Algorithms and Computability

Lecture 12: Nondeterministic Polynomial Time

Christian Schilling (christianms@cs.aau.dk)

slides courtesy of Martin Zimmermann

Yesterday in Algorithms and Computability

We have compared algorithmic complexity:

- Study of concrete algorithms and precise running times
- Difference between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ is huge
- Running times depend on model of computation

and complexity theory:

- Study of problems (languages) rather than algorithms
- Difference between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ may just depend on choice of model (but people still try to find optimal (for fixed model) algorithms)
- Results should be valid for most models of computation

Run Time of a Deterministic Turing Machine

Definition

Let M be a halting DTM

- Let $time_M(w)$ denote the number of configurations in the unique run of M on input w
- Let $T\colon \mathbb{N} \to \mathbb{R}_{>0}$. We say that M runs within time T if $time_M(w) \leq T(|w|)$ for all inputs w

Run Time of a Deterministic Turing Machine

Definition

Let M be a halting DTM

- Let $time_M(w)$ denote the number of configurations in the unique run of M on input w
- Let $T \colon \mathbb{N} \to \mathbb{R}_{>0}$. We say that M runs within time T if $time_M(w) \leq T(|w|)$ for all inputs w

We are interested in classifying problems (languages) according to their asymptotic time complexity

Definition

Let $T \colon \mathbb{N} \to \mathbb{R}_{>0}$ be a function. The complexity class $\textsc{Time}(T)$ is defined as

$$\textsc{Time}(T) = \{L(M) \mid M \text{ is a halting DTM that runs within time } \mathcal{O}(T)\}$$

The Complexity Class P

Definition

The complexity class P (polynomial time) is defined as

$$\mathrm{P} = \bigcup_{k \geq 0} \mathrm{TIME}(n^k)$$

- Robust definition (can use other deterministic (!) models of computation, e.g., multi-tape Turing machines)
- Cobham's thesis: A problem can be efficiently computed if and only if it is in P

Agenda

1. Motivation
2. Nondeterministic Time Complexity
3. Polynomial-Time Reductions

Knapsack

- Recall the Knapsack problem:

  A thief has a knapsack holding at most W kg of loot. The thief robs a store that has items $1, \ldots, n$ of weight w_j and value c_j (each item only once). What is the maximal value the thief can put in the knapsack?

- This is an optimization problem

  But we only focus on decision problems here

- Recall the Knapsack problem:

  A thief has a knapsack holding at most W kg of loot. The thief robs a store that has items $1, \ldots, n$ of weight w_j and value c_j (each item only once). What is the maximal value the thief can put in the knapsack?

- This is an optimization problem

  But we only focus on decision problems here

- As a decision problem:

$$\{W, T, w_1, \ldots, w_n, c_1, \ldots, c_n \in \mathbb{N} \mid \exists b_1, \ldots, b_n \in \{0,1\} \text{ s.t. } \sum_{j=1}^{n} b_j \cdot w_j \leq W \text{ and } \sum_{j=1}^{n} b_j \cdot c_j \geq T\}$$

Knapsack

$$\{W, T, w_1, \ldots, w_n, c_1, \ldots, c_n \in \mathbb{N} \mid \exists b_1, \ldots, b_n \in \{0,1\} \text{ s.t.}$$
$$\sum\nolimits_{j=1}^{n} b_j \cdot w_j \leq W \text{ and } \sum\nolimits_{j=1}^{n} b_j \cdot c_j \geq T\}$$

Knapsack

$$\{W, T, w_1, \ldots, w_n, c_1, \ldots, c_n \in \mathbb{N} \mid \exists b_1, \ldots, b_n \in \{0,1\} \text{ s.t. } \sum_{j=1}^{n} b_j \cdot w_j \leq W \text{ and } \sum_{j=1}^{n} b_j \cdot c_j \geq T\}$$

Consider a backpack of size $W = 100$, the threshold $T = 200$, and the following items:

item j	1	2	3	4	5	6
weight w_j	10	20	60	25	50	35
value c_j	5	70	20	105	30	35

Is $b_1 = 0,\ b_2 = 1,\ b_3 = 0,\ b_4 = 1,\ b_5 = 1,\ b_6 = 0$ a valid solution?

Knapsack

$$\{W, T, w_1, \ldots, w_n, c_1, \ldots, c_n \in \mathbb{N} \mid \exists b_1, \ldots, b_n \in \{0,1\} \text{ s.t.}$$
$$\textstyle\sum_{j=1}^{n} b_j \cdot w_j \leq W \text{ and } \sum_{j=1}^{n} b_j \cdot c_j \geq T\}$$

Consider a backpack of size $W = 100$, the threshold $T = 200$, and the following items:

item j	1	2	3	4	5	6
weight w_j	10	20	60	25	50	35
value c_j	5	70	20	105	30	35

Is $b_1 = 0,\ b_2 = 1,\ b_3 = 0,\ b_4 = 1,\ b_5 = 1,\ b_6 = 0$ a valid solution?
Yes, because

$$w_2 + w_4 + w_5 = 20 + 25 + 50 = 95 \leq 100 = W$$

and

$$c_2 + c_4 + c_5 = 70 + 105 + 30 = 205 \geq 200 = T$$

Knapsack

$$\{W, T, w_1, \ldots, w_n, c_1, \ldots, c_n \in \mathbb{N} \mid \exists b_1, \ldots, b_n \in \{0,1\} \text{ s.t. } \sum\nolimits_{j=1}^{n} b_j \cdot w_j \leq W \text{ and } \sum\nolimits_{j=1}^{n} b_j \cdot c_j \geq T\}$$

Now, consider a backpack of size 15, the threshold $T = 50$, and the following items:

item j	1	2	3	4	5	6	7
weight w_j	2	3	5	7	1	4	1
value c_j	10	5	15	7	6	18	3

Is there a valid solution?

