Commentaries on Problems

JUDGE TEAM

ACM ICPC 2017 ASIA TSUKUBA REGIONAL

Differences from Previous Years

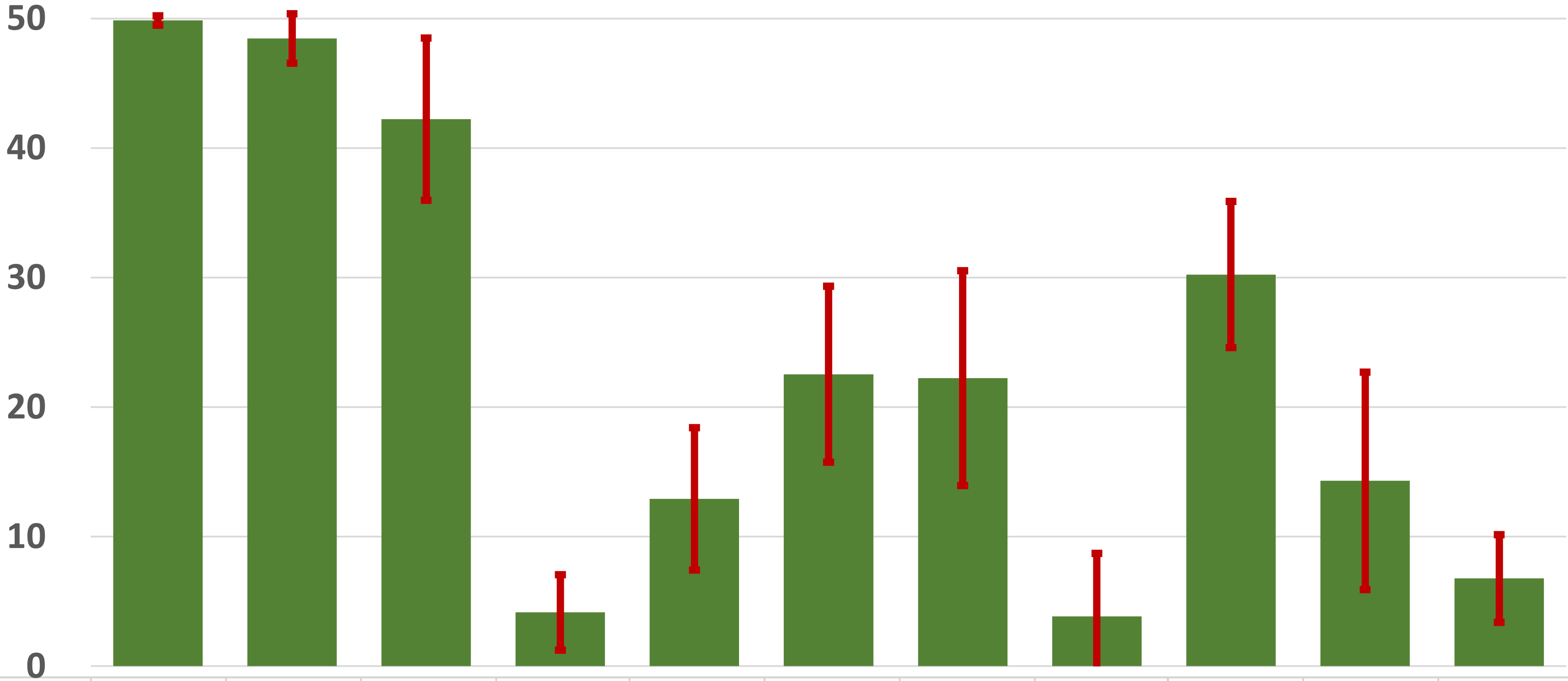
Python introduced, like World Finals.
Both Python 2 and 3.

Order of problems shuffled.
The first three problems are the easiest,
but others are in a random order.

Estimated Order of Difficulty

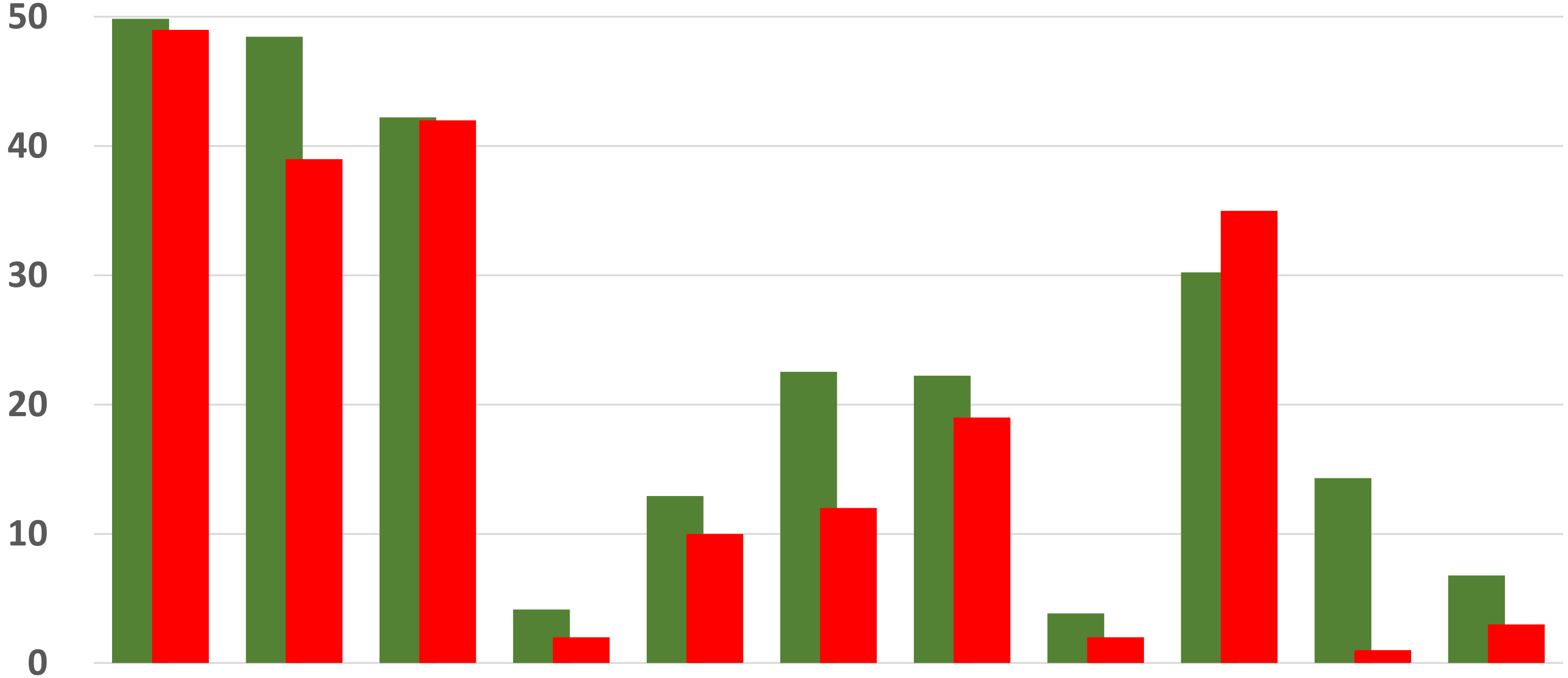
	← Easiest								Hardest →		
Coding	A	B	C	I	G	J	E	F	H	K	D
Analysis	A	B	C	G	I	F	E	J	K	D	H

Predicted # of Correct Answers



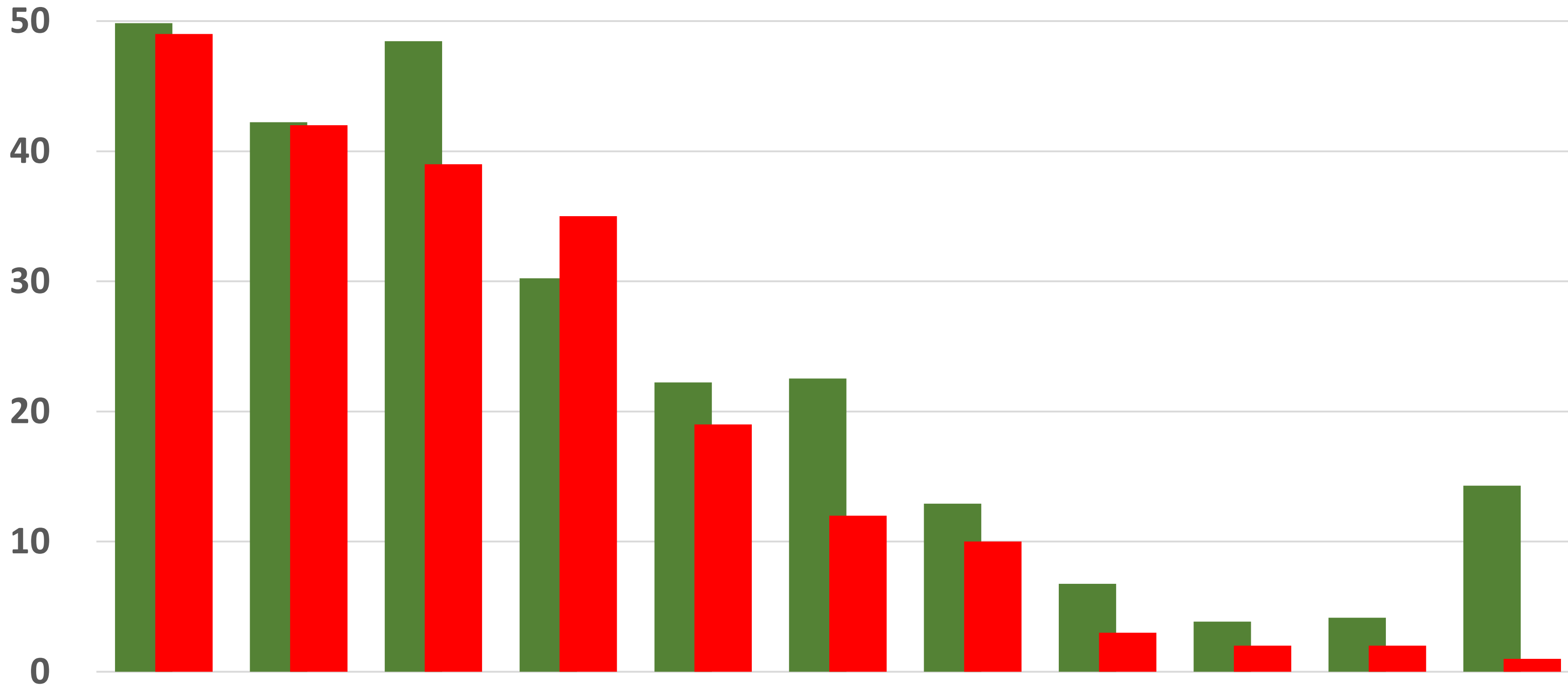
	A	B	C	D	E	F	G	H	I	J	K
Average	49.85	48.46	42.23	4.15	12.92	22.54	22.23	3.85	30.23	14.31	6.77
Std. Dev.	0.36	1.91	6.27	2.90	5.50	6.79	8.29	4.87	5.65	8.40	3.38

Estimated vs. Actual



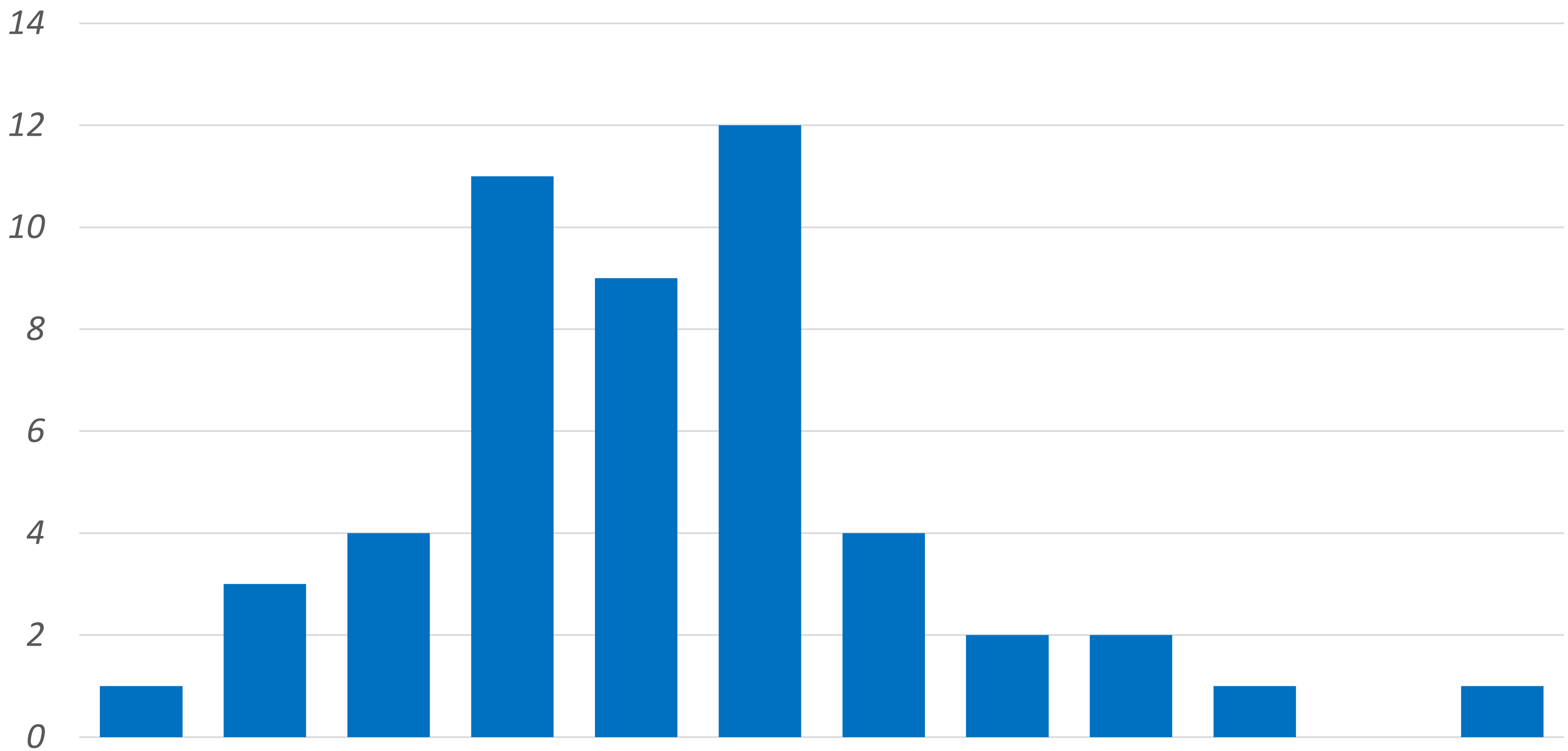
	A	B	C	D	E	F	G	H	I	J	K
Estimated	49.85	48.46	42.23	4.15	12.92	22.54	22.23	3.85	30.23	14.31	6.77
Actual	49	39	42	2	10	12	19	2	35	1	3

Estimated vs. Actual



	A	C	B	I	G	F	E	K	H	D	J
Estimated	49.85	42.23	48.46	30.23	22.23	22.54	12.92	6.77	3.85	4.15	14.31
Actual	49	42	39	35	19	12	10	3	2	2	1

problems solved & # teams

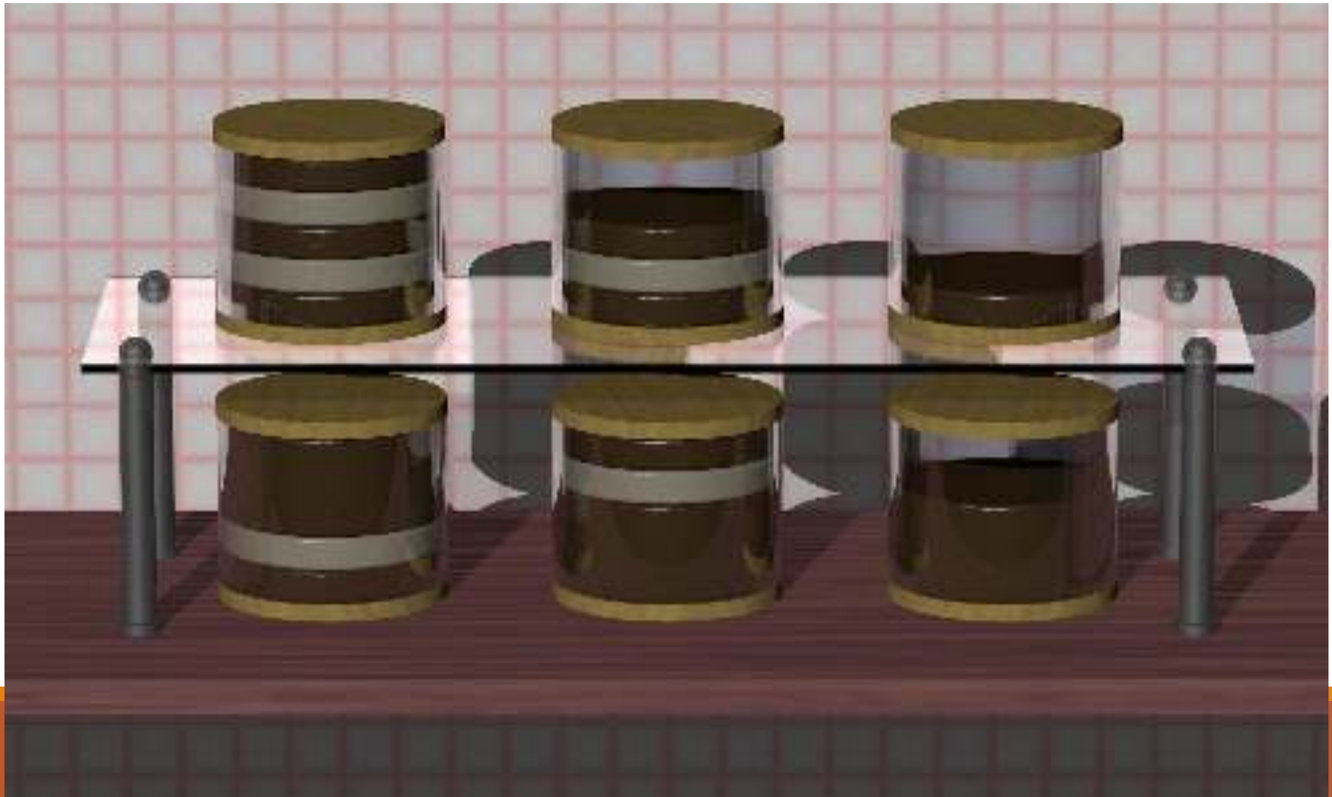


Solved	0	1	2	3	4	5	6	7	8	9	10	11
Teams	1	3	4	11	9	12	4	2	2	1	0	1

A: Secret of Chocolate Pole

Story

- Wendy makes poles of chocolate.
- Different poles may have different "side views."

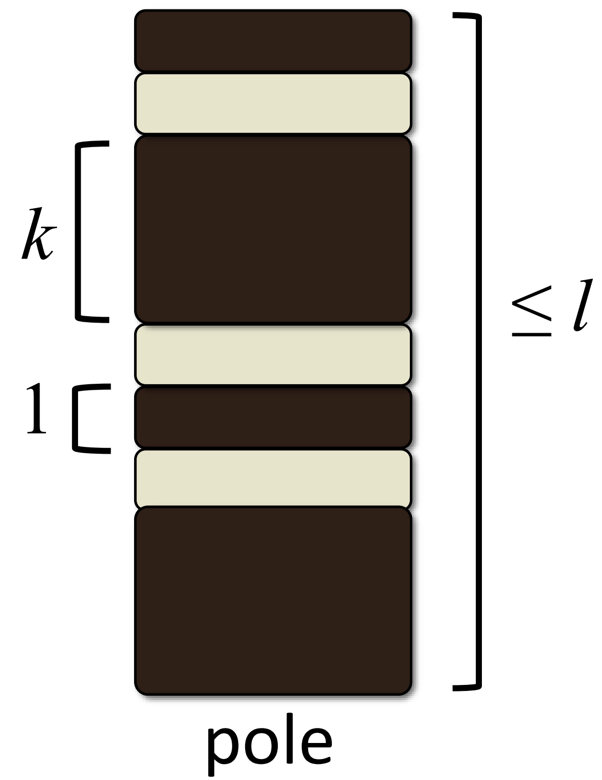


Problem

Count the number of possible distinct "side views."

