LAST TIME

- If NP has polynomial size circuits, then the polynomial hierarchy collapses [Conditional result]
- Given more time, we can solve strictly more problems [Unconditional result]
- Proof technique: Diagonalization

TODAY

- What can we say about landscape between P and NP?
- Limits of diagonalization — it cannot prove $P \neq NP$ (or $P = NP$)
- Move on to SPACE COMPLEXITY
  Measure memory usage

Aside: Because of reorganizations of lectures due to travelling we are sort of half a lecture out of sync.

So I will try to finish one lecture in one half and start new topic in next half for a few lectures...

What lies between P and NP?

If P=NP, nothing (clearly)

But what if P ≠ NP?

LADNER'S THEOREM

If P ≠ NP, then there exists a strict, infinite hierarchy of complexity classes between P and NP

Guided exercise for problem set 2:
- Prove vanilla version of this statement
- Most of details can be found in textbook
- Want you to go through the proof and make sure you understand it
- Write nice, complete exposition aimed at student finishing 420, say
- So practice also writing and presentation skills.

LADNER'S THEOREM, VANILLA VERSION

If P ≠ NP, then there exists a language $L \in NP \setminus P$ that is not NP-complete

Caveat: This language L looks quite contrived...
But interesting to know it exists.

Main idea: Padding.

Let $P: \mathbb{N} \to \mathbb{N}$ be some function such that $P(n)$ is computable in time polynomial in n.

Define SAT_P to be (CNF)SAT with all size-n formulas φ padded with $n^{P(n)}$ 1's

$$SAT_P = \{\varphi 0 1^{n^{P(n)}} \mid \varphi \in CNFSAT \text{ and } n = |\varphi|\}$$

That is: given string x, scan from back until first 0. Let φ be everything before that 0. Set $n = |\varphi|$ = length of this. Have string 1^k after 0.
$x \in SAT_P$ if (a) $\varphi \in CNFSAT$ and (b) $k = n^{P(n)}$

OBSERVATIONS

- If $P(n) \in O(1)$, then SAT_P NP-complete
- If $P(n) = \Omega(n/\log n)$, then $SAT_P \in P$

Proof: Problem set 2.

Want to choose padding function in some clever way so that SAT_P is too hard to be in P (assuming $P \neq NP$) but too easy to be NP-complete (because the padding gives extra time)

Here is our padding function

H(n)

```
if n ≤ 4
    return 1
else
    i := 0;  failed := TRUE
    while i < log log n and failed
        failed := FALSE;  i := i + 1;
        for all x ∈ {0,1}* with |x| ≤ log n
            simulate M_i on x for i · |x|^i steps
            if M_i didn't terminate
                failed := TRUE
            else
                let b := output of M_i(x)
                split x = φ 0 1^k and
                    let s := |φ|
                check that b = 1 if and only if
                    φ ∈ CNFSAT and k = s^H(s)
                else
                    failed := TRUE
        endfor
    endwhile
    return i
```

Recursive call

Checking if M_i decides SAT_H correctly 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) $H(n)$ is computed in time polynomial in n

(3) $SAT_H \in P$ if and only if $H(n) = O(1)$
[i.e., there exists a K such that $\forall n\ H(n) \leq K$]

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

Proofs: Problem set 2 (plus read Arora-Barak)

Now assume $P \neq NP$

(i) Suppose $SAT_H \in P$.
Then we can show that $CNFSAT \in P$
But CNFSAT NP-complete. Contradiction.

(ii) Suppose SAT_H NP-complete
Then we can reduce CNFSAT to SAT_H efficiently
But if so can compose reductions and compress CNFSAT instance so much that they are solvable in polynomial time.
Contradiction.

Detailed proof: Problem set 2.

Are there more interesting and natural non-NP-complete languages in NP \ P ?

Obviously, we don't know

But FACTORING and GRAPH ISOMORPHISM are candidates

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) $\overline{SAT} \in P^{SAT}$

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

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 $B := \emptyset$, $i := 1$.

has had its status wrt B decided

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 ON THE AGENDA

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

LECTURE 8½

News

- Pset 1 posted. Due Tue October 6
- Please read info about pset rules and peer evaluation process carefully
- Anything unclear? ⇒ Post a (private) question at Piazza

So far

Focus on running time as limited resource

At the end of the day, most interesting measure (?)

Today

Focus on memory usage

Second most fundamental resource (?)

Get new cplx classes and complete problems for them

Some similarities with time-bounded computation, but also some striking differences

DEF 1 A language $L \subseteq \{0,1\}^*$ is in $SPACE(s(n))$ if

$\exists$ constant c

Turing machine M

s.t. M decides L

M's heads never visit more than $c \cdot s(n)$ distinct locations for any input of length n.