Knapsack

$$\{W, T, w_1, \ldots, w_n, c_1, \ldots, c_n \in \mathbb{N} \mid \exists b_1, \ldots, b_n \in \{0,1\} \text{ s.t.} \\ \sum_{j=1}^{n} b_j \cdot w_j \leq W \text{ and } \sum_{j=1}^{n} b_j \cdot c_j \geq T\}$$

Now, consider a backpack of size 15, the threshold $T = 50$, and the following items:

item j	1	2	3	4	5	6	7
weight w_j	2	3	5	7	1	4	1
value c_j	10	5	15	7	6	18	3

Is there a valid solution?
Yes, e.g., $b_1 = 1$, $b_2 = 1$, $b_3 = 1$, $b_4 = 0$, $b_5 = 1$, $b_6 = 1$, $b_6 = 0$

Knapsack

$$\{W, T, w_1, \ldots, w_n, c_1, \ldots, c_n \in \mathbb{N} \mid \exists b_1, \ldots, b_n \in \{0,1\} \text{ s.t.}$$
$$\textstyle\sum_{j=1}^{n} b_j \cdot w_j \leq W \text{ and } \sum_{j=1}^{n} b_j \cdot c_j \geq T\}$$

Now, consider a backpack of size 15, the threshold $T = 50$, and the following items:

item j	1	2	3	4	5	6	7
weight w_j	2	3	5	7	1	4	1
value c_j	10	5	15	7	6	18	3

Is there a valid solution?

Yes, e.g., $b_1 = 1$, $b_2 = 1$, $b_3 = 1$, $b_4 = 0$, $b_5 = 1$, $b_6 = 1$, $b_6 = 0$

Algorithm to compute a solution (or conclude that there is none)?

Satisfiability

$$\varphi = (x_0 \vee x_2) \wedge (x_0 \vee \neg x_3) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (x_2 \vee \neg x_4) \wedge \\ (x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Satisfiability

$$\varphi = (x_0 \vee x_2) \wedge (x_0 \vee \neg x_3) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (x_2 \vee \neg x_4) \wedge$$
$$(x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

$$\varphi = (1 \vee x_2) \wedge (x_0 \vee \neg x_3) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (x_2 \vee \neg x_4) \wedge$$
$$(x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (x_0 \vee \neg x_3) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (x_2 \vee \neg x_4) \wedge$$
$$(x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee \neg x_3) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (x_2 \vee \neg x_4) \wedge$$
$$(x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (x_2 \vee \neg x_4) \wedge$$
$$(x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (x_2 \vee \neg x_4) \wedge \\ (x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (x_1 \vee \neg x_4) \wedge (x_2 \vee \neg x_4) \wedge$$
$$(x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee \neg x_4) \wedge (x_2 \vee \neg x_4) \wedge$$
$$(x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (x_2 \vee \neg x_4) \wedge$$
$$(x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee \neg x_4) \wedge$$
$$(x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge$$
$$(x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge \\ (1 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge$$
$$(1 \vee 0) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge$$
$$(1 \vee 0) \wedge (1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge$$
$$(1 \vee 0) \wedge (1 \vee 0) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge \\ (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge \\ (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge \\ (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge \\ (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 1) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge \\ (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 1) \wedge (1 \vee x_6) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge$$
$$(1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 1) \wedge (1 \vee 1) \wedge (x_5 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge \\ (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee x_6)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge$$
$$(1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee 1)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge$$
$$(1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee 1)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1) \wedge (1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge$$
$$(1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee 1)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1) \wedge (1) \wedge (1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge \\ (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee 1)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\begin{aligned}\varphi = &(1) \wedge (1) \wedge (1) \wedge (1) \wedge (1 \vee 0) \wedge \\ &(1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee 1)\end{aligned}$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1) \wedge (1) \wedge (1) \wedge (1) \wedge (1) \wedge$$
$$(1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee 1)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1) \wedge (1) \wedge (1) \wedge (1) \wedge (1) \wedge$$
$$(1) \wedge (1 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee 1)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

$$\varphi = (1) \wedge (1) \wedge (1) \wedge (1) \wedge (1) \wedge \\ (1) \wedge (1) \wedge (1 \vee 0) \wedge (1 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee 1)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

$$\varphi = (1) \wedge (1) \wedge (1) \wedge (1) \wedge (1) \wedge$$
$$(1) \wedge (1) \wedge (1) \wedge (1 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee 1)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1) \wedge (1) \wedge (1) \wedge (1) \wedge (1) \wedge$$
$$(1) \wedge (1) \wedge (1) \wedge (1) \wedge (1 \vee 1) \wedge (1 \vee 1)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1) \wedge (1) \wedge (1) \wedge (1) \wedge (1) \wedge$$
$$(1) \wedge (1) \wedge (1) \wedge (1) \wedge (1) \wedge (1 \vee 1)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$$\varphi = (1) \wedge (1) \wedge (1) \wedge (1) \wedge (1) \wedge$$
$$(1) \wedge (1) \wedge (1) \wedge (1) \wedge (1) \wedge (1)$$

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

$\varphi = 1$ **Yes**!

Is $x_0 = x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 1$ a satisfying assignment?

Satisfiability

$\varphi = 1$ **Yes**!

