1 NL-COMPLETE

Last time, we talked about L and NL.
A \leq_L B means A can reduce to B in log-space time.

If A \leq_L B and B \leq_L C then A \leq_L C \hfill (b)
In particular, if A \leq_L B, and B \in L, then A \in L.

Now that we showed \leq_L is transitive, it makes sense to talk about NP-completeness with respect to log-space reductions.

1.1 Definition 1 \mathcal{L} is NL-complete if \mathcal{L} \in NL and \forall \mathcal{L}' \in NL then \mathcal{L}' \leq_L \mathcal{L}

Definition 2 \text{PATH} =\{ < G, s, t > : G is a directed graph, s.t. there exists a path from } s \text{ to } t \text{ in } G \}\. G \text{ is given by its adjacency matrix.}

1.2 Theorem: Path is NL-complete

Proof
• (1) \text{PATH} \in NL
  Suppose G has n vertices. If there is a < G, s, t > \in \text{PATH}, then there exists a path of length at most n from s to t.
  Take a nondeterministic walk of length at most n on G starting at s. At each step along the walk, we can erase the previous vertex and update the counter by 1. If any of the nondeterministic choices encounters t then accept, else reject.

• (2) \text{PATH} is NL-hard
  Let \mathcal{L} \in NL, we want to show \mathcal{L} \leq_L \text{PATH} and want to find implicitly logspace computable function f : \{0,1\}^n \rightarrow \{0,1\}^* s.t. x \in \mathcal{L} \iff f(x) \in \text{PATH}
  We already know that there exists a Nondeterministic logspace Turing Machine M that decides L. Let \text{G}_{M,x} be the configuration graph of M on input x. M(x) = 1 \iff there exists a path from \text{C}_{\text{start}} to \text{C}_{\text{accept}} in \text{G}_{M,x} \iff f(x) \in < \text{G}_{M,x}, \text{C}_{\text{start}}, \text{C}_{\text{accept}} > \text{ (There are poly(n) vertices, and each configuration is log n space.) Verify that } < \text{G}_{M,x}, \text{C}_{\text{start}}, \text{C}_{\text{accept}} > \text{ is polytime and poly(x) bits long.}
  We need to show this adjacency matrix \text{G}_{M,x} is implicitly logspace computable, i.e. given configuration C and C', in space O(|C| + |C'|) = O(log(|x|)), a deterministic machine can examine C, C' and check whether C' is one of the configurations that can follow C according to the transition function of M.

2 coNL

Just as we define coNP and coNP completeness, now we can talk about coNL completeness.
\mathcal{L} is NL-complete \iff \overline{\mathcal{L}} is co-NL-complete.

In particular, \overline{\text{PATH}} = \{ < G, s, t > : G is a directed graph, and there is no path from s to t. \} is co-NL-complete.
2.1 Theorem: \{Immerman - Szelepcsényi 88’\}

NL = coNL (\text{PATH} \in \text{NL})

The read-once certificate characterize NL in term of certificates.
Language \(L\) is in \(NL\) if deterministic logspace Turing Machine \(M\) (the verifier) with an additional read-once input tape and a polynomial \(p: \mathbb{N} \to \mathbb{N}\) s.t \(\forall x \in \{0,1\}^*\),

\[ x \in L \iff \exists u \in \{0,1\}^{p(x)}, M(x,u) = 1 \]

\((x\) is an input tape, \(u\) is written on the special read-once input tape of \(M\))

\(M\) was at most \(O(\log(|x|))\) space on its read/write work-tape (for all \(x\)) It’s poly time to read-once from left to right.

Proof of Theorem:

Intuitive proof idea
We want to give a read-once poly-size certificate that there is no path in \(G\) from \(s\) to \(t\) (if there isn’t one) that can be verified in logspace. Let \(C_i\) be the set of vertices at distance at most \(i\) from \(s\). Suppose \(|C_n| = k\) and there was a read-once certificate proving this. We show that in this case we are done:

For each \(v \in C_n\), there is a simple certificate that proves \(v \in C_n\). Just give the path of length of at most \(n\) and start at \(s\), end at \(v\), which is \(s \to v\) path. Now the certificate for “t is not reachable from s” is given by the \(k\) certificates for \(v \in C_n\), and \(\forall v \in C_n\), given in increasing order of the vertices.

For instance, suppose \(v_1^* \leq v_2^* \leq ... \leq v_n^* \in C_n\). Given certificate for \(v_i^*\) in \(C_n\) in increasing order of \(i\),

\[ \bullet \text{ verifier maintains a counter.} \]

\[ \bullet \text{ verify that all certificates are correct.} \]

\[ \bullet \text{ t is not one of the vertices reachable with increasing of the order.} \]

We still need to certify \(|C_i|\). We will do this iteratively:

Claim 3 Given \(|C_i|, \forall v, we can certify if v \in C_{i+1} or not.\)

Proof of [: Claim 3] If \(v \in C_{i+1}\), give as the certificate the path of length \(\leq i + 1\) from \(s\) to \(v\). If \(v \not\in C_{i+1}\), list in lexicographical order each vertex in \(C_i\) along with its certificate in \(C_i\). The verifier tells that \(v\) is not a neighbor of those vertices in \(C_i\). Note that this certificate runs in \(n^i \log n\) time: \(n \log n\) if \(v \in C_{i+1}\) or \(n^i \log n\) if \(v \not\in C_{i+1}\), another factor of \(n\) for each \(v\), and another factor of \(n\) for each \(i\). So the certificate is polynomial length.

