80240233: Computational Complexity
ITCS, Tsinghua Univesity, Fall 2007

Lecture 2
12 October 2007

Instructor: Andrej Bogdanov

Notes by: Decheng DAI

This lecture we are going to discuss NP-completeness and the classes of decision problems, such as NP, EXP and NEXP the relations between them.

1 NP completeness

Many problems in the class NP appear to be difficult in the same way; there is no better way of finding a solution than doing an exhaustive search over all possible witnesses. NP-completeness gives a partial explanation of this phenomenon. We will focus on decision problems; a similar notion can also be defined for search problems.

For two decision problems A and B, we say A (polynomial-time) reduces to B, denoted by $A \leq B$, if there is a polynomial-time computable function f (a reduction) such that $x \in A$ if and only if $f(x) \in B$. So if $A \leq B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$. We say a decision problem L is NP-complete if $L \in \mathrm{NP}$ and $A \leq L$ for all $A \in \mathrm{NP}$. So an NP-complete problem is in P if and only if $\mathrm{P} = \mathrm{NP}$.

NP-complete problems are abundant, and this is a remarkable phenomenon. However their mere existence is interesting already. Here is an example:

BH (BOUNDED HALTING): On input $(\langle N \rangle, x, 1^t)$, where $\langle N \rangle$ is the description of a NTM, accept iff N accepts x in time at most t.

Claim 1. BH *is* NP*-complete.*

Proof. Showing that $\mathrm{BH} \in \mathrm{NP}$ requires a bit of technical work. We have to design a NTM for BH that efficiently simulates Turing Machine N which is provided to us explicitly as an input. This can be done in several ways; any reasonable simulation should have this property. We won't worry about the details of this.

Now let A be an NP decision problem and let N be a non-deterministic Turing Machine that decides A. Suppose A runs in time at most $p(n)$ on inputs of length n. The reduction f will then map instance x of A to instance $(\langle N \rangle, x, 1^{p(|x|)})$ of BH. Now it is easy to see that $x \in A$ if and only if N accepts x in time $p(|x|)$, which in turn holds if and only if $(\langle N \rangle, x, 1^{p(|x|)}) \in \mathrm{BH}$. □

The NP-completeness of BH is interesting, though for us it will be much more convenient to work with other NP-complete problems. In a few lectures we will show that $\mathrm{BH} \leq \mathrm{SAT}$. Since reductions are transitive, we obtain the following important consequence.

Theorem 2 (Cook, Levin). SAT *is* NP*-complete.*

2 EXP and NEXP

The following two classes are exponential time analogues of P and NP.

EXP is the class of all decision problems that can be solved by TMs running in the $2^{O(n^c)}$ for some positive constant c. We can define similarly, NEXP is the class of all decision problems that can be solved by NTMs running in the $2^{O(n^c)}$ for some positive constant c.

For that it is easy to simulate the nondeterminism by enumerating all possible strings, we have enough reason to guess that NP is in the EXP.

Claim 3. NP $\subseteq$ EXP

Proof. We simulate every NTM N that runs in time $t(n)$ by a TM M that runs in time $2^{O(t(n))}$.

Because N can read at most $t(n)$ symbols on its input type, so we use M to enumerate all possible ways N can run, this takes at most $O(t(n) \times 2^{t(n)}) = 2^{O(t(n))}$ steps by M. □

Because every problem in NP can be solved in exponential time by a brute force search for the certificate, and we can use a NTM to simulate a DTM , P $\subseteq$ NP $\subseteq$ EXP $\subseteq$ NEXP. Although we don't know whether P $\neq$ NP or NP $\neq$ EXP. But a basic idea lead us to think that the more running time given, the more things the TM can do. So we will show that P $\neq$ EXP and NP $\neq$ NEXP in the next section.

3 Time Hierarchy Theorems

The Time Hierarchy Theorem shows that allowing Turing Machines more computation time strictly increases the class of languages that they can decide.

The idea of hierarchy theorem is to construct a DTM D which always runs in exponential time and output differently with any DTM running in polynomial time.

Consider the following Turing Machine D:

D: Given input $(\langle M\rangle, x)$,
Run M on input $(\langle M\rangle, x)$ for $|x|^{\log|x|}$ steps, and
If M accepts, reject;
Otherwise (if M rejects or doesn't halt), accept.

It is easy to see that, for any given input with length n, D will run $n^{\log(n)}$ steps and it can not be simulated in polynomial time TM. The second thing is that for any given TM, it outputs the different result with it. So it is sufficient for us to claim that D is not in P.

Claim 4. *Turing machine D runs in exponential time.*

Proof. Assume that D simulates the TM M for t steps. In every step, it can scan the whole type including the description for $\langle M\rangle$ and x. And after this it can know what simulation it can do.

