Notes for Lecture 2

1 Space-Bounded Complexity Classes

A machine solves a problem using space $s(\cdot)$ if, for every input x, the machine outputs the correct answer and uses only the first $s(|x|)$ cells of the tape. For a standard Turing machine, we can't do better than linear space since x itself must be on the tape. So we will often consider a machine with multiple tapes: a read-only "input" tape, a read/write "work" or "memory" tape, and possibly a write-once "output" tape. Then we can say the machine uses space s if for input x, it uses only the first $s(|x|)$ cells of the work tape.

We denote by $\l$ the set of decision problems solvable in $O(\log n)$ space. We denote by **PSPACE** the set of decision problems solvable in polynomial space. A first observation is that a space-efficient machine is, to a certain extent, also a time-efficient one. In general we denote by **SPACE**$(s(n))$ the set of decision problems that can be solved using space at most $s(n)$ on inputs of length n.

Theorem 1 *If a machine always halts, and uses $s(\cdot)$ space, with $s(n) \geq \log n$, then it runs in time $2^{O(s(n))}$.*

Proof: Call the "configuration" of a machine M on input x a description of the state of M, the position of the input tape, and the contents of the work tape at a given time. Write down $c_1, c_2, \ldots, c_t$ where c_i is the configuration at time i and t is the running time of $M(x)$. No two c_i can be equal, or else the machine would be in a loop, since the c_i completely describes the present, and therefore the future, of the computation. Now, the number of *possible* configurations is simply the product of the number of states, the number of positions on the input tape, and the number of possible contents of the work tape (which itself depends on the number of allowable positions on the input tape). This is

$$O(1) \cdot n \cdot |\Sigma|^{s(n)} = 2^{O(s(n))+\log n} = 2^{O(s(n))}$$

Since we cannot visit a configuration twice during the computation, the computation must therefore finish in $2^{O(s(n))}$ steps. □

NL is the set of decision problems solvable by a non-deterministic machine using $O(\log n)$ space. **NPSPACE** is the set of decision problems solvable by a non-deterministic machine using polynomial space. In general we denote by **NSPACE**$(s(n))$ the set of decision problems that can be solved by non-deterministic machines that use at most $s(n)$ bits of space on inputs of length n.

Analogously with time-bounded complexity classes, we could think that **NL** is exactly the set of decision problems that have "solutions" that can verified in log-space. If so, **NL** would be equal to **NP**, since there is a log-space algorithm V that verifies solutions to SAT. However, this is unlikely to be true, because **NL** is contained in **P**. An intuitive reason why not all problems with a log-space "verifier" can be simulated in **NL** is that an **NL** machine does not have enough memory to keep track of all the non-deterministic choices that it makes.

Theorem 2 $\mathbf{NL} \subseteq \mathbf{P}$.

PROOF: Let L be a language in **NL** and let M be a non-deterministic log-space machine for L. Consider a computation of $M(x)$. As before, there are $2^{O(s(n))} = n^{O(1)}$ possible configurations. Consider a directed graph in which vertices are configurations and edges indicate transitions from one state to another which the machine is allowed to make in a single step (as determined by its δ). This graph has polynomially many vertices, so in polynomial time we can do a depth-first search to see whether there is a path from the initial configuration that eventually leads to acceptance. This describes a polynomial-time algorithm for deciding L, so we're done. □

2 Reductions in NL

We would like to introduce a notion of completeness in **NL** analogous to the notion of completeness that we know for the class **NP**. A first observation is that, in order to have a meaningful notion of completeness in **NL**, we cannot use polynomial-time reductions, otherwise any **NL** problem having at least a YES instance and at least a NO instance would be trivially **NL**-complete. To get a more interesting notion of **NL**-completeness we need to turn to weaker reductions. In particular, we define *log space* reductions as follows:

Definition 1 *Let A and B be decision problems. We say A is log space reducible to B, $A \leq_{\log} B$, if $\exists$ a function f computable in log space such that $x \in A$ iff $f(x) \in B$, and $B \in L$.*

Theorem 3 *If $B \in \mathbf{L}$, and $A \leq_{\log} B$, then $A \in \mathbf{L}$.*

PROOF: We consider the concatenation of two machines: M_f to compute f, and M_B to solve B. If our resource bound was polynomial time, then we would use $M_f(x)$ to compute $f(x)$, and then run M_B on $f(x)$. The composition of the two procedures would given an algorithm for A, and if both procedures run in polynomial time then their composition is also polynomial time. To prove the theorem, however, we have to show that if M_f and M_B are log space machines, then their composition can also be computed in log space.

