Lecture 10: Space Complexity III

Arijit Bishnu

27.03.2010

Outline

Outline

Space Complexity Classes

Definition for Recapitulation

PSPACE $= \bigcup_{c>0} \text{SPACE}(n^c)$. The class PSPACE is an analog of the class P.

Space Complexity Classes

Definition for Recapitulation

PSPACE $= \bigcup_{c>0} \text{SPACE}(n^c)$. The class PSPACE is an analog of the class P.

Definition for Recapitulation

NPSPACE $= \bigcup_{c>0} \text{NSPACE}(n^c)$. The class NPSPACE is an analog of the class NP.

Space Complexity Classes

Definition for Recapitulation

PSPACE $= \bigcup_{c>0} \text{SPACE}(n^c)$. The class PSPACE is an analog of the class P.

Definition for Recapitulation

NPSPACE $= \bigcup_{c>0} \text{NSPACE}(n^c)$. The class NPSPACE is an analog of the class NP.

Definition

L $= \text{SPACE}(\log n)$.

Space Complexity Classes

Definition for Recapitulation

PSPACE $= \bigcup_{c>0} \text{SPACE}(n^c)$. The class PSPACE is an analog of the class P.

Definition for Recapitulation

NPSPACE $= \bigcup_{c>0} \text{NSPACE}(n^c)$. The class NPSPACE is an analog of the class NP.

Definition

$\text{L} = \text{SPACE}(\log n)$.

Definition

$\text{NL} = \text{NSPACE}(\log n)$.

Examples

Is the following in L?

Check whether the following language is in L?

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

Examples

Is the following in L?

Check whether the following language is in L?

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

Is the following in L?

Check whether the following language is in L?

$$A = \{0^k 1^k \mid k \geq 0\}$$

A Certificate Based Definition of NL

Recall the Definition of the Class NP

A language $L \subseteq \{0,1\}^*$ is in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M such that for every $x \in \{0,1\}^*$,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$, then we call u a certificate for x (w.r.t. language L and machine M).

A Certificate Based Definition of NL

Recall the Definition of the Class NP

A language $L \subseteq \{0,1\}^*$ is in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M such that for every $x \in \{0,1\}^*$,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$, then we call u a certificate for x (w.r.t. language L and machine M).

- The idea for this definition was that the nondeterministic choices of the NDTM that result in acceptance can be viewed as a polynomial sized certificate that $x \in L$ and vice versa.

A Certificate Based Definition of NL

Recall the Definition of the Class NP

A language $L \subseteq \{0,1\}^*$ is in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M such that for every $x \in \{0,1\}^*$,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$, then we call u a certificate for x (w.r.t. language L and machine M).

- The idea for this definition was that the nondeterministic choices of the NDTM that result in acceptance can be viewed as a polynomial sized certificate that $x \in L$ and vice versa.
- A logspace NDTM, according to the concept of configuration graph that we studied, can generate a certificate that is polynomially long but it does not have the space to store it.

A Certificate Based Definition of NL continued...

- So, we assume a separate read-only tape for the logspace machine. We call this tape to be the certificate tape.

A Certificate Based Definition of NL continued...

- So, we assume a separate read-only tape for the logspace machine. We call this tape to be the certificate tape.
- At each step, the machine's head on this tape can either stay in place or move right. Thus, the tape is a read once tape.

The Certificate Based Definition of NL

The Certificate Based Definition of NL

The Certificate Based Definition of NL

Definition: Class NL

A language $L \subseteq \{0,1\}^*$ is in NL if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and

- a deterministic TM M using at most $O(\log |x|)$ space on its read/write tape for every input x and
- M has a certificate tape,

such that for every $x \in \{0,1\}^*$,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1$$

$M(x,u)$ denotes the output of M where x is placed on its input tape and u is placed on its certificate tape.

Example

Is the following in NL?

Check whether the following language is in NL?
PATH $= \{< G, s, t > \mid G$ is a directed graph in which there is a path from s to $t\}$. G has n nodes.

Example

Is the following in NL?

Check whether the following language is in NL?
PATH $= \{< G, s, t > \mid G$ is a directed graph in which there is a path from s to $t\}$. G has n nodes.

Solution: PATH $\in$ NL

Generate the certificate on the certificate tape and verify it.

Relations between Classes L, NL and P

The Relation between L, NL and P

$L \subseteq NL \subseteq P$

Relations between Classes L, NL and P

The Relation between L, NL and P

$L \subseteq NL \subseteq P$

Proof

Relations between Classes L, NL and P

The Relation between L, NL and P

$L \subseteq NL \subseteq P$

Proof

- It is easy to follow the relation $L \subseteq NL$.