So the simulation can be done in time,

$$O(t \times (|\langle M\rangle| + |x| + t)) = O(|x|^{\log|x|} \times (|\langle M\rangle| + |x| + |x|^{\log|x|})) = 2^{O(|\langle M\rangle|+|x|)} \qquad \square$$

This claim confirm that D runs in exponential time,and the next claim shows it does not decide any problem in P.

Claim 5. *for every M that runs in polynomial time, there exist an input x, such that $D(\langle M\rangle, x) \neq M(\langle M\rangle, x)$*

Proof. Suppose M runs in time $t(n)$,and choose an input x, s.t. $|x|^{\log|x|} > t(|x|)$, then D can simulate M and output the opposite.

$$D(\langle M\rangle, x) \neq M(\langle M\rangle, x) \qquad \square$$

The idea for nondeterministic version is similarly to construct a NTM S that runs in NEXP, it does something different from every NTM N that runs in polynomial time.

We try to define the NTM similar to the definition of D:

S: Given input $(\langle N\rangle, x)$, where N is a NTM
Run N on input $(\langle N\rangle, x)$ for $|x|^{\log|x|}$ steps, and
If N accepts, reject;
If N rejects or doesn't halt, accept.

It is not easy for S to simulate N and accept/reject an input, for that it has to check all possible non-determine branches of N. So it must takes S to simulate for at least $2^{|x|^{\log|x|}}$ steps. So, we can not use this definition of S. The similar idea can go on only if we can find another input string x_0 which is a very short prefix of x, and S simulate N for input x_0, the simulation steps will be $2^{|x_0|^{\log|x_0|}}$, because $x_0 << x$, it will still be in $2^{O(|x|^c)}$ time. For this reason, we have to choose some "special length" and force N that $N(\langle N\rangle, x) = N(\langle N\rangle, x_0)$ when x is of this special length. So we consider the following Turing Machine S:

S: Given input $(\langle N\rangle, x)$, where N is a NTM
If x of a *special length* l_k,
Simulate $N(\langle N\rangle, x_0)$ for $|x_0|^{\log|x_0|}$ steps deterministically, where
x_0 is a prefix of x of length $l_{k-1} + 1$
If N accepts, reject; otherwise, accept.
Otherwise,
Simulate $N(\langle N\rangle, x1)$ non-deterministically for $|x1|^{\log|x1|}$ steps and
Output same result.

We will define exactly what the "special lengths" are in a minute. As we mentioned before, it must be confirmed that S can simulate in NEXP. When x is of a special length, the running time is

$$2^{|x_0|^{\log|x_0|}} \times (|\langle N\rangle| + |x|)^2 \leq 2^{O(|\langle N\rangle|+|x|)}$$

as long as $|x_0|^{\log|x_0|} \leq |x|$, which we can achieve by defining $l_k = (l_{k-1}+1)^{\log l_{k-1}+1}$. When x is not of a special length, the running time is

$$|x1|^{\log|x1|} \times (|\langle N\rangle| + |x|)^2 \leq 2^{O(|\langle N\rangle|+|x|)}$$

So N runs in exponential time. We now show that S behaves differently from any other polynomial time NTM.

Claim 6. *For every N that runs in non-deterministic polynomial-time, there is an input y s.t. $S(\langle N\rangle, y) \neq N(\langle N\rangle, y)$.*

Proof. For any N, let x be a sufficiently long string of 1s of some special length l_k. Then $N(|N|, x_0)$ will terminate within $|x_0|^{\log|x_0|}$ steps so by construction we must have

$$S(\langle N\rangle, x) \neq N(\langle N\rangle, x_0) \tag{1}$$

If $N(\langle N\rangle, x_0) \neq S(\langle N\rangle, x_0)$, we are done (by choosing $y = x_0$ in the Claim). Otherwise,

$$N(\langle N\rangle, x_0) = S(\langle N\rangle, x_0) = N(\langle N\rangle, x_01)$$

since x_0 is not of a special length. Now if $N(\langle N\rangle, x_01) \neq S(\langle N\rangle, x_01)$, we are done again. Otherwise,

$$N(\langle N\rangle, x_01) = S(\langle N\rangle, x_01) = N(\langle N\rangle, x_011).$$

Continuing in this way, we either find a string of the form $x_01\ldots1$ such that $N(\langle N\rangle, x_01\ldots1) \neq S(\langle N\rangle, x_01\ldots1)$, or it must be that

$$\begin{aligned} N(\langle N\rangle, x_0) &= S(\langle N\rangle, x_0) = N(\langle N\rangle, x_01) \\ &= S(\langle N\rangle, x_01) = N(\langle N\rangle, x_011) \\ &= \ldots \\ &= S(\langle N\rangle, x_01\ldots1) \\ &= S(\langle N\rangle, x) \end{aligned}$$

which contradicts (1). □