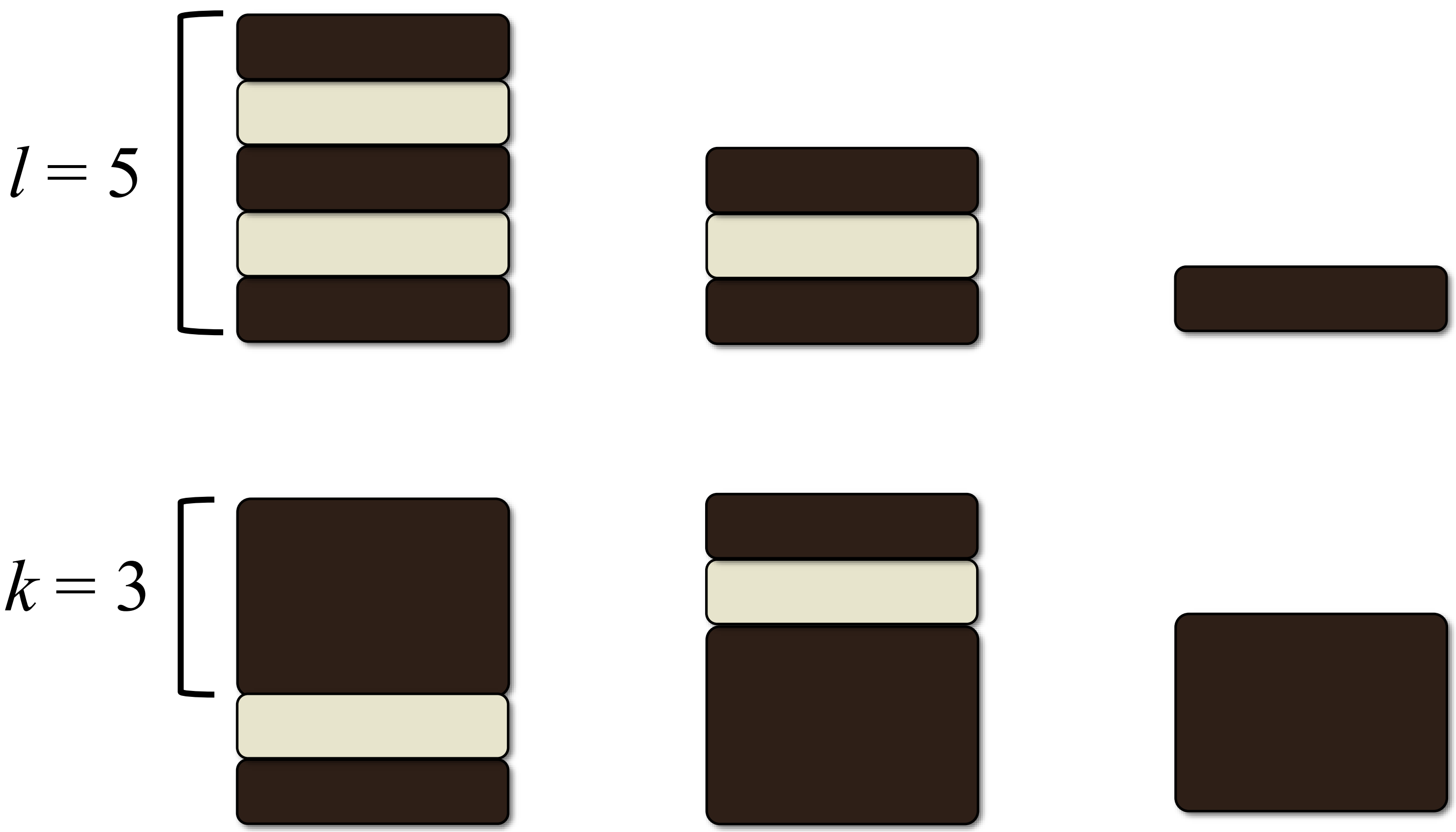
Conditions:

- Pole consists of dark and white chocolate, stacked alternately.
- Top and bottom are dark.
- Conditions on height
  - dark block: 1 cm, k cm
  - white block: 1 cm
  - pole: $\leq l$ cm



Example

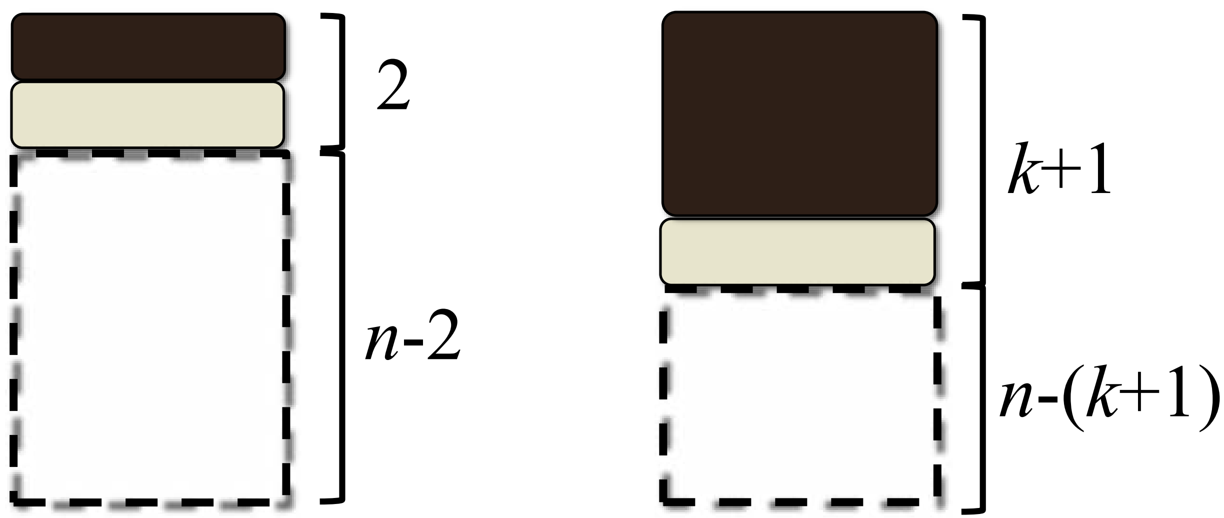
$l = 5, k = 3 \Rightarrow \text{answer} = 6$



Solution --- dynamic programming

1. Consider the number of side views of exactly n high, $S(n)$
2. Sum up $S(n)$ for $0 \leq n \leq l$

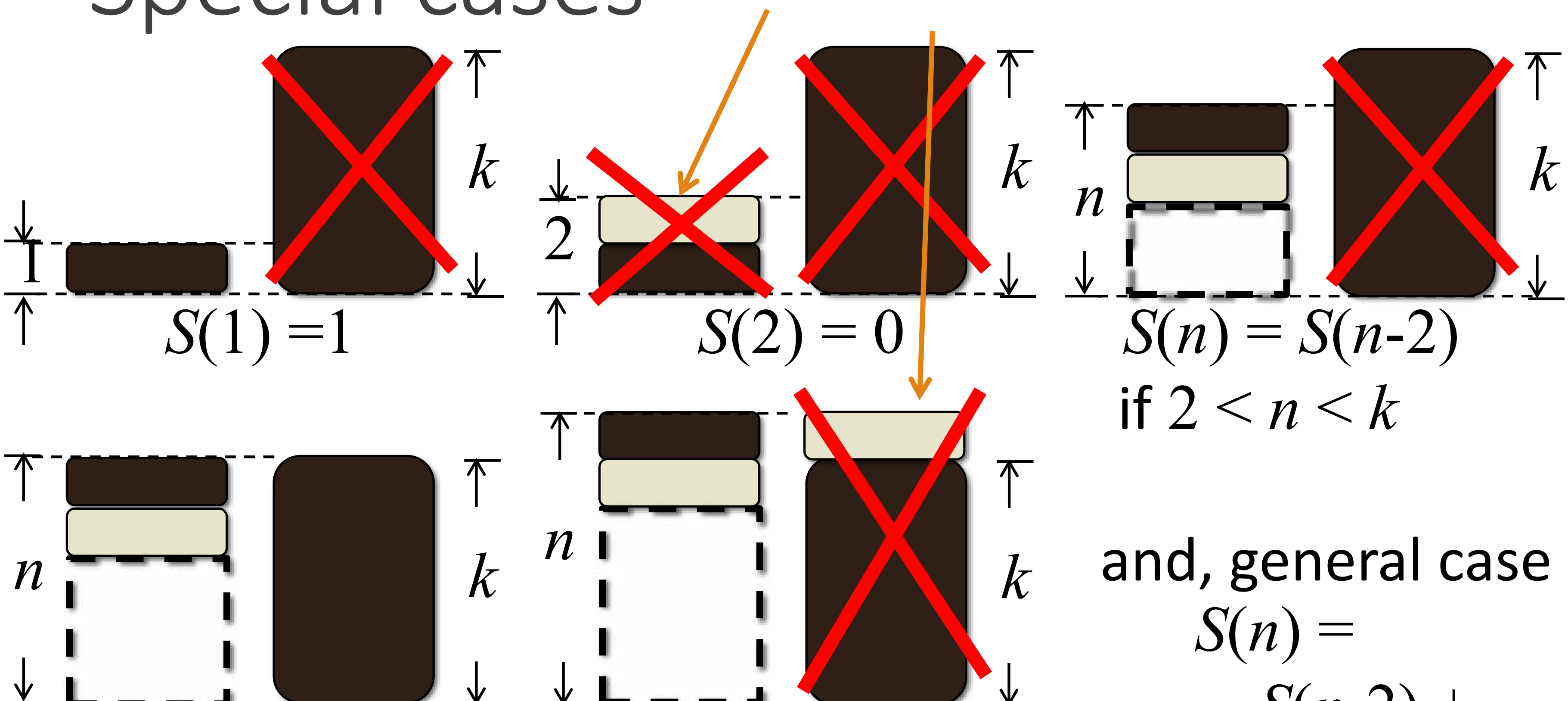
For large n,



$$S(n) = S(n-2) + S(n-(k+1))$$

Special cases

Top should be dark



$S(1) = 1$

$S(2) = 0$

$S(n) = S(n\text{-}2)$ if $2 < n < k$

$S(n) = S(n\text{-}2) + 1$ if $n = k$

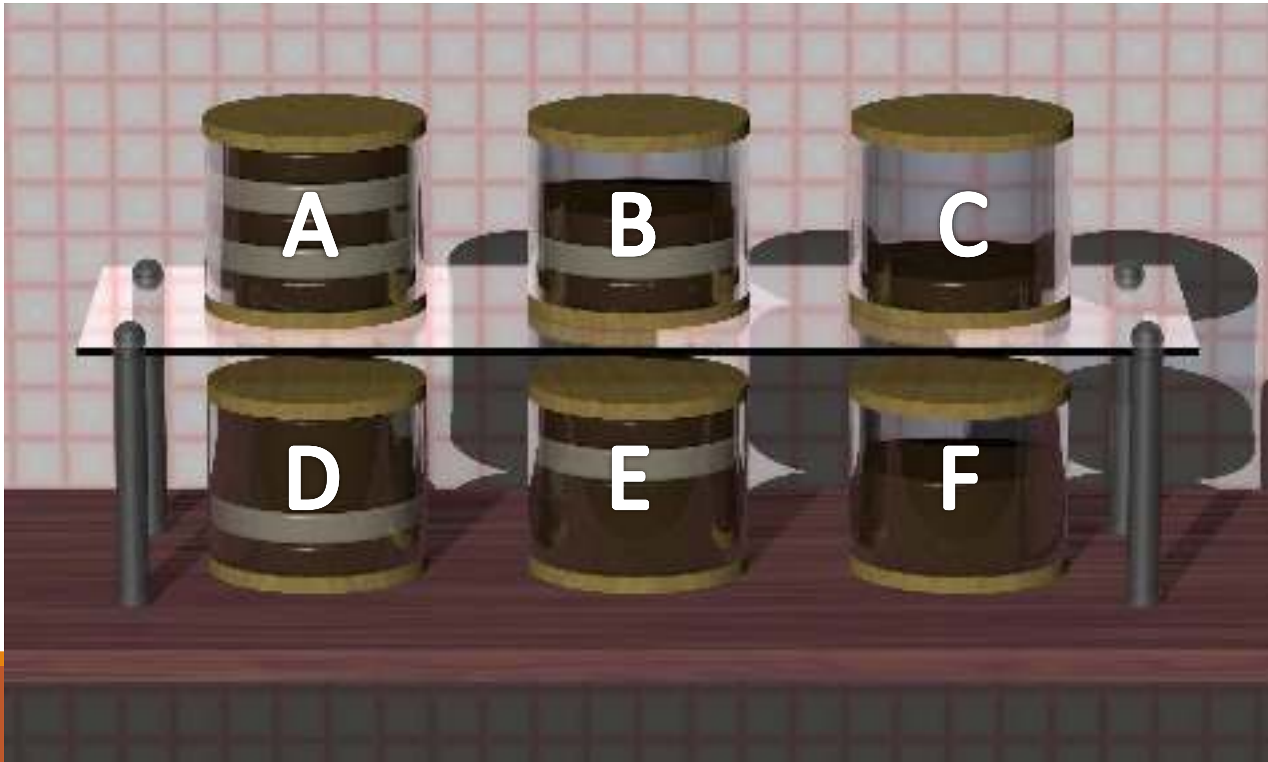
$S(n) = S(n\text{-}2)$ if $n = k + 1$

and, general case

$S(n) = S(n\text{-}2) + S(n\text{-}(k{+}1))$ if $n > k + 1$

Remaining Mystery: The secret of chocolate poles

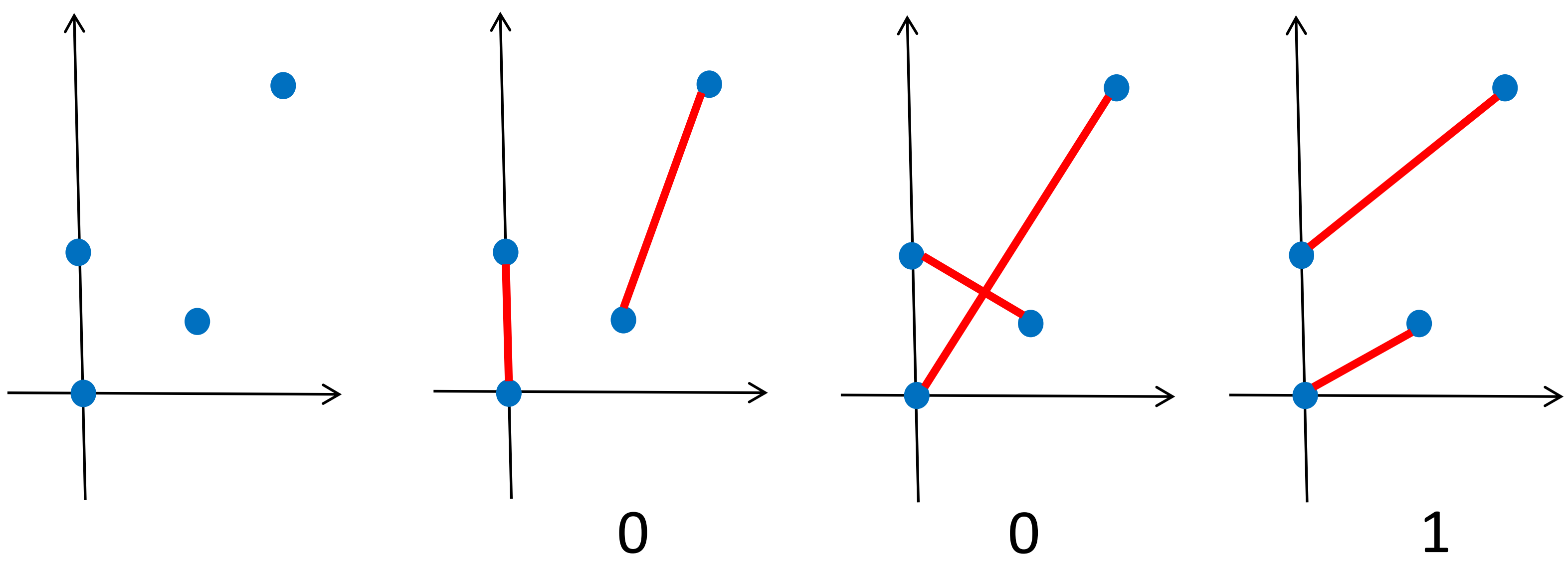
- Wendy was a spy
- She was developing a secret coding that uses the patterns of chocolate poles



Problem B: Parallel Lines

Problem Summary

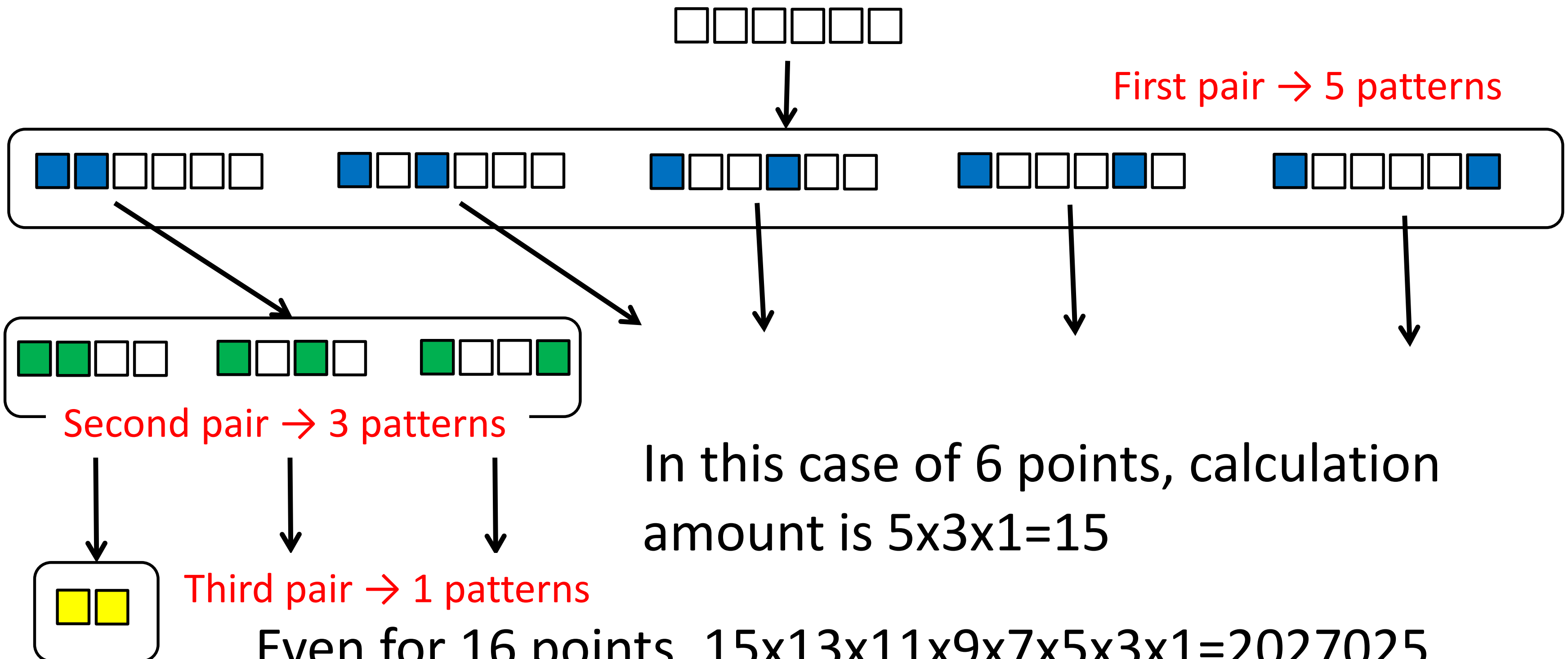
- Couple all the points into pairs
- Draw a connecting line between the points of each point pairs
- Count number of the parallel line pairs
- Answer the maximum number of the parallel line pairs



number of parallel line pairs

Couple all the points into pairs

- For example of 6 points, couple the points into pairs as follows.



In this case of 6 points, calculation amount is 5x3x1=15

Even for 16 points, 15x13x11x9x7x5x3x1=2027025

- DO NOT make permutations of all the points. 16! is TOO LARGE.

Judge that two vectors are parallel

- For vectors v_1 and v_2 , $|v_1 \times v_2| = 0$ holds when v_1 and v_2 are parallel.

$$v_1 = (x_1, y_1, z_1)$$
$$v_2 = (x_2, y_2, z_2)$$
$$v_1 \times v_2 = (y_1 z_2 - z_1 y_2, z_1 x_2 - x_1 z_2, x_1 y_2 - y_1 x_2)$$

- In this case, both of v_1 and v_2 are on XY-plane.

$$v_1 = (x_1, y_1, 0)$$
$$v_2 = (x_2, y_2, 0)$$
$$v_1 \times v_2 = (0, 0, x_1 y_2 - y_1 x_2)$$

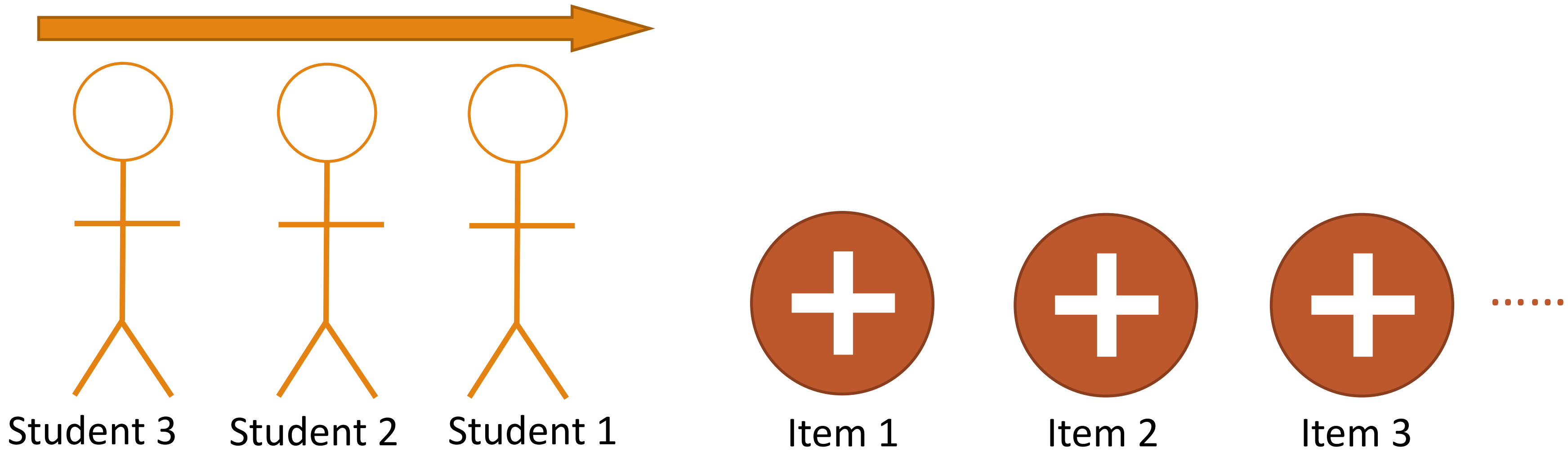
← z component

- So you can judge it by computing $x_1 y_2 - y_1 x_2 = 0$

C: Medical Checkup

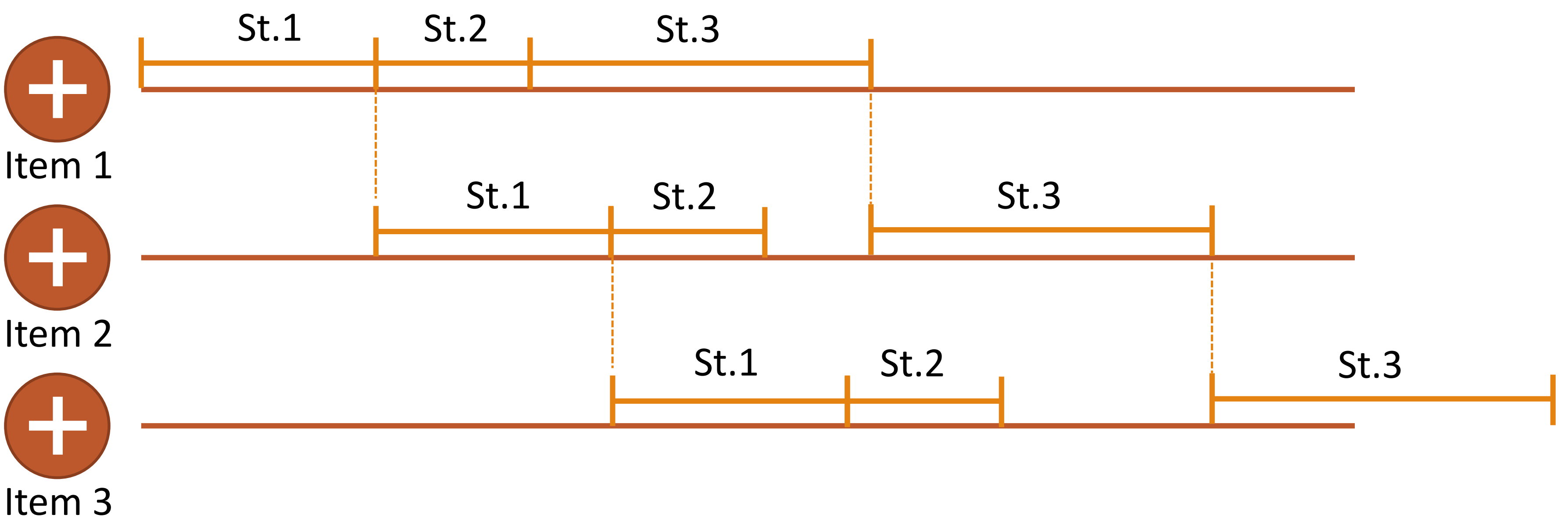
Problem:

- Students need to undergo checkups in order.
- The *i*-th student takes h_i unit time to finish each checkup item.
- Find the items students are being checked up or waiting for at specified time.



Solution:

Consider a time sequence diagram.



