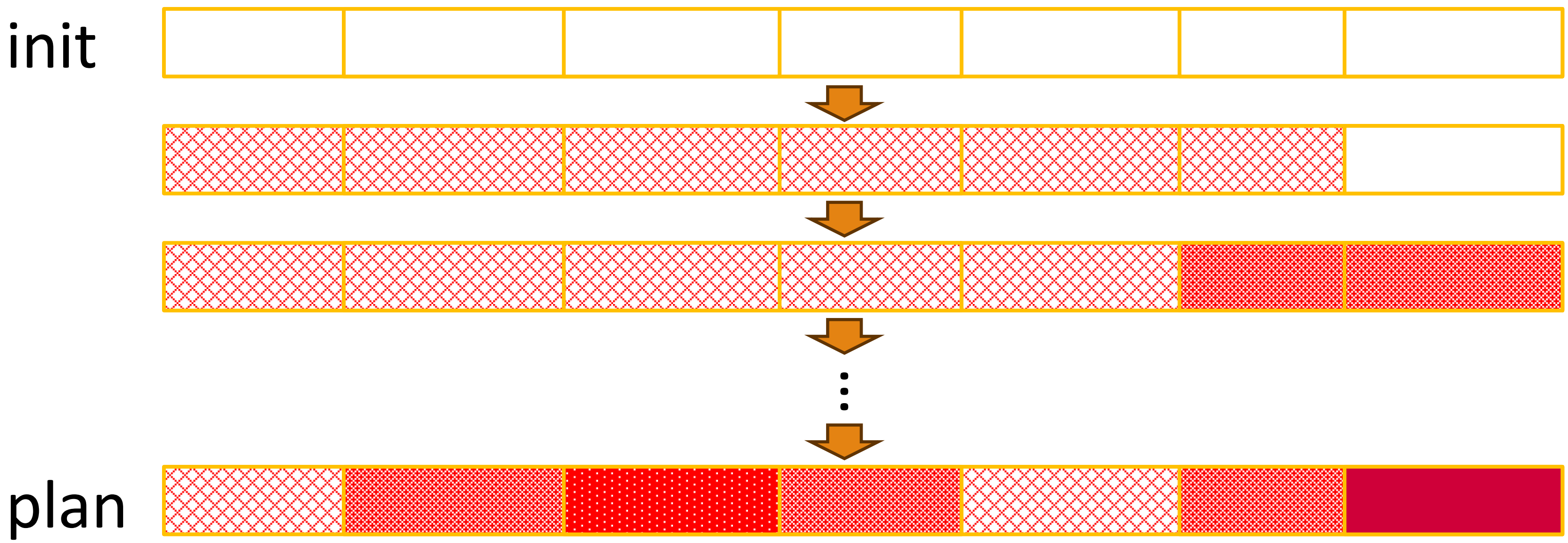
Problem



Dye white ribbon and make the "planned pattern."

- Contiguous segments can be dyed in a single step
- A darker color masks lighter color

Compute the minimum possible number of dyeing steps



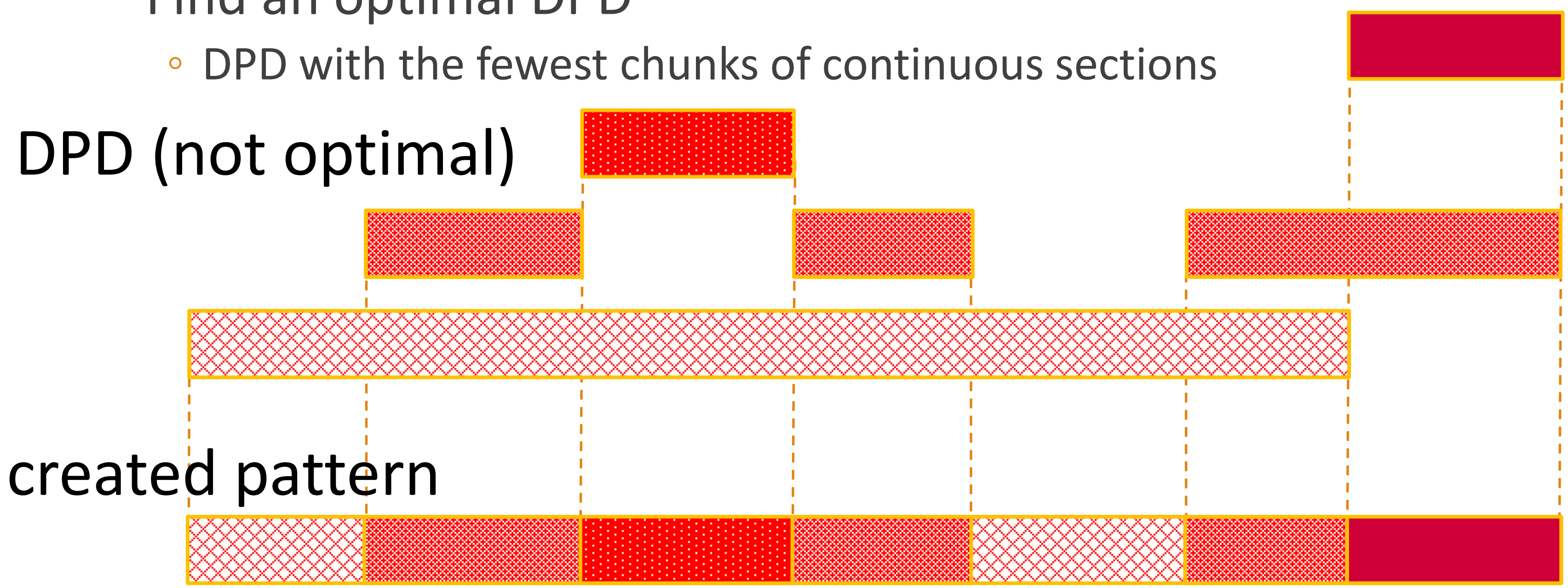
Dyeing Process Diagram (DPD)

Consider the dyeing process using a layered diagram, DPD

- Dye a *chunk* of contiguous sections at once
- Dye from lighter to darker colors

Find an optimal DPD

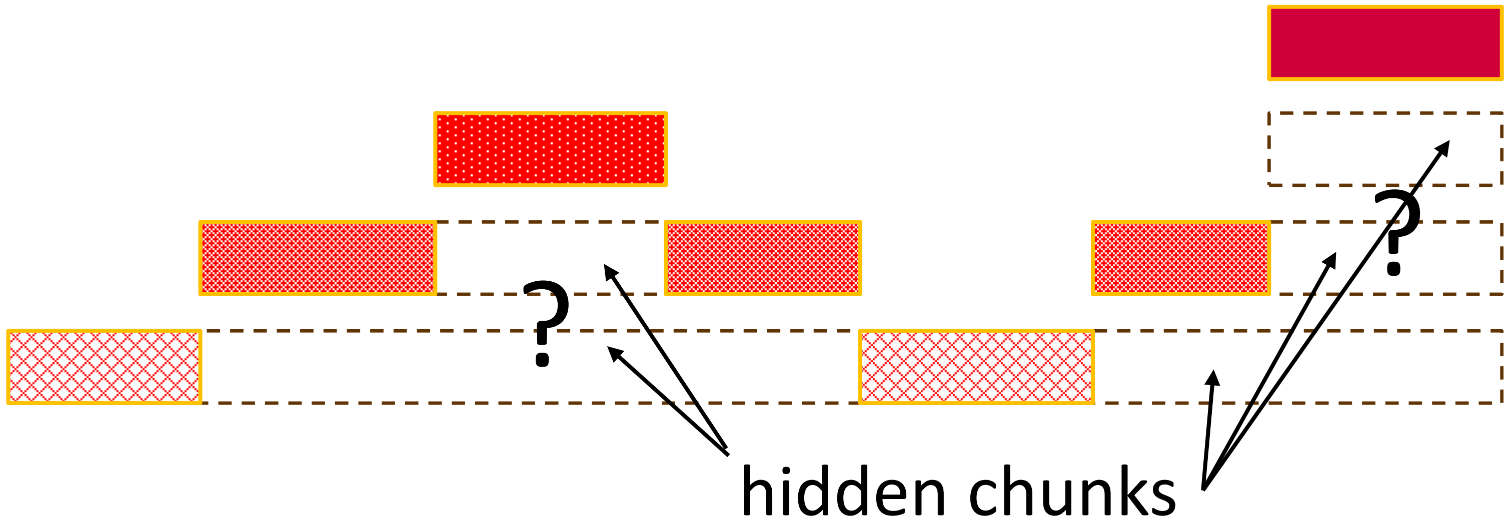
- DPD with the fewest chunks of continuous sections



Key Observation

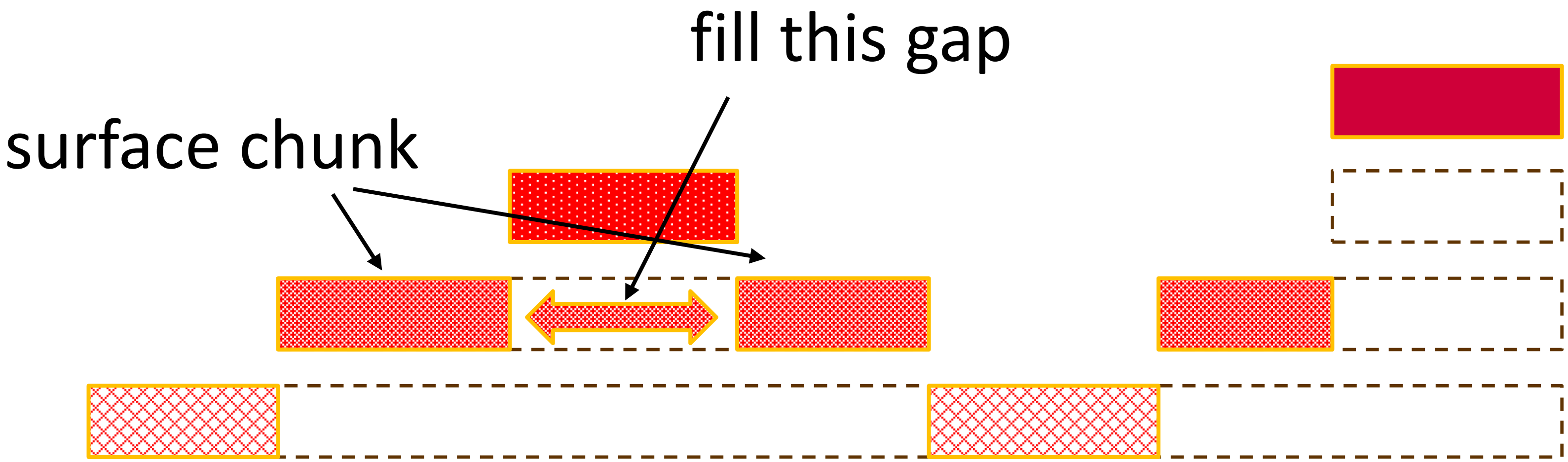
Any DPD will give the planned pattern if the surface is correct for each section.

Our Problem:
Fill some of the hidden chunks and make an optimal DPD



Approach

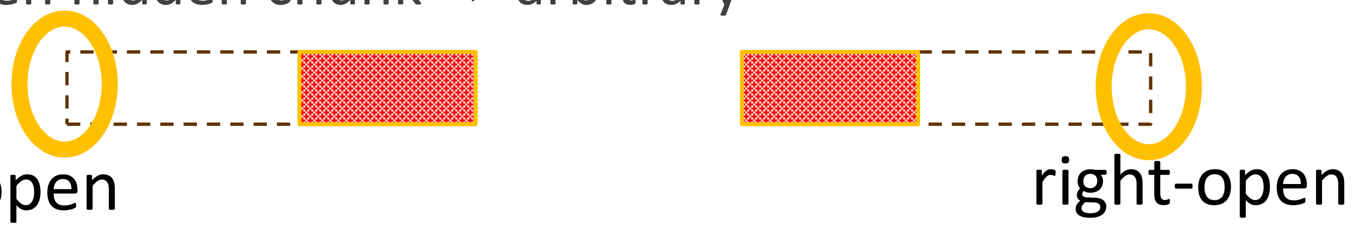
Fill the hidden chunk between surface chunks and merge them into a single chunk.



Cases

- hidden chunk between chunks => must be filled

- half-open hidden chunk => arbitrary

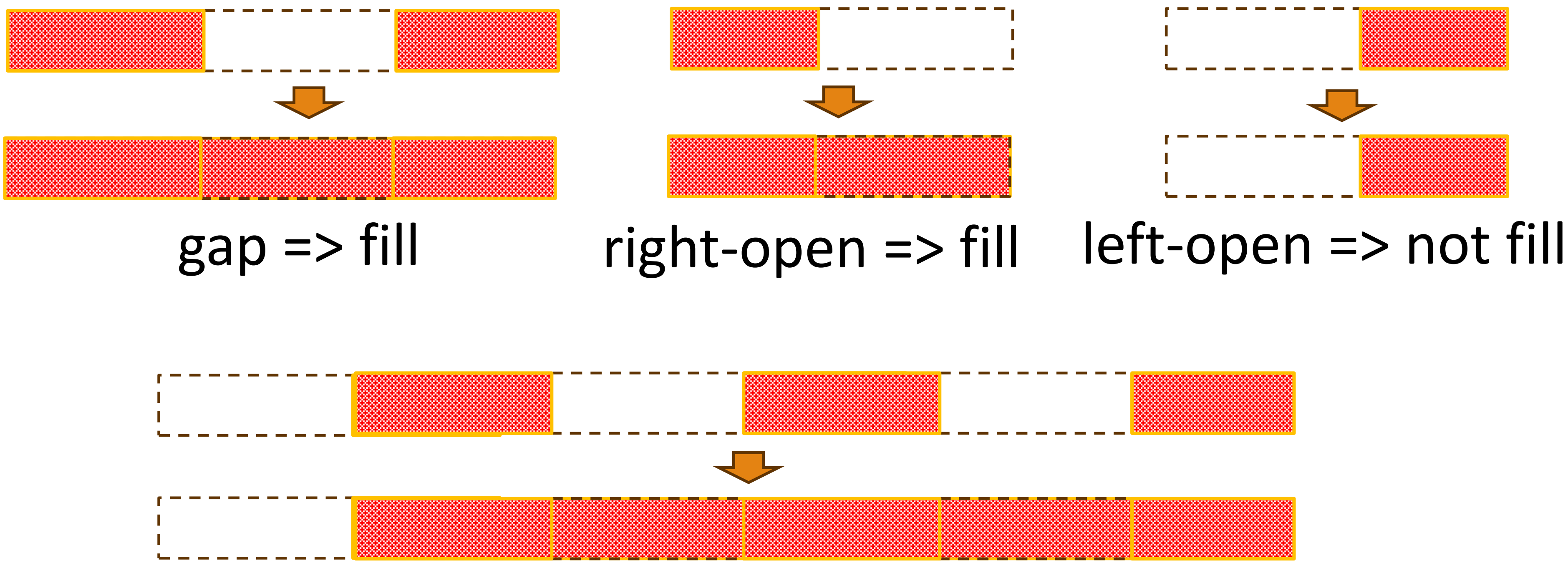


- double-open hidden chunk => must not be filled

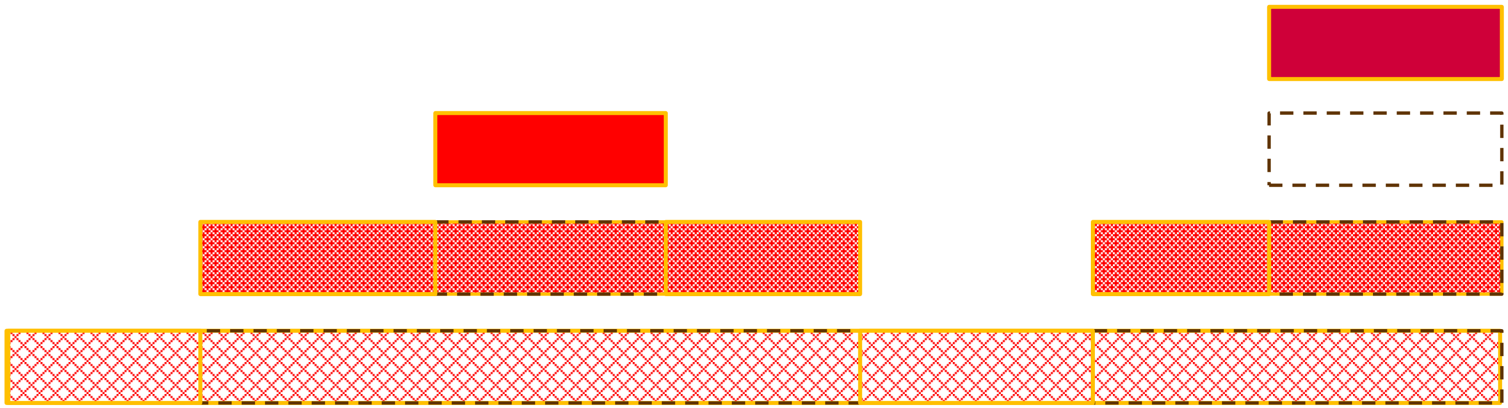
Typical submitted wrong answers filled this case

One of the Solutions

Fill hidden chunks that follow to surface chunks.



