Last time we moved on to space complexity.

- Amount of memory used on read-write work tapes (read-only input tape doesn't count)

\[
\text{DTIME} (s(n)) \leq \text{SPACE} (s(n)) \\
\leq \text{NSPACE} (s(n)) \\
\leq \text{DTIME} (2^{O(s(n))})
\]

Configuration graph \( G_M, x \)
- Vertices = possible states of TM \( M \) on input \( x \)
- Edges = transitions
- DTM: out-degree 1
- NDTM: out-degree 2

Can we say more about how space complexity classes such as
- \( \text{PSPACE} \) = polynomial-space computation
- \( \text{NPSPACE} \) = non-deterministic polynomial-space computation
- \( L \) = logarithmic space computation
- \( NL \) = non-deterministic log-space computation

relate to other complexity classes that we know and love?
**Proposition 7**  
**NPS PSPACE**

**Proof**  
- Reduce from CNF-SAT.
- Check all truth value assignments in lexicographic order (linear space in size of CNF formula).
- Accept if satisfying assignment found.
- Otherwise reject once all assignments tested.

**Open Problem 8**  
**NP ≠ L ?**

**Example 9**

Let \( \text{PATH} = \{ <G,s,t> | \exists \text{path from } s \text{ to } t \text{ in } G \} \) 
\( \text{PATH} \in \text{NL} \)

**Proof**  
If there is a path, there is one of length \( \leq n = |V(G)| \)
Keep counter \([1..n]\) \(-\text{log}_n \text{G's}\) 
Walk nondeterministically (guess next vertex and check in input tape that this is OK). 
Accept if reached \( t \) before counter exceeds \( n \) 
(Vertex indices also require space \( O(\log n) \).) 

Is \( \text{PATH} \) in \( L \)?  
Excellent question

Would imply \( L = \text{NL} \) (i.e., \( \text{PATH} \) is \( \text{NL} \)-complete, will be discussed later.

Interestingly, [Rengold '05] proved that \( \text{UNDIRECTED PATH} \) is in \( L \) \( \mathbb{E} \) (Major result)
THEOREM 10 SPACE HIERARCHY THEOREM

[sleems, Hartmanis & Lewis '65]
If \( f, g \) are space-constrainable functions s.t. \( f(n) = o(g(n)) \) then \( \text{SPACE}(f(n)) \neq \text{SPACE}(g(n)) \)

Proof: will skip this. Might be good exercise.

DEF 11 PSPACE-COMPLETE LANGUAGES
\( L \) is PSPACE-hard if \( L \leq_p L' \) for every \( L \in \text{PSPACE} \). If in addition \( L' \in \text{PSPACE} \) then \( L' \) is PSPACE-complete.

A (not so interesting) PSPACE-complete language

\[ \text{SPACE-BOUNDED TM} = \{ \langle M, x, 1^n \rangle \mid \text{M accepts } x \text{ in space } n \} \]

Note: problem set 1.

Let's look at a more interesting problem.

DEF 12 A quantified Boolean formula (QBF) is a formula on the form
\[ \varphi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi(x_1, \ldots, x_n) \]

where \( Q_i \in \{ \forall, \exists \} \)
\( x_i \) ranges over \( \{0, 1\} \)
\( \varphi \) is a DNF/CNF formula
(not necessary, and Arora-Papadoni don't require this)

PREPARED NORMAL FORM: all quantifiers to the left.
Can easily convert to prenex.
Can easily convert to CNF/3-CNF (skip details)

Note: QBFs have determined truth value - either true or false.

Example 13

\[ \forall x \exists y (x \land y) \lor (\neg x \land \neg y) \]
"for all \( x \) exists \( y \) s.t. \( x = y \)" - true

\[ \forall x \forall y (x \land y) \lor (\neg x \land \neg y) \]
"for all \( x \) and all \( y \) they are always equal" - false

SAT - QBF with all quantifiers \( \exists \)
UNSAT - QBF \( \forall \ldots \forall \) (and negated CNF inside)

**THEM 14** [Stockmeyer & Meyer '73]
The language

\[ \text{TQBF} = \{ \psi \mid \psi \text{ is a true QBF} \} \]
is \( \text{PSPACE} \)-complete

**Proof** \( \text{TQBF} \subseteq \text{PSPACE} \) (sketch)

Let \( \psi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi(x_1, \ldots, x_n) \) \( |\psi| = m \)

Base case: If all variable set to values, just evaluate \( \varphi \) in \( O(m) \) time and space.
Induction step

\[ \forall x, \psi \]

\[ \text{set } x_i = 0, \text{ evaluate, save } \]
\[ \text{set } x_i = 1, \text{ evaluate, save } \]

\[ \forall x, \psi, \text{ one iff both values one} \]
\[ O(1) \text{ extra space} \]

\[ \exists x, \psi' \]

Similar, just check if one of \( x_i = 0 \) and \( x_i = 1 \) yields one value.

