

# CSE200 Lecture Notes

## Space Complexity

Lecture by Russell Impagliazzo  
Notes by William Matthews

Lecture April 17, 2010

### 1 Space (Memory) Complexity

We assume that the input is read only, and that the output is write only, and we don't count either as part of the memory (space) usage.

**Lemma 1.** *If  $f$  and  $g$  are functions computable in  $DSPACE(S(n))$ , then  $g \circ f = g(f(x)) \in DSPACE(S(\exp(S(n))))$ .*

**Corollary 1.** *If  $f$  and  $g$  are functions computable in  $L = DSPACE(\log n)$  then  $g \circ f \in L = DSPACE(\log n)$*

Suppose we first run a machine for  $f$  to compute  $f(x)$  on its output tape, and then use this same tape as the input to  $g$ . The output tape doesn't count as memory used by  $f$ , and it doesn't count as memory used by  $g$ , but it *does* count as memory used by a machine which computes  $g \circ f$  using this approach. Thus, this approach may well use too much space.

Instead, use two additional counters: one to store the position on the output tape of  $f$ , and one to store the position on the input tape of  $g$ .

**Lemma 2.** *If any space  $S(n) \geq \log n$  algorithm runs for more than  $\exp(S(n))$  time, then it doesn't terminate.*

*Proof.* The number of configurations is at most  $\exp(S(n))$ . If we run for more than the number of configurations steps, then we must be in the same configuration twice, and therefore will repeat the same sequence of steps forever.  $\square$

**Corollary 2.**  *$DSPACE(S(n)) \subseteq DTIME(\exp(O(S(n))))$  for  $S(n) \geq \log n$ .*

**Corollary 3.** *If  $f \in DSPACE(S(n))$  for  $S(n) \geq \log n$ , then  $|f(x)| \leq \exp(O(S(n)))$ .*

Returning to the simulation of  $g \circ f$ : Simulate  $g(y)$  where we think of  $y = f(x)$ , keeping track of the position  $i$  of the position of the input tape head for  $g$ . Every time that we need the value of  $y_i$ , we compute  $f(x)$  from scratch until it outputs the  $i^{\text{th}}$  bit. Then resume the simulation of  $g$ . Since we reuse the same memory for each simulation of  $f$  it uses a total of  $S(n)$  space. Simulating  $g$  uses space  $S(|f(x)|) \leq S(\exp(S(n)))$ , and space  $O(S(n))$  to keep track of the position of the tape heads. Thus, we conclude Lemma 1.

### 2 Reductions

For languages  $A$  and  $B$ , a log-space mapping reduction from  $A$  to  $B$ , written  $A \leq_{mL} B$ , is a function  $f \in L$  such that  $x \in A \iff f(x) \in B$ . From Lemma 1, if  $A \leq_{mL} B$  and  $B \in L$  then  $A \in L$ . Also from Lemma 1, if  $A \leq_{mL} B$  and  $B \leq_{mL} C$  then  $A \leq_{mL} C$ .

### 3 Completeness

Given a notion of reduction, we can define a notion of completeness.

Consider the language Directed Graph Reachability,  $DGR = \{\langle G, u, v \rangle \mid \text{there is a directed path from } u \text{ to } v \text{ in the graph } G = (V, E), \text{ represented by an adjacency matrix}\}$ .

**Theorem 1.** *DGR is NL-complete under  $\leq_{mL}$  reductions.*

*Proof.* First,  $DGR \in NL$ : We store in memory a vertex  $a \in V$  and maintain the invariant that along a non-rejecting computation path there exists a path in  $G$  from  $u$  to  $a$ . Begin with  $a = u$ . If  $a = v$  accept. Non-deterministically choose  $b \in V$ . If  $(a, b) \notin E$  reject, otherwise set  $a = b$  and continue. If we run for more than  $|V|$  steps, reject.

This algorithm uses  $\log n$  space for  $a$ ,  $\log n$  space for  $b$ , and  $\log n$  space to count steps, for a total of  $O(\log n)$  memory.

If there exists a path in  $G$  from  $u$  to  $v$ , then there exists a path from  $u$  to  $v$  without any cycles of length at most  $|V|$ . When the algorithm non-deterministically chooses the vertices along this path it will accept.

If the algorithm accepts, then the sequence of values for  $a$  must correspond to a path from  $u$  to  $v$  in  $G$ .

Second, we must show that for all  $A \in L$  we have  $A \leq_{mL} DGR$ : We will construct a graph where vertices correspond to configurations of a machine for  $A$ ,  $u$  is the initial configuration, and  $v$  is a fixed accepting configuration, say with all tapes erased and in a specific accepting state; and where edges correspond to valid non-deterministic transitions between configurations.

Formally, let  $M$  be a non-deterministic log-space TM for  $A$  (which erases its tapes, etc. before accepting), and let  $x$  be any input to  $M$ .  $V =$  set of all configurations of  $M$  on inputs of length  $n$ .  $|V| = \text{poly}(n)$  and each  $v \in V$  has an  $O(\log n)$  bit representation.  $E = \{(c_1, c_2) \mid M \text{ can go from configuration } c_1 \text{ to } c_2, \text{ on input } x, \text{ in one step}\}$ . For each pair  $c_1, c_2$  of  $O(\log n)$  bit configurations, using an  $O(\log n)$  bit counter for the  $i^{\text{th}}$  bit of  $x$ , we can compute whether  $(c_1, c_2) \in E$  straightforwardly in log-space.

The output is  $\langle G = (V, E), u, v \rangle$  where  $u$  is the start configuration and  $v$  is the unique accept configuration.

By construction (essentially) there is a path from  $u$  to  $v$  in  $G$  if and only if there is an accepting computation of  $M$  on input  $x$  if and only if  $x \in A$ .  $\square$

### 4 Relationship to Time

It follows directly from Theorem 1 that  $NL \subseteq P$ . For any  $A \in NL$ , there exists  $f \in L$  such that  $x \in A \iff f(x) \in DGR$ . Since  $f \in L$ , we know  $f \in FP$ . Therefore  $A \leq_{mp} DGR$  and  $DGR \in P$ , so  $A \in P$ .

$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$ . By diagonalization similar to the time hierarchy theorem, we know  $L \subset PSPACE$ . We will show  $NL \subset PSPACE$ , but we don't know which of the other inclusions are proper.

### 5 $NSPACE(S(n)) \subseteq DSPACE(S(n)^2)$

**Theorem 2 (Savitch's Theorem).**  $NSPACE(S(n)) \subseteq DSPACE(S(n)^2)$  for  $S(n) \geq \log n$

**Corollary 4.**  $NL \subseteq DSPACE(\log^2 n)$

*Proof.* From Theorem 1, we know  $DGR$  is NL-complete. Thus, if we give a  $DSPACE(\log^2 n)$  algorithm for  $DGR$  we can conclude that  $NL \subseteq DSPACE(\log^2 n)$ .

1. Begin with  $\ell = n$
2. Savitch( $G, u, v, \ell$ ):
3. For each  $a \in V$  do:
4. Savitch( $G, u, a, \ell/2$ )

5. If so, erase your memory except  $a$  and try
6.  $\text{Savitch}(G, a, v, \ell/2)$
7. If so, accept

Each level of recursion uses  $O(\log n)$  memory, and we have  $\log n$  levels of recursion for a total of  $O(\log^2 n)$  memory. (We have omitted a few details from the algorithm concerning base cases, etc. but the omitted details don't change anything.)

Savitch's Theorem for space  $S(n) > \log n$  follows the same outline for appropriately adjusted space.  $\square$