Data Structures and Algorithms

(CS210A)

Semester I – **2014-15**

Lecture 37

- **A new algorithm design paradigm: Greedy strategy**

  part IV

Problems solved till now

1. Job Scheduling Problem
2. Mobile Tower Problem
3. MST

Ponder over this question before moving ahead

Problem 1

Job scheduling Problem

INPUT:

- A set J of n jobs $\{j_1, j_2, \ldots, j_n\}$
- job j_i is specified by two real numbers

  s(i): start time of job j_i

  f(i): finish time of job j_i
- A single server

We could not say anything about the **complete solution** of this problem.

Constraints:

- Server can execute at most one job at any moment of time and a job.
- **Job j_i**, if scheduled, has to be scheduled during[**s**(i), **f**(i)] only.

Aim:

To select the **largest** subset of **non-overlapping** jobs which can be executed by the server.

All that we could do was to make
a local observation

Let $x \in J$ be the job with earliest finish time.

Lemma1 : There exists an optimal solution for J in which x is present.

Let $J' = J \setminus \textbf{Overlap}(x)$

Lemma 1 gives very small information about the optimal solution ☹
How to use it to compute this solution ?

J(original instance)

Greedy step

J' (**smaller** instance)

Opt(J)

Lemma1

Opt(J')

Opt(J)= Opt(J') + 1 -- (i)

Equation (i) hints at recursive solution of the problem ☺

Theorem: $\mathbf{Opt}(J) = \mathbf{Opt}(J') + 1$.

- Proof has two parts

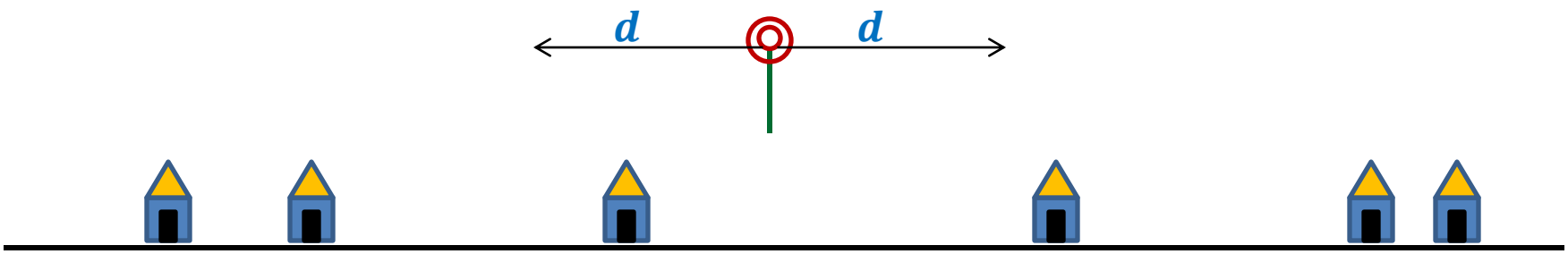
  $\mathbf{Opt}(J) \geq \mathbf{Opt}(J') + 1$

  $\mathbf{Opt}(J') \geq \mathbf{Opt}(J) - 1$

- Proof for each part is a proof by **construction**

Problem 2

Mobile Tower Problem



We could not say anything about the **complete solution** of this problem.

Problem statement:

There is a set H of n houses located along a road.

We want to place mobile towers such that

- Each house is **covered** by at least one mobile tower.
- The number of mobile towers used is **least** possible.

All that we could do was to make a local observation

Lemma 2: There is an optimal solution for the problem in which the leftmost tower is placed at distance d to the right of the first house.

Let x be the tower loca

Let H' = $H \setminus$ {all house

Lemma 1 gives very small information about the optimal solution ☹
How to use it to compute this solution ?

H(original instance)

Opt(H)

Opt(H)= **Opt**(H') + **1**

Greedy step

Lemma 2

H'(**smaller** instance)

Opt(H')

Equation **(i)** hints at recursive solution of the problem ☺

What is a greedy strategy ?

A strategy that is

- Based on some **local** approach
- With the **objective to optimize** some function.

Note:

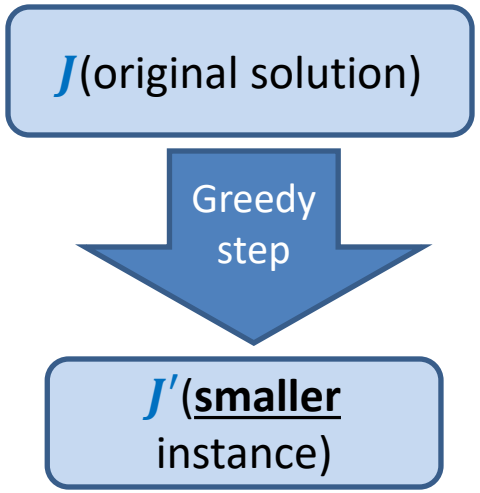
Recall that the divide and conquer strategy takes a **global approach**.

Design of a greedy algorithm

Let **A** be an instance of an optimization problem.

1. Make **a local observation** about the solution.

2. Use this observation to express optimal solution of **A** in terms of
   - Optimal solution of a smaller instance **A'**
   - Local step



3. This gives a recursive solution.
4. Transform it into iterative one.

MST

Input: an undirected graph $G=(V,E)$ with $w: E \rightarrow \mathbb{R}$,

Aim: compute a **spanning tree** (V, E'), $E' \subseteq E$ such that $\sum_{e \in E'} w(e)$ is **minimum.**

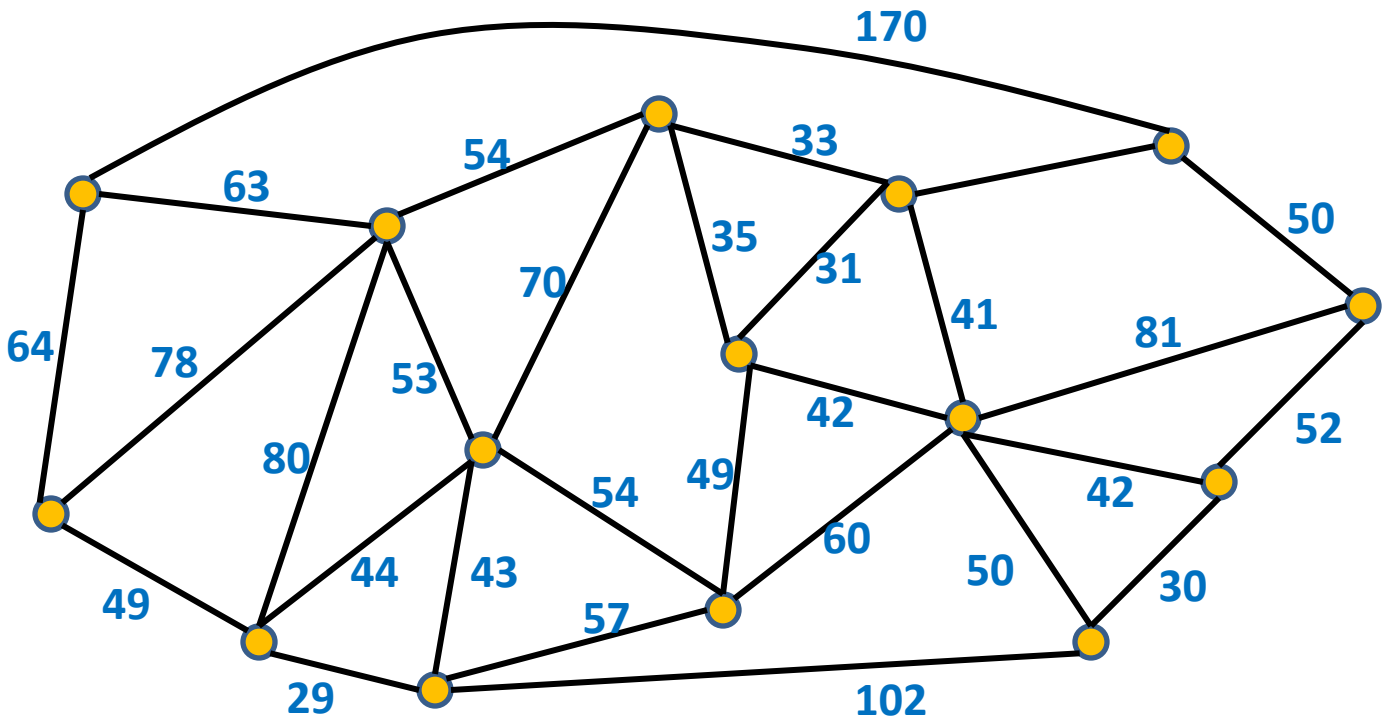
If you have understood a generic way to design a greedy algorithm, then try to solve the MST problem.