Total space usage something like \( O(m+n) \)

\[ L \in \text{SPACE} \Rightarrow L \in \text{P} \text{ if } TQBF \]

\[ M \text{ decides } L \text{ in space } s(n) \]

Want to construct QBF \( \psi \) of size \( O(s(n)^2) \)

s.t. \( \exists \psi \) one \( \iff M \text{ accepts } x \)

Let \( m = K \cdot s(n) \) = \#bits needed to encode config of \( M \) on input \( x \).

By Claim 5.3, \( \exists \text{CNF } \varphi_{M,x} \) s.t. for \( C, C' \in \{0,1\}^m \), \( \varphi_{M,x}(C, C') = 1 \) if \( C \) and \( C' \) adjacent TM configs.

Use \( \varphi_{M,x} \) to define \( \psi \) s.t. \( \psi(C, C') = 1 \) if

\[ \exists \text{ path } C \leftrightarrow C' \text{ in } G_{M,x} \]

Plug in Start and Accept \( \Rightarrow \) Done!
Inductive definition
\[ \psi_i(C, C') = 1 \text{ iff } \exists \text{ path } C \rightarrow C' \text{ of length } \leq 2^i \]

\[ \psi_0 = \psi_{m, k} \]

After \( O(m) \) steps, get \( \psi = \psi_{0(m)} \).

If \( \exists \) path of length \( 2^i \), then \( \exists \) midpoint \( C'' \) s.t.

\[ \psi_{i-1}(C, C'') \land \psi(C'', C') \]

Why not \( \psi(C, C') = \exists C'' \psi_{i-1}(C, C'') \land \psi_{i-1}(C'', C') \)?

Not good: size doubles at each step \( \Rightarrow \) exponential blow-up.

Need poly-size formula!

**Attempt 2**

there are 2 variables in variables each

\[ \psi_i(C, C') = \exists C'' \land D^2 \land D^2 \left( (D^2 = C \land D^2 = C'') \lor (D^2 = C'' \land D^2 = C') \right) \]

\[ \Rightarrow \psi_{i-1}(D^2, D^2) \]

\[ \land \land \Rightarrow \text{ are just convenient shorthands.} \]

Can convert to CNF and presolve without problems

"There is a midpoint \( C'' \) s.t. whenever

\( D^1 \) is the starting point and \( D^2 \) is the midpoint of \( D^2 \) is the midpoint and \( D^2 \) is the endpoint \( C' \),

then there is a path from \( D^1 \) to \( D^2 \) in length \( \leq 2^{i-1} \). The rest is just details..."
A funny observation

Proof of Thm 14 established that anything in PSPACE reduces to TQBF
via analysis of $G_{mix}$.

But we never used out-degree 1 criterion.

So... $G_{mix}$ could have been graph for
NDTM $M$.

So... TQBF is NPSPACE-hard.

**Corollary 15**

\[ \text{PSPACE} = \text{NSPACE}. \]

Can actually prove something slightly more precise

**Theorem 16 (Savitch's Theorem '70)**

For any space-constructible \( s(n) \geq \log n \)

\[ \text{NSPACE} \left( s(n) \right) \subseteq \text{SPACE} \left( s(n)^2 \right). \]

*Proof sketch* Implement reduction in Thm 14
as recursive top-down procedure.

Start with upper bound \( 2^{O(s(n))} \)

Check for all vertices in $G_{mix}$ if can be midpoint

\( O(s(n)) \) space. Recurse

\( O(s(n)) \) space per recursive call + \( O(s(n)) \) recursive

calls $\Rightarrow$ space \( O(s(n)^2) \).
PSPACE: Optimal strategies for playing games

View QBF as game

\[ \exists x_1, \forall x_2 \exists x_3 \forall x_4 \ldots \ \exists (x_1, x_2, x_3, x_4, \ldots) \]

\[ \exists \text{-player wants to choose } x_i \text{ such that for any choice by } \forall \text{-player of } x_2 \text{ the formula } \exists \text{ can be forced to true} \]

\[ \forall \text{-player wants to choose } x_2 \text{ such that no choice for } x_3 \text{ by } \exists \text{-player can make } \exists \text{ true} \]

\[ \exists \text{-player has winning strategy } \iff \text{QBF true} \]

\[ \forall \text{-player } \neg \exists \text{ } \iff \text{QBF false} \]

Can model other 2-player games with perfect information in this way.

Many such games are PSPACE-complete.

Hard to see how winning strategy for 1st player could have concise description for all responses to 2nd player moves.

i.e., we are arguing that it seems likely that

\[ \text{NP} \neq \text{PSPACE} \]

(but this is open)

Moving on next to sublinear space...
When studying logarithmic space and reducing between problems, polynomial-time reductions are no good

So powerful that the reduction can solve the problem.

