Please sign up at Piazza

piazza.com/kth.se/fall2013/dd2446

(even if you are not taking the course for credit but just listening)

If you are taking the course for credit, please register via the KTH "personal menu" (previously known as "my pages").

Complexity theory is deep and conceptually hard stuff (at least for me).

Suggested approach:

- skim chapter <u>before</u> lecture (but don't get stuck)
- Attend lecture
- Afterwards (and ASAP), read lecture notes and chapter again and more carefully

So far — Computational model (Turing machine)
Cplx class P — efficiently solvable (decision) problems
Cplx class NP — efficiently verifiable (decision) problems
CNFSAT & 3-SAT NP-complete
coNP (set of strings not in L is NP-language)
EXP, NEXP

Today: Can we prove that more time allows solving more problem?
Are there any complexity classes between P and NP?
Diagonalization
Oracles

How to prove that two cplx classes are different?
Find a language in one class that is not in the other.

Every language $L \in \mathcal{C}$ decided by some TM M_L that runs within resource bounds specified by $\mathcal{C}$

Separate $\mathcal{C}_1$ and $\mathcal{C}_2$ by finding TM M in $\mathcal{C}_1$ that differs from every TM in $\mathcal{C}_2$ on at least one input

Then $L = \{ x \mid M(x) = 1 \}$ is a language separating $\mathcal{C}_1$ and $\mathcal{C}_2$

Essentially only known tool to do this: DIAGONALIZATION

L 23 II

DIAGONALIZATION

Recall: Turing machine specified by
- finite alphabet Σ (symbols)
- finite set of possible states Q
- transition function (program) mapping
  $Q \times \{$ read symbols on tapes $\}$ to
  $Q \times \{$ written symbols on tapes $\} \times \{$ head movements $\}$

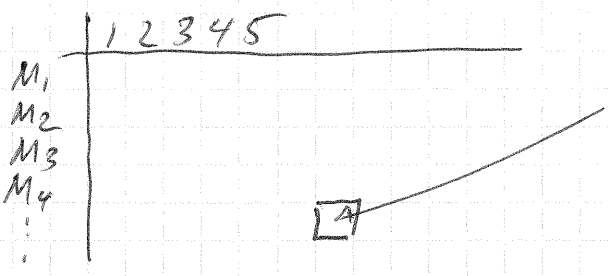
Can agree on some encoding of TMs as (finite) binary strings. Let's use encoding such that
(a) exists "stop marker", and padding with more bits after stop marker has no effect but encodes same machine.
(b) "syntax error" encoding identified with trivial TM that immediately halts and rejects, say.

Then
(1) Every string $x \in \{0,1\}^*$ represents a TM M_x. Given $i \in \mathbb{N}$ write M_i to denote Turing machine encoded by i written in binary
(2) Every TM M is represented by infinitely many strings / infinitely many integers
(3) This representation is efficient in that given x, we can simulate M_x on the universal Turing machine $\mathcal{U}$ with at most a logarithmic overhead.

Write table with rows and columns indexed by integers.

Interpret: Rows $\Leftrightarrow$ TMs

Columns $\Leftrightarrow$ inputs



(i,j) contains $M_i(j)$

output of TM M_i on input j (in binary)

~~Construct~~ TM by walking diagonally downwards to the left, making sure at least one mismatch per row $\Rightarrow$ Contradiction; such TM can't exist.

TIME HIERARCHY THEOREM

If f, g time-constructible functions satisfying $f(n) \log f(n) = o(g(n))$, then

$$DTIME(f(n)) \subsetneq DTIME(g(n))$$

$\subsetneq$ strict subset containment. Also $\subset$ $\subsetneq$

Time-constructible?