Lemma 2

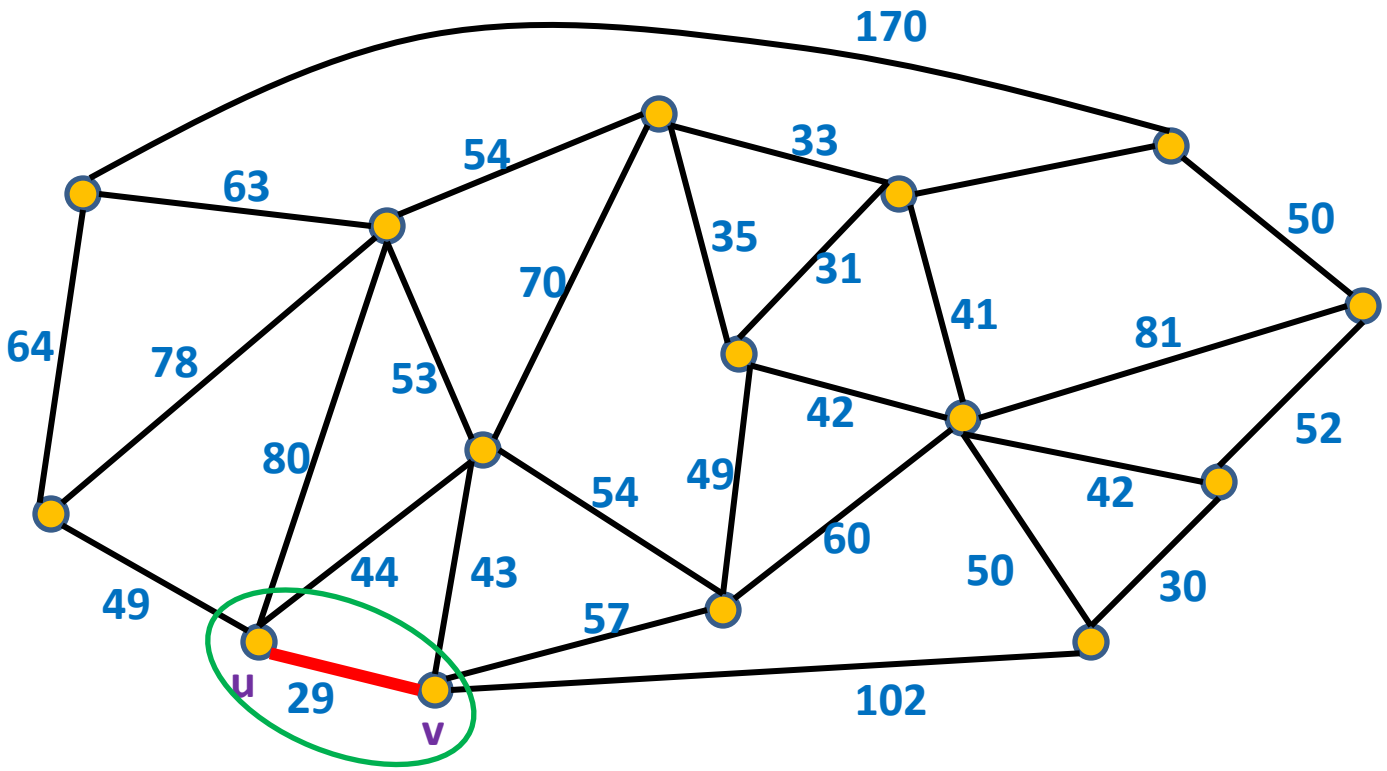
If $e_0 \in E$ is the edge of **least weight** in G, then there is a **MST** T containing e_0.

How to use this **Lemma** to design an algorithm for **MST** ?

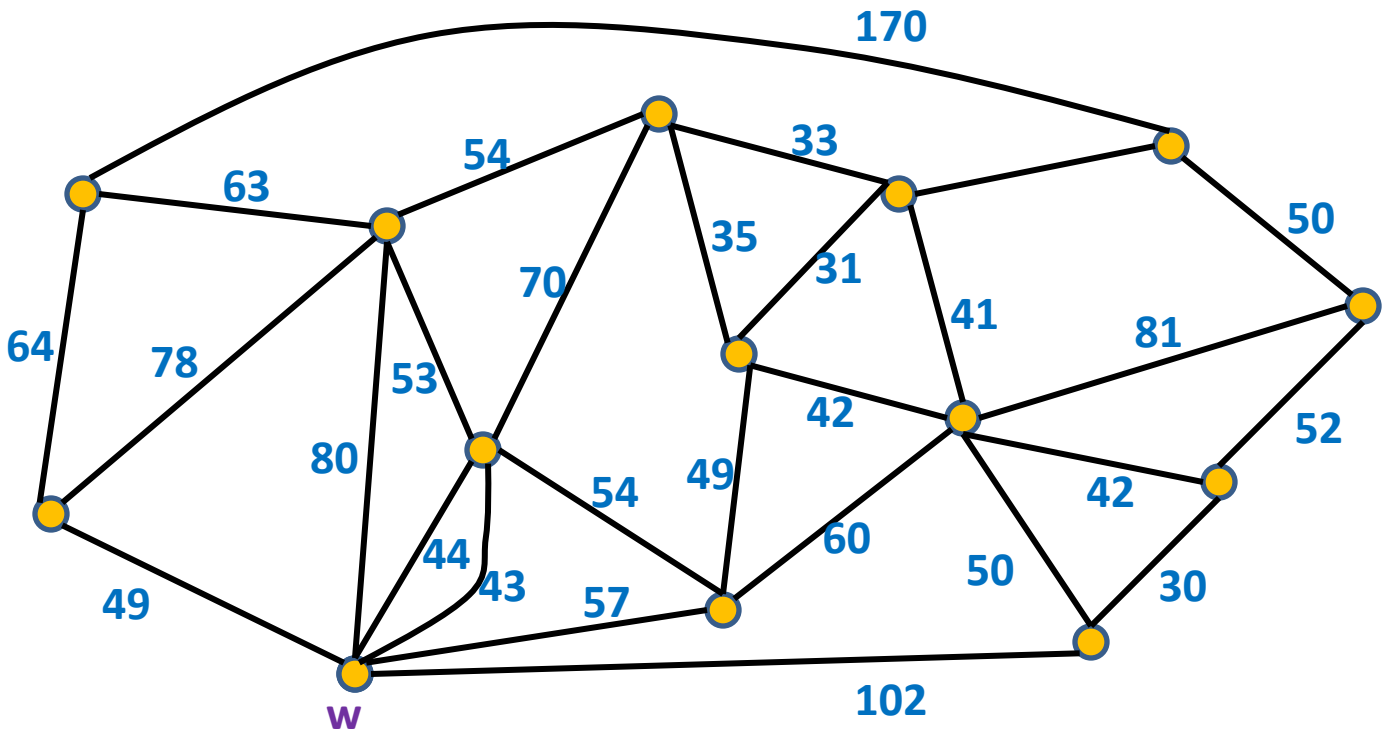
How to compute a MST ?



How to compute a MST ?



How to compute a MST ?



This is graph G'

How to compute a MST ?

Let (u,v) be the least weight edge in G=(V, E). Transform G into G' as follows.

- **Remove** vertices u and v and **add** a new vertex w
- For each edge (u,x)ϵ E, add edge (w,x) in G'.
- For each edge (v,x)ϵ E, add edge (w,x) in G'.
- In case of multiple edges between w and x, keep only the **lighter** weight edge.

Theorem1: **W_MST(G) = W_MST(G') + w(u,v)**

Proof: (by **construction)**

1. W_MST(G) ≤ W_MST(G') + w(u,v) ← straightforward

2. W_MST(G') ≤ W_MST(G) - w(u,v) ← Use **Lemma 3**

(Give all details of the proof as a homework)

Problem 4
Overlapping Intervals

The aim of this problem is to make you realize that

it is sometime very nontrivial to design a greedy algorithm. In particular, it is quite challenging to design the smaller instance.

In the end semester exam of the course, no problem of this level of difficulty will be asked ☺

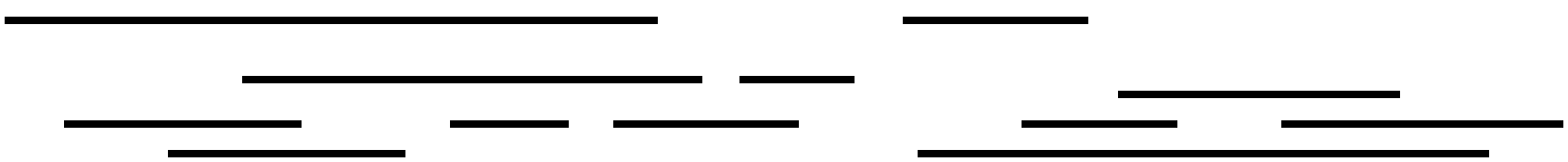
Overlapping Intervals

Problem statement:

Given a set **A** of n intervals, compute smallest set **B** of intervals so that

for every interval **I** in **A\B**, there is some interval in **B** which overlaps/intersects with **I**.

