Theory of Computation
Time and Space Complexity Classes

Time for Deciding a Language

- Let us consider $A = \{\mathtt{0}^n\mathtt{1}^n : n \geq 0\}$.
- How much time does a single-tape TM need to decide A?
- Consider
  M_1 = "On input string w:
  1. Scan the tape and reject if a `0` appears after a *1*.
  2. Repeat if `0` or `1` appear on the tape:
     1. Scan across the tape, cross a `0` and a `1`.
  3. If `0`'s or `1`'s still remain, reject. Otherwise, accept."
- How much "time" does M_1 need for an input w?

Time Complexity

Definition 1

Let M be a TM that halts on all inputs. The running time (or time complexity) of M is the function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the running time of M on any input of length n.

- If $f(n)$ is the running time of M, we say M runs in time $f(n)$ and M is an $f(n)$ time TM.
- In worst-case analysis, the longest running time of all inputs of a particular length is considered.
- In average-case analysis, the average of all running time of inputs of a particular length is considered instead.
- We only consider worst-case analysis in the course.

Big-O and Small-O

Definition 2

Let $f, g : \mathbb{N} \to \mathbb{R}^+$. $\underline{f(n) = O(g(n))}$ if there are $c, n_0 \in \mathbb{Z}^+$ such that for all $n \geq n_0$,

$$f(n) \leq c(g(n)).$$

- $g(n)$ is an upper bound (or an asymptotic upper bound) for $f(n)$.
- $n^c (c \in \mathbb{R}^+)$ is a polynomial bound.
- $2^{n^d} (d \in \mathbb{R}^+)$ is an exponential bound.

Definition 3

Let $f, g : \mathbb{N} \to \mathbb{R}^+$. f (n) = o (g (n)) if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

That is, for any $c \in \mathbb{R}$, there is an n_0 that $f(n) < c(g)$ for all $n \geq n_0$.

Time Complexity of M_1

- Recall
  M_1 = "On input string w:
  1. Scan the tape and reject if a `0` appears after a *1*.
  2. Repeat if `0` or `1` appear on the tape:
     1. Scan across the tape, cross a `0` and a `1`.
  3. If `0`'s or `1`'s still remain, reject. Otherwise, accept."
- Let $|w| = n$.
  - Step 1 takes $O(n)$ (precisely, $\leq n$).
  - Step 2 has $O(n)$ iterations (precisely, $\leq n/2$).
    - An iteration takes $O(n)$ (precisely, $\leq n$).
  - Step 3 takes $O(n)$ (precisely, $\leq n$).
- The TM M_1 decides $A = \{\texttt{0}^n\texttt{1}^n : n \geq 0\}$ in time $O(n^2)$.
  - $O(n^2) = O(n) + O(n) \times O(n) + O(n)$.

Time Complexity Class

Definition 4

Let $t : \mathbb{N} \to \mathbb{R}^+$. The time complexity class $TIME(t(n))$ is the collection of all languages that are decided by a $O(t(n))$ time TM.

- $A = \{\texttt{0}^n\texttt{1}^n : n \geq 0\}$ is decided by M_1 in time $O(n^2)$. $A \in TIME(n^2)$.
- Time complexity classes characterizes languages, not TM's.
  - We don't say $M_1 \in TIME(n^2)$.
- A language may be decided by several TM's.
- Can A be decided more quickly asymptotically?

Models and Time Complexity

- Consider the following TM:
  M_2 = "On input string w:
  1. Scan the tape and reject if a `0` appears after a `1`.
  2. Repeat if `0` or `1` appear on the tape:
     1. Scan the tape and check if the total number of `0`'s and `1`'s is even. If not, reject.
     2. Scan the tape, cross every other `0` from the first `0`, and cross every other `1` from the first `1`.
  3. If `0`'s or `1`'s still remain, reject. Otherwise, accept."
- Analysis of M_2.
  - Step 1 takes $O(n)$.
  - Step 2 has $O(\lg n)(= \log_2(n))$ iterations (why?). At each iteration,
    - ★ Step 1 takes $O(n)$.
    - ★ Step 2 takes $O(n)$.
  - Step 3 takes $O(n)$.
- M_2 decides A in time $O(n \lg n)$.
  - $O(n \lg n) = O(n) + O(\lg n) \times O(n) + O(n)$.

Models and Time Complexity

- Consider the following two-tape TM:
  M_3 = "On input string w:
  1. Scan tape 1 and reject if a `0` appears after a `1`.
  2. Scan tape 1 and copy the `0`'s onto tape 2.
  3. Scan tape 1 and cross a `0` on tape 2 for a `1` on tape 1.
  4. If all `0`'s are crossed off before reading all `1`'s, reject. If some `0`'s are left after reading all `1`'s, reject. Otherwise, accept."
- Analysis of M_3.
  - Each step takes $O(n)$.
- For the same language $A = \{\mathtt{0}^n\mathtt{1}^n : n \geq 0\}$.
  - The TM M_1 decides A in time $O(n^2)$, the TM M_2 decides A in time $O(n \lg n)$, and the two-tape M_3 decides A in time $O(n)$.
- In computability theory, all reasonable variants of TM's decide the same language (Church-Turing thesis).
- In complexity theory, different variants of TM's may decide the same in different time.

Complexity Relationship with Multitape TM's

Theorem 5

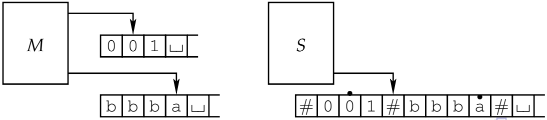
Let $t(n)$ be a function with $t(n) \geq n$. Every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape TM.

Proof.

We analyze the simulation of a k-tape TM M is by the TM S. Observe that each tape of M has length at most $t(n)$ (why?).
For each step of M, S has two passes:

- The first pass gathers information ($O(kt(n))$).
- The second pass updates information with at most k shifts ($O(k^2t(n))$).

