

CSE200: Complexity theory

Space complexity

Shachar Lovett

January 29, 2020

1 Space complexity

We would like to be able to meaningfully study languages that can be computed in sub-linear space. This requires making a few variants to the Turing machine model we studied so far.

Definition 1.1 (Space complexity for languages). *Let $S : \mathbb{N} \rightarrow \mathbb{N}$ be a (computable) function. A language $L \subset \{0, 1\}^*$ is in space S if there exists a TM M which computes L . The TM has two tapes:*

1. *A read-only input tape.*
2. *A read-write work tape. We assume that on any input x , when running $M(x)$ the TM uses at most $O(S(|x|))$ many work tape cells.*

The class $SPACE(S)$ is the class of all such languages.

Similar to nondeterministic time, we can also define nondeterministic space. The definition is the same, except that we allow a NTM M . We denote by $NSPACE(S)$ the class of all such languages.

We also define space complexity for non-boolean functions. For these, as the output is not boolean, we have to be a bit careful.

Definition 1.2 (Space complexity for functions). *A function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ is in (implicit) $SPACE(S)$ if*

- (i) $|f(x)| \leq 2^{O(S(|x|))}$.
- (ii) *The language $\{(x, i) : f(x)_i = 1\}$ is in $SPACE(S)$.*

These together imply that we can compute specific bits of $f(x)$ in space $S(|x|)$. In particular, specifying the coordinate i requires at most space $S(|x|)$.

Some common classes of interest are:

- Polynomial space: $PSPACE = \cup_{c \geq 1} SPACE(n^c)$
- Nondeterministic polynomial space: $NPSPACE = \cup_{c \geq 1} NSPACE(n^c)$
- Logarithmic space: $LOGSPACE = SPACE(\log n)$ (it is sometimes abbreviated as L)
- Nondeterministic logarithmic space: $NLOGSPACE = NSPACE(\log n)$ (it is sometimes abbreviated as NL)

We would typically only consider space $S(n) \geq \log n$. The reason is that most algorithms require at least that much space, for example to describe a point. Surprisingly, many algorithms can be computed in sub-linear space. For example, summing two n -bit integers can be done in space $O(\log n)$. On the other hand, many efficient graph algorithms (such as BFS or DFS) require space linear in the number of vertices. Here are some more examples:

Claim 1.3. $SAT \in PSPACE$.

Proof. We can enumerate the possible assignments and try each one. □

We will later show that in fact $PH \subset PSPACE$.

Claim 1.4. Let $BALANCED = \{x \in \{0,1\}^* : x \text{ has the same number of 0's and 1's}\}$. Then $BALANCED \in L$.

Proof. Go over the input and count the number of 0's and 1's, then compare them. □

The last claim is left as exercise.

Claim 1.5. Let $SUMEQUAL = \{(x, y, z) \in \{0,1\}^* : x + y = z\}$, where we consider x, y, z as integers written in binary. Then $SUMEQUAL \in L$.

2 Complete languages

Log-space reductions are space-efficient reductions. They are useful when studying completeness for space-bounded computation.

Definition 2.1 (Log-space reductions). Let A, B be two languages. We say that A reduces to B under log-space reductions, denoted $A <_L B$, if there exists a log-space computable function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that $x \in A$ iff $f(x) \in B$.

Claim 2.2. Let A, B, C be language. Let $S(n) = \Omega(\log n)$ be a space bound.

1. If $A <_L B$ and $B <_L C$ then $A <_L C$.
2. If $A <_L B$ and $B \in SPACE(S)$ then $A \in SPACE(S)$.

Proof. We only prove the first item, the second is similar. Let x be an input for A . Let $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be the reduction from A to B computable in log-space, and let $g : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be the reduction from B to C computable in log-space. We need to show that $g(f(x))$ is also computable in log-space. Define the languages:

$$L_1 = \{(x, i) : f(x)_i = 1\}, \quad L_2 = \{(x, i) : g(x)_i = 1\}, \quad L_3 = \{(x, i) : g(f(x))_i = 1\}.$$

By assumption L_1, L_2 are in log-space, and we need to show that L_3 is also in log-space. Let M_1 be a log-space TM deciding L_1 and M_2 be a log-space TM deciding L_2 . Consider the following TM M_3 which will decide L_3 . Let x be the input and let $y = f(x)$. Consider running M_2 on (y, i) , where in addition we keep a counter for the input head location of M_2 . If the input head is at coordinate j then it needs to read y_j . We compute it “on-the-fly” by computing $f(x)_j$ using M_1 .

