Complexity

Note: Strange structure due to finals topics. Will improve later.

D:A *deterministic Turing machine with k tapes* M_k is a tuple $(\Sigma, Q, \delta, q_0, F)$. δ is the usual transition function, Q the set of all states, Σ language, F accepting states, q_0 starting state.

D:A *nondeterministic Turing machine* is a machine such that δ produces a set of future states instead of just one state.

Notation: In this text, L will always be a language, and Φ, Ψ functions as well as f, g. M, N are DTM unless stated otherwise, M_k with k tapes.

TODO: Blum complexity measure.

TODO: Recursive, computable notions.

Speedups and compressions

T(Alphabet compression.**)**:If f is a time-constructible function $f: \mathbb{N} \to \mathbb{N}$ with a TM with alphabet Γ, then f is computable in time $4 \log |\Gamma| T(n)$ with a 4-character alphabet TM.

P:Use alphabet $0, 1, \to, |$. Encode ternary and higher objects into binary.

T(Linear space compression**)**: L is accepted by M_k with $S(n)$, then $\forall r \in \mathbb{N} \exists N_k$ such that N_k accepts L with $S_N(n) = \lceil S(n)/r \rceil$.

P:Since we do not care about time, we can encode a r-tuple of characters into one character. We also need special characters so that we know on which character is our tape. Moving may then be replaced with changing a aḅc to abç.

R:$\forall r \in \mathbb{N}: \text{DSPACE}(S(n)) = DSPACE(S(n)/r)$.

R:$\forall r \in \mathbb{N}: \text{NSPACE}(S(n)) = NSPACE(S(n)/r)$.

T(Tape reduction**)**:If we have a M_k accepting L, then there is M_1 accepting L with equal $S(n)$.

P:Again, we do not care about time, so we simply store all k characters on one tape and and have special characters for storing state of the emulated heads.

T(Tape time reduction**)**:If we have a M_k accepting L in time $T(n)$, there is a M_1 accepting it in time $5kT(n)^2$.

P:Input is at the beginning of the 1 tape. Encode the other tapes after it. Create a larger alphabet with all characters having an extra flag whether the head is on them or not. Sweep the tape twice, first time load the marked characters to your "registers", the second time do the transition.

L:Let $k \geq 2, r > 0$. If we have a M_k accepting L with time complexity $T(n)$, there is a N that accepts L in time $n + \lceil n/r \rceil + 6\lceil t/r \rceil]$.

P:Assume we can write on the input tape. First, encode the input on the first work tape and make it r times denser. Then use input tape as work tape. In every step, read one character to the left and one character to the right of the current head position. This can be done in 4 moves. In r steps of the original M_k, only r characters can be changed from the current head position. The M_k could only modify two of the three positions, and we can edit them using 2 steps. Of course, this increases the complexity of the transition function.

T(Linear Speedup**)**:Let $k \geq 2, \varepsilon > 0$. Then if M_k accepts L in time $T(n) \geq \omega(n)$, there exists N s.t. it accepts L in time $\varepsilon T(n)$.

P:Simply choose r in the Lemma based on ε, usually things like $21/\varepsilon$.

Constructible functions

D:A function $\Phi: \mathbb{N} \to \mathbb{N}$ is recursive if there is a TM that can compute $1^{f(n)}$ with input of size 1^n.

D:A function Φ is T/S-enumerable in T/S $O(f) \equiv$ it is recursive and $\exists$ TM such that the TM enumerates it within $cf(n)$ steps for input of size n.

D:A function Φ is T/S-constructible (time-constructible or space-constructible) $\equiv \exists$ a TM such that it halts after exactly $\Phi(n)$ steps for input of size n.

L:Let $f_1 + f_2$ and f_2 be a T-constructible functions and $\exists \varepsilon > 0 \exists n_0 \forall n \geq n_0 f_1(n) \geq \varepsilon f_2(n) + (1+\varepsilon)n$. Then f_1 is time constructible also.

P:Apply speedup to find a machine pacing exactly at f_1.

T(Time Equivalence**)**:Let $f: \mathbb{N} \to \mathbb{N}$ be a function s.t. $\exists \varepsilon > 0 \exists n_0 \forall n \geq n_0 : f(n) \geq (1+\varepsilon)n$. Then f time-enumerable in time $O(f) \Leftrightarrow f$ time constructible.

P:Backwards direction immediate, forwards direction by previous lemma.

T(Space Equivalence**)**:Space-enumerability and constructibility coincide.

P:We can use space compression to get it exactly, no lemmas required.

T:Every time constructible f is space constructible.

Complexity classes

TODO:DSPACE, DTIME

T:Φ recursive function $\to$ $\exists$ recursive language $L \notin DTIME/DSPACE(\Phi(n))$. **P:** Diagonal argument. Order machines, inputs $L \equiv \{x_i | M_i$ does not accept x_i in time $\Phi(|x_i|)\}$.

T:

- $\Phi, \Psi: \mathbb{N} \to \mathbb{N}$ and $\Psi \in \Omega(\Phi)$ with Ψ being space-constructible $\to \exists L \in DSPACE(\Psi) \setminus DSPACE(\Phi)$.
- $\Phi, \Psi : \mathbb{N} \to \mathbb{N}$ and $\Psi \in \Omega(\Phi \log \Phi)$ with Ψ being time-constructible $\to \exists L \in DTIME(\Psi) \setminus DTIME(\Phi)$.