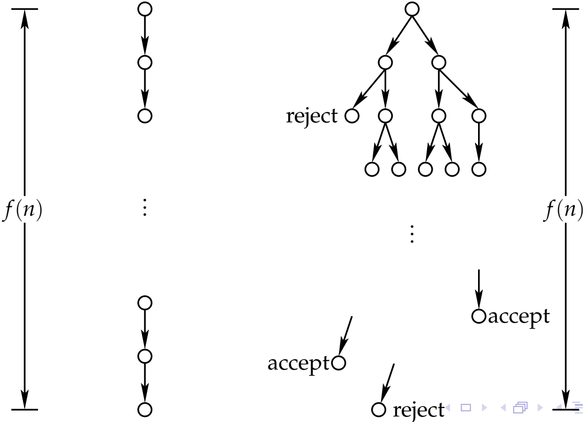
Hence S takes $O(n) + O(k^2t^2(n))$ ($= O(n) + O(t(n)) \times O(k^2t(n))$). Since $t(n) \geq n$, we have S runs in time $O(t^2(n))$ (k is independent of the input). $\square$



Time Complexity of Nondterministic TM's

Definition 6

Let N be a nondeterministic TM that is a decider. The running time of N is a function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of steps among any branch of N's computation on input of length n.



Complexity Relationship with NTM's

Theorem 7

Let $t(n)$ be a function with $t(n) \geq n$. Every $t(n)$ time single-tape NTM has an equivalent $2^{O(t(n))}$ time single-tape TM.

Proof.

Let N be an NTM running in time $t(n)$. Recall the simulation of N by a 3-tape TM D with the address tape alphabet $\Sigma_b = \{\mathtt{1}, \mathtt{2}, \ldots, \mathtt{b}\}$ (b is the maximal number of choices allowed in N).

Since N runs in time $t(n)$, the computation tree of N has $O(b^{t(n)})$ nodes.

For each node, D simulates it from the start configuration and thus takes time $O(t(n))$. Hence the simulation of N on the 3-tape D takes $2^{O(t(n))}(= O(t(n)) \times O(b^{t(n)}))$ time.

By Theorem 5, D can be simulated by a single-tape TM in time $(2^{O(t(n))})^2 = 2^{O(t(n))}$. □

The Class P

- It turns out that reasonable deterministic variants of TM's can be simulated by a TM with a polynomial time overhead.
  - multitape TM's, TM's with random access memory, etc.
- The polynomial time complexity class is rather robust.
  - That is, it remains the same with different computational models.

Definition 8

P is the class of languages decidable in polynomial time on a determinsitic single-tape TM. That is,

$$P = \bigcup_k TIME(n^k).$$

- We are interested in intrinsic characters of computation and hence ignore the difference among variants of TM's in this course.
- Solving a problem in time $O(n)$ and $O(n^{100})$ certainly makes lots of difference in practice.

The Class *NP*

Definition 9

A verifier for a language A is an algorithm V where

$$A = \{w : V \text{ accepts } \langle w, c \rangle \text{ for some } c\}.$$

c is a certificate or proof of membership in A. A polynomial time verifier runs in polynomial time in the length of w (not $\langle w, c \rangle$). A language A is polynomially verifiable if it has a polynomial time verifier.

- Note that a certificate has a length polynomial in $|w|$.
  - Otherwise, V cannot run in polynomial time in $|w|$.

Definition 10

NP is the class of languages that have polynomial time verifiers.

Hamiltonian Paths

- A Hamiltonian path in a directed graph G is a path that goes through every node exactly once. Consider

$$\begin{aligned} &HAMPATH = \\ &\qquad \{\langle G, s, t\rangle : \quad G \text{ is a directed graph with a Hamiltonian path} \\ &\qquad\qquad\qquad\qquad \text{from } s \text{ to } t\}. \end{aligned}$$

- $HAMPATH \in NP$.
  - Verifying whether c is a Hamiltonian path from s to t can be done in polynomial time.
  - A certificate for $\langle G, s, t\rangle \in HAMPATH$ is a Hamiltonian path from s to t.
- Finding a Hamiltonian path from s to t seems harder.

NP and NTM's

Theorem 11

A language is in NP if and only if it is decided by a nondeterministic polynomial time Turing machine.

Proof.

Let V be a verifier for a language A running in time n^k. Consider
$N =$ "On input w of length n:

1. Nondeterministically select string c of length $\leq n^k$.
2. Run V on $\langle w, c \rangle$.
3. If V accepts, accept; otherwise, reject."

Conversely, let the NTM N decide A and c the address of an accepting configuration in the computation tree of N. Consider
$V =$ "On input $\langle w, c \rangle$:

1. Simulate N on w from the start configuration by c.
2. If the configuration with address c is accepting, accept; otherwise, reject." $\square$

The Nondeterministic Time Complexity Class

Definition 12

NTIME (t (n)) = { L : L is a language decided by a O (t (n)) time NTM }.

Corollary 13

$$NP = \bigcup_k NTIME(n^k).$$

- Recall that class $TIME(t(n))$ and

$$P = \bigcup_k TIME(n^k).$$

The Class *coNP*

Definition 14

$coNP = \{L : \overline{L} \in NP\}$.

- $\overline{HAMPATH} \in coNP$ since $\overline{\overline{HAMPATH}} = HAMPATH \in NP$.
  - $\overline{HAMPATH}$ does not appear to be polynomial time verifiable.
  - What is a certificate showing there is no Hamiltonian path?
- We do not know if *coNP* is different from *NP*.
- Recall
  - *P* is the class of languages which membership can be decided quickly.
  - *NP* is the class of languages which membership can be verified quickly.
-
$$L \in P \text{ implies } L \in NP \text{ for every language } L.$$

P vs NP

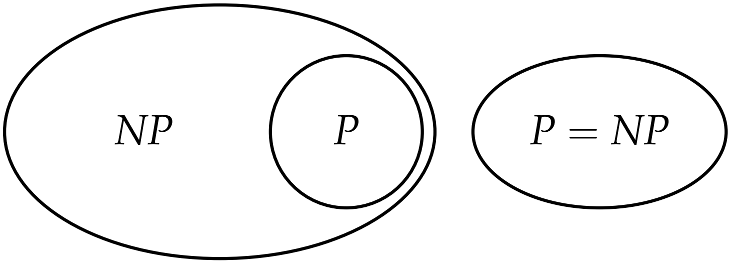


Figure: Possible Relation between P and NP

- To the best of our knowledge, we only know

$$NP \subseteq EXPTIME = \bigcup_k TIME(2^{n^k}). \qquad \text{(Theorem 7)}$$

- Particularly, we do no know if $P \stackrel{?}{=} NP$.