The total space requirements are: $O(\log n)$ for simulating $M_2(y)$; $O(\log n)$ for remembering the input head location of M_2 ; $O(\log n)$ for simulating $M_1(x)$. So the total space used is $O(\log n)$, which means that M_3 is also a log-space machine. \square

Definition 2.3 (Complete languages). *Let \mathcal{C} be a class of computable languages. A language A is \mathcal{C} -hard under log-space reductions if for any $B \in \mathcal{C}$ we have $B <_L A$. If also $A \in \mathcal{C}$ then we say A is \mathcal{C} -complete under log-space reductions. When \mathcal{C} is a space class (like PSPACE or NL) we would abbreviate that A is simply \mathcal{C} -complete, where the log-space reductions would be implicit.*

We next proceed to give a complete language for various space classes. Note that any language in L is (trivially) also L -complete, so to make it interesting, we should consider richer classes.

3 Nondeterministic log-space complete language

The configuration of a deterministic Turing machine of space $S(n)$ can be described in space $O(S(n))$ (tape contents; head locations; state). Hence, we can represent the Turing machine by a graph. Nodes of this graph are all the possible configurations, where the graph will have $2^{O(S(n))}$ many vertices. The transition function maps one configuration to the next, hence this is a directed graph with out-degree at most 1. The following is a warm-up.

Claim 3.1. *The following language is in L :*

$$\text{CONN-DEG1} = \{(G, s, t) : G \text{ directed graph with out-degree } \leq 1, \text{ with a path from } s \text{ to } t\}.$$

Proof. Assume G is given as an adjacency matrix. We can in log-space scan the input and for a vertex v find if it has an outgoing edge to another vertex u or not. Let $v = s$ be the start vertex. Iteratively move v to the next vertex until either: $v = t$ (in which case we accept), v has no outgoing edge (reject), we made more than n steps (run into a loop, reject). \square

Theorem 3.2. *The following language is NL-complete:*

$$\text{CONN} = \{(G, s, t) : G \text{ directed graph with a path from } s \text{ to } t\}.$$

Proof. Lets show first that $\text{CONN} \in \text{NL}$. The “proof” is a description of a path from s to t , which is a sequence of nodes $p = (v_0, v_1, \dots, v_m)$. The verifier on input G, s, t, p does the following:

1. Verify that $v_0 = s$ and $v_m = t$.
2. For $i = 0, \dots, m - 1$, verify that the edge (v_i, v_{i+1}) is in G .

Note that the verifier can be implemented in log-space, as each verification step can be done in $O(\log n)$ space, as well as the counter over i .

Next, we show that CONN is NL-hard. Fix a language $A \in \text{NL}$. Let x be an input for which we want to check whether $x \in A$. We will build a function $f(x) = (G, s, t)$ running in log-space so that $x \in L \iff f(x) \in \text{CONN}$. Recall that by definition, $A \in \text{NL}$ means that there exists a NTM M running in space $O(\log |x|)$ that computes L . We may assume that when M accepts an input, its head is in the start position and the work tape is empty, so that there is just one accepting configuration.

Define a graph $G = G_x$ which describes its configuration graph:

- **Vertices:** The vertices correspond to configurations of M . They include description of the tape head locations, the content of the work tape, and the state. These can be described using $O(\log |x|)$ bits.
- **Edges:** Given two nodes u, v , there is a direct edge $u \rightarrow v$ if the NTM M can reach v from u using one of its two transition functions.