With the claim, we go through all of the vertices in lexicographical order and give the appropriate count for each one. We maintain a counter for the number of vertices in \(C_i\). This completes the proof.

3 Polynomial Hierarchy and Alternations

Now we consider what the space complexity for SAT is. It is conceivable that \(SAT \in L\). It is also conceivable that \(SAT \in DTIME(n)\). However, we will show that it cannot be in both.:
Before we prove this theorem, we must introduce the concept of Polynomial Hierarchy and Alterations:

**Definition 5** \( \Sigma^p_i := \{L : \exists \ a \ \text{polynomial time TM} \ M \ \text{and polynomial} \ q \ \text{such that} \ x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \ M(x,u,v) = 1\} \)

**Definition 6** \( \Pi^p_i := \{L : \exists \ a \ \text{polynomial time TM} \ M \ \text{and polynomial} \ q \ \text{such that} \ x \in L \iff \forall u \in \{0,1\}^{q(|x|)} \ M(x,u,v) = 1\} \)

Note that \( NP, Co-NP \subseteq \Sigma^p_2 \cap \Pi^p_2 \).

We can also have more general classes with more than 2 quantifiers:

**Definition 7** \( \Sigma^p_i := \{L : \exists \ a \ \text{polynomial time TM} \ M \ \text{and polynomial} \ q \ \text{such that} \ x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} (\forall u_2 \in \{0,1\}^{q(|x|)}) \ldots, Q_i, u_i \in \{0,1\}^{q(|x|)} M(x,u,v) = 1\}, \) where \( Q_i = \exists \) if \( i \) is odd, and \( Q_i = \forall \) if \( i \) is even.

**Definition 8** \( \Pi^p_i := \{L : \exists \ a \ \text{polynomial time TM} \ M \ \text{and polynomial} \ q \ \text{such that} \ x \in L \iff \forall u_1 \in \{0,1\}^{q(|x|)} (\exists u_2 \in \{0,1\}^{q(|x|)}) \ldots, Q_i, u_i \in \{0,1\}^{q(|x|)} M(x,u,v) = 1\}, \) where \( Q_i = \forall \) if \( i \) is odd, and \( Q_i = \exists \) if \( i \) is even.

Finally, we define the polynomial hierarchy class, \( PH \).

**Definition 9** \( PH := \bigcup_{i \geq 1} \Pi^p_i = \bigcup_{i \geq 1} \Sigma^p_i \).

One may see the phrase “Polynomial Hierarchy does not collapse.” This means \( \forall i, \Sigma^p_i \subset \Pi^p_{i+1} \) and \( \Pi^p_i \subset \Sigma^p_{i+1} \), where the containment is strict. If any of these containments happen to be equal for some \( i \), then the entire union of \( PH \) would collapse to that \( i \).

Here is an example of a language in one of the polynomial hierarchy classes:

**Example 10** \( IND-SET := \{\langle G, k \rangle : G \text{ is a graph with an independent set of size } k\} \). \( IND-SET \) is in \( NP \), since if we are given a set of \( k \) vertices, we can check to see if they are independent.

**EXACT-IND-SET := \{\langle G, k \rangle : \text{The largest independent set of graph } G \text{ has size } k\} \). For this we need to check that it has an independent set of size \( k \), and for all other collections of more than \( k \) vertices, they are not independent. So this has both a “\( \exists \)” clause and a “\( \forall \)” clause. Since for this particular language, the order of the \( \exists \) and \( \forall \) does not matter, it is both in \( \Sigma^p_2 \) and in \( \Pi^p_2 \).

We study Polynomial Hierarchy and Alterations to model interesting questions, shed light on \( TIME \) versus \( SPACE \), and in particular, study \( TQBF \) which is \( PSPACE \)-complete. We note that \( PH \subseteq PSPACE \), and they are equal only if \( PH \) collapses.

**Theorem 11** If \( \Sigma^p_i = \Pi^p_i \), then \( PH \subseteq \Pi^p_i \).

In particular, if \( P=NP \), then \( P=PH \).

## 4 Alternating Turing Machines

**Definition 12** An Alternating Turing Machine (ATM) has the same properties as a normal Turing Machine except that each state also has an attached \( \exists \) or \( \forall \). The ATM accepts after a \( \exists \) if at least one path following that state accepts, and accepts after a \( \forall \) if all paths following that state accept.

Given an ATM we can look at its time and space complexities, but we can also study the maximum number of times it alternates between \( \exists \) and \( \forall \) states.