Satisfiability

- Let $\mathbb{B} = \{0, 1\}$ be the truth values.
- A Boolean variable takes values from $\mathbb{B}$.
- Recall the Boolean operations

$$
\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
0 \wedge 0 & = & 0 & 0 \vee 0 & = & 0 & & & \\
0 \wedge 1 & = & 0 & 0 \vee 1 & = & 1 & \overline{0} & = & 1 \\
1 \wedge 0 & = & 0 & 1 \vee 0 & = & 1 & \overline{1} & = & 0 \\
1 \wedge 1 & = & 1 & 1 \vee 1 & = & 1 & & &
\end{array}
$$

- A Boolean formula is an expression constructed from Boolean variables and opearations.
  - $\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$ is a Boolean formula.
- A Boolean formula is satisfiable if an assignments of 0's and 1's to Boolean variables makes the formula evaluate to 1.
  - ϕ is satisfiable by taking $\{x \mapsto 0, y \mapsto 1, z \mapsto 0\}$.

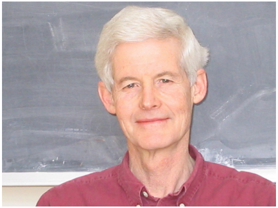
The Satisfiability Problem

- The satisfiability problem is to test whether a Boolean formula is satisfiable.
- Consider

$$SAT = \{\langle \phi \rangle : \phi \text{ is a satisfiable Boolean formula}\}.$$

Theorem 15 (Cook-Levin)

$SAT \in P$ if and only if $P = NP$.



Polynomial Time Reducibility

Definition 16

$f : \Sigma^* \to \Sigma^*$ is a <u>polynomial time computable function</u> if a polynomial time TM M halts with only $f(w)$ on its tape upon any input w.

Definition 17

A language A is <u>polynomial time mapping reducible</u> (<u>polynomial time reducible</u>, or <u>polynomial time many-one reducible</u>) to a language B (written $A \leq_P B$) if there is a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ that

$$w \in A \text{ if and only if } f(w) \in B \text{ for every } w.$$

f is called the <u>polynomial time reduction</u> of A to B.

- Recall the definitions of computable functions and mapping reducibility.

Properties about Polynomial Time Reducibility

Theorem 18

If $A \leq_P B$ and $B \in P$, $A \in P$.

Proof.

Let the TM M decide B and f a polynomial time reduction of A to B. Consider

$N =$ "On input w:

1. Compute $f(w)$.
2. Run M on $f(w)$."

Since the composition of two polynomials is again a polynomial, N runs in polynomial time. □

The $3SAT$ Problem

- A literal is a Boolean variable or its negation.
- A clause is a disjunction ($\vee$) of literals.
  - $x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4$ is a clause.
- A Boolean formula is in conjunctive normal form (or a CNF-formula) if it is a conjunction ($\wedge$) of clauses.
  - $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_2 \vee x_2 \vee \overline{x_5}) \wedge (x_4 \vee x_6)$ is a CNF-formula.
- In a satisfiable CNF-formula, each clause must contain at least one literal assigned to 1.
- A Boolean formula is a 3CNF-formula if it is a CNF-formula whose clauses have three literals.
  - $(x_1 \vee \overline{x_3} \vee x_4) \wedge (x_2 \vee x_2 \vee \overline{x_5}) \wedge (x_4 \vee x_5 \vee \overline{x_6})$ is a 3CNF-formula.
- Consider

$$3SAT = \{\langle \phi \rangle : \phi \text{ is a satisfiable 3CNF-formula}\}.$$

$3SAT \leq_P CLIQUE$

Theorem 19

$3SAT \leq_P CLIQUE$.

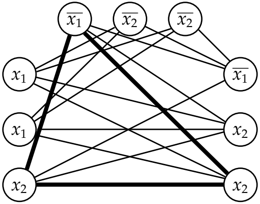
Proof.

Given a 3CNF-formula $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$, we would like to find a graph G and a number k such that $\langle \phi \rangle \in 3SAT$ if and only if $\langle G, k \rangle \in CLIQUE$. We need gadgets to simulate Boolean variables and clauses in ϕ.

- For each clause $a_i \vee b_i \vee c_i$, add three corresponding nodes to G.
  - G has $3k$ nodes.
- For each pair of nodes in G, add an edge except when
  - the pair of nodes correspond to literals in a clause.
  - the pair of nodes correspond to complementary literals.

We next show that ϕ is satisfiable if and only if G has a k-clique.

$3SAT \leq_P CLIQUE$



$(x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

Proof.

Suppose ϕ has a satisfying assignment. Each clause has at least one literal assigned to 1. We pick a node corresponding to true literal from each clause. Any pair of the chosen nodes do not belong to the same clause. Since a literal and its complement cannot be 1 simultaneously, any pair of the chosen nodes are not complementary. Hence there is an edge between any pair of the chosen nodes. We have a k-clique.
Conversely, suppose there is a k-clique. Since there is no edge between any two nodes in a clause, the k-clique must have one node from each of the k clauses. Moreover, there is no edge between complementary literals. Either a literal or its complement appears in the k-clique but not both. ϕ is satisfied by the assignment making literals in the clique true.
It is easy to see that G can be constructed from ϕ in polynomial time. $\square$

NP-Completeness

Definition 20

A language B is <u>NP-complete</u> if

- B is in NP; and
- every A in NP is polynomial time reducible to B.

Theorem 21

If B is NP-complete and $B \in P$, then $P = NP$.

Theorem 22

If $C \in NP$, B is NP-complete, and $B \leq_P C$, then C is NP-complete.

Proof.

Since B is NP-complete, there is a polynomial time reduction f of A to B for any $A \in NP$. Since $B \leq_P C$, there is a polynomial time reduction g of B to C. $g \circ f$ is a polynomial time reduction of A to C. $\square$

Cook-Levin Theorem

Theorem 23

SAT is NP-complete.

