Mathematical Induction II

Discrete Mathematics

Andrei Bulatov

Principle of Mathematical Induction

- **Principle of mathematical induction**:

  To prove that a statement that assert that some property $P(n)$ is true for all positive integers n, we complete two steps

  Basis step: We verify that $P(1)$ is true.

  Inductive step: We show that the conditional statement

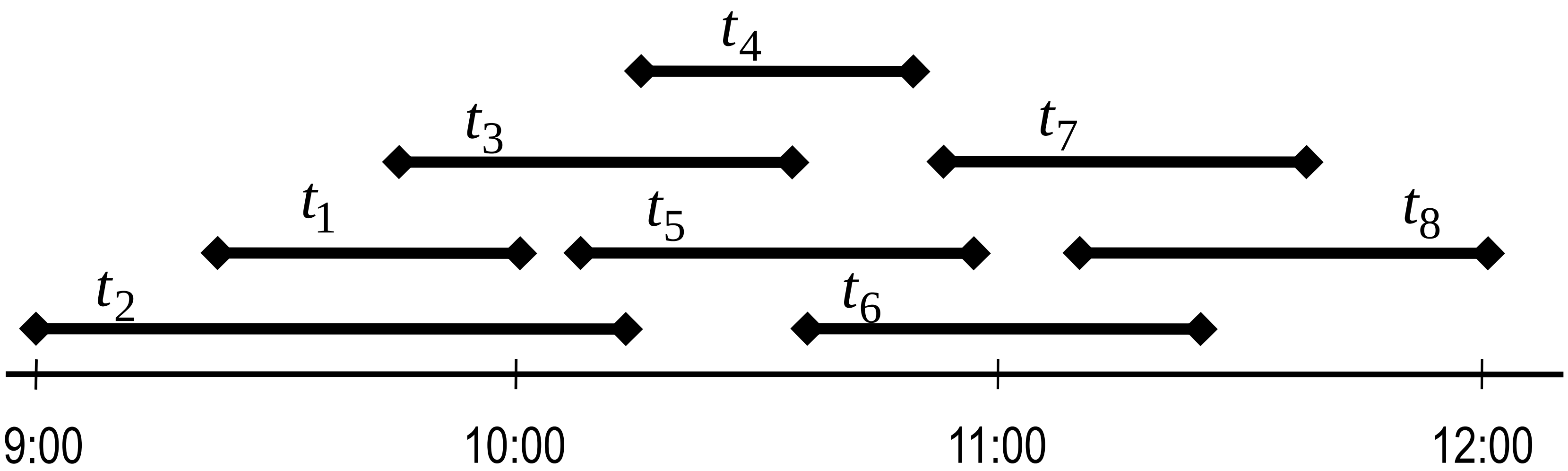
  $$P(k) \rightarrow P(k + 1) \text{ is true for all positive integers } k$$

- To prove the conditional statement, we assume that $P(k)$ is true (it is called *inductive hypothesis*) and show that under this assumption $P(k + 1)$ is also true