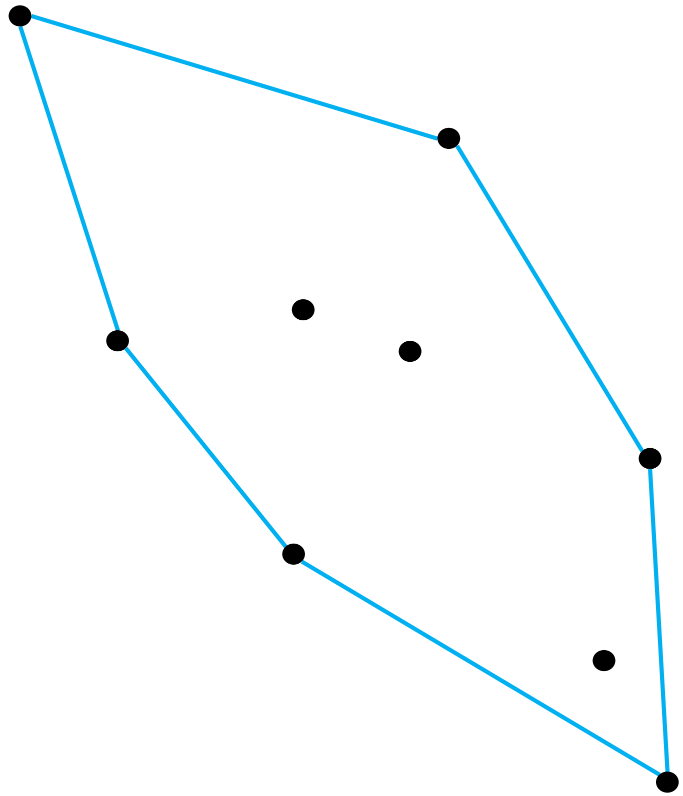
Let's call the *i*-th student is *important* if $h_k < h_i$ for all $k < i$.

- Non-important student just follows a preceding student.
- Important student moves as if he/she ignores all others.

=> Student moves with uniform linear motion. O(n) time.

D: Making Perimeter of the Convex Hull Shortest

Problem: Given a set of planar points, make the convex hull of the set shortest by eliminating two points



The convex hull of a set of planar points is the smallest convex polygon that has all the points in the set on its edges or inside of it.

Finding the Convex Hull

Many algorithms have been proposed.

- Gift wrapping (Jarvis march): $O(nh)$ ← Too slow as n and h can be as large as 10^5
- Graham scan: $O(n \log n)$
- Andrew's algorithm: $O(n \log n)$
- Divide and conquer: $O(n \log n)$
- Chan's algorithm: $O(n \log h)$

n = # of points in the set

h = # of vertices of the convex hull

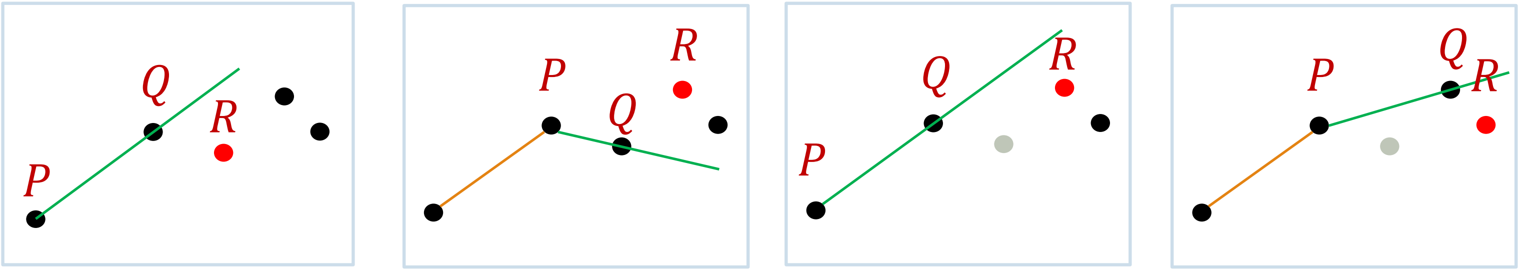
h may be as large as n in this problem

Andrew's Monotone Chain

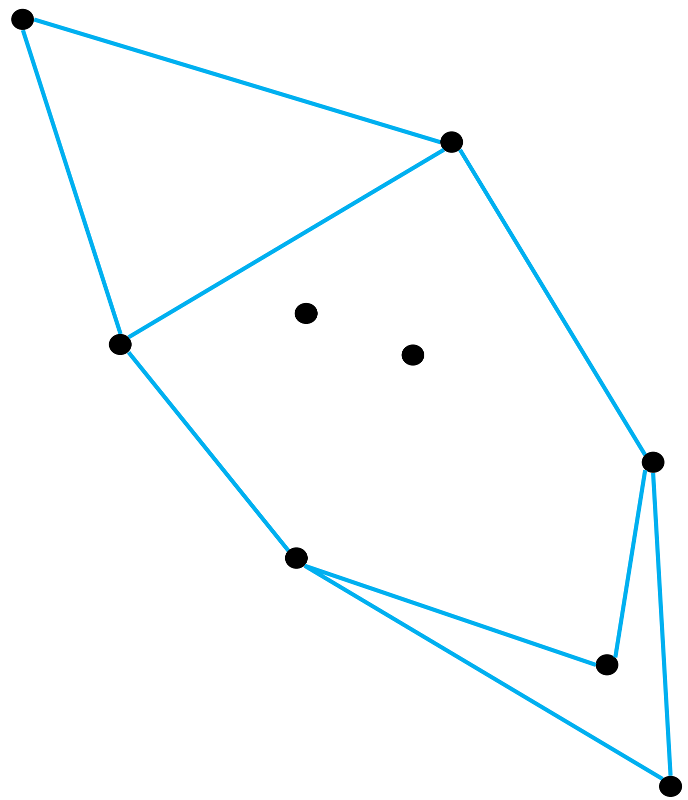
To construct the upper half of the convex hull:

1. Sort the points with their x coordinates, start a left to right scan, naming two leftmost points P and Q
2. If the next point R is below the line $\overline{PQ}$, remember P in the candidate point stack, let P be Q, Q be R, and repeat this step
3. If R is above $\overline{PQ}$, Q cannot be a vertex of the convex hull; let Q be P, pop P from the stack, and go back to 2
4. If no more point is left, stop

The lower half can be constructed similarly



The Convex Hull can be Made Shorter by Eliminating Some Points



Naïve Solution

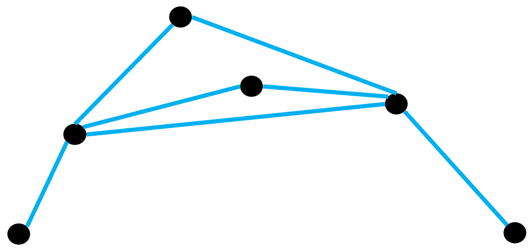
- Consider all possible subsets after eliminating two of the points
- Find the convex hulls of each of them

This algorithm is too slow

- There are $n(n-1)/2$ ways to eliminate two points
- Time complexity of $O(n \log n)$ is required to find the convex hull of each of the subset
- The total time complexity will be $O(n^3 \log n)$

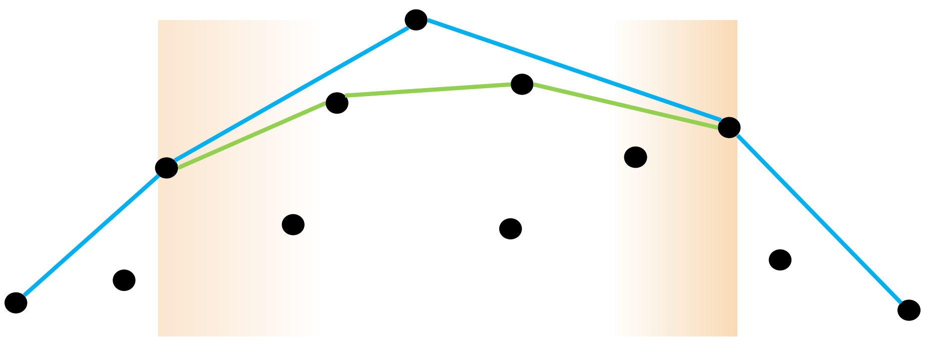
Eliminating Two Points, One by One

- One of the points is on the original convex hull; Otherwise, the convex hull won't change
- Elimination will result in a new convex hull
- Another point is to be eliminated from those on the new convex hull, which either was
  - Already on the original convex hull, or
  - Added newly because of the first elimination



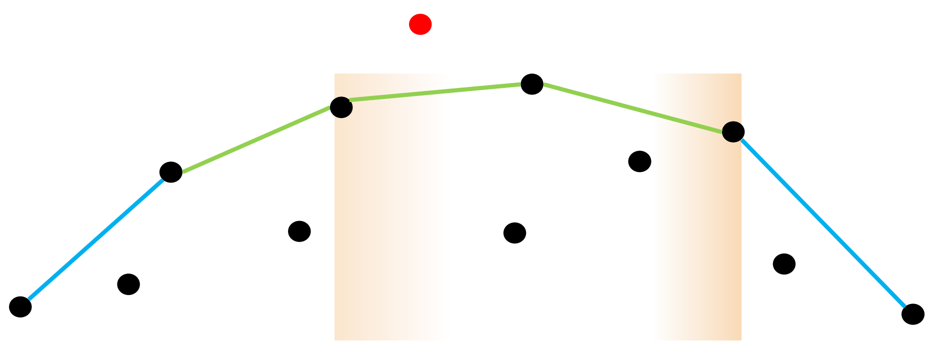
Eliminating One Point

- Find the original convex hull: $O(n \log n)$
- Find the new convex hulls for when each point on the original convex hull is eliminated (h cases)
  - Candidate new vertices of the hull are those between two adjacent original hull points
  - Each point is checked only twice, $2n$ times in total, keeping the complexity of $O(n \log n)$



Eliminating a Newly Added Point

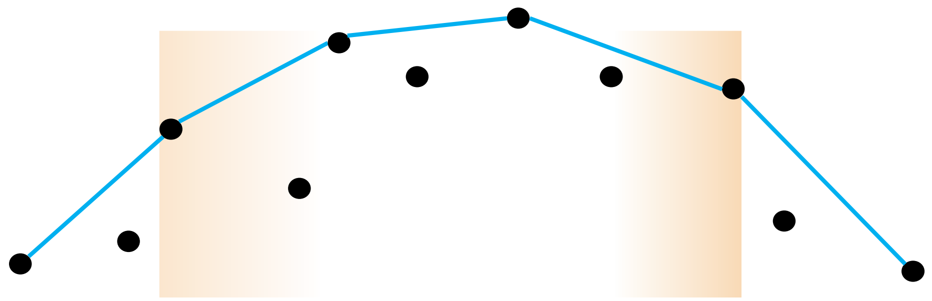
- Eliminating one of newly added points requires inspecting only those between two adjacent points on the new convex hull
- The total number of points investigated is $2n$ again, not affecting the total computational complexity of $O(n \log n)$



Eliminating Two Points on the Original Convex Hull

When two points are adjacent on the original hull

- Points to investigate are those between two hull points adjacent to the eliminated two
- Each point is checked only three times, and thus $3n$ checks in total are made



Eliminating Two Point on the Original Convex Hull *(cont.)*

When two points are *not* adjacent

- The gains of shortening the convex hull perimeter are independent;
  the sum of their gains is the net gain
- But considering all the $h(h-3)/2 \cong n^2/2$ combinations is too costly…

Finding the Best Combination without Too Much Cost

Keep the list of the best 4 candidate points: $O(h)$

- The #1 candidate can be adjacent to only 2 of the 3 other points in the list
- If #1 is not adjacent to #2, the answer is #1+#2
- Otherwise, if #1 is not adjacent to #3, #1+#3
- If neither, #1 cannot be adjacent to #4;
  The answer is the better of #1+#4 and #2+#3

Key Points

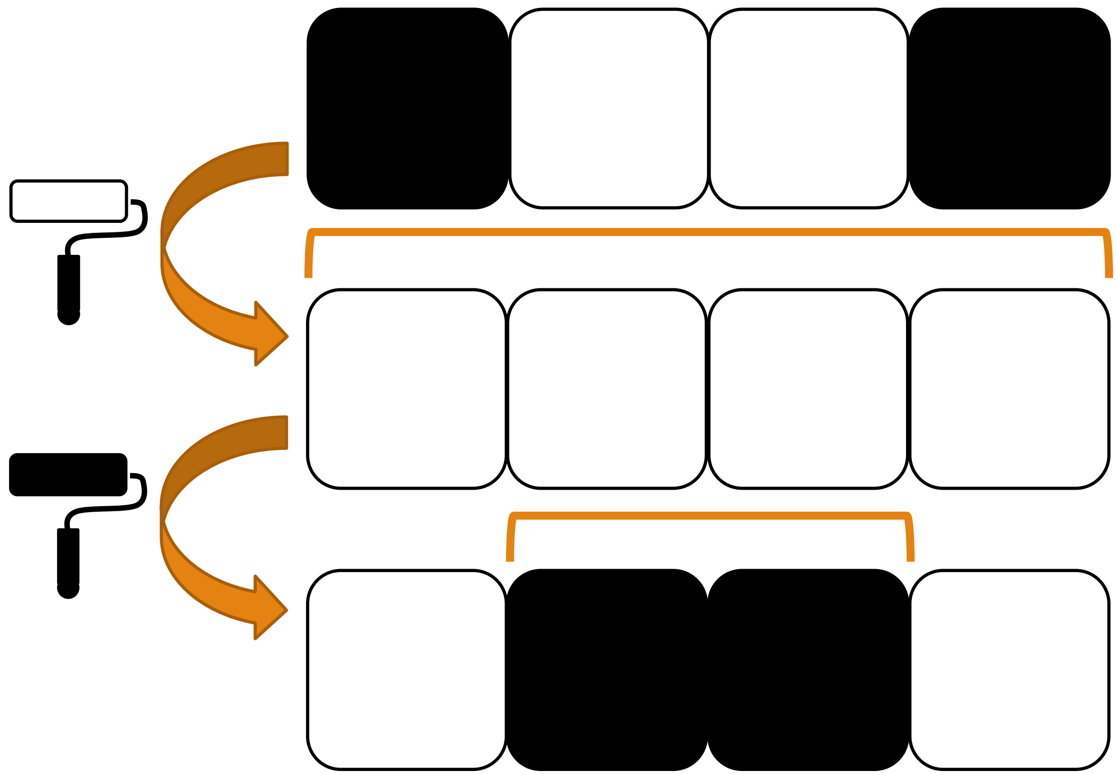
- Focusing on the differences may drastically reduce the computational complexity
- Take all possibilities into consideration

E: Black or White

Problem Summary

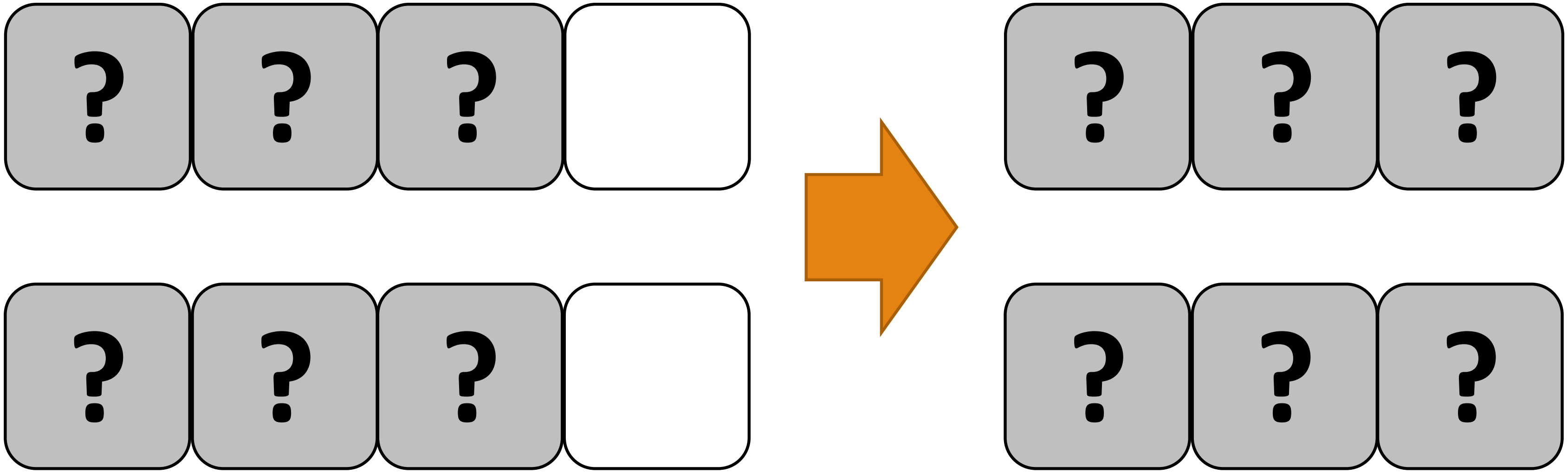
Paint a row of bricks into desired colors
The number of bricks painted in one stroke is at most k
Calculate the minimum number of strokes

$1 \leq k \leq n \leq 500000$



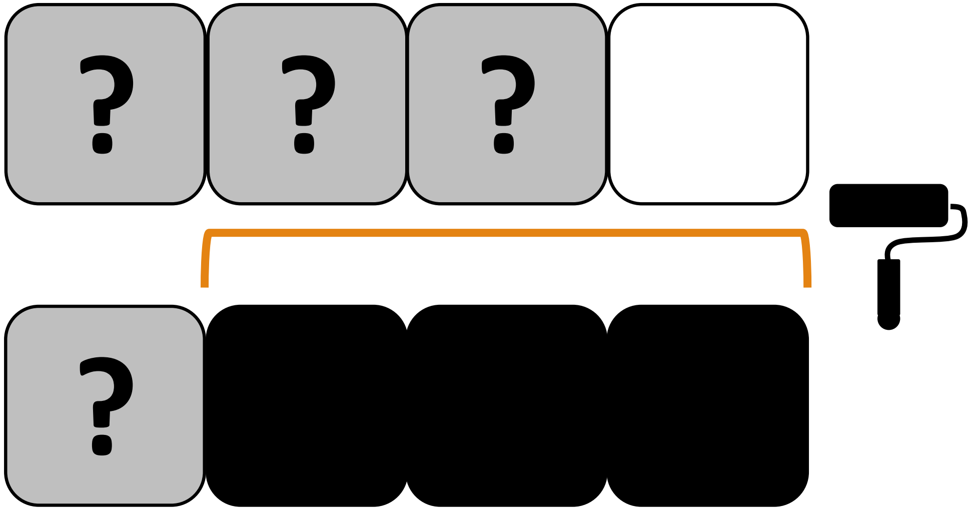
Focus on the last brick

If the initial color of the last brick is the **same** as the desired color, we can ignore the brick



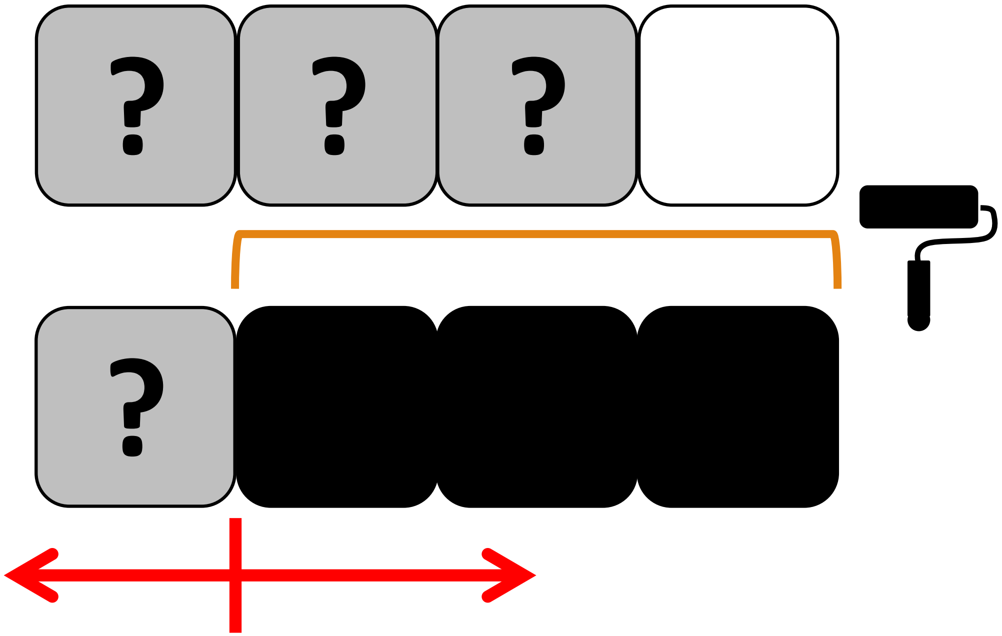
Focus on the last brick

Otherwise, we need to paint the last brick



Focus on the last brick

Otherwise, we need to paint the last brick

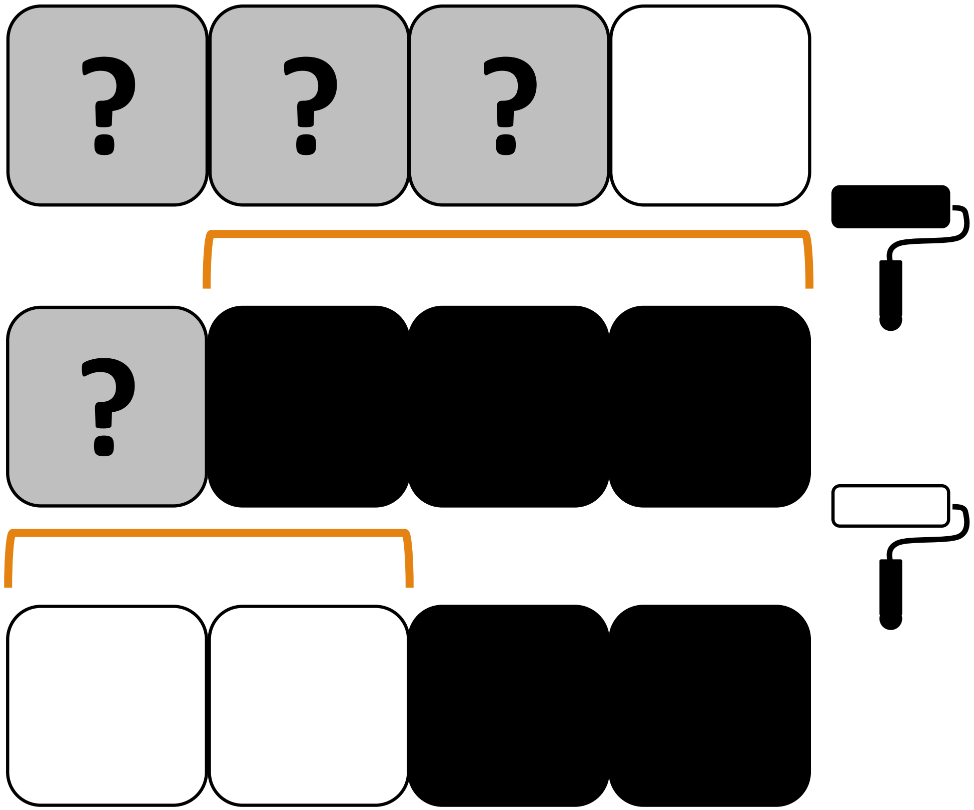


The left side is independent of the right side

→ The left side can be calculated recursively

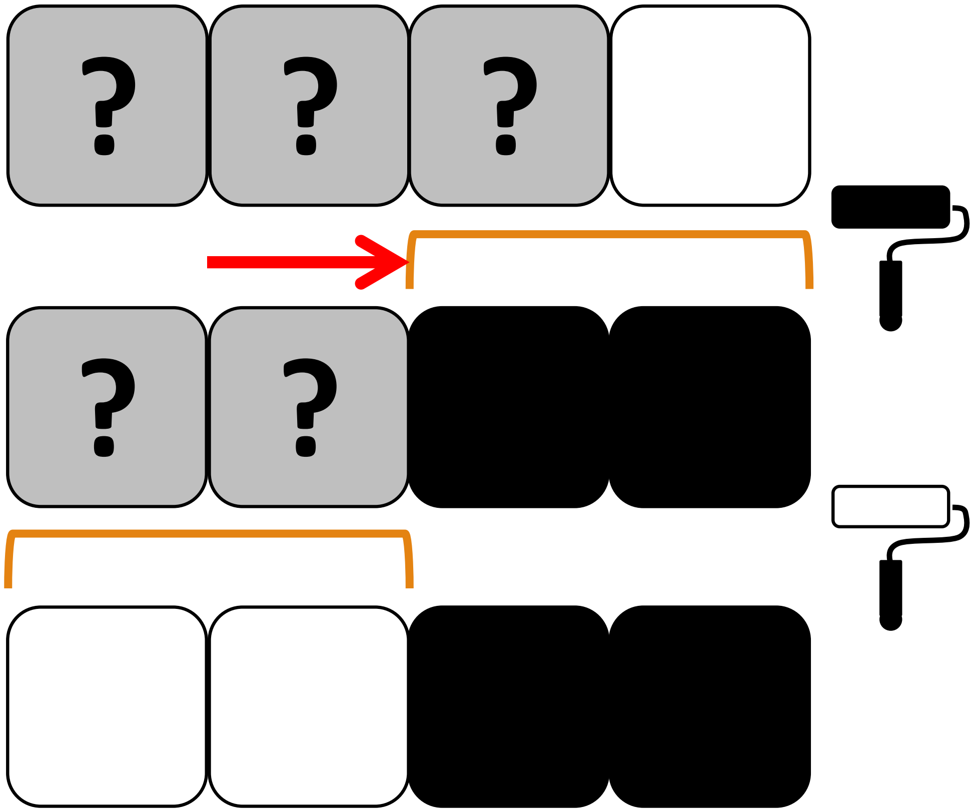
Why independent?