Resulting DPD

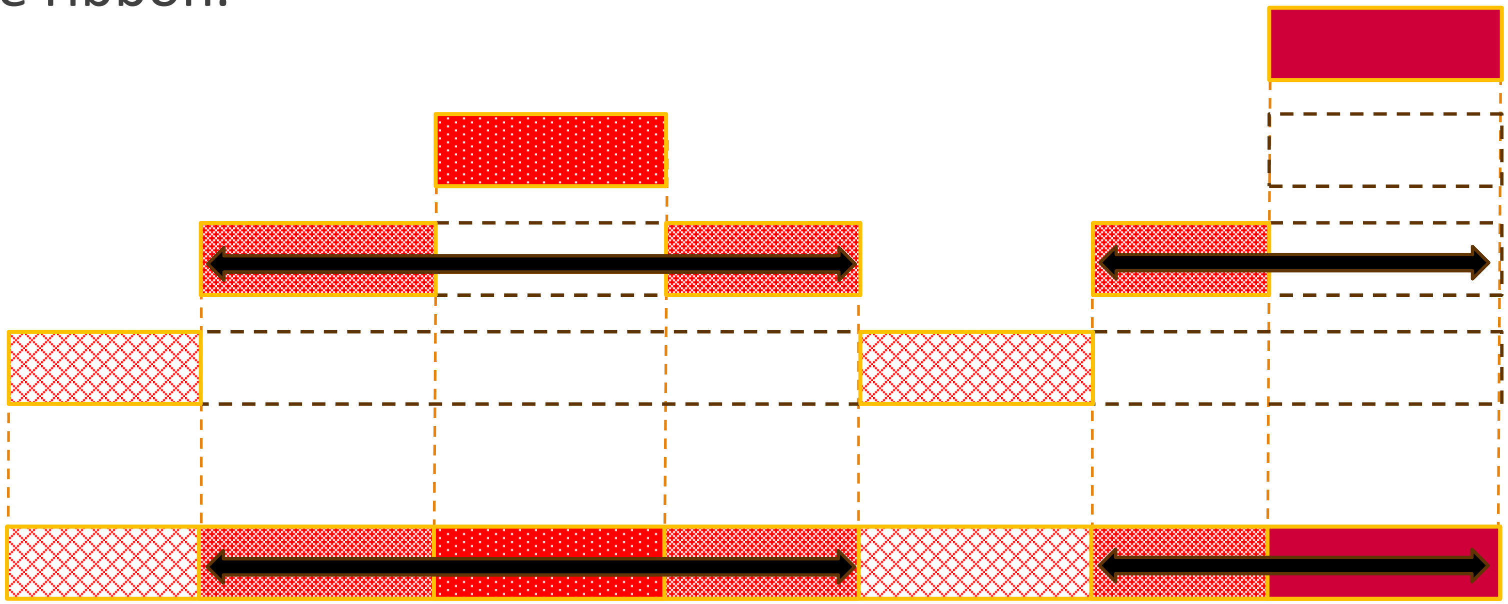


Algorithm

For each color, scan the planned pattern from left to right.

Start making a merged chunk when the color appear.

Stop merging when a lighter color appear or at the end of the ribbon.

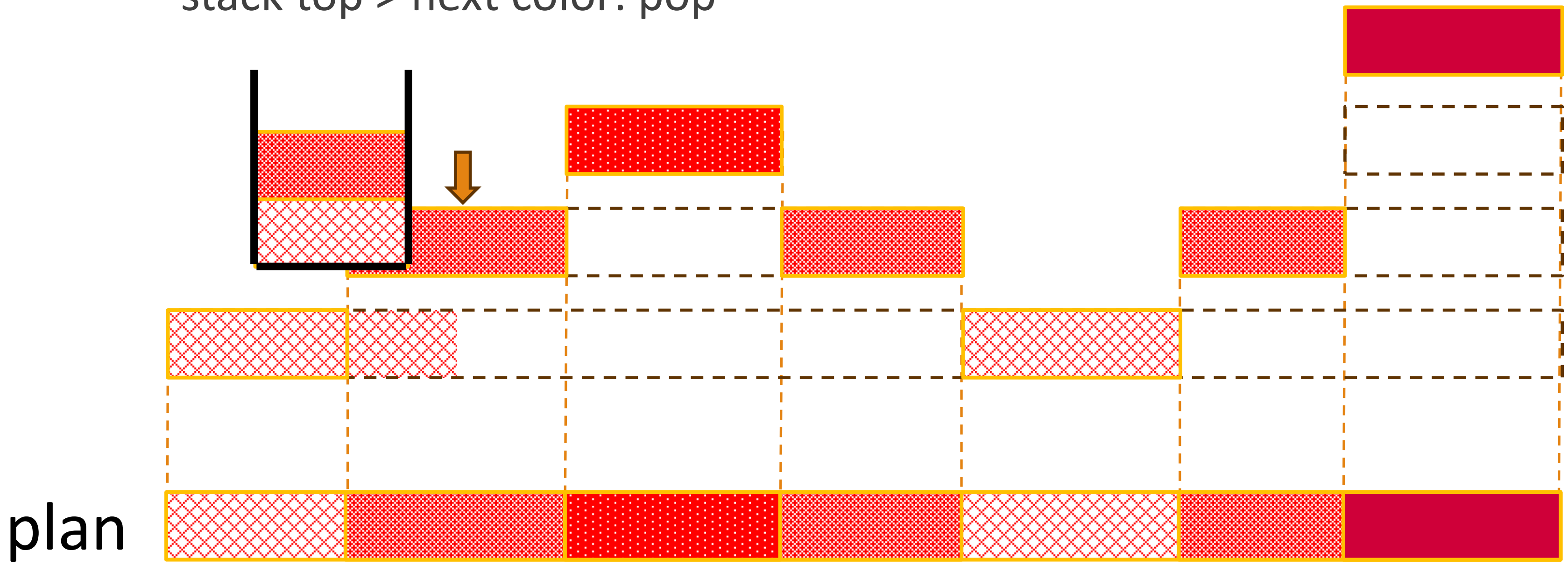


plan

O(n) Algorithm

Scan from left to right once while managing a stack of the colors of "merging layers"

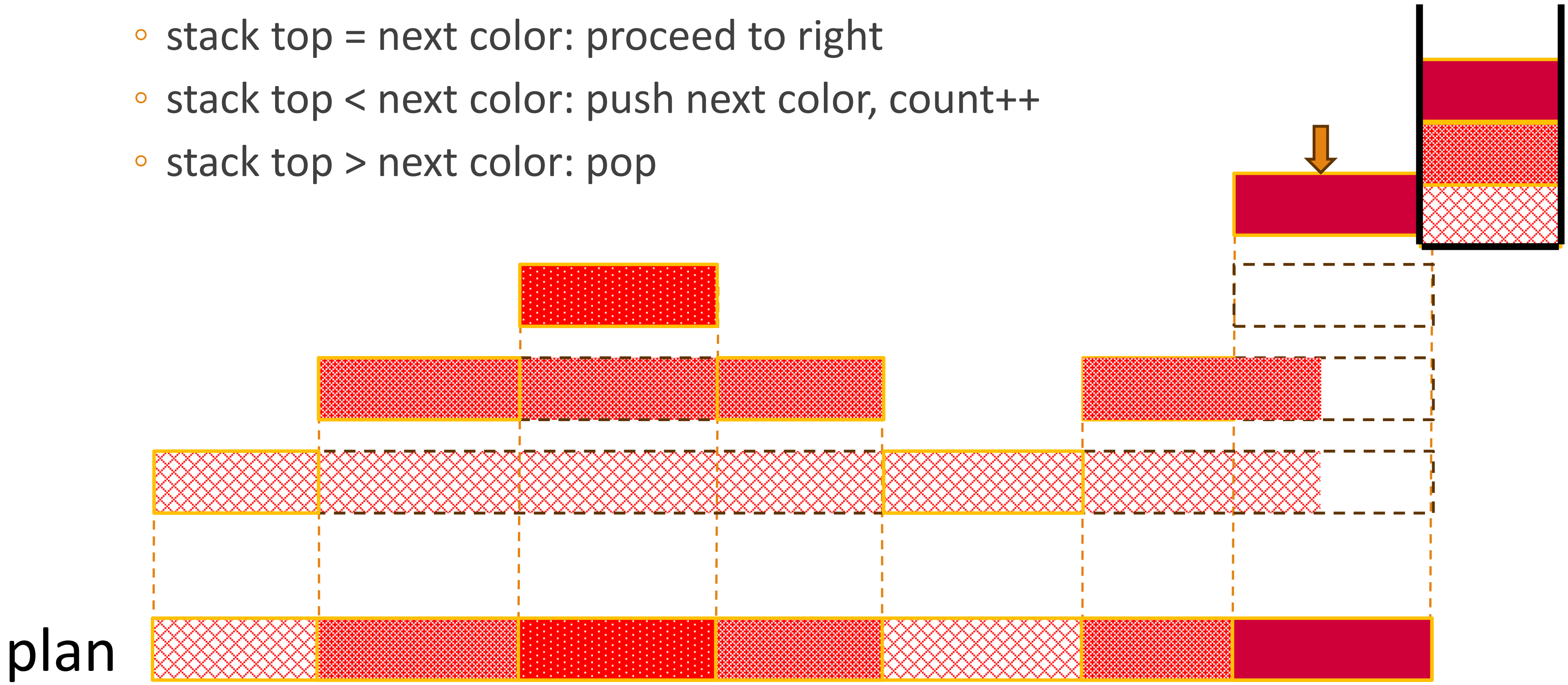
- stack top = next color: proceed to right
- stack top < next color: push next color, count++
- stack top > next color: pop



O(n) Algorithm

Scan from left to right once while managing a stack of the colors of “merging layers”

- stack top = next color: proceed to right
- stack top < next color: push next color, count++
- stack top > next color: pop



B: The Sparsest Number in Between

PROPOSER: ETSUYA SHIBAYAMA
AUTHOR: ETSUYA SHIBAYAMA

Problem Descriptions

Input: two positive integers a and b $(a \leq b)$

Smallest

Challenge: find the **sparsest** integer between a and b, inclusive

Definition: x is **sparser** than y if and only if the binary representation of x has a smaller number of 1's than that of y

Example

When 10 and 15 are given

Decimal	Binary	# of 1's
10	1010	**2**
11	1011	3
12	1100	**2**
13	1101	3
14	1110	3
15	1111	4

The Integers in Between

The Answer

Sparsest

Solution

Since a and b can be large (up to 10^{18}), a naïve search like the following does not work

```
for (long long i = a; i <= b; i++) {
  // do some work
}
```

Proper division of cases, like a mathematical proof, can help you

Division of Cases

Case 1: a is a power of two (2^n)

The answer is a itself

a's binary rep. has just a single 1, and thus sparsest and smallest

	Decimal	Binary
a	8	1000
answer	8	1000
b	19	10011

[Hereafter, we assume that a is not a power of two]

Division of Cases

Case 2a: the binary rep. of a is shorter than that of b

The answer is the smallest power of two greater than a

Suppose for instance that 14 and 33 are given

	Decimal	Binary
a	14	1110
answer	16	10000
b	33	100001

Obviously, $14 \leq 16(= 2^4) \leq 33$, and 16 is the smallest among the sparsest

Division of Cases

Case 2b: the binary reps. of a and b are of the same length

The binary rep. of the answer must share the same common prefix as a's and b's

The rest of the binary rep. of the answer can be found in a similar manner as case 1 or 2a

Unless $a = b$, the rest parts of the a's and b's binary reps. always start with 0 and 1, respectively.

	Decimal	Binary	
		Common prefix	Suffix
a	43	101	011
answer	44	101	100
b	47	101	111

How to deal with binary reps.

You may use bit operations

You may also first convert numbers to strings and then use string operations

C: Omnes Viae Yokohamam Ducunt?

PROPOSER: MASATOSHI KITAGAWA
AUTHOR: MASATOSHI KITAGAWA

Problem

Given a weighted undirected graph $G = (V, E)$.

- p_v: weight of a vertex v (significance value)
- q_e: weight of an edge e (vulnerability)
- $s \in G$: Yokohama

Minimize the cost (total risk severity) of spanning trees of G.

Problem(Cost)

For a spanning tree $T = (V, E_T)$ of G,

$$\text{cost of } T := \sum_{e \in E_T} q_e \sum_{v} p_v \, .$$

The second sum is taken over all $v \in V$ inaccessible from s in $T - \{e\}$.

Problem(Cost)

For a spanning tree $T = (V, E_T)$ of G,

$$\text{cost of } T := \sum_{e \in E_T} q_e \sum_{v} p_v \, .$$

The second sum is taken over all $v \in V$ inaccessible from s in $T - \{e\}$.

Minimum spanning tree problem?

Problem(Cost)

For a spanning tree $T = (V, E_T)$ of G,

$$\text{cost of } T := \sum_{e \in E_T} q_e \sum_{v} p_v \,.$$

The second sum is taken over all $v \in V$ inaccessible from s in $T - \{e\}$.

Minimum spanning tree problem?

No!

The cost of e depends on T.

Rewrite Cost

$$\text{cost of } T := \sum_{e \in E_T} q_e \sum_{v} p_v \,.$$

Swapping the two sums,

$$\text{cost of } T = \sum_{v \in V} p_v \sum_{e} q_e \,.$$

The second sum is taken over all e in the $s - v$ path in T.

Rewrite Cost

$$\text{cost of } T := \sum_{e \in E_T} q_e \sum_{v} p_v .$$

Swapping the two sums,

$$\text{cost of } T = \sum_{v \in V} p_v \sum_{e} q_e .$$

The second sum is taken over all e in the s—v path in T.

$$\text{cost of } T = \sum_{v \in V} p_v \text{ (distance from } s \text{ to } v \text{ in } T).$$

Solution

$$\text{cost of } T = \sum_{v \in V} p_v \, (\text{distance from } s \text{ to } v \text{ in } T)$$

This cost is minimized when T is a shortest-path tree rooted at s.

Solution

$$\text{cost of } T = \sum_{v \in V} p_v \,(\text{distance from } s \text{ to } v \text{ in } T)$$

This cost is minimized when T is a shortest-path tree rooted at s.

Dijkstra's algorithm!

D: Tree Generators

PROPOSER: MITSURU KUSUMOTO
AUTHOR: MITSURU KUSUMOTO

Parsing!!

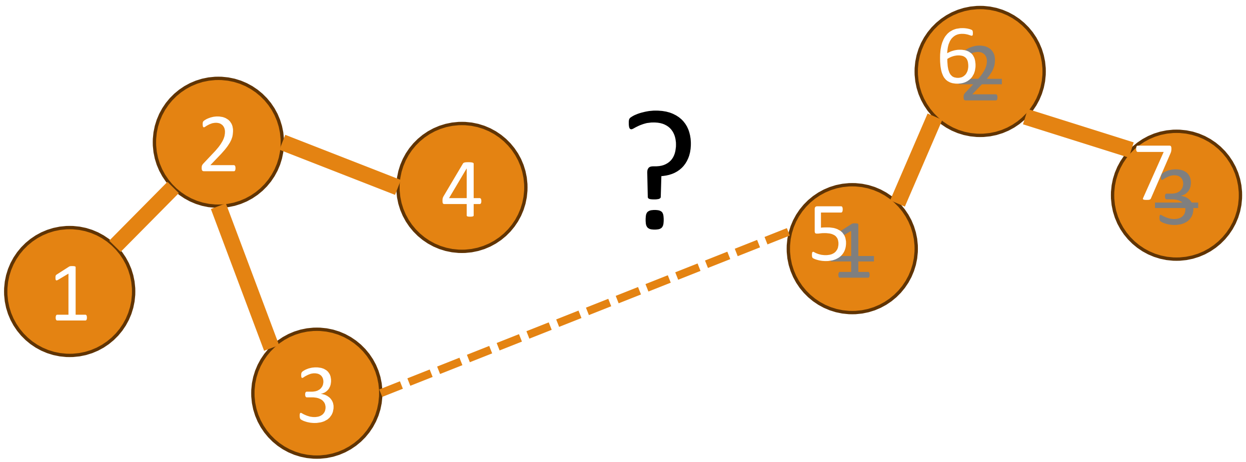
Syntax is

E ::= 1 | (E E)

‘1’ = Single vertex

It’s a tree

'(E_1 E_2)' = Add one edge **randomly** between two trees generated by E_1 & E_2



Generated from E_1

Generated from E_2
(labels are increased)

Input: Two expressions

Output: # of trees generated from them in common modulo 998244353

Solve in linear time.