Does

$$\varphi = (x_0 \vee x_2 \vee \neg x_4) \wedge (\neg x_0 \vee \neg x_3 \vee \neg x_1) \wedge (x_1 \vee \neg x_3 \vee \neg x_2) \wedge$$
$$(x_0 \vee \neg x_2 \vee x_1) \wedge (x_1 \vee \neg x_5 \vee x_2) \wedge (x_2 \vee x_5 \vee x_4) \wedge$$
$$(\neg x_1 \vee x_2 \vee x_3) \wedge (x_5 \vee \neg x_3 \vee \neg x_1) \wedge (x_1 \vee \neg x_2 \vee x_4)$$

have a satisfying assignment?

Similarities

- Knapsack:

$$\{W, T, w_1, \ldots, w_n, c_1, \ldots, c_n \in \mathbb{N} \mid \exists b_1, \ldots, b_n \in \{0,1\} \text{ s.t. } \sum\nolimits_{j=1}^{n} b_j \cdot w_j \leq W \text{ and } \sum\nolimits_{j=1}^{n} b_j \cdot c_j \geq T\}$$

- Boolean satisfiability:

$$\mathrm{SAT} = \{\varphi \mid \text{there exists a satisfying assignment for } \varphi\}$$

- Knapsack:

$$\{W, T, w_1, \ldots, w_n, c_1, \ldots, c_n \in \mathbb{N} \mid \exists b_1, \ldots, b_n \in \{0,1\} \text{ s.t. } \sum\nolimits_{j=1}^{n} b_j \cdot w_j \leq W \text{ and } \sum\nolimits_{j=1}^{n} b_j \cdot c_j \geq T\}$$

- Boolean satisfiability:

$$\mathrm{SAT} = \{\varphi \mid \text{there exists a satisfying assignment for } \varphi\}$$

Similarities

- Both problems ask for the existence of a certificate "proving" that the input is in the language
  (a bit vector $b_1 \cdots b_n$ resp. an assignment)
- Certificates are easy to verify, i.e, in polynomial time
- Certificates are short, i.e., polynomial in the input length

More

Many other important problems share these traits:

- Nonprimality: $\{n \in \mathbb{N} \mid n = a \cdot b \text{ for } a, b > 1\}$

More

Many other important problems share these traits:

- Nonprimality: $\{n \in \mathbb{N} \mid n = a \cdot b \text{ for } a, b > 1\}$
- Hamiltonian path (a path that visits each vertex exactly once):

$$\{G \mid G \text{ is a graph with a Hamiltonian path}\}$$

More

Many other important problems share these traits:

- Nonprimality: $\{n \in \mathbb{N} \mid n = a \cdot b \text{ for } a, b > 1\}$
- Hamiltonian path (a path that visits each vertex exactly once):

  $$\{G \mid G \text{ is a graph with a Hamiltonian path}\}$$

- Clique (a k-clique is a set of k vertices all pairwise connected by an edge): $\{(G, k) \mid G \text{ is a graph with a } k\text{-clique}\}$

More

Many other important problems share these traits:

- Nonprimality: $\{n \in \mathbb{N} \mid n = a \cdot b \text{ for } a, b > 1\}$
- Hamiltonian path (a path that visits each vertex exactly once):

$$\{G \mid G \text{ is a graph with a Hamiltonian path}\}$$

- Clique (a k-clique is a set of k vertices all pairwise connected by an edge): $\{(G, k) \mid G \text{ is a graph with a } k\text{-clique}\}$
- Linear programming: Does a linear program have a solution in $\mathbb{R}$ whose value is at least t?
  Note that this problem is not quite the same (why?)

More

Many other important problems share these traits:

- Nonprimality: $\{n \in \mathbb{N} \mid n = a \cdot b \text{ for } a, b > 1\}$
- Hamiltonian path (a path that visits each vertex exactly once):

  $$\{G \mid G \text{ is a graph with a Hamiltonian path}\}$$

- Clique (a k-clique is a set of k vertices all pairwise connected by an edge): $\{(G, k) \mid G \text{ is a graph with a } k\text{-clique}\}$
- Linear programming: Does a linear program have a solution in $\mathbb{R}$ whose value is at least t?
  Note that this problem is not quite the same (why?)
- 3-coloring (can the vertices of a graph be colored red/green/blue such that all vertices connected by an edge have different colors?): $\{G \mid G \text{ is a graph with a 3-coloring}\}$

More

Many other important problems share these traits:

- Nonprimality: $\{n \in \mathbb{N} \mid n = a \cdot b \text{ for } a, b > 1\}$
- Hamiltonian path (a path that visits each vertex exactly once):

  $$\{G \mid G \text{ is a graph with a Hamiltonian path}\}$$

- Clique (a k-clique is a set of k vertices all pairwise connected by an edge): $\{(G, k) \mid G \text{ is a graph with a } k\text{-clique}\}$
- Linear programming: Does a linear program have a solution in $\mathbb{R}$ whose value is at least t?
  Note that this problem is not quite the same (why?)
- 3-coloring (can the vertices of a graph be colored red/green/blue such that all vertices connected by an edge have different colors?): $\{G \mid G \text{ is a graph with a 3-coloring}\}$

Side note: some problems (are believed to) have different complexity

Quiz 1

Is verifying a certificate easier than finding a certificate?

Quiz 1

Is verifying a certificate easier than finding a certificate?

