Tutorial 7

Gidon Rosalki

2025-05-14

1 Non deterministic Turing machines

Definition 1.1 (Nondeterministic Turing Machine - NTM)**.** *A non-deterministic Turing Machine is a 7-tuple*

$$M = \langle \Sigma, \Gamma, \sqcup, Q, Q_0, F, \delta \rangle$$

Where:

- Σ *is a finite alphabet*
- Γ *is a finite tape alphabet, such that* $\Sigma \subseteq \Gamma$
- $\sqcup \in \Gamma \setminus \Sigma$ *is the blank symbol*
- Q *is a finite, non empty, set of states*
- $Q_0 \subseteq Q$ *is the set of initial states*
- $F \subseteq Q$ *is the set of final states*
- $\delta : (Q \setminus F) \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$ *is the transition function*

The transition function maps a state and a tape symbol to a set of possible actions, where each action consists of a next state, a symbol to write, and a direction to move the tape head (left or right).

Definition 1.2 (Acceptance)**.** *Let N be an NTM and let $w \in \Sigma^*$. We say that N accepts w if there exists at least one accepting run of N on w - that is, a run that ends in q_{acc}. In this case, we write $w \in L(N)$.*

Example 1 (CLIQUE)**.** *Let $G = (V, E)$ be a graph. A **clique** is a set $S \subseteq V$uch that there is an edge between every pair of vertices in S. Define the following decision problem:*

$$CLIQUE = \{\langle G, k \rangle : \textit{There is a clique of size } k \textit{ in } G\}$$

Solution. We will describe an NTM that decides $CLIQUE$. The machine non-deterministically writes on the tape the names of k distinct vertices from V, separated by #. This guessing is done by replacing a blank section of the tape with symbols from the encoding of vertex names, one at a time, using non-deterministic transitions. Then, the machine checks deterministically whether all $\binom{k}{2}$ pairs among the chosen vertices are connected by edges in G. For each pair, it searches through the encoding of E to see if the corresponding edge exists.
If all pairs are connected, then the machine accepts, otherwise it rejects.
This machine runs in non-deterministic, polynomial time. Guessing the k vertex names takes polynomial time, since $k \leq |V|$. Checking all $\binom{k}{2}$ pairs for the existence of pairs takes polynomial time, by scanning the input encoding of G. Therefore, $CLIQUE$ can be solved by an NTM in polynomial time. □

We shall however note, that a trivial deterministic algorithm would try all subsets of k vertices and check if any of them forms a clique. However, the number of such subsets is $\binom{|V|}{k}$, which is exponential in the input size when k is not a constant because:

$$\binom{|V|}{\frac{|V|}{2}} \approx 2^{|V|}$$

Since each check takes polynomial time but must be repeated exponentially many times, this yields an exponential-time algorithm overall. Thus, the trivial approach does not give a polynomial-time solution.

2 P and NP

Definition 2.1 (Runtime of a TM)**.** *The **runtime** of a TM M on an input $w \in \Sigma^*$ is the number of steps that M performs before it halts.*

Definition 2.2 (Runtime of an NTM)**.** *The **runtime** of an NTM N on an input $w \in \Sigma^*$, is the number of steps that N performs, before halting, in the longest run on w.*

$$runtime_N(w) = \max\{|r| : r \textit{ is a run of } N \textit{ on } w\}$$

Definition 2.3 (Runtime complexity)**.** *For $f : \mathbb{N} \to \mathbb{N}$, we say that a TM/NTM T runs in time $O(f(n))$ if*

$$\forall w \in \Sigma^* \ runtime_T(w) \in O(f|w|)$$

Definition 2.4. *For $f : \mathbb{N} \to \mathbb{N}$,*

$$TIME(f(n)) = \{L \subseteq \Sigma^* : L \text{ can be decided by a TM that runs in } O(f(n))\}$$
$$NTIME(f(n)) = \{L \subseteq \Sigma^* : L \text{ can be recognised by an NTM that runs in } O(f(n))\}$$

Definition 2.5 (P/NP)**.** *The class of languages that can be decided in polynomial time, by the relevant TM/NTM:*

$$P = \bigcup_{k=1}^{\infty} TIME\left(n^k\right)$$
$$NP = \bigcup_{k=1}^{\infty} NTIME\left(n^k\right)$$

There is of course, the million dollar question, does $P = NP$? The proof is left as an exercise for the reader (note, finding a proof, or disproof, will result in getting 100% in the course, and a million dollars, but that is far less interesting).