If we overpaint bricks, we can shorten the first stroke
This is not affected by colors



Why independent?

If we overpaint bricks, we can shorten the first stroke
This is not affected by colors



Calculate the right side

- All bricks have the same color now
- We can choose any length
- The number of borders between black and white increases at most 2 in one stroke

$$\#\ of\ minimum\ strokes = ceil\left(\frac{\#\ of\ borders\ in\ desired\ colors}{2}\right)$$

This is always feasible

DP in $O(nk)$

- If the initial color of x is the same as the desired color
$$dp[x] = dp[x-1]$$
- Otherwise
$$dp[x] = \min_{x-k \le i \le x-1} \left(dp[i] + ceil\left(\frac{b[i+1..x]}{2}\right) + 1 \right)$$
- Answer is $dp[n]$

* $b[i+1..x]$: # of borders in desired colors from $i+1$ to x

Speed up

$$dp[x] = \min_{x-k \le i \le x-1} \left(dp[i] + ceil\left(\frac{b[i+1..x]}{2}\right) + 1 \right)$$

Speed up

$$dp[x] = \min_{x-k \le i \le x-1}\left(dp[i] + ceil\left(\frac{b[i+1..x]}{2}\right) + 1\right)$$

$$dp[x] = \min_{x-k \le i \le x-1}\left(dp[i] + ceil\left(\frac{b[0..x] - b[0..i+1]}{2}\right) + 1\right)$$

Speed up

$$dp[x] = \min_{x-k \le i \le x-1}\left(dp[i] + ceil\left(\frac{b[i+1..x]}{2}\right) + 1\right)$$

$$dp[x] = \min_{x-k \le i \le x-1}\left(dp[i] + ceil\left(\frac{b[0..x] - b[0..i+1]}{2}\right) + 1\right)$$

$$dp[x] = \min_{x-k \le i \le x-1}\left(ceil\left(\frac{2dp[i] - b[0..i+1] + b[0..x] + 2}{2}\right)\right)$$

Speed up

$$dp[x] = \min_{x-k \le i \le x-1}\left(dp[i] + ceil\left(\frac{b[i+1..x]}{2}\right) + 1\right)$$

$$dp[x] = \min_{x-k \le i \le x-1}\left(dp[i] + ceil\left(\frac{b[0..x] - b[0..i+1]}{2}\right) + 1\right)$$

$$dp[x] = \min_{x-k \le i \le x-1}\left(ceil\left(\frac{2dp[i] - b[0..i+1] + b[0..x] + 2}{2}\right)\right)$$

$$dp[x] = ceil\left(\frac{\min_{x-k \le i \le x-1}(2dp[i] + b[0..i+1]) + b[0..x] + 2}{2}\right)$$

Speed up

$$dp[x] = \min_{x-k \le i \le x-1} \left(dp[i] + ceil\left(\frac{b[i+1..x]}{2}\right) + 1 \right)$$

$$dp[x] = \min_{x-k \le i \le x-1} \left(dp[i] + ceil\left(\frac{b[0..x] - b[0..i+1]}{2}\right) + 1 \right)$$

$$dp[x] = \min_{x-k \le i \le x-1} \left(ceil\left(\frac{2dp[i] - b[0..i+1] + b[0..x] + 2}{2}\right) \right)$$

$$dp[x] = ceil\left(\frac{\min_{x-k \le i \le x-1} (2dp[i] + b[0..i+1]) + b[0..x] + 2}{2}\right)$$

This can be calculated in $O(1)$ with deque or $O(\log n)$ with segment tree

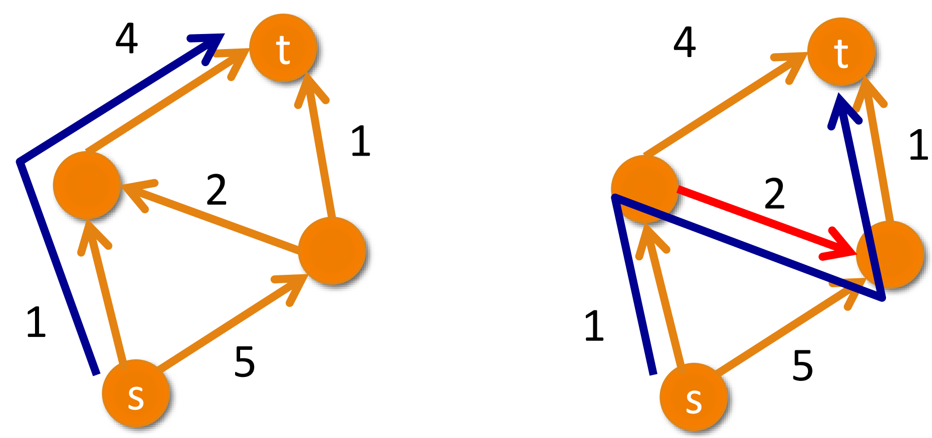
Summary

- Calculate the left and right side independently after the first stroke
- The left side can be calculated recursively
- The right side can be calculated only by # of borders
- Speed up DP with cumulative sum and data structure
- The time complexity is $O(n)$ or $O(n \log n)$

F: Pizza Delivery

Problem

- Given a directed positive-weighted graph.
- When the direction of i-th edge is reversed, how does the distance from s to t change, shorter, longer, or unchanging?
- Answer it about each edge.



Shorter or Not?

- We denote the distance from u to v on a graph G by d(G, u, v).
- Let’s reverse an edge
  - remove e = (u, v, c)
  - add e’ = (v, u, c)
- d(G - e + e’, s, t) < d(G, s, t) iff every shortest path on G + e’ must run through e’ and must not run through e.
- Check d(G, s, v) + c + d(G, u, t) is shorter or not.
- Calculate d(G, s, ・) and d(G, ・, t) with Dijkstra’s algorithm.

Longer or Not? (1/2)

- Assume d(G - e + e’, s, t) is not shorter.
- Prop:
  - let A = d(G + e’, s, u) + c + d(G + e’, v, t)
  - let B = d(G + e’, s, v) + c + d(G + e’, u, t)
  - At least one of A or B is larger than d(G + e’, s, t).
- Proof:
  - 2 d(G + e’, s, t) < A + B, since
  - d(G + e’, s, t) ≦ d(G + e’, s, u) + d(G + e’, u, t) and
  - d(G + e’, s, t) ≦ d(G + e’, s, v) + d(G + e’, v, t)

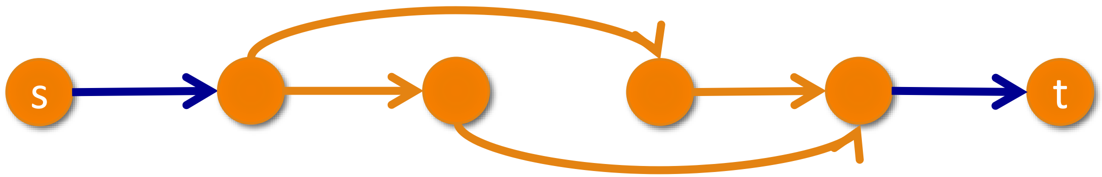
Longer or Not? (2/2)

- Assume d(G - e + e’, s, t) is not shorter.
- Let H be a subgraph of all shortest paths on G.

1. When e is in all shortest path of G
   - e is a bridge of H.
   - d(G - e + e’, s, t) > d(G, s, t), since the prop.
2. Otherwise
   - Removing e doesn’t change shortest path of G.
   - d(G - e + e’, s, t) = d(G, s, t)

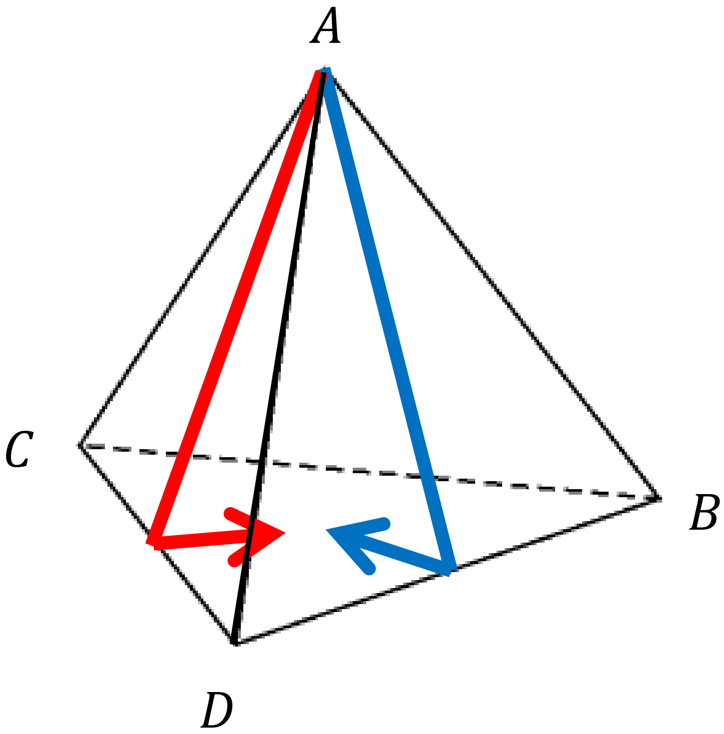
Enumerate Bridges

- Bridge: an edge, when it is removed, the number of connected components increases.
- In this case, when a bridge is removed, s and t are disconnected.
- Graph H is a DAG. Bridges are enumerated with simple calculation by topological order.



G: Rendezvous on a Tetrahedron

Problem Summary

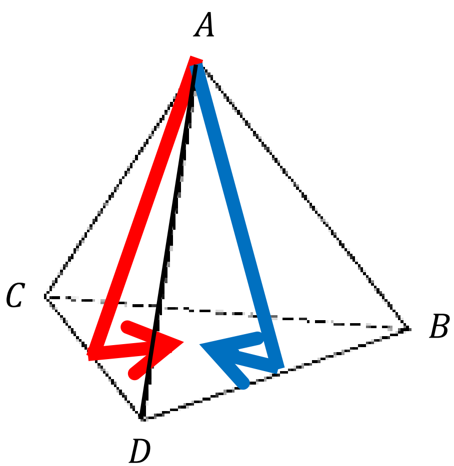


- Two worms crawled on the surface of a regular tetrahedron
- The trails were straight
- The unit length of the trails was the length of the edge of the tetrahedron
- Answer whether two worms stopped on the same face or not

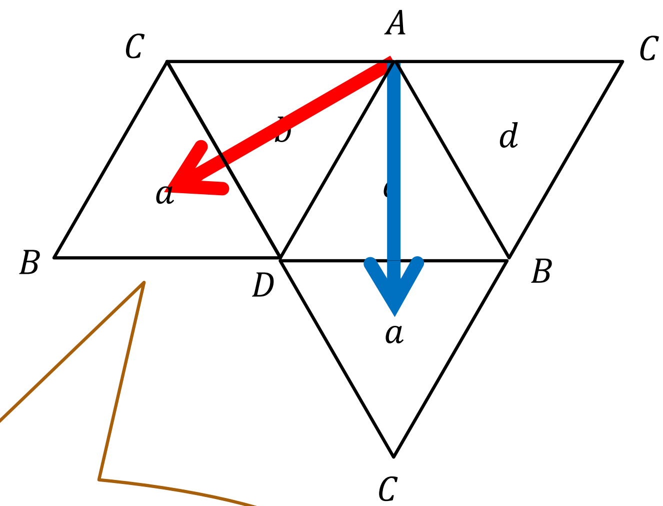
Trails on the Unfolding

The trails are straight lines on the unfolding.

Regular Tetrahedron

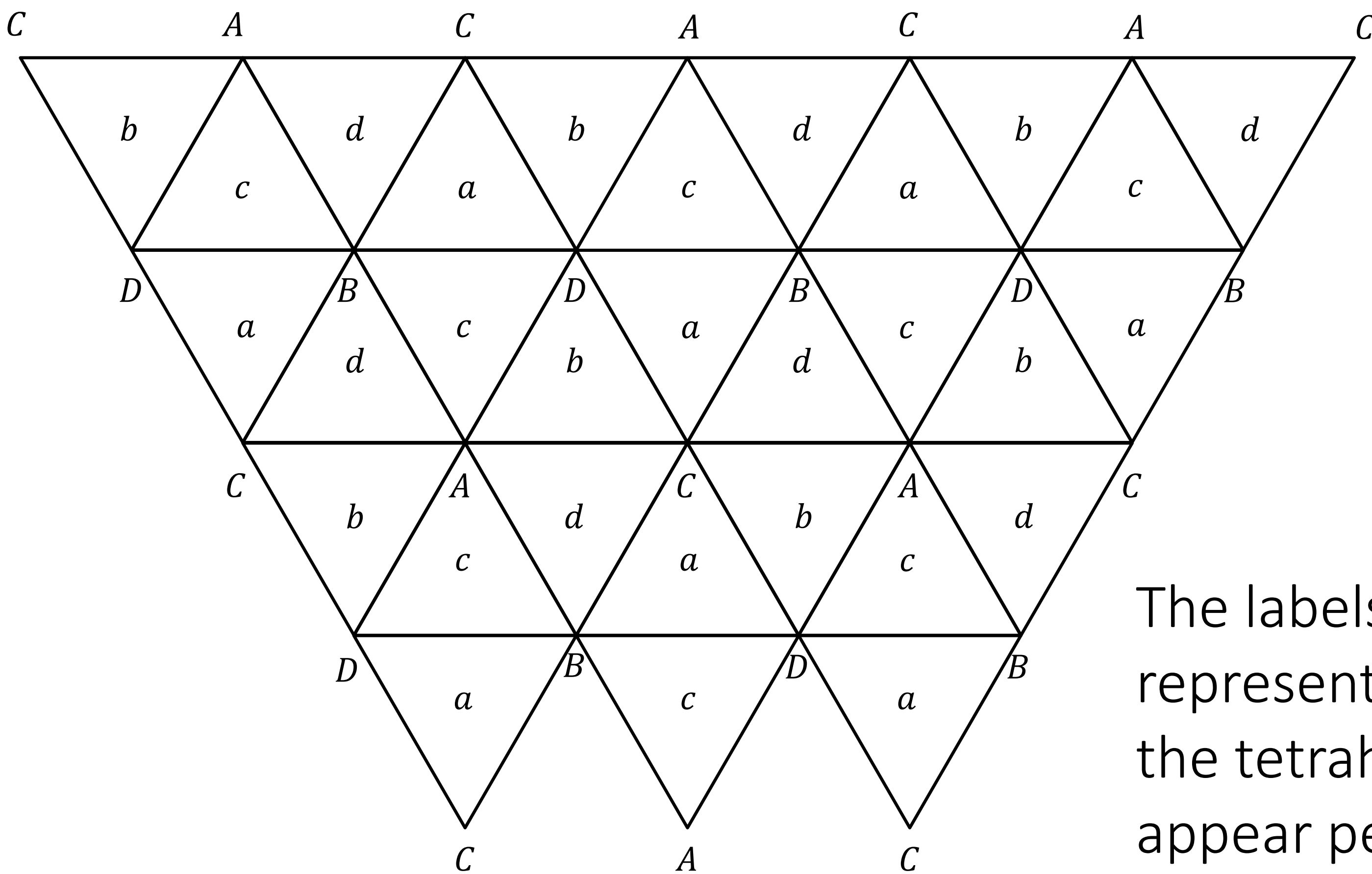


Unfolding



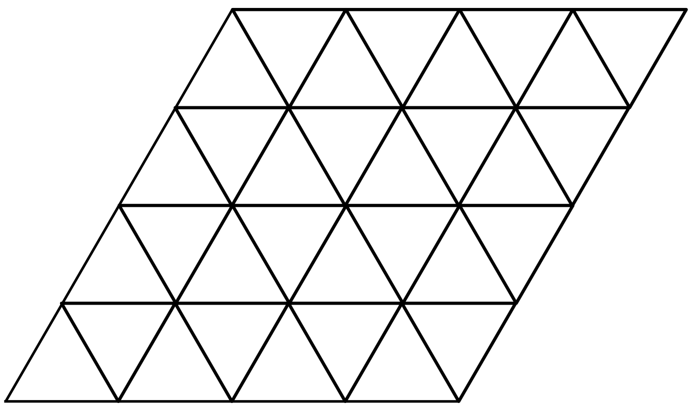
Out of the unfolding?
Expand the unfolding.

Expanding the Unfolding



The labels, which represent the faces of the tetrahedron, appear periodically.

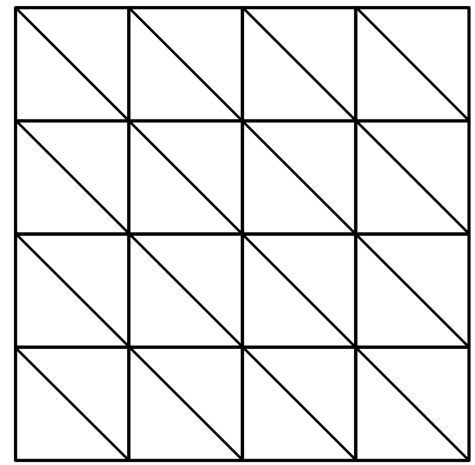
To Simplify Discrimination



Expanded Unfolding

transform

$$\begin{bmatrix} 1 & \frac{1}{\sqrt{3}} \\ 0 & \frac{2}{\sqrt{3}} \end{bmatrix}$$

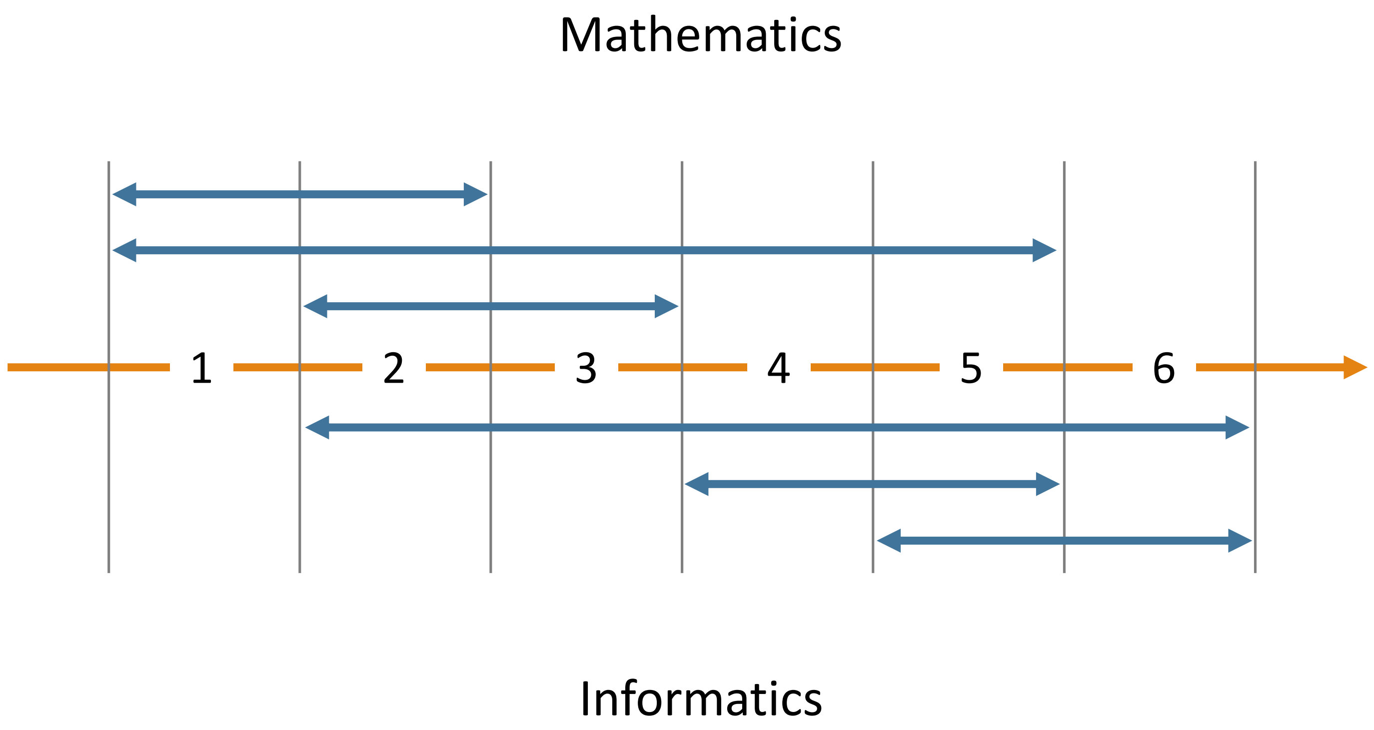


Discriminating the faces by

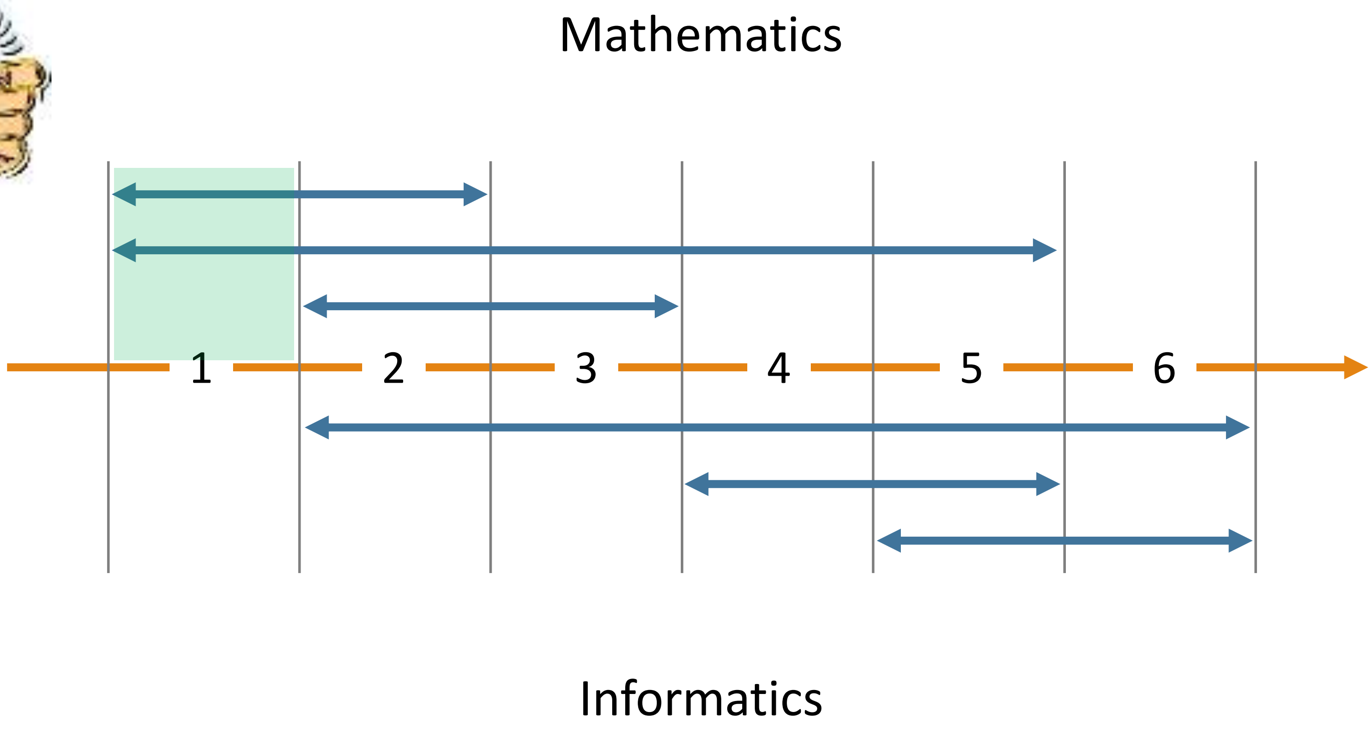
- parity of integer part of x coordinate
- parity of integer part of y coordinate
- comparing fractional part of x and y
- No iteration is needed: $O(1)$

