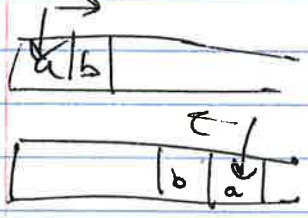
Complexity Classes.

We begin our final unit entirely on computational complexity. This lecture will simply consist of some early and motivating theorems, mostly before the development of NP-completeness.

First we need a good computational model of a "hard" or "easy" computation. There is a reason I have been beating you with Turing machines. Turing machines make an excellent model for complexity. Recall that a Turing machine ~~takes~~ performs a constant amount of work in unit time. If more work is to be done, successive steps must be taken. This is exactly what makes it an excellent model. There did historically exist functional but Turing-complete models of computation. They do not have this property [$\lambda x[xx]$]

The variant and choice of Turing machine does matter. For now, we set aside the Nondeterministic Turing machine and only consider reasonable and realizable models. Consider the language $PAL = \{ww^R \mid w \in \Sigma^*\}$, the set of even palindromes. Here is an algorithm to decide PAL on a one tape DTM. check the first and n^{th} symbol, then the $n-1^{th}$ and 2nd symbol, and so on. The limitation of this machine is that it is not random access, and to read the last symbol takes a linear number of steps. To decide ww^R, this takes $n+(1)+(n-2)+1+\cdots = O(n^2)$ by ~~guesses~~ Guass' trick.

We can give a better algorithm on a two tape DTM as follows. Copy input to second tape, reset one tape head



top both heads in opposite directions checking them against each other. These three steps each take linear time giving a $3n$ time algorithm on our two tape DTM.

Obviously any stronger model must also take atleast linear time, as to decide if $ww^R \in PAL$ must look at all the ~~digits~~ symbols for correctness. Can a one tape DTM decide PAL in $O(n)$ or even $o(n^2)$ time? Surprisingly, no. On a ~~ten~~ one tape deterministic Turing machine, PAL takes $\Omega(n^2)$ (and so $\Theta(n^2)$) steps to decide it. There are two proofs, a more classic combinatorial one, and one which uses Kolmogorov complexity! I recommend this proof as your final project.

let $TIME(f(n)) =$ class of languages decidable by a Turing machine in $f(n)$ steps. Similarly for $NTIME(f(n))$, $SPACE(f(n))$, $NSPACE(f(n))$

Define $P = \bigcup_{k=0}^{\infty} TIME(n^k)$ Why is P a good definition of "efficient"?

1) Most problems seem to have naive, trivial brute force solutions in EXP. If $\exists$ a polynomial time algorithm, then either the problem is trivial, ridiculous, or we have some deeper intuition about what the problem actually is

2) polynomials are closed under operations which continue our intuition of "efficient" is also closed under. if f, g are polynomials, then fg, $f+g$, $f(g)$ are also polynomials.

3) Although there exist languages with $O(n^{100})$ algorithms with $\Omega(n^{99})$, we don't have any real examples of this. LLL gave an $O(n^8)$ algorithm to ~~find~~ find a short orthonormal basis of a lattice. Its polytime but infeasible. Then the engineers got a hold of the problem and made it practically efficient on all inputs I could test.

4) All Turing machine variants appear to simulate each other with at most polynomial overhead. A word-RAM machine does in T steps takes a one tape DTM T^4 steps. a definition of P is equivalent for all these models, what we call the extended Church-Turing thesis.

let $NP = \bigcup_{k=0}^{\infty} NTIME(n^k)$. You pray know NP = languages verifiable deterministically in polynomial time. We prove these definitions are equivalent. let $NP_1 = \bigcup NTIME(n^k)$ ~~$A \in NP_2 \Leftrightarrow$~~ $A \in NP_2$ $\Leftrightarrow$ $A = \{ w \mid \exists V \text{ accepts } \langle w, c \rangle \}$.

let $A \in NP_2$, then $\exists$ a polytime verifier V, which runs in time $O(n^k)$. we build an NTM to decide A as follows.

N on input w:
  guess c max length n^k
  run $V(w, c)$
  accept iff V accepts.

clearly, N runs in polynomial time so $NP_2 \subseteq NP_1$

let $A \in NP_1$, Then $\exists$ a polynomial time NTM to decide A. we show A is polytime verifiable. our witness c is just our nondeterministic choices.

V on input $\langle w, c \rangle$
  simulate N deterministically on w
  if faced with a nondeterministic choice, use the next bit of c.
  if this branch of N's computation accepts, we accept.

so V is a deterministic polytime verifier, $NP_1 \subseteq NP_2$.

we prove $P \subseteq NP$ in two ways.