Relations between Classes L, NL and P

The Relation between L, NL and P

$L \subseteq NL \subseteq P$

Proof

- It is easy to follow the relation $L \subseteq NL$.
- About the second part, look at the number of vertices (i.e. configurations) in the configuration graph corresponding to a language A in NL. The number of vertices is $2^{O(\log n)}$ which is surely a polynomial in n, the input size.

Relations between Classes L, NL and P

The Relation between L, NL and P

$L \subseteq NL \subseteq P$

Proof

- It is easy to follow the relation $L \subseteq NL$.
- About the second part, look at the number of vertices (i.e. configurations) in the configuration graph corresponding to a language A in NL. The number of vertices is $2^{O(\log n)}$ which is surely a polynomial in n, the input size.
- Just run through the polynomial number of configurations in polynomial time to find the accept or reject configuration. So, $A \in P$. Hence, $NL \subseteq P$.

Outline

Reductions

- The essence of reductions is to set up difficulty hierarchy among problems keeping in mind the complexity phenomenon (say P vs NP) we want to understand.

Reductions

- The essence of reductions is to set up difficulty hierarchy among problems keeping in mind the complexity phenomenon (say P vs NP) we want to understand.
- The reduction cannot be more powerful than the weaker class.

Reductions

- The essence of reductions is to set up difficulty hierarchy among problems keeping in mind the complexity phenomenon (say P vs NP) we want to understand.
- The reduction cannot be more powerful than the weaker class.
- So, to understand the relation between P and NP, we used polynomial reduction.

Reductions

- The essence of reductions is to set up difficulty hierarchy among problems keeping in mind the complexity phenomenon (say P vs NP) we want to understand.
- The reduction cannot be more powerful than the weaker class.
- So, to understand the relation between P and NP, we used polynomial reduction.
- We understand that we cannot use polynomial reduction for understanding the complexity question whether L = NL?

Reductions

- The essence of reductions is to set up difficulty hierarchy among problems keeping in mind the complexity phenomenon (say P vs NP) we want to understand.
- The reduction cannot be more powerful than the weaker class.
- So, to understand the relation between P and NP, we used polynomial reduction.
- We understand that we cannot use polynomial reduction for understanding the complexity question whether L = NL?
- So, we use logspace reductions.

Reductions

- The essence of reductions is to set up difficulty hierarchy among problems keeping in mind the complexity phenomenon (say P vs NP) we want to understand.
- The reduction cannot be more powerful than the weaker class.
- So, to understand the relation between P and NP, we used polynomial reduction.
- We understand that we cannot use polynomial reduction for understanding the complexity question whether L = NL?
- So, we use logspace reductions.
- Any reduction takes an input instance of size n and maps it to another instance of size at least n.

Reductions

- The essence of reductions is to set up difficulty hierarchy among problems keeping in mind the complexity phenomenon (say P vs NP) we want to understand.
- The reduction cannot be more powerful than the weaker class.
- So, to understand the relation between P and NP, we used polynomial reduction.
- We understand that we cannot use polynomial reduction for understanding the complexity question whether L = NL?
- So, we use logspace reductions.
- Any reduction takes an input instance of size n and maps it to another instance of size at least n.
- But, a logspace machine does not have the memory space to write the reduced instance. So, what to do?

Logspace Reduction

Definition: Logspace Computable Function

A logspace reducer is a TM M with $O(\log n)$ space (symbols) on its read/write tape that computes a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ where $f(x)$ is the string remaining on the output tape after M halts when it is started with $x \in \{0,1\}^*$ on its input tape. We term such an f as a logspace computable function.

Logspace Reduction

Definition: Logspace Computable Function

A logspace reducer is a TM M with $O(\log n)$ space (symbols) on its read/write tape that computes a function $f : \{0,1\}^* \to \{0,1\}^*$ where $f(x)$ is the string remaining on the output tape after M halts when it is started with $x \in \{0,1\}^*$ on its input tape. We term such an f as a logspace computable function.

Definition: Logspace Reducibility

A language A is logspace reducible to a language B, denoted as $A \leq_L B$, if there is a logspace computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that $x \in A$ iff $f(x) \in B$ for every $x \in \{0,1\}^*$.

Properties of Logspace Reducibility

Transitivity

If $A \leq_L B$ and $B \leq_L C$, then $A \leq_L C$.

Proof

Left as an exercise.

Properties of Logspace Reducibility

Theorem

If $A \leq_L B$ and $B \in \mathsf{L}$, then $A \in \mathsf{L}$.

Proof

- Does the idea from $\leq_P$ carry over?

Properties of Logspace Reducibility

Theorem

If $A \leq_L B$ and $B \in \mathsf{L}$, then $A \in \mathsf{L}$.

Proof

- Does the idea from $\leq_P$ carry over?
- We need to show a logspace machine M_A for A. Let f be the logspace computable function corresponding to $\leq_L$.

Properties of Logspace Reducibility

Theorem

If $A \leq_L B$ and $B \in \mathsf{L}$, then $A \in \mathsf{L}$.

