

Logarithmic Space

7.1

When considering space complexity, the "used space" is meant to represent temporary storage needed by the TM to perform a task.

We want to focus on sub-linear storage.

⇒ we need to differentiate

- memory used for the computation
- memory used for storing input and output

Note: input and output memory should not be abused for temporary storage.

- ⇒
- input tape is read-only (end read only to end of input)
 - output tape is write-only
 - working tape is read-write: is the only one that counts

Space complexity $\mathcal{O}(n)$: number of work tape cells scanned on all inputs of length n .

$$\text{DSPACE}(\mathcal{O}(n)) = \{ L \mid L = \mathcal{L}(D) \text{ for some DTM } D \text{ with space complexity } \mathcal{O}(n) \}$$

$$\text{NSPACE}(\mathcal{O}(n)) = \{ L \mid L = \mathcal{L}(N) \text{ for some NTM } N \text{ with space complexity } \mathcal{O}(n) \}$$

In order to avoid dependence on the tape alphabet, we will consider the binary space complexity: number of bits that can be stored in the tape cells + finite state of the machine

$$\text{binary space complexity} = \text{number of tape cells} \cdot \log_2 |\Gamma| + \log_2 |Q|$$

Difference is immaterial:

added constant	$\log_2 Q $	end
constant factor	$\log_2 \Gamma $	

12/1/2016

What is the smallest amount of meaningful space?

- constant space:
 - finite state machines
 - recognize the regular languages
- to compute the input length, we require logarithmic space.

However, meaningful computations that require more than constant space (i.e. beyond regular languages) can be done with less than log-space:

$$\exists \text{ } L(n) = \log \cdot \log n : \text{ DSPACE}(O(L)) \not\equiv \text{ DSPACE}(O(1))$$

Example: For $k \in \mathbb{N}$, let w_k be the concatenation of all k -bit long strings in lexicographic order separated by #'s

$$w_k = 0^k \# 0^{k-1} 1 \# 0^{k-2} 10 \# 0^{k-2} 11 \# \dots \# 1^k$$

Then $S = \{w_k \mid k \in \mathbb{N}\}$ is not regular, and can be decided by a DTM that uses $\log \log$ space.

Proof: [exercise]

Note that $|w_k| > 2^k$ and thus $O(\log k) = O(\log \log |w_k|)$

- On the other hand, $\log \log n$ is the smallest amount of space that is more useful than constant space.

Spec used in composition of computations:

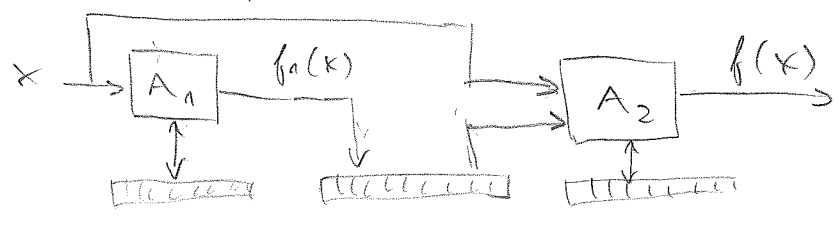
We consider the composition of two functions: over the binary alphabet $\Sigma = \{0, 1\}$:

$$f_1: \Sigma^* \rightarrow \Sigma^* \quad \text{computable in space } S_1(n) \text{ by } A_1$$

$$f_2: \Sigma^* \times \Sigma^* \rightarrow \Sigma^* \quad \text{--- } S_2(m) \text{ by } A_2$$

We want to compute $f(x) = f_2(x, f_1(x))$ by composing the algorithms A_1 and A_2 .

1) Trivial composition: without attempt to economize storage



Let $l_1(n) = \max_{x \in \Sigma^n} |f_1(x)|$... maximum output produced by A_1 on inputs of length n

Then $S_{\text{trivial}}(n) = S_1(n) + S_2(n + l_1(n)) + l_1(n) + \dots$ for storing the result of A_1 $+ \delta(n)$

with $\delta(n) = O(\log(l_1(n) + S_2(n + l_1(n))))$

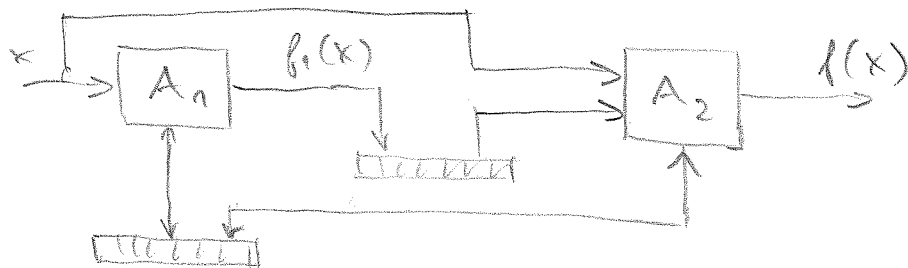
Note: $\delta(n)$ takes into account that A_2 refers to two storage devices, (one for storing $f_1(x)$ and one as its work tape), and it needs to maintain its location on both parts.