Clearly, we don’t want reduction to be more powerful than actual algorithm. Hence, let us insist on reductions in logarithmic space.

Ok, good, but...

How can a log-space reduction compute polynomial-size output?

Two solutions:

1. Write-only output tape on which space doesn’t count.

Write once  write and move right
Never read; never move left

2. Compute reduction bit by bit

Get equivalent definitions (good exercise to show, we go for option 2)
DEF 1

\[ f: \{0, 1\}^* \rightarrow \{0, 1\}^* \text{ implicitly logspace computable if} \]

- a) \( f \) polynomially bounded \( (\exists c \in \mathbb{R} \mid |f(x)| \leq c|x|^c) \)
- b) \( \ell_f = \sum_{i} \langle x, i \rangle \mid f(x)_i = 1 \) \( \ell_f' = \sum_{i} \langle x, i \rangle \mid i \leq |f(x)| \) are both in \( L \)

Language \( B \) is **logspace reducible** to language \( C \), denoted \( B \leq_L C \) if \( \exists \) implicitly logspace computable \( f \) s.t. \( x \in B \iff f(x) \in C \)

\( C \) is **NL-complete** if \( C \in NL \) and \( \forall B \in NL \quad B \leq_L C \)

**Proposition 2**

1. \( B \leq_L C \text{ and } C \leq_L D \rightarrow B \leq_L D \)
2. \( B \leq_L C \text{ and } C \leq_L L \rightarrow B \leq_L L \)

**Proof**

Not hard but needs a bit of care. See textbook.

**Theorem 3**

PATH is NL-complete

Recall PATH = \( \{ \langle G, s, t \rangle \mid \exists \text{ path } s \rightarrow t \text{ in digraph } G \} \)

**Proof**

Argued PATH is NL last time.

Let \( B \) in NL decided by \( M \) in log space.

Define \( f(x) \) to be configuration graph \( G_{M, x} \)

Together with \( |C_{start}| = s \) and \( t = \text{Accept} \).

Represent an adjacency matrix

- \( 2 \) in position \( (C, C') \) if \( C, C' \) legal transition.
- \( \frac{\text{size } G_{M, x}}{2} \) has \( \leq \) 2 space vertices. \( g(\log) = \text{poly} \rightarrow \text{OK} \).

Computation given \( C, C' \), look up current state and representations and check that \( C' \) is one of two possible configs to follow from \( C \).
Certificate-style definition of \( \text{NP} \)?

For every \( x \in \text{B \subseteq \text{universy}} \)

s.t. \( M(x, y) = 1 \) and \( M \) runs in log-space

Need to be careful!

Suppose \( x \in \text{CNF formula} \), \( y \) satisfying assignment

Let \( M \) look up clauses in \( x \) one by one

then look up assignments in \( y \)

check that every clause satisfied.

Paves that \( \text{CNF-SAT} \in \text{NP} \)

Hence \( \text{NP} = \text{NL} \) (and \( \text{P} = \text{NP} \)) — great!

Fix: Make certificate read-once

**Definition**

Certificate-style definition of \( \text{NL} \)

\( c \) is in \( \text{NL} \) if exists deterministic TM \( M \) (verifier)

with

- read-only input tape

- read-once certificate tape \( [ \text{read or more right each step} ] \)

- read-only tapes with \( O(\log |x|) \) space bound

s.t.

\( x \in c \iff \exists u \in \{0, 1\}^{p(|x|)} \) s.t. \( M(x, u) = 1 \)

(for some fixed poly \( p \) depending on \( c \)).

**Lemma 5**

Definitions 4 gives exactly the same class \( \text{NL} \)

**Proof**

Exercise (not hard but useful).
Complements of space-bounded complexity classes

$$PATH = \{ \langle G, s, t \rangle \mid \text{No path } s \rightarrow t \text{ in digraph } G \}$$

PATH is coNL (since PATH is NL)

In fact, PATH is coNL-complete (since PATH is NL-complete).

Log space NDTM deciding PATH:

Just walk nondeterministically for \(IV(5)\) steps from \(s\), reject if didn’t reach \(t\).
Most computation paths might reject, but if I path then one branch will find it.

Log space NDTM deciding PATH

Walk nondet & accept if didn’t reach \(t\)?
A non-starter...

How can you make sure all branches find a path \(s \rightarrow t\) for a no instance of PATH?!

Obviously can’t be done, right? Seems clear that NL \(\neq\) coNL, right? Wrong.