Trees generated

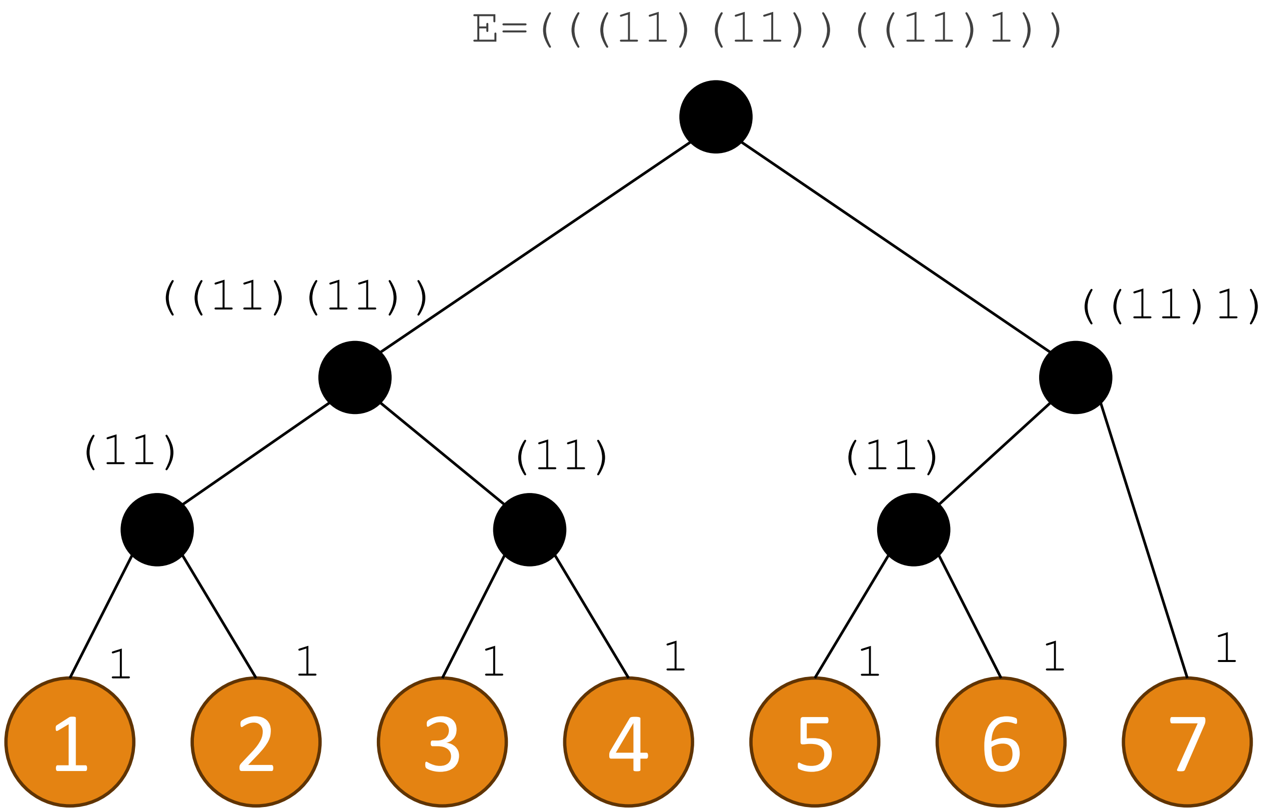
So, what kind of trees can be generated?

Assume that generated trees contains n vertices.

After parsing an expression, you can obtain triples (a_i, b_i, c_i) (i=1,...,n-1) such that

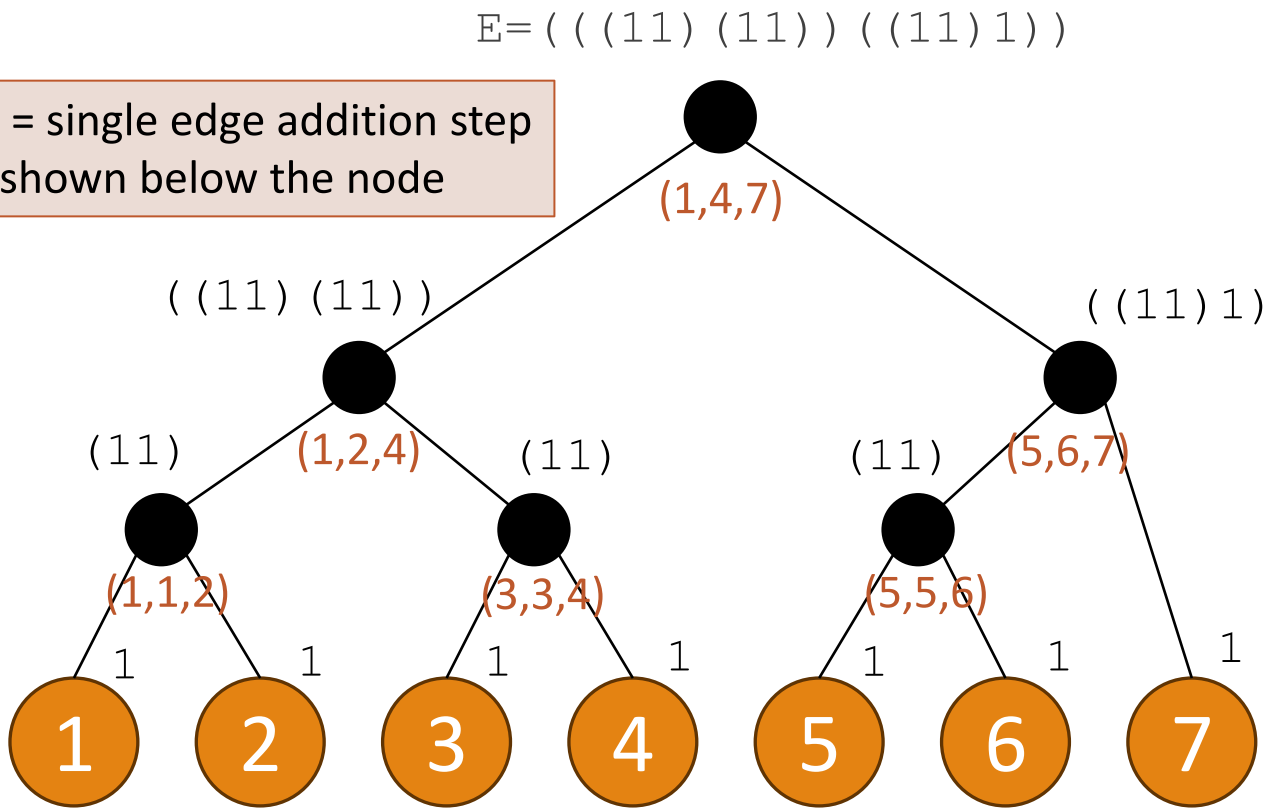
Each edge is randomly chosen from $[a_i, b_i] \times [b_i+1, c_i]$

Example (Sample 3)



Example (Sample 3)

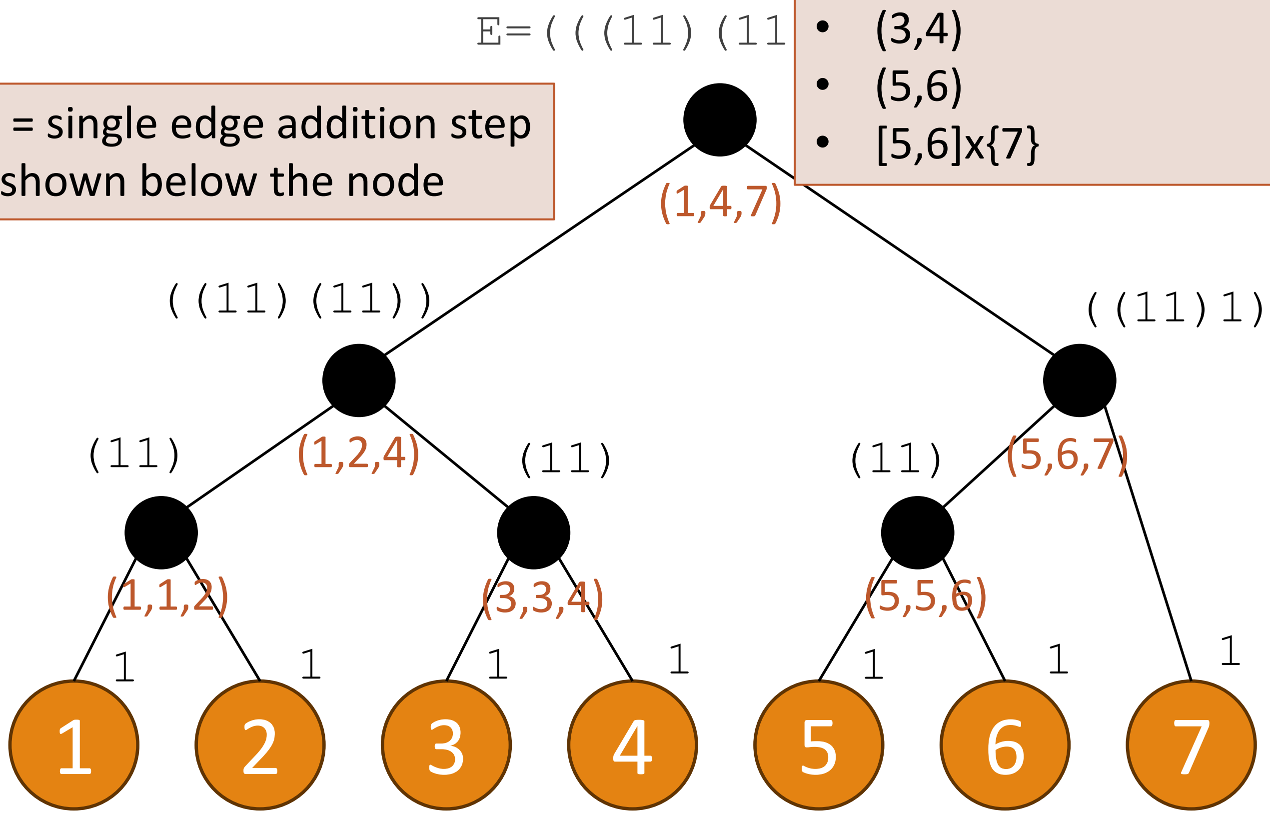
Black node = single edge addition step
Triples are shown below the node



Example (Sample 3)

Black node = single edge addition step
Triples are shown below the node

This represents adding edges from
- [1,4]x[5,7]
- [1,2]x[3,4]
- (1,2)
- (3,4)
- (5,6)
- [5,6]x{7}



Trees generated (2)

So, what kind of trees can be generated?

Assume that generated trees contains n vertices.

After parsing an expression, you can obtain ~~triples (a_i, b_i, c_i)~~ (i=1,…,n-1) such that

Each edge is randomly chosen from $[a_i, b_i] \times [b_i+1, c_i]$

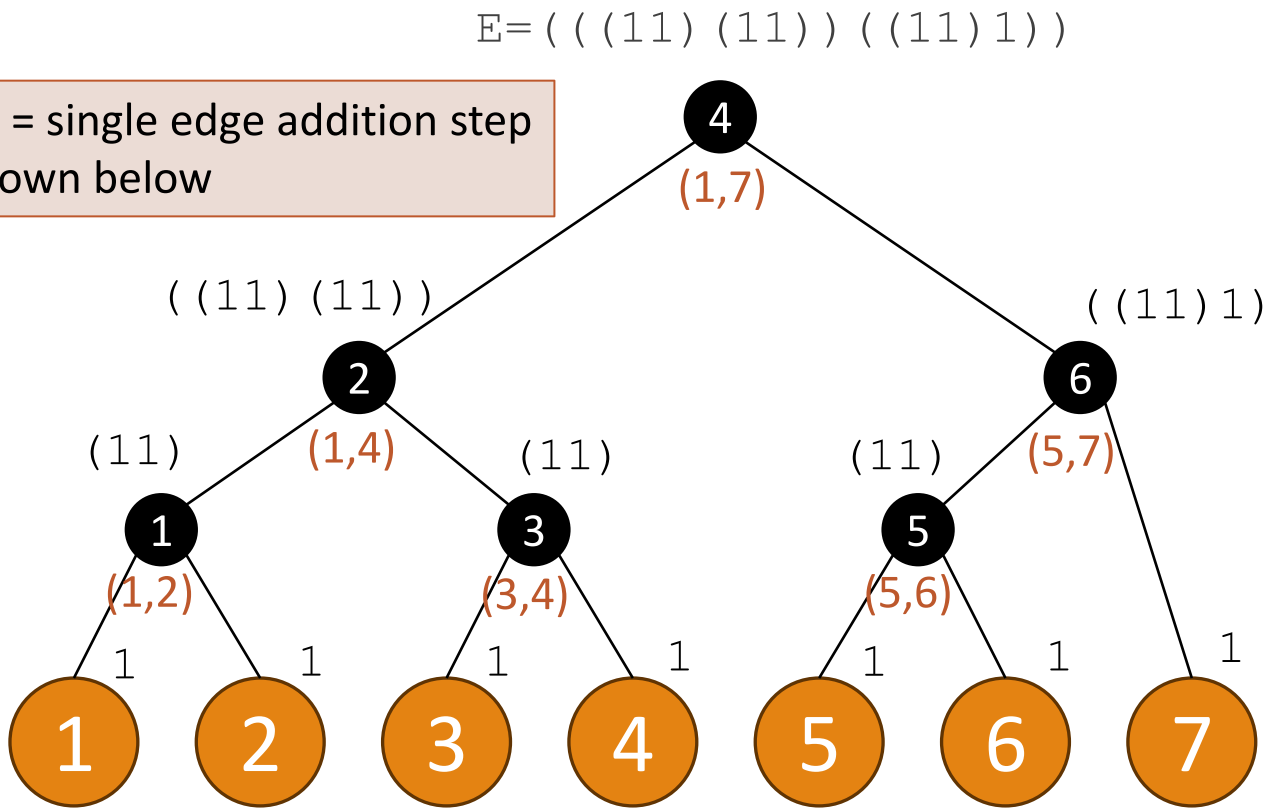
Values b_i appears <u>just once</u> in the triples; Let's simplify this:

Each edge is randomly chosen from $[a_i, i] \times [i+1, c_i]$

This is like, **the gap between *i* and *i*+1 is generating an edge.**

Example revised

Black node = single edge addition step
(a_i, c_i) is shown below



Solution

Suppose that pairs (a'_i, c'_i) are obtained from the other expression.

Then, the solution is

$$\prod_{i=1}^{n-1}(i - \max(a_i, a'_i) + 1) \times (\min(c_i, c'_i) - i).$$

The remaining part is the proof for this.

Correspondence (1)

Suppose that a tree is generated from E.
For each edge, can we identify which gap generated it?

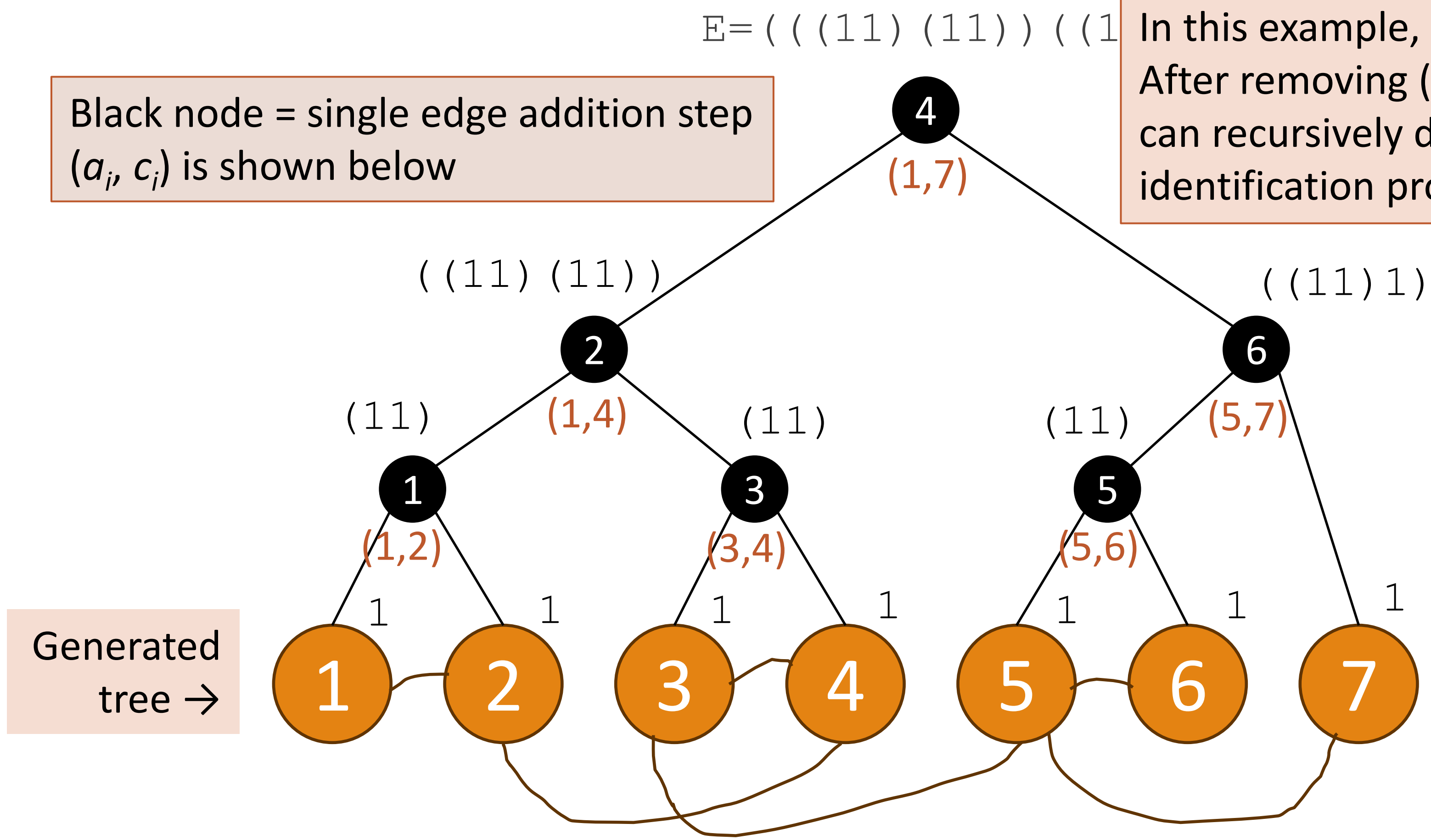
Answer: We can uniquely identify.

