Computability and Complexity

Lecture 10, Thursday August 31, 2023

Ari Feiglin

Recall that we showed that L is closed under log-space reductions. The proof that we provided can be used to prove the following more general statement:

Proposition 10.1:

For every space function $S(n) \geq \log(n)$, $\mathsf{DSPACE}(O(f(n)))$ and $\mathsf{NSPACE}(O(f(n)))$ are closed under log-space reductions.

Theorem 10.2 (Savitch's Theorem):

$$\mathbf{NL} \subseteq \mathsf{DSPACE}\big(O(\log(n)^2)\big)$$

Proof:

We will show that $\textsf{st-conn} \in \mathsf{DSPACE}\big(O(\log(n)^2)\big)$ and since $\mathsf{DSPACE}\big(O(\log(n)^2)\big)$ is closed under log-space reductions, we get that for every $S \in \mathbf{NL}$, since there exists a log-space reduction from S to st-conn, $S \in \mathsf{DSPACE}\big(\log(n)^2\big)$. So we must formulate an algorithm for solving st-conn in $O(\log(n)^2)$ time.

Given a tuple (G, u, v, k) where G is a graph, u and v are vertices, and k is a natural number, let us define

$$\varphi(G,u,v,k) = \begin{cases} 1 & \text{There exists a path from } u \text{ to } v \text{ whose length is } \leq k \\ 0 & \text{Else} \end{cases}$$

Obviously $(G, s, t) \in \textsf{st-conn}$ if and only if $\varphi(G, u, v, |V|) = 1$. So let us define an algorithm to compute φ:

1. **function** $M(G, u, v, k)$
2. **if** $(k = 1)$ **return** 1 **if** $(u, v) \in E$, **else** 0
3. **for** $(w \in V)$
4. $\sigma_1 \leftarrow M\big(G, u, w, \lceil \frac{k}{2} \rceil\big)$
5. $\sigma_2 \leftarrow M\big(G, w, v, \lfloor \frac{k}{2} \rfloor\big)$
6. **if** $(\sigma_1 = 1$ **and** $\sigma_2 = 1)$ **return** 1
7. **end for**
8. **return** 0
9. **end function**

This computes φ, as if $\varphi(G, u, v, k) = 1$ then there exists a path from u to v whose length is $\leq k$ and if we take the midway point on this path to be w, then $\sigma_1 = 1$ and $\sigma_2 = 1$, so M will return one. If $\varphi(G, u, v, k) = 0$ then there is no path from u to v, so there cannot exist a point whose distance from u is $\leq \lceil \frac{k}{2} \rceil$ and from w is $\leq \lfloor \frac{k}{2} \rfloor$, as then the concatenation of these paths forms a path from u to v whose length is $\leq k$. So if $\varphi(G, u, v, k) = 0$ then M returns zero.

At each level of the iteration, we need three pointers: one to u, one to v, and one to w. Each of these pointers take $\log(n)$ space. This is because after we finish the recursive call on line 4, we can get rid of all the pointers used by the recursion and reuse it for the recursion on line 5. So we get the recursive function for the space complexity

$$S(k) = O(\log(n)) + S\left(\frac{k}{2}\right)$$

Expanding this t times gives

$$S(k) = O(\log(n)) + O(\log(n)) + \cdots + O(\log(n)) + S\left(\frac{k}{2^t}\right) = O(t \cdot \log(n)) + S\left(\frac{k}{2^t}\right)$$

setting $t = \log(k)$ gives

$$S(k) = O(\log(n)\log(k)) + S(1)$$

thus the space complexity of the algorithm is $O(\log(n)\log(k))$ as required.

Note:

The reuse of the space used in line 4 for line 5 is important, as otherwise our space complexity would satisfy

$$S(k) = O(\log(n)) + 2S\left(\frac{k}{2}\right)$$

this is in $O(\log(n)^3)$.

So to determine if $(G, s, t) \in \mathsf{st\text{-}conn}$, we need only to run $M(G, s, t, |V|)$ which takes $O(\log(n) \cdot \log(|V|)) \subseteq O(\log(n)^2)$ time, as required. ■

Theorem 10.3 (Generalized Savitch's Theorem):

For every $S(n) \geq \log(n)$,

$$\mathsf{NSPACE}(S(n)) \subseteq \mathsf{DSPACE}\big(O(S(n)^2)\big)$$

We can use our proof of **Savitch's Theorem** with the graph of configurations of the non-deterministic Turing machine which solves the problem in $O(S(n))$-space and determine the connectivity of the graph. But we will prove this another way.

Proof:

Let $A \in \mathsf{NSPACE}(S(n))$, so there exists a non-deterministic Turing machine N which decides A in $S(n)$ space. Let us define

$$A' = \left\{ x10^{2^{S(|x|)}} \;\middle|\; x \in A \right\}$$

We claim $A' \in \mathbf{NL}$. Let us define the non-deterministic Turing machine $N'(y)$ which functions as follows:

(1) First verify that y is of the form $x10^{2^{S(|x|)}}$, ie. that the string of zeroes at the end has a length of $2^{S(|x|)}$. To do this, we need to count how many zeroes there are at the end of y, so we'd need to count at most $|y|$ bits, meaning the counter whose size is $\log|y|$.

(2) Simulate $N(x)$ and return the answer.

We know $N(x)$ takes $S(|x|)$ space, but notice that

$$|y| = |x| + 1 + 2^{S(|x|)} \geq 2^{S(|x|)} \implies S(|x|) \leq \log|y|$$

and so the space needed to run $N(x)$ is bound by $\log|y|$. So N' requires $O(\log|y|)$ space, meaning it is in **NL**. By **Savitch's Theorem**, there exists a deterministic Turing machine M' which decides A' in $\log(n)^2$ time.

So we define $M(x)$ which simulates $M'\big(x10^{2^{S(|x|)}}\big)$. M then obviously decides A', and it requires

$$O\left(\left(\log\left|x10^{2^{S(|x|)}}\right|\right)^2\right) = O\left(\log\left(|x| + 1 + 2^{S(|x|)}\right)^2\right) = O\left(\log\left(2^{3S(|x|)}\right)^2\right) = O(S(|x|)^2)$$

space, as required.

There is actually a nuance here, since in order to simulate $M'\big(x10^{2^{S(|x|)}}\big)$, we must somehow access the string $x10^{2^{S(|x|)}}$, whose length is exponential in $S(|x|)$. But we need not actually store $x10^{2^{S(|x|)}}$ on the working tape, instead we can store a pointer to where M' is pointer, and from this we can determine whether to simulate a bit in x (if the pointer is pointing at x), or 1