A



Overlapping Intervals

Problem statement:

Given a set **A** of n intervals, compute smallest set **B** of intervals so that

for every interval **I** in **A\B**, there is some interval in **B** which overlaps/intersects with **I**.

another optimal solution ☺

Overlapping Intervals

Strategy 1

Interval with **maximum length** should be there in optimal solution

Intuition:

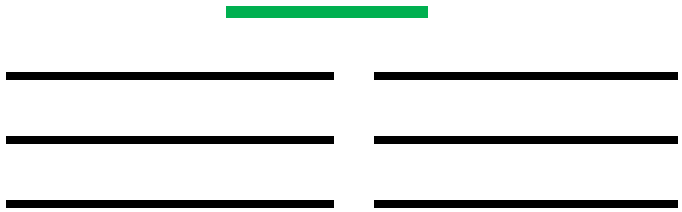
Selecting such an interval will **cover maximum** no. of other intervals

There is a counter example ☹

Overlapping Intervals

Strategy 1

Interval with **maximum length** should be there in optimal solution



Intuition:

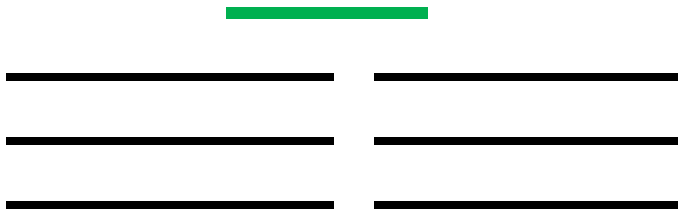
Selecting such an interval will **cover maximum** no. of other intervals

There is a counter example ☹

Overlapping Intervals

Strategy 2

Interval that **overlaps maximum no.** of intervals should be there in optimal solution



Intuition:

Selecting such an interval will **cover maximum** no. of other intervals

There is a counter example ☹

Overlapping Intervals

Strategy 2

Interval that **<u>overlaps maximum no.</u>** of intervals should be there in optimal solution

Not an optimal solution ☹

Overlapping Intervals

Strategy 2

Interval that **overlaps maximum no.** of intervals should be there in optimal solution

An optimal solution has size 2.

Think for a while :

After failure of two strategies, how to proceed to design the algorithm.

Overlapping Intervals

Let I* be the interval with earliest finish time.

Let I' be the interval with **maximum** finish time overlapping I*.

Lemma1: There is an optimal solution for set **A** that contains **I'.**

Proof:(sketch) :

If I* is overlapped by any other interval in the optimal solution, say I^,

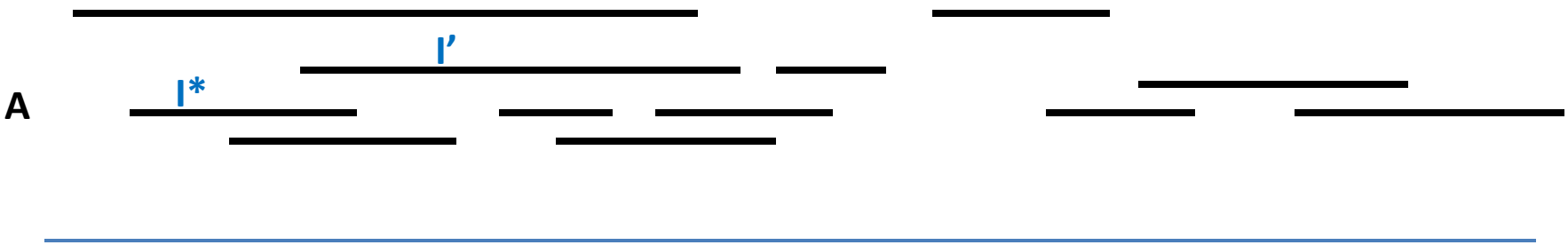
I^ will surely overlap all intervals that are overlapped by I'.

➔ Swapping I^ by I' will still give an optimal solution.

Exploit the fact that I* has earliest finish time for this claim.

Overlapping Intervals

Question: How to obtain smaller instance **A'** using **Lemma 1** ?

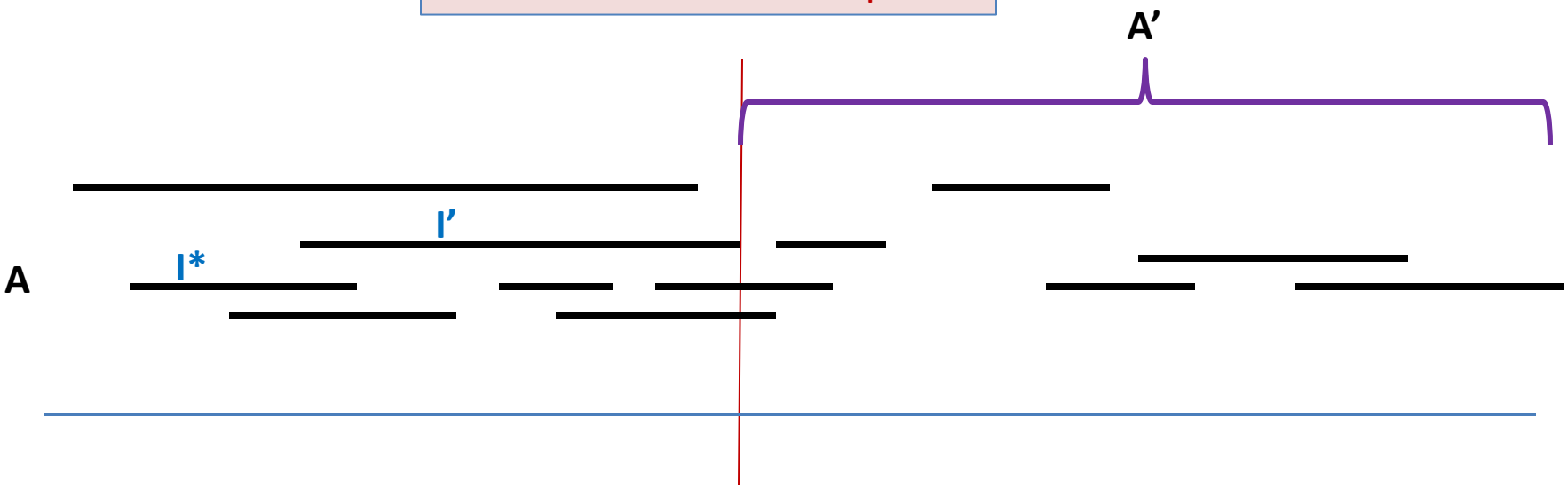


Overlapping Intervals

Question: How to obtain smaller instance **A'** using **Lemma 1** ?

Naive approach : remove from **A** all intervals which overlap with **I'**. This is **A'**.

There is a counter example ☹

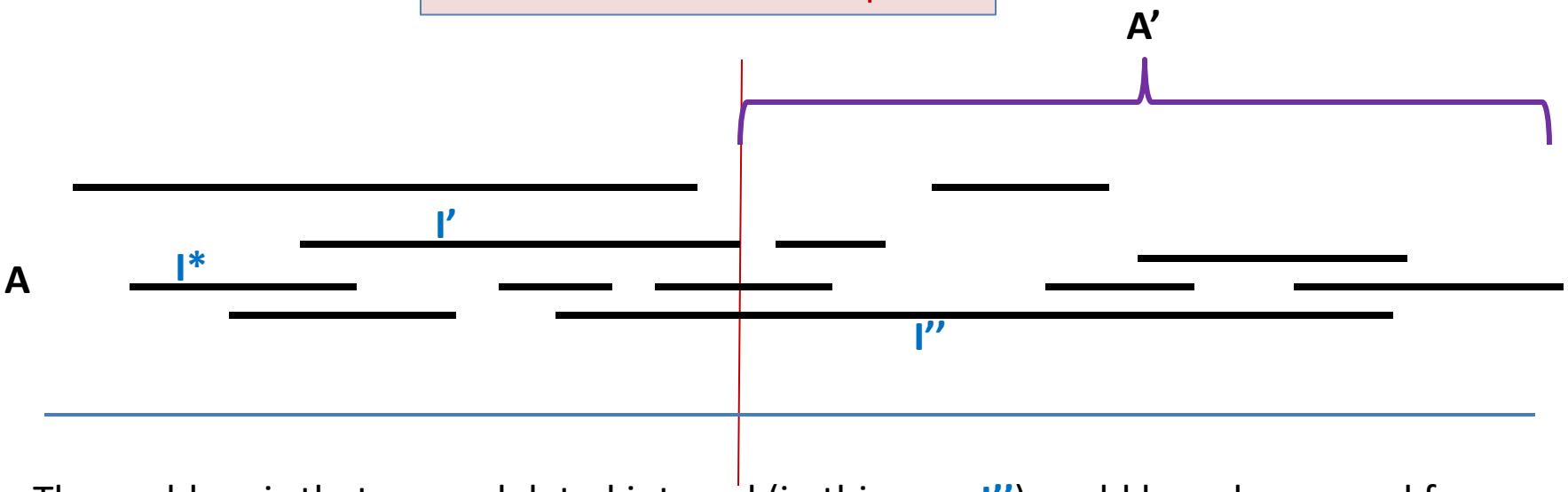


Overlapping Intervals

Question: How to obtain smaller instance **A'** using **Lemma 1** ?

