Advanced Algorithms CS3172, 02/03

Time: Monday and Friday, 10-11.

www: Do check the website regularly.

Lecture Course comes in 2 parts:
Part I (Dix) Introduction to Complexity Classes,
Part II (Rydeheard) Specific Algorithms.

Organisation:

Part I (Dix): 8 lectures, one week free (3/7 March), one week to discuss the homework (10/14 March).

Part II (Rydeheard): 8 lectures, one week free (5/9 May), one week to discuss the homework (12/16 May).

Exam:

Overview

1. Turing Machines
2. Complexity Classes
3. Hierarchies, Complete Problems

3 Hierarchies, P/NP

3.1 The Structure of PSPACE

3.2 Completeness, Hardness

3.3 Examples

3.1 The structure of PSPACE

We define the most important complexity classes and look into their structure.

Definition 3.1 (P, NP, PSPACE, NSPACE)
We define the following complexity classes:

$$\begin{aligned}
\mathbf{P} &:= \bigcup_{i\geqslant 1} \mathbf{DTIME}(n^i) \\
\mathbf{NP} &:= \bigcup_{i\geqslant 1} \mathbf{NTIME}(n^i) \\
\mathbf{PSPACE} &:= \bigcup_{i\geqslant 1} \mathbf{DSPACE}(n^i) \\
\mathbf{NSPACE} &:= \bigcup_{i\geqslant 1} \mathbf{NSPACE}(n^i)
\end{aligned}$$

The intuition is as follows:

- Problems in **P** are efficiently solvable,
  whereas those in **NP** require exponential time.
- **PSPACE** is a huge class, way above **P** and **NP**.
- **DSPACE**$(\log n)$ is a small class within **P** and very small in **PSPACE**.

What are the precise relations between the complexity classes introduced in Definition 3.1?

Using Theorem 2.6 it is obvious that (remember the hierarchies introduced on slide 65)

$$\begin{array}{ccc} \textbf{PSPACE} & = & \textbf{NSPACE} \\ \bigcup_{i \geqslant 1} \textbf{NSPACE}(\log^i n) & = & \bigcup_{i \geqslant 1} \textbf{DSPACE}(\log^i n) \end{array}$$

This gives us (using Theorem 2.7)

$$\textbf{DSPACE}(\log n) \subseteq \textbf{P} \subseteq \textbf{NP} \subseteq \textbf{PSPACE}$$

where at least one containment is proper (but it is unknown which).

- How do we show that a particular problem is in a certain class?

Reduce it to a known one!

But we have to have one to start with! Yes, and this is the most difficult step. A suitable problem is SAT introduced in Example 1.2.

- Can we find problems in a class that are the most difficult ones in that class?

There are several ways to define a most difficult problem. They depend on which notion of reducibility we use.

Based on a particular reducibility notion, the answer to our question is Yes: such problems are called **complete problems** for the given class under the chosen reducibility notion.

Definition 3.2 (polynomial-time-, log-space reducibility)

Let $\mathcal{L}_1, \mathcal{L}_2$ be languages.

poly-time: We say that $\mathcal{L}_2$ is poly-time reducible to $\mathcal{L}_1$, if there is a **polynomial-time bounded DTM** that produces for each input w an output $f(w)$ such that

$$w \in \mathcal{L}_2 \text{ if and only if } f(w) \in \mathcal{L}_1$$

log-space: We say that $\mathcal{L}_2$ is log-space reducible to $\mathcal{L}_1$, if there is a $\log n$ **space bounded offline DTM that always halts** and produces for each input w an output $f(w)$ such that

$$w \in \mathcal{L}_2 \text{ if and only if } f(w) \in \mathcal{L}_1$$

The output tape is write-only and the head never moves left. Space on the output tape is not counted.

The notions just introduced are important because of the following

Lemma 3.1 (Properties of poly-time and log-space reductions)

1. Let $\mathcal{L}_2$ be poly-time reducible to $\mathcal{L}_1$. Then

$\mathcal{L}_2$ is in **NP**	if and only if	$\mathcal{L}_1$ is in **NP**
$\mathcal{L}_2$ is in **P**	if and only if	$\mathcal{L}_1$ is in **P**

2. Let $\mathcal{L}_2$ be log-space reducible to $\mathcal{L}_1$. Then

$\mathcal{L}_2$ is in **P**	if and only if	$\mathcal{L}_1$ is in **P**
$\mathcal{L}_2$ is in **DSPACE**$(\log^k n)$	if and only if	$\mathcal{L}_1$ is in **DSPACE**$(\log^k n)$
$\mathcal{L}_2$ is in **NSPACE** $(\log^k n)$	if and only if	$\mathcal{L}_1$ is in **NSPACE** $(\log^k n)$

3. The composition of two log-space (resp. poly-time) reductions is itself a log-space (resp. poly-time) reduction.

4. Log-space reducibility implies poly-time reducibility.

The next slide shows a log-space reduction of $\mathcal{L}_{\text{sat}}$ to $\mathcal{L}_{\text{ILP}}$.

