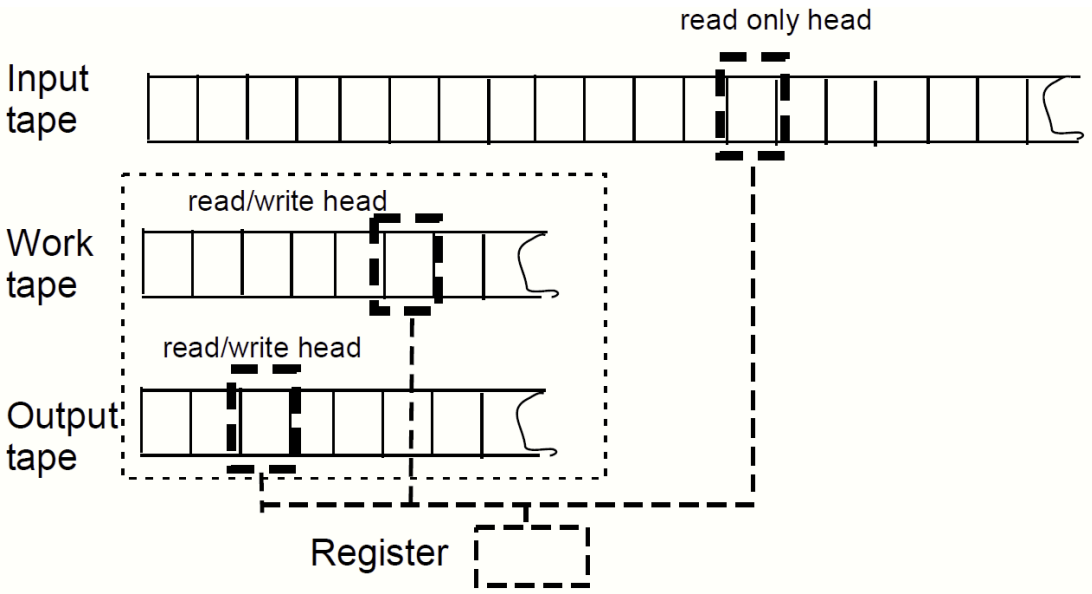
Definition 4.1 *(Space-bounded computation)*
Let $S : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0, 1\}^*$. We say that $L \in$ **SPACE**$(s(n))$ if there is a constant c and a TM M deciding L such at most $c \cdot s(n)$ locations on M's work tapes (excluding the input tape) are ever visited by M's head during its computation on every input of length n.
Similarly, we say that $L \in$ **NSPACE**$(s(n))$ if there is an NDTM M deciding L that never uses more than $c \cdot s(n)$ nonblank tape locations on length n inputs, regardless of its nondeterministic choices.



We will require however that S(n) > log n since the input tape has length n, and we would like the machine to at least be able to "remember" the index of the cell of the input tape that it is currently reading.

In-Class:

1. What is an efficient space bound for *finding the largest number in a sequence*
2. What is an efficient space bound for *sorting a sequence of numbers.*
3. What is an efficient space bound for *performing binary search on a sequence.*

DTIME(S(n)) ⊆ **SPACE**(S(n))

SPACE(S(n)) machine can run for much longer than S(n) steps, since space can be *reused*.

DTIME(S(n)) ⊆ **SPACE**(S(n)) ⊆ **NSPACE**(S(n)) ⊆ **DTIME**($2^{O(S(n))}$)

Configuration graphs:

1. A configuration of a TM consists of the contents of all tape cells, locations of all pointers, and the state, at a particular point in its execution.

2. For every TM M and input x ∈ {0, 1}*, the configuration graph of M on input x, denoted $G_{M,x}$, is a **directed graph** whose nodes correspond to possible configurations that M can reach from the starting configuration $C_{x,start}$ (where the input tape is initialized to contain x).

3. The graph has a directed edge from a configuration C to a configuration C' if C' can be reached from C in one step according to M's transition function.

4. If M is deterministic then the graph has **out-degree 1**, and if M is non-deterministic then it has an out-degree **at most *b***.

5. Also note that we can assume that M's computation on x does not repeat the same configuration twice (as otherwise it will enter into an infinite loop) and hence that the graph is a directed acyclic graph (DAG).