Note that G has out-degree ≤ 2 . Let s be the node corresponding to the initial configuration, t the accepting configuration. A path in G from s to t corresponds to a computation path of M that accepts x . Moreover, we can compute the vertices (or specific bits in them) and edges (i.e. adjacency matrix) in log-space, by simulating one step of M . So, the reduction is in log-space. \square

The connectivity question makes sense also for undirected graphs. Surprisingly, this can be solved in log-space! The algorithm to do so is quite involved, and builds on beautiful math of expander graphs.

Theorem 3.3 (Reingold [2]). *The following language is in L:*

$$\text{CONN-UNDIRECTED} = \{(G, s, t) : G \text{ undirected graph with a path from } s \text{ to } t\}.$$

4 PSPACE complete language

A quantified boolean formula (QBF) is a formula of the form

$$\psi = Q_1x_1Q_2x_2\dots Q_nx_n\varphi(x_1,\dots,x_n)$$

where $Q_i \in \{\exists, \forall\}$ and φ is a CNF formula. This extends SAT or UNSAT, which allow for only one quantifier. Define the language:

$$\text{TQBF} = \{\text{true QBF formulas}\}.$$

Theorem 4.1. *TQBF is PSPACE-complete under poly-time reductions (and even under log-space reductions).*

Proof. We first show $\text{TQBF} \in \text{PSPACE}$. Given a QBF $\psi = Q_1x_1\dots Q_nx_n\varphi(x_1,\dots,x_n)$ define

$$f_i(\psi; a_1, \dots, a_i) = Q_{i+1}x_{i+1}\dots Q_nx_n\varphi(a_1, \dots, a_i, x_{i+1}, \dots, x_n).$$

This is a “partial assignment” to some of the inputs. We want to compute $\psi = f_0(\psi)$, we will do so inductively. If $Q_{i+1} = \exists$ then $f_i(\psi; a_1, \dots, a_i) = f_{i+1}(\psi; a_1, \dots, a_i, 0) \vee f_{i+1}(\psi; a_1, \dots, a_i, 1)$, and if $Q_{i+1} = \forall$ then $f_i(\psi; a_1, \dots, a_i) = f_{i+1}(\psi; a_1, \dots, a_i, 0) \wedge f_{i+1}(\psi; a_1, \dots, a_i, 1)$. Assume we have a Turing machine M_i computing f_i in space S_i . Then M_i runs M_{i+1} twice (using the same memory), plus it needs $O(1)$ overhead memory for bookkeeping. Clearly f_n is computable in P and in particular in PSPACE. Hence f_0 is computable in PSPACE.

We next argue that TQBF is PSPACE-hard. Let $L \in \text{PSPACE}$. Let G be the configuration graph of L , where we can think of the input as part of the configuration if we wish. Each vertex of G is represented by $m = \text{poly}(n)$ bits. Define

$$\psi_i(u, v) = \text{there is a path from } u \text{ to } v \text{ of length } \leq 2^i.$$

Let s, t be the vertices representing the start and end configuration. We would like to compute $\psi_m(u, v)$. A simple way to do so is by using

$$\psi_{i+1}(u, v) = \exists w, \psi_i(u, w) \wedge \psi_i(w, v).$$

However, this will produce a formula of exponential size. To get a formula of polynomial size, we use the trick

$$\psi_{i+1}(u, v) = \exists w \forall a, b((a = u \wedge b = w) \vee (a = w \wedge b = v)) \implies \psi_i(a, b).$$

Clearly we can transfer ψ_i to ψ_{i+1} in polynomial time. With some care, we can make this into a CNF. We some more care, this entire operation can actually be done in log-space. \square

5 Nondeterministic vs deterministic space

It is conjectured that $NL = LOGSPACE$. The following theorems provide some partial answers.

Theorem 5.1 (Savitch [3]). $NSPACE \subset SPACE(S^2(n))$. In particular, $NPSPACE = PSPACE$ and $NL \subset SPACE(\log^2 n)$.