P: Part 1. Create a machine M which allocates Ψ memory and accepts only if any single-tape machine M_w refuses w in space Ψ. If this language was in DSPACE(Φ), there would be a machine N recognizing this. This machine N can be assumed to be single-tape and always stops. We can emulate this machine, and so we have a contradiction.

Part 2. Similarly, only we use $\Phi \log \Phi$ for the tape reduction theorem. We set time to be so high that the would-be machine can be emulated safely.

T(Relationship Theorem**)**:

- DTIME $\subseteq$ NTIME.
- DSPACE $\subseteq$ NSPACE.
- DTIME $\subseteq$ DSPACE.
- NTIME $\subseteq$ DSPACE for s-c. functions.
- $L \in \text{DSPACE}(\Phi(n)) \wedge \Phi(n) \geq \log n \to L \in \text{DTIME}(c_L^{\Phi(n)})$, with $c_L \equiv c(L)$.
- $L \in \text{NSPACE}(\Phi(n)) \wedge \Phi(n) \geq \log n \to L \in \text{DTIME}(c_L^{\Phi(n)})$, with $c_L \equiv c(L)$.
- $L \in \text{NTIME}(\Phi(n)) \to L \in \text{DTIME}(c_L^{\Phi(n)})$ with $c_L \equiv c(L)$.

Savitch Theorem

T(Savitch**)**:If $\Phi(n) \geq \log n$ and Φ is space-constructible $\to NSPACE(\Phi(n)) \subseteq DSPACE(\Phi^2(n))$.

P:USTCON is an NL-complete problem. (Basically, any problem in NL can be solved through graph search of the state space.) We can solve USTCON by recursion – basically, for every middle vertex v, we check whether there is a $k/2$ sv-path and $k/2$ vt path. Recursion takes $O(\log n)$ steps and we remember $O(\log n)$ at every step – $O(\log^2 n)$ units of memory in total.

R:$PSPACE = NPSPACE$.

NP-complete problems

T(Cook-Levin**)**:There exists a NPc problem.

P:Proved for $\csc TILE$. Simulate operations on the Turing machine tape by tiling.

Standard transformations:

$\csc TILE \propto \csc SAT$: $x_{ijk} = 1 \equiv$ position (i, j) contains a tile of type k. Add clauses that ensure validity of transformations of tiles.

$\csc SAT \propto \csc 3 - SAT$: Unwrap the long clauses into smaller ones using substitution.

$\csc 3 - SAT \propto \csc 3 - COLOR$: Set a triangle to define T/F colours. Connect all literal vertices to the third colour. Use a gadget so that no three variables in the clause get F.

$\csc SAT \propto \csc CLIQUE$: Set vertices to literals so that every clique is an internally consistent assignment.

PSPACE

D:PSPACE $\equiv$ a class of problems solvable with polynomial space.

D:$\csc TQBF$ $\equiv$ language on a universe of fully quantified Boolean formulas, containing true ones.

T:$\csc TQBF$ is a PSPACE-complete problem.

P:$\csc TQBF \in$ PSPACE: If it has n variables and formula is of size m, evaluating it recursively uses $poly(n+m)$ memory. In fact, you can do it in $O(n+m)$ space since you restrict yourself on the same formula.

$\csc TQBF$ is complete: Transform every problem searching on a Turing machine to a $\csc TQBF$ in a similar way like in the Savitch theorem. Specifically, there exists a SAT formula that can decide adjacency, use it inductively to declare formulas that calculate 2^{i-1} adjacency.

Polynomial Hierarchy

D:Σ_2^p is a class $x \in L \equiv \exists u \in 0, 1^{|q(x)|} \forall v \in 0, 1^{|q(x)|} M(x, u, v) = 1$.

Contains NP and coNP. EXACT INDSET example.

D:Π_2^p complement of Σ_2^p. Contains EXACT INDSET also.

D:Σ_i^p and Π_i^p defined analogously.

O:If $P = NP$ then $PH = P$.

P:Induction, simply use $P = NP$ to do away with the extra $\exists$.

T:If there is a PH-complete language, then PH collapses.

P:(Sketch.) If there is such a language, it is a member of some class, and then the higher classes collapse.

T:$\Sigma_i - SAT$ is a Σ_i^p-complete problem.

Pseudopolynomial alg., Strong NP-completeness

D: An algorithm is *strongly NP-complete* $\equiv$ it is NP-complete even if all its input numerical variables are bounded by the size of the input.

O:Bin packing is strongly NP-complete.

O:Subset sum, backpack problem and its related problems are not strongly NP complete, because they employ pseudopolynomial algorithms.

#P, #P-completeness

D:$\#P \equiv$ a class of enumerating problems (i.e. functions) where the associated decision problems are polynomial.

D:A function from $\{0,1\}*$ to $\{0,1\}*$ is in $FP \equiv$ it can be computed by a TM in polynomial time.

Q:FP = #P is an analog of P = NP.

D:A function g is in FP^f if it can be computed by a polynomial-time TM with oracle access to f.

D:A function f is #P-complete if it is in $\#P$ and every $g \in \#P$ is in FP^f.

T:If #CYCLE has a polynomial-time solution then $P = NP$.

T:#SAT is #P-complete.

Some #P problems can be approximated easily, others not so much. Even easy problem like #CYCLE or #PM are hard. #PM is #P-complete.