Naive approach : remove from **A** all intervals which overlap with **I'**. This is **A'**.

There is a counter example ☹



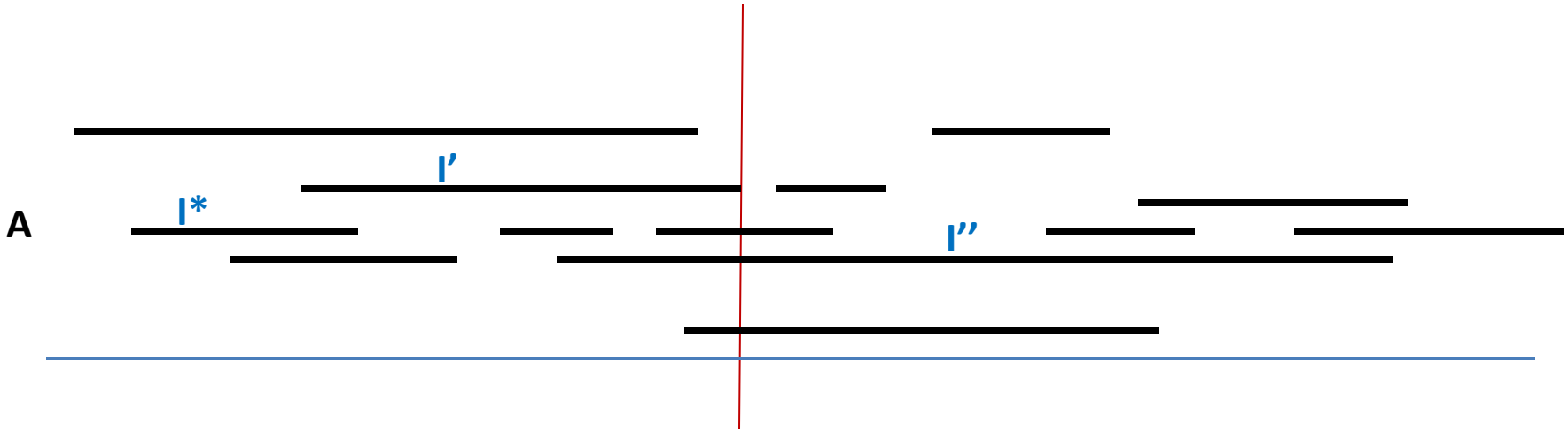
The problem is that some deleted interval (in this case **I''**) could have been used for intersecting many intervals if it were not deleted. But deleting it from the instance disallows it to be selected in the solution.

Overview of the approach

In order to make sure we do not delete intervals (like **I''** in the previous slide) if they are essential to be selected to cover many other intervals, we make some **observations** and introduce a terminology called **Uniquely covered** interval. It turns out that we need to keep **I''** in the smaller instance if there is an interval there which is uniquely covered by **I''** . Otherwise, we may discard **I''**.

Carefully constructing A’

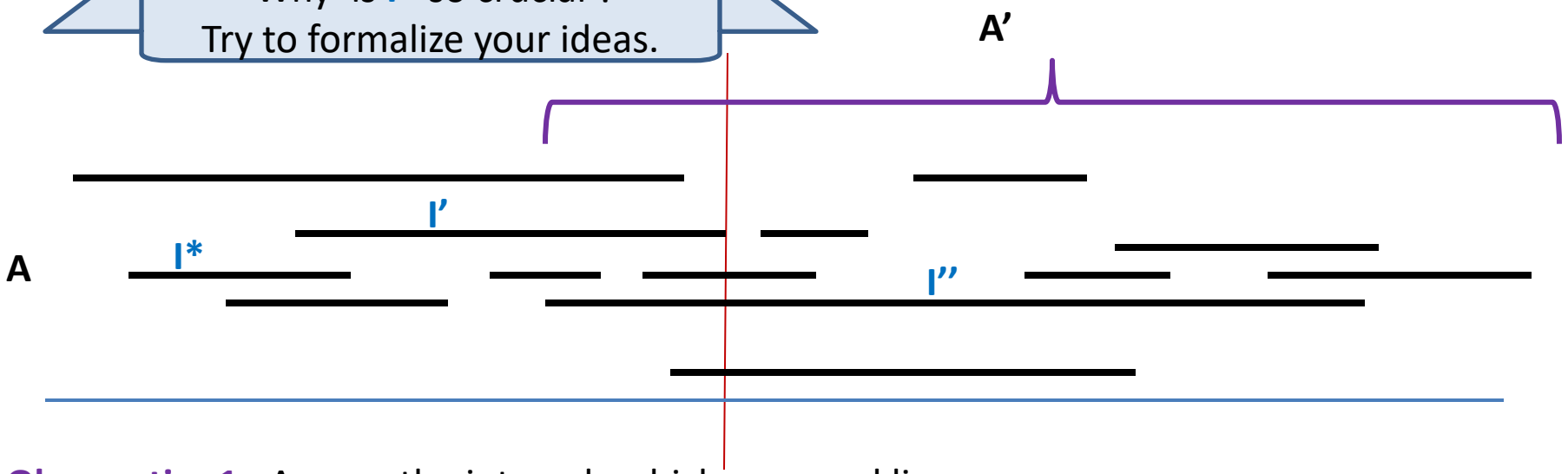
Question: Among the intervals that overlap I’, which intervals should be kept in **A’** ?



Carefully constructing A’

Question: Among the intervals that overlap **I’**, which intervals should be kept in **A’** ?

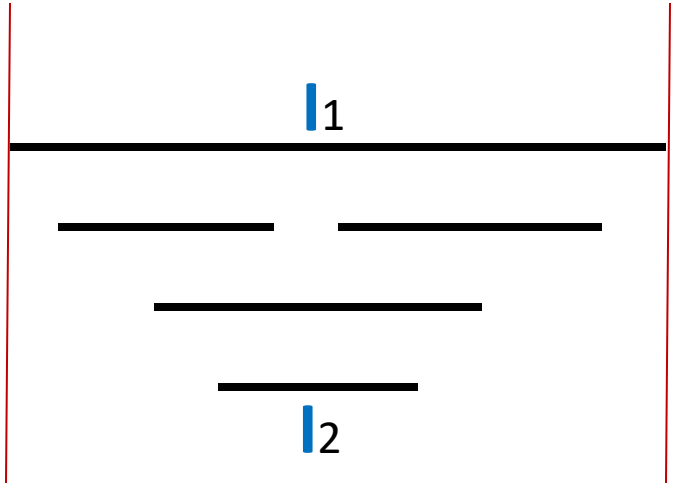
Look at the current example,
Why is **I’’** so crucial ?
Try to formalize your ideas.



Observation1: Among the intervals which cross red line,
only the interval with **maximum finish time** (**I’’** in this picture) may be required.

But how to decide whether to keep **I’’** or not ?
Convince yourself that this is an important point.

Uniquely covered interval



I_2 is said to be uniquely covered by I_1 if

- I_2 is fully covered by I_1
- Every interval overlapping I_2 is also full covered by I_1.

Lemma2 : There is an optimal solution containing I_1.

Proof: Surely I_2 or some other interval overlapping it must be there in the optimal solution. If we replace that interval by I_1, we still get a solution of the same size and hence an optimal solution.

We are now ready to give description/construction of the smaller instance **A'** from **A**.

- There will be two cases.
- We shall then prove that **|Opt(A)|** = **|Opt(A')|** + 1 for each of these cases.