Proof

- Does the idea from $\leq_P$ carry over?
- We need to show a logspace machine M_A for A. Let f be the logspace computable function corresponding to $\leq_L$.
- M_A computes individual symbols of $f(x)$ as requested by M_B, which is B's machine.

Properties of Logspace Reducibility

Theorem

If $A \leq_L B$ and $B \in \mathsf{L}$, then $A \in \mathsf{L}$.

Proof

- Does the idea from $\leq_P$ carry over?
- We need to show a logspace machine M_A for A. Let f be the logspace computable function corresponding to $\leq_L$.
- M_A computes individual symbols of $f(x)$ as requested by M_B, which is B's machine.
- M_A keeps track of where M_B's input head would be on $f(x)$.

Properties of Logspace Reducibility

Theorem

If $A \leq_L B$ and $B \in \mathsf{L}$, then $A \in \mathsf{L}$.

Proof

- Does the idea from $\leq_P$ carry over?
- We need to show a logspace machine M_A for A. Let f be the logspace computable function corresponding to $\leq_L$.
- M_A computes individual symbols of $f(x)$ as requested by M_B, which is B's machine.
- M_A keeps track of where M_B's input head would be on $f(x)$.
- Every time M_B moves, M_A restarts the computation of f on x from the beginning and ignores all output except for the desired location of $f(x)$. This takes too much time but who cares as we are bothered about space.

Properties of Logspace Reducibility

Theorem

If $A \leq_L B$ and $B \in \mathsf{L}$, then $A \in \mathsf{L}$.

Proof

- Does the idea from $\leq_P$ carry over?
- We need to show a logspace machine M_A for A. Let f be the logspace computable function corresponding to $\leq_L$.
- M_A computes individual symbols of $f(x)$ as requested by M_B, which is B's machine.
- M_A keeps track of where M_B's input head would be on $f(x)$.
- Every time M_B moves, M_A restarts the computation of f on x from the beginning and ignores all output except for the desired location of $f(x)$. This takes too much time but who cares as we are bothered about space.
- We need to store a single symbol of $f(x)$ at any point.

Outline

NL-completeness

Definition

A language A is NL-complete if

- $A \in \mathrm{NL}$.
- for every B in NL, we have $B \leq_L A$.

NL-completeness

Definition

A language A is NL-complete if

- $A \in \mathrm{NL}$.
- for every B in NL, we have $B \leq_L A$.

Corollary

If any NL-complete language is in L, then $\mathrm{L} = \mathrm{NL}$.

NL-completeness

Definition

A language A is NL-complete if

- $A \in \mathrm{NL}$.
- for every B in NL, we have $B \leq_L A$.

Corollary

If any NL-complete language is in L, then $\mathrm{L} = \mathrm{NL}$.

Proof

Easy and left as an exercise.

PATH is NL-complete

PATH

PATH $= \{< G, s, t > \mid G$ is a directed graph in which there is a path from s to $t\}$. G has n nodes.

Theorem

PATH is NL-complete.

Proof

- PATH $\in$ NL.

PATH is NL-complete

PATH

PATH $= \{< G, s, t > \mid G$ is a directed graph in which there is a path from s to $t\}$. G has n nodes.

Theorem

PATH is NL-complete.

Proof

- PATH $\in$ NL.
- Let A be any language in NL that was decided by a logspace TM M. We have to define a logspace computable function f that logspace reduces A to PATH.

PATH is NL-complete

PATH

PATH $= \{< G, s, t > \mid G$ is a directed graph in which there is a path from s to $t\}$. G has n nodes.

Theorem

PATH is NL-complete.

Proof

- PATH $\in$ NL.
- Let A be any language in NL that was decided by a logspace TM M. We have to define a logspace computable function f that logspace reduces A to PATH.
- Look at the configuration graph $G_{M,x}$. The number of vertices $= 2^{O(\log n)}$.

PATH is NL-complete

PATH

PATH $= \{< G, s, t > \mid G$ is a directed graph in which there is a path from s to $t\}$. G has n nodes.

Theorem

PATH is NL-complete.

Proof

- PATH $\in$ NL.
- Let A be any language in NL that was decided by a logspace TM M. We have to define a logspace computable function f that logspace reduces A to PATH.
- Look at the configuration graph $G_{M,x}$. The number of vertices $= 2^{O(\log n)}$.
- $f(x)$ has a relation with $G_{M,x}$.

The Proof Continued

Proof (continued)

- In $G_{M,x}$, $\exists$ a path from C_s^x to C_{acc} iff M accepts x.

The Proof Continued

Proof (continued)

- In $G_{M,x}$, $\exists$ a path from C_s^x to C_{acc} iff M accepts x.
- $G_{M,x}$ is represented by an adjacency matrix $\mathcal{A}$.