Technical condition which we won't go into

All "natural" functions $f(n) \geq n$ that you can think of are time-constructible

E.g. $f(n) = n \log n$, $f(n) = n^2$, $f(n) = 2^n$ etc

Will prove:

TIME HIERARCHY THEOREM, VANILLA VERSION

$$DTIME(n) \subsetneq DTIME(n^{1.5})$$

Proof Let D be following TM:

On input x, run universal TM U for $|x|^{1.4}$ steps to simulate execution of M_x on x

If U halts with output $b \in \{0,1\}$,
output opposite answer $1-b$

Else output 0.

~~~

How set time bound for TM? E.g.

- compute $|x|$
- then compute $|x|^{1.4}$ & store in counter
- then fill dedicated worktape with special marker symbol, until counter decreased to 0.
- Now move back to start of work tape, start simulation, and at every step move right on "timer tape"
- abort if ever see non-marker symbol on timer tape

~~~

D decides some language, namely $L_D = \{x \mid D(x) = 1\}$

D runs in time $n^{1.4}$ by construction

Hence $L_D \in DTIME(n^{1.5})$ (by same margin).

We claim $L_D \notin DTIME(n)$

For contradiction, assume $\exists M$ that on any x runs in $\leq c|x|$ steps (for some fixed c) and outputs $D(x)$.

M can be simulated (on any x) in time $O(|x| \log |x|)$ by U. ($\leq c'|x|\log|x|$ for some c')

Fix large enough N s.t. $n^{1.4}$ is larger than this if $n \geq N$.

Pick some x of length $\geq N$ s.t.
$M_x = M$ (possible by (2) above)

Then – on input x, D will simulate M on x
– M will have time to terminate and output $M(x)$
– By def of D we have $D(x) = 1 - M(x) \neq M(x)$
– But M decides L_D by assumption, so $M(x) = D(x)$

Contradiction. Hence no such M exists, QED □

There is also a time hierarchy theorem for nondeterministic computation

NONDETERMINISTIC TIME HIERARCHY THEOREM
If f, g are time-constructible functions satisfying $f(n+1) = o(g(n))$, then

$$NTIME(f(n)) \subsetneq NTIME(g(n))$$

Proof more subtle. Will skip this.

Most problems studied in NP are known either to be in P or to be NP-complete.
So can it be that every problem in NP is either in P or NP-complete?
(Results of that flavour known as DICHOTOMY THEOREMS.)

Answer: If $P = NP$, then yes (trivially).
If $P \neq NP$, then no, in very strong sense.

LADNER'S THEOREM

If $P \neq NP$, then exists strict infinite hierarchy of cplx classes between P and NP

Will prove:

LADNER'S THEOREM VANILLA VERSION

If $P \neq NP$, then exists $L \in NP \setminus P$ that is not NP-complete.

Let's break the bad news right away: This language L looks very contrived... But interesting to know it exists.

Proof Let $H : \mathbb{N} \to \mathbb{N}$ be some function.

Define SAT_H to be (CNF)SAT with all length-n formulas padded with $n^{H(n)}$ 1's

$$SAT_H = \{ \psi 0 1^{n^{H(n)}} \mid \psi \in SAT \text{ and } n = |\psi| \}$$

That is: given x, scan from back until first 0. Let ψ be everything before this 0; set $n = |\psi|$. Have string 1^k after 0.

$x \in SAT_H$ if (a) $\psi \in SAT$ and (b) $k = n^{H(n)}$.

Want to chose function $H : \mathbb{N} \to \mathbb{N}$ in some clever way so that SAT_H is too hard to be in P (assuming $P \neq NP$) but too easy to be NP-complete (because the padding gives us enough extra time).

Define $H(n)$ by following algorithm

Initialize $H(1) = H(2) = H(3) = H(4) = 1$
for $m := 5$ up to n
  $i := 0$
  repeat
    $i := i + 1$
    failed := FALSE
    for all $x \in \{0,1\}^*$ with $|x| \leq \log m$
      simulate M_i on x for $i \cdot |x|^i$ steps
      if M_i didn't terminate
        failed := TRUE
      else
        let b = output of $M_i(x)$
        divide $x = \psi\, 0\, 1^k$ and
        let $s = |\psi|$
        check that $b = 1$
          if and only if
            $\psi \in SAT$ and $k = s^{H(s)}$
        else
          failed := TRUE
    endfor
  until NOT failed or $i \geq \log\log m$
  $H(m) := i$
endfor

Checking if M_i decides SAT_H on all strings of at most logarithmic size

CLAIMS ABOUT H

(1) H is well-defined (i.e., the algorithm computes a specific function)

(2) $H(n)$ can be computed in time polynomial in n

(3) $SAT_H \in P$ iff $H(n) = O(1)$ [i.e. $\exists C$ s.t. $H(n) \le C$ $\forall n$]

(4) If $SAT_H \notin P$ then $H(n) \to \infty$ as $n \to \infty$

Defer proofs till later. Now assume $P \neq NP$.