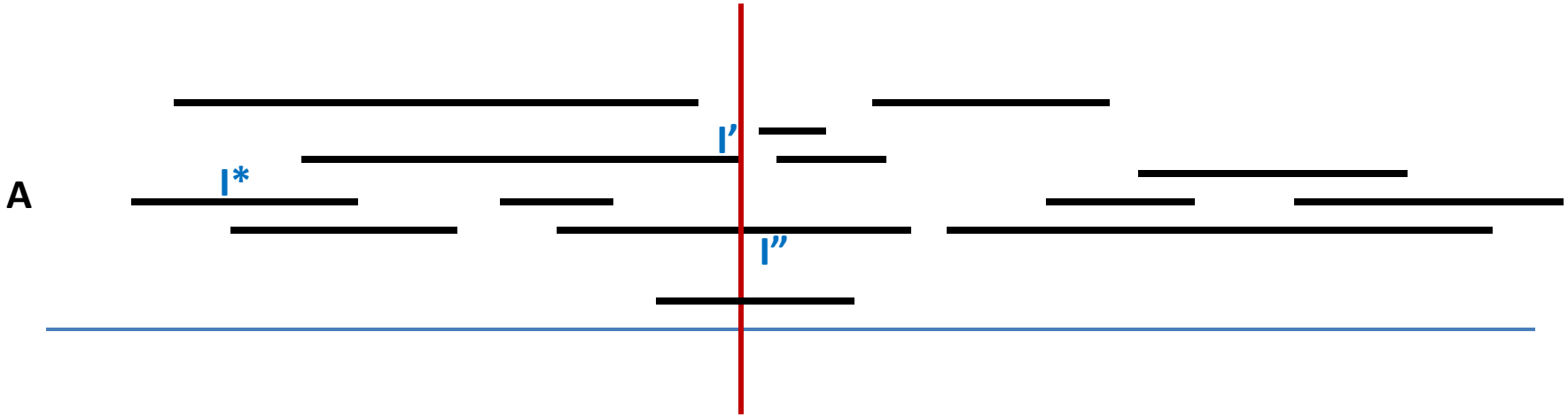
Important note:

- The reader is advised to full understand **Lemma1**, **Lemma2**, **Observation1**, and the notion of **Uniquely covered interval**.
- Also fully internalize the notations **I***, **I'**, and **I"**.

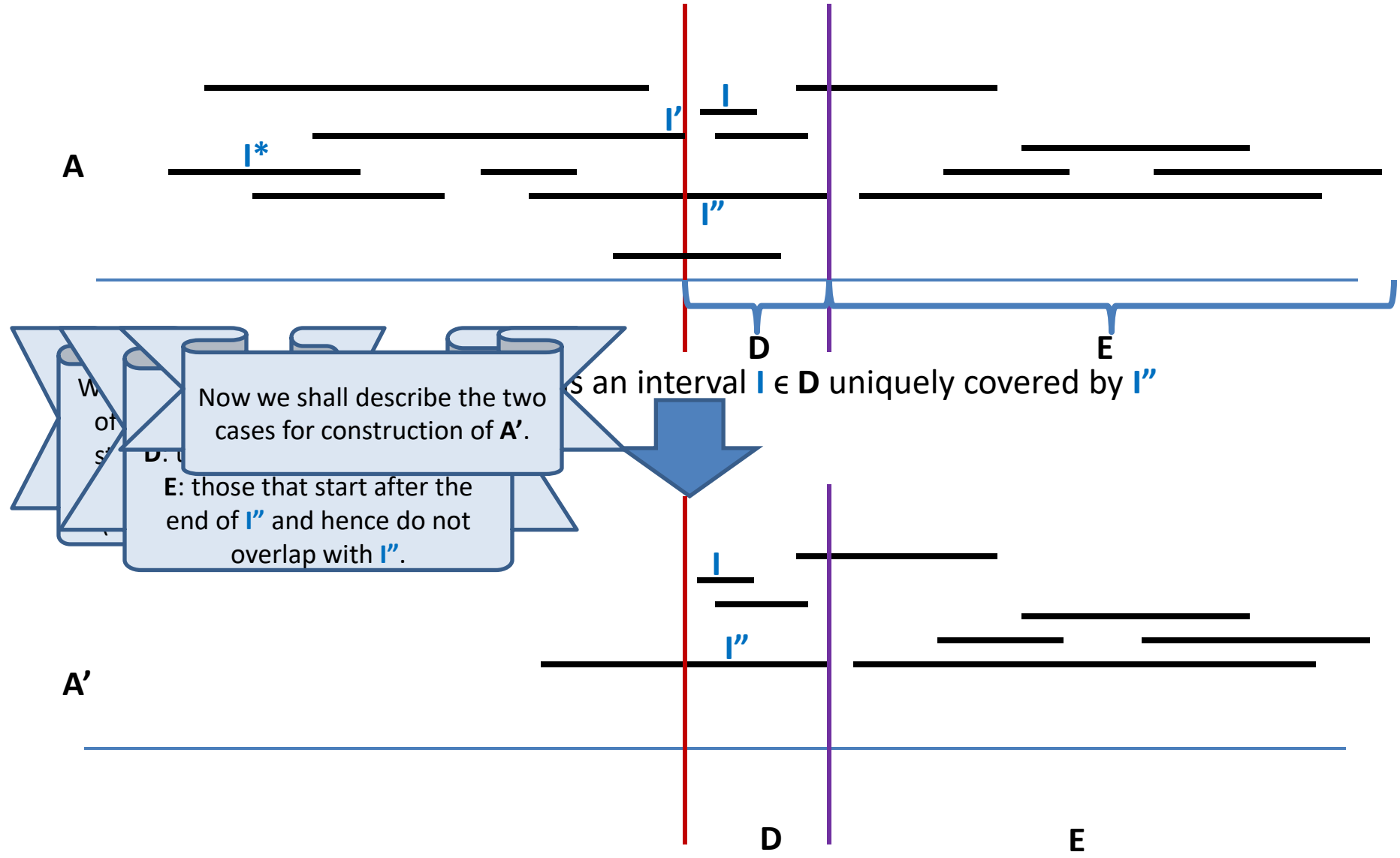
This will help the reader understand the rest of the solution.