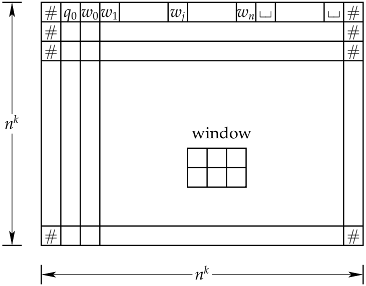
Proof.

For any Boolean formula ϕ, an NTM nondeterministically choose a truth assignment. It checks whether the assignment satisfies ϕ. If so, accept; otherwise, reject. Hence $SAT \in NP$.

Let $A \in NP$ and the NTM N decide A in n^k time. For any input w, a tableau for N on w is an $n^k \times n^k$ table whose rows are the configurations along a branch of the computation of N on w. A tableau of size $n^k \times n^k$ has $n^k \times n^k$ cells. We assume each configuration starts and ends with a # symbol. A tableau is accepting if any of its rows is an accepting configuration.

Each accepting tableau for N on w corresponds to an accepting computation of N on w. We therefore construct a Boolean formula ϕ such that ϕ is satisfiable if and only if there is an accepting tableau for N on w.

Cook-Levin Theorem



Proof (cont'd).

Let $C = Q \cup \Gamma \cup \{\#\}$ where Q and Γ are the states and the tape alphabet of N. For $1 \leq i, j \leq n^k$ and $s \in C$, the Boolean variable $x_{i,j,s}$ denotes the content of the cell $cell[i, j]$. That is, $x_{i,j,s}$ is 1 if and only if $cell[i, j] = s$. To force each cell to contain exactly one symbol from C, consider

$$\phi_{\text{cell}} = \bigwedge_{1 \leq i,j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{s,t \in C, s \neq t} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right].$$

Cook-Levin Theorem

Proof (cont'd).

To force the tableau to begin with the start configuration, consider

$$\begin{aligned}\phi_{\text{start}} &= x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ &\quad x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \cdots \wedge x_{1,n+2,w_n} \wedge \\ &\quad x_{1,n+3,\sqcup} \wedge \cdots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}.\end{aligned}$$

To force an accepting configuration to appear in the tableau, consider

$$\phi_{\text{accept}} = \bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_{\text{accept}}}.$$

To force the configuration at row i yields the configuration at row $i+1$, consider a window of 2×3 cells. For example, assume $\delta(q_1, \mathtt{a}) = \{(q_1, \mathtt{b}, R)\}$ and $\delta(q_1, \mathtt{b}) = \{(q_2, \mathtt{c}, L), (q_2, \mathtt{a}, R)\}$. The following windows are valid:

a	q_1	b
q_2	a	c

a	q_1	b
a	a	q_2

a	a	q_1
a	a	b

#	b	a
#	b	a

a	b	a
a	b	q_2

b	b	b
c	b	b

Cook-Levin Theorem

Proof.

Since C is finite, there are only a finite number of valid windows. For any window W $\begin{array}{c|c|c} c_1 & c_2 & c_3 \\ \hline c_4 & c_5 & c_6 \end{array}$, consider

$$\psi_W = x_{i,j-1,c_1} \wedge x_{i,j,c_2} \wedge x_{i,j+1,c_3} \wedge x_{i+1,j-1,c_4} \wedge x_{i+1,j,c_5} \wedge x_{i+1,j+1,c_6}$$

To force every window in the tableau to be valid, consider

$$\phi_{\text{move}} = \bigwedge_{1 \le i \le n^k, 1 \le j < n^k} \left(\bigvee_{W \text{ is a valid}} \psi_W \right).$$

Finally, consider the following Boolean formula:

$$\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}.$$

$|\phi_{\text{cell}}| = O(n^{2k})$, $|\phi_{\text{start}}| = O(n^k)$, $|\phi_{\text{accept}}| = O(n^{2k})$, and $|\phi_{\text{move}}| = O(n^{2k})$. Hence $|\phi| = O(n^{2k})$. Moreover, ϕ can be constructed from N in time polynomial in n. □

Corollary 24

3SAT is NP-complete.

Proof.

We convert the Boolean formula ϕ in the proof of Theorem 23 into a 3CNF-formula. We begin by converting ϕ into a CNF-formula.
Observe that the conjunction of CNF-formulae is again a CNF-formula. Note that ϕ_{cell}, ϕ_{start}, and ϕ_{accept} are already in CNF (why?). ϕ_{move} is of the following form:

$$\bigwedge_{1 \leq i \leq n^k, 1 \leq j < n^k} \left(\bigvee_{W \text{ is valid}} (l_1 \wedge l_2 \wedge l_3 \wedge l_4 \wedge l_5 \wedge l_6) \right)$$

By the law of distribution, ϕ_{move} can be converted into a CNF-formula. Note that the conversion may increase the size of ϕ_{move}. Yet the size is independent of $|w|$. Hence the size of the CNF-formula ϕ still polynomial in $|w|$.
To a clause of k literals into clauses of 3 literals, consider $l_1 \mapsto (l_1 \vee l_1 \vee l_1)$, $l_1 \vee l_2 \mapsto (l_1 \vee l_2 \vee l_2)$, and
$l_1 \vee l_2 \vee \cdots l_p \mapsto (l_1 \vee l_2 \vee z_1) \wedge (\overline{z_1} \vee l_3 \vee z_2) \wedge \cdots \wedge (\overline{z_{p-3}} \vee l_{p-1} \vee l_p)$. □

More *NP*-Complete Problems

- To find more *NP*-complete problems, we apply Theorem 22.
- Concretely, to show C is *NP*-complete, do
  - prove C is in *NP*; and
  - find a polynomial time reduction of an *NP*-complete problem (say, $3SAT$) to C.
- In Theorem 19, we have shown $3SAT \leq_P CLIQUE$. Therefore

Corollary 25

CLIQUE is NP-complete.

Space Complexity

Definition 26

Let M be a TM that halts on all inputs. The space complexity of M is $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of tape cells that M scans on any input of length n.
If the space complexity of M is $f(n)$, we say M runs in space $f(n)$.

Definition 27

If N is an NTM wherein all branches of its computation halts on all inputs. The space complexity of N is $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of tape cells that N scans on any branch of its computation for any input of length n.
If the space complexity of N is $f(n)$, we say N runs in space $f(n)$.