I am here after the seventh lecture.

Recall the integer linear programming problem $\mathcal{L}_{\text{ILP}}$ from slide 28. How is this problem related to $\mathcal{L}_{\text{sat}}$?

Suppose we are given a formula $\phi : \psi_1 \wedge \cdots \wedge \psi_l$ where each ψ_i is a disjunction of literals (over variables $x_1, \ldots, x_n$).

We construct an ILP problem $\langle \mathbf{A}, \mathbf{b} \rangle$ as follows:

Matrix A: $2n$ columns of $\mathbf{A}$ correspond to $x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}$. The first n rows each contain exactly two consecutive 1's (starting from position i at row i). The next n rows each contain exactly two consecutive -1's (starting from position i at row $2i$). The next l rows contain 1' at exactly the positions determined by the clauses ψ_i. Finally, we have n rows containing only one 1 (at position x_i) and another n rows containing only one 1 (at position $\overline{x_i}$).

Vector b: The first n entries are 1, the next n entries are -1, the next l (number of clauses) are 1. Finally, we have $2n$ entries which are 0.

$$\phi \in \mathcal{L}_{\text{sat}} \text{ if and only if } \langle \mathbf{A}, \mathbf{b} \rangle \in \mathcal{L}_{\text{ILP}}$$

Why? Just split off the matrix multiplication into its many inequalities and find out what they really mean.

Theorem 3.1 (Characterisation of NP)

A language $\mathcal{L}$ is in **NP** if and only if there is a language $\mathcal{L}'$ in **P** and there is $k \geq 0$ such that for all $w \in \Sigma$

$$w \in \mathcal{L} \text{ if and only if } \text{there is } c : \langle w, c \rangle \in \mathcal{L}' \text{ and } |c| < |w|^k.$$

c is called a witness (or certificate) of w in $\mathcal{L}$. A DTM accepting the language $\mathcal{L}'$ is called a verifier of $\mathcal{L}$.

A decision problem is in **NP**, if and only if every yes-instance has a succinct certificate (i.e. its length is polynomial in the length of the input) which can be verified in polynomial time.

E.g. checking "nonprimeness", graphs being hamiltonian, satisfiability.

3.2 Completeness, Hardness

Definition 3.3 (Completeness, Hardness)

A language $\mathcal{L}$ is called **NP complete**, if $\mathcal{L} \in \mathbf{NP}$ and every language $\mathcal{L}' \in \mathbf{NP}$ is log-space reducible to $\mathcal{L}$.

A language $\mathcal{L}$ is called **NP hard**, if every language $\mathcal{L}' \in \mathbf{NP}$ is log-space reducible to $\mathcal{L}$.

A language $\mathcal{L}$ is called **PSPACE complete**, if $\mathcal{L} \in \mathbf{PSPACE}$ and every language $\mathcal{L}' \in \mathbf{PSPACE}$ is poly-time reducible to $\mathcal{L}$.

A language $\mathcal{L}$ is called **NSPACE hard**, if every language $\mathcal{L}' \in \mathbf{PSPACE}$ is poly-time reducible to $\mathcal{L}$.

It is worth noting the following:

- If a **NP** hard problem is shown to be in **P**, then **P =NP**.
- If a **PSPACE** hard problem is shown to be in **P**, then **P =PSPACE**.
- If **P** $\neq$ **NP**, then no **NP** complete problem is solvable in polynomial time.

Complexity classes defined on deterministic Turing machines are closed under complementation: If a language $\mathcal{L}$ belongs to it, then its complement belongs to it as well (just run the old machine and swap the answers).

Does that hold for NP as well?

Nobody knows!!

The last complexity class we introduce is **co-NP**.

Definition 3.4 (co-NP)
co-NP is the class of languages $\mathcal{L}$ whose complements $\overline{\mathcal{L}}$ $(= \Sigma^* \setminus \mathcal{L})$ are in **NP**.