Proof. This is a stripped down version of the proof that $TQBF \in PSPACE$. Let $L \in NSPACE(S)$. Let $x \in \{0, 1\}^n$ be a potential input for which we want to decide if $x \in L$. Let G be the configuration graph of a Turing machine deciding L on x . Membership in L is equivalent to checking if there is a directed path in G between vertices s, t . Define

$$\psi_i(u, v) = \text{there is a path between } u \text{ and } v \text{ of length } \leq 2^i.$$

Then

$$\psi_{i+1}(u, v) = \exists w, \psi_i(u, w) \wedge \psi_i(w, v).$$

In order to compute ψ_{i+1} we run ψ_i twice, but since we run it sequentially both applications can use the same space. We need additional space $S(n)$ to enumerate the value of w . Note that as $|G| \leq 2^{S(n)}$ then there is a path between s, t if $\psi_{S(n)}(s, t) = 1$. This takes $O(S^2(n))$ space to compute. \square

Let

$$UNCONN = \{(G, s, t) : \text{there is no directed path in } G \text{ from } s \text{ to } t\}$$

We can define $coNL$ (languages which can be refuted in log-space) and get that $UNCONN$ is $coNL$ -complete. It was a surprise when it was proven in the late 80s that NL is closed under complementation, namely that $NL = coNL$.

Theorem 5.2 (Immerman [1] and Szelepcsényi [4]). $UNCONN \in NL$. Hence $NL = coNL$.

Proof. Recall that by definition, a language L is in nondeterministic log-space if there exists a NTM which computes L and runs in log-space. For the proof it will be easier to consider an equivalent model: $L \in NL$ if there exists a TM M and $c > 0$ such that

(i) $x \in L \iff \exists y, |y| \leq |x|^c, M(x, y) = 1$

(ii) $M(x, y)$ uses $O(\log |x|)$ space

(iii) y is given in a separate tape (witness tape) on which the head can only move to the right. That is, y is read one symbol at a time, and the head can never return to previously read symbols.

The equivalence proof is basically as follows: y describes the sequence of choices of which transition function of the NTM to take.

Let G be a directed graph and s, t vertices in G . Define

$$C_i = \{\text{vertices reachable from } s \text{ in at most } i \text{ steps}\}.$$

We already know that there is a witness that $v \in C_i$ which can be verified in log-space: a list of vertices v_0, \dots, v_j for $j \leq i$ such that $v_0 = s, v_j = v$ and each two nodes in the path are adjacent. Note that indeed, this witness can be verified in log-space while being read from left to right. The length of this witness is $O(n \log n)$.

We will show that there exist more complicated witnesses, still verifiable in space $O(\log n)$ and in a read-once left-to-right fashion, for:

- (a) A witness that $v \notin C_i$, given that we know $|C_i|$.
- (b) A witness for $|C_i|$, given that we know $|C_{i-1}|$.

Detailed description:

(a) A witness for $v \notin C_i$ given that we know $|C_i|$ is the following: for each $u \in C_i$, in increasing order, we give the witness that $u \in C_i$. The verifier then verifies that:

- The nodes u are in increasing order;
- The witness for each u is correct;
- The number of u encountered equals $|C_i|$.
- The node v was not one of the nodes u given.

Observe that the verifier needs only log-space, and reads the witness from left to right. The length of the witness is $O(|C_i| \cdot n \log n) = O(n^2 \log n)$.

(b) A witness for $|C_i|$ given $|C_{i-1}|$ is the following: for each vertex $v \in G$ in order,

- Whether $v \in C_i$ or not;
- If $v \in C_i$, a node w such that $(w, v) \in G$ and a witness that $w \in C_{i-1}$.
- If $v \notin C_i$, for every node w such that $(w, v) \in G$, a witness that $w \notin C_{i-1}$.

Observe that the verifier needs only log-space, and reads the witness from left to right. The length of the witness is $O(n^3 \log n)$.

The final proof is a concatenation of all the relevant witnesses. That is: first, a witness that $|C_0| = 1$; then a witness for $|C_1|$ given that we know $|C_0|$, then a witness for $|C_2|$ given that we know $|C_1|$, and so on. At the end, we add a witness that $t \notin C_n$ given that we know $|C_n|$. The total witness length is then $O(n^4 \log n)$, and it can be verified in log-space. \square

References

- [1] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on computing*, 17(5):935–938, 1988.
- [2] O. Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):17, 2008.
- [3] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.
- [4] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.