1 first note every deterministic TM is also a nondeterministic one. so $P \subseteq NP$.

2 If $A \in P$, we prove A is also verifiable in polynomial time.

V on input $\langle w, c \rangle$ (ignore c.)
  if $w \in A$
    accept
  else reject.

since $A \in P$, $\exists$ polytime to decide A

---

let $PSPACE = \bigcup_{k=0}^{\infty} SPACE(n^k)$. convince yourself $P \subseteq PSPACE$.

let $L = logspace = SPACE(\log(n))$.

$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq NPSPACE \subseteq EXP \subseteq NEXP \subseteq EXPSPACE \subseteq NEXPSPACE$.

$L \subseteq NL$ obvs

$NL \subseteq P$ since $\exists$ a polytime algorithm for an NL-complete problem.

$P \subseteq NP$ as proved.

$NP \subseteq PSPACE$ since $SAT \in SPACE(n)$

  recall $\forall L \in NP$, $L \leq_p SAT$

actually $PSPACE = NPSPACE$

  we will prove this by Savitch's theorem.

consider the following two chains.

$L \subseteq P \subseteq NP \subseteq PSPACE$. Since $L \subsetneq PSPACE$, one of the containments in this chain is strict. Note $P \stackrel{?}{=} PSPACE$ is also an open problem since $P = PSPACE \Rightarrow P = NP$. Similarly if $L = P$.

$P \subseteq NP \subseteq EXP$. We know $P \subsetneq EXP$, so again one is strict in this containment chain. proving $NP = EXP \Rightarrow P \neq NP$.

btw $EQ_{REX} = \{\langle R_1, R_2 \rangle$ are regex with exponentiation$\}$ is $\notin PSPACE$ complete.

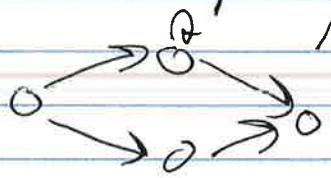
we seriously hope, but can't prove $P \neq NP$. Since if $P \neq NP$ we get: random generators are indistinguishable from pseudo random ones are many functions exist

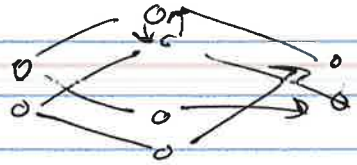
ALL of cryptography!!

∃OWFs are quite natural in nature. Go ahead, unbake a cake. Unkill your gold fish.

$L \in SPACE(f(n))$ $\forall c < f(n)$ $L \in SPACE(f(n) - c)$.

This is why we use big O. Why we write $O(1)$ instead of $O(c)$. proof by induction if M uses $S(n)$ space, we give M' to use $S(n) - 1$ space. Use the left most cell not on the tape to hold states!





Similarly if L is decidable in $f(n)$, it is decidable in $f(n)/c$ $\forall c$. Why? make Σ super big.

SPACE HIERARCHY THEOREM

more space = more power. if f is space constructable,

$SPACE(o(f(n))) \subsetneq SPACE(O(f(n)))$

we proceed by diagonalization.

D on input w:

$n = |w|$

mark of $f(n)$ space. if more is used, reject

if $w \neq \langle M \rangle 1^*$ for some M, reject

run M on w. If it takes more than $2^{f(n)}$ steps reject

if M accepts

reject

if M rejects

accept

Final reduction

we give an algorithm to run in $O(f(n))$ space which can't run in $o(f(n))$ space. if M runs in $o(f(n))$ space, we ensure D behaves differently than M on input $\langle M \rangle$. if $\langle M \rangle$ uses more space, we don't care.

because of asymptotics, we may not "kick in" yet so we give it a few chances. instead of running M on $\langle M \rangle$, we run M on $\langle M \rangle 10^*$ Eventually D will differ on M on some $\langle M \rangle 10^k$ k big enough for the difference to kick in.

final detail, if any machine uses $o(f(n))$ space, it uses $\geq 2^{O(f(n))}$ time. This is to prevent machines which infinite loop in finite space.

D on input w_i Let M_1, M_2 be machines which run in $o(f(n))$ space.
if M_i rejects w_i in $O(f(n))$ space
accept
else reject.

note D uses $O(f(n))$ space so $L(D) \in SPACE(O(f(n)))$.

with a bit more work, $\forall \epsilon_1, \epsilon_2 \in \mathbb{R}$ with $0 \leq \epsilon_1 < \epsilon_2$

$$SPACE(n^{\epsilon_1}) \subsetneq SPACE(n^{\epsilon_2}) \quad \forall \epsilon$$

$L \subsetneq PSPACE$. $PSPACE \subsetneq EXPSPACE$.