Suppose $SAT_H \in P$. Then $H(n) \le C$.
Let M decide SAT_H in polytime.
Then here is a poly-time algo for SAT
- given ψ of size $n = |\psi|$
- compute $H(n)$
- run M on $\psi 0 1^{n^{H(n)}}$ and answer as M does

Runs in polytime if $H(n) \le C = O(1)$.
But SAT is NP-complete and $P \neq NP$ $\not\Rightarrow$ Hence $SAT_H \notin P$

Suppose SAT_H NP-complete. Then $\exists$ reduction from SAT to SAT_H computable in time $O(n^k)$ for some k.

Then for any $\delta > 0$ (say, in particular, $\delta = 1/3$) there must exist K such that formulas of size $n \ge K$ get mapped to padded formulas of size $\le n^\delta$

f maps φ of size n to
$\psi 0^{s^{H(s)}}$ for $s = |\psi|$

Total size $\leq$ running time of reduction $\leq n^k$
Suppose ψ always of size $\geq n^\delta$. Then

$\psi 0^{s^{H(s)}}$ has size $\geq n^\delta + 1 + n^{\delta H(s)}$

where $s \to \infty$ as $n \to \infty$
Then $H(s) \to \infty$ as well
But then reduction will yield string of size $\geq$
$\omega(n^k)$ — contradiction

But if so we can get poly-time alg for SAT

- If $|\varphi| \leq K$, solve by brute force
  constant time

- If $|\varphi| = n > K$ run reduction f to
  get ψ of size $\leq \sqrt[3]{n}$
  Then run f on ψ to get ψ of size $\leq (n^{1/3})^{1/3} = n^{1/9}$
  etc until the size shrinks
  to below K. Contradiction

Hence SAT_H is not NP-complete.

So $SAT_H \notin P$ and not NP-complete, QED.

Proofs of claims

Claim 1: Exercise

Claim 2: Exercise.

Claim 3 ($\Rightarrow$) Suppose $SAT_H \in P$ decided by M in time $\leq c \cdot n^c$

$\exists\, i > c$ s.t. $M = M_i$

Then for $n > 2^{2^i}$ we have $H(n) \leq i$

($\Leftarrow$) Suppose $H(n) = O(1)$. Then H takes (at least) same value i infinitely often.

Then M_i decides SAT_H ($i \cdot n^i$ and runs in poly time)

Suppose not - $\exists x$ s.t. M_i gives wrong answer on x.

Then for all $n > 2^{|x|}$ M_i would be discarded since $M_i(x)$ is wrong, and $H(n) \neq i$. contradiction.

Claim 4 Follows from ($\Leftarrow$) direction of claim 3.

NP-complete / non-complete

More interesting languages in $NP \setminus P$?

FACTORING and GRAPH ISOMORPHISM

are candidates, but we really don't know.

L3 XI

What are "diagonalizing techniques"
Relies only on

(I) Effective representation of TMs by strings

(II) Ability of a TM to simulate ~~other~~ any TM without much overhead (in time or space).

DEF (somewhat informal)

Oracle TM M has special read-write oracle tape.
To execute M, specify oracle language
$O \subseteq \{0,1\}^*$

At any time, M can write string x on oracle tape, jump to oracle state, and then in one time step get answer on oracle tape
1 if $x \in O$, 0 if $x \notin O$.

Output of M with oracle O on x
denoted $M^O(x)$.

Nondeterministic oracle machines defined analogously.