Space Complexity Classes

Definition 28

Let $f : \mathbb{N} \to \mathbb{R}^+$. The space complexity classes, $SPACE(f(n))$ and $NSPACE(f(n))$, are

$$\begin{aligned} SPACE(f(n)) &= \{L : L \text{ is decided by an } O(f(n)) \text{ space TM}\} \\ NSPACE(f(n)) &= \{L : L \text{ is decided by an } O(f(n)) \text{ space NTM}\} \end{aligned}$$

$SAT \in SPACE(n)$

Example 29

Give a TM that decides SAT in space $O(n)$.

Proof.

Consider

M_1 = "On input $\langle \phi \rangle$ where ϕ is a Boolean formula:

1. For each truth assignment to $x_1, x_2, \ldots, x_m$ of ϕ, do
   1. Evaluate ϕ on the truth assignment.
2. If ϕ ever eavluates to 1, accept; otherwise, reject."

M_1 runs in space $O(n)$ since it only needs to store the current truth assignment for m variables and $m \in O(n)$. □

Savitch's Theorem

Theorem 30 (Savitch)

For $f : \mathbb{N} \to \mathbb{R}^+$ with $f(n) \geq n$, $NSPACE(f(n)) \subseteq SPACE(f^2(n))$.

Proof.

Let N be an NTM deciding A in space $f(n)$. Assume N has a unique accepting configuration c_{accept} (how?). We construct a TM M deciding A in space $O(f^2(n))$. Let w be an input to N, c_1, c_2 configurations of N on w, and $t \in \mathbb{N}$. Consider $CANYIELD$ = "On input $c_1, c_2,$ and t:

1. If $t = 1$, test whether $c_1 = c_2$, or c_1 yields c_2 in N. If either succeeds, accept; otherwise, reject.
2. If $t > 1$, for each configuration c_m of N on w do
    1. Run $CANYIELD(c_1, c_m, \frac{t}{2})$.
    2. Run $CANYIELD(c_m, c_2, \frac{t}{2})$.
    3. If both accept, accept.
3. Reject."

Observe that $CANYIELD$ needs to store the step number, c_1, c_2, and t for recursion.

Savitch's Theorem

Proof (cont'd).

We select a constant d so that N has at most $2^{df(n)}$ configurations where $n = |w|$.
$M =$ "On input w:

1. Run $CANYIELD(c_{\text{start}}, c_{\text{accept}}, 2^{df(n)})$."

Since $t = 2^{df(n)}$, the depth of recusion is $O(\lg 2^{df(n)}) = O(f(n))$. Moreover, $CANYIELD$ can store its step number, c_1, c_2, t in space $O(f(n))$. Thus M runs in space $O(f(n) \times f(n)) = O(f^2(n))$.
A technical problem for M is to compute $f(n)$ in space $O(f(n))$. This can be avoided as follows. Instead of computing $f(n)$, M tries $f(n) = 1, 2, 3, \ldots$. For each $f(n) = i$, M calls $CANYIELD$ as before but also checks if N reaches a configuration of length $i + 1$ from c_{start}. If N reaches c_{accept}, M accepts as before. If N reaches a configuration of length $i + 1$ but fails to reach c_{accept}, M continues with $f(n) = i + 1$. Otherwise, all configurations of N have length $\leq f(n)$. N still fails to reach c_{accept} in $2^{df(n)}$ time. Hence M rejects. □

The Class *PSPACE*

Definition 31

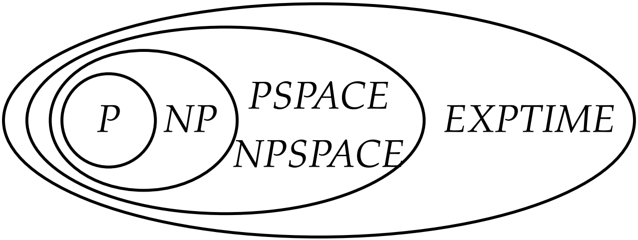
PSPACE is the class of languages decidable by TM's in polynomial space. That is,

$$PSPACE = \bigcup_k SPACE(n^k).$$

- Consider the class of langauges decidable by NTM's in polynomial space $NPSPACE = \bigcup_k NSPACE(n^k)$.
- By Savitch's Theorem, $NSPACE(n^k) \subseteq SPACE(n^{2k})$. Clearly, $SPACE(n^k) \subseteq NSPACE(n^k)$. Hence $NPSPACE = PSPACE$.
- Recall $SAT \in SPACE(n)$ and $ALL_{\text{NFA}} \in coNSPACE(n)$. By Savitch's Theorem, $\overline{ALL_{\text{NFA}}} \in NSPACE(n) \subseteq SPACE(n^2)$. Hence $ALL_{\text{NFA}} \in SPACE(n^2)$ (why?). $SAT, ALL_{\text{NFA}} \in PSPACE$.

P, NP, $PSPACE$, and $EXPTIME$

- $P \subseteq PSPACE$
  - A TM running in time $t(n)$ uses space $t(n)$ (provided $t(n) \geq n$).
- Similarly, $NP \subseteq NPSPACE$ and thus $NP \subseteq PSPACE$.
- $PSPACE \subseteq EXPTIME = \cup_k TIME(2^{n^k})$
  - A TM running in space $f(n)$ has at most $f(n)2^{O(f(n))}$ different configurations (provided $f(n) \geq n$).
    - ★ A configuration contains the current state, the location of tape head, and the tape contents.
- In summary, $P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME$.
  - We will show $P \neq EXPTIME$.



PSPACE-Completeness

Definition 32

A language B is <u>*PSPACE*-complete</u> if it satisfies

- $B \in PSPACE$; and
- $A \leq_P B$ for every $A \in PSPACE$.

If B only satisfies the second condition, we say it is <u>*PSPACE*-hard</u>.

- We do not define "polynomial space reduction" nor use it.
- Intuitively, a complete problem is most difficult in the class.
- If we can solve a complete problem, we can solve all problems in the same class easily.
- Polynomial space reduction is not easy at all.
  - Recall $SAT \in SPACE(n)$.