H: Homework

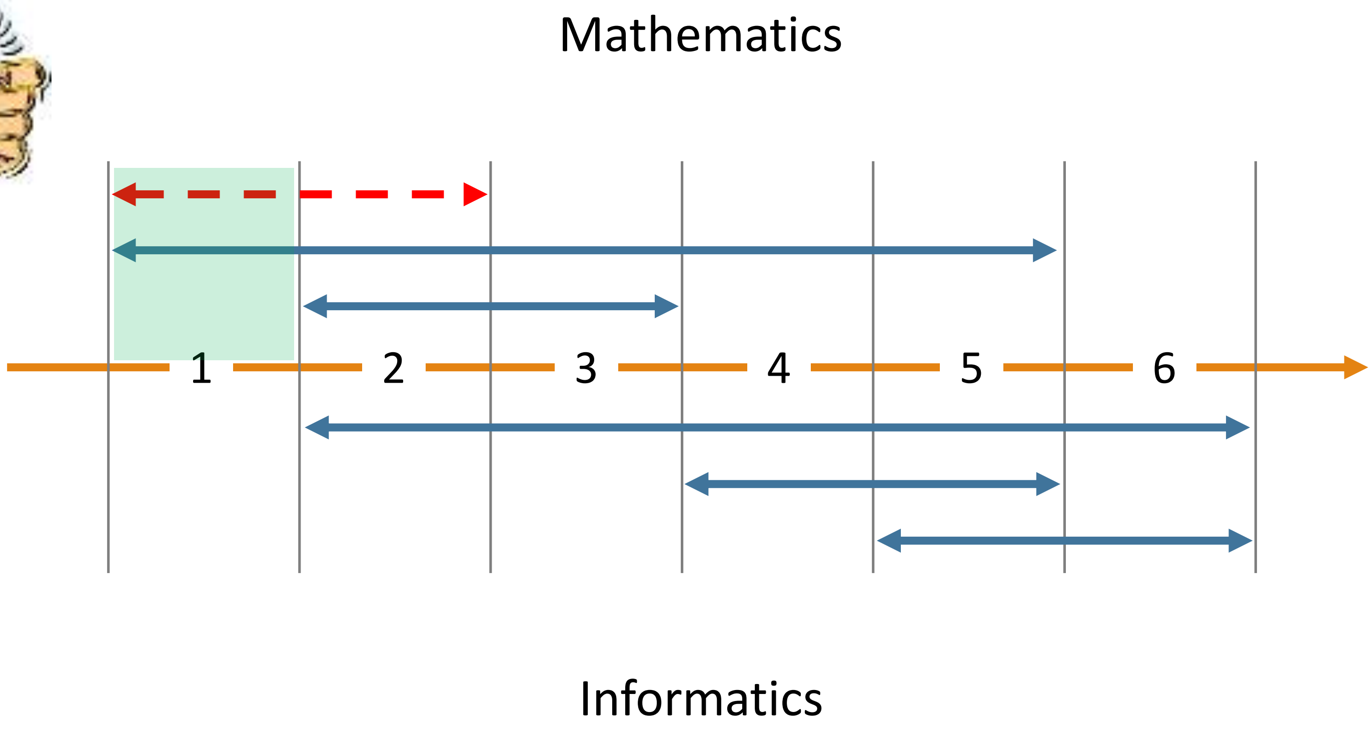
Problem Summary



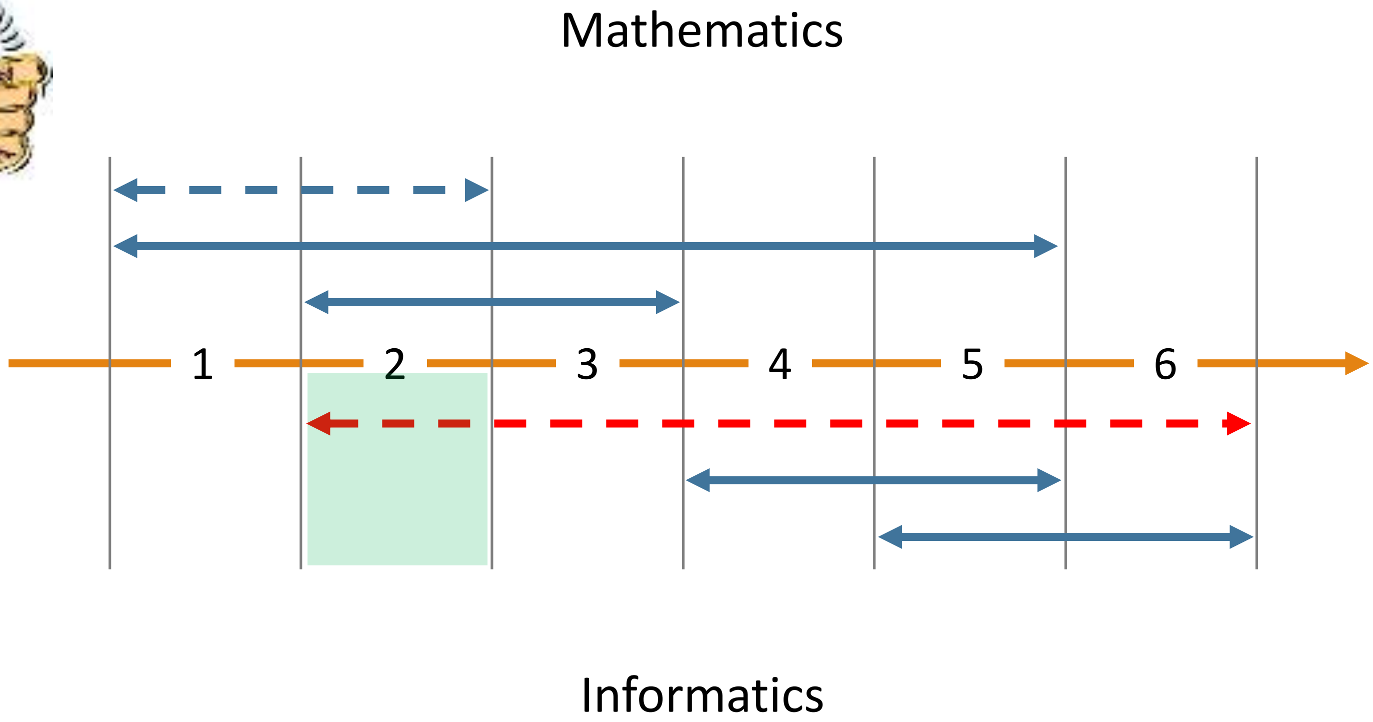
Problem Summary



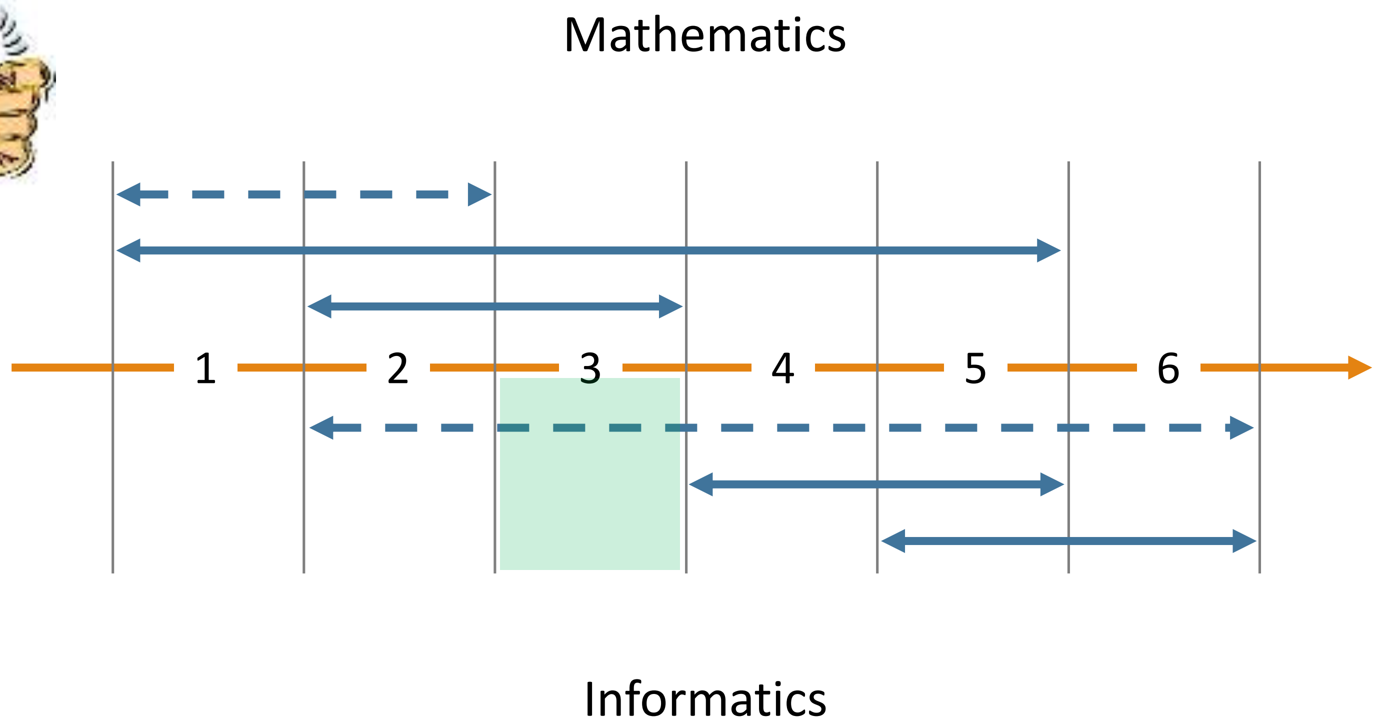
Problem Summary



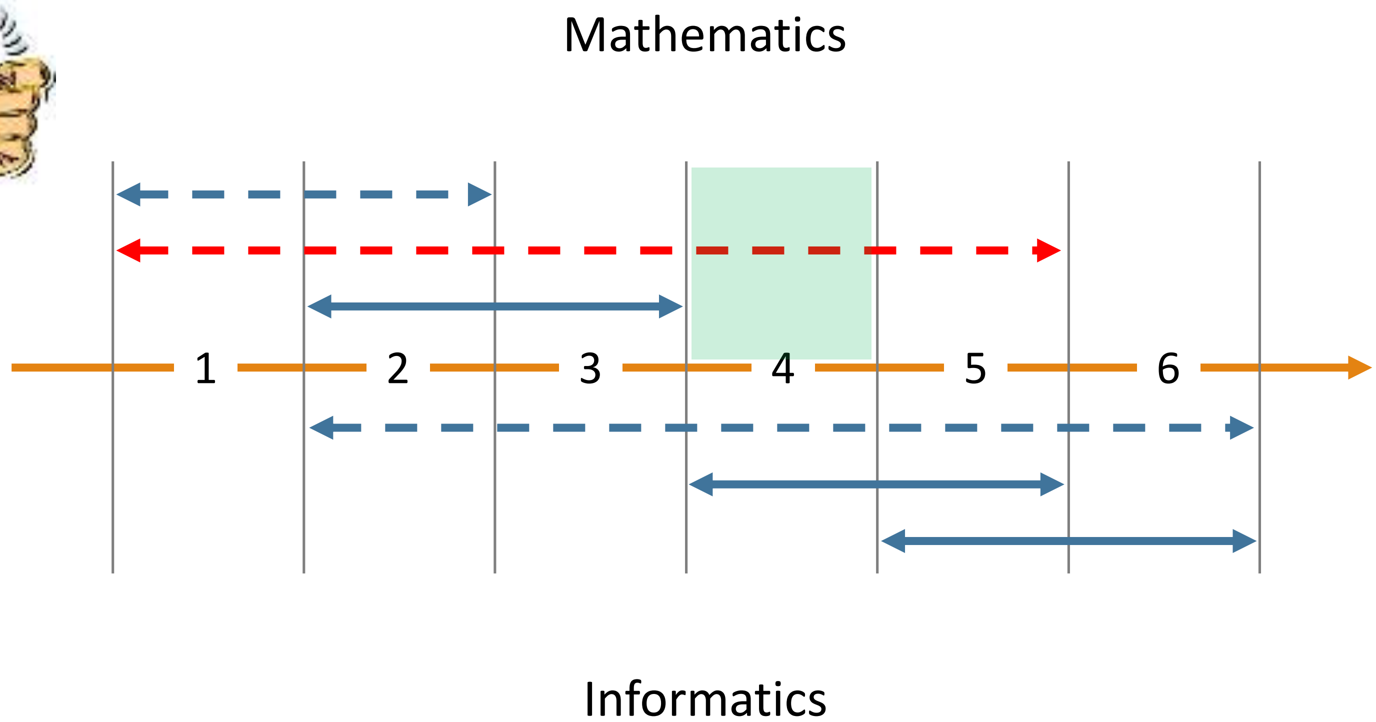
Problem Summary



Problem Summary

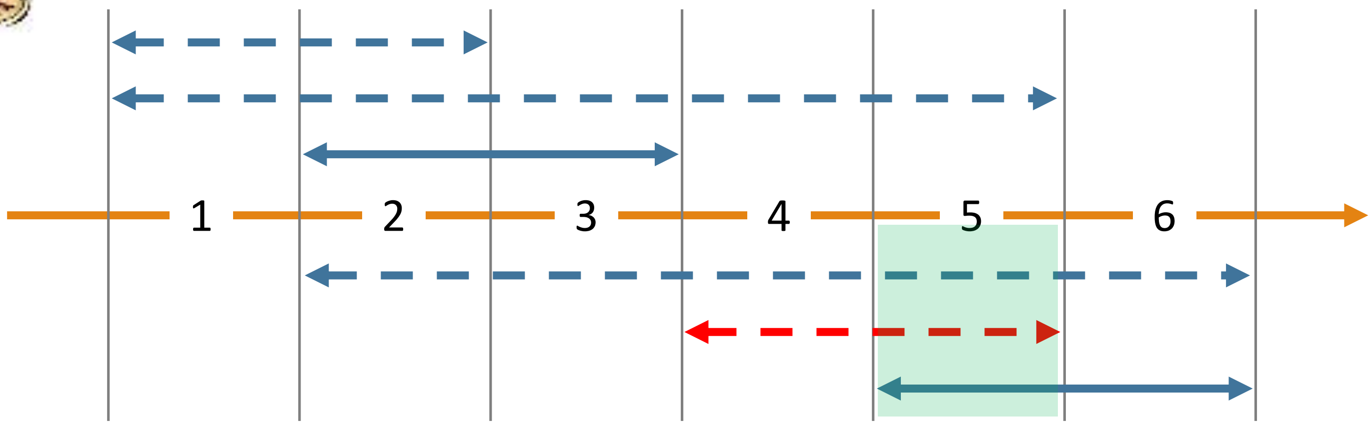


Problem Summary



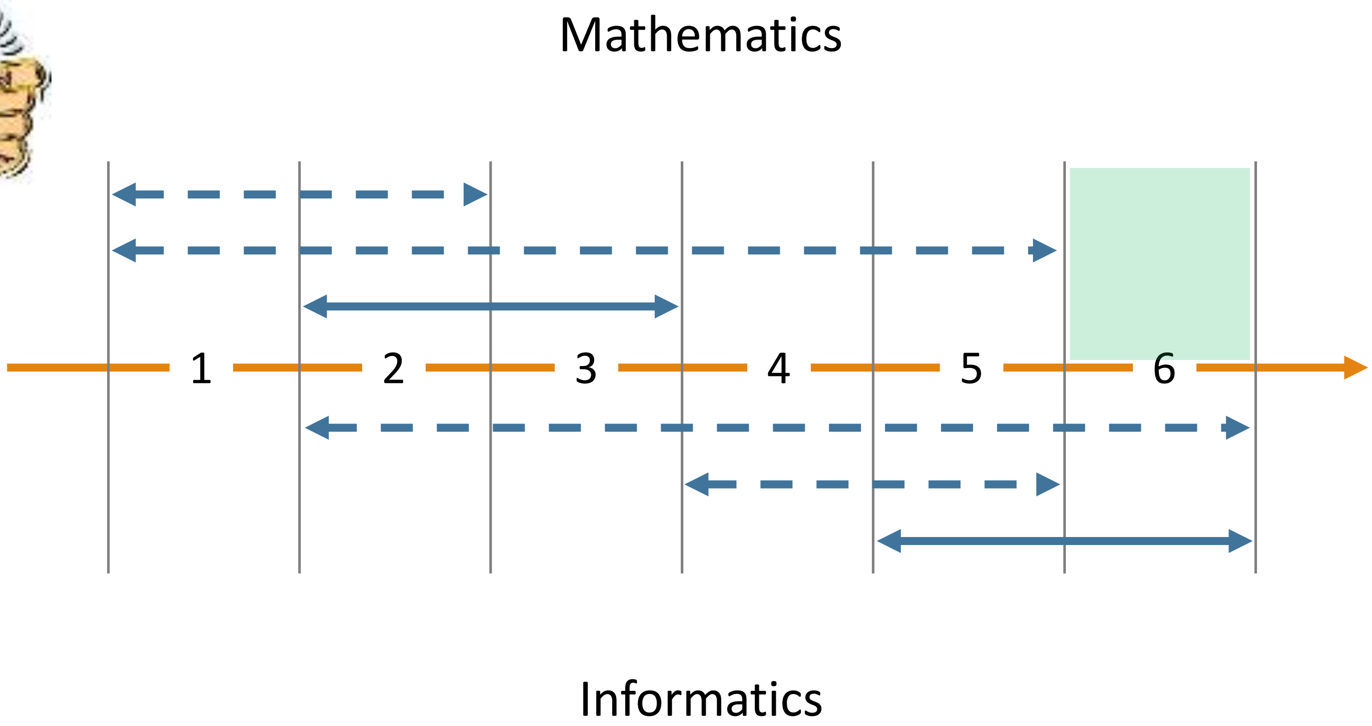
Problem Summary

Mathematics



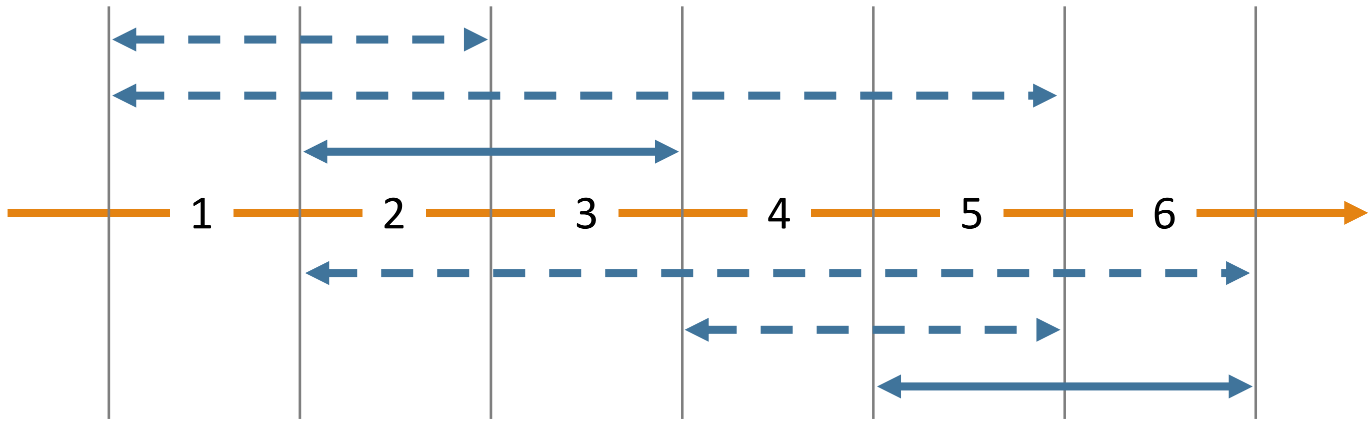
Informatics

Problem Summary



Problem Summary

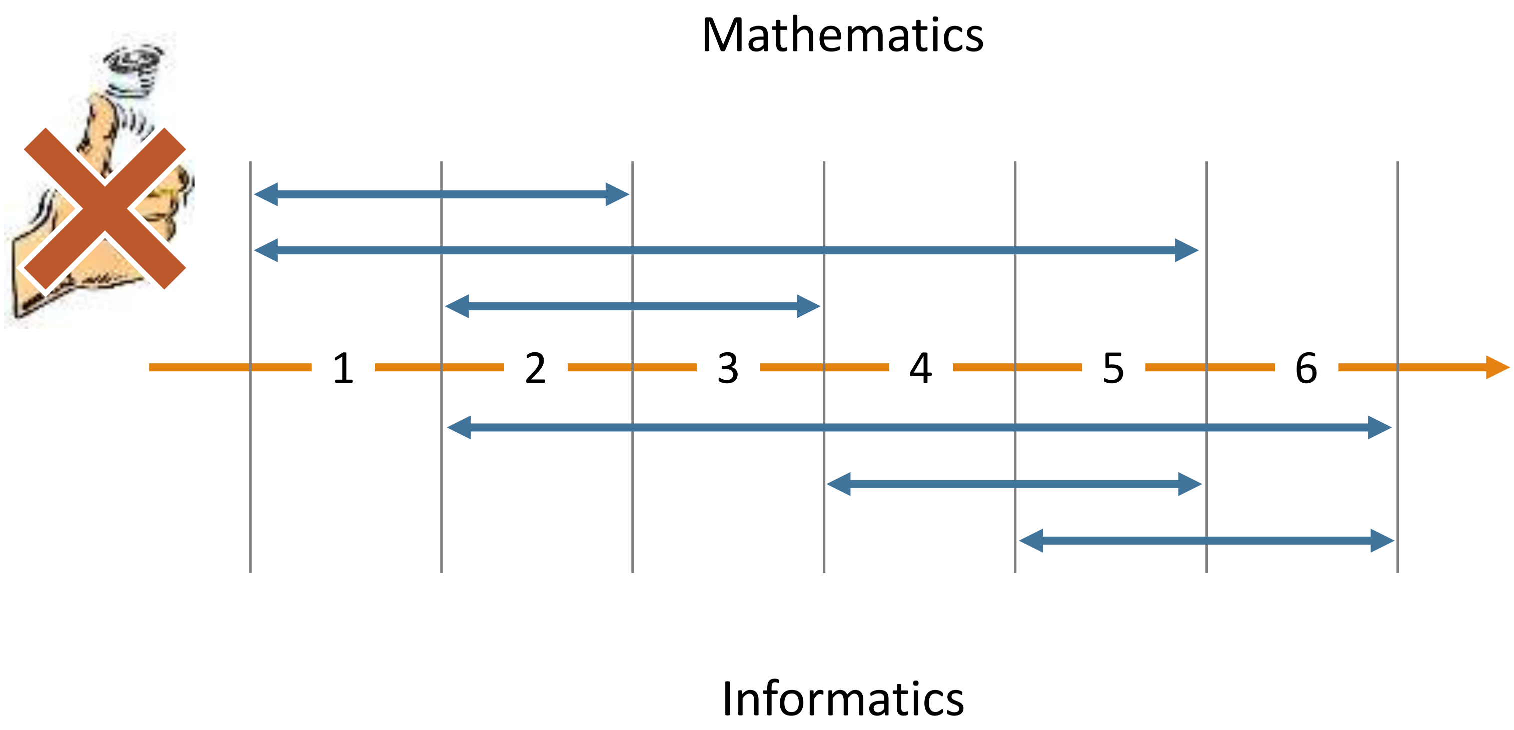
He has completed 4 assignments.