Generally: widely believed to be true (but no proof yet)

Agenda

Nondeterminism to the Rescue

Nondeterminism allows a Turing machine to find a certificate: For example, to turn any sequence of n consecutive X's into all $0/1$ vectors of length n, use the following transitions:

$$\delta(q, X) \to \{(q, 0, +1), (q, 1, +1)\}$$

Nondeterminism to the Rescue

Nondeterminism allows a Turing machine to find a certificate: For example, to turn any sequence of n consecutive X's into all 0/1 vectors of length n, use the following transitions:

$$\delta(q, X) \to \{(q, 0, +1), (q, 1, +1)\}$$

$$[q, XXX, 1]$$

Nondeterminism to the Rescue

Nondeterminism allows a Turing machine to find a certificate: For example, to turn any sequence of n consecutive X's into all $0/1$ vectors of length n, use the following transitions:

$$\delta(q, X) \to \{(q, 0, +1), (q, 1, +1)\}$$

$$[q, XXX, 1]$$

$$[q, 0XX, 2]$$

Nondeterminism to the Rescue

Nondeterminism allows a Turing machine to find a certificate: For example, to turn any sequence of n consecutive X's into all 0/1 vectors of length n, use the following transitions:

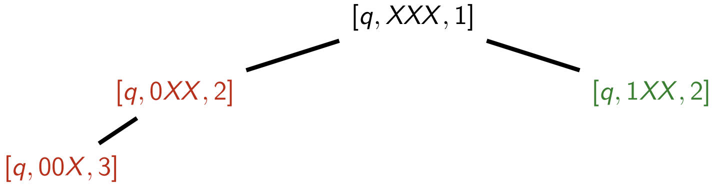
$$\delta(q, X) \to \{(q, 0, +1), (q, 1, +1)\}$$

$$[q, XXX, 1]$$

$$[q, 0XX, 2] \qquad\qquad [q, 1XX, 2]$$

Nondeterminism allows a Turing machine to find a certificate: For example, to turn any sequence of n consecutive X's into all 0/1 vectors of length n, use the following transitions:

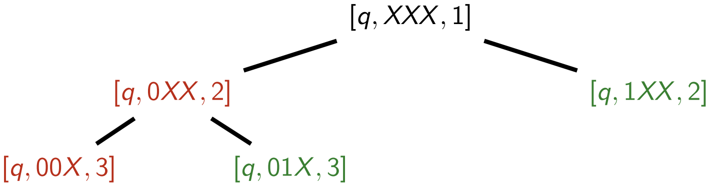
$$\delta(q, X) \to \{(q, 0, +1), (q, 1, +1)\}$$



Nondeterminism to the Rescue

Nondeterminism allows a Turing machine to find a certificate: For example, to turn any sequence of n consecutive X's into all 0/1 vectors of length n, use the following transitions:

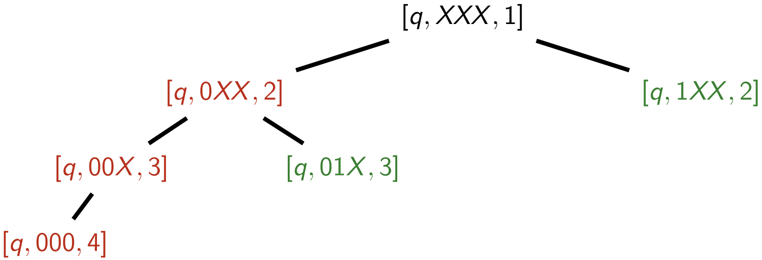
$$\delta(q, X) \rightarrow \{(q, 0, +1), (q, 1, +1)\}$$



Nondeterminism to the Rescue

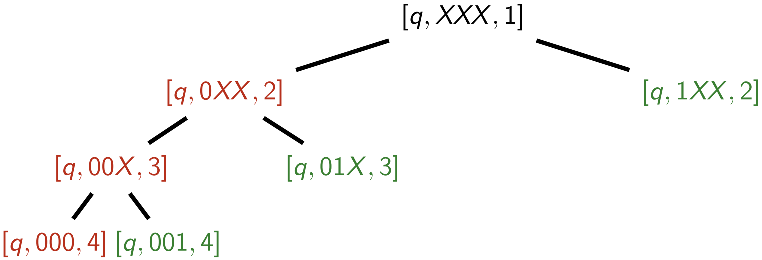
Nondeterminism allows a Turing machine to find a certificate: For example, to turn any sequence of n consecutive X's into all $0/1$ vectors of length n, use the following transitions:

$$\delta(q, X) \to \{(q, 0, +1), (q, 1, +1)\}$$



Nondeterminism allows a Turing machine to find a certificate: For example, to turn any sequence of n consecutive X's into all 0/1 vectors of length n, use the following transitions:

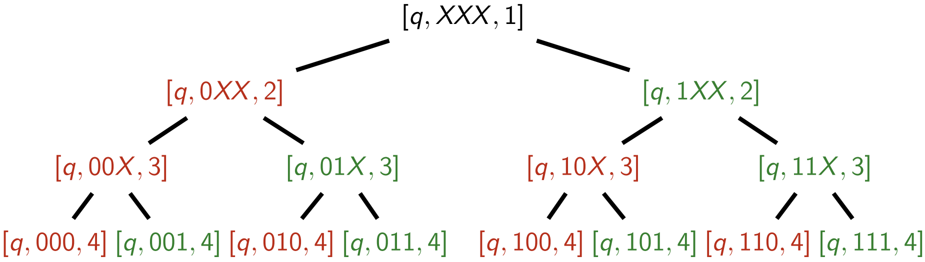
$$\delta(q, X) \to \{(q, 0, +1), (q, 1, +1)\}$$



Nondeterminism to the Rescue

Nondeterminism allows a Turing machine to find a certificate: For example, to turn any sequence of n consecutive X's into all 0/1 vectors of length n, use the following transitions:

$$\delta(q, X) \to \{(q, 0, +1), (q, 1, +1)\}$$



A Nondeterministic Algorithm for SAT

Intuition:

- Guess an assignment (candidate certificate) and
- verify that it satisfies the formula (i.e., that it is a certificate)

A Nondeterministic Algorithm for SAT

Intuition:

- Guess an assignment (candidate certificate) and
- verify that it satisfies the formula (i.e., that it is a certificate)

Given input w:

