Computational Complexity

Lecture 6
NL-Completeness and NL=co-NL
Story, so far
Story, so far

Time/Space Hierarchies
Story, so far

- Time/Space Hierarchies
- Relations across complexity measures
Story, so far

- Time/Space Hierarchies
- Relations across complexity measures
- SAT is NP-complete, TQBF is PSPACE-complete
Story, so far

- Time/Space Hierarchies
- Relations across complexity measures
- SAT is NP-complete, TQBF is PSPACE-complete
- Today
Story, so far

- Time/Space Hierarchies
- Relations across complexity measures
- SAT is NP-complete, TQBF is PSPACE-complete
- Today
- Log-space reductions
Story, so far

- Time/Space Hierarchies
- Relations across complexity measures
- SAT is NP-complete, TQBF is PSPACE-complete

Today

- Log-space reductions
- An NL-complete language: PATH
Story, so far

- Time/Space Hierarchies
- Relations across complexity measures
- SAT is NP-complete, TQBF is PSPACE-complete
- Today
  - Log-space reductions
  - An NL-complete language: PATH
- NSPACE = co-NSPACE (one less kind to worry about!)
NL-completeness
NL-completeness

For any two (non-trivial) languages $L_1, L_2$ in P, $L_2 \leq_p L_1$
NL-completeness

- For any two (non-trivial) languages $L_1, L_2$ in $P$, $L_2 \leq_P L_1$

- So if $X \subseteq P$, all languages in $X$ are $X$-complete (w.r.t $\leq_P$)
NL-completeness

- For any two (non-trivial) languages $L_1, L_2$ in $P$, $L_2 \leq_P L_1$
- So if $X \subseteq P$, all languages in $X$ are $X$-complete (w.r.t $\leq_P$)
- Need a tighter notion of reduction to capture “(almost) as hard as it gets” within $X$
Log-Space Reduction
Log-Space Reduction

Many-one reduction: \( L_2 \leq_L L_1 \) if there is a TM, \( M \) which maps its input \( x \) to \( f(x) \) such that
Log-Space Reduction

Many-one reduction: $L_2 \leq_L L_1$ if there is a TM, $M$ which maps its input $x$ to $f(x)$ such that

$x \in L_2 \Rightarrow f(x) \in L_1$ and $x \notin L_2 \Rightarrow f(x) \notin L_1$
Log-Space Reduction

Many-one reduction: $L_2 \leq L_1$ if there is a TM, $M$ which maps its input $x$ to $f(x)$ such that

- $x \in L_2 \Rightarrow f(x) \in L_1$ and $x \notin L_2 \Rightarrow f(x) \notin L_1$

- $M$ uses only $O(\log|x|)$ work-tape
Log-Space Reduction

Many-one reduction: \( L_2 \leq_L L_1 \) if there is a TM, \( M \) which maps its input \( x \) to \( f(x) \) such that

\( x \in L_2 \Rightarrow f(x) \in L_1 \) and \( x \notin L_2 \Rightarrow f(x) \notin L_1 \)

\( M \) uses only \( O(\log|x|) \) work-tape

Is allowed to have a write-only output tape, because \( |f(x)| \) may be \( \text{poly}(|x|) \)
Log-Space Reduction

Many-one reduction: $L_2 \leq_L L_1$ if there is a TM, $M$ which maps its input $x$ to $f(x)$ such that

- $x \in L_2 \Rightarrow f(x) \in L_1$ and $x \notin L_2 \Rightarrow f(x) \notin L_1$

- $M$ uses only $O(\log|x|)$ work-tape

- Is allowed to have a write-only output tape, because $|f(x)|$ may be $\text{poly}(|x|)$

- Equivalently: $f$ “implicitly computable” in log-space
Log-Space Reduction

Many-one reduction: $L_2 \leq_L L_1$ if there is a TM, $M$ which maps its input $x$ to $f(x)$ such that

- $x \in L_2 \Rightarrow f(x) \in L_1$ and $x \notin L_2 \Rightarrow f(x) \notin L_1$