Constructing A’ from A

Constructing A’ from A



Constructing A’ from A

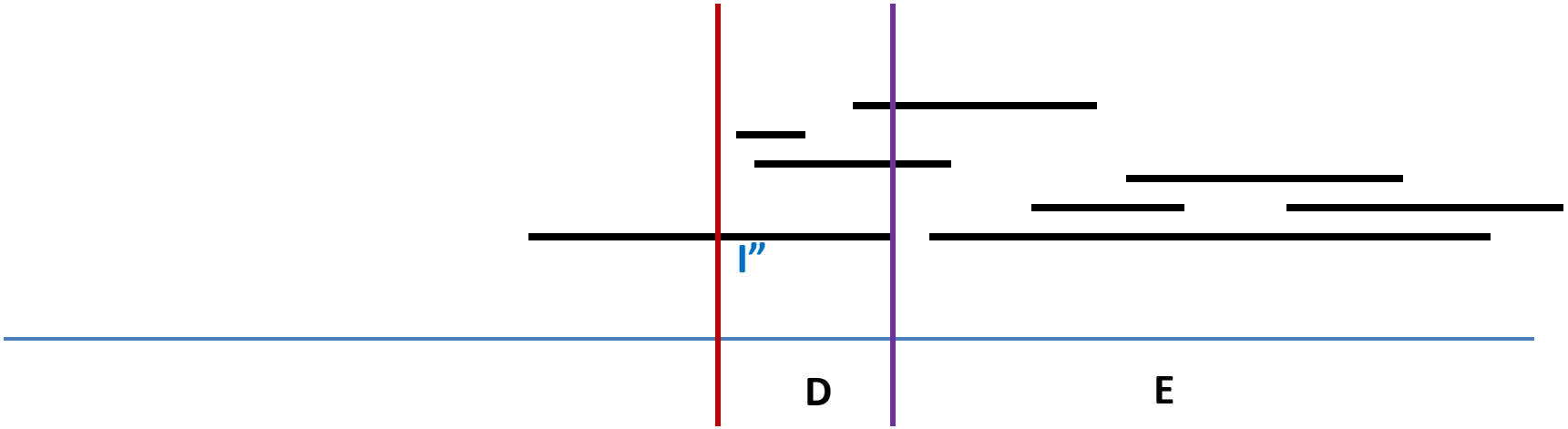


Constructing A' from A

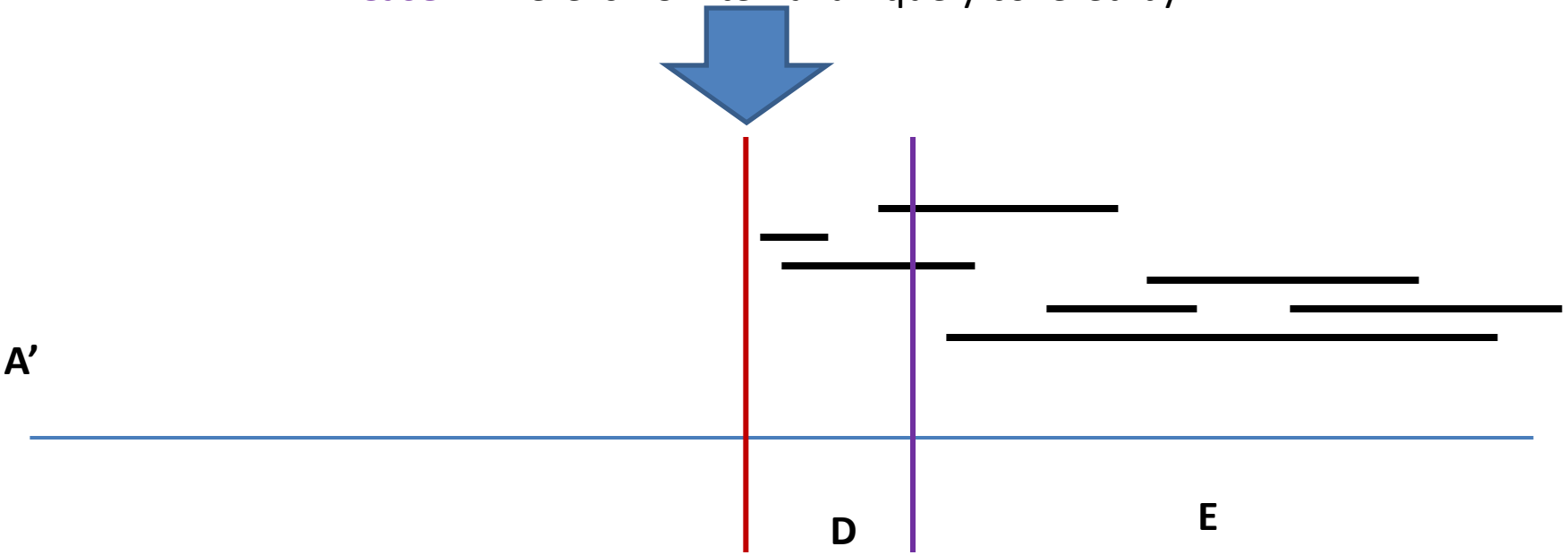
If there is an interval **I** ϵ **D** uniquely covered by **I"**, then we define **A'** as follows. Remove all intervals from **A** which overlap with **I'** (this was our usual way of defining **A'** in our wrong solution). Now add **I"** to this set. This set is the smaller instance **A'** for **Case 1**.

We shall now define **A'** for **Case 2.**

Constructing A’ from A



Case2: There is **no** interval uniquely covered by **I”**



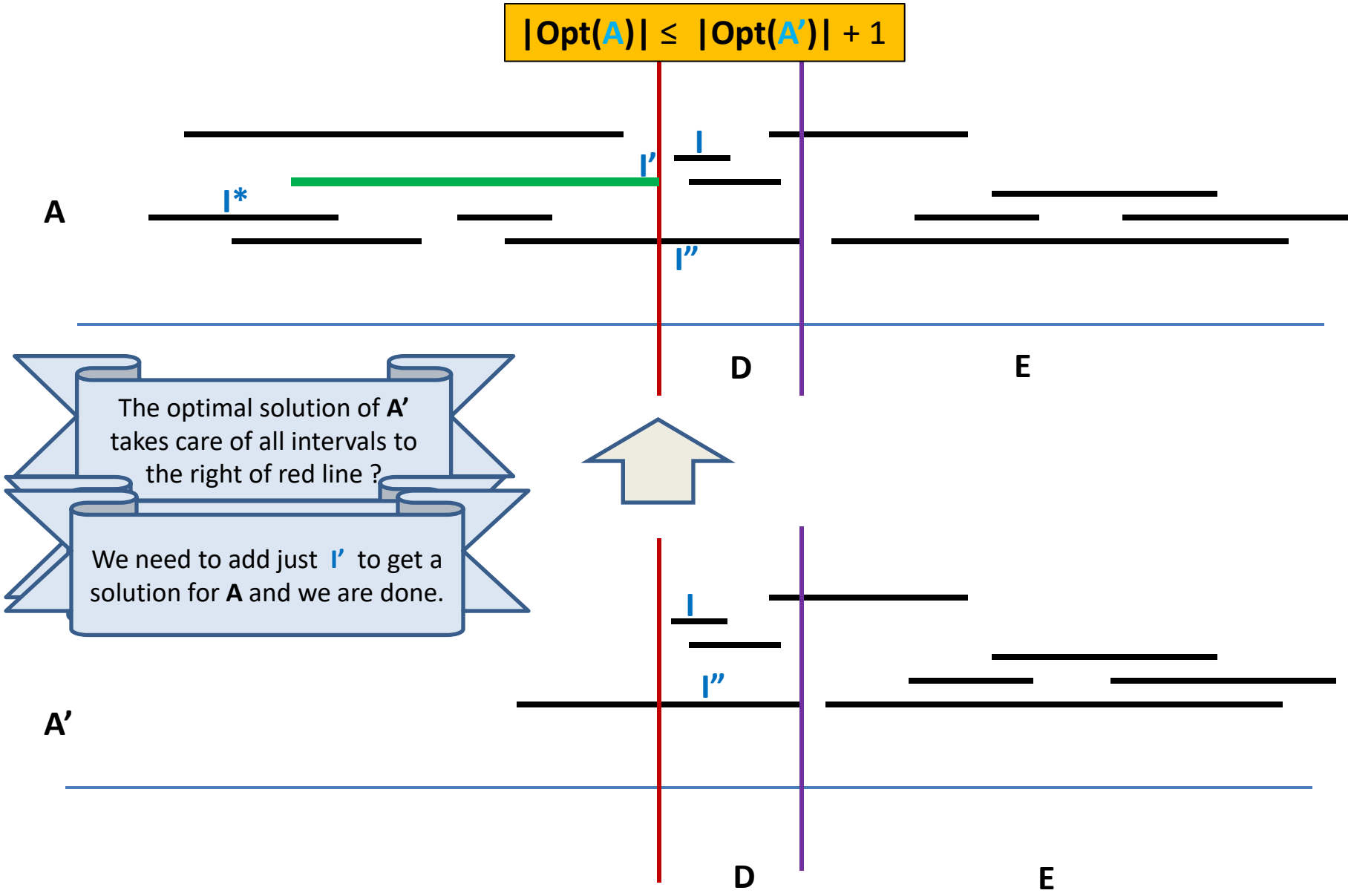
Constructing A’ from A

If there is no interval in **D** uniquely covered by **I”**, then we define **A’** as follows. Remove all intervals from **A** which overlap with **I’** (this was our usual way of defining **A’** in our wrong solution). This set is the smaller instance **A’** for **Case 2**.