**Theorem 6** NL = coNL

Immerman 88, Szelescsényi 87

**Proof** Show that PATH is NL.

Same ideas yield stronger statement (which we will not prove)

**Corollary 7** For every space constructible \(s(n) > \log n\), it holds that
\(NSPACE(s(n)) = coNSPACE(s(n))\)
Proof of Thm 6

Provide read-once certificate for \( \text{NP-complete language \ PARTITION} \).

Important read-once access to certificate but can scan graph \( G \) as many times as wanted (but not store on work tape).

\[
R(i) := \{ v \in V(G) \mid \exists \text{ path } s \\text{ to } v \\text{ of length } \leq i \}
\]

\[
n = |V(G)| \quad \text{denote } V(G) = \{1, 2, \ldots, n\} \quad n \in \mathbb{N}
\]

Want to certify \( x \notin R(n) \).

Starting point: certifying \( V \notin R(i) \) easy.

Give vertices in path \( u_0 = v, u_1, u_2, \ldots, u_i = v \) for \( i \leq i \).

Verification - read vertices one by one:
- keep \( u_j \) and \( u_{j+1} \) in memory - log space
- keep \( j \) in memory - log space
- at each step, check \( (u_j, u_{j+1}) \in E(G) \)
- check that \( j \) never exceeds \( i \).

Let such a certificate be denoted

\[
\text{Is MEMBER (} v, i) = v \in R(i)
\]

Use this to construct two other types of certificates.
(A) **Membership Expansion** $(i, s, r')$

Assuming $|R(i-1)| = r'$

proof that $|R(i)| = r$

(B) **Not Member** $(v, i, r)$

Assuming that $|R(i')| = r$

proof that $v \notin R(i)$

Suppose we can build such read-once verifiable subcertificates. Then we're done!

We all know $R(0) = \emptyset$ and $|R(0)| = 1$

Let $r_i = |R(i)|$

Here is the certificate

\[
\begin{align*}
\text{Membership Expansion} & \ (1, r_2, 1), \\
\text{Membership Expansion} & \ (2, r_2, r_1), \\
\text{Membership Expansion} & \ (3, r_3, r_2), \\
\text{Membership Expansion} & \ (n, r_n, r_{n-1}), \\
\text{Not Member} & \ (v, i, r) \\
\end{align*}
\]

- check each line in read-once fashion
- keep counter $i$ and neighborhood size $r_i$
- log in space
- finally verify nonmembership certificate
- each expansion certificate for step $j$ is verified using stored $r_{j-1}$ log in space
\[ \text{NotMember} \ (v, i, r) \]

Suppose \( R(i) = \{ u_1, u_2, \ldots, u_r \} \) \( u_i < u_{i+1} \ldots < u_r \). Let certificate be sorted list of \( u_i \)'s with membership certificates

\[
\begin{align*}
  u_1 &: \text{IsMember} \ (u_1, i) \\
  u_2 &: \text{IsMember} \ (u_2, i) \\
  \vdots \\
  u_r &: \text{IsMember} \ (u_r, i)
\end{align*}
\]

Denote this by \( \text{ListMembers} \ (i, r) \).

**Verification**

- \( r \) is known
- go over list and read \( u_j \)
- for each \( u_j \), check certificate of membership
- check \( u_j > u_{j-1} \)
- check \( u_j \neq v \)
- check \# \( u_j \)-vertices = \( r \)

\[ \text{MembershipExp} \ (i, r, r') \]

Use auxiliary certificate \( \text{NotMemberOrNeighbour} \ (v, i, r') \)

Assuming \( \mid R(i) \mid = r' \), prove that \( v \notin R(i+1) \)

\[ \text{ListMembers} \ (i, r') \]

**Verification**

- Create list and verify \( u_j \)'s as above
- For each \( u_j \), check \( u_j \neq v \)
- and that \((u_j, v) \notin E(G)\)
Now we can write down
\[
\text{Membership Expansion} \ (i, \tau, r') \nonumber
\]
as an ordered list of subcertificates for vertices \(1, 2, \ldots n\)

If vertex \(j \in R(i)\), the line for \(j\) is
\[
\boxed{j: \ \text{isMember} \ (j, i)} \nonumber
\]

If vertex \(j \notin R(i)\), the line for \(j\) is
\[
\boxed{j: \ \text{notMemberOrNeighbour} \ (j, i-1, r')} \nonumber
\]

which is \(= \text{listMembers} \ (i-1, r')\)

\[\text{Verification}\]
- For each \(j\), check correctness of membership or non-membership certificate.
- Count total # members; check that sum is \(= r\).

This concludes the proof.

\[\text{SUMMARY OF THE COURSE SO FAR:}\]
\[L \subseteq NL \subseteq P \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXP}\]

Some inclusions must be strict since
- \(L \not\subseteq \text{PSPACE}\) (space hierarchy theorem)
- \(P \not\subseteq \text{EXP}\) (time hierarchy theorem)

But we don't know which... (Probably most, often all)