- $M$ uses only $O(\log|\times|)$ work-tape

  Is allowed to have a write-only output tape, because $|f(x)|$ may be poly($|\times|$)

Equivalently: $f$ “implicitly computable” in log-space

A log-space machine $M'$ to output the bit $f_i(x)$ on input $(x,i)$
Log-Space Reduction

Many-one reduction: \( L_2 \leq_L L_1 \) if there is a TM, \( M \) which maps its input \( x \) to \( f(x) \) such that

\( x \in L_2 \Rightarrow f(x) \in L_1 \) and \( x \notin L_2 \Rightarrow f(x) \notin L_1 \)

\( M \) uses only \( O(\log|x|) \) work-tape

Is allowed to have a write-only output tape, because \( |f(x)| \) may be \( \text{poly}(|x|) \)

Equivalently: \( f \) “implicitly computable” in log-space

A log-space machine \( M' \) to output the bit \( f_i(x) \) on input \( (x,i) \)

\( M' \) from \( M \): to keep a counter and output only the \( i^{th} \) bit
Log-Space Reduction

Many-one reduction: $L_2 \leq_L L_1$ if there is a TM, $M$ which maps its input $x$ to $f(x)$ such that

- $x \in L_2$ ⇒ $f(x) \in L_1$ and $x \notin L_2$ ⇒ $f(x) \notin L_1$
- $M$ uses only $O(\log|x|)$ work-tape

Is allowed to have a write-only output tape, because $|f(x)|$ may be poly($|x|$)

Equivalently: $f$ “implicitly computable” in log-space

A log-space machine $M'$ to output the bit $f_i(x)$ on input $(x, i)$

$M'$ from $M$: to keep a counter and output only the $i^{th}$ bit

$M$ from $M'$: keep a counter and repeatedly call $M$ on each $i$
Log-Space Reduction
Log-Space Reduction

Log-space reductions “compose”: $L_2 \leq_L L_1 \leq_L L_0 \Rightarrow L_2 \leq_L L_0$
Log-Space Reduction

- Log-space reductions "compose": $L_2 \leq_L L_1 \leq_L L_0 \Rightarrow L_2 \leq_L L_0$

- Given $M_{2-1}$ and $M_{1-0}$ build $M_{2-0}$:
Log-Space Reduction

Log-space reductions “compose”: $L_2 \leq_L L_1 \leq_L L_0 \Rightarrow L_2 \leq_L L_0$

Given $M_{2-1}$ and $M_{1-0}$ build $M_{2-0}$:

Start running $M_{1-0}$ without input. When it wants to read $i^{th}$ bit of input, run $M_{2-1}$ (with a counter) to get the $i^{th}$ bit of its output
Log-Space Reduction

- Log-space reductions “compose”: $L_2 \leq_L L_1 \leq_L L_0 \Rightarrow L_2 \leq_L L_0$

- Given $M_{2-1}$ and $M_{1-0}$ build $M_{2-0}$:
  - Start running $M_{1-0}$ without input. When it wants to read $i^{th}$ bit of input, run $M_{2-1}$ (with a counter) to get the $i^{th}$ bit of its output
  - Space needed: $O(\log(|f(x)|) + \log(|x|)) = O(\log(|x|))$, because $|f(x)|$ is poly($|x|$)
Log-Space Reduction

- Log-space reductions “compose”: $L_2 \leq_L L_1 \leq_L L_0 \Rightarrow L_2 \leq_L L_0$

- Given $M_{2-1}$ and $M_{1-0}$ build $M_{2-0}$:
  - Start running $M_{1-0}$ without input. When it wants to read $i^{th}$ bit of input, run $M_{2-1}$ (with a counter) to get the $i^{th}$ bit of its output.
  - Space needed: $O(\log(|f(x)|) + \log(|x|)) = O(\log(|x|))$, because $|f(x)|$ is poly($|x|$).