1. Check whether w encodes a formula φ of propositional logic. If not, reject
2. Determine the set of propositions (variables) in φ
3. For each proposition, guess a truth value
4. Evaluate the formula w.r.t. these truth values
5. If it evaluates to 1, accept. Otherwise, reject

A Nondeterministic Algorithm for SAT

Intuition:

- Guess an assignment (candidate certificate) and
- verify that it satisfies the formula (i.e., that it is a certificate)

Given input w:

1. Check whether w encodes a formula φ of propositional logic. If not, reject $\mathcal{O}(|w|^2)$
2. Determine the set of propositions (variables) in φ
3. For each proposition, guess a truth value
4. Evaluate the formula w.r.t. these truth values
5. If it evaluates to 1, accept. Otherwise, reject

A Nondeterministic Algorithm for SAT

Intuition:

- Guess an assignment (candidate certificate) and
- verify that it satisfies the formula (i.e., that it is a certificate)

Given input w:

1. Check whether w encodes a formula φ of propositional logic. If not, reject $\mathcal{O}(|w|^2)$
2. Determine the set of propositions (variables) in φ $\mathcal{O}(|w|)$
3. For each proposition, guess a truth value
4. Evaluate the formula w.r.t. these truth values
5. If it evaluates to 1, accept. Otherwise, reject

A Nondeterministic Algorithm for SAT

Intuition:

- Guess an assignment (candidate certificate) and
- verify that it satisfies the formula (i.e., that it is a certificate)

Given input w:

1. Check whether w encodes a formula φ of propositional logic. If not, reject $\mathcal{O}(|w|^2)$
2. Determine the set of propositions (variables) in φ $\mathcal{O}(|w|)$
3. For each proposition, guess a truth value $\mathcal{O}(|w|)$
4. Evaluate the formula w.r.t. these truth values
5. If it evaluates to 1, accept. Otherwise, reject

A Nondeterministic Algorithm for SAT

Intuition:

- Guess an assignment (candidate certificate) and
- verify that it satisfies the formula (i.e., that it is a certificate)

Given input w:

1. Check whether w encodes a formula φ of propositional logic. If not, reject $\mathcal{O}(|w|^2)$
2. Determine the set of propositions (variables) in φ $\mathcal{O}(|w|)$
3. For each proposition, guess a truth value $\mathcal{O}(|w|)$
4. Evaluate the formula w.r.t. these truth values $\mathcal{O}(|w|^2)$
5. If it evaluates to 1, accept. Otherwise, reject

A Nondeterministic Algorithm for SAT

Intuition:

- Guess an assignment (candidate certificate) and
- verify that it satisfies the formula (i.e., that it is a certificate)

Given input w:

1. Check whether w encodes a formula φ of propositional logic. If not, reject $\mathcal{O}(|w|^2)$
2. Determine the set of propositions (variables) in φ $\mathcal{O}(|w|)$
3. For each proposition, guess a truth value $\mathcal{O}(|w|)$
4. Evaluate the formula w.r.t. these truth values $\mathcal{O}(|w|^2)$
5. If it evaluates to 1, accept. Otherwise, reject $\mathcal{O}(1)$

Run Time of a Nondeterministic Turing Machine

Definition

Let M be a halting DTM

- Let $time_M(w)$ denote the number of configurations in the unique run of M on input w

Run Time of a Nondeterministic Turing Machine

Definition

Let M be a halting NTM

- Let $time_M(w)$ denote the maximal number of configurations in any branch of the computation tree of M on input w

Run Time of a Nondeterministic Turing Machine

Definition

Let M be a halting NTM

- Let $time_M(w)$ denote the maximal number of configurations in any branch of the computation tree of M on input w
- Let $T\colon \mathbb{N} \to \mathbb{R}_{>0}$. We say that M runs within time T if $time_M(w) \leq T(|w|)$ for all inputs w (no change here)

Run Time of a Nondeterministic Turing Machine

Definition

Let M be a halting NTM

- Let $time_M(w)$ denote the maximal number of configurations in any branch of the computation tree of M on input w
- Let $T \colon \mathbb{N} \to \mathbb{R}_{>0}$. We say that M runs within time T if $time_M(w) \leq T(|w|)$ for all inputs w (no change here)

We are still interested in classifying problems (languages) according to their asymptotic time complexity

Definition

Let $T \colon \mathbb{N} \to \mathbb{R}_{>0}$ be a function. The complexity class $\mathrm{NT{\scriptsize IME}}(T)$ is defined as

$$\mathrm{NT{\scriptsize IME}}(T) = \{L(M) \mid M \text{ is a halting NTM that runs within time } \mathcal{O}(T)\}$$

The Complexity Class NP

Definition

The complexity class NP (nondeterministic polynomial time) is defined as

$$\mathrm{NP} = \bigcup_{k \geq 0} \mathrm{NT\scriptstyle{IME}}(n^k) \qquad \left[\text{Reminder: } \mathrm{P} = \bigcup_{k \geq 0} \mathrm{T\scriptstyle{IME}}(n^k)\right]$$

- Robust definition (can use other nondeterministic models of computation, e.g., multi-tape Turing machines)

The Complexity Class NP

Definition

The complexity class NP (nondeterministic polynomial time) is defined as

$$\mathrm{NP} = \bigcup_{k \geq 0} \mathrm{NT}\textsc{ime}(n^k) \qquad \left[\text{Reminder: } \mathrm{P} = \bigcup_{k \geq 0} \mathrm{T}\textsc{ime}(n^k)\right]$$