6. M accepts the input x *iff* there exists a (directed) path in $G_{M,x}$ from C_{start} to C_{accept}

Proof: **NSPACE**(S(n)) ⊆ **DTIME**($2^{O(S(n))}$)
Configuration graph consists of $2^{O(S(n))}$ nodes and each node is O(S(n)) length.

PROOF OF THEOREM 4.2: Clearly $\mathbf{DTIME}(S(n)) \subseteq \mathbf{SPACE}(S(n)) \subseteq \mathbf{NSPACE}(S(n))$, and so we just need to show $\mathbf{NSPACE}(S(n)) \subseteq \mathbf{DTIME}(2^{O(S(n))})$. By enumerating over all possible configurations, we can construct the graph $G_{M,x}$ in $2^{O(S(n))}$-time and check whether $C_{\mathtt{start}}$ is connected to $C_{\mathtt{accept}}$ in $G_{M,x}$ using the standard (linear in the size of the graph) breadth-first search algorithm for connectivity (e.g., see [CLRS01]). ■

$$\mathbf{PSPACE} = \cup_{c>0}\mathbf{SPACE}(n^c)$$
$$\mathbf{NPSPACE} = \cup_{c>0}\mathbf{NSPACE}(n^c)$$
$$\mathbf{L} = \mathbf{SPACE}(\log n)$$
$$\mathbf{NL} = \mathbf{NSPACE}(\log n)$$

P ⊆ NP ⊆ PSPACE

Show that 3SAT ∈ **PSPACE.**

Show that **NP** ⊆ **PSPACE.**

Show that TAUTOLOGY ⊆ **PSPACE.**

Show that **coNP** ⊆ **PSPACE**

$P \subseteq NP \cup coNP \subseteq PSPACE$

PSPACE Completeness: We will use Karp reduction here as well! Why do you think that is?

A PSPACE Complete language: **TQBF**

$$\Psi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \varphi(x_1, \dots, x_n)$$

Let's try to understand the recursive algorithm using the example of f(n) = 2*f(n-1)

TQBF can be solved *recursively* using a space-reusing bottom up recursive algorithm

TQBF $\in$ **PSPACE.** We design a (space-recycling) recursive algorithm A as follows. Consider an input of the form $\Psi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \varphi(x_1, \dots, x_n)$.
Base case: If $n = 0$, then φ is a constant, so just output it.
Recursive case:
If $Q_1 = \exists$:

- Run $A(\Psi|_{x_1=0})$
- Run $A(\Psi|_{x_1=1})$
- Accept if either run accepts.

If $Q_1 = \forall$:

- Run $A(\Psi|_{x_1=0})$
- Run $A(\Psi|_{x_1=1})$
- Accept if both runs accept.

Correctness holds by induction on the number of quantifiers of the formula.

Why is TQBF **PSPACE-Hard**: Any L $\in$ PSPACE can be encoded as a string of TQBF. The proof is similar to the Cook-Levin theorem which we have omitted in the course.

Recall that Sudoko was NP-Complete

Puzzles are essence of NP-Completeness

Whereas **2-Player perfect information games** are essence of PSPACE-Completeness

What is a 2-Player perfect information game?

2 players alternately make moves, and
board is visible to both (perfect information)
Does player 1 have a winning strategy? (requires us to search the tree of all possibilities)

searching the tree of all possible moves. Note that the first player has a winning strategy iff there is a 1st move for player 1 such that for every possible 1st move of player 2 there is a 2nd move of player 1 such that.... (and so on) such that at the end player 1 wins. The interplay of existential and universal quantifiers in the previous line suggests the following example of a game.

In NP, we also had to search for the certificate/"winning strategy", but unlike NP, even describing the certificate seems to require exponential time here! do you see why?

LINSPACE = SPACE(n)

3SAT ∈ LINSPACE

Show that the following languages are in **L**.

$$\text{EVEN} = \{x : x \text{ has an even number of 1s}\}$$

$$\text{MULT} = \{(\llcorner n \lrcorner, \llcorner m \lrcorner, \llcorner nm \lrcorner) : n \in \mathbb{N}\}$$

PATH = {G, s, t : G is a directed graph in which there is a path from s to t} ∈ **NL**

Nondeterministically walk from *s* along every possible path.
Keep count of how many nodes have been encountered.
If *t* is encountered, accept, else if *n* nodes have been encountered, *reject*.