Why? Traversing the generation steps in the parsed tree from top to bottom will give one-to-one correspondence between edges and generations.

Example revised (2)

There can be **only one** edge between [1,4]x[5,7]. In this example, it's (3,5). After removing (3,5), we can recursively do this identification process.

Black node = single edge addition step (a_i, c_i) is shown below



Correspondence (2)

Now, suppose that a tree is generated from both E_1 and E_2.

For an edge (j, k), if

(j, k) is generated from the gap between *i* and *i*+1 in E_1, and
(j, k) is generated from the gap between *i'* and *i'*+1 in E_2,

then, we denote as $\pi(i) = i'$.

The mapping π is bijective. We can show that π must be **an identity function**. This justifies the solution mentioned.

Proof by infinite decent

Suppose that π is not an identity. This means there exists a pair $i \neq j$ s.t. $j=\pi(i)$.

From some observation, $a_i \leq j < c_i$.

For $k=1,\ldots,n-1$, let $f(k) = c_k - a_k$.

Then, $f(i) = c_i - a_i > f(j)$ holds. If we continue this, we have

$$f(i) > f(j) > f(j') > f(j') > \ldots \quad \text{for } j'=\pi(j),\ j''=\pi(j'),\ \ldots$$

However, since π is bijective, this deduction eventually results in $\boldsymbol{f(i) > f(i)}$. This is contradiction. Thus, π is identity.

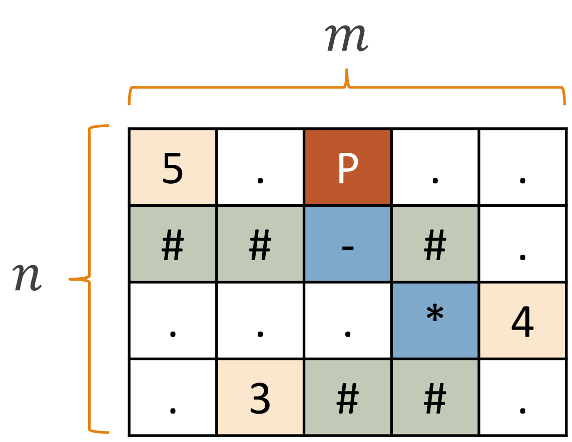
E: E-Circuit Is Now on Sale!

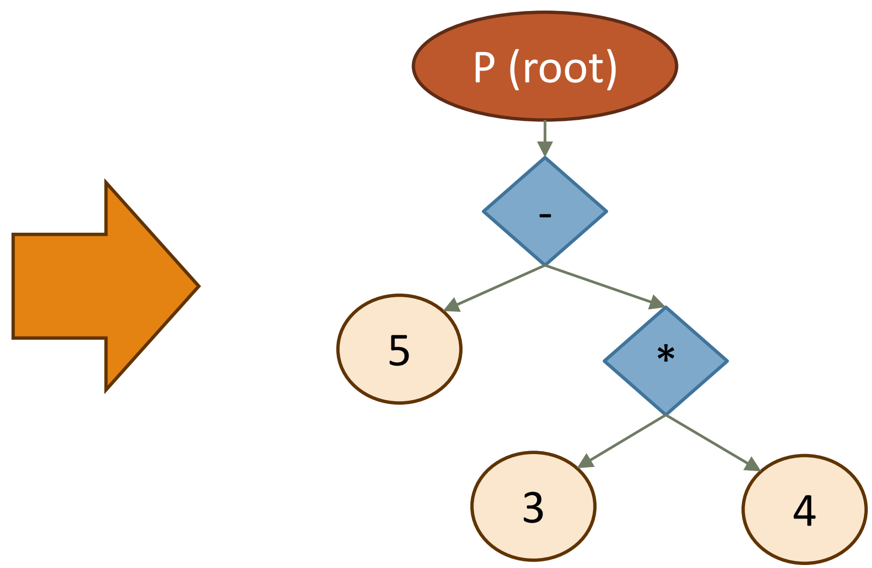
PROPOSER: SHINYA SHIROSHITA
AUTHOR: SHINYA SHIROSHITA

Problem

You are given a tree of a mathematical formula ***embedded in a grid space***.

Your task is to calculate the result of the formula.





$$1 \leq n, m \leq 50.$$

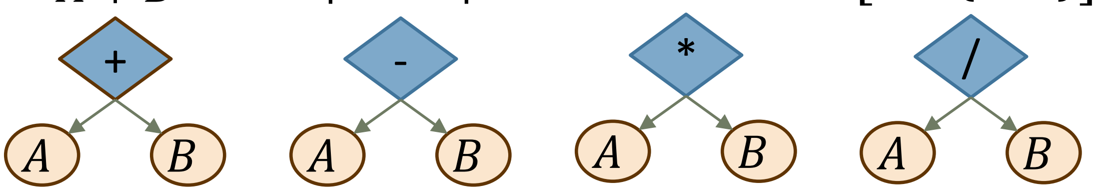
Node types

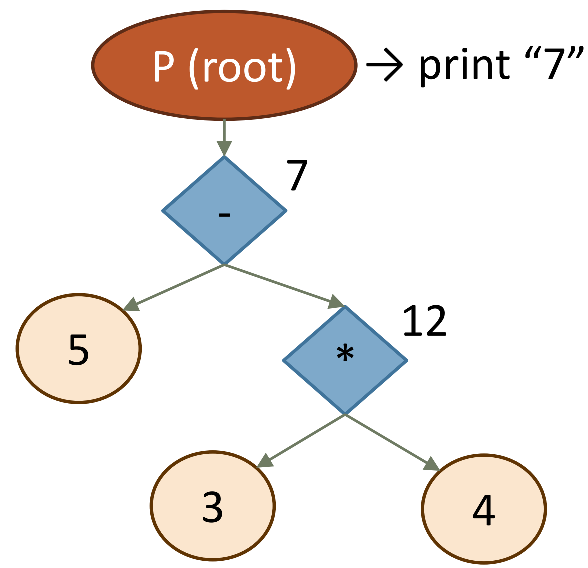
Following nodes are provided.

- Printer (P) : the root node.
- Digit (0-9) : a leaf node with a value.
- Operator (+-*/) : a node applying an arithmetic operation.

(“#” forms edges connecting nodes.)

$A + B$ $\quad |A - B|$ $\quad A \times B$ $\quad \left\lfloor \frac{\max\{A,B\}}{\min\{A,B\}} \right\rfloor$





Solution

Traverse the tree from the printer recursively.

- For an operator cell,
  - Traverse a subtree of one connection and memorize the result.
  - Traverse the other connection and apply the operator's calculation.

Be careful about careless mistakes!

Overflow, out of range, infinite loop, …

It is wasteful to get penalties by careless mistakes.

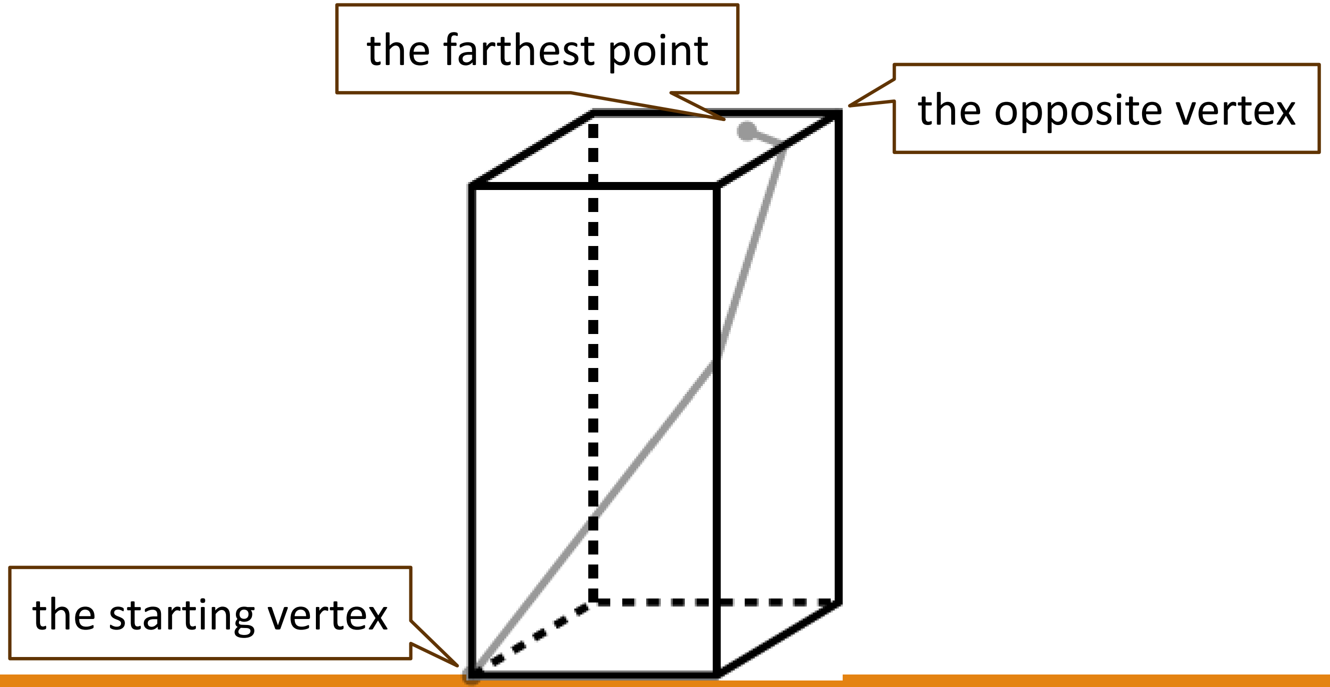
F: The Farthest Point

PROPOSER: F.YAMAGUCHI
AUTHOR: F.YAMAGUCHI

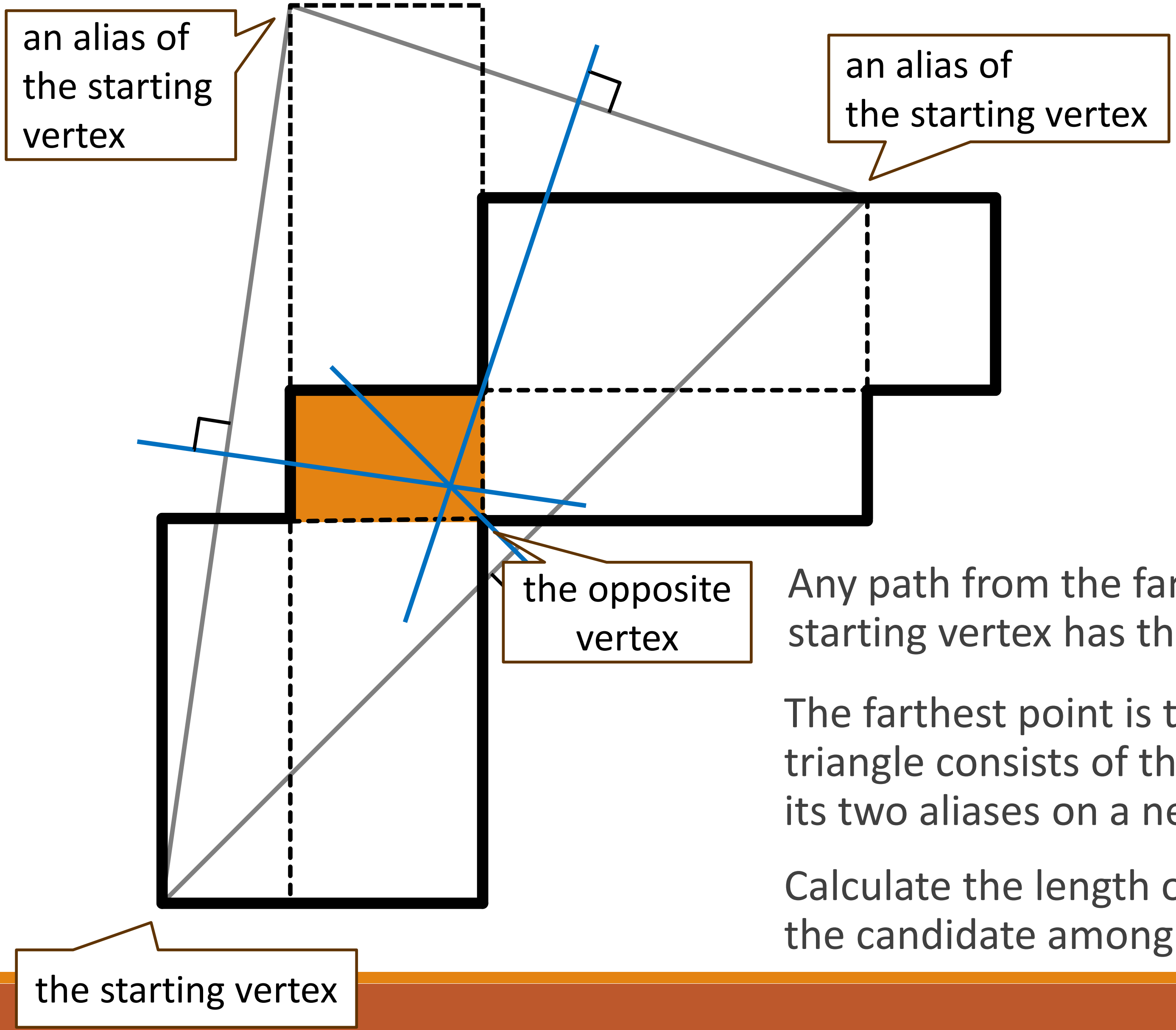
Problem

Given: the size (edge lengths) of a rectangular cuboid

Write a program which computes the distance from a vertex to its farthest point on the surface of the cuboid.



Core Idea



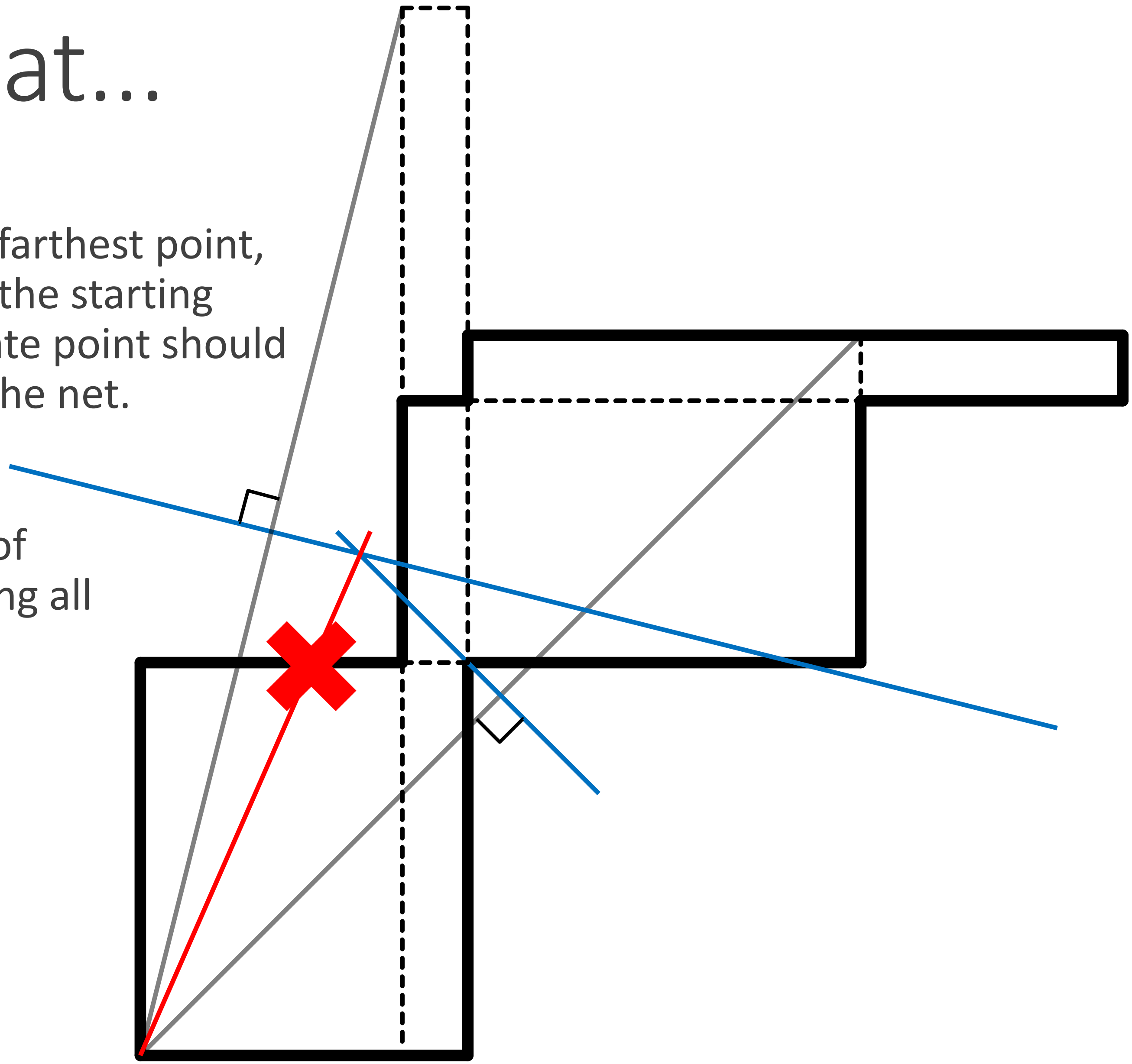
Any path from the farthest point to the starting vertex has the same distance.

