Assignment-Winter Camp

Computational Complexity

Due: **31st December(Tentative) EOD**

Instructions

- **You will not be RATIFIED without submitting the assignment**.
- We do not expect solutions to be completely formal, but they are to be stated in a way wherein they seem to provide logical conclusions or arguments.
- You may consult Sipser and other standard references, but you must not copy the exact solutions.
- You are allowed to use AI to understand the solutions to specific problems, but the codes or solutions that you submit must not be AI-generated.
- You are allowed to discuss questions in the group, but you should not copy the solutions of each other.
- Failure to comply with the instructions might result in deratification
- The solutions for the assignment(both theoretical and project) must be submitted by creating a repository on github.
- The solutions for the theoretical questions should be submitted either in **handwritten** form, or in **LaTeX** (which is recommended but not compulsory).
- You need to choose **ONLY ONE** project out of these 2 for final submission. You can use Python\C\C++ as your programming language, and submit your programming assignment by uploading it in the repository itself.
- Your repository name should be named as **"Computational_Complexity_Roll-No_Name"**

Resources

Read through these resources to cover topics which we were unable to cover in the classes, and also for further and better understanding of the topic.

Read through these resources to cover topics which we were unable to cover in the classes, and also for further and better understanding of the topic.

- Distinguishability Myhill Nerode Theorem.
- Examples for constructing DFAs and NFAs.
- What are graphs? - Youtube.
- Graphs implementation - Youtube.
- Graphs implementation - Youtube.

Section I: Automata Theory & Formal Languages

Problem 1. The Hierarchy of Power Prove that the language $L = \{ww \mid w \in \{0,1\}^*\}$ is **not** a Context-Free Language (CFL).
Hint: Use the Pumping Lemma for Context-Free Languages. Be careful with your choice of string s to ensure the adversary cannot pump within the boundary of the two w's.
Since this was not taught in class, you can read the CFL and pumping lemma parts of sipser to solve this question.

Problem 2. The Cost of Determinism (State Explosion) Consider the language $L_k = \{w \in \{0,1\}^* \mid \text{the } k\text{-th symbol from the end of } w \text{ is } 1\}$.

1. Construct a formal NFA N_k with exactly $k+1$ states that recognizes L_k. Provide the state diagram or formal transition function.
2. Prove that any DFA recognizing L_k requires at least 2^k states.
   Hint: Use a "fooling set" argument. Show that for any two distinct strings $x, y \in \{0,1\}^k$, there exists a suffix z that distinguishes them (i.e., $xz \in L_k \iff yz \notin L_k$), implying they must lead to different states in a deterministic machine.
3. Based on the above, conclude whether the Subset Construction algorithm for converting NFAs to DFAs is optimal in the worst case.

Additionally, answer these for general automata:

- Why can multiple NFAs correspond to the same DFA after determinization?
- Can two distinct DFAs recognize the same language even if their transition structures differ? Prove your claim.

Problem 3. The Perfect Shuffle Let A and B be languages over Σ. Define the *perfect shuffle* of A and B as:

$$\{w_1v_1w_2v_2 \dots w_kv_k \mid w = w_1 \dots w_k \in A, v = v_1 \dots v_k \in B, |w_i| = |v_i| = 1\}$$

Prove that the class of **Regular Languages** is closed under the perfect shuffle operation.
Hint: Construct a formal DFA $(Q, \Sigma, \delta, q_0, F)$ that tracks the state of machines for both A and B simultaneously.

Problem 4. Application of Myhill–Nerode Theorem Let L be the language $L = \{0^n1^n \mid n \geq 0\}$. Use the Myhill-Nerode Theorem to prove that L is **not** regular.
Specifically:

1. Define an infinite set of strings $S \subseteq \{0,1\}^*$.
2. Show that for any two distinct strings $x, y \in S$, the strings are *distinguishable* with respect to L. Two strings x and y are distinguishable if there exists a "separator" string z such that $xz \in L \iff yz \notin L$.
3. Conclude why this implies L cannot be recognized by any finite automaton.

Section II: Computability & Undecidability

Problem 5. Regularity is Undecidable Prove that the following language is undecidable:

$$REGULAR_{TM} = \{\langle \mathcal{M} \rangle \mid L(\mathcal{M}) \text{ is a regular language}\}$$

Hint: Use a mapping reduction from A_{TM}. Construct a machine $\mathcal{M}'$ that accepts a regular language (like Σ^*) if $\mathcal{M}$ accepts w, and a non-regular language (like 0^n1^n) otherwise.