- Similarly, $L$ (the class of problems decidable in log-space) is downward closed under log-space reductions.
Log-Space Reduction

Log-space reductions “compose”: $L_2 \leq_L L_1 \leq_L L_0 \Rightarrow L_2 \leq_L L_0$

Given $M_{2-1}$ and $M_{1-0}$ build $M_{2-0}$:

Start running $M_{1-0}$ without input. When it wants to read $i^{th}$ bit of input, run $M_{2-1}$ (with a counter) to get the $i^{th}$ bit of its output

Space needed: $O(\log(|f(x)|) + \log(|x|)) = O(\log(|x|))$, because $|f(x)|$ is poly($|x|$)

Similarly, $L$ (the class of problems decidable in log-space) is downward closed under log-space reductions

$L_2 \leq_L L_1 \in L \Rightarrow L_2 \in L$
NL-completeness
NL-completeness

L₀ is NL-Hard if for all L₁ in NL, L₁ ≤L L₀
**NL-completeness**

- \( L_0 \) is NL-Hard if for all \( L_1 \) in NL, \( L_1 \leq_L L_0 \)

- \( L_0 \) is NL-complete if it is NL-hard and is in NL
NL-completeness

- $L_0$ is NL-Hard if for all $L_1$ in NL, $L_1 \leq_L L_0$
- $L_0$ is NL-complete if it is NL-hard and is in NL
- Can construct trivial NL-complete language
NL-completeness

- $L_0$ is NL-Hard if for all $L_1$ in NL, $L_1 \leq_L L_0$
- $L_0$ is NL-complete if it is NL-hard and is in NL
- Can construct trivial NL-complete language
  
  $$\{ (M,x,1^n,1^s) \mid \exists w, |w| < n, \text{M accepts (x};w) \text{ in space log}(s) \} \text{ (where M takes w in a read-once tape)}$$
**NL-completeness**

- $L_0$ is NL-Hard if for all $L_1$ in NL, $L_1 \leq_L L_0$
- $L_0$ is NL-complete if it is NL-hard and is in NL
- Can construct trivial NL-complete language
  
  $\{ (M,x,1^n,1^s) | \exists w, |w|<n, M \text{ accepts } (x;w) \text{ in space } \log(s) \}$ (where $M$ takes $w$ in a read-once tape)

- Interesting NLC language: PATH
Directed Path
Directed Path

PATH = \{(G,s,t) \mid G \text{ a directed graph with a path from } s \text{ to } t\}
Directed Path

PATH = \{(G,s,t) \mid G \text{ a directed graph with a path from } s \text{ to } t\}

G using some representation, of size say, $n^2$ ($n=\#vertices$)
Directed Path

\( \text{PATH} = \{(G,s,t) \mid G \text{ a directed graph with a path from } s \text{ to } t\} \)

- \( G \) using some representation, of size say, \( n^2 \) (\( n = \#\text{vertices} \))

- Such that, if two vertices \( x,y \) on work-tape, can check for edge \((x,y)\)
Directed Path

PATH = \{(G,s,t) \mid G \text{ a directed graph with a path from } s \text{ to } t\}

- G using some representation, of size say, $n^2$ ($n=\#\text{vertices}$)
- Such that, if two vertices $x,y$ on work-tape, can check for edge $(x,y)$

PATH in NL
Directed Path

PATH = \{ (G, s, t) \mid G \text{ a directed graph with a path from } s \text{ to } t \} 

G using some representation, of size say, \( n^2 \) (\( n = \# \text{vertices} \))

Such that, if two vertices \( x, y \) on work-tape, can check for edge \( (x, y) \)

PATH in NL

Certificate \( w \) is the path (poly(n) long certificate)
Directed Path

\[ \text{PATH} = \{ (G, s, t) \mid G \text{ a directed graph with a path from } s \text{ to } t \} \]

\[ G \text{ using some representation, of size say, } n^2 (n=\#\text{vertices}) \]

\[ \text{Such that, if two vertices } x, y \text{ on work-tape, can check for edge } (x, y) \]

\[ \text{PATH in NL} \]

\[ \text{Certificate } w \text{ is the path (poly(n) long certificate)} \]

\[ \text{Need to verify adjacent vertices are connected: need keep only two vertices on the work-tape at a time} \]
Directed Path

PATH = \{(G,s,t) \mid G \text{ a directed graph with a path from } s \text{ to } t\}

G using some representation, of size say, $n^2$ ($n=\#\text{vertices}$)

Such that, if two vertices $x,y$ on work-tape, can check for edge $(x,y)$

PATH in NL

Certificate $w$ is the path (poly$(n)$ long certificate)

Need to verify adjacent vertices are connected: need keep only two vertices on the work-tape at a time

Note: $w$ is scanned only once
Seen PATH before?
Seen PATH before?