The farthest point is the circumcenter of the triangle consists of the starting vertex and its two aliases on a net.

Calculate the length of the longest path to the candidate among sufficient net settings.

Note that…

For a candidate of the farthest point, the segment between the starting vertex and the candidate point should not cross any edge of the net.

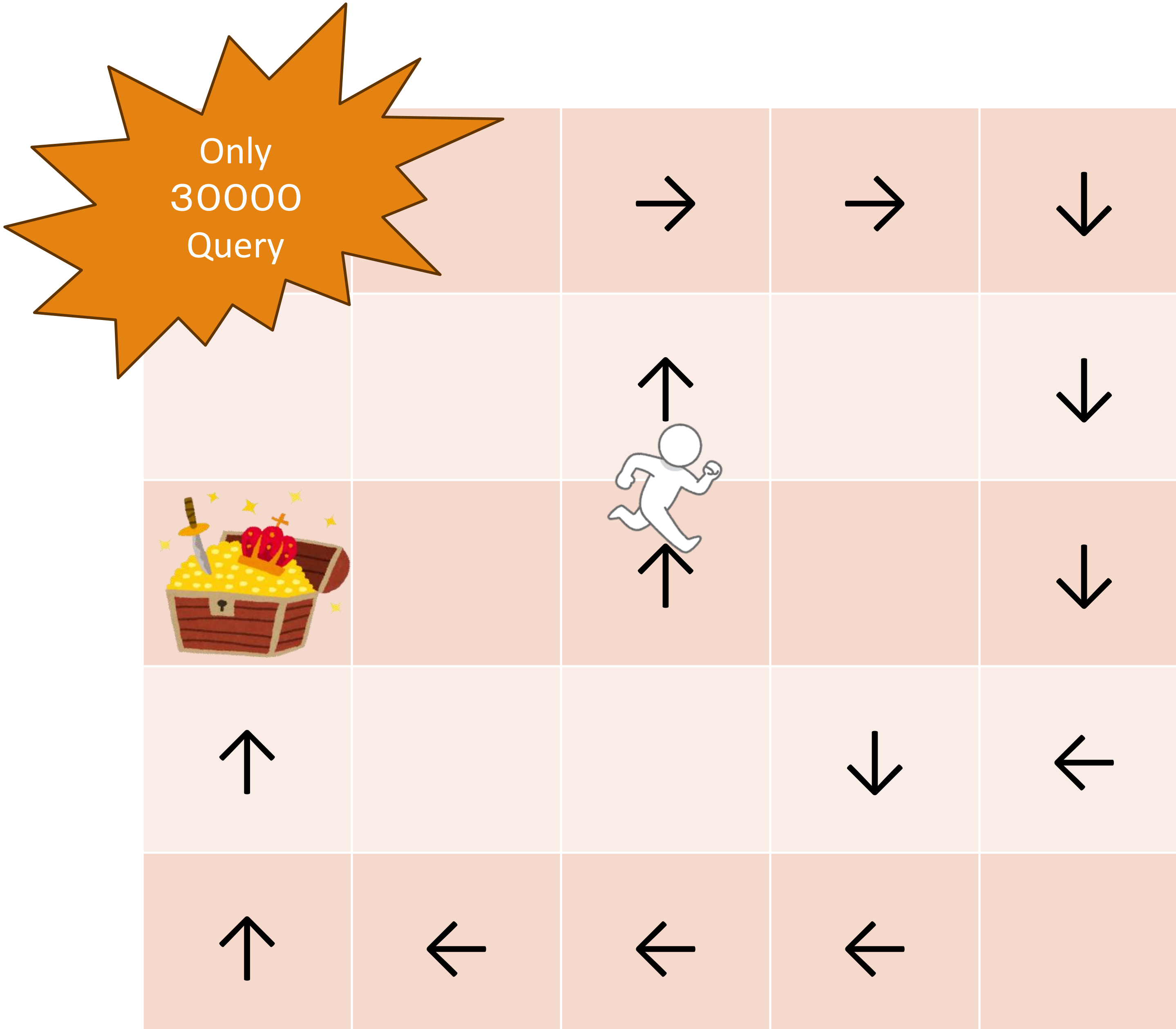


Distance is the length of the shortest path among all possible paths.

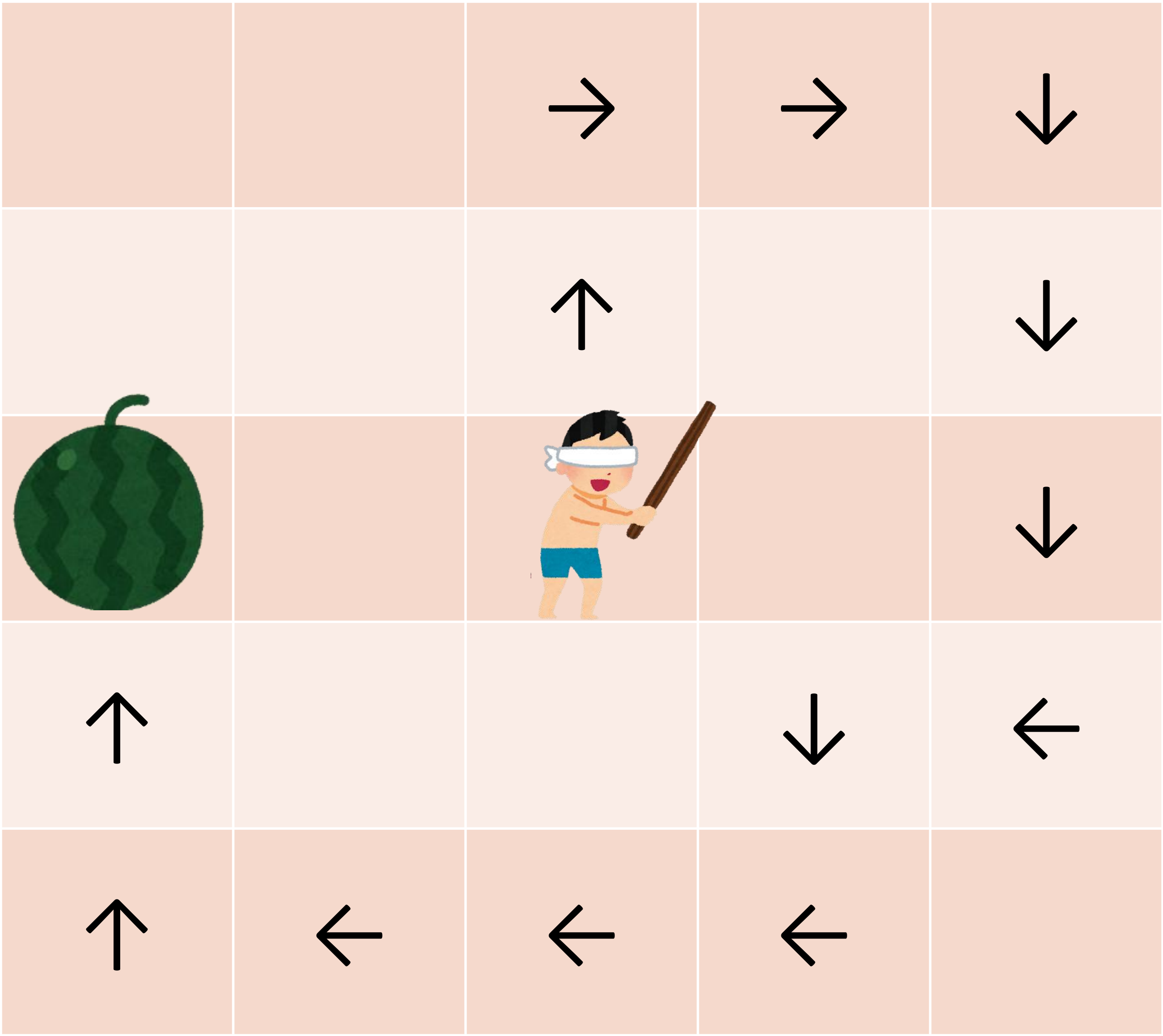
The distance from the starting point is not a convex function.

G: Beyond the Former Exploer

PROPOSER: KOHEI MORITA
AUTHOR: KOHEI MORITA
+ MITSURU KUSUMOTO


Only
30000
Query

Original idea



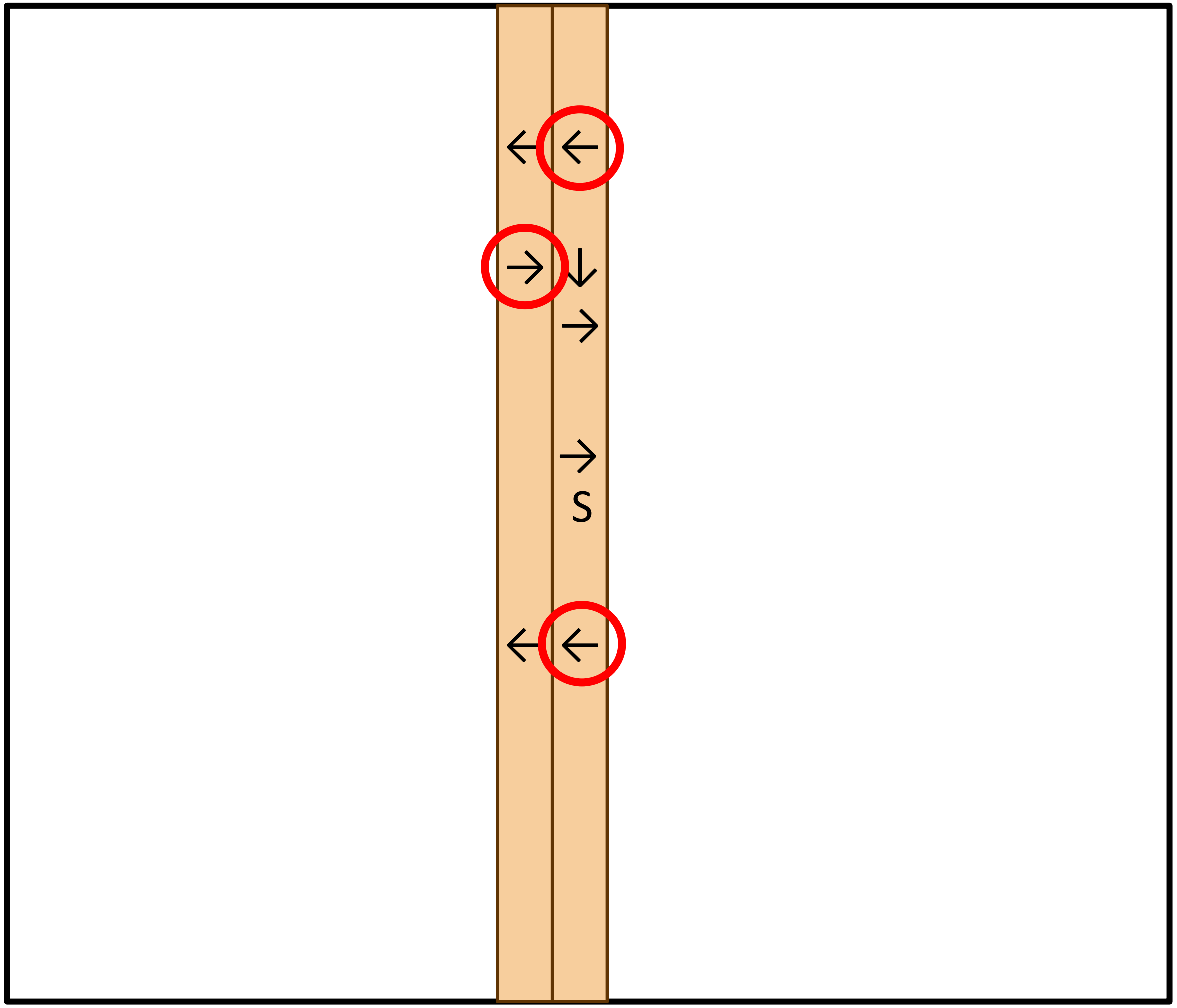
Solution

Interactive Problem

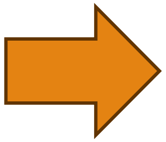
Solution

Interactive Problem

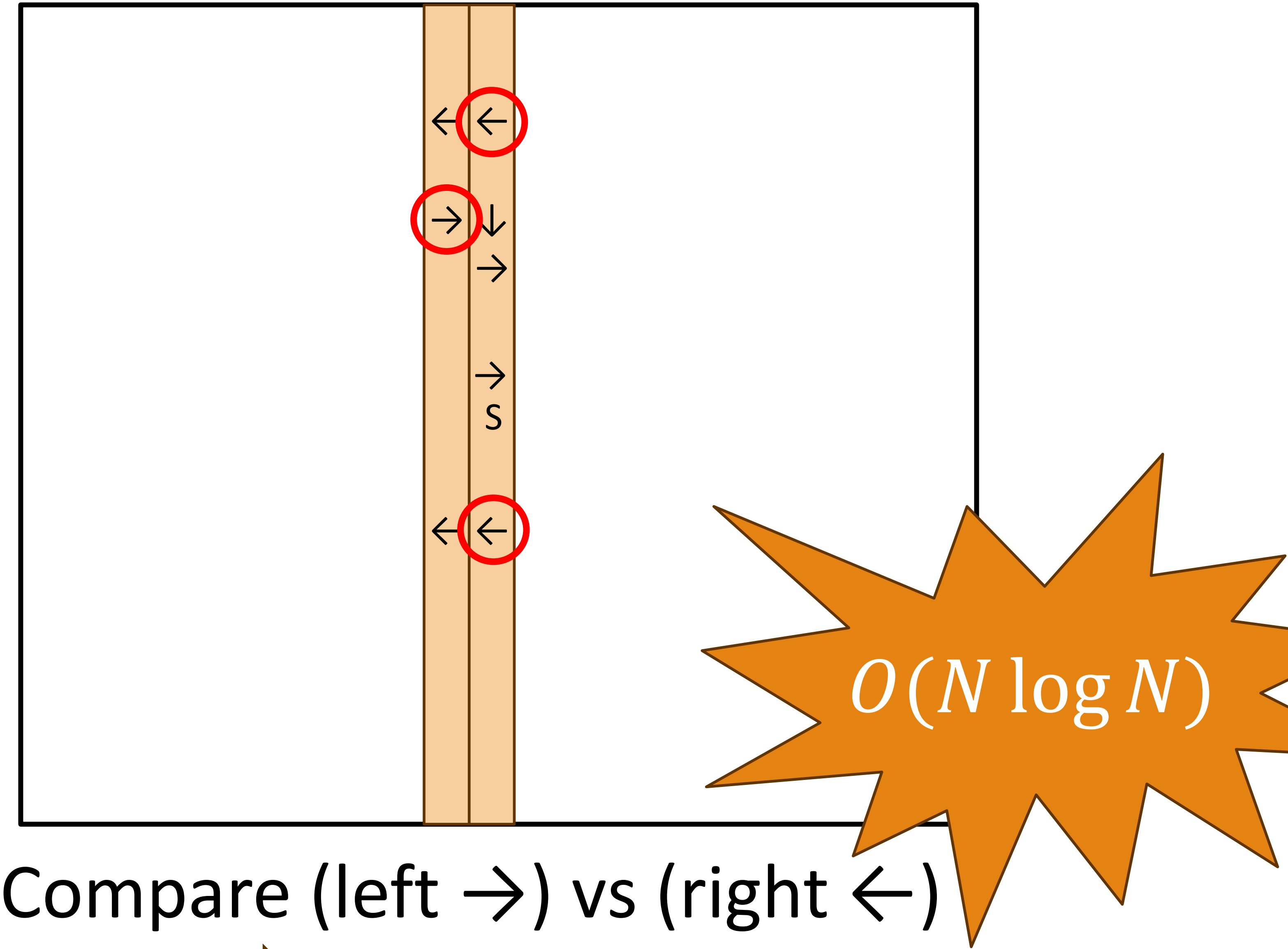
As usual:
Binary Search



Compare (left →) vs (right ←)



Left region has G(goal)



Compare (left →) vs (right ←)

Left region has G(goal)

AC .... ?

$O(N \log N)$ =

$4N = 8000$ x $\log N = 11$ =

88000 queries

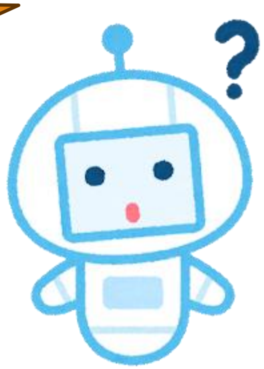
AC .... ?

$O(N \log N)$ =

$4N = 8000$ x $\log N = 11$ =

88000 queries

Only
30000
Query



AC .... ?