#Assignments he completes depends on the coin flips.
What is the maximum/minimum?

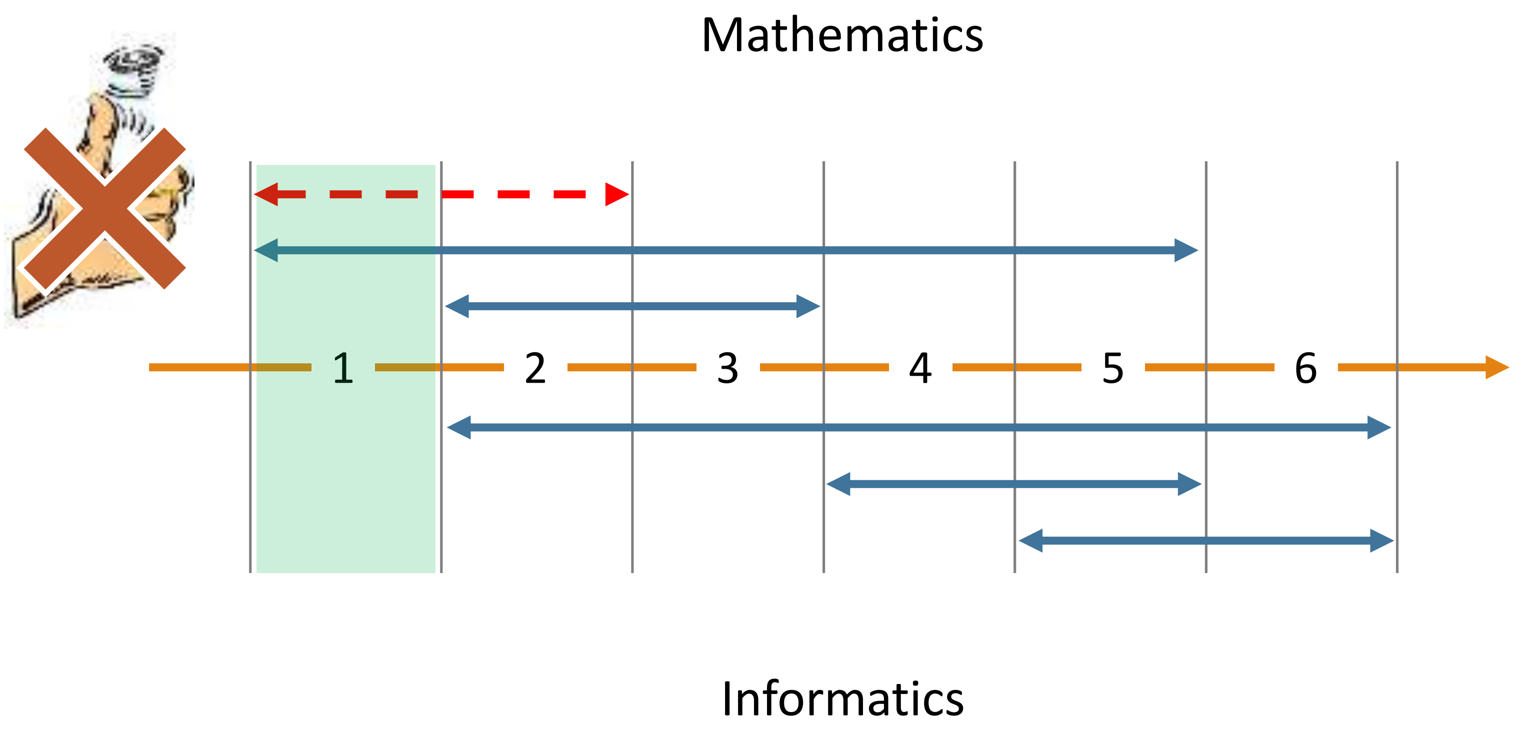
Maximum (Easy)

A simple greedy algorithm works.



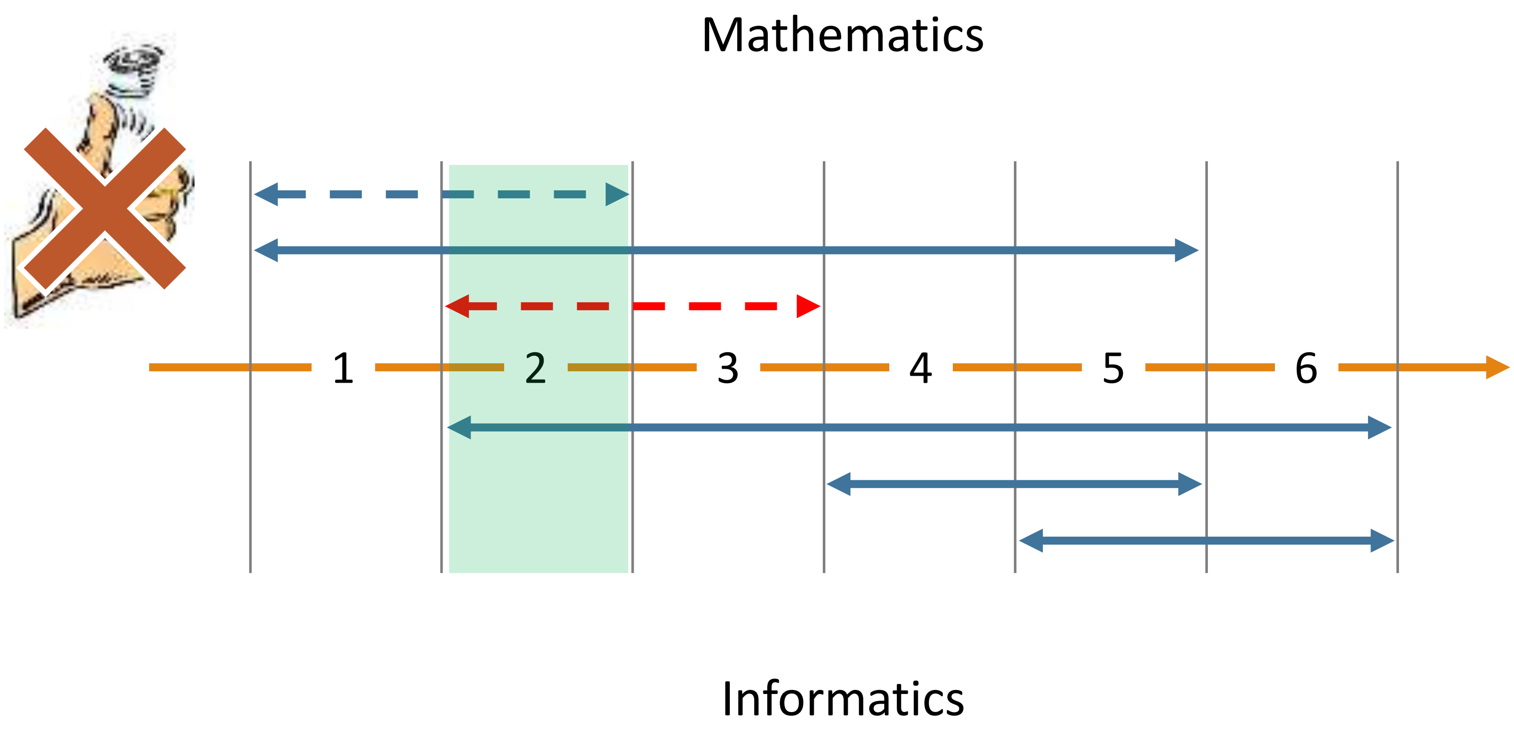
Maximum (Easy)

A simple greedy algorithm works.



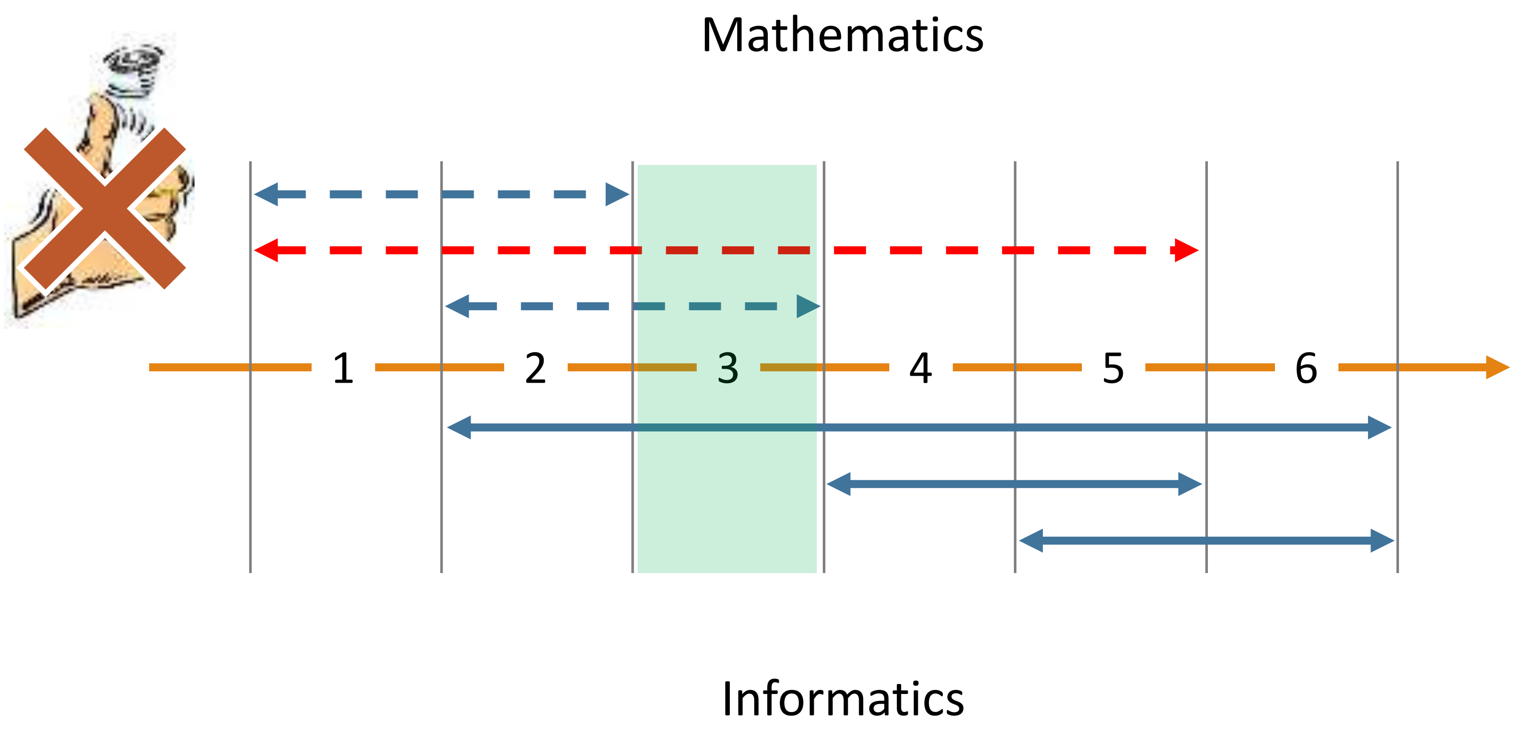
Maximum (Easy)

A simple greedy algorithm works.



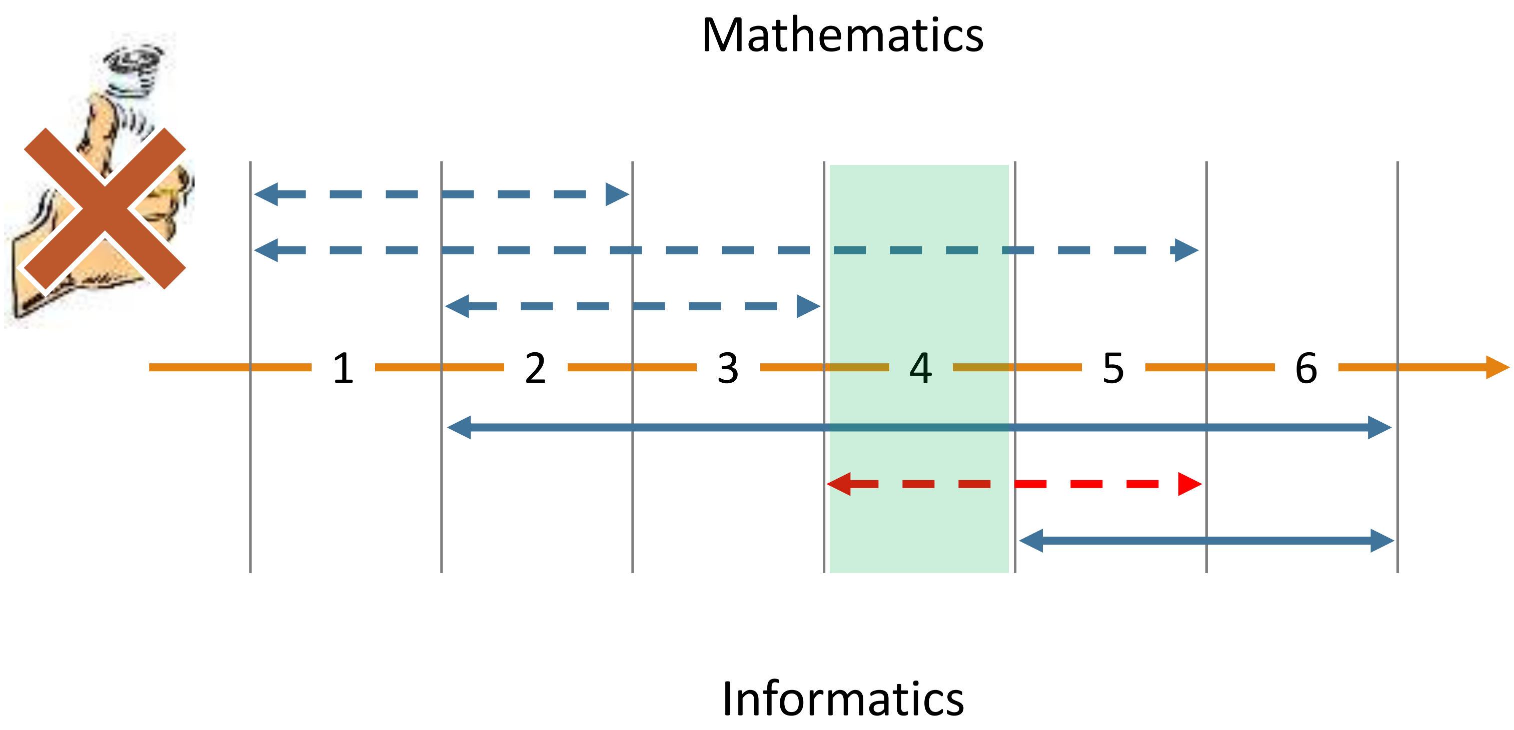
Maximum (Easy)

A simple greedy algorithm works.



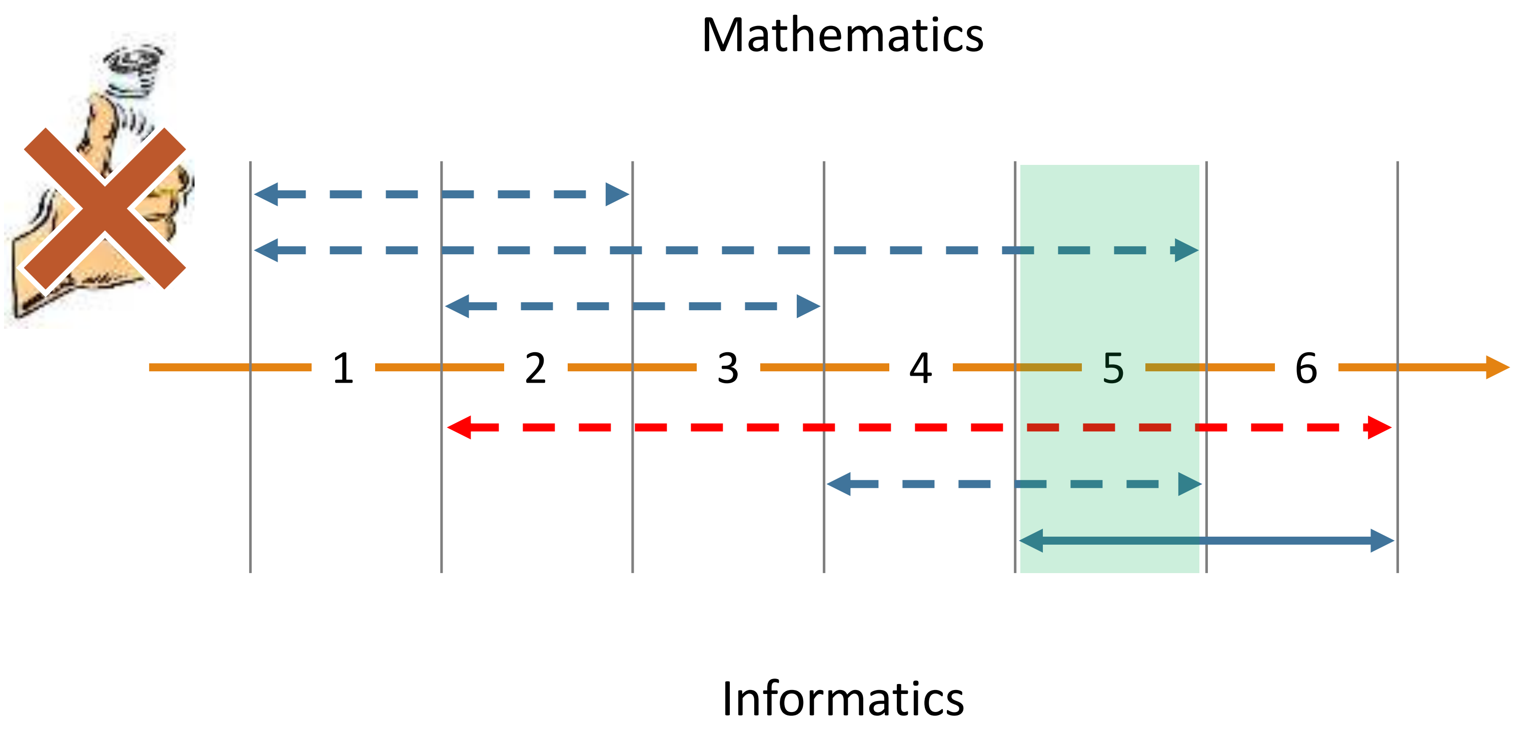
Maximum (Easy)

A simple greedy algorithm works.



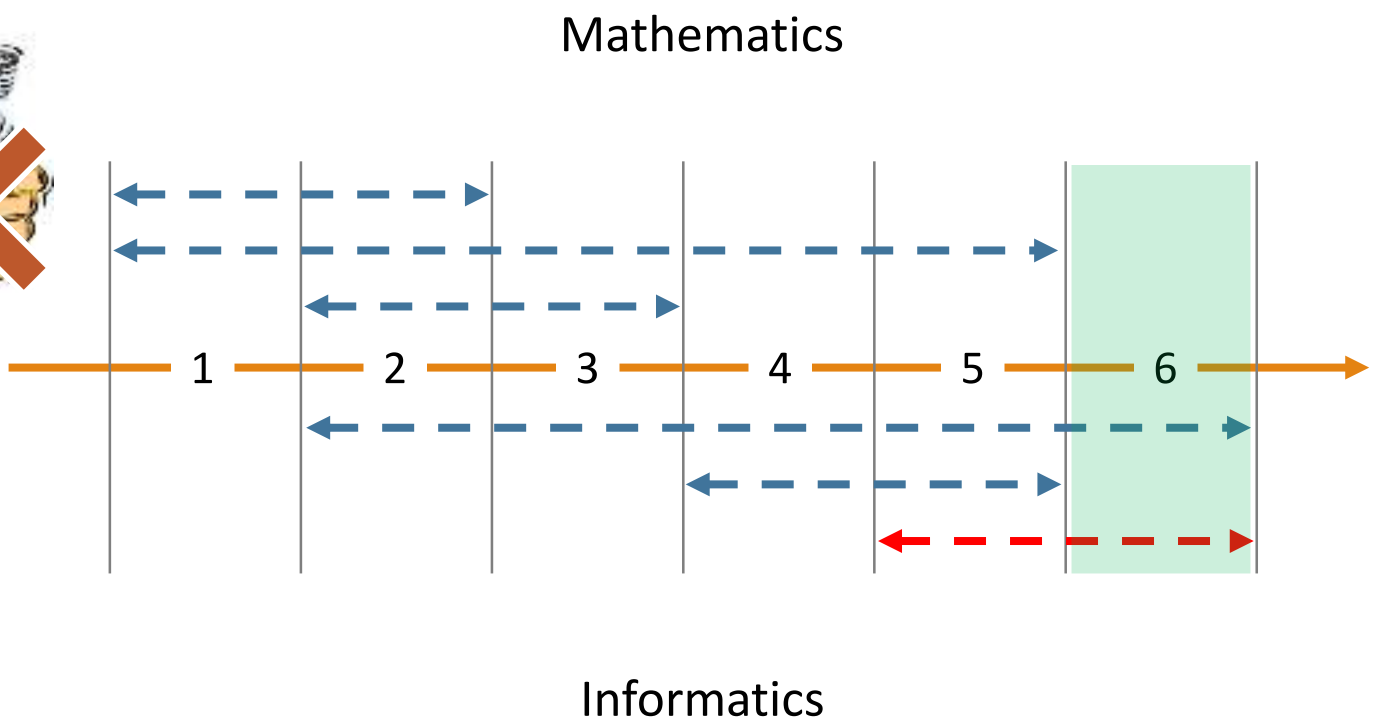
Maximum (Easy)

A simple greedy algorithm works.