Definition 2.6 (EXP)**.** *The class of languages that can be decided in exponential time by a TM*

$$EXP = \bigcup_{k=1}^{\infty} \left(2^{n^k}\right)$$

If a language can be decided by a TM that runs in polynomial or exponential time, then in particular it is decidable. Therefore it follows easily from the definition that $P, EXP \subseteq R$. Moreover, in the next homework we will show that $NP \subseteq EXP$, (by describing a TM that simulates all possible runs of the NTM), thus also $NP \subseteq R$

2.1 Closure properties of P/NP

Theorem 1 (P is closed under union)**.** *P is closed under union:*

Proof . Let $L_1, L_2 \in P$. Then there are 2 TMs, M_1, M_2, that decide L_1, L_2 respectively, and run in polynomial time. Let us define a new TM M, which operates as follows on the input $x \in \Sigma^*$:

- Run M_1 on x, and accept if it accepts
- Run M_2 on x, and accept if it accepts
- If neither accepted, then reject

The proof that $L(M) = L_1 \cup L_2$ is identical to the case of union in R (shown in a previous recitation). The runtime of M_1 and M_2 is polynomial, so the sum is also polynomial.
We will also note that during the run of M_1 on x, the machine M must keep a copy of x for later (to clean up the tape for M_2). This involves overhead: after each step that M_1 performs, M may have to move the copy. The cost is polynomial in $|x|$, per step. So the overall runtime of M is also polynomial. □

Theorem 2 (NP is closed under union)**.** *NP is closed under union*

Proof . Let $L_1, L_2 \in NP$. Then there are 2 TMs, N_1, N_2, that recognise L_1, L_2 respectively, and run in polynomial time. Let us define a new TM N, which recognises $L_1 \cup L_2$ as follows:

- On the input w, N will choose non-deterministically whether to simulate N_1 or N_2
- It then simulates the chosen machine on input w
- If the simulation accepts, then N accepts, otherwise it rejects

Since both N_1 and N_2 run in polynomial time, and the non-deterministic choice adds only a constant overhead, the machine N also runs in non-deterministic polynomial time (The maximum of the time of N_1 and N_2).
Moreover, N accepts w **if and only if** $w \in L_1 \vee w \in L_2$, so $L(N) = L_1 \cup L_2 \in NP$ □

Theorem 3 (P is closed under complement)**.** *Proof .* Let $L \in P$, and let M be a deterministic TM that decides L in polynomial time. We shall construct a machine M' by swapping the accepting and rejecting states of M. Since this transformation only modifies the final states and leaves the computation unchanged, M' still runs in polynomial time and decides the complement language L. Hence, $L \in P$. □

This proof does not extend to NP. Consider CLIQUE, and now consider the complement language

$$\overline{CLIQUE} = \{\langle G, k\rangle : \text{Every size } k \text{ subsets of vertices is not a clique}\}$$

To decide this language, we must confirm that **no** subset of size k forms a clique. A non-deterministic machine cannot verify this efficiently: it can guess a subset and reject if it is a clique, but if it guesses a non-clique, that tells us nothing about the existence of a different subset that might be a clique. Thus, flipping accept and reject states of the non-deterministic machine for $CLIQUE$ does not yield a valid machine for $\overline{CLIQUE}$, since rejection does not mean "no" in the presence of non-determinism. In fact, it is currently unknown whether NP is closed under complement; that is, whether $NP = coNP$ remains an open question.

3 Graph problems

We shall encode graphs as follows: For a graph $G = (V, E)$, we denote by $\langle G\rangle$ its encoding as a word over $\{0, 1, \#\}$. It contains a binary name for each vertex in V , and a pair of such names for each edge in E. Notice that this encoding is the same size as the graph $|V| + |E|$ (up to a polynomial).

Example 2. *Let $L = \{\langle G\rangle : G$ is a connected graph$\}$. We will show that $L \in P$ (remember, we did this in DAST using DFS).*

Solution.

- Choose an arbitrary vertex $v_0 \in V$
- Perform DFS starting at v_0
- If all vertices are visited, then the graph is connected, otherwise it is not

The time complexity taught in DAST was $O(|V| + |E|)$. This does not hold for TMs, since every iteration (checking whether or not a node has been visited, and finding its neighbours), requires scanning the tape several times. The TM is not as efficient, but, importantly, the difference in efficiency is polynomial. This is because the main difference between TMs and the model we are used to (random access memory) is that the TM cannot "jump" to a memory cell, so we pay a (polynomial) price in time per step. □

There are many important decision problems in graph theory, for example:

- An **independent set** in a graph $G = (V, E)$ is a subset $S \subseteq V$ such that no two vertices in S are connected by an edge
- A **Hamiltonian path** is a path that visits every vertex in a graph *exactly once*.