Analysis of Algorithms

- Consider the following problem

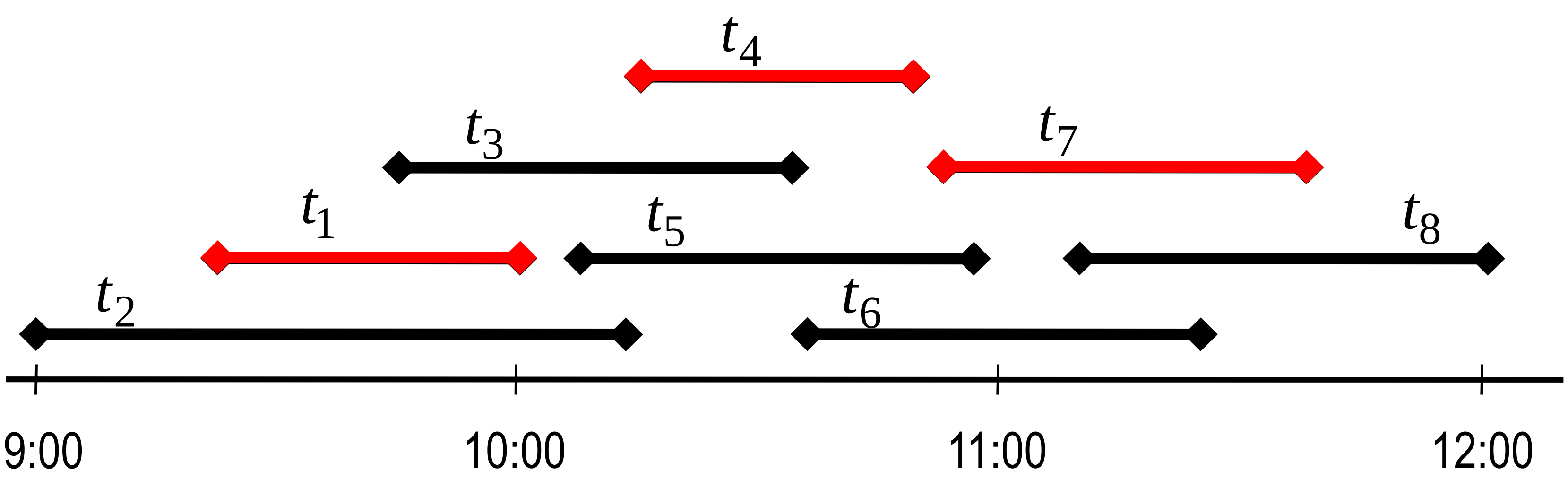
  There is a group of proposed talks to be given. We want to schedule as many talks as possible in the main lecture room. Let $t_1, t_2, \ldots, t_m$ be the talks, talk t_j begins at time b_j and ends at time e_j. (No two lectures can proceed at the same time, but a lecture can begin at the same time another one ends.) We assume that $e_1 \le e_2 \le \ldots \le e_m$.



Greedy Algorithm

- Greedy algorithm:

  At every step choose a talk with the earliest ending time among all those talks that begin after all talks already scheduled end.
- We prove that the greedy algorithm is optimal in the sense that it always schedules the most talks possible in the main lecture hall.



Greedy Algorithm (cntd)

- Let P(n) be the proposition that if the greedy algorithm schedules n talks, then it is not possible to schedule more than n talks.
- Basis step. Suppose that the greedy algorithm has scheduled only one talk, t_1. This means that every other talk starts before e_1, and ends after e_1. Hence, at time e_1 each of the remaining talks needs to use the lecture hall. No two talks can be scheduled because of that. This proves P(1).
- Inductive step. Suppose that P(k) is true, that is, if the greedy algorithm schedules k talks, it is not possible to schedule more than k talks.

  We prove P(k + 1), that is, if the algorithm schedules k + 1 talks then this is the optimal number.

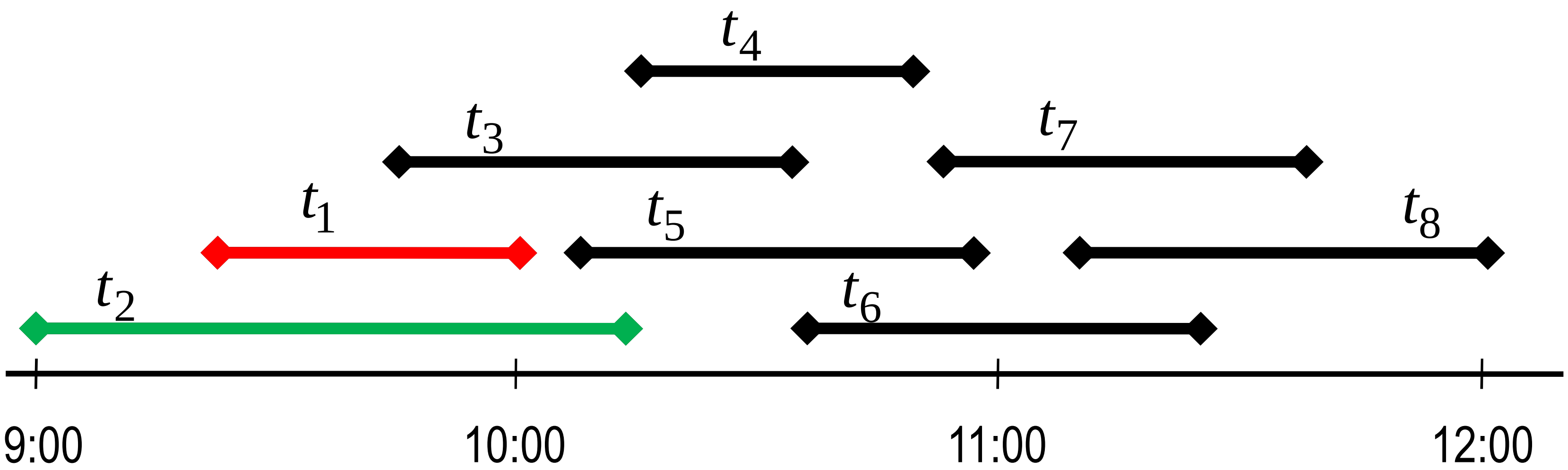
Greedy Algorithm (cntd)

- Suppose that the algorithm has selected k + 1 talks.