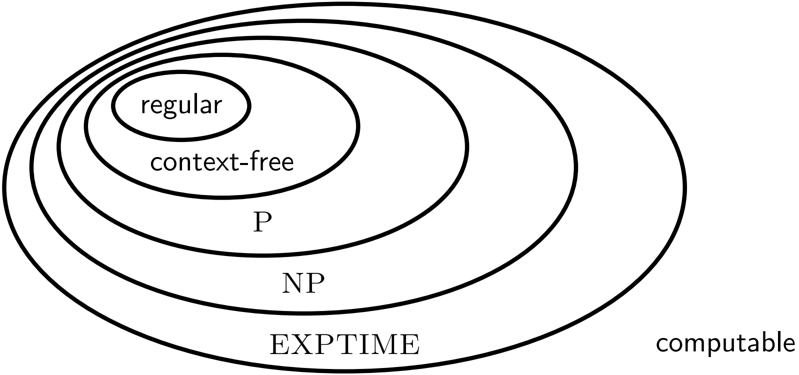
- Robust definition (can use other nondeterministic models of computation, e.g., multi-tape Turing machines)

Remark

$\mathrm{P} \subseteq \mathrm{NP}$

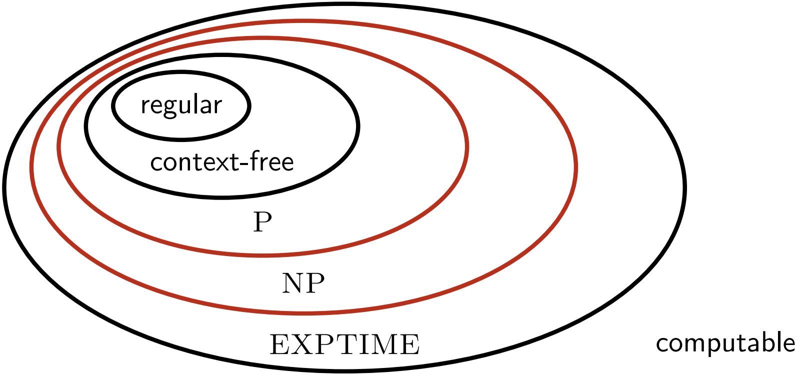
- Because every halting DTM with polynomial time complexity is a halting NTM with polynomial time complexity

Complexity Classes



- EXPTIME: problems solved by DTMs in exponential time

Complexity Classes



- EXPTIME: problems solved by DTMs in exponential time
- It is unknown whether the marked inclusions are strict
- We only know that $\mathrm{P} \subsetneq \mathrm{EXPTIME}$

Another Example

- A nondeterministic polynomial-time algorithm for Clique
- Recall that we encode a graph G by (a word over $\mathbb{B}$ representing) its adjacency matrix, i.e., entry (i, j) is 1 if and only if there is an edge from the i-th to the j-th vertex

Another Example

- A nondeterministic polynomial-time algorithm for Clique
- Recall that we encode a graph G by (a word over $\mathbb{B}$ representing) its adjacency matrix, i.e., entry (i, j) is 1 if and only if there is an edge from the i-th to the j-th vertex

On input $w \in \mathbb{B}^*$ and $k \in \mathbb{N}$ (given in binary):

1. Check whether $|w| = n^2$ for some $n \in \mathbb{N}$. If not, reject
2. If $k > n$, reject
3. Guess a bit vector $c_1 \cdots c_n$ of length n with exactly k 1's
4. For each pair (i, j) with $c_i = c_j = 1$, check whether there is an edge from the i-th to the j-th vertex
5. If yes, accept. Otherwise, reject

Another Example

- A nondeterministic polynomial-time algorithm for Clique
- Recall that we encode a graph G by (a word over $\mathbb{B}$ representing) its adjacency matrix, i.e., entry (i, j) is 1 if and only if there is an edge from the i-th to the j-th vertex

On input $w \in \mathbb{B}^*$ and $k \in \mathbb{N}$ (given in binary):

1. Check whether $|w| = n^2$ for some $n \in \mathbb{N}$. If not, reject

   $\mathcal{O}(|w|^3)$
2. If $k > n$, reject
3. Guess a bit vector $c_1 \cdots c_n$ of length n with exactly k 1's
4. For each pair (i, j) with $c_i = c_j = 1$, check whether there is an edge from the i-th to the j-th vertex
5. If yes, accept. Otherwise, reject

Another Example

- A nondeterministic polynomial-time algorithm for Clique
- Recall that we encode a graph G by (a word over $\mathbb{B}$ representing) its adjacency matrix, i.e., entry (i, j) is 1 if and only if there is an edge from the i-th to the j-th vertex

On input $w \in \mathbb{B}^*$ and $k \in \mathbb{N}$ (given in binary):

1. Check whether $|w| = n^2$ for some $n \in \mathbb{N}$. If not, reject $\mathcal{O}(|w|^3)$
2. If $k > n$, reject $\mathcal{O}(|w|)$
3. Guess a bit vector $c_1 \cdots c_n$ of length n with exactly k 1's
4. For each pair (i, j) with $c_i = c_j = 1$, check whether there is an edge from the i-th to the j-th vertex
5. If yes, accept. Otherwise, reject

Another Example

- A nondeterministic polynomial-time algorithm for Clique
- Recall that we encode a graph G by (a word over $\mathbb{B}$ representing) its adjacency matrix, i.e., entry (i, j) is 1 if and only if there is an edge from the i-th to the j-th vertex

On input $w \in \mathbb{B}^*$ and $k \in \mathbb{N}$ (given in binary):

1. Check whether $|w| = n^2$ for some $n \in \mathbb{N}$. If not, reject $\mathcal{O}(|w|^3)$
2. If $k > n$, reject $\mathcal{O}(|w|)$
3. Guess a bit vector $c_1 \cdots c_n$ of length n with exactly k 1's $\mathcal{O}(|w|)$
4. For each pair (i, j) with $c_i = c_j = 1$, check whether there is an edge from the i-th to the j-th vertex
5. If yes, accept. Otherwise, reject