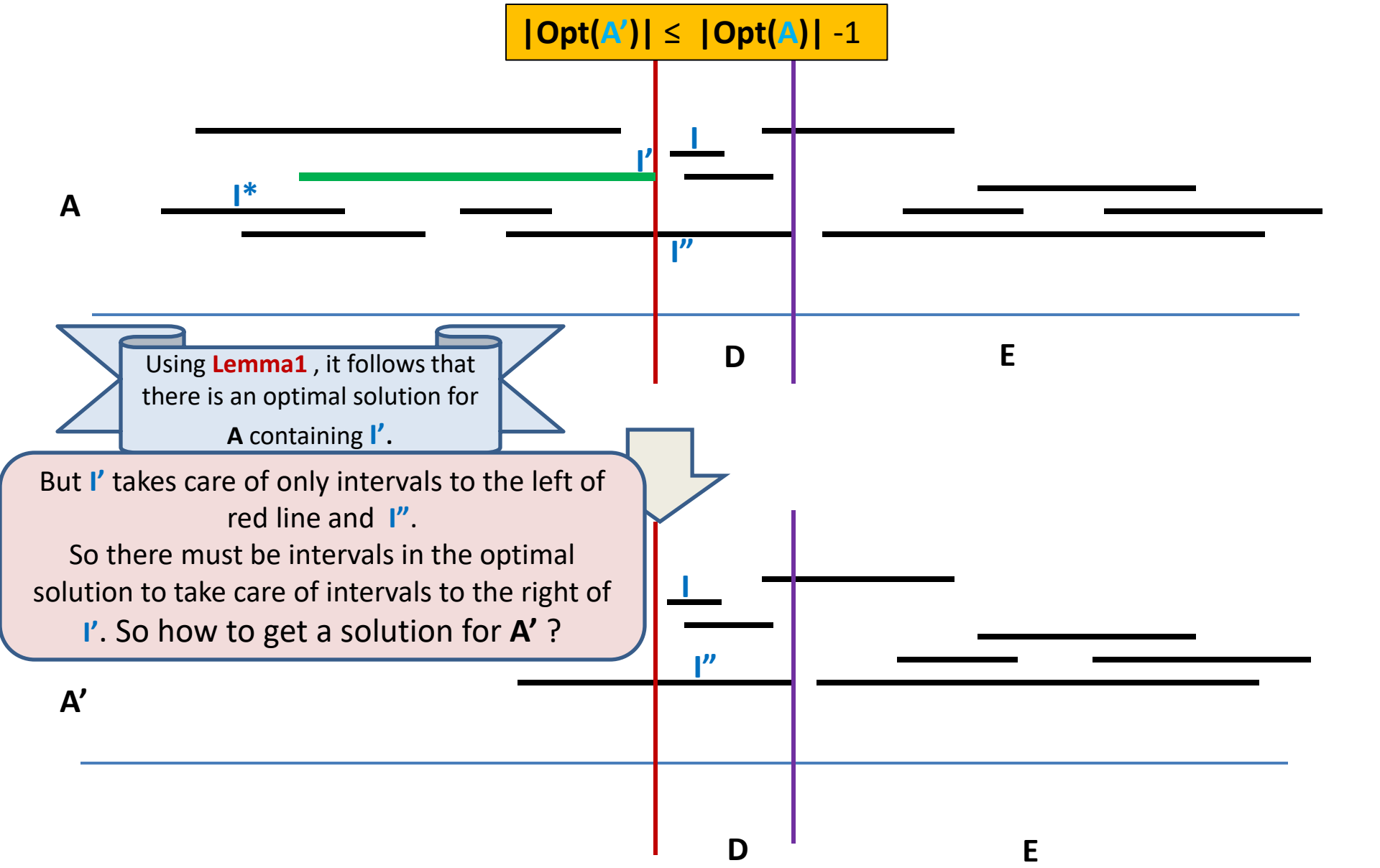
Theorem1: |Opt(A)| = |Opt(A')| + 1

We shall prove this theorem for case 1 as well as case 2.

Case1: There is an interval I ∈ **D** uniquely covered by I”



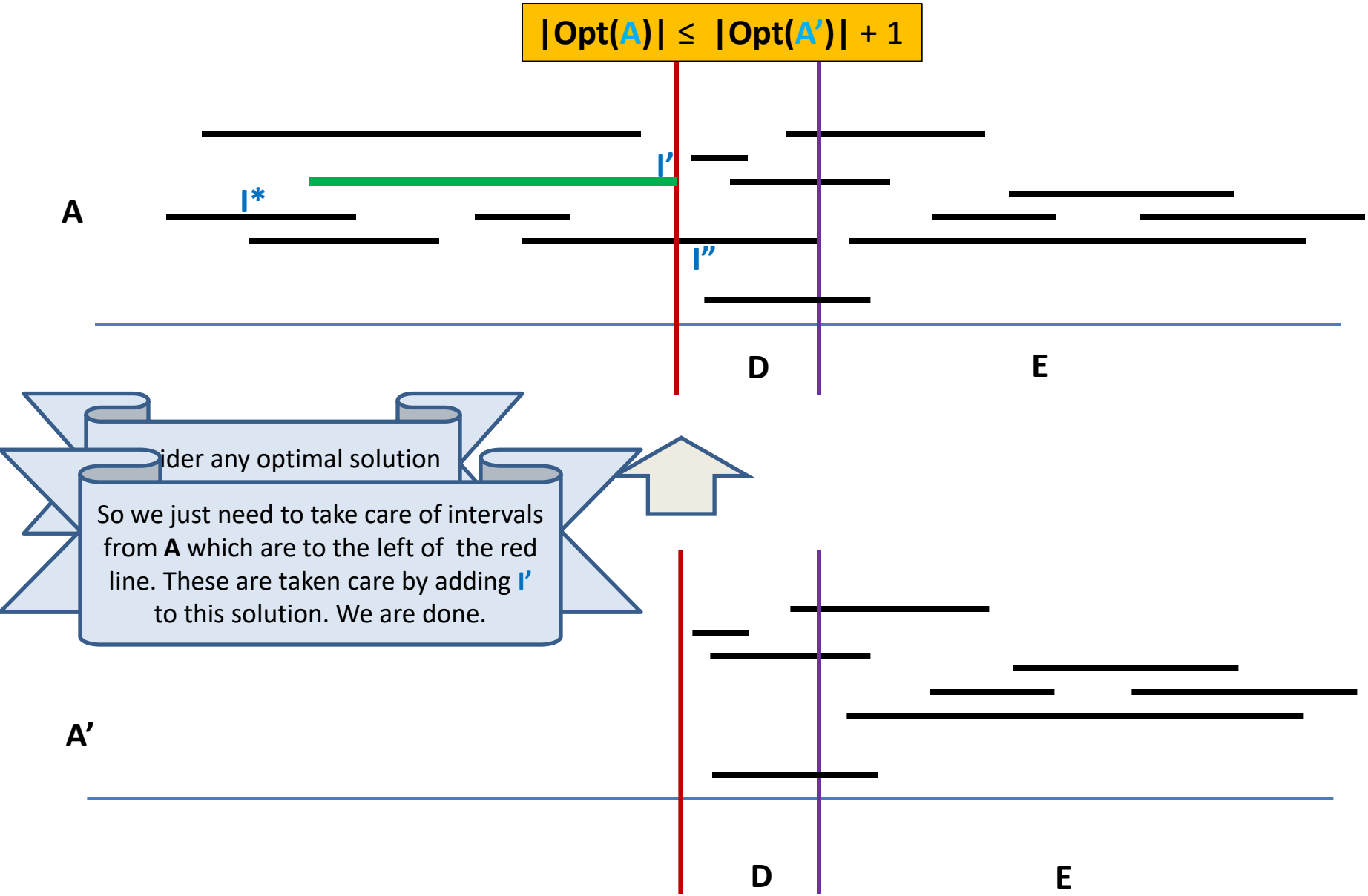
Case1: There is an interval I ∈ **D** uniquely covered by I”



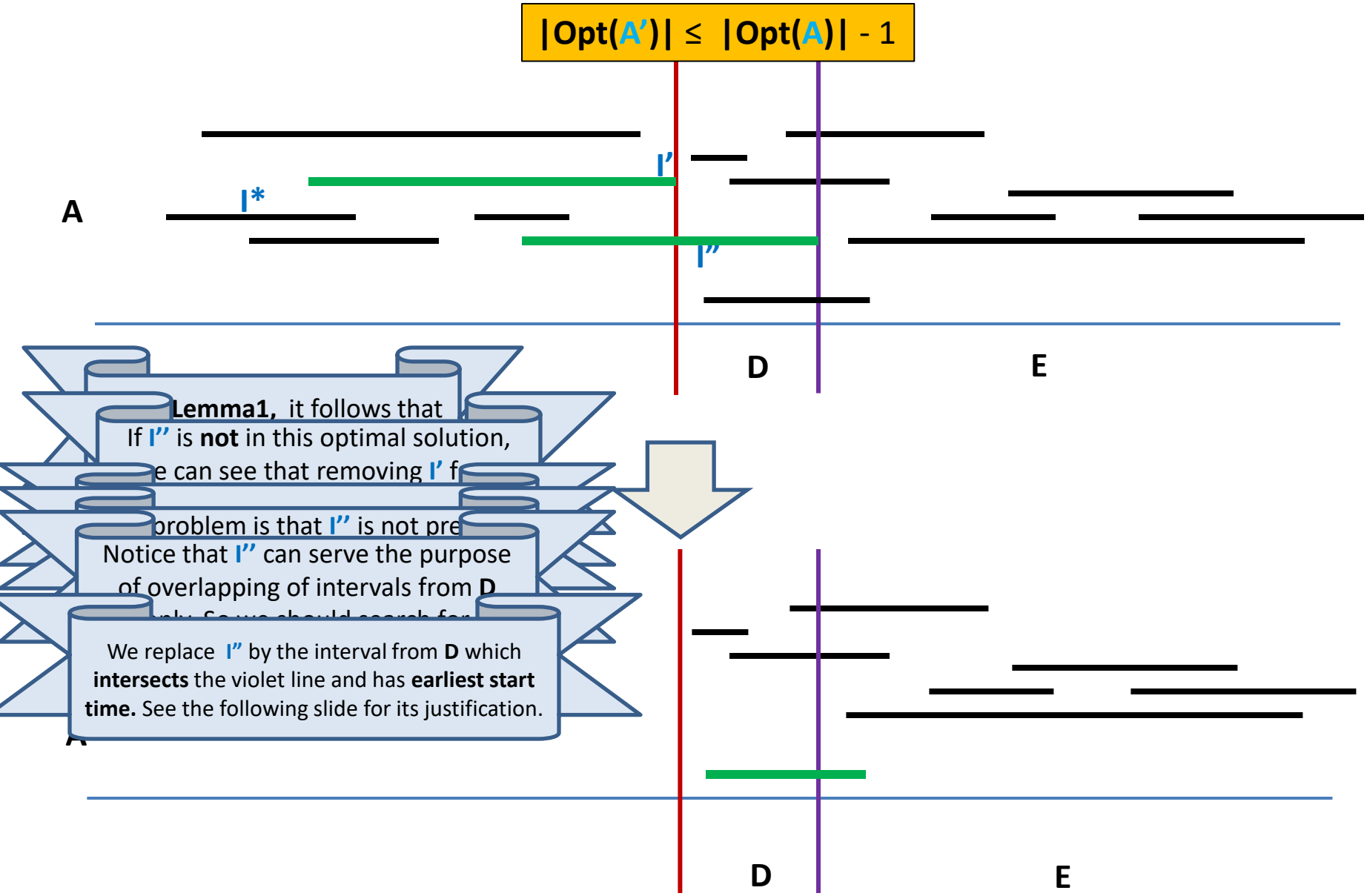
This finishes the proof of **Theorem** for **Case 1**.

