December 1, 2025

Problem Set 5

Exercise 1. Groups and Solvability. Groups are an algebraic structure which can be thought of as defining some sort of symmetry. They occur frequently in many areas of CS theory and mathematics. Particularly, they are crucial to the proof of Barrington's theorem. (Which we will be having a talk on this week!)

Definition. *A group G is a set together with operation $* : G \times G \to G$, such that the following holds:*

- *Operation $*$ is associative. That is, $(g_1 * g_2) * g_3 = g_1 * (g_2 * g_3)$ for all $g_1, g_2, g_3 \in G$.*
- *There exists identity $e \in G$ such that $g * e = e * g = g$ for all $g \in G$.*
- *For any element $g \in G$, there exists inverse g^{-1} such that $g * g^{-1} = g^{-1} * g = e$.*

Throughout this exercise, you may always assume G is a group.

Typically, if the group operation is clear via the context, we do not manually specify it. For instance, if we ever refer to the group $\mathbb{R}$, it will be implied we are taking the group operation to be addition (multiplication wouldn't work since 0 has no multiplicative inverse). It is also common to write the group operation however is convenient; for example, for $x, y \in G$, we can write $x * y$ as xy.

1. ($\star$) Prove that $\mathbb{Q}$ (the set of rational numbers) forms a group under addition. Does it also form a group under multiplication? (Notice that using $*$ is uncomfortable to describe addition; feel free to use $+$.)
2. ($\star$) Consider $M_n(\mathbb{R})$, the set of $n \times n$ matrices with real entries, where we take matrix multiplication to be our operation. Is this a group?
3. ($\star\star$) A group is *abelian* if for all $x, y \in G$, we have $xy = yx$. Give an example of an abelian and non-abelian group.

Definition. *$H \subseteq G$ is a subgroup of G if*

- *For all $h_1, h_2 \in H$, we get $h_1 * h_2 \in H$. (We use the operation defined for group G.)*
- *Identity $e \in G$ is contained in H.*
- *If $h \in H$, then $h^{-1} \in H$.*

We notate this as $H \leq G$.

Observe that a subgroup of a group is also a group.

4. ($\star$) Prove that every non-trivial group has at least two subgroups. (The trivial group is a group with one element.)

5. ($\star\star$) Show that $\mathbb{Z}$ (the set of integers) is a subgroup of $\mathbb{Q}$.

Definition. *A group G is said to be generated by set $S \subseteq G$ if every element of G can be expressed as a combination of elements in S and inverses of elements in S.*

6. ($\star\star$) Group G is *cyclic* it is generated by some set S such that $|S| = 1$. Prove that $2\mathbb{Z}$ is cyclic. Give an example of a group which is not cyclic.

7. ($\star\star$) Prove $\mathbb{R}$ is not a finitely generated group (there exists no finite set generating $\mathbb{R}$). Hint: $\mathbb{R}$ is not countable.

Definition. *The commutator of $x, y \in G$ is $[x, y] = x^{-1}y^{-1}xy$. The commutator subgroup $[G, G] \leq G$ is the subgroup of G generated by $\{[x, y] \mid x, y \in G\}$.*

The size of the commutator subgroup can be thought of measuring how "non-abelian" a group is in some sense.

8. ($\star$) What is the commutator subgroup of an abelian group?

Definition. *Let G' be the commutator subgroup of G. Group G is solvable if $G \geq G' \geq G'' \ldots$ eventually reaches the trivial group (the group containing only one element).*

9. ($\star$) Prove that every abelian group is solvable.

10. ($\star\star\star\star$) Prove that every finite group with an odd number of elements is solvable.

Exercise 2. Boolean Circuits. Boolean circuits are another model of computation (different from TMs). They are directed acyclic graphs with only one sink node. Each node represents a logic gate (AND, OR, NOT, etc.) or an input. The sink node will give the output to the computation. The inputs are in boolean and each the computation essentially follows the topological ordering of the DAG. Each node computes the boolean function it is assigned, terminating at the sink node for the output value.

1. ($\star$) The set of logic gates "allowed" on a circuit is called the basis. Formally, this is the set of boolean functions that the nodes on the DAG get assigned. Show that {AND, NOT} is a complete basis: that this is equivalent to {AND, OR, NOT}.