The following are **unknown** to science:

1. **P $\neq$ NP.**
2. **NP $\neq$ co-NP.**
3. **P $\neq$ PSPACE.**
4. **NP $\neq$ PSPACE.**

3.3 Some reductions

SAT, 3-CNF are NP complete

Definition 3.5 (SAT, k-CNF, k-DNF)
We have already introduced the satisfiability problem in Example 1.2 on slide 25. A *literal* l_i is a variable x_i or its negation $\neg x_i$. A *clause* is a disjunction of literals. Let us define the following:

DNF: A formula is in *disjunctive normal form* (DNF), if it is of the form $(l_{11} \wedge \ldots \wedge l_{1n_1}) \vee \ldots \vee (l_{m1} \wedge \ldots \wedge l_{mn_m})$.

CNF: A formula is in *conjunctive normal form* (CNF), if it is of the form $(l_{11} \vee \ldots \vee l_{1n_1}) \wedge \ldots \wedge (l_{m1} \vee \ldots \vee l_{mn_m})$.

k-DNF: A formula is in k-DNF, if it is in DNF and every disjunct contains exactly k literals.

k-CNF: A formula is in k-CNF, if it is in CNF and every conjunct contains exactly k literals.

We also use the notation k-SAT for k-CNF: each clause contains at most k literals (they can of course vary from clause to clause, so that the whole formula can contain an arbitrary number of literals).

DNF: What is the time complexity of the satisfiability problem for formulae in DNF?

2-CNF: What is the time complexity for the satisfiability problem for 2-CNF?

Reducibility: How can we reduce a satisfiability problem in CNF to one into DNF and vice versa? What does this imply about the poly-time or log-space reducibility of SAT wrt to DNF and SAT wrt. CNF to each other?

Theorem 3.2 (NP complete Problems)
The following are **NP** complete problems: $\mathcal{L}_{\text{sat}}$, $\mathcal{L}_{\text{3-sat}}$, integer linear programming, the problem whether a graph contains a hamiltonian cycle.

Proof: not trivial! ∎

3.4 Graph Reachability, Vertex Cover

Theorem 3.3 (Completeness of $\mathcal{L}_{\text{reach}}$)
$\mathcal{L}_{\text{reach}}$ (graph reachability) is log-space complete for **NSPACE**$(\log n)$.

Theorem 3.4 (Completeness of Vertex Cover)
Vertex Cover is **NP** complete.

Homework 4 (Reducibilities, P/NP)

1. Suppose there is a function $f : \mathbb{N} \to \mathbb{N}$ mapping integers of length k onto integers of length k such that

   (a) $f \in \mathbf{P}$,

   (b) $f^{-1} \notin \mathbf{P}$.

   Show that

   $$\{\langle x,y\rangle : \ f^{-1}(x) < y\} \in (\mathbf{NP} \cap \mathbf{co\text{-}NP}) \setminus \mathbf{P}$$

2. Suppose we have a DTM M that decides a language $\mathcal{L}$. If $x \in \mathcal{L}$, M says yes in polynomial time. If $x \notin \mathcal{L}$, M says no in exponential time. Is $\mathcal{L} \in \mathbf{P}$?

3. Classify the following problems to **t** (true), **f** (false), or **u** (unknown to science, i.e. depending on the P/NP problem).
   - SAT $\in \mathbf{P}$.
   - If a problem in **NP** can be solved in polynomial time, then $\mathbf{P} = \mathbf{NP}$.
   - If a problem in **NP** can be shown not to be solvable in polynomial time, then $\mathbf{P} \neq \mathbf{NP}$.
   - SAT can be polynomially reduced to 1-CNF.
   - All problems in **P** can be polynomially reduced to each other.
   - Some problems in **NP** can not be polynomially reduced to each other.

References

Hopcroft, J. and J. Ullman (1979). *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison Wesely.

Lin, S. and T. Rado (1965). Computer studies of turing machine problems. *Journal of the ACM 12*(2), 196–212.

Papadimitriou, C. (1994). *Computational Complexity*. Addison-Wesley.

Rado, T. (1962). On non-computable functions. *The Bell System Technical Journal XLI*(3), 877–884.

Turing, A. M. (1936). On Computable Numbers with an Application to the Entscheidungsproblem. *Proc. London Mathematical Soc., series 2 42*, 230–265. corrections ibid., 43:544–546.