CS 120: Intro to Algorithms and their Limitations

Lecture : — Tuesday November 7, 2023

Pset Due: November 15, 2023 **Denny Cao**

---

§1 Church-Turing Thesis

Claim 1.1 (Extended Church-Turing Thesis) — Every physically realizable *deterministic sequential* model of computation is simulable by Word-RAM *with at most polynomial slowdown.*

§2 Problem-Independent Size Measure

- Measure input size in bits to ecode

Example 2.1

Graph w/$|V| = n, |E| = m$

- $(1 + m) \log n = N$

Example 2.2

3-SAT formula: n variables and m clauses.

- For each m clause, need what each of the variables are and if they are negated. $m \cdot 3 \cdot (1 + \log n)$, must specify each of the iterals, for each iteral, need 1 bit for if negated, and a variable or their negation.

Example 2.3

Array of size n from a universe $[U]$.

- $n \log U = N$
- Thus, `RadixSort` is time $O(N + 2^N)$ (find explanation for why 2^N, but it had something to do with how we can find the upperbound of $\log N$ or something)

Lemma 2.4 (Reductions)

If $\Pi \leq_{T,f} \Gamma$, then:

- $\exists$ alg solving Γ in time $g(n) \to \exists$ alg solving Π in time $O(g(f(n) + T(n)))$.
- Contrapositive: $\nexists$ alg solving Π in time $O(g(f(n)) + T(n)) \to \nexists$ alg solving Γ in time $O(g(n))$

- If only polynomial runtimes, easier lemma

Lemma 2.5 (Simpler Lemma w/Polynomial Time Complexity)
If $\Pi \leq \Gamma$, then:

- $\exists$ alg solving Γ in time something polynomial $n^{O(1)} \rightarrow \exists$ alg solving Π in polynomial time $n^{O(1)}$.
- Contrapositive: $\nexists$ alg solving Π in $n^{O(1)} \rightarrow \nexists$ alg solving Γ in $n^{O(1)}$.

Proof. Reduction R from Π to Γ which runs in time $O(n^c)$ for some c. The reduction gives an algorithm by oracle replacement. Suppose $\Gamma \in \text{TIME}(n^d)$. We replace oracle calls with runs of $O(n^d)$, giving an algorithm that solves Π with runtime:

- Reduction: $O(n^c)$
- Number of oracle calls: $O(n^c)$. We cannot now just substitute n^c for n for the oracle replacement, as the size may be bigger
- Size of memory starts at $\leq n$, grows at ≤ 1 per step, thus `mem_size` $\leq n^c \rightarrow$ all oracle calls are of size $\leq n^c$.
- Now, we can find upperbound of each oracle call runtime: maximum size of n: $O((n^c)^d)$, thus total runtime of oracle calls is $O(n^{cd+c})$

□

Remark 2.6. We made the proof easier; there is something to fill in for when we call MALLOC, as MALLOC does not just grow memory by 1—could be an entire word for example.

Remark. Can 3-coloring be solved in polynomial time?

Theorem 2.7
The problem `HALT-IN-2`^{n}, whose input is a Word-RAM program P, $n \in \mathbb{N}$ and output `True` $\iff$ P runs in time $\leq 2^n$ on all inputs of size n, then the **problem cannot be solved in polynomial time**.

Proof. CS 121 :D □

- Running on all inputs takes time 2^n, exponential time.
- If `HALT-IN-2`^{n} is Π, then a reduction Γ cannot be solved in polynomial time
- Must come up with a reduction

§3 Complexity Classes

Definition 3.1 (Complexity Classes)**.** Sets of problems defined by difficulty of solving.

- We define $\text{Time}_{\text{Search}}(f(n))$ = the set of Problems s.t. $\exists$ Word-RAM model solving Π in time $O(f(n))$.

Definition 3.2 (Polynomial Search)**.** $P_{\text{Search}} = \bigcup_{c\in\mathbb{N}} \text{TIME}_{\text{Search}}(n^c)$.

Definition 3.3 (Exponential Search)**.** $\text{EXP}_{\text{Search}} = \bigcup_{c\in\mathbb{N}} \text{TIME}_{\text{Search}}(2^{n^c})$.

> **Remark 3.4.** Any problem in P_{Search} is in $\text{EXP}_{\text{Search}}$, as they would be bounded by it.

Definition 3.5 (Decision Problems)**.** $P = \bigcup_c \text{Time}(n^c)$ and $\text{EXP} = \bigcup_c \text{Time}(2^{n^c})$

- Uses bit length n
- Comparison between complexity classes using Theorem 2.7: EXP contains P, but wonder if EXP = P. We now know that $\texttt{HALT-IN-}2^n \in \text{EXP} \wedge \notin P$, and thus $P \subsetneq E \times P$.