$O($ ... $N)$

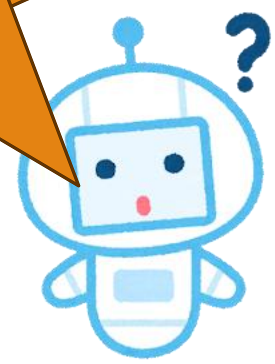
=

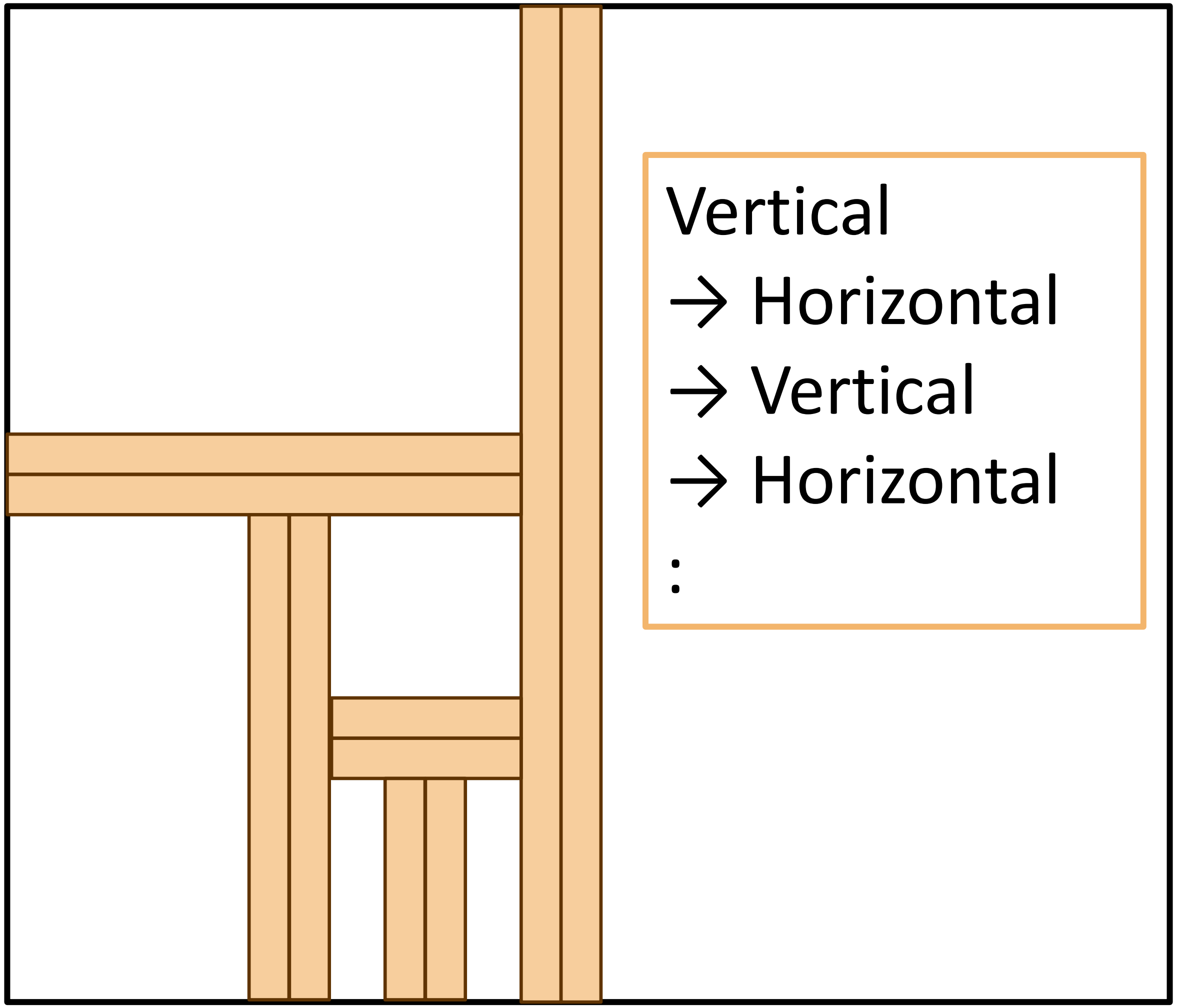
88000

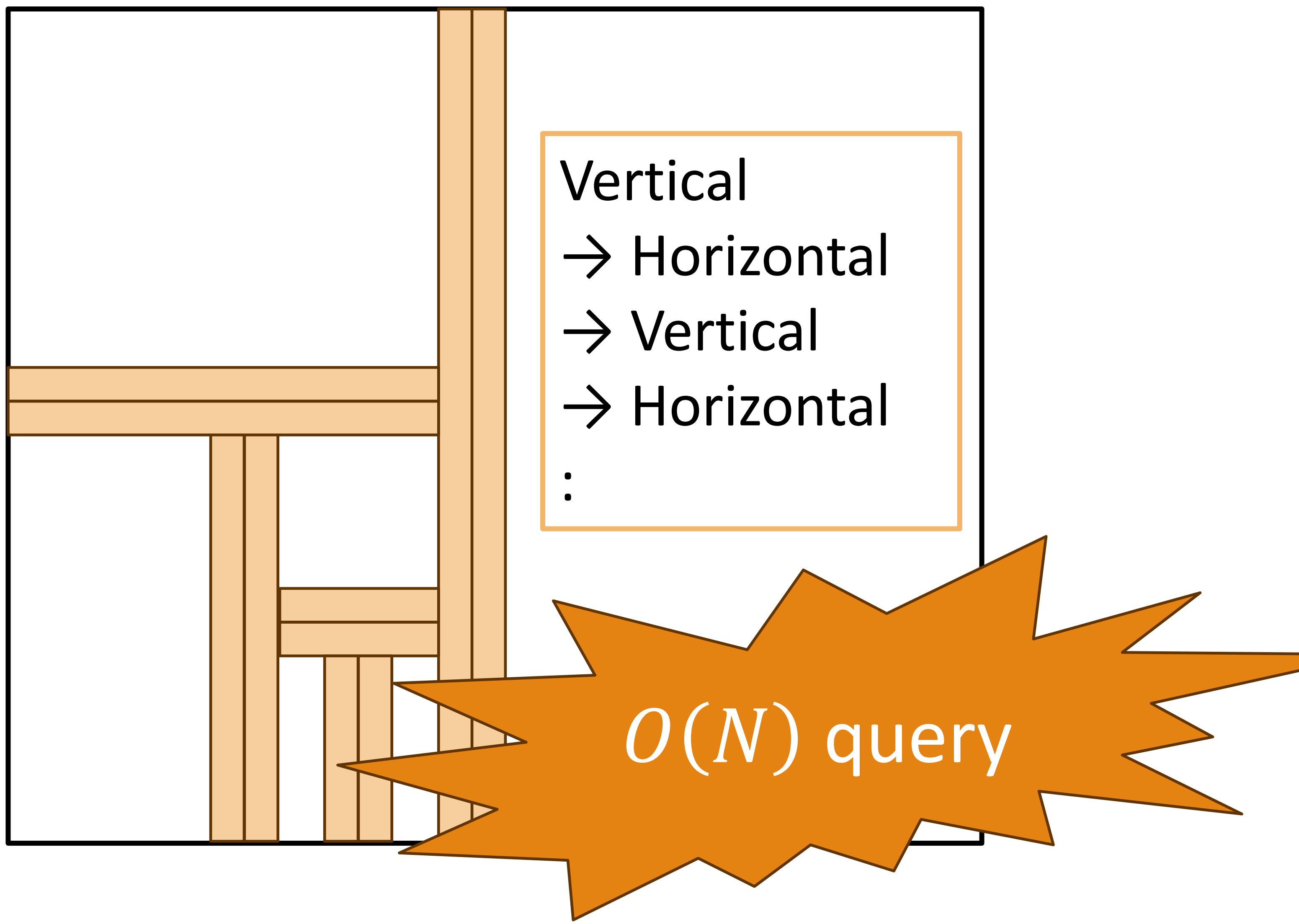
$O(N \log N)$ is insufficient

00

ry




Vertical
→ Horizontal
→ Vertical
→ Horizontal
:


Vertical
→ Horizontal
→ Vertical
→ Horizontal
:
$O(N)$ query

Full editorial(1/3)

The core idea is that for a continuous region with a grid, it is possible to determine whether the goal (G) is included in the region without examining every cell. To know this, it is sufficient to count the number of times the path "enters" and "exits" the region. This can be done by knowing only the boundary parts, that is, the cells in the region that touch the outside and the cells outside that touch the region.

Based on this consideration, for example, by examining all the cells in the i-th and $i+1$-th columns, it is possible to determine whether the goal is on the left or right.

Therefore, by performing a binary search on the range of columns where the goal might exist, it is possible to identify the goal with $O(N \log N)$ queries. However, since it is necessary to examine $4N$ cells in one step of the search, this approach is hard to get accept in terms of the number of queries.

Full editorial(2/3)

A further improvement is to reduce the number of queries by searching in the order of (split the region vertical) → horizontal → vertical → horizontal → ... like a KD-tree. The logic is as follows:

- First, examine the central 2 columns ($4N$ cells) to determine whether the goal is on the left or right.

- Examine the central 2 rows ($2N$ cells) to determine whether the goal is up or down. This will make the region where the goal might exist a square of $N \times N$ (the size will vary by ± 1 depending on which of the four sides is chosen).

- Examine the central 2 columns ($2N$ cells) to determine whether the goal is on the left or right.

- Examine the central 2 rows (N cells) to determine whether the goal is up or down.

- ... and repeat this search.

Since the number of cells required for the search is halved every 2 steps, the order of the number of queries for the entire search improves to $\mathrm{O}(N)$. Specifically, estimating the constants, the number of cells required for the search is $4N + 2N + 2N + N + N + \cdots = 12N$ cells, so there is enough margin against the query limit of 30000.

Full editorial(3/3)

The next consideration is how to move. In fact, there is a solution that requires only $O(1)$ extra moves per step, that is, a total of $12N + O(\log N)$ queries (Bonus), but it is assumed to be complicated to implement.

Since there is a margin of about $3N$ queries, it is desired to simplify the implementation as appropriate.

Various approaches can be considered, but one example is to assume that "the starting point of each step is at the center of the region." In other words,

- First, examine the central 2 columns ($4N$ cells) to determine whether the goal is on the left or right.
    - Move to the center of the new region with $N/2$ queries.

- Examine the central 2 rows ($2N$ cells) to determine whether the goal is up or down.
    - Move to the center of the new region with $N/2$ queries.

- Examine the central 2 columns ($2N$ cells) to determine whether the goal is on the left or right.
    - Move to the center of the new region with $N/4$ queries.

- Examine the central 2 rows (N cells) to determine whether the goal is up or down.
    - Move to the center of the new region with $N/4$ queries.

- ... and repeat this search.

With this approach, the extra moves increase by about $2N$ cells, but this is within the acceptable range.

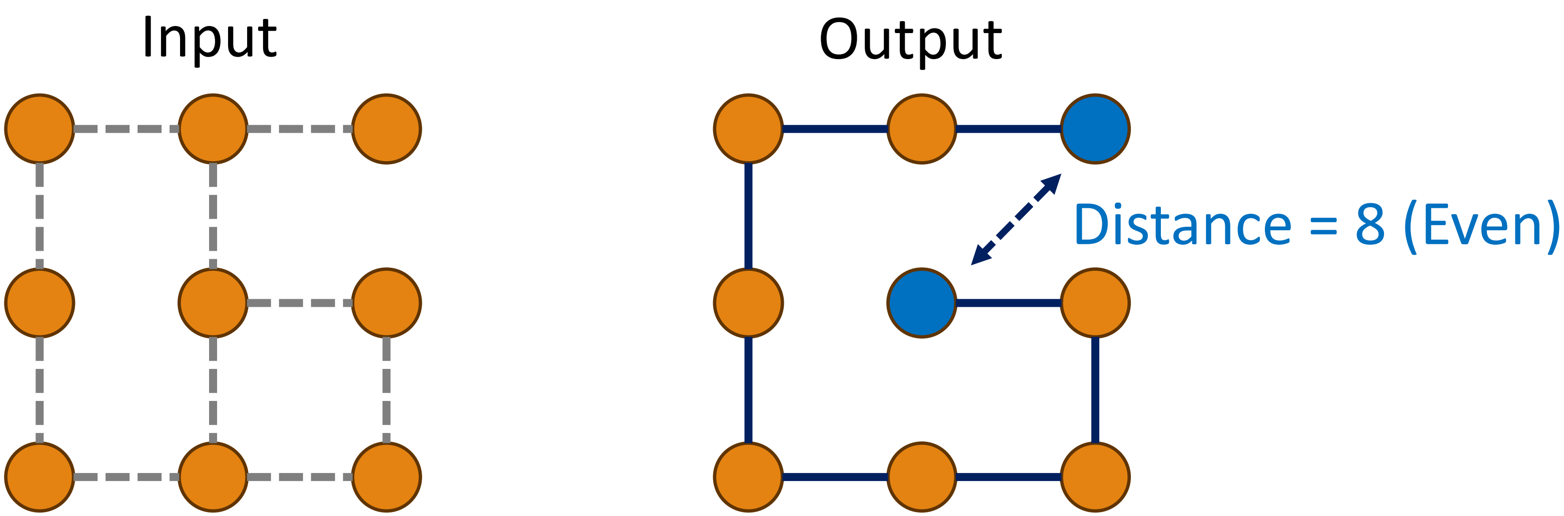
H: Remodeling the Dungeon 2

PROPOSER: YUTARO YAMAGUCHI
AUTHOR: RYOTARO SATO

Problem Statement

Given connected graph on $h \times w$ 2D grid ($h, w \leq 400$), find subset of edges (or report it is impossible) such that:

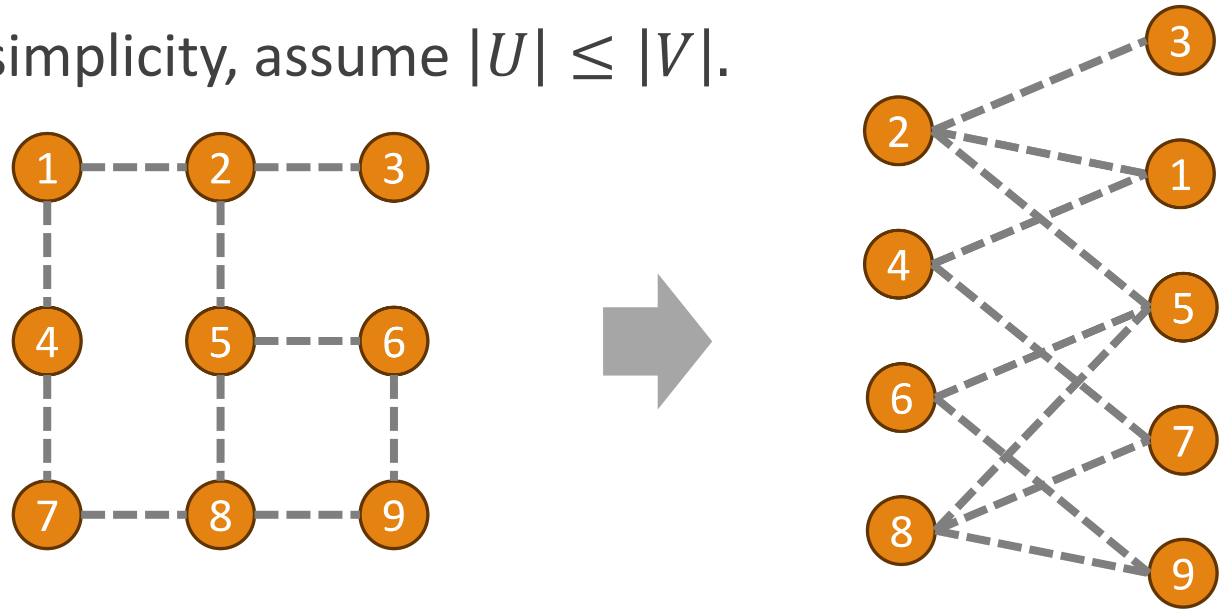
- All vertices and selected edges form a **tree**.
- Distances between all pair of leaves are **even**.



Grid Graph Is Bipartite

Let input graph G be represented as (U, V, E).

For simplicity, assume $|U| \leq |V|$.

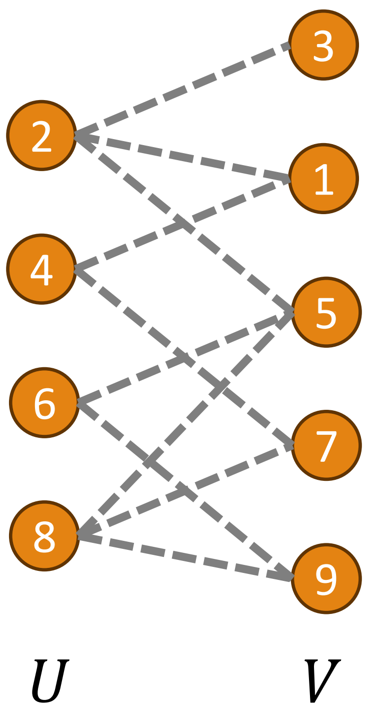


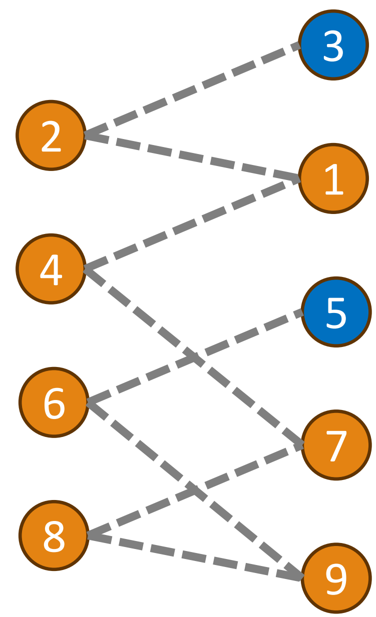
$$U = \{2, 4, 6, 8\}$$
$$V = \{3, 1, 5, 7, 9\}$$
$$E = \{(\text{all edges})\}$$

Rephrased Constraint

Distances between all pairs of leaves are <u>**even**</u>.