  First, we show that there is an optimal scheduling that contains t_1

  Indeed, if we have a schedule that begins with the talk t_i , i > 1, then this first talk can be replaced with t_1.

  To see this, note that, since $e_1 \le e_i$, all talks scheduled after t_1 still can be scheduled.

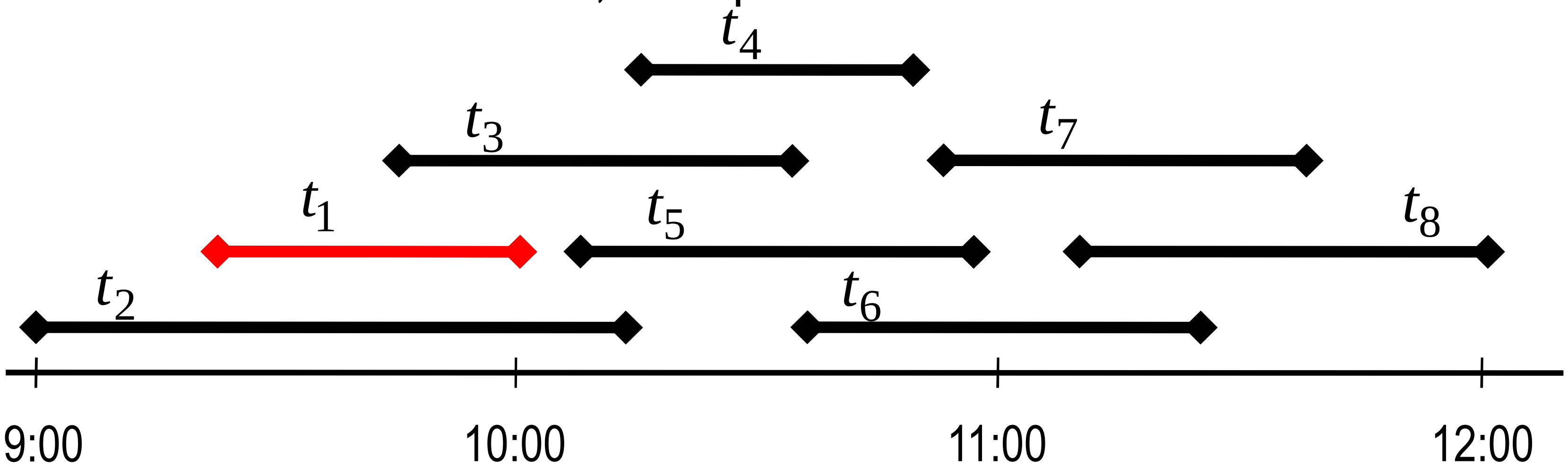


Greedy Algorithm (cntd)

- Once we included t_1, scheduling the talks so that as many as possible talks are scheduled is reduced to scheduling as many talks as possible that begin at or after time e_1.

  The greedy algorithm always schedules t_1, and then schedules k talks choosing them from those that start at or after e_1.

  By the induction hypothesis, it is not possible to schedule more than k such talks. Therefore, the optimal number of talks is k + 1.



Why Induction Works? Well Ordering

- One of the axioms of positive integers is the principle of well-ordering:

  Every non-empty subset of N contains the least element.

- Note that the sets of all integers, rational numbers, and real numbers do not have this property.

- Suppose that mathematical induction is not valid.

  Then there is a predicate $P(n)$ such that $P(1)$ is true,

  $\forall k\,(P(k) \to P(k+1))$ is true, but there is n such that $P(n)$ is false

  Let $T \subseteq N$ be the set of all n such that $P(n)$ is false.

  By the principle of well-ordering T contains the least element a

  As $P(1)$ is true, $a \neq 1$.