TQBF

- Recall the universal quantifier $\forall$ and the existential quantifier $\exists$.
- When we use quantifiers, we should specify a universe.
  - $\forall x \exists y[x < y \wedge y < x + 1]$ is false if $\mathbb{Z}$ is the universe.
  - $\forall x \exists y[x < y \wedge y < x + 1]$ is true if is the universe.
- A quantified Boolean formula is a quantified Boolean formula over the universe $\mathbb{B}$.
- Any formula with quantifiers can be converted to a formula begins with quantifiers.
  - $\forall x[x \geq 0 \implies \exists y[y^2 = x]]$ is equivalent to $\forall x \exists y[x \geq 0 \implies y^2 = x]$.
  - This is called prenex normal form.
- We always consider formulae in prenex normal form.
- If all variables are quantified in a formula, we say the formula is fully quantified (or a sentence).
- Consider

$$TQBF = \{\langle \phi \rangle : \phi \text{ is a true fully quantified Boolean formula}\}.$$

TQBF is *PSPACE*-Complete

Theorem 33

TQBF is PSPACE-complete.

Proof.

We first show $TQBF \in PSPACE$. Consider
$T =$ "On input $\langle \phi \rangle$ where ϕ is a fully quantified Boolean formula:

1. If ϕ has no quantifier, it is a Boolean formula without variables. If ϕ evaluates to 1, accept; otherwise, reject.
2. If ϕ is $\exists x \psi$, call T recursively on $\psi[x \mapsto 0]$ and $\psi[x \mapsto 1]$. If T accepts either, accept; otherwise, reject.
3. If ϕ is $\forall x \psi$, call T recursively on $\psi[x \mapsto 0]$ and $\psi[x \mapsto 1]$. If T accepts both, accept; otherwise, reject.

The depth of recursion is the number of variables. At each level, T needs to store the value of one variable. Hence T runs in space $O(n)$.

TQBF is *PSPACE*-Complete

Proof (cont'd).

Let M be a TM deciding A in space n^k. For any string w, we construct a quantified Boolean formula ϕ such that M accepts w if and only if ϕ is true. More precisely, let c_1, c_2 be collections of variables representing two configurations, and $t > 0$, we construct a formula $\phi_{c_1,c_2,t}$ such that $\phi_{c_1,c_2,t} \wedge c_1 = \mathsf{c}_1 \wedge c_2 = \mathsf{c}_2$ is true if and only if M can go from the configuration c_1 to the configuration c_2 in $\leq t$ steps.

To construct $\phi_{c_1,c_2,1}$, we check if $c_1 = c_2$, or the configuration represented by c_1 yields the configuration represented by c_2 in M. We use the technique in the proof of Cook-Levin Theorem. That is, we construct a Boolean formula stating that all windows on the rows c_1, c_2 are valid. Observe that $|\phi_{c_1,c_2,1}| \in O(n^k)$. For $t > 1$, let

$$\phi_{c_1,c_2,t} = \exists m \forall c_3 \forall c_4 \left[((c_3 = c_1 \wedge c_4 = m) \vee (c_3 = m \wedge c_4 = c_2)) \implies \phi_{c_3,c_4,\frac{t}{2}} \right]$$

Note that $|\phi_{c_1,c_2,t}| = \gamma n^k + |\phi_{c_3,c_4,\frac{t}{2}}|$ for some constant γ.

Assume M has a unique accepting configuration c_{accept}. Choose a constant d so that M has at most 2^{dn^k} configurations on w. Then $\phi_{c_{\text{start}},c_{\text{accept}},2^{dn^k}}$ is true if and only if M accepts w. Moreover, the depth of recursion is $O(\lg 2^{dn^k}) = O(n^k)$. Each level increases the size of $\phi_{c_1,c_2,t}$ by $O(n^k)$. Hence $|\phi_{c_{\text{start}},c_{\text{accept}},2^{dn^k}}| \in O(n^{2k})$. □

TQBF is *PSPACE*-Complete

- Do we really need quantified Boolean formulae?
- For $t > 1$, consider

$$\phi_{c_1,c_2,t} = \exists m[\phi_{c_1,m,\frac{t}{2}} \wedge \phi_{m,c_2,\frac{t}{2}}].$$

- Recall that $\phi_{c_1,c_2,1}$ is an unquantified Boolean formula.
- We can construct an unquantified formula $\Phi_{c_1,c_2,t}$ such that $\langle \phi_{c_1,c_2,t} \rangle \in TQBF$ if and only if $\langle \Phi_{c_1,c_2,t} \rangle \in SAT$.
- Hence $PSPACE \subseteq NP$?!
- Note that $|\phi_{c_1,c_2,t}| \geq 2|\phi_{c_1,c_2,\frac{t}{2}}|$. $|\phi_{c_1,c_2,2^{dn^k}}|$ is in fact of size $O(2^{n^k})$.
- Quantifiers allow us to "reuse" subformula!

TQBF is *PSPACE*-Complete

- Do we really need quantified Boolean formulae?
- For $t > 1$, consider

$$\phi_{c_1,c_2,t} = \exists m[\phi_{c_1,m,\frac{t}{2}} \wedge \phi_{m,c_2,\frac{t}{2}}].$$

- Recall that $\phi_{c_1,c_2,1}$ is an unquantified Boolean formula.
- We can construct an unquantified formula $\Phi_{c_1,c_2,t}$ such that $\langle \phi_{c_1,c_2,t} \rangle \in TQBF$ if and only if $\langle \Phi_{c_1,c_2,t} \rangle \in SAT$.
- Hence $PSPACE \subseteq NP$?!
- Note that $|\phi_{c_1,c_2,t}| \geq 2|\phi_{c_1,c_2,\frac{t}{2}}|$. $|\phi_{c_1,c_2,2^{dn^k}}|$ is in fact of size $O(2^{n^k})$.
- Quantifiers allow us to "reuse" subformula!

TM's with Sublinear Space

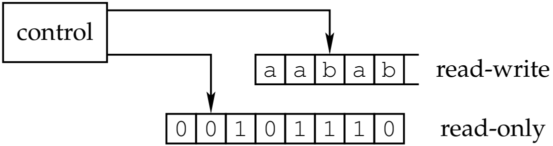


Figure: Schematics for TM's using Sublinear Space

- For sublinear space, we consider TM's with two tapes.
  - a read-only input tape containing the input string; and
  - a read-write work tape.
- The input head cannot move outside the portion of the tape containing the input.
- The cells scanned on the work tape contribute to the space complexity.