$\Leftrightarrow$ Either U or V does not contain any leaves.





U has no leaves

Cases of $|U| = |V|$

When $|U| = |V|$, **always infeasible**!

∵) Suppose $G = (U, V, E)$ has no leaves in U and satisfies $|U| = |V|$. Clearly, $|E| \geq 2|U| = |U| + |V|$, which implies G' contains cycle(s). ■

Hereafter, we assume $|U| < |V|$ and U has no leaves.

Updated Problem Statement

Given $G = (U, V, E)$ $(|U| < |V|)$, find $E' \subset E$ such that:

- Each vertex in U has **two (or more) adjacent edges** in E'
- (U, V, E') is a **tree** graph

Then, how to assign two edges for each vertex in U without making any cycles?

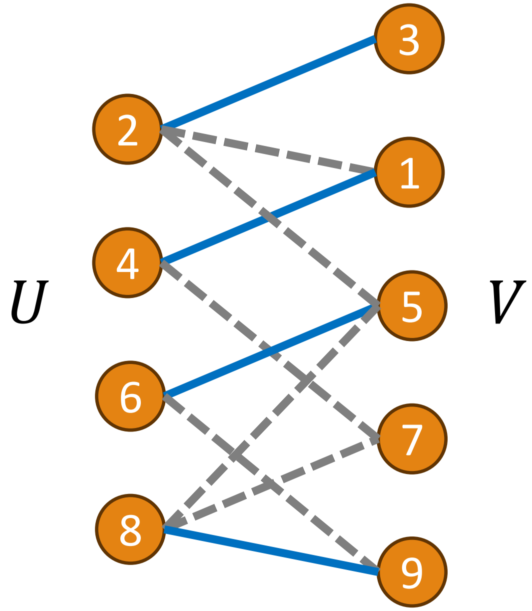
Note: This is typical matroid intersection problem, but naive implementations of general MI instances are too slow for prepared testcases!

Step 1. Maximum Matching

First, find maximum matching of (U, V, E).

Can be done by Hopcroft–Karp algorithm, $O(|E|^{1.5})$.

If size of matching $< |U|$, infeasible!

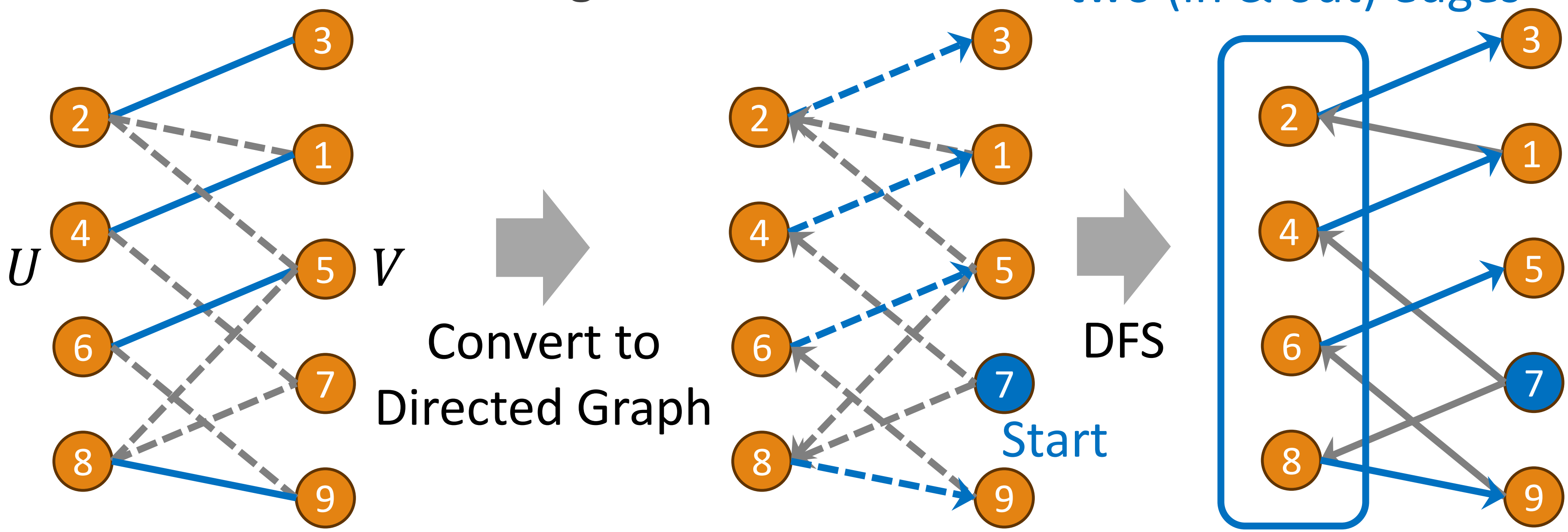


Step 2. DFS

Run DFS on the **directed** graph, from all vertices in V NOT used in matching. Edges' directions:

- NOT used in matching: V-to-U
- USED in matching: U-to-V

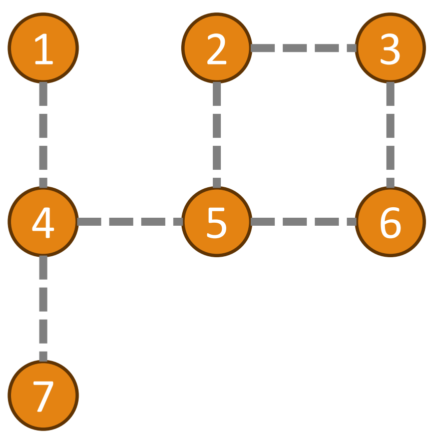
Each vertex has two (in & out) edges

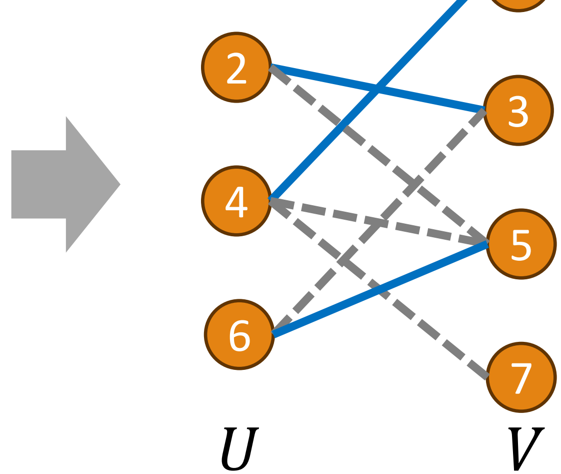


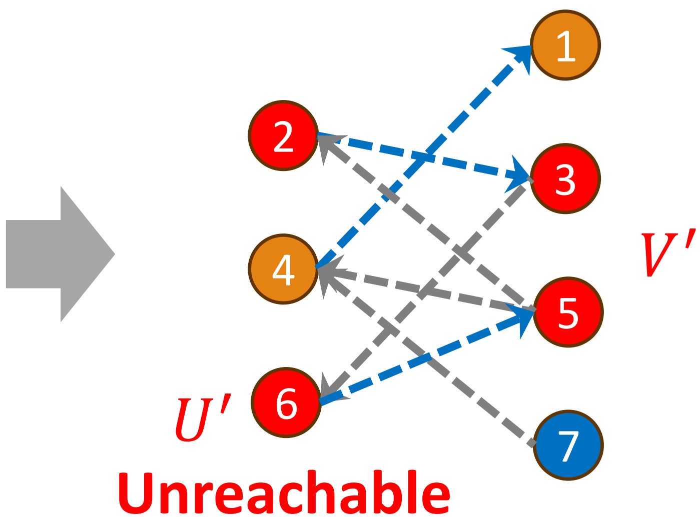
Infeasible Cases

If some vertices are unreachable by DFS, infeasible!

∵) $U' \subset U$: set of all unreachable vertices.
$V' \subset V$: set of all vertices adjacent to any vertex in U'.
We can prove $|V'| \leq |U'|$ (Exercise), which means it is impossible to choose $2|U'|$ edges adjacent to U' without making any cycle. ■







Step 3. Make Graph Connected

We obtained the DFS forest that satisfies degree constraints.

Finally, do not forget to adopt additional edges to make graph connected!

Overall complexity: $O\left((hw)^{1.5}\right)$ (bounded by finding max matching).

I: Greatest of the Greatest Common Divisors

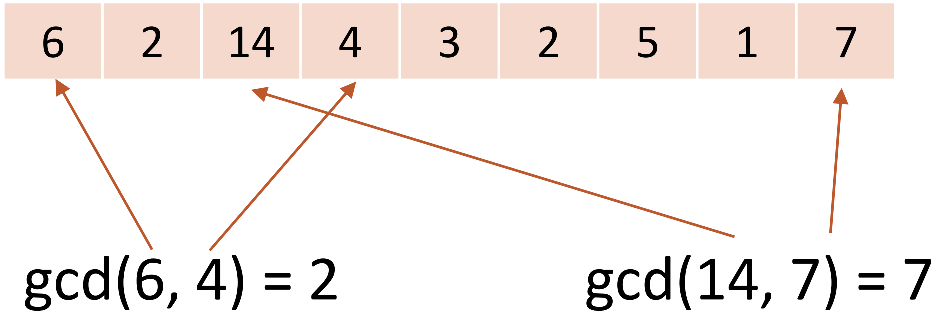
PROPOSER: TOMOHIRO OKA
AUTHOR: TOMOHIRO OKA

Problem

Given positive integer sequence and intervals.

Choose a pair of 2 indices from the interval (L, R), consider the GCD of the values.

Output the greatest of the GCD among all pairs from the interval.



Solution

- Common divisor
  - ⇨ A value appears twice in the interval as a divisor.
- Read sequence from left to right
- Manage the rightmost and second rightmost indices

6	2	14	4	3	2	5	1	-

d	1	2	3	4	5	6	7	...
f(d)	8	6	5	4	7	-1	3	
s(d)	7	4	1	-1	-1	-1	-1	

Solution

- Common divisor
  - ⇨ A value appears twice in the interval as a divisor.
- Read sequence from left
- Manage the rightmost and second rightmost indices

6	2	14	4	3	2	5	1	7

Update d=1, 7

d	1	2	3	4	5	6	7	...
f(d)	9	6	5	4	7	-1	9	
s(d)	8	4	1	-1	-1	-1	3	

Solution

- When i-th element is updated, solve all queries that have R=i
  - f(d) ≦ R is satisfied for all d
- If L ≦ s(d) is satisfied, then d is a common divisor in the interval
  - Find argmax_d { L ≦ s(d) }

6	2	14	4	3	2	5	1	7

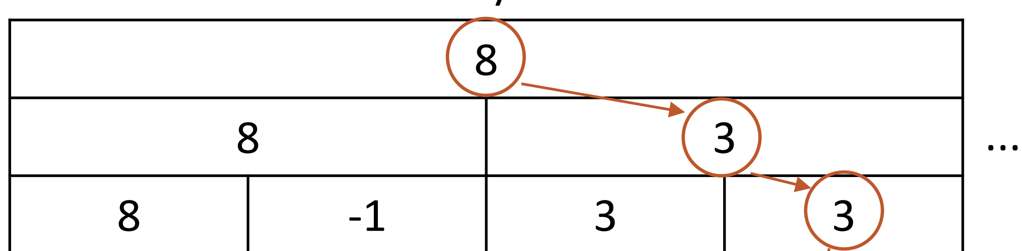
d	1	2	3	4	5	6	7	...
f(d)	9	6	5	4	7	-1	9	
s(d)	8	4	1	-1	-1	-1	3	

Solution

- Build a segment tree (range maximum query) for s(d)
- Binary search on segment tree

6	2	14	4	3	2	5	1	7

Binary search when L=3 ⇨ answer d=7



d	1	2	3	4	5	6	7	…
s(d)	8	4	1	-1	-1	-1	3	

Summary

- Group intervals by R
- Read the sequence from left to right
- Update second rightmost indices of divisors, and the segment tree
- Binary search with the condition { L ≦ s(d) } on the tree
- Maximum d is the greatest GCD

J: Mixing Solutions

PROPOSER: NAOKI MARUMO
AUTHOR: NAOKI MARUMO
+ KOHEI MORITA

Problem

Mix parts of n solutions to make a new solution

Given:

- amount
- **range** of concentration

} of each prepared solution

- amount
- target concentration c

} of the mixed solution

Minimize: **Max. Error** $\max_{x \in [l,r]} |x - c|$

$[l, r]$: concentration range of the mixed solution

Reformulate Problem

Mix parts of n solutions to make a new solution

Given:

- amount
- **range** of concentration

} of each prepared solution

- amount
- target concentration c

} of the mixed solution

Minimize: **Max. Error** $\max_{x \in [l,r]} |x - c| = \max\{c - l, r - c\}$

$[l, r]$: concentration range of the mixed solution

Reformulate Problem

We want to solve

$$\min_{l,r} \max\{c - l, r - c\}$$

but what range does $(l, r) \in \mathbb{R}^2$ move over?

$[l, r]$: concentration range of the mixed solution

Reformulate Problem

We want to solve

$$\min_{l,r} \max\{c - l, r - c\}$$

but what range does $(l, r) \in \mathbb{R}^2$ move over?

The set of possible $(l, r) \in \mathbb{R}^2$ is a **convex polygon!**

- **Note:** A convex combination of two valid mixings is also valid

$[l, r]$: concentration range of the mixed solution

Reformulate Problem

We want to solve

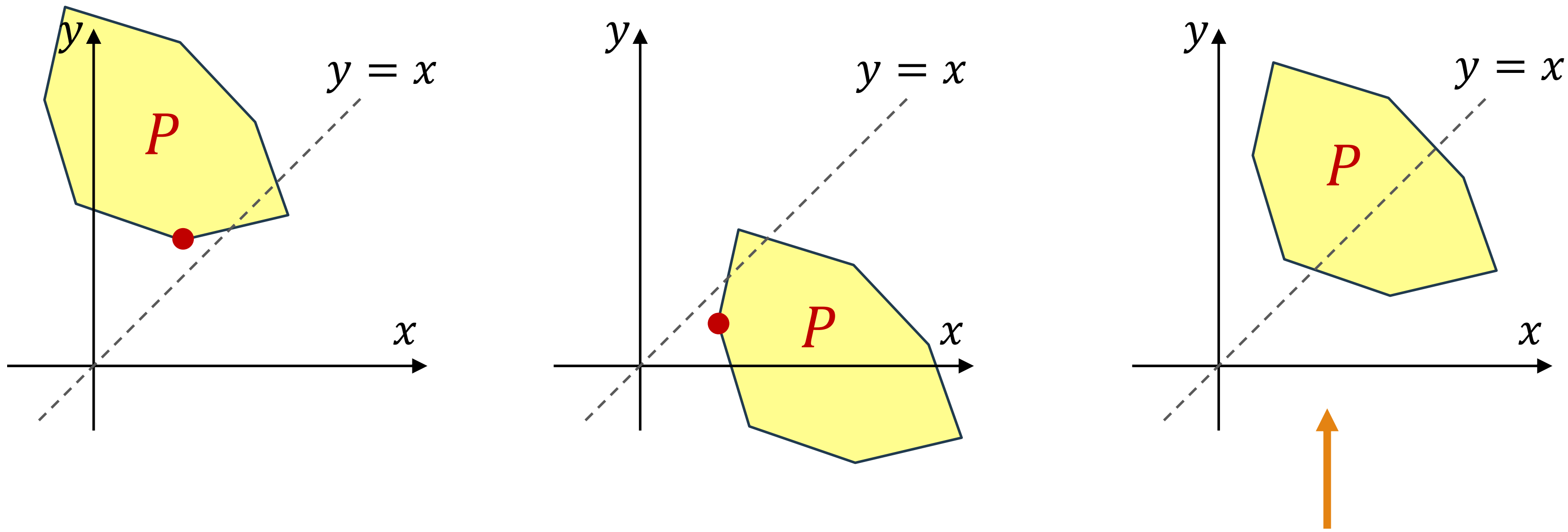
$$\min_{l,r} \max\{c - l, r - c\} = \min_{(x,y)\in P} \max\{x, y\}$$