From these, we can define the following decision problems:

$$IS = \{\langle G, k\rangle : \text{There is an independent set of size } k \text{ in } G\}$$
$$HAM - PATH = \{\langle G\rangle : \text{There is a Hamiltonian path in } G\}$$

A natural question arises: which of these problems are in P , and which are in NP? How can we formally compare the complexity of such problems?
To answer this, we introduce the notion of polynomial-time reductions, which help us classify problems, and understand the relationships between them within the complexity hierarchy.

4 Polynomial reductions

Definition 4.1 (Computable in polynomial time)**.** *A function $f : \Sigma^* \to \Sigma^*$ is computable in polynomial time if there exists a TM M_f that runs in polynomial time, and for every input $x \in \Sigma^*$, when M_f halts, the word $f(x)$ is written to the tape*

Definition 4.2 (Polynomial reduction)**.** *A polynomial reduction from a language $A \subseteq \Sigma^*$ to a language $A \subseteq \Sigma^*$ is a function $f : \Sigma^* \to \Sigma^*$ from A to B that is computable in polynomial time. If such a reduction exists, we denote $A \leq_p B$.*

A big reason we introduced reductions was for the reduction theorem, so let us extend it to this set of definitions:

Theorem 4 (Reduction theorem for P)**.** *If $A \leq_p B$, and $B \in P$, then $A \in P$*

Proof . Suppose $A \leq_p B$. There is a TM M_f that computes the reduction in polynomial time. Since $B \in P$, there is some TM M_B that decides B in polynomial time. We define M_A which runs as follows on input $x \in \Sigma^*$:

- Run M_f on x to obtain $f(x)$
- Run M_B on $f(x)$, and accept or reject accordingly

Correctness is the same as the reduction theorem for mapping reductions. It remains to prove that this algorithm indeed runs in polynomial time.
The output of $M_f(x)$ is of length $O\left(|x|^k\right)$ for some constant k, and the runtime of M_B on an input of size $O\left(|x|^k\right)$ is $O\left(\left(|x|^k\right)^m\right)$ for some constant m. Therefore the running time is $O\left(|x|^{mk}\right)$, which is polynomial in $|x|$. □

Theorem 5. *Consider the languages*

$$CLIQUE = \{\langle G, k\rangle : \textit{There is a clique of size } k \textit{ in } G\}$$
$$IS = \{\langle G, k\rangle : \textit{There is an independent set of size } k \textit{ in } G\}$$

Prove that $CLIQUE \leq_p IS$

Proof . The idea is that cliques, and independent sets are complementary: we can convert between them by complementing the graph.

- Construction: Let $f(\langle G, k\rangle) = \left\langle \overline{G}, k\right\rangle$, where $\overline{G}$ is obtained from G by removing the edges, and then adding edges between vertices that were not connected before, or formally: $\overline{G} = \left(V, \overline{E}\right)$ where

$$\overline{E} = \{\{u, v\} : u \neq v \in G \wedge \{u, v\} \notin E\}$$

- Correctness:
  - Suppose $\langle G, k\rangle \in CLIQUE$. Then there is some set of vertices S of size k such that there is an edge between every two vertices in S. By definition of G, there are no edges at all between the vertices in S in G, so it is an independent set of size k and we have $\langle G, k\rangle \in IS$
  - Suppose $\left\langle \overline{G}, k\right\rangle \in IS$. Then there exists a set $S \subseteq V$ of size k such that there are no edges between any two vertices in S. Thus S is a clique in G, so $\langle G, k\rangle \in CLIQUE$
- Complexity: A TM can iterate over every $\{u, v\} \in V \times V$, check if it's in E and add or not add it to G accordingly. There are $O\left(|V|^2\right)$ pairs to check, and each takes polynomial time (we iterate over the edges of G and perform string comparisons). Thus the reduction can be computed in polynomial time

□

We don't know whether $CLIQUE$ and IS are in P or not. However now we know that if $IS \in P$ then $CLIQUE \in P$. There is also a polynomial reduction in the other direction (see exercise 7). So they "go together": either both are in P or both are not in P. We will see that this situation applies to many other problems.