Recall the definition of a Turing machine M that has a log space complexity bound: M has one read-only input tape, one write-only output tape, and uses a log space work tape. A naive implementation of the composition of M_f and M_B would be to compute $f(x)$, and then run M_B on input $f(x)$; however $f(x)$ needs to be stored on the work tape, and this implementation does not produce a log space machine. Instead we modify M_f so that on input x and i it returns the i-th bit of $f(x)$ (this computation can still be carried out in logarithmic space). Then we run a simulation of the computation of $M_B(f(x))$ by using the modified M_f as an "oracle" to tell us the value of specified positions of $f(x)$. In order to simulate $M_B(f(x))$ we only need to know the content of one position of $f(x)$ at a time, so the simulation can be carried with a total of $O(\log |x|)$ bits of work space. □

Using the same proof technique, we can show the following:

Theorem 4 *if $A \leq_{\log} B, B \leq_{\log} C$, then $A \leq_{\log} C$.*

3 NL Completeness

Armed with a definition of log space reducibility, we can define **NL**-completeness.

Definition 2 *A is* **NL***-hard if* $\forall B \in \mathbf{NL}, B \leq_{\log} A$. *$A$ is* **NL***-complete if* $A \in \mathbf{NL}$ *and A is* **NL***-hard.*

We now introduce a problem STCONN (s,t-connectivity) that we will show is **NL**-complete. In STCONN, given in input a directed graph $G(V, E)$ and two vertices $s, t \in V$, we want to determine if there is a directed path from s to t.

Theorem 5 *STCONN is* **NL***-complete.*

Proof:

1. STCONN $\in$ **NL**.

   On input *G(V,E), s,t*, set p to s. For $i = 1$ to $|V|$, nondeterminsitically, choose a neighboring vertex v of p. Set $p = v$. If $p = t$, accept and halt. Reject and halt if the end of the *for* loop is reached. The algorithm only requires $O(\log n)$ space.

2. STCONN is **NL**-hard.

   Let $A \in \mathbf{NL}$, and let M_A be a non-deterministic logarithmic space Turing Machine for A. On input x, construct a directed graph G with one vertex for each configuration of $M(x)$, and an additional vertex t. Add edges (c_i, c_j) if $M(x)$ can move in one step from c_i to c_j. Add edges (c, t) from every configuration that is accepting, and let s be the start configuration. M accepts x iff some path from s to t exists in G. The above graph can be constructed from x in log space, because listing all nodes requires $O(\log n)$ space, and testing valid edges is also easy.

□

4 Savitch's Theorem

What kinds of tradeoffs are there between memory and time? STCONN can be solved deterministically in linear time and linear space, using depth-first-search. Is there some sense in which this is optimal? Nondeterministically, we can search using less than linear space. Can searching be done deterministically in less than linear space?

We will use Savitch's Theorem to show that STCONN can be solved deterministically in $O(\log^2 n)$, and that every **NL** problem can be solved deterministically in $O(\log^2 n)$ space. In general, if A is a problem that can be solved nondeterministically with space $s(n) \geq \log n$, then it can be solved deterministically with $O(s^2(n))$space.

Theorem 6 *STCONN can be solved deterministically in $O(\log^2 n)$ space.*

PROOF: Consider a graph $G(V, E)$, and vertices s, t. We define a recursive function REACH(u, v, k) that accepts and halts iff v can be reached from u in $\leq k$ steps. If $k = 1$, then REACH accepts iff (u, v) is an edge. If $k \geq 2, \forall w \in V - \{u, v\}$, compute REACH$(u, w, \lfloor k/2 \rfloor)$ and REACH$(w, v, \lceil k/2 \rceil)$. If both accept and halt, accept. Else, reject.