- $(x, y) \coloneqq (c - l, r - c)$
- (x, y) also moves over a convex polygon, P

$[l, r]$: concentration range of the mixed solution

Three cases

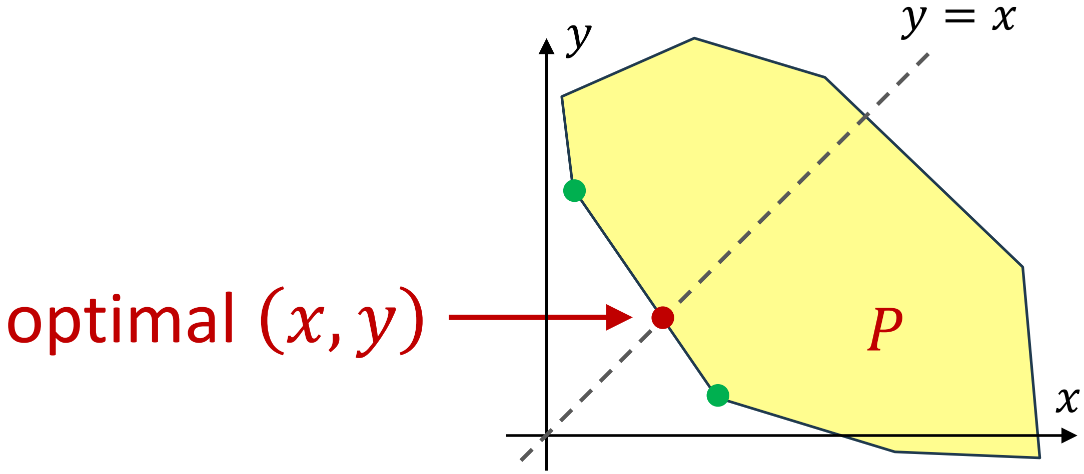
$$\min_{(x,y)\in P} \max\{x, y\}$$



Most interesting case

Outline of solution

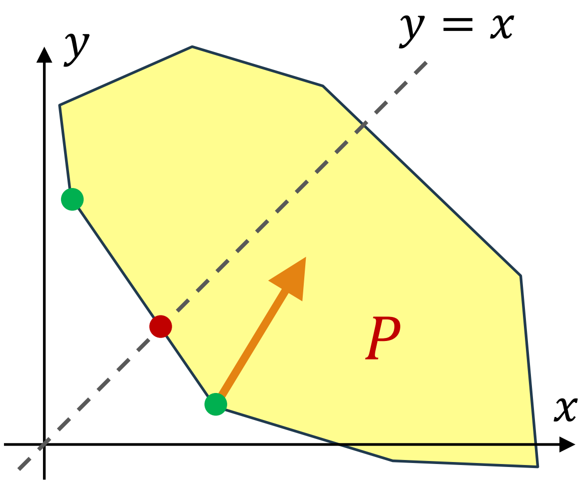
$$\min_{(x,y)\in P} \max\{x, y\}$$



- The red point can be easily obtained from two adjacent green vertices
- Compute the green vertices by **binary search**

Compute green vertices

- Each vertex is the point minimizing the inner product with a vector
- Given a vector, the vertex that minimizes the inner product can be computed greedily in $O(n \log n)$ time
- Green vertices can be computed by binary searching the vectors



Details

For the binary search, we can

- precompute $\Theta(n^2)$ candidate vectors and sort them
  → $\mathrm{O}(n^2 \log n)$ solution

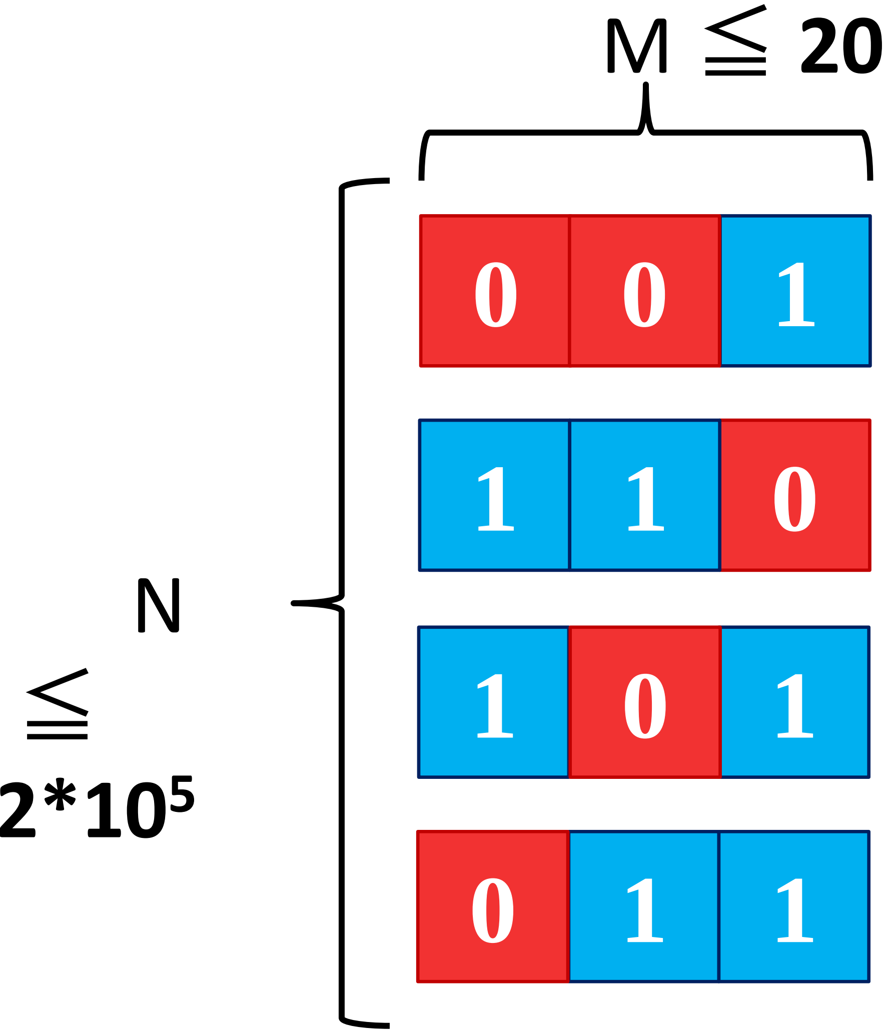
- use $\left(\frac{1}{B}, \frac{B-1}{B}\right), \left(\frac{2}{B}, \frac{B-2}{B}\right), \dots, \left(\frac{B-1}{B}, \frac{1}{B}\right)$ with $B = \Theta(M^2)$
  → $\mathrm{O}(n \log n \log M)$ solution

Problem J can also be solved using the **minimax theorem** or **LP duality**, without relying on geometric insights

K: Scheduling Two Meetings

PROPOSER: KAZUHIRO INABA
AUTHOR: KAZUHIRO INABA

Problem



Given many M-bits words, find a pair (a,b) such that

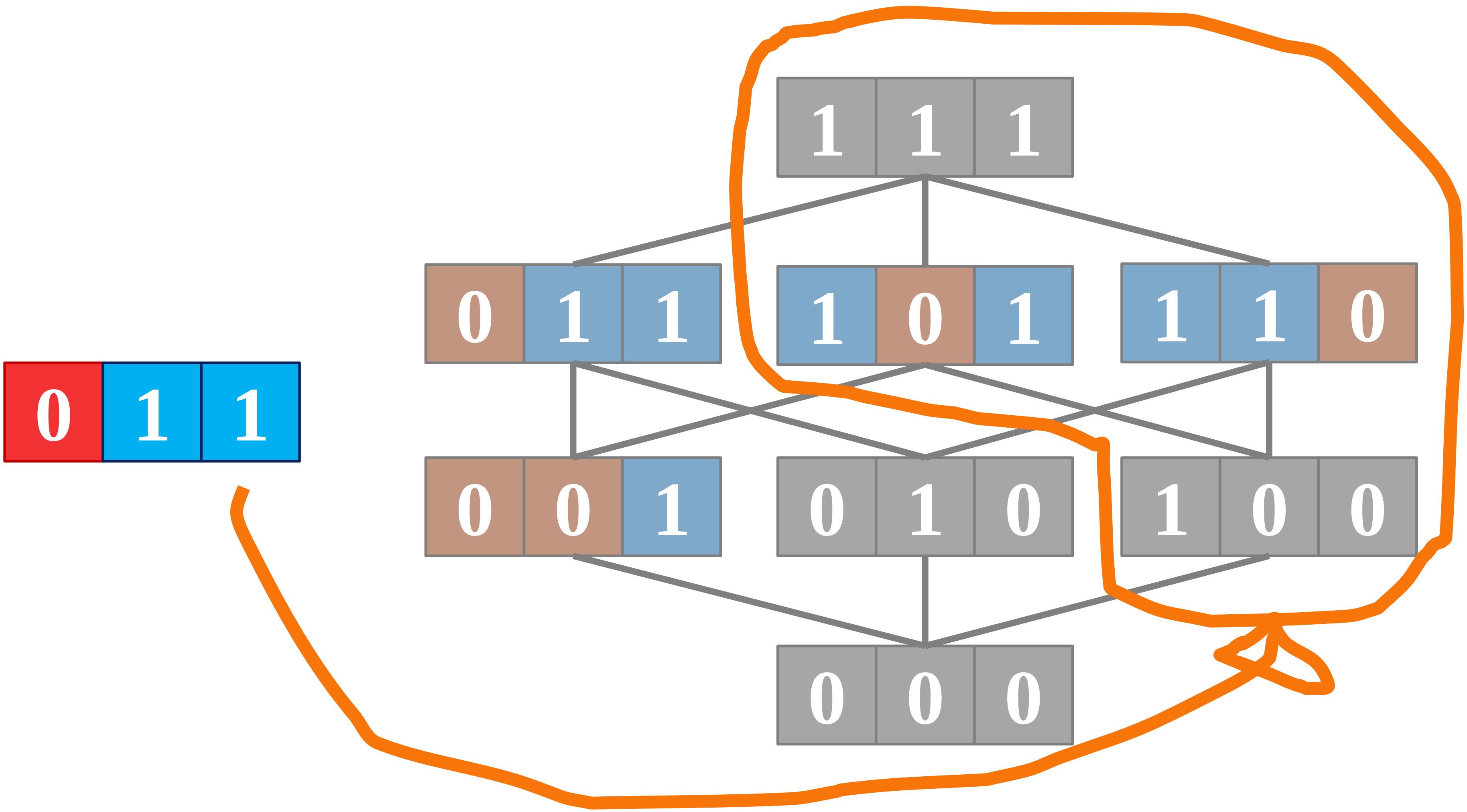
a|b = 11…11
a&b has the most 1s

Solution

For each word, find the best buddy!
Naïve $\Theta(N^2)$ loop will hit TLE, though…

	001	110	101	011
001		0		
110	0		1	1
101		1		1
011		1	1	

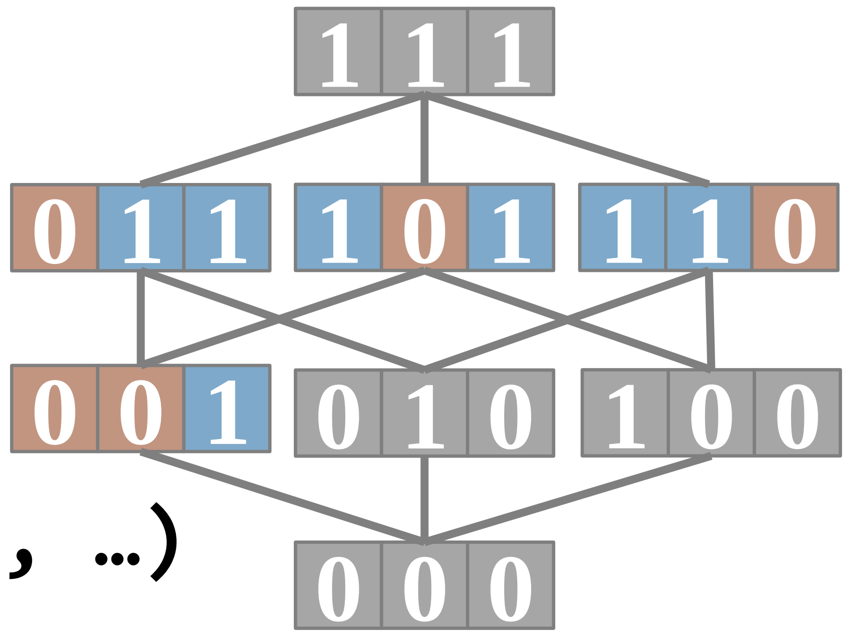
The best buddy => word with the most **1**s among **supersets** of the **complement**!



Solution

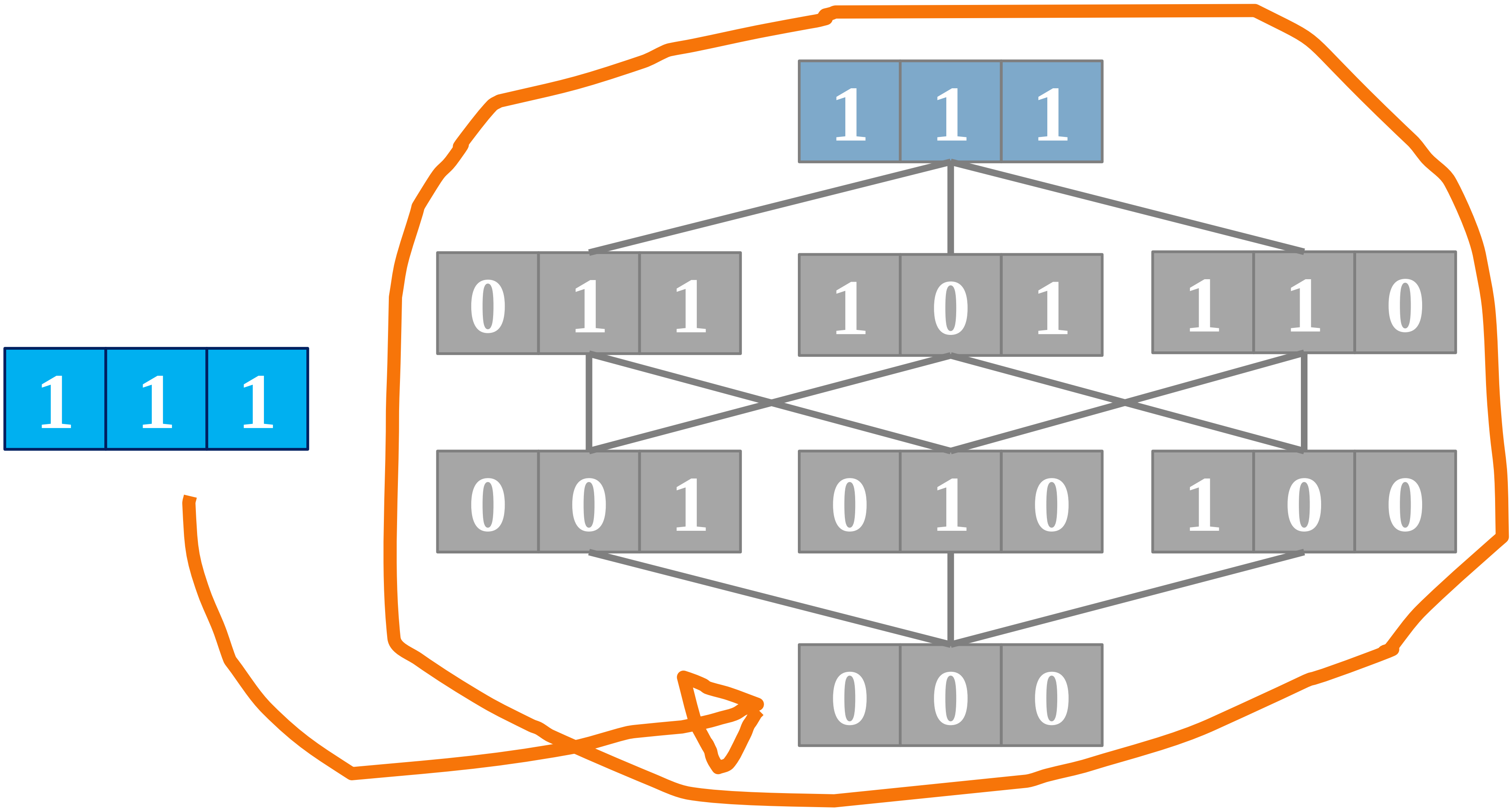
The diagram can be preprocessed in **$O(M \cdot 2^M)$** time dynamic programming. **$O(N + M \cdot 2^M)$** time in total.

dp[bitpat] :=
  the best word in input
  among supersets of bitpat




```
dp[bitpat]
 = max(dp[bitpat|1<<i], …)
    for i in 0 .. M-1
```

The corner case, that many teams were trapped



L: Peculiar Protocol

PROPOSER: KAZUHIRO INABA
AUTHOR: SOH KUMABE

Problem

Given an array $a_1, \dots a_n$ ($n \leq 500$) and integers d, r

We can repeat following:

- Take interval $[p, q]$ with $a_p + \cdots + a_q = kd + r$ for some integer k
- Remove these elements and squeeze the sequence

Maximize the total of k's

Subproblem

Given an array $a_1, \dots a_n$ ($n \leq 500$) and integers d, r

We can repeat following:

- Take interval $[p, q]$ with $a_p + \cdots + a_q = kd + r$ for some integer k
- Remove these elements and squeeze the sequence

First, consider the following variant:

Maximize the total of k's,
assuming we remove all elements

Subproblem

First, consider the following variant:

Maximize the total of k's,
assuming we remove all elements

If this variant on all intervals $a_p, \dots, a_q$ are solved,
remaining task is the following $O(n^3)$-time DP on intervals

$DP[p][q] =$ answer for the original problem
on instance $(a_p, \dots, a_q)$

Subproblem

Given an array $a_1, \dots a_n$ ($n \leq 500$) and integers d, r

We can repeat following:

- Take interval $[p, q]$ with $a_p + \cdots + a_q = kd + r$ for some integer k
- Remove these elements and squeeze the sequence

Assuming we remove all elements,
maximizing the total k's is equivalent to
minimizing the number of operations

Proof: total k's = $\frac{\sum a_i - r \cdot \#operations}{d}$

Subproblem

Minimize the number of operations,
assuming we remove all elements

Let b be the minimum positive integer with

$$a_1 + \cdots + a_n \equiv br \pmod{d}$$

We can show either b operation is enough,
or it is impossible to remove entire sequence

Proof: if we use $b' > b$ operations,
we can unify last $b' - b + 1$ operations into one operation

Subproblem

So the problem is:
Is it possible to remove entire sequence?

This is solved by the following $O(n^3)$-time DP on intervals

$DP[p][q] =$ maximum number of operations
to remove whole interval $(a_p, \dots, a_q)$