In proving $\text{NSPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))})$ (e.g. NL $\subseteq$ P)
Seen PATH before?

In proving $\text{NSPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))})$ (e.g. $\text{NL} \subseteq \text{P}$)

- Every problem in $\text{NL}$ Karp reduces to $\text{PATH}$
Seen PATH before?

- In proving $\text{NSPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))})$ (e.g. $\text{NL} \subseteq \text{P}$)
- Every problem in NL Karp reduces to PATH
- $\text{PATH} \in \text{P}$
Seen PATH before?

- In proving NSPACE(S(n)) ⊆ DTIME(2^{O(S(n))}) (e.g. NL ⊆ P)
- Every problem in NL Karp reduces to PATH
- PATH ∈ P
- In Savitch's theorem
Seen PATH before?

- In proving NSPACE(S(n)) ≤ DTIME(2^{O(S(n))}) (e.g. NL ⊆ P)
- Every problem in NL Karp reduces to PATH
- PATH ⊆ P
- In Savitch’s theorem
  - PATH ⊆ DSPACE(log^2(n))
PATH is NL-complete
PATH is NL-complete

Log-space reducing any NL language $L_1$ to PATH
PATH is NL-complete

Log-space reducing any NL language $L_1$ to PATH

Given input $x$, output $(G,s,t)$ where $G$ is the configuration graph $G(M,x)$, where $M$ is the NTM accepting $L_1$, and $s,t$ are start, accept configurations
PATH is NL-complete

- Log-space reducing any NL language $L_1$ to PATH

- Given input $x$, output $(G,s,t)$ where $G$ is the configuration graph $G(M,x)$, where $M$ is the NTM accepting $L_1$, and $s,t$ are start, accept configurations

- Outputting $G$: Cycle through all pairs of configurations, checking if there is an edge between them, outputting 0 or 1 in the adjacency matrix
PATH is NL-complete

Log-space reducing any NL language $L_1$ to PATH

Given input $x$, output $(G,s,t)$ where $G$ is the configuration graph $G(M,x)$, where $M$ is the NTM accepting $L_1$, and $s,t$ are start, accept configurations

Outputting $G$: Cycle through all pairs of configurations, checking if there is an edge between them, outputting 0 or 1 in the adjacency matrix

Edge checking done using $M$'s transition table
PATH is NL-complete

- Log-space reducing any NL language $L_1$ to PATH

  - Given input $x$, output $(G,s,t)$ where $G$ is the configuration graph $G(M,x)$, where $M$ is the NTM accepting $L_1$, and $s,t$ are start, accept configurations

  - Outputting $G$: Cycle through all pairs of configurations, checking if there is an edge between them, outputting 0 or 1 in the adjacency matrix

  - Edge checking done using $M$’s transition table

  - Need to store only two configurations at a time in the work-tape
**PATH is NL-complete**

- Log-space reducing any NL language $L_1$ to PATH

  - Given input $x$, output $(G,s,t)$ where $G$ is the configuration graph $G(M,x)$, where $M$ is the NTM accepting $L_1$, and $s,t$ are start, accept configurations

- Outputting $G$: Cycle through all pairs of configurations, checking if there is an edge between them, outputting 0 or 1 in the adjacency matrix

  - Edge checking done using $M$’s transition table

  - Need to store only two configurations at a time in the work-tape

Note: in fact $O(S)$-space reduction from $L \in \text{NSPACE}(S)$ to PATH
If $\text{PATH} \in \text{co-NL}$
If PATH ∈ co-NL