Let $S(k)$ be the worst-case space use of REACH$(\cdot, \cdot, k)$. The space required for the base case $S(1)$ is a counter for tracking the edge, so $S(1) = O(\log n)$. In general, $S(k) = O(\log n) + S(k/2)$ for calls to REACH and for tracking w. So, $S(k) = O(\log k * \log n)$. Since $k \leq n$, the worst-case space use of REACH is $O(\log^2 n)$. □

Essentially the same proof applies to arbitrary non-deterministic space-bounded computations.

Theorem 7 (Savitch's Theorem) *For every function $s(n)$ computable in space $O(s(n))$,* **NSPACE**$(s) =$ **SPACE**$(O(s^2))$

PROOF: We begin with a nondeterministic machine M, which on input x uses $s(|x|)$ space. We define $REACH(c_i, c_j, k)$, as in the proof of Theorem 6, which accepts and halts iff $M(x)$ can go from c_i to c_j in $\leq k$ steps. We compute $REACH(c_0, c_{\text{acc}}, 2^O(s|x|))$ for all accepting configurations c_{acc}. If there is a call of REACH which accepts and halts, then M accepts. Else, M rejects. If REACH accepts and halts, it will do so in $\leq 2^{O(|x|)}$ steps.

Let $S_R(k)$ be the worst-case space used by REACH$(\cdot, \cdot, k)$: $S_R(1) = O(s(n)), S_R(k) = O(s(n)) + S_R(k/2)$. This solves $S_R = s(n) * \log k$, and, since $k = 2^O(s(n))$, we have $S_R = O(s^2(n))$. □

Comparing Theorem 6 to depth-first-search, we find that we are exponentially better in space requirements, but we are no longer polynomial in time.

Examining the time required, if we let $t(k)$ be the worst-case time used by REACH$(\cdot, \cdot, k)$, we see $t(1) = O(n + m)$, and $t(k) = n(2 * T(k/2))$, which solves to $t(k) = n^{O(\log k)} = O(n^{O(\log n)})$, which is super-polynomial. Savitch's algorithm is still the one with the best known space bound. No known algorithm achieves polynomial log space and polynomial time simultaneously, although such an algorithm is known for *undirected* connectivity.

5 Undirected Connectivity

In the undirected $s - t$ connectivity problem (abbreviated ST-UCONN) we are given an undirected graph $G = (V, E)$ and two vertices $s, t \in V$, and the question is whether that is a path between s and t in G.

While this problem is not known to be complete for **NL**, and it probably is not, ST-UCONN is complete for the class **SL** of decision problems that are solvable by *symmetric* non-deterministic machines that use $O(\log n)$ space. A non-deterministic machine is symmetric if whenever it can make a transition from a global state s to a global state s' then the transition from s' to s is also possible. The proof of **SL**-completeness of ST-UCONN is identical to the proof of **NL**-completeness of ST-CONN except for the additional observation that the transition graph of a symmetric machine is undirected.

For ST-UCONN there exists an algorithm running in polynomial time and $O(\log^2 n)$ space (but the polynomial has very high degree), due to Nisan [Nis94]. There is also an

algorithm that has $O(\log^{4/3} n)$ space complexity and superpolynomial time complexity, due to Armoni, Ta-Shma, Nisan and Wigderson [ATSWZ97], improving on a previous algorithm by Nisan, Szemeredy and Wigderson [NSW92].

6 Randomized Log-space

We now wish to introduce *randomized* space-bounded Turing machine. For simplicity, we will only introduce randomized machines for solving decision problems. In addition to a read-only input tape and a read/write work tape, such machines also have a read-only *random* tape to which they have *one-way* access, meaning that the head on that tape can only more, say, left-to-right. For every fixed input and fixed content of the random tape, the machine is completely deterministic, and either accepts or rejects. For a Turing machine M, an input x and a content r of the random tape, we denote by $M(r, x)$ the outcome of the computation.