Another Example

- A nondeterministic polynomial-time algorithm for Clique
- Recall that we encode a graph G by (a word over $\mathbb{B}$ representing) its adjacency matrix, i.e., entry (i, j) is 1 if and only if there is an edge from the i-th to the j-th vertex

On input $w \in \mathbb{B}^*$ and $k \in \mathbb{N}$ (given in binary):

1. Check whether $|w| = n^2$ for some $n \in \mathbb{N}$. If not, reject $\mathcal{O}(|w|^3)$
2. If $k > n$, reject $\mathcal{O}(|w|)$
3. Guess a bit vector $c_1 \cdots c_n$ of length n with exactly k 1's $\mathcal{O}(|w|)$
4. For each pair (i, j) with $c_i = c_j = 1$, check whether there is an edge from the i-th to the j-th vertex $\mathcal{O}(|w|^3)$
5. If yes, accept. Otherwise, reject

Another Example

- A nondeterministic polynomial-time algorithm for Clique
- Recall that we encode a graph G by (a word over $\mathbb{B}$ representing) its adjacency matrix, i.e., entry (i, j) is 1 if and only if there is an edge from the i-th to the j-th vertex

On input $w \in \mathbb{B}^*$ and $k \in \mathbb{N}$ (given in binary):

1. Check whether $|w| = n^2$ for some $n \in \mathbb{N}$. If not, reject $\mathcal{O}(|w|^3)$
2. If $k > n$, reject $\mathcal{O}(|w|)$
3. Guess a bit vector $c_1 \cdots c_n$ of length n with exactly k 1's $\mathcal{O}(|w|)$
4. For each pair (i, j) with $c_i = c_j = 1$, check whether there is an edge from the i-th to the j-th vertex $\mathcal{O}(|w|^3)$
5. If yes, accept. Otherwise, reject $\mathcal{O}(1)$

Problems in NP

The halting NTMs we have seen show that

- SAT and
- Clique

are in NP

Problems in NP

The halting NTMs we have seen show that

- SAT and
- Clique

are in NP

In fact, the other problems we have seen are also in NP:

- Nonprimality
- Hamiltonian path
- Linear programming
- 3-coloring
- And many more

Quiz 2

Is

$$\{\varphi \mid \text{all assignments satisfy } \varphi\}$$

in NP?

Quiz 2

Is

$$\{\varphi \mid \text{all assignments satisfy } \varphi\}$$

in NP?

- Nobody knows!
- We have swapped existential for universal quantification
- Suddenly, we need to test all assignments – nondeterminism does not help anymore

Quiz 2

Is

$$\{\varphi \mid \text{all assignments satisfy } \varphi\}$$

in NP?

- Nobody knows!
- We have swapped existential for universal quantification
- Suddenly, we need to test all assignments – nondeterminism does not help anymore
- However, if we find one assignment that does not satisfy φ, we can answer "no" ("dual problem" of SAT)
- Hence, a nondeterministic algorithm can solve the complement problem efficiently

Quiz 2

Is

$$\{\varphi \mid \text{all assignments satisfy } \varphi\}$$

in NP?

- Nobody knows!
- We have swapped existential for universal quantification
- Suddenly, we need to test all assignments – nondeterminism does not help anymore
- However, if we find one assignment that does not satisfy φ, we can answer "no" ("dual problem" of SAT)
- Hence, a nondeterministic algorithm can solve the complement problem efficiently
- Open question whether this problem and SAT have different complexity

Agenda

1. Motivation
2. Nondeterministic Time Complexity
3. **Polynomial-Time Reductions**

Polynomial-Time Computable Functions

Definition

A function $f\colon \Sigma_1^* \to \Sigma_2^*$ is a computable function
if and only if there exists a DTM M_f
that on every input $w \in \Sigma_1^*$

- always halts and accepts
- with just $f(w)$ on its tape and
- the head at the first letter of $f(w)$

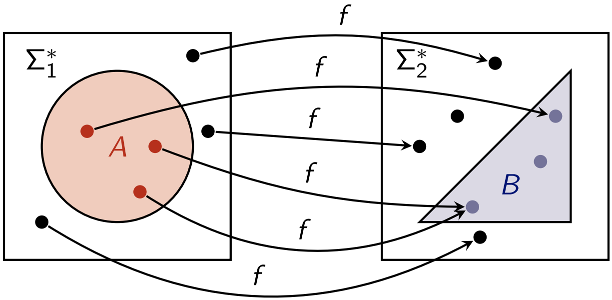
Definition

A function $f\colon \Sigma_1^* \to \Sigma_2^*$ is a polynomial-time computable function if and only if there exists a DTM M_f with polynomial time complexity that on every input $w \in \Sigma_1^*$

- always halts and accepts
- with just $f(w)$ on its tape and
- the head at the first letter of $f(w)$

Examples

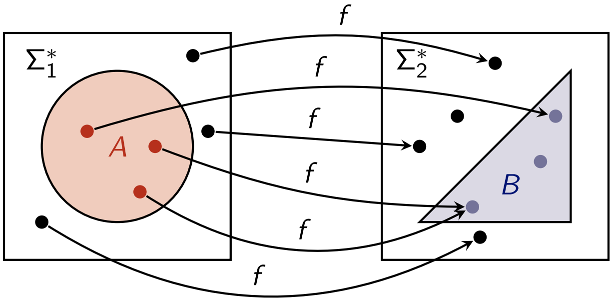
- $f(w) = ww$ is polynomial-time computable
- $f(\ulcorner m \urcorner \# \ulcorner n \urcorner) = \ulcorner m \cdot n \urcorner$ is polynomial-time computable (where $\ulcorner n \urcorner$ denotes the binary encoding of $n \in \mathbb{N}$)