$P^O = \{$ all languages decidable in poly time DTM with oracle access to $O\}$

$NP^O = \{$ -11- NDTM -11- $\}$

Examples

(1) $UNSAT \in P^{SAT}$

Write down formula on oracle tape
Make query to SAT
Give the opposite answer as output

(2) If $O \in P$ then $P^O = P$

Oracle calls are not needed
Can compute answer by simulating machine deciding O in poly time

(3) $EXPCOM = \{\langle M, x, 1^n \rangle : M$ outputs 1 on x within 2^n steps$\}$

Then $P^{EXPCOM} = NP^{EXPCOM} = EXP$

Recall $EXP = \bigcup_{c \in \mathbb{N}} DTIME(2^{n^c})$

Suppose $L \in DTIME(2^{n^c})$ decided by M
Write down M, x, and then 1^{n^c} (i.e. n^c 1's) on oracle tape
Can be done in poly time
Query EXPCOM and answer the same
$\Rightarrow EXP \subseteq P^{EXPCOM}$
$P^O \subseteq NP^O$ for any oracle language O (why?)

Suppose $L' \in NP^{EXPCOM}$ decided by M'
running in time $O(n^k)$
At most $2^{O(n^k)}$ nondeterministic choices
which is exponential
At most that many oracle# calls – can also be computed in exponential time
Exponential × exponential = exponential, so $L' \in EXP$.

Regardless of what the oracle O is (I) and (II) holds for oracle TMs (if simulating machine is also given the oracle O.

Hence, any theorem about TMs that uses only (I) + (II) holds for oracle TMs (the theorem RELATIVIZES).

The answer to $P \stackrel{?}{=} NP$ can't be a relativizing theorem, since there are oracles to flip the answer both ways!

THEOREM (Baker, Gill, Solovay '75)
There exist oracles A and B s.t.
$P^A = NP^A$ and $P^B \neq NP^B$

Proof Set A to be EXPCOM

For any language B, let
$U_B = \{ 1^n : \exists x \text{ s.t. } |x| = n \text{ and } x \in B \}$

For any B, $U_B \in NP^B$
On input 1^n, guess x of length n, write on oracle tape, query B.

Want to build B s.t. $U_B \notin P^B$.
If so, proof finished

High-level intuition:
Any TM for U_B has to run in subexponential time.
Can only query vanishing small part of strings of length $\{0,1\}^n$ - exponentially many. Make sure any TM "queries the wrong strings."

Construction of B

M_i : Turing machine encoded by (binary expansion of) i

Construct B in stages. Stage i will make sure M_i doesn't solve U_B in time $\leq 2^n/10$.
Initially $B := \emptyset$, $i := 1$.

Stage i:

B contains finite # strings so far
Fix n s.t. no string of length $\geq n$ in B.
Run M_i on 1^n for $2^n/10$ steps.

Oracle queries y

case 1 status of y decided in previous stages — answer accordingly

case 2 status of y undecided — answer $y \notin B$

~~All run~~

Suppose M_i finishes on 1^n and answers b

Note - Have only decided status of $\leq 2^n/10$ strings in $\{0,1\}^n$
- For all of them, answered no.

If $b=1$, decide that no string in $\{0,1\}^n$ is in B
$\Rightarrow 1^n \notin U_B$, and M_i is wrong on 1^n

If $b=0$, pick some string $y \in \{0,1\}^n$ not queried and decide that $y \in B$
$\Rightarrow 1^n \in U_B$, so M_i is wrong on 1^n

Only remaining worry. What if we didn't allow M_i to finish? What if it runs in polynomial time p s.t. $p(n) \geq 2^n/10$ for this n?

The TM M_i will be repeated infinitely often for larger and larger i. Finally, will get some n' s.t. $p(n') \ll 2^{n'}/10$, and for this n' the proof will work.

SUMMING UP

- Diagonalization can be used to separate cplx classes.
- In particular, $P \neq EXP$
- If $P \neq NP$, then there is infinite hierarchy of cplx classes between P & NP
- But diagonalization not enough to settle $P \stackrel{?}{=} NP$, since any such proof works for oracle TMs and different ~~TMs~~ oracles give different answers to $P \stackrel{?}{=} NP$...

NEXT TIME

- Memory consumption as the limiting factor
- Space-bounded cplx classes