L ⊆ NL ⊆ P ⊆ NP ⊆ PSPACE ⊆ NPSPACE ⊆ EXP ⊆ NEXP

NL-Completeness:
When choosing the type of reduction to define completeness for a complexity class, we must keep in mind the complexity phenomenon we seek to understand. In this case, the complexity question is whether or not **NL** = **L**.

logspace reductions = the reduction *f* is *implicitly logspace computable*.

DEFINITION 8.21

A ***log space transducer*** is a Turing machine with a read-only input tape, a write-only output tape, and a read/write work tape. The head on the output tape cannot move leftward, so it cannot read what it has written. The work tape may contain $O(\log n)$ symbols. A log space transducer M computes a function $f: \Sigma^* \longrightarrow \Sigma^*$, where $f(w)$ is the string remaining on the output tape after M halts when it is started with w on its input tape. We call f a ***log space computable function***. Language A is ***log space reducible*** to language B, written $A \leq_{\mathrm{L}} B$, if A is mapping reducible to B by means of a log space computable function f.

THEOREM 8.23

If $A \leq_{\mathrm{L}} B$ and $B \in \mathrm{L}$, then $A \in \mathrm{L}$.

PROOF A tempting approach to the proof of this theorem is to follow the model presented in Theorem 7.31, the analogous result for polynomial time reducibility. In that approach, a log space algorithm for A first maps its input w to $f(w)$, using the log space reduction f, and then applies the log space algorithm for B. However, the storage required for $f(w)$ may be too large to fit within the log space bound, so we need to modify this approach.

Instead, A's machine M_A computes individual symbols of $f(w)$ as requested by B's machine M_B. In the simulation, M_A keeps track of where M_B's input head would be on $f(w)$. Every time M_B moves, M_A restarts the computation of f on w from the beginning and ignores all the output except for the desired location of $f(w)$. Doing so may require occasional recomputation of parts of $f(w)$ and so is inefficient in its time complexity. The advantage of this method is that only a single symbol of $f(w)$ needs to be stored at any point, in effect trading time for space.

THEOREM 8.25

PATH is NL-complete.

PROOF IDEA Example 8.19 shows that *PATH* is in NL, so we only need to show that *PATH* is NL-hard. In other words, we must show that every language A in NL is log space reducible to *PATH*.

The idea behind the log space reduction from A to *PATH* is to construct a graph that represents the computation of the nondeterministic log space Turing machine for A. The reduction maps a string w to a graph whose nodes correspond to the configurations of the NTM on input w. One node points to a second node if the corresponding first configuration can yield the second configuration in a single step of the NTM. Hence the machine accepts w whenever some path from the node corresponding to the start configuration leads to the node corresponding to the accepting configuration.

Let's say NTM A decides A in O(log n) space. Any one edge of the configuration graph can be computed using logspace.

For configurations c_1 and c_2 of M on *w*, the pair (c_1, c_2) is an edge of G if c2 is one of the possible next configurations of M starting from c_1. Each node takes O(log n) space.

Two important problems were posed regarding SPACE complexity:

1. Is NSPACE = DSPACE?
2. Is NSPACE = co-NSPACE?

A negative answer to the 2nd question implies a negative answer to the first one. Why?

Immerman–Szelepcsényi Theorem

PATH-complement $\in$ NL

therefore, NL = coNL

$$\mathrm{L} \subseteq \mathrm{NL} = \mathrm{coNL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE}.$$

Savitch's Theorem

NSPACE(f) $\subseteq$ SPACE(f^2)

therefore PSPACE = NPSPACE

$$\mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE}$$
$$\mathsf{PSPACE} \subseteq \mathsf{EXPTIME} \subseteq \mathsf{EXPSPACE}$$
$$\mathsf{NL} \subsetneq \mathsf{PSPACE} \subsetneq \mathsf{EXPSPACE}$$
$$\mathsf{P} \subsetneq \mathsf{EXPTIME}$$

Note: PH = Polynomial Hierarchy (haven't covered)

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME}.$$