If PATH ∈ co-NL, then co-NL ⊆ NL
If \( \text{PATH} \in \text{co-NL} \)

- If \( \text{PATH} \in \text{co-NL} \), then \( \text{co-NL} \subseteq \text{NL} \)
- For any \( L \in \text{co-NL} \), we have \( L \leq_L \text{PATH}^c \) (as \( L^c \leq_L \text{PATH} \)), and
  - if \( \text{PATH}^c \in \text{NL} \), then \( L \in \text{NL} \) (NL is downward closed under \( \leq_L \))
If $\text{PATH} \in \text{co-NL}$

- If $\text{PATH} \in \text{co-NL}$, then $\text{co-NL} \subseteq \text{NL}$
- For any $L \in \text{co-NL}$, we have $L \preceq_L \text{PATH}^c$ (as $L^c \preceq_L \text{PATH}$), and if $\text{PATH}^c \in \text{NL}$, then $L \in \text{NL}$ (NL is downward closed under $\preceq_L$)
- Implies $\text{co-NL} = \text{NL}$ (why?)
If $\text{PATH} \in \text{co-NL}$

- If $\text{PATH} \in \text{co-NL}$, then $\text{co-NL} \subseteq \text{NL}$

- For any $L \in \text{co-NL}$, we have $L \leq L^c \text{PATH}^c$ (as $L^c \leq L \text{PATH}$), and if $\text{PATH}^c \in \text{NL}$, then $L \in \text{NL}$ (NL is downward closed under $\leq$)

- Implies $\text{co-NL} = \text{NL}$ (why?)

- If $Y \subseteq X$, then $\text{co-Y} \subseteq \text{co-X}$. Consider $X = \text{NL}, Y = \text{co-NL}$. 
If $\text{PATH} \in \text{co-NL}$
If $\text{PATH} \in \text{co-NL}$

In fact, $\text{PATH} \in \text{co-NL}$ implies $\text{co-NSPACE}(S) = \text{NSPACE}(S)$
If \( \text{PATH} \in \text{co-NL} \)

- In fact, \( \text{PATH} \in \text{co-NL} \) implies \( \text{co-NSPACE}(S) = \text{NSPACE}(S) \)
- Recall: \( O(S) \)-space reduction from \( L \in \text{NSPACE}(S) \) to \( \text{PATH} \)
If PATH ∈ co-NL

- In fact, PATH ∈ co-NL implies co-NSPACE(S) = NSPACE(S)

- Recall: O(S)-space reduction from L ∈ NSPACE(S) to PATH

- i.e., from L' ∈ co-NSPACE(S) to PATH^c
If $\text{PATH} \in \text{co-NL}$

- In fact, $\text{PATH} \in \text{co-NL}$ implies $\text{co-NSPACE}(S) = \text{NSPACE}(S)$

- Recall: $O(S)$-space reduction from $L \in \text{NSPACE}(S)$ to $\text{PATH}$

  - i.e., from $L' \in \text{co-NSPACE}(S)$ to $\text{PATH}^c$

- Size of the new instance is at most $N = 2^{O(|L|)}$
If $\text{PATH} \in \text{co-NL}$

- In fact, $\text{PATH} \in \text{co-NL}$ implies $\text{co-NSPACE}(S) = \text{NSPACE}(S)$

- Recall: $O(S)$-space reduction from $L \in \text{NSPACE}(S)$ to PATH

  - i.e., from $L' \in \text{co-NSPACE}(S)$ to $\text{PATH}^c$

  - Size of the new instance is at most $N = 2^{O(|S|)}$

- $\text{PATH}^c \in \text{NL}$ implies an NTM that decides if the instance is in $\text{PATH}^c$ in $\text{NSPACE}(\log N) = \text{NSPACE}(S)$
If PATH ∈ co-NL

- In fact, PATH ∈ co-NL implies co-NSPACE(S) = NSPACE(S)

- Recall: O(S)-space reduction from L ∈ NSPACE(S) to PATH

- i.e., from L’ ∈ co-NSPACE(S) to PATH\(^c\)