Problem 6. Rice's Theorem Let P be a property of Turing-recognizable languages. We say P is a *non-trivial property* if there exists a TM $\mathcal{M}_1$ such that $L(\mathcal{M}_1)$ has property P, and a TM $\mathcal{M}_2$ such that $L(\mathcal{M}_2)$ does not.
Prove that *For any non-trivial property P, the language $L_P = \{\langle \mathcal{M} \rangle \mid L(\mathcal{M})$ has property $P\}$ is undecidable.*

Problem 7. Linear Bounded Automata A Linear Bounded Automaton (LBA) is a restricted TM where the tape head is not permitted to move off the portion of the tape containing the input.
Prove that $A_{LBA} = \{\langle M, w \rangle \mid M \text{ is an LBA that accepts } w\}$ is **decidable**.
Hint: Analyze the total number of possible configurations for an LBA with input size n.

Section III: P, NP, and Reductions

Problem 8. 3-Coloring Verification and Reductions Answer the following:

1. Give a formal polynomial-time verifier for the language 3-COLOR. Clearly describe the certificate and the verification procedure.
2. Prove that 3SAT reduces to 3-COLOR.

Problem 9. 3SAT to CLIQUE Provide a complete, rigorous proof that 3SAT $\leq_P$ CLIQUE.

1. Describe the polynomial-time reduction function $f(\phi) \to \langle G, k \rangle$.
2. Prove the forward direction: If ϕ is satisfiable, G has a k-clique.
3. Prove the backward direction: If G has a k-clique, ϕ is satisfiable.

Section IV: Advanced Complexity (OPTIONAL)

Problem 10. Space Complexity Prove that $TQBF$ (True Quantified Boolean Formulas) is **PSPACE-Complete**.
*Hint: You may assume $TQBF \in$ **PSPACE**. Focus your proof on the PSPACE-hardness, showing how a computation tableau of a polynomial-space machine can be encoded into a Quantified Boolean Formula.*

Problem 11. BPP and Error Reduction The class **BPP** consists of languages decided by probabilistic Turing machines with an error probability $\leq 1/3$.
Prove the **Amplification Lemma**: For any language $L \in$ **BPP** and any polynomial $k(n)$, there exists a probabilistic algorithm for L with error probability bounded by $2^{-k(n)}$.

Problem 12. Interactive Proofs Describe the Interactive Proof protocol for **Graph Non-Isomorphism (GNI)**.

1. Define the role of the Prover (P) and the Verifier (V).
2. Show that if $G_1 \ncong G_2$, the Verifier accepts with probability 1.
3. Show that if $G_1 \cong G_2$, the Verifier accepts with probability $\leq 1/2$.

Section V: Final Projects

Problem 13. Project A: Implementation and Analysis of Multi-Tape Turing Machines
Objective: Implement a simulator for Two-Tape Turing Machines that demonstrates how additional computational resources (an extra tape) can make certain problems more efficient to solve. Through this implementation, you will bridge theoretical automata concepts with practical programming. Following are the requirements:

- **Base Implementation:** A Single-Tape Turing Machine simulator.
- **Extension of the Single-Tape Turing Machine:** A Two-Tape Turing Machine simulator.
- **Examples:** It is also desirable that you include example problems for both of the above simulators(at least one for each), illustrating practical applications of Turing Machines.
- **Compare the efficiency of the two automata:** A code implementation that clearly demonstrates the efficiency differences between the two.
- **Neat And Clean Code:** Clean and clear code(s), which must be well-commented along with all the implementation. Also, the codes must be simple in nature in terms of demonstration(try to use basic commands/libraries/functions for better understanding)

Problem 14. Project B. Reductions: Solving Logic with Graphs *Complexity theory relies on "Reductions"—translating Problem A into Problem B to prove hardness. You will implement a Karp Reduction.*

- **The Reduction:** Implement a function `reduce_sat_to_vc(cnf_formula)` that transforms a 3-SAT instance into a Vertex Cover instance.
  - Construct the "Variable Gadgets" and "Clause Gadgets" as described in standard complexity literature.
  - Output a graph G and an integer k such that G has a vertex cover of size k **if and only if** the formula is satisfiable.
- **Verification:** Use a Brute Force Vertex Cover solver (implemented recursively) to solve the generated graphs. Verify that the answer matches the boolean satisfiability of the original formula.

Submission format:

- The solutions for the theoretical questions should be submitted either in handwritten form, or in LaTeX(which is recommended but not compulsory).
- You need to choose **ONLY ONE** project out of these 2 for final submission, you can use Python, C, or $C++$ as your programming language, and submit your programming assignment by making a repo in github following the instructions given in the Whatsapp group.