(ON READ-WRITE TAPES EXCLUDING INPUT TAPE!)

$L \in NSPACE(s(n))$ if $\exists$ NDTM M deciding L

s.t. at most $c \cdot s(n)$ read-write locations are visited for any input of length n and <u>any</u> non-det choices.

Important points

- Input stored on read-only input tape
  Doesn't count towards memory
  $\Rightarrow$ Possible to do computations in <u>sublinear space</u>

- Decision problem: Need not use output tape other than for answer yes/no (0/1).
  Focus on work tape(s)

- Look at only <u>space-constructible $s(n)$</u>
  $\exists$ TM that computes $s(|x|)$ in $O(s(|x|))$ space given input x. (Technical condition that we will ignore.)

- Space bounds of interest $\geq \log n$ — want TM to be able to remember positions on input tape.

Clearly $DTIME(s(n)) \subseteq SPACE(s(n))$
Can visit at most one tape position per time step
But space can be **reused**
Use space $S(n)$ to count from 0 to $2^{s(n)} - 1$
This is (almost) all that we know.

<u>THM 2</u> $DTIME(s(n)) \subseteq SPACE(s(n)) \subseteq NSPACE(s(n))$
$\subseteq DTIME(2^{O(s(n))})$.

(Will be proven shortly.)

In fact, can do slightly better

THM 3 (Hopcroft, Paul, Valiant '77)

$$DTIME(s(n)) \subseteq SPACE(s(n)/\log s(n))$$

So space is strictly more powerful than time as resource. (Probably won't do anything close to proving Thm 3.)

One more point:

– What about termination in Def 1?

Can require it. Not necessary, really.

After $2^{O(s(n))}$ steps, worktapes has looked exactly the same, read/write heads have been exactly the same place, (ND) TM state has been exactly the same, etc at two time steps $t_1 < t_2$.

So can ignore computation during interval $[t_1 + 1, t_2]$.

If $\exists$ accepting computation, only $2^{O(s(n))}$ steps needed. Can equip any TM with "clock" that terminates after $2^{O(s(n))}$ steps. Only $O(s(n))$ space needed to count the time.

Back to proof of Thm 2 ...

DEF 4 Configuration graph of TM M

Configuration of M | consists of | at time t |

- program counter / state
- head positions
- contents of all tape positions that can possibly be visited (for same input length)

Configuration graph of M on input $x \in \{0,1\}^*$ | space $s(n)$ TM |

Directed graph $G_{M,x}$

Vertices: all possible configs C with input $= x$ and $c \cdot s(|x|)$ worktape cells

Edges: (C, C') if C' can be reached from C in one step acc to M's transition function.

Deterministic TM: out-degree 1

Nondeterministic TM: out-degree 2

Assume M has "clean-up phase" erasing all worktapes before halting $\Rightarrow$ one unique accepting config C_{accept}.

M accepts $x \iff \exists$ path in $G_{M,x}$ from C_{start} to C_{accept}

CLAIM 5

1. Every vertex in $G_{M,x}$ can be described using $K \cdot s(n)$ bits ($K = O(1)$ depending on alphabet, # tapes, # states)
2. $G_{M,x}$ has at most $2^{O(s(n))}$ vertices
3. $\exists$ $O(s(n))$-size CNF formula $\varphi_{M,x}$ s.t. $\varphi_{M,x}(C, C') = 1$ iff (C, C') edge in $G_{M,x}$

Proof

1. Sort of by description in Def 4
2. Follows from 1.
3. Use Cook-Levin-style reasoning

Formula contains lots of local consistency checks

- tape contents are correct one time step later
- jump to correct state given read bits and previous state
- et cetera

$O(s(n))$ checks

Each check involves constant # bits = variables. ☐

Proof of Thm 2

Only need to show $NSPACE(s(n)) \subseteq DTIME(2^{O(s(n))})$.

Construct $G_{M,x}$ in $2^{O(s(n))}$ time.

Do BFS to check if Caccept reachable from Cstart. ☐

DEF 6 Some cplx classes of particular interest

$$PSPACE = \bigcup_{c \in \mathbb{N}^+} SPACE(n^c)$$

$$NSPACE = \bigcup_{c \in \mathbb{N}^+} NSPACE(n^c)$$

$$L = SPACE(\log n)$$

$$NL = NSPACE(\log n)$$

Next time

Talk about PSPACE, NPSPACE, L, NL

Relate to other complexity classes, such as NP

Leads to discussion of complete problems

And more...