- Size of the new instance is at most N = 2^{O(|l|)}

- PATH\(^c\) ∈ NL implies an NTM that decides if the instance is in PATH\(^c\) in NSPACE(log N) = NSPACE(S)

- Then L’ ∈ co-NSPACE(S) is also in NSPACE(S), by composing space-bounded computations. So, co-NSPACE(S) ⊆ NSPACE(S)
If $\text{PATH} \in \text{co-NL}$

- In fact, $\text{PATH} \in \text{co-NL}$ implies $\text{co-NSPACE}(S) = \text{NSPACE}(S)$
- Recall: $O(S)$-space reduction from $L \in \text{NSPACE}(S)$ to $\text{PATH}$
  - i.e., from $L' \in \text{co-NSPACE}(S)$ to $\text{PATH}^c$
  - Size of the new instance is at most $N = 2^{O(l_s)}$
- $\text{PATH}^c \in \text{NL}$ implies an NTM that decides if the instance is in $\text{PATH}^c$ in $\text{NSPACE}(\log N) = \text{NSPACE}(S)$
- Then $L' \in \text{co-NSPACE}(S)$ is also in $\text{NSPACE}(S)$, by composing space-bounded computations. So, $\text{co-NSPACE}(S) \subseteq \text{NSPACE}(S)$
- Hence $\text{co-NSPACE}(S) = \text{NSPACE}(S)$
If $\text{PATH} \in \text{co-NL}$
If PATH $\in$ co-NL

- If PATH $\in$ co-NL then $\text{NSPACE}(S) = \text{co-NSPACE}(S)$
If \( \text{PATH} \in \text{co-NL} \)

- If \( \text{PATH} \in \text{co-NL} \) then \( \text{NSPACE}(S) = \text{co-NSPACE}(S) \)
- In particular \( \text{NL} = \text{co-NL} \)
If $\text{PATH} \in \text{co-NL}$

- If $\text{PATH} \in \text{co-NL}$ then $\text{NSPACE}(S) = \text{co-NSPACE}(S)$

- In particular $\text{NL} = \text{co-NL}$

- And indeed, $\text{PATH} \in \text{co-NL}$!
If PATH ∈ co-NL

- If PATH ∈ co-NL then NSPACE(S) = co-NSPACE(S)
- In particular NL = co-NL
- And indeed, PATH ∈ co-NL!
- There is a (polynomial sized) certificate that can be verified in log-space, that there is no path from s to t in a graph G
$\text{PATH}^c \in \text{NL}$
PATH$^c \in \text{NL}$

Certificate for $(s,t)$ connected is just the path
\[ \text{PATH}^c \in \text{NL} \]

- Certificate for \((s,t)\) connected is just the path.
- What is a certificate that \((s,t)\) not connected?
PATH\(_c \in NL\)

Certificate for \((s,t)\) connected is just the path

What is a certificate that \((s,t)\) not connected?

- size \(c\) of the connected component of \(s\), \(C\); a list of all \(v \in C\) (with certificates) in order; and (somehow) a certificate for \(c = |C|\)
Certificate for (s,t) connected is just the path

What is a certificate that (s,t) not connected?

size c of the connected component of s, C; a list of all v ∈ C (with certificates) in order; and (somehow) a certificate for c = |C|

Log-space, one-scan verification of certified C (believing |C|): scan list, checking certificates, counting, ensuring order, and that t not in the list. Verify count.
Certificate for (s,t) connected is just the path

What is a certificate that (s,t) not connected?

size $c$ of the connected component of s, $C$; a list of all $v \in C$ (with certificates) in order; and (somehow) a certificate for $c = |C|$

Log-space, one-scan verification of certified $C$ (believing $|C|$): scan list, checking certificates, counting, ensuring order, and that t not in the list. Verify count.