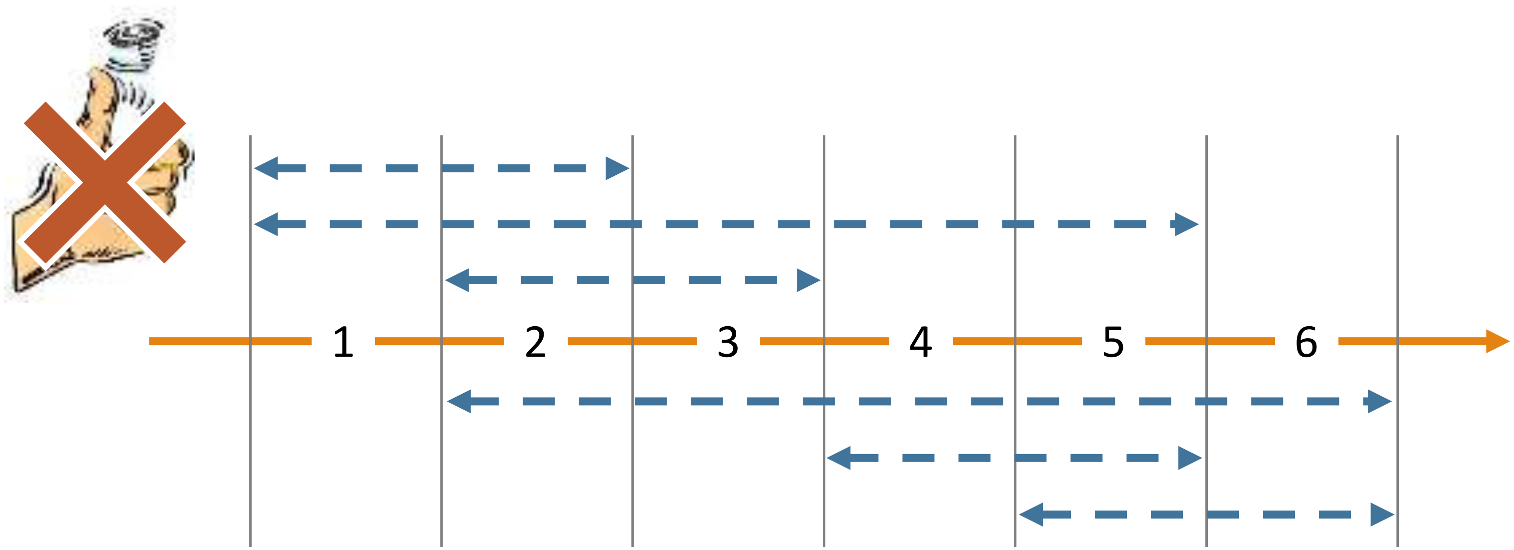
Maximum (Easy)

A simple greedy algorithm works.



Maximum (Easy)

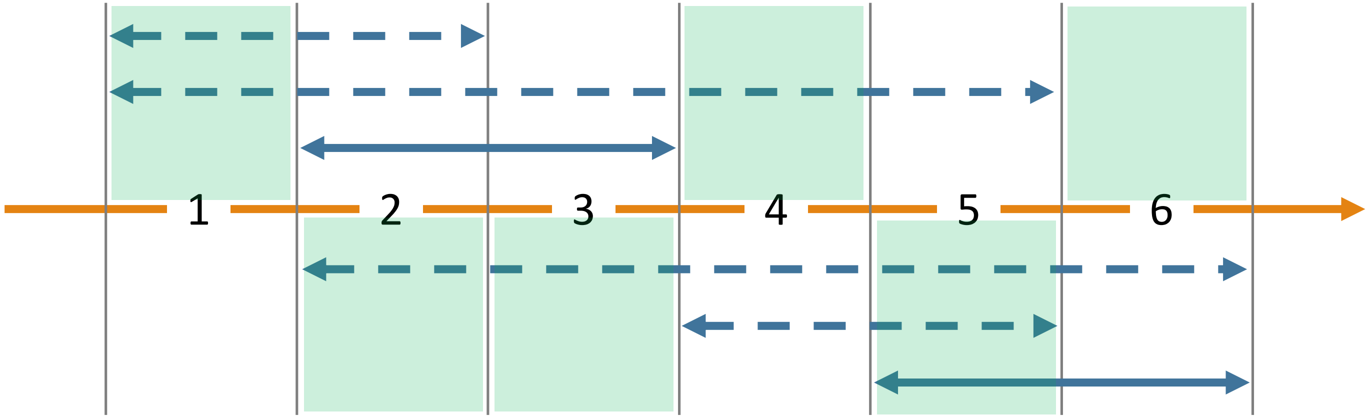
A simple greedy algorithm works.



He has completed 6 assignments.

Minimum (Difficult)

Key observation: His strategy is optimal.



Even if he can predict the future coin flips, he cannot complete more assignments.

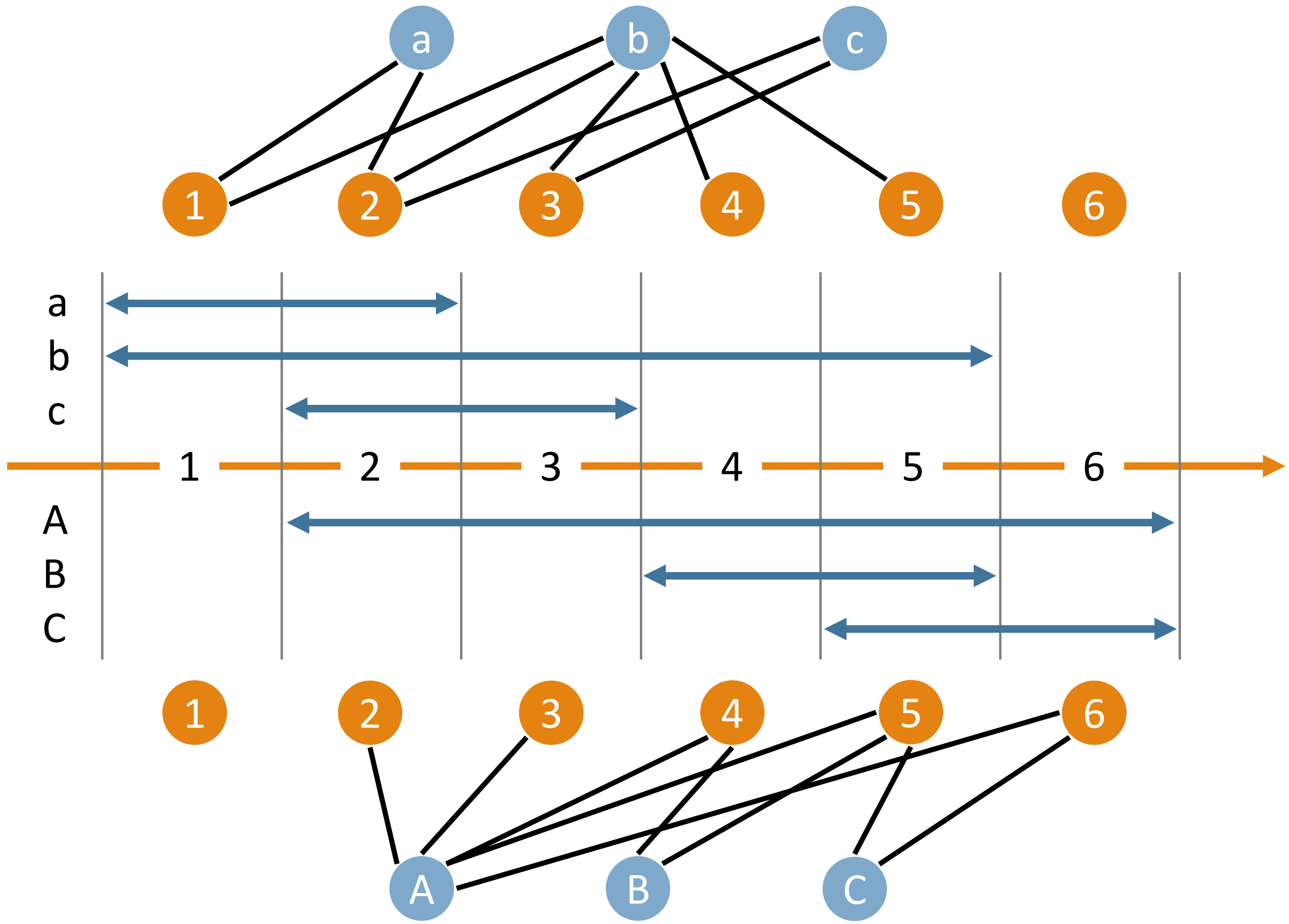
Minimum (Difficult)

$$\min_{\text{coin flips}} \text{\#completed assignments by his strategy}$$

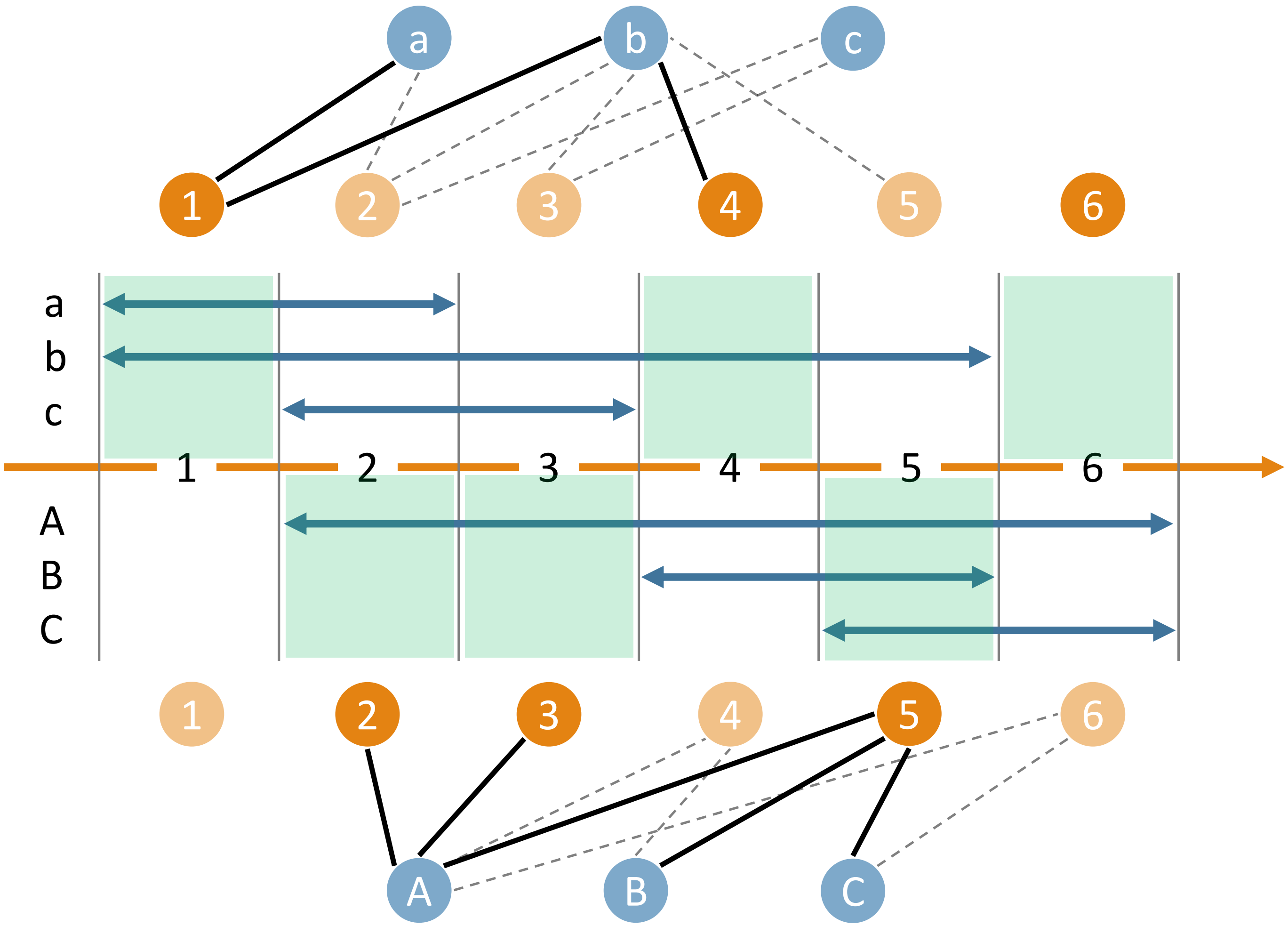
$$= \min_{\text{coin flips}} \max_{\text{scheduling}} \text{\#completed assignments}$$

Instead of using the greedy scheduling, we use the bipartite matching.