2. ($\star$) Construct a boolean circuit that outputs 1 if and only if its input as two 2-bit binary numbers are equal.

Note that in part 2. the circuit can only compute 4-bit long inputs. This is different from TMs as one TM can compute the language on all sizes of inputs: TMs are uniform models of computation. On the other hand, circuits are non-uniform models. Therefore, to recognize a language $L \subseteq \Sigma^*$, there needs to be a circuit for each input size, which is called a circuit family C. Formally, $C = \{C_n : n \in \mathbb{N}\}$, where each C_n is the circuit within the family that has n inputs.

3. ($\star\star$) Construct the circuit family that on input $\langle x, y\rangle$ accepts (outputs 1) if and only if $x = y$. (Here you can interpret the input as the concatenated xy.)

4. ($\star\star$) Construct the circuit family that on input $\langle x, y, z\rangle$ accepts if and only if $x + y = z$.

The depth of the circuit is the size of the longest path from an input to the output. Since circuits are DAGs, each level of the DAG can be computed in parallel. So the depth essentially becomes how much time the computation takes (if there is no limit to parallelism). As a result, shallow depth circuits are of particular importance in complexity theory.
Another important resource is size, which is just the number of total nodes in the circuit (in relation to the input size).
The maximum number of inputs the gates are allowed to have is called fan-in.

5. ($\star\star\star$) Construct the circuit family for the language in part 4. but using circuits with constant depth and polynomial size. The gates have unbounded fan-in (so there is no restriction on their size).

The construction in part 5. leads to a more general classes of problems: Problems with poly-size constant depth circuits with unbounded fan-in. This class is called AC^0. In general poly-size $O(\log^i n)$ depth circuits with unbounded fan-in is called AC^i (for Alternating Circuits). For problems with circuits with the same constraints but with bounded fan-in are called NC^i (for Nick's Class in honor of Nick Pippenger by Steve for first describing the class).

6. ($\star\star\star$) Show that any unbounded fan-in boolean gate can be made with a poly-size log-depth bounded fan-in boolean circuit. Conclude that $\forall i \geq 0 : AC^i \subseteq NC^{i+1}$. (Note that only the $AC^0 \subsetneq NC^1$ is known to be strict.)

Exercise 3. Branching Programs. Branching Programs are also another model of computation. They are directed acyclic graphs with one source and two sinks. Each node has out-degree 2 (assuming a binary alphabet) and they are each labeled by a position in the input (Note that the edges can both go into the same node). So on input x, a node labeled x_i means that if the i^{th} bit of the input is 1, then follow the left branch, and if it is 0, follow the right branch. This is done starting from the single source node and following it down to the sinks. One sink is the accepting state and the other is the rejecting state. So reaching a sink means that the branching program either accepted or rejected that input.
Also note that similar to boolean circuits, these programs are also non-uniform: Each program only works on one input length. So, to recognize a language $L \subseteq \Sigma^*$, there needs to be a branching program for each input size, which is called a branching program family B. Formally $B = \{B_n : n \in \mathbb{N}\}$, where each B_n is the branching program within the family that has n inputs.

1. ($\star$) Construct a branching program family for $PARITY$, that is the language of strings with an odd number of 1's.

2. ($\star\star$) For a fixed n. Construct a branching program family that recognizes the language of strings with 0 (mod n) number of 1's.

One resource is the width of the branching program, that is the maximum number of nodes on one level of the branching program (a level on its topological ordering, which is a partial order).

3. ($\star\star$) Notice that the $PARITY$ language above can be recognized by a width-2 branching program family. Construct such a branching program family.

4. ($\star\star$) Similarly, the mod n language can be recognized by a width-n branching program family. Construct such a branching program family.

Another resource is the size of the branching program: The number of nodes within the branching program (in relation to the size). For all of the questions we will restrict ourselves to branching programs with polynomial size.

5. ($\star\star$) Construct a branching program family for $MAJORITY$, which is the language with strings that have majority 1's

6. ($\star\star$) Construct a branching program family for $EQUALITY$. That is the languages with strings that have equal 1's and 0's. (Note that the odd and even numbered inputs may want to have vastly different branching programs).

7. ($\star\star$) Construct a branching program family that recognizes $1000\Sigma^* + 010^*$.