  We have $P(a-1)$ is true. However, since $P(a-1) \to P(a)$, we get a contradiction

Recursively Defined Functions

- Induction mechanism can be used to define things.
- To define a function $f\colon \mathbb{N} \to \mathbb{R}$ we complete two steps:

  Basis step: define $f(1)$

  Inductive step: For all k define $f(k + 1)$ as a function of $f(k)$,

  or, more general, as a function of $f(1), f(2), \dots, f(k)$.
- Give a recursive definition of $f(n) = 2^n$

  Basis step: $f(0) = 1$

  Inductive step: $f(k + 1) = 2 \cdot f(k)$.

Factorial

- Another useful recursively defined function is factorial
- $f(n) = n!$

  Basis step: $0! = 1$

  Inductive step: $(k + 1)! = k! \cdot (k + 1)$

n	n!
0	1
1	1
2	2
3	6
4	24

n	n!
5	120
6	720
7	5040
8	40320
9	362880

Fibonacci Numbers

- Usually, Fibonacci numbers are thought of as a sequence of natural numbers, but as we know such a sequence can also be viewed as a function from $\mathbb{N}$.
- F(n)
- Basis step: F(1) = F(2) = 1
- Inductive step: F(k + 1) = F(k) + F(k – 1)

n	1	2	3	4	5	6	7	8	9	10	11	12	13
F(n)	1	1	2	3	5	8	13	21	34	55	89	144	233

Binet's formula

$$F(n) = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}}$$

where φ is the golden ratio

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618033988749$$

Recursively Defined Sets and Structures

- Induction can be used to define structures
- We need to complete the same two steps:

  Basis step: Define the simplest structure possible

  Inductive step: A rule, how to build a bigger structure from smaller ones.

Well Formed Propositional Statements

- What is a well formed statement?

  $(p \to q) \wedge \neg r$ is well formed

  $(p \to q) \neg \wedge r$ is not

- Recursive definition of well formed formulas
- Basis step: A primitive statement is a well formed statement
- Inductive step: If Φ and Ψ are well formed statements, then

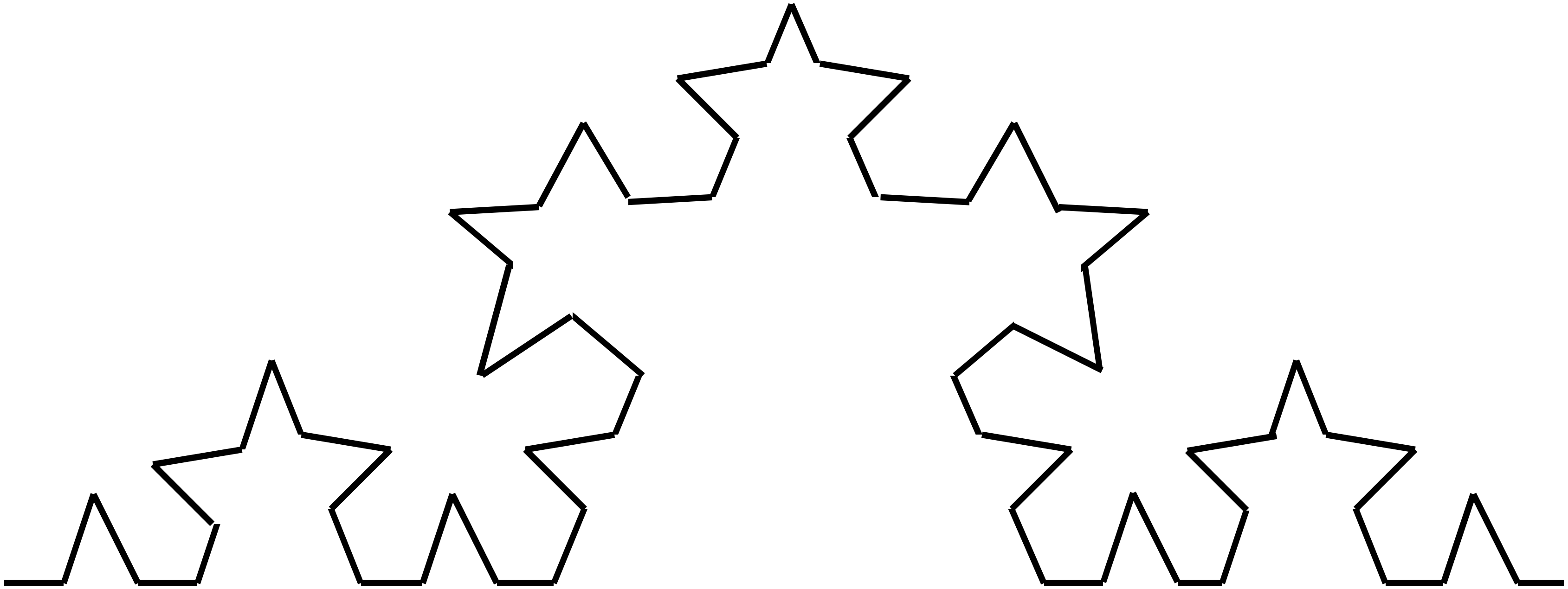
  $\neg \Phi$, $(\Phi \wedge \Psi)$, $(\Phi \vee \Psi)$, $(\Phi \to \Psi)$, $(\Phi \leftrightarrow \Psi)$, $(\Phi \oplus \Psi)$