We shall now analyze **Case2** and prove **Theorem** for this case as well.

Case2: There is **no** interval uniquely covered by I”



Case2: There is **no** interval uniquely covered by I”



Let **Ï** be the interval in **D** which intersects the violet vertical line (has finish time greater than that of **I”**) and has **earliest** start time. It suffices if we can show that every interval of **D** overlaps with **Ï** . We proceed as follows. Consider any interval **Ĭ** in **D.** There are two cases.

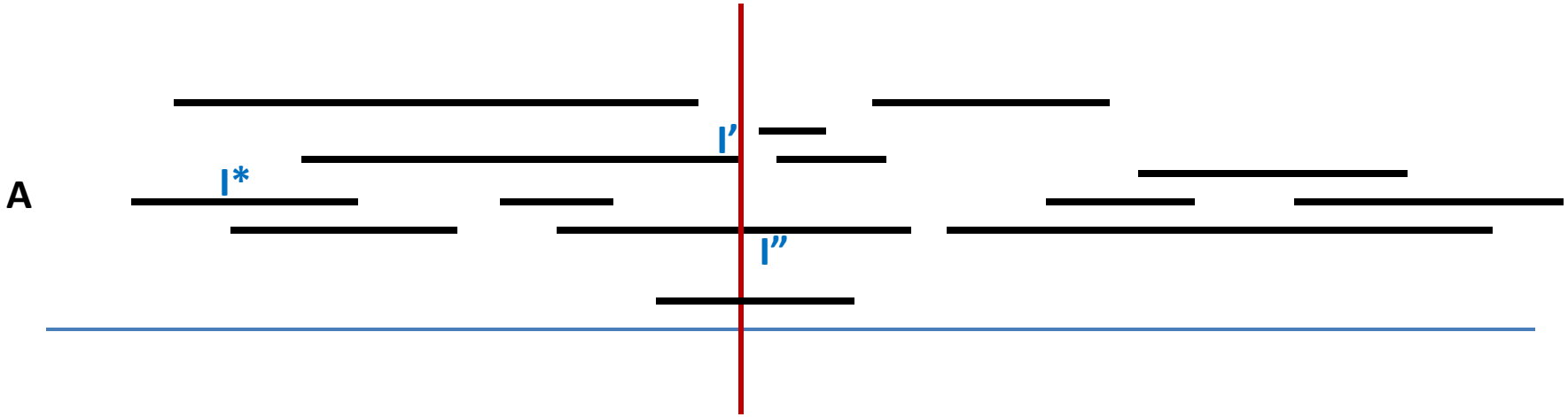
- Finish time of **Ĭ** is less than that of **I”**. In other words, **Ĭ** does not intersects the violet line. In this case, there must be some other interval in **D** that overlaps **Ĭ** and intersects the violet line (otherwise, **Ĭ** would be uniquely covered by **I”**); since start time of **Ï** is less than this interval, so **Ĭ** is overlapped by **Ï** as well.
- Finish time of **Ĭ** is more than **I”**. In other words, **Ĭ** does intersect the violet line. Hence **Ĭ** overlaps with **Ï** as well since the latter also intersects the violet line.

➔ Hence if remove **I’** and **I”** from the given optimal solution of **A**, and add **Ï** to it, we get a solution for **A’**. Since optimal solution for **A’** has to be smaller or equal in size related to this solution, we get **|Opt(A’)|** ≤ **|Opt(A)|** - 1 for **Case 2**.

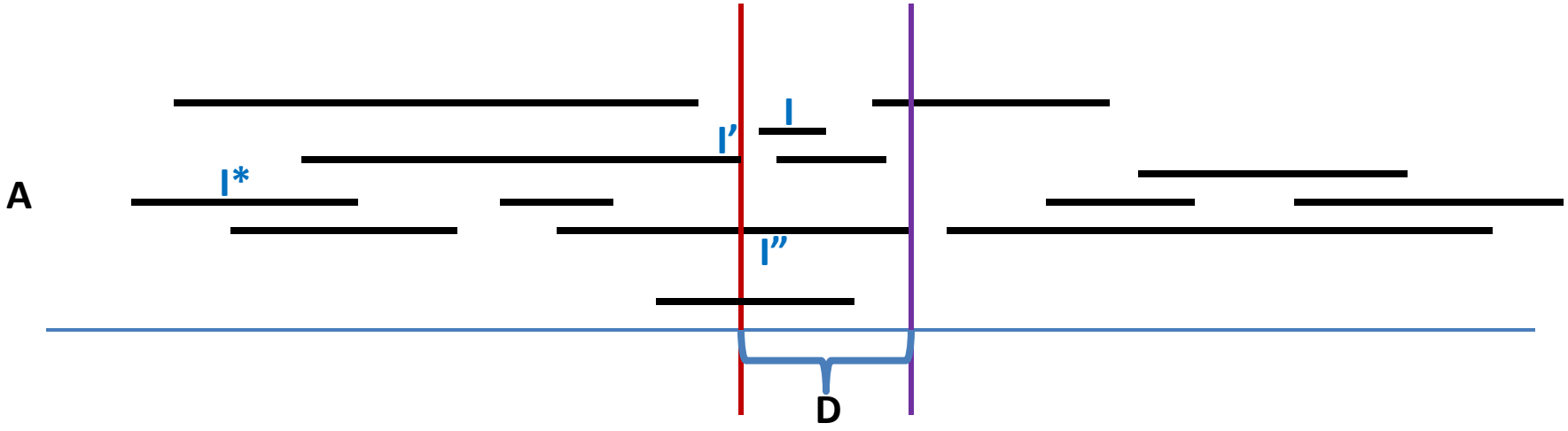
Hence we have proved **Theorem1: |Opt(A)|** = **|Opt(A’)|** + 1

Now in order to design the algorithm for our problem based on the greedy strategy, we just need to determine whether the smaller instance **A’** corresponds to **Case 1** or **Case 2**.

How to distinguish between Case1 and Case2 ?



How to distinguish between Case1 and Case2 ?



Thanks to **Aayush Ojha** and **Shubham Jain** for correcting me. Earlier I had written **start** instead of **finish** here.
(Feel grateful to teach such bright students)

Let

I : the interval in **D** with **earliest finish time**.

Mf(**I**,**D**) : the finish time of that interval in **D** that overlaps **I** and has max. finish time.

If **f**(**I"**) > **Mf**(**I**,**D**)

then **Case 1** (keep **I"** in **A'**)

else **Case 2** (there is no need to keep **I"** in **A'**)

Algorithm

From the discussion till now,

- We have an algorithm for the problem.
- A polynomial bound on the time complexity is obvious.

A necessary (but not sufficient) condition to score **A*** in the course is the following exercise.

Exercise: Provide the most efficient implementation of the algorithm discussed in the class along with its pseudocode.

Submit handwritten (but very neat) report as a solution of this exercise.

If someone has an alternate algorithm, he/she has to provide proof of correctness along with the pseudocode.