This can be done by storing the two pointers on the tape (which has overall length $l_1(n) + S_2(n + l_1(n))$)

$\Rightarrow 2 \text{ times } \log_2(l_1(n) + S_2(n + l_1(n))) \text{ bits.}$

2) Naive composition:

We economize storage by reusing the work-tape of A_1 for the computation of A_2



We get $\Delta_{\text{naive}}(n) = \max(\Delta_1(n) + \Delta_2(n + l_1(n))) +$
 $+ l_1(n) + \dots$ for storing the result of A_1
 $+ \delta(n)$

Note: both in the naive and in the trivial composition, the most costly storage may be that for the intermediate result.

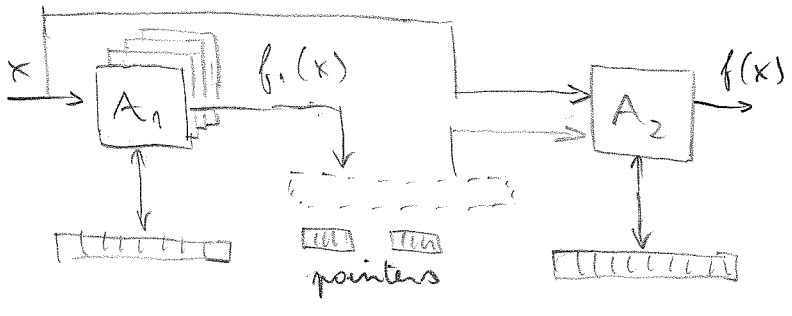
E.g., when: $\Delta_1(n) = \Delta_2(n) = \log n$, but the output of A_1 is of length polynomial in n , the overall space used is not $\log n$, but a polynomial in n .

Can we avoid to explicitly store the intermediate result?

We can, by paying some price in computation time.

Idea: instead of storing the output of A_1 on the tape, we recompute it, whenever A_2 needs to access it

3) Emulative composition:



The intermediate result is not stored explicitly. (we view A_2 as if it was accessing a virtual storage device)

Theorem: Let $\Sigma, f_1, f_2, f, \Omega_1, \Omega_2, l_1$ be as above, i.e.

$$\Sigma = \{0, 1\}$$

$$f_1: \Sigma^* \rightarrow \Sigma^*$$

computable in space $\Omega_1(n)$

$$f_2: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$$

--- " --- $\Omega_2(n)$

$$f(x) = f_2(x, f_1(x))$$

$$l_1(n) = \max_{x \in \Sigma^n} |f_1(x)|$$

Then f is computable in space Ω where

$$\Omega(n) = \Omega_1(n) + \Omega_2(n + l_1(n)) +$$

$$+ O(\log(n + l_1(n))) +$$

$$+ \delta'(n)$$

where $\delta'(n) = O(\log(\Omega_1(n) + \Omega_2(n + l_1(n))))$

Note: in the overall space used in the emulative composition, we do not have anymore the term $l_1(n)$ which accounts for storing the intermediate result $f_1(x)$.

Instead, we need to store pointers (i.e., counters) to a virtual storage for $f_1(x)$.

Log-space reductions

Let L_1 and L_2 be two languages.

A function R is a log-space reduction of L_1 to L_2 if

- 1) $w \in L_1 \Leftrightarrow R(w) \in L_2$
- 2) R is computable in log-space

We have already shown that a TM that has space complexity $\Delta(n)$ (for $\Delta(n)$ a polynomial) has running time bounded by $2^{O(\Delta(n))}$.

It is easy to see that the same result holds for $\Delta(n)$ being any function that is at least $O(\log(n))$. Hence, we get that every logspace reduction is also a poly-time reduction:

(since $2^{O(\log(n))} = O(p(n))$, where $p(n)$ is a polynomial)

In fact, all reductions used to show NP-hardness and encountered in practice are log-space reductions.