  are well formed statements

- Such a definition can be used by various algorithms, for example, parsing

Fractals

- Fractals are curves defined recursively
- Basis step: Fractal of level 0 is just a segment
- Inductive step: Divide every segment of the fractal of level k into 3 equal parts and remove the middle one. Insert in this place two sides of a equilateral triangle



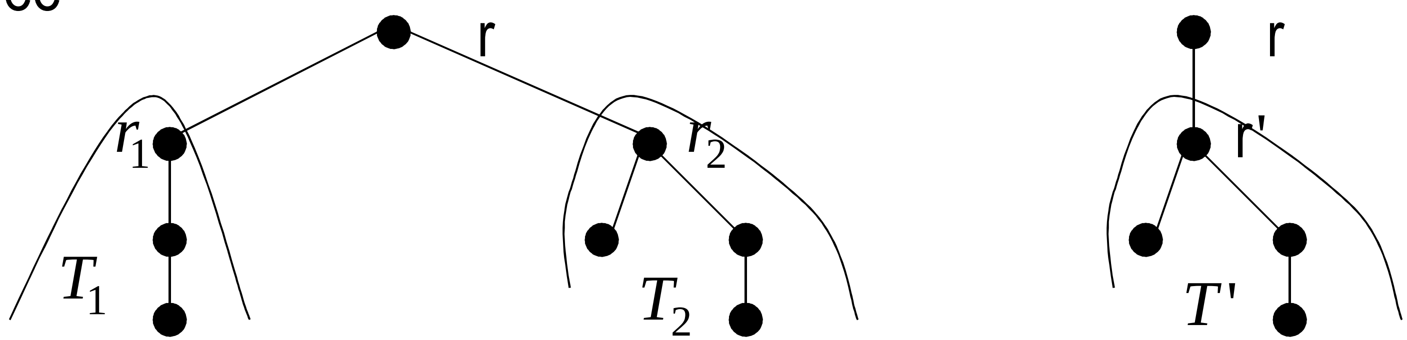
Rooted Trees

- A binary tree is a graph formed by the following recursive definition

- Basis case: A single vertex is a binary tree

- Inductive step: Suppose that T_1, T_2 are disjoint binary trees with roots $r_1, r_2,$ respectively. Then the graph formed by starting with a root r, and adding an edge from r to each of the vertices $r_1, r_2,$ is also a binary tree.

  Or T' is a binary tree with the root r'. Then the graph formed by starting with a root r, and adding an edge from r to r' is also a binary tree



Structural Induction

- To prove properties or design algorithms working with recursively defined structures we need structural induction
- To prove a statement using structural induction we complete two steps

  Basis step: Prove that the property is true for the simplest structure

  Inductive step: Assuming that the property is true for all simpler structures, prove it for a more complex structure

Structural Induction (cntd)

- Height of a binary tree, h(T). Recursive definition:
- Basis step: The height of a single vertex r is 0. h(r) = 0
- Inductive step: If a tree T is built from trees T_1, T_2 as shown in the inductive step, then $h(T) = 1 + \max(h(T_1), h(T_2))$

- We prove that the number of vertices in a binary tree, n(T), satisfies the inequality $n(T) \le 2^{h(T)+1} - 1$
- Basis step: For a single vertex $1 = n(r) \le 2^{0+1} - 1 = 1$
- Inductive step: Let T be formed from T_1, T_2

  We have $n(T) = 1 + n(T_1) + n(T_2)$

$$\le 1 + (2^{h(T_1)+1} - 1) + (2^{h(T_2)+1} - 1)$$

$$\le 1 + 2(2^{\max(h(T_1),h(T_2))+1} - 1) = 1 + 2^{h(T)+1} - 2$$

$$= 2^{h(T)+1} - 1$$

Homework

Exercises from the Book:

No. 3, 4a, 7b (page 244)