Definition

Let $A \subseteq \Sigma_1^*$ and $B \subseteq \Sigma_2^*$ be languages. We say that A is mapping reducible to B, written $A \leq_m B$, if and only if

1. there is a computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that
2. for every $w \in \Sigma_1^*$: $w \in A \Leftrightarrow f(w) \in B$



Definition

Let $A \subseteq \Sigma_1^*$ and $B \subseteq \Sigma_2^*$ be languages. We say that A is polynomial-time reducible to B, written $A \leq_p B$, if and only if

1. there is a polynomial-time computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that
2. for every $w \in \Sigma_1^*$: $w \in A \Leftrightarrow f(w) \in B$

Applications

Recall: Let $A \leq_m B$

- If B is computable, then A is computable
- If B is computably-enumerable, then A is computably-enumerable

A similar result holds for polynomial-time reductions and the complexity classes P and NP

Theorem

Let $A \leq_p B$

1. *If $B \in \mathrm{P}$, then $A \in \mathrm{P}$*
2. *If $B \in \mathrm{NP}$, then $A \in \mathrm{NP}$*

Proof

We prove "If $B \in \mathrm{P}$, then $A \in \mathrm{P}$". The proof for NP is analogous

- Let M be a halting DTM for B that runs within time $\mathcal{O}(n^k)$

Proof

We prove "If $B \in \mathrm{P}$, then $A \in \mathrm{P}$". The proof for NP is analogous

- Let M be a halting DTM for B that runs within time $\mathcal{O}(n^k)$
- As $A \leq_p B$, there is a polynomial-time computable function f such that $w \in A \Leftrightarrow f(w) \in B$. Then, there is a DTM M' computing f; say it runs within time $\mathcal{O}(n^{k'})$

Proof

We prove "If $B \in \mathrm{P}$, then $A \in \mathrm{P}$". The proof for NP is analogous

- Let M be a halting DTM for B that runs within time $\mathcal{O}(n^k)$
- As $A \leq_p B$, there is a polynomial-time computable function f such that $w \in A \Leftrightarrow f(w) \in B$. Then, there is a DTM M' computing f; say it runs within time $\mathcal{O}(n^{k'})$
- Consider the following DTM M'': On input w:
  1. Simulate M' (yielding $f(w)$ on the tape)
  2. Simulate M on $f(w)$
  3. Accept if M accepts $f(w)$, reject if M rejects $f(w)$

Proof

We prove "If $B \in \mathrm{P}$, then $A \in \mathrm{P}$". The proof for NP is analogous

- Let M be a halting DTM for B that runs within time $\mathcal{O}(n^k)$
- As $A \leq_p B$, there is a polynomial-time computable function f such that $w \in A \Leftrightarrow f(w) \in B$. Then, there is a DTM M' computing f; say it runs within time $\mathcal{O}(n^{k'})$
- Consider the following DTM M'': On input w:
  1. Simulate M' (yielding $f(w)$ on the tape)
  2. Simulate M on $f(w)$
  3. Accept if M accepts $f(w)$, reject if M rejects $f(w)$
- M'' is halting, as M' and M both always halt

Proof

We prove “If $B \in \mathrm{P}$, then $A \in \mathrm{P}$”. The proof for NP is analogous

- Let M be a halting DTM for B that runs within time $\mathcal{O}(n^k)$
- As $A \leq_p B$, there is a polynomial-time computable function f such that $w \in A \Leftrightarrow f(w) \in B$. Then, there is a DTM M' computing f; say it runs within time $\mathcal{O}(n^{k'})$
- Consider the following DTM M'': On input w:
  1. Simulate M' (yielding $f(w)$ on the tape)
  2. Simulate M on $f(w)$
  3. Accept if M accepts $f(w)$, reject if M rejects $f(w)$
- M'' is halting, as M' and M both always halt
- M'' computes A because $w \in A \Leftrightarrow f(w) \in B$

Proof

We prove "If $B \in \mathrm{P}$, then $A \in \mathrm{P}$". The proof for NP is analogous

- Let M be a halting DTM for B that runs within time $\mathcal{O}(n^k)$
- As $A \leq_p B$, there is a polynomial-time computable function f such that $w \in A \Leftrightarrow f(w) \in B$. Then, there is a DTM M' computing f; say it runs within time $\mathcal{O}(n^{k'})$
- Consider the following DTM M'': On input w:
  1. Simulate M' (yielding $f(w)$ on the tape)
  2. Simulate M on $f(w)$
  3. Accept if M accepts $f(w)$, reject if M rejects $f(w)$
- M'' is halting, as M' and M both always halt
- M'' computes A because $w \in A \Leftrightarrow f(w) \in B$
- M'' runs within time $(n^{k'})^k = n^{k' \cdot k}$
- Hence, $A \in \mathrm{P}$

Conclusion

We have seen:

- The complexity class NP: The class of languages accepted by polynomial-time NTMs
- Polynomial-time reductions

Conclusion

We have seen:

- The complexity class NP: The class of languages accepted by polynomial-time NTMs
- Polynomial-time reductions

Recall: $\mathrm{P} \subseteq \mathrm{NP}$

The (literally) Million Dollar Question

Is verifying certificates easier than finding certificates or not:

$$\mathrm{P} = \mathrm{NP} \text{ or } \mathrm{P} \subsetneq \mathrm{NP}?$$

- One of the most challenging and most important questions of (theoretical) computer science
- More on that next time

Reading

Sections 3.1 to 3.4 of "Computability and Complexity"
(pages 141 to 178):

Note

- The book uses slightly different notation and definitions
- These sections also cover the complement class of NP (called coNP), which is not covered in this course