List has $|C|$ many $v \in C$, without repeating
Certificate for ICI
Certificate for $|C|$:

Let $C_i :=$ set of nodes within distance $i$ of $s$. Then $C = C_N$.
Certificate for $|C|$

- Let $C_i :=$ set of nodes within distance $i$ of $s$. Then $C = C_N$
- Tail recursion to verify $|C_N|$:
Certificate for $|C|$:

- Let $C_i :=$ set of nodes within distance $i$ of $s$. Then $C = C_N$.
- Tail recursion to verify $|C_N|$: 
  - Read $|C_{N-1}|$, believing it verify $|C_N|$, forget $|C_N|$.
Certificate for $|C|$:

Let $C_i :=$ set of nodes within distance $i$ of $s$. Then $C = C_N$

Tail recursion to verify $|C_N|$:
- Read $|C_{N-1}|$, believing it verify $|C_N|$, forget $|C_N|$;
- Read $|C_{N-2}|$, believing it verify $|C_{N-1}|$, forget $|C_{N-1}|$; ...


Certificate for $|C|$ 

- Let $C_i :=$ set of nodes within distance $i$ of $s$. Then $C = C_N$
- Tail recursion to verify $|C_N|$: 
  - Read $|C_{N-1}|$, believing it verify $|C_N|$, forget $|C_N|$;
  - Read $|C_{N-2}|$, believing it verify $|C_{N-1}|$, forget $|C_{N-1}|$; ...
  - Base case: $|C_0| = 1$
Certificate for $|C|$ 

Let $C_i :=$ set of nodes within distance $i$ of $s$. Then $C = C_N$

Tail recursion to verify $|C_N|$: 
- Read $|C_{N-1}|$, believing it verify $|C_N|$, forget $|C_N|$;
- Read $|C_{N-2}|$, believing it verify $|C_{N-1}|$, forget $|C_{N-1}|$; ...
- Base case: $|C_0|=1$

Believing $|C_{i-1}|$ verify $|C_i|$: for each vertex $v$ certificate that $v \in C_i$ or that $v \notin C_i$ (these certificates are poly($N$) long)
Certificate for \(|C|\)

- Let \(C_i :=\) set of nodes within distance \(i\) of \(s\). Then \(C = C_N\)
- Tail recursion to verify \(|C_N|\):
  - Read \(|C_{N-1}|\), believing it verify \(|C_N|\), forget \(|C_N|\);
  - Read \(|C_{N-2}|\), believing it verify \(|C_{N-1}|\), forget \(|C_{N-1}|\); ...
- Base case: \(|C_0| = 1\)
- Believing \(|C_{i-1}|\) verify \(|C_i|\): for each vertex \(v\) certificate that \(v \in C_i\) or that \(v \notin C_i\) (these certificates are poly(N) long)
- Certificate that \(v \notin C_i\) given (i.e., believing) \(|C_{i-1}|\): list of all vertices in \(C_{i-1}\) in order, with certificates. As before verify \(C_{i-1}\) believing \(|C_{i-1}|\) (scan and ensure list is correct/complete), but also check that no node in the list has \(v\) as a neighbor.
Certificate for $t \in \mathbb{C}_N$
Certificate for $t \in C_N$
Certificate for $t \in C_N$
Certificate for $t \in \mathcal{C}_N$
Certificate for $t \in \mathbb{C}_N$

$t \in \mathbb{C}_N / \mid \mathbb{C}_N \mid$

$\mid \mathbb{C}_N \mid$ vertices
Certificate for $t \in \mathbb{C}_N$

$t \in \mathbb{C}_N / |\mathbb{C}_N| \quad \quad \quad |\mathbb{C}_N|$

$|\mathbb{C}_N|$ vertices

$v_i \in \mathbb{C}_N \quad \text{path}(s,v_i)$
Certificate for $t \in C_N$
Certificate for $t \in C_N$
Certificate for $t \in C_N$
Certificate for $t \in \mathbb{C}_N$

- $t \in \mathbb{C}_N / |\mathbb{C}_N|$
- $|\mathbb{C}_N|$
- $|\mathbb{C}_N| / |\mathbb{C}_{N-1}|$
- $|\mathbb{C}_{N-1}|$

- $|\mathbb{C}_N| \text{ vertices}$
- $|\mathbb{C}_{N-1}| \text{ all } N \text{ vertices}$