We say that a decision problem L belongs to the class **RL** (for *randomized* log-space) if there is a probabilistic Turing machine M that uses $O(\log n)$ space on inputs of length n and such that

- For every $x \in L$, $\mathbf{Pr}_r[M(r, x) \text{ accepts }] \geq 1/2$
- For every $x \notin L$, $\mathbf{Pr}_r[M(r, x) \text{ accepts }] = 0$.

It is easy to observe that any constant bigger than 0 and smaller than 1 could be equivalently used instead of 1/2 in the definition above. It also follows from the definition that $\mathbf{L} \subseteq \mathbf{RL} \subseteq \mathbf{NL}$.

The following result shows that, indeed, $\mathbf{L} \subseteq \mathbf{SL} \subseteq \mathbf{RL} \subseteq \mathbf{NL}$.

Theorem 8 *The problem ST-UCONN is in* **RL**.

We will not give a proof of the above theorem, but just describe the algorithm. Given an undirected graph $G = (V, E)$ and two vertices s, t, the algorithm performs a random walk of length $100 \cdot n^3$ starting from s. If t is never reached, the algorithm rejects.

input: $G = (V, E)$, s, t

$v \leftarrow s$
for $i \leftarrow 1$ to $100 \cdot n^3$
    pick at random a neighbor w of v
    if $w = t$ then halt and accept
    $v \leftarrow w$ reject

The analysis of the algorithm is based on the fact that if we start a random walk from a vertex s of an undirected vertex G, then all vertices in the connected component of s are likely to be visited at least once after $\Theta(n^3)$ steps.

The best known deterministic simulation of **RL** uses $O((\log n)^{3/2})$ space, and is due to Saks and Zhou [SZ95].

References

[ATSWZ97] R. Armoni, A. Ta-Shma, A. Wigderson, and S. Zhou. $SL \subseteq L^{\frac{4}{3}}$. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 230–239, 1997.

[Nis94] N. Nisan. RL $\subseteq$ SC. *Computational Complexity*, 4(1), 1994.

[NSW92] N. Nisan, E. Szemeredi, and A. Wigderson. Undirected connectivity in $O(\log^{1.5} n)$ space. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 24–29, 1992.

[SZ95] Michael Saks and Shiyu Zhou. $RSPACE(S) \subseteq DSPACE(S^{3/2})$. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 344–353, 1995.

Exercises

1. It is possible to prove that there is an algorithm that given a Turing machine M (of the type with a read/only input tape and a work tape), an input x and a space bound s, decides whether $M(x)$ accepts and uses $\leq s$ bits of space; the algorithm uses $O(s)$ bits of space. Using the above result, argue that if $s(n) > \log n$ is a function computable in $O(s(n))$ space, then $\mathbf{SPACE}(s(n)\log n) \not\subseteq \mathbf{SPACE}(s(n))$.

2. Prove that $\mathbf{P} \neq \mathbf{SPACE}(O(n))$.

   [This is a "trick" question. It is not know how to prove $\mathbf{P} \not\subseteq \mathbf{SPACE}(O(n))$ or how to prove $\mathbf{SPACE} \not\subseteq \mathbf{P}$, so the proof has to start by assuming the classes are equal and then reach a contradiction, without explicitely showing a problem in one class that cannot belong to the other. Hint: if A and B are decision problems, $A \leq B$ and $B \in \mathbf{P}$ then $A \in \mathbf{P}$; what would happen if the same were true for $\mathbf{SPACE}(O(n))$?]

3. Define the class **BPL** (for *bounded-error probabilistic log-space*) as follows. A decision problem L is in **BPL** if there is a log-space probabilistic Turing machine M such that

   - If $x \in L$ then $\mathbf{Pr}_r[M(r,x) \text{ accepts }] \geq 2/3$;
   - If $x \notin L$ then $\mathbf{Pr}_r[M(r,x) \text{ accepts }] \leq 1/3$.

   Then

   (a) Prove that $\mathbf{RL} \subseteq \mathbf{BPL}$.

   (b) Prove that $\mathbf{BPL} \subseteq \mathbf{SPACE}(O((\log n)^2)$.

   (c) This might be somewhat harder: prove that $\mathbf{BPL} \subseteq \mathbf{P}$.