Space Complexity Classes L and NL

Definition 34

$\underline{L}$ ($= SPACE(\log n)$) is the class of languages decidable by a TM in logarithmic space.
$\underline{NL}$ ($= NSPACE(\log n)$) is the class of languages decidable by an NTM in logarithmic space.

Example 35

$A = \{\mathtt{0}^k\mathtt{1}^k : k \geq 0\} \in L$.

Proof.

Consider
$M =$ "On input w:

1. Check if w is of the form $\mathtt{0}^*\mathtt{1}^*$. If not, reject.
2. Count the number of $\mathtt{0}$'s and $\mathtt{1}$'s on the work tape.
3. If they are equal, accept; otherwise, reject." $\square$

PATH is in *NL*

Example 36

Recall $PATH = \{\langle G, s, t\rangle : G$ is a directed graph with a path from s to $t\}$. Show $PATH \in NL$.

Proof.

Consider
$N =$ "On input $\langle G, s, t\rangle$ where G is a directed graph with nodes s and t:

1. Repeat m times (m is the number of nodes in G)
   1. Nondeterministically select the next node for the path. If the next node is t, accept.
2. Reject.

N only needs to store the current node on the work tape. Hence N runs in space $O(\lg n)$. □

- We do not know if $PATH \in L$.

Configurations of TM's with Sublinear Space

Definition 37

Let M be a TM with a separate read-only input tape and w an input string. A configuration of M on w consists of a state, the contents of work tape, and locations of the two tape heads.

- Note that the input w is no longer a part of the configuration.
- If M runs in space $f(n)$ and $|w| = n$, the number of configurations of M on w is $n2^{O(f(n))}$.
  - Suppose M has q states and g tape symbols. The number of configurations is at most $qnf(n)g^{f(n)} \in n2^{O(f(n))}$.
- Note that when $f(n) \geq \lg n$, $n2^{O(f(n))} = 2^{O(f(n))}$.

Savitch's Theorem Revisited

- Recall that we assume $f(n) \geq n$ in the theorem.
- We can in fact relax the assumption to $f(n) \geq \lg n$.
- The proof is identical except that we are simulating an NTM N with a read-only input tape.
- When $f(n) \geq \lg n$, the depth of recursion is $\lg(n2^{O(f(n))}) = \lg n + O(f(n)) = O(f(n))$. At each level, $\lg(n2^{O(f(n))}) = O(f(n))$ space is needed.
- Hence $NSPACE(f(n)) \subseteq SPACE(f^2(n))$ when $f(n) \geq \lg n$.

Log Space Reducibility

Definition 38

A log space transducer is a TM with a read-only input tape, a write-only output tape, and a read-write work tape. The work tape may contain $O(\lg n)$ symbols.

Definition 39

$f : \Sigma^* \to \Sigma^*$ is a log space computable function if there is a log space transducer that halts with $f(w)$ in its work tape on every input w.

Definition 40

A language A is log space reducible to a language B (written $A \leq_L B$) if there is a log space computable function f such that $w \in A$ if and only if $f(w) \in B$ for every w.

Properties about Log Space Reducibility

Theorem 41

If $A \leq_L B$ and $B \in L$, $A \in L$.

Proof.

Let a TM M_B decide B in space $O(\lg n)$. Consider

$M_A =$ "On input w:

1. Compute the first symbol of $f(w)$.
2. Simulate M_B on the current symbol.
3. If M_B ever changes its input head, compute the symbol of $f(w)$ at the new location.
   - More precisely, restart the computation of $f(w)$ and ignore all symbols of $f(w)$ except the one needed by M_B.
4. If M_B accepts, accepts; otherwise, reject.

$\square$

- Can we write down $f(w)$ on M_B's work tape?
  - No. $f(w)$ may need more than logarithmic space.

Properties about Log Space Reducibility

Theorem 41

If $A \leq_L B$ and $B \in L$, $A \in L$.

Proof.

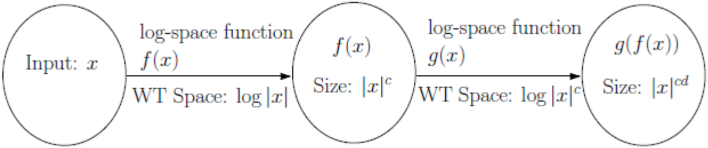
Let a TM M_B decide B in space $O(\lg n)$. Consider
M_A = "On input w:

1. Compute the first symbol of $f(w)$.
2. Simulate M_B on the current symbol.
3. If M_B ever changes its input head, compute the symbol of $f(w)$ at the new location.
    - More precisely, restart the computation of $f(w)$ and ignore all symbols of $f(w)$ except the one needed by M_B.
4. If M_B accepts, accepts; otherwise, reject. □

- Can we write down $f(w)$ on M_B's work tape?
    - No. $f(w)$ may need more than logarithmic space.

Properties about Log Space Reducibility

- We know that polynomial-time reductions are transitive:
  If $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$
- We also crucially used the following similar property:
  If $A \leq_p B$ and $B \in P$, then $A \in P$
  If $A \leq_p B$ and $B \in NP$, then $A \in NP$
- Do we have similar results under $\leq_L$?
- Difficulty:



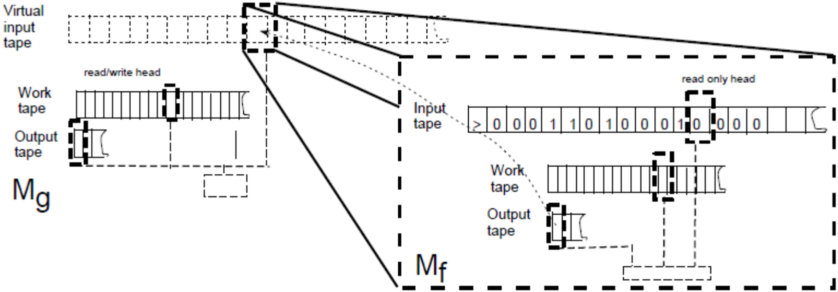
- Total space used $O(\log |x| + \log |x|^c) = O(\log |x|)$. Problem?
- We have to store intermediate result $f(x)$ of size $|x|^c$.