$v_i \in \mathbb{C}_N \text{ path}(s, v_i)$

$v_1 \neq t$
Certificate for \( t \in \mathbb{C}_N \)

\[
\begin{align*}
t & \in \mathbb{C}_N \\
|\mathbb{C}_N| & / |\mathbb{C}_{N-1}| \\
|\mathbb{C}_{N-1}| &
\end{align*}
\]

\( |\mathbb{C}_N| \)

\( |\mathbb{C}_{N-1}| \)

\( |\mathbb{C}_{N-2}| \)

\( |\mathbb{C}_{N-1}| \)

\( v_i \in \mathbb{C}_N \) path(s,\( v_i \))

\( v_i \in \mathbb{C}_N \) path(s,\( v_i \))

\( v_i \neq t \)
Certificate for $t \in C_N$

$t \in C_N / |C_N|$

$|C_N|$

$|C_N| / |C_N-1|$

$|C_N-1|$

$|C_N|$

$|C_N| / |C_N-1|$

$v_i \in C_N$ path(s,$v_i$)

$v_i \in C_N$ path(s,$v_i$)

$v_i \in C_N$

$v_i \neq t$
Certificate for $t \in C_N$
Certificate for $t \in C_N$

$t \in C_N / |C_N|$

$|C_N|$

$|C_N| / |C_{N-1}|$

$|C_{N-1}|$

$|C_N| / |C_N-1|$

$v_i \in C_N$ path(s,$v_i$)

$v_i \neq t$

$v_i \in C_N$ path(s,$v_i$)

$v_i \in C_N$ path(s,$v_i$)

$v_i \in C_N / |C_{N-1}|$

$|C_{N-1}|$ vertices

$|C_N|$ vertices

-all N vertices

v_i \in C_N / |C_{N-1}|
Certificate for $t \in \mathbb{C}_N$

$t \in \mathbb{C}_N / |\mathbb{C}_N|$

$|\mathbb{C}_N|$

$|\mathbb{C}_N| / |\mathbb{C}_{N-1}|$

$|\mathbb{C}_{N-1}|$

$|\mathbb{C}_N| / |\mathbb{C}_{N-1}|$

$|\mathbb{C}_{N-1}|$

$|\mathbb{C}_N| / |\mathbb{C}_{N-1}|$

$|\mathbb{C}_{N-1}|$

$v_i \in \mathbb{C}_N$ \hspace{1cm} path$(s,v_i)$

$v_i \in \mathbb{C}_{N-1}$ \hspace{1cm} path$(s,v_j)$

$v_i \neq t$

$v_i \in \mathbb{C}_N$

$v_i \in \mathbb{C}_{N-1}$

$v_j \in \mathbb{C}_{N-1}$

all $N$ vertices

$|\mathbb{C}_N|$ vertices

$|\mathbb{C}_{N-1}|$ vertices
Certificate for $t \in C_N$

- $t \in C_N / |C_N|$
- $|C_N|$
- $|C_N| / |C_{N-1}|$
- $|C_{N-1}|$

- $v_i \in C_N$ path($s,v_i$)
- $v_i \in C_N$ path($s,v_i$)
- $v_i \in C_N$
- $v_i \notin C_N / |C_{N-1}|$
- $v_j \in C_{N-1}$ path($s,v_j$)
- $v_j \not\rightarrow v_i$

$|C_N|$ vertices

all $N$ vertices

$|C_N| / |C_{N-1}|$ vertices

$v_i \in C_N$

$v_j \in C_{N-1}$ path($s,v_j$)
Certificate for $t \in C_N$
Certificate for $t \in C_N$

- $t \in C_N / |C_N|$
- $|C_N|$
- $|C_N| / |C_{N-1}|$
- $|C_{N-1}|$

- $v_i \in C_N$, $\text{path}(s, v_i)$
- $v_i \in C_N$, $\text{path}(s, v_i)$
- $v_i \notin C_N / |C_{N-1}|$
- $v_i \in C_N$

- $v_j \in C_{N-1}$, $\text{path}(s, v_j)$
- $v_j \Leftrightarrow v_i$