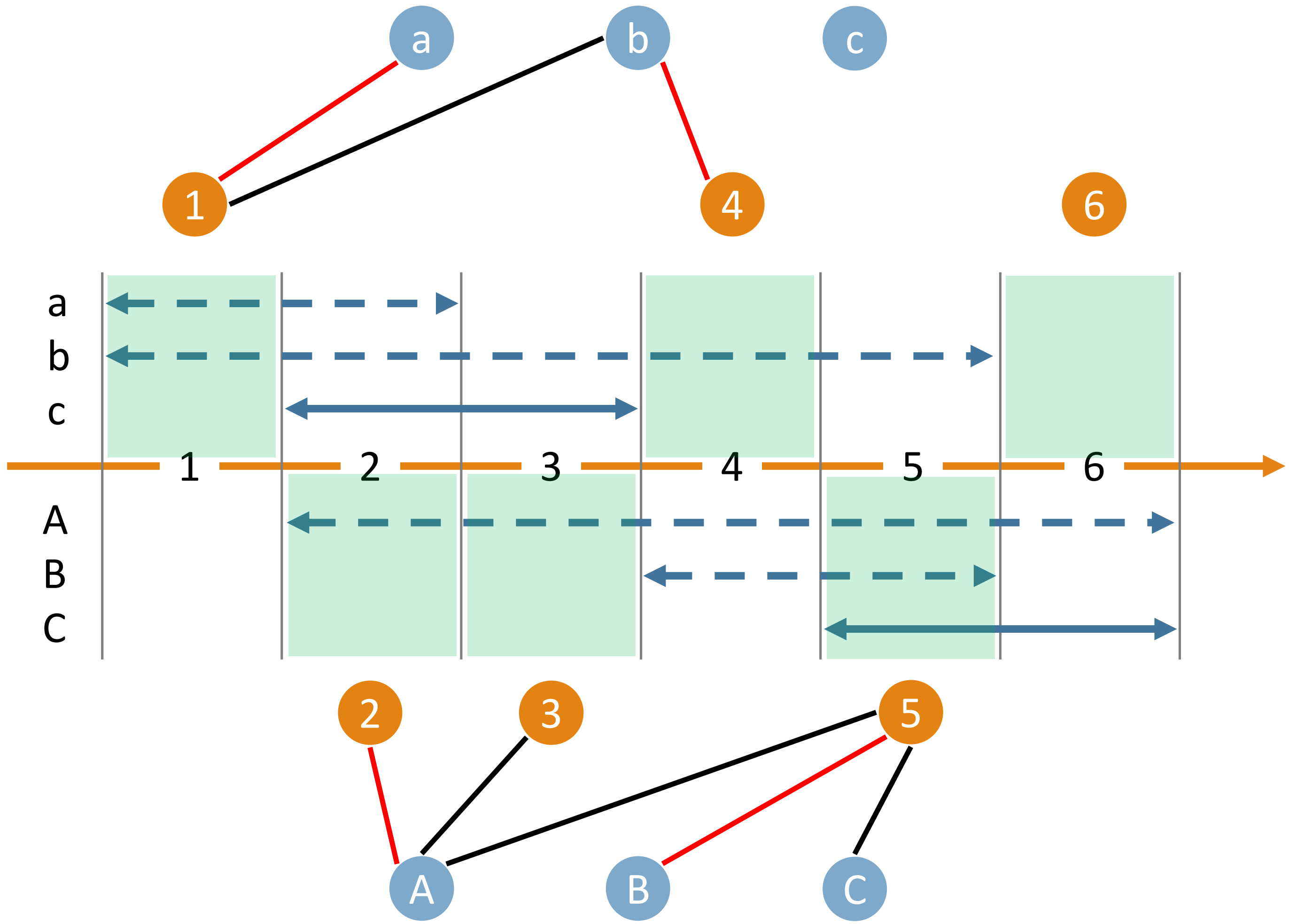
Bipartite matching



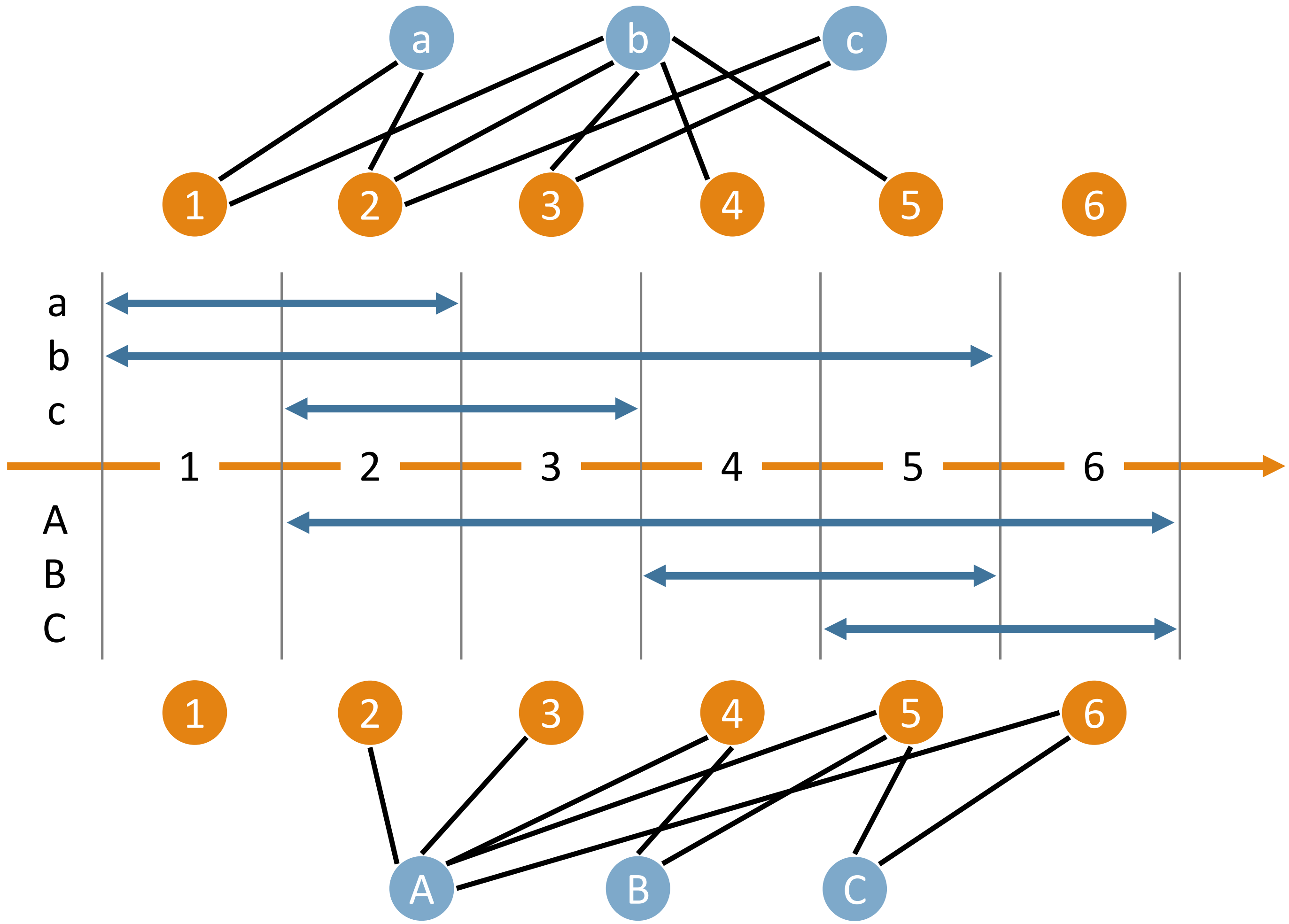
Bipartite matching



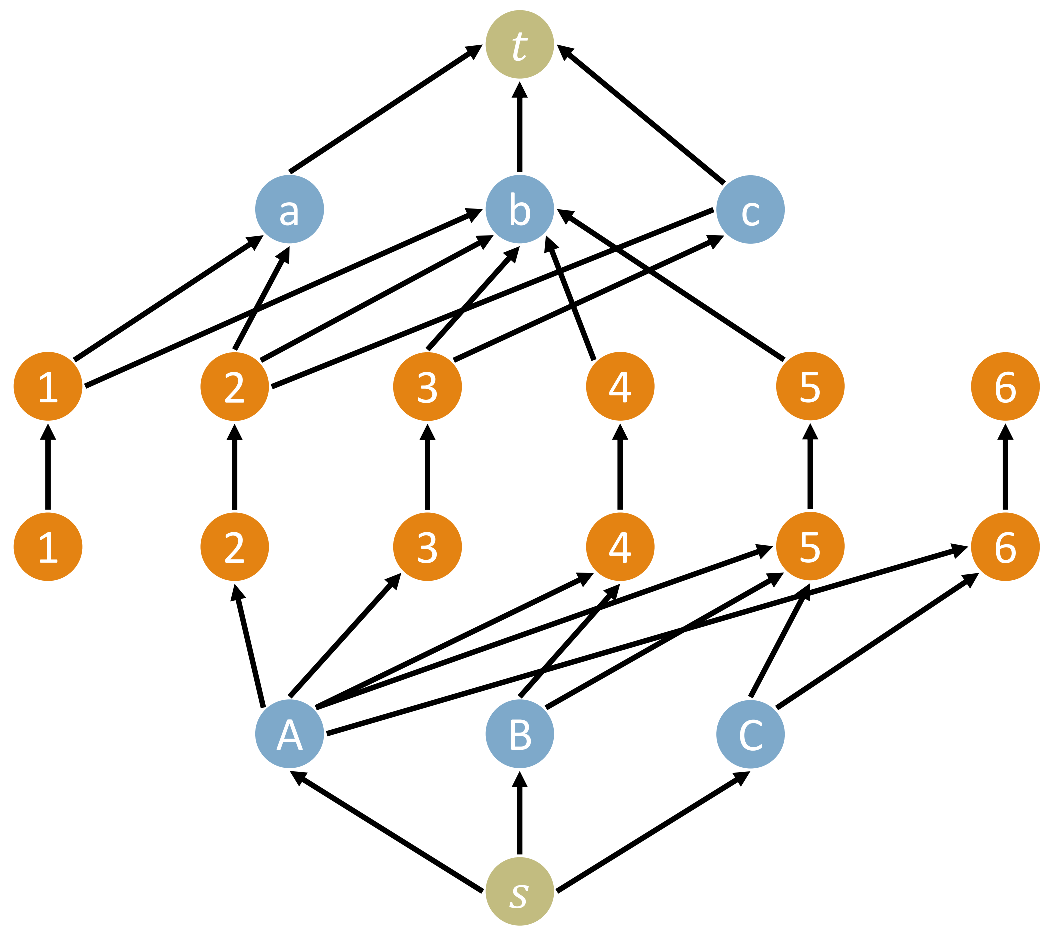
Bipartite matching



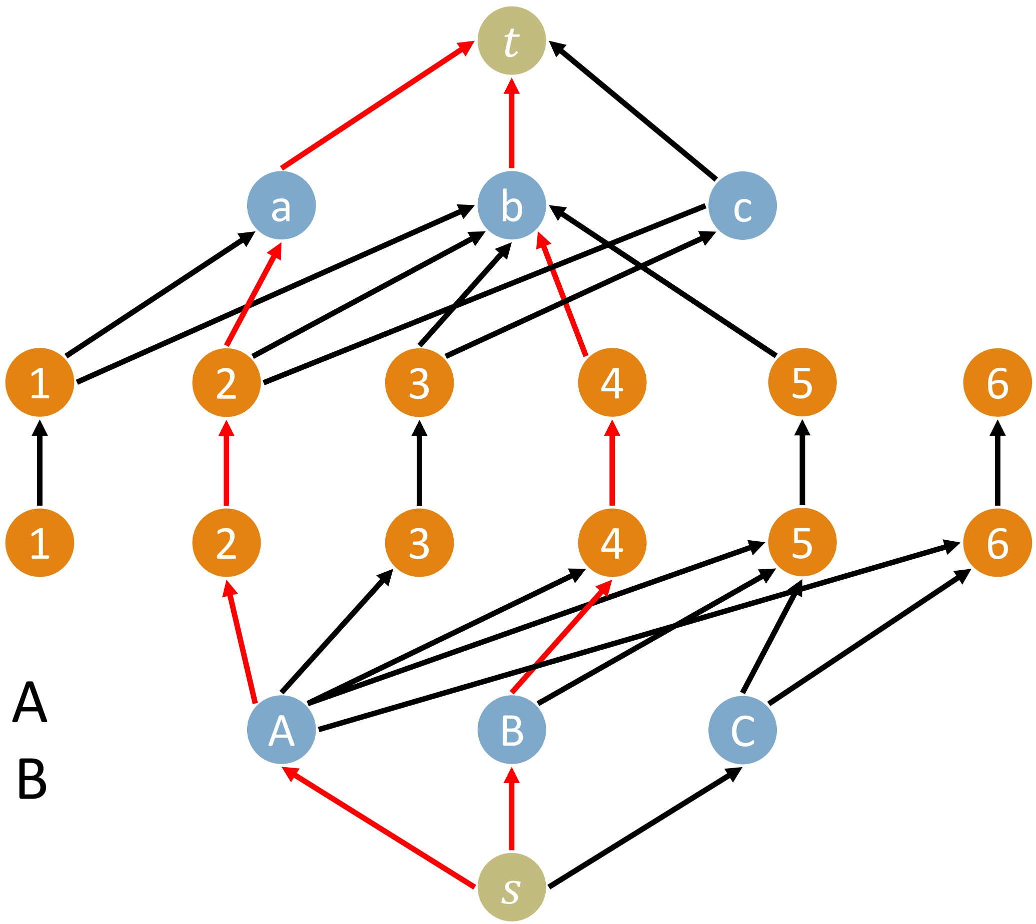
Connecting two graphs



Connecting two graphs

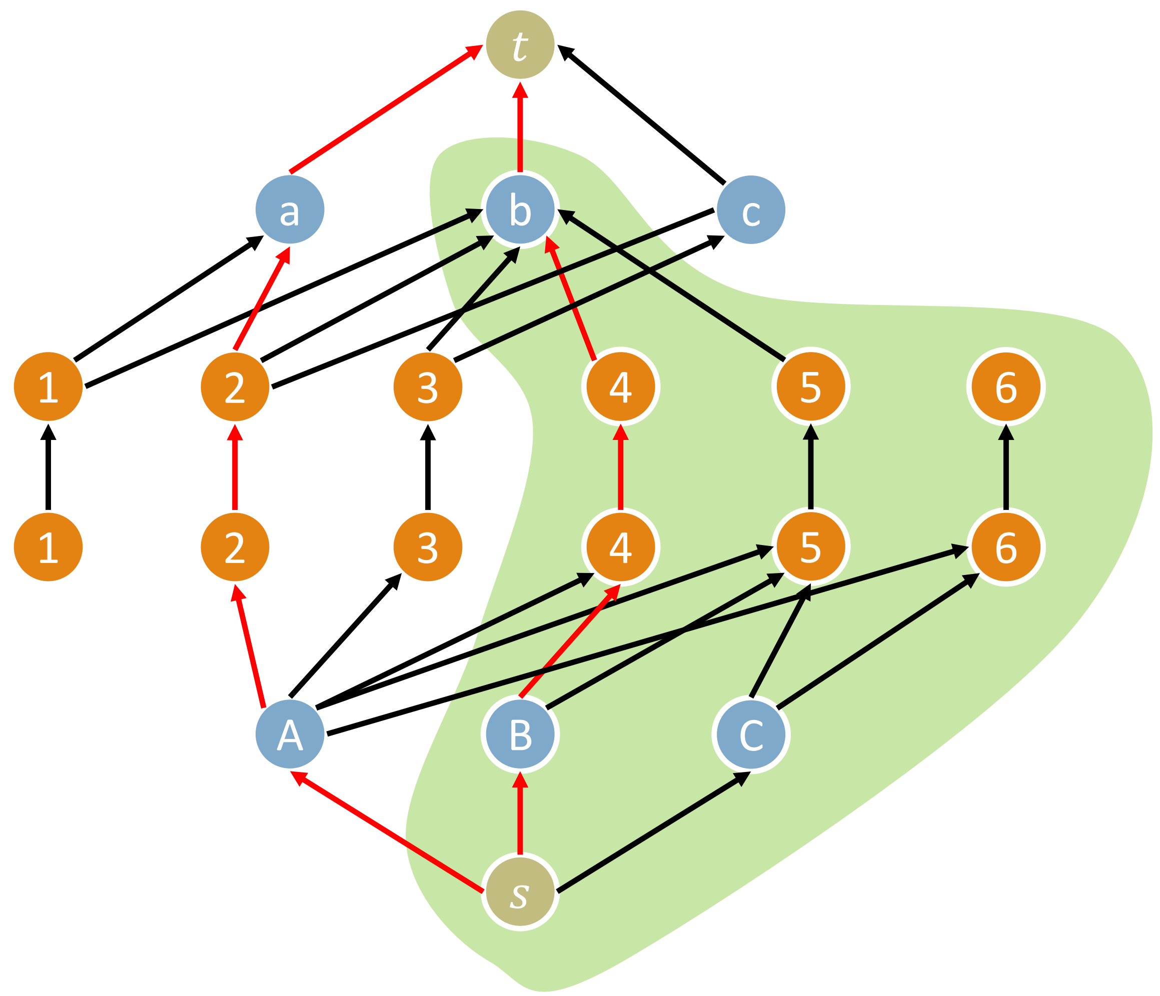


minimum ≥ maximum flow



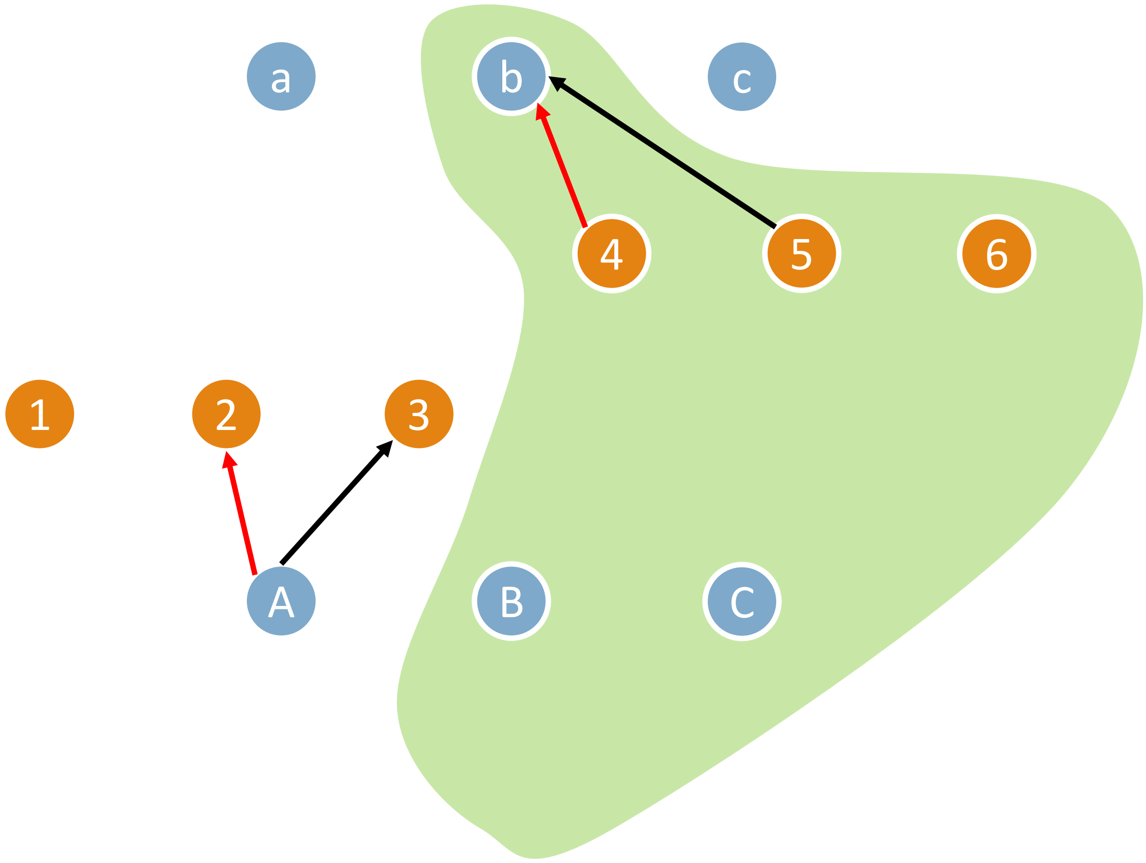
Day 2: a or A
Day 4: b or B

minimum ≤ maximum flow



minimum ≤ maximum flow

These are maximum matchings.



I: Starting a Scenic Railroad Service

Problem:

Plan the number of seats of a new tourist train.

There are two policies:
(P1) Each passenger can choose a preferable seat in the available ones.

(P2) Each passenger is assigned a seat by the railroad operator.

Key Point for Policy-1 (1/2)

For a passenger p, let $s(p)$ be the number of passengers whose travel sections overlap that of p.

The number of seats should be, at least, $s(p)$.

Reason: Assume that the reservation of p is the last one. If the number of seat is less than $s(p)$, all of the seats might be reserved by the other passengers. Thus, there may be no seats for p.

Key Point for Policy-1 (2/2)

Answer:

$s1 = \max\ s(p)$ for all passenger p,

Algorithm: $t(p)$ is computed easily.

$s(p) = N - t(p)$

N: [total number of passengers]

$t(p)$: [passengers whose sections do not overlap that of p]

$t(p)$= [alight before p] ∪ [board after p]

Key Point for Policy-2

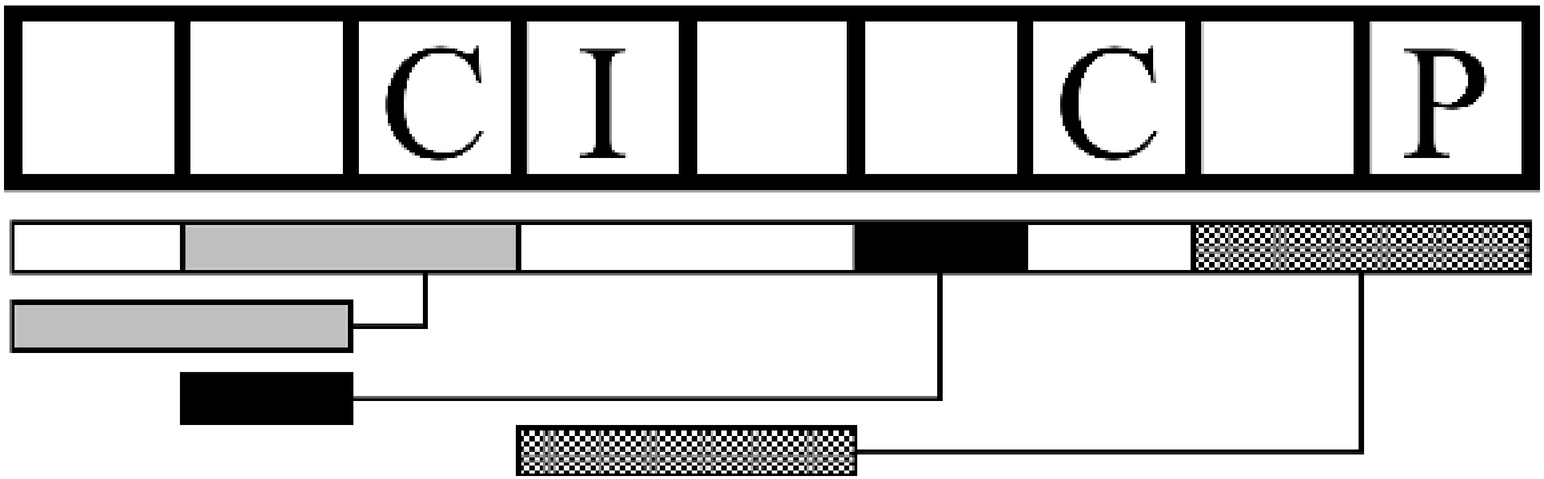
Answer:
$s2$ is the maximum number of passengers whose travel sections overlap each other.

Reason:
if the number of seats less than $s2$, there are a passenger with no seat.

Algorithm:
Count the maximum number of passengers for all stations.

J: String Puzzle

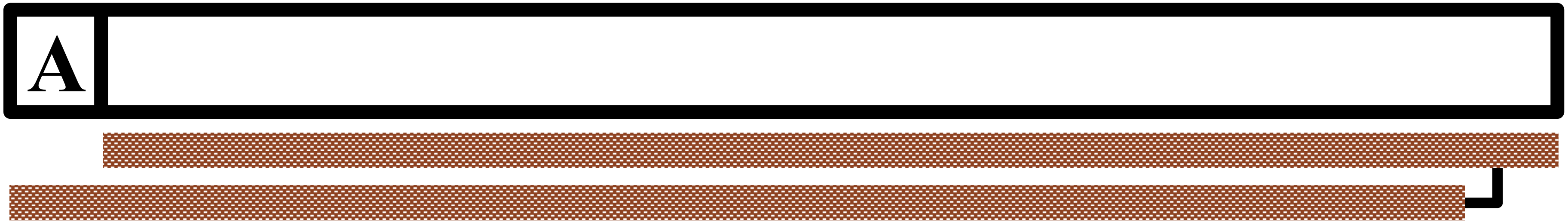
Problem Summary



- Letters at some positions of a secret string, and
- some info on **identical substrings**

are known. Guess the letters in other positions!

Hint on Overlapping Substrings: It's Powerful



Example:

"The substring of the range [1 .. 10^9-1] and that of [2 .. 10^9] are the same."

Hint on Overlapping Substrings: It's Powerful

- Char at [1] and [2] are the same.

Hint on Overlapping Substrings: It's Powerful

- Char at [1] and [2] are the same.
- Char at [2] and [3] are the same.

Hint on Overlapping Substrings: It's Powerful

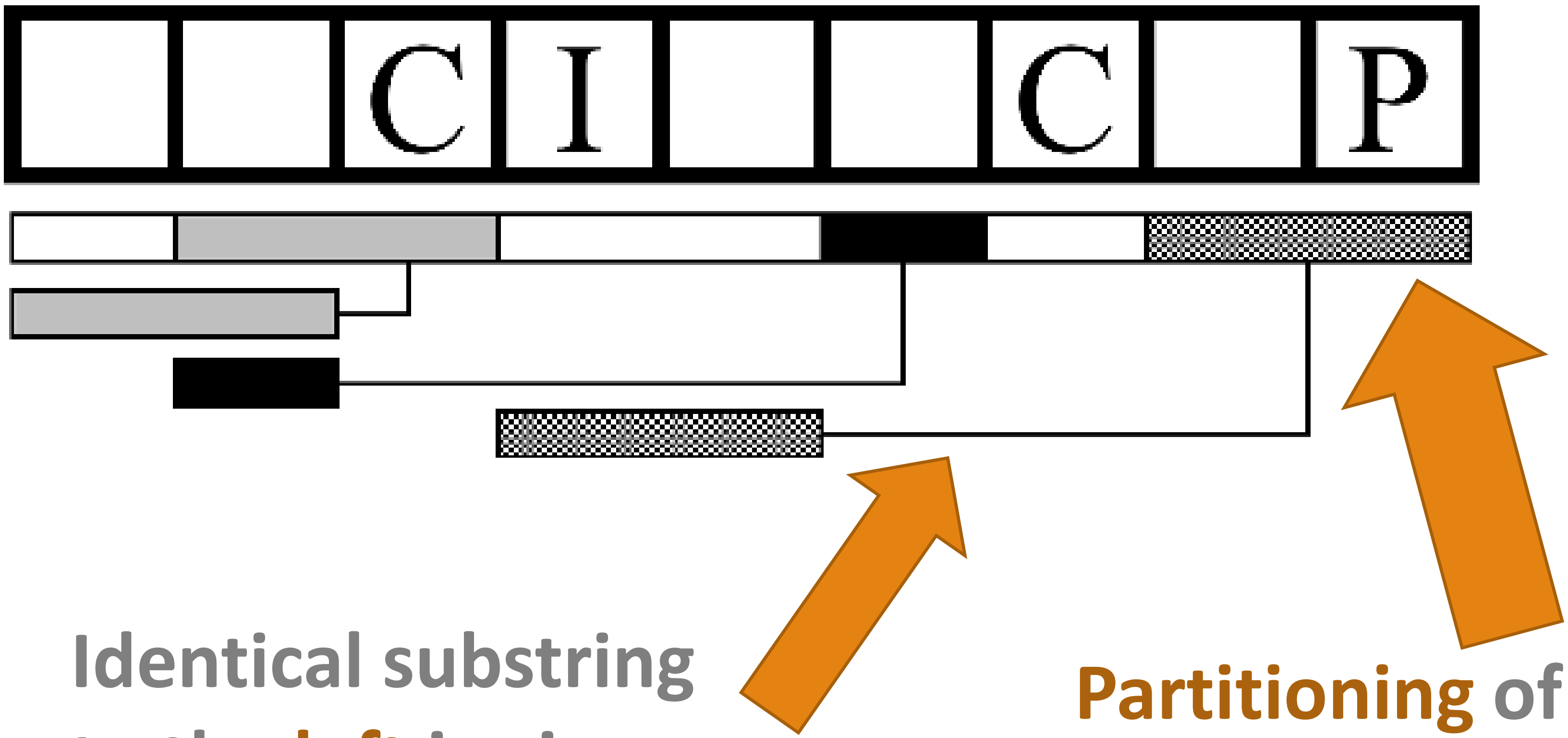
- Char at [1] and [2] are the same.
- Char at [2] and [3] are the same.
- …
- Char at [10^9-1] and [10^9] are the same.

Hint on Overlapping Substrings: It's Powerful

Single hint may reveal all the 10^9 letters of the string.

➔ Infeasible to propagate all info to every position by BFS, Union-Find, etc.

Key Observation



Identical substring to the left is given

Partitioning of the secret string

Solution:
Canonicalize to the Leftmost Position

- Each position has at most one hint that goes to the left. So…
  - Copy each known character to the left most position traversing the hints.
  - For each query, travers the hints to the left most position and check the letter.
- O($|\#hint|^2$) running time

Background: LZ77 Compression

Storing the **"previous occurrence of the identical substring"** instead of bare characters is a very popular compression method. (Used in "zip" tool, etc.)

K: Counting Cycles

Problem:

Given an undirected graph $G = (V, E)$
Find the number of simple cycles

Conditions:
- G is connected
- $m \leqq n + 15$

Observation

Given an undirected graph $G = (V, E)$
Find the number of simple cycles

Conditions:
- G is connected
- $m \leqq n + 15$

These conditions imply that

G is a tree + k additional edges
($k \leqq 16$)

Upper-bound of #Cycles

- Each additional edge creates a cycle (called a "**fundamental cycle**")
- Any cycle is generated by taking **XOR** of some fundamental cycles
(see: "**cycle space**", "**cycle basis**")

Thus, the number of cycles is at most 2^k; hence, we can solve this problem if **the complexity of finding each cycle is reduced!**

Reducing complexity

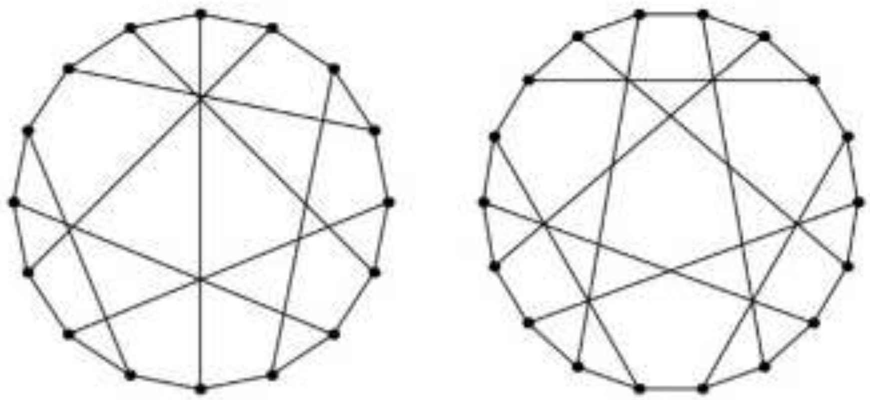
Standard enumeration algorithm requires O(*nm*) or O(n+m) per cycle, which is too expensive for *n* = 100,000

- **Contracting vertices with degree at most two** does not affect the solution
- Resulting graph has at most ***2k* vertices**

After this preprocessing, any enumeration algorithm will work

Further Information

- Best-known #Cycles for k = 16 is 41400, which is attained by the Tutte—Coxeter graph and 8-cage



- Cycle enumeration in O(n+m) delay is presented by Johnson [1975]