The classes L and NL:

(note: we use S to denote languages, to avoid confusion with L)

$$L = \{S \mid S = \mathcal{L}(D) \text{ for some DTM } D \text{ with log-space complexity}\}$$

$$= \bigcup_c \text{DSPACE}(c \cdot \log_2 n)$$

$$NL = \{S \mid S = \mathcal{L}(M) \text{ for some NTM } M \text{ with log-space complexity}\}$$

$$= \bigcup_c \text{NSPACE}(c \cdot \log_2 n)$$

From the above observation on the relationship between space and time complexity, we get immediately that $L \subseteq P$

$$NL \subseteq NP$$

Since also for a log-space NTM the number of different configurations is polynomial, the same argument shows also $NL \subseteq P$.

Note: the exact relationship between L, NL, P, NP is still an open problem

Prototypical problems for L and NL are graph-connectivity problems:

st-CONN: (source-target connectivity in directed graphs):

input: a directed graph $G = (V, E)$

two vertices s, t in V

output: yes iff there is a path in G from s to t
(denoted $s \rightarrow_G t$)

UCONN (connectivity in undirected graphs)

input: an undirected graph $G = (V, E)$

output: yes iff G is connected

In order to define completeness w.r.t L or NL , we need to make use of log-space reductions (rather than polytime red.)

Theorem: st-CONN is complete for NL (under log-space red.)

Proof:

1) st-CONN $\in NL$

We construct a NTM M that decides st-CONN using only log-space.

Let $G = (V, E)$ be the input graph, and s, t two vertices in V .

M stores on the tape:

- two vertices v_1, v_2 of G : $2 \cdot \log_2 |V|$ bits
 - a counter c of the traversed nodes: $\log_2 |V|$ bits
- $\Rightarrow O(\log_2 |G|)$ bits

Algorithm implemented by M :

```

 $v_1 \leftarrow s; c \leftarrow 0;$ 
while  $v_1 \neq t$  and  $c < |V|$ 
do guess a vertex  $v_2;$ 
  if  $(v_1, v_2) \in E$  then  $v_1 \leftarrow v_2$ 
  else  $c \leftarrow c + 1$ 
else reject
if  $v_1 = t$  then accept
else reject

```

It is easy to see that M accepts iff there is a path from s to t in G .

2) st -CONN is NL-hard

Given a log-space NTM M and an input w , we have to construct a directed graph $G_{M,w} = (V, E)$ and two vertices s, t in V st.

$w \in L(M)$ iff $s \rightarrow_{G_{M,w}} t$

- The set V of vertices is the set of IDs of M , where each ID consists of:
 - the content of the work tape
 - the TM state
 - the head position on work tape and on input tape
 (Note: for a fixed NTM M , each ID requires space logarithmic in $|w|$)

- The directed edges represent single transitions between the configurations.
 - an edge (v_1, v_2) depends only on N and on the single input bit of w at the position indicated by the input-tape head position in v_1 .
- The start vertex s is the one given by the initial ID of N on w .
- The end vertex t is the one corresponding to a canonical accepting ID (we can assume that N , when it accepts, erases the whole tape and moves left)

Then $s \xrightarrow{G_{N,w}} t$ iff N on w reaches the accepting ID
 iff $w \in L(N)$

Notice that, for a fixed machine N , $(G_{N,w}, s, t)$ can be constructed in log-space in $|w|$. (by enumerating through pairs of IDs and outputting as edges only the valid ones). \square

The proof above shows that st-CONN captures the essence of NL.

What about L and UCONN?

The complexity of UCONN was open for more than 20 years.

Theorem [Reingold, 2004]

$$UCONN \in L$$

Proof: very complicated

note that traditional breadth-first and depth-first graph search algorithms use linear space.

NL vs. coNL:

7.12

$\text{coNL} = \{ S \mid \bar{S} = \Sigma^* \setminus S \in \text{NL} \}$... complements of problems in NL

Recall that the acceptance condition of non-deterministic computations is symmetric in nature:

For $S \in \text{NL}$ and M a log-space NTM accepting S :

- if $w \in S$ then there exists an accepting computation of M on w
- if $w \notin S$ then all computations of M on w are not accepting

Note that:

- non-det-time complexity classes are not known to be closed under complementation (e.g. NP vs. coNP)
- non-det. space complexity classes are closed under complementation
 - for classes above PSPACE, this follows immediately from the quadratic simulation of non-deterministic computations via deterministic ones (e.g. PSPACE = NPSPACE)
 - for L vs. NL, it is open whether $L \neq \text{NL}$

However, we have the following result

Theorem [Immerman, Szepietowski 1988]: $\text{NL} = \text{coNL}$

Proof: complicated