Transitivity of $\leq_L$

Goal: To compute the string $g(f(x))$, given x

- Imagine that we have computed $f(x)$, and its on Tape 1
- The tape-head for Tape 1 is at the start position.
- Now, given this imaginary input string, start computing $g(f(x))$ on Tape 2, just like before
- We know that the work tape Tape 2 needs $\log |f(x)|$ space
- At each step:
  - Read one bit of $f(x)$ from Tape 1 from tape-head position
  - Read one bit of work-tape from tape-head position
  - Move Tape 1, Tape 2 heads by transition function
  - Write one bit on Tape 2, maybe write one bit on Output tape
- Read one bit of $f(x)$ from Tape 1 from tape-head position
  - Don't have $f(x)$ lying around on the imaginary Tape 1
  - Instead, store position of Tape 1 head: $O(\log |f(x)|)$ space
  - Need to read $f(x)_i$: compute using $\log |x|$ space
  - Increment or decrement the pointer for Tape 1 head

Transitivity of $\leq_L$



NL-Completeness

Definition 42

A language B is NL-complete if

- $B \in NL$; and
- $A \leq_L B$ for every $A \in NL$.

- Note that we require $A \leq_L B$ instead of $A \leq_P B$.
- We will show $NL \subseteq P$ (Corollary 46).
- Hence every two problems in NL (except $\emptyset$ and Σ^*) are polynomial time reducible to each other (why?).

Corollary 43

If any NL-complete language is in L, then L = NL.

NL-Completeness

Theorem 44

PATH is NL-complete.

Proof.

Let an NTM M decide A in $O(\lg n)$ space. We assume M has a unique accepting configuration. Given w, we construct $\langle G, s, t\rangle$ in log space such that M accepts w if and only if G has a path from s to t.

Nodes of G are configurations of M on w. For configurations c_1 and c_2, the edge (c_1, c_2) is in G if c_1 yields c_2 in M. s and t are the start and accepting configurations of M on w respectively.

Clearly, M accepts w if and only if G has a path from s to t. It remains to show that G can be computed by a log space transducer. Observe that a configuration of M on w can be represented in $c \lg n$ space for some c. The transducer simply enumerates all string of legnth $c \lg n$ and outputs those that are configurations of M on w. The edges (c_1, c_2)'s are computed similarly. The transducer only needs to read the tape contents under the head locations in c_1 to decide whether c_1 yields c_2 in M. □

$NL \subseteq P$

Corollary 45

$NL \subseteq P$.

Proof.

A TM using space $f(n)$ has at most $n2^{O(f(n))}$ configurations and hence runs in time $n2^{O(f(n))}$. A log space transducer therefore runs in polynomial time. Hence any problem in NL is polynomial time reducible to $PATH$. The result follows by $PATH \in P$. $\square$

- The polynomial time reduction in the proof of Theorem 34 can be computed in log space.
- Hence $TQBF$ is $PSPACE$-complete with respect to log space reducibility.

$NL = coNL$

Theorem 46 (Immerman + Szelepcsényi)

$NL = coNL$.

Proof.

We will give an NTM M deciding $\overline{PATH}$ in space $O(\lg n)$. Hence $\overline{PATH} \in NL$. Recall that $PATH$ is NL-complete. For any $A \in NL$, we have $A \leq_L PATH$. Hence $\overline{A} \leq_L \overline{PATH}$. Since $\overline{PATH} \in NL$, $\overline{A} \in NL$. That is, $\overline{\overline{A}} = A \in coNL$. We have $NL \subseteq coNL$. For any $B \in coNL$, we have $\overline{B} \in NL$. Hence $\overline{B} \leq_L PATH$. Thus $B = \overline{\overline{B}} \leq_L \overline{PATH}$. Since $\overline{PATH} \in NL$, we have $B \in NL$. We have $coNL \subseteq NL$.

$NL = coNL$

Proof (cont'd).

[H] On $\langle G, s, t\rangle$ $c_0 = 1$ G has m nodes $i = 0, \ldots, m-1$ $c_{i+1} = 1$ *c_{i+1} counts the nodes reached from s in $\leq i+1$ steps node $v \neq s$ in G $d = 0$ *d recounts the nodes reached from s in $\leq i$ steps node u in G Nondeterministically **continue** Nondeterministically follow a path of length $\leq i$ from s Reject if the path does not end at u $d = d + 1$ (u, v) is an edge in G $c_{i+1} = c_{i+1} + 1$ **break** $d \neq c_i$Reject *check if the result is correct c_m = number of nodes reached from s

$NL = coNL$

Proof (cont'd).

[H] $d = 0$ *d recounts the nodes reached from s node u in G Nondeterministically **continue** Nondeterministically follow a path of length $\leq m$ from s Reject if the path does not end at u $u = t$Reject *do not count t $d = d + 1$ $d \neq c_m$Reject Accept The NTM M counts the nodes reached from s in the first phrase. The variable c_i is the number of nodes reached from s in $\leq i$ steps. Initially, $c_0 = 1$. To compute c_{i+1} from c_i, M goes through each node $v \neq s$ in G. For each v, M tries to find all nodes reached from s in $\leq i$ steps. For each such node u, M increments d. It also increments c_{i+1} if u points to v. If $d = c_i$, M has found all node reached from s in $\leq i$ steps. Hence c_{i+1} is correct. M proceeds to compute c_{i+2}.

At the second phrase, M counts nodes reached from s but excluding t. If s reaches the same set of nodes, t is not reachable from s. M accepts.

M needs to store u, v, c_i, c_{i+1}, d, i and a pointer to the head of a path. M runs in $O(\lg n)$ space. □

L, NL, P, and $PSPACE$

- The relationship between different complexity classes now becomes

$$L \subseteq NL = coNL \subseteq P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME$$

- We will prove $NL \subsetneq PSPACE$ in the next chapter.
- Hence at least on inclusion is propcer.
  - But we do not know which one.