The Proof Continued

Proof (continued)

- In $G_{M,x}$, $\exists$ a path from C_s^x to C_{acc} iff M accepts x.
- $G_{M,x}$ is represented by an adjacency matrix $\mathcal{A}$.
- $\mathcal{A}[i,j] = 1$ iff there is an edge from configuration (vertex) C_i to configuration (vertex) C_j in $G_{M,x}$.

The Proof Continued

Proof (continued)

- In $G_{M,x}$, $\exists$ a path from C_s^x to C_{acc} iff M accepts x.
- $G_{M,x}$ is represented by an adjacency matrix $\mathcal{A}$.
- $\mathcal{A}[i,j] = 1$ iff there is an edge from configuration (vertex) C_i to configuration (vertex) C_j in $G_{M,x}$.
- Can $\mathcal{A}$ be computed by a logspace reduction?

The Proof Continued

Proof (continued)

- In $G_{M,x}$, $\exists$ a path from C_s^x to C_{acc} iff M accepts x.
- $G_{M,x}$ is represented by an adjacency matrix $\mathcal{A}$.
- $\mathcal{A}[i,j] = 1$ iff there is an edge from configuration (vertex) C_i to configuration (vertex) C_j in $G_{M,x}$.
- Can $\mathcal{A}$ be computed by a logspace reduction?
- Given a pair $< C_i, C_j >$, a deterministic machine can in space $|C_i| + |C_j| = O(\log |x|)$ examine C_i and C_j and can determine whether C_j is one of the at most two configurations that can follow C_i according to δ of M.

NL and P

A relook at the Theorem NL $\subseteq$ P

- For any $A \in$ NL, we have $A \leq_L$ PATH.

NL and P

A relook at the Theorem NL $\subseteq$ P

- For any $A \in$ NL, we have $A \leq_L$ PATH.
- The TM that computes $\leq_L$ goes through at most polynomial configurations ($2^{O(\log n)}$) and hence, runs in polynomial time.

NL and P

A relook at the Theorem NL $\subseteq$ P

- For any $A \in$ NL, we have $A \leq_L$ PATH.
- The TM that computes $\leq_L$ goes through at most polynomial configurations ($2^{O(\log n)}$) and hence, runs in polynomial time.
- So, any $A \in$ NL is poly-time reducible to PATH, i.e. $A \leq_P$ PATH.

NL and P

A relook at the Theorem NL $\subseteq$ P

- For any $A \in$ NL, we have $A \leq_L$ PATH.
- The TM that computes $\leq_L$ goes through at most polynomial configurations ($2^{O(\log n)}$) and hence, runs in polynomial time.
- So, any $A \in$ NL is poly-time reducible to PATH, i.e. $A \leq_P$ PATH.
- PATH $\in$ P.

NL and P

A relook at the Theorem NL $\subseteq$ P

- For any $A \in$ NL, we have $A \leq_L$ PATH.
- The TM that computes $\leq_L$ goes through at most polynomial configurations ($2^{O(\log n)}$) and hence, runs in polynomial time.
- So, any $A \in$ NL is poly-time reducible to PATH, i.e. $A \leq_P$ PATH.
- PATH $\in$ P.
- $A \leq_P$ PATH and PATH $\in$ P, so $A \in$ P.

Outline

NL and its complement class coNL

$\overline{\text{PATH}}$

A decision problem for $\overline{\text{PATH}}$ accepts when there is no path from s to t in G.

NL and its complement class coNL

$\overline{\text{PATH}}$

A decision problem for $\overline{\text{PATH}}$ accepts when there is no path from s to t in G.

Theorem

$\overline{\text{PATH}} \in \text{NL}$

NL and its complement class coNL

$\overline{\text{PATH}}$

A decision problem for $\overline{\text{PATH}}$ accepts when there is no path from s to t in G.

Theorem

$\overline{\text{PATH}} \in \text{NL}$

Proof

Left as an exercise.
Hints: Use the certificate definition of NL.

NL and its complement class coNL

$\overline{\text{PATH}}$

A decision problem for $\overline{\text{PATH}}$ accepts when there is no path from s to t in G.

Theorem

$\overline{\text{PATH}} \in \text{NL}$

Proof

Left as an exercise.
Hints: Use the certificate definition of NL.

Theorem

$\text{NL} = \text{coNL}$.

Outline

A Relation between NSPACE and coNSPACE

Theorem

For every space constructible $S(n) > \log n$, $\mathrm{NSPACE}(S(n)) = \mathrm{coNSPACE}(S(n))$.

Inclusion Relation

$\mathrm{DTIME}(x) \subseteq \mathrm{SPACE}(x) \subseteq \mathrm{NSPACE}(x) = \mathrm{coNSPACE}(x) \subseteq \mathrm{DTIME}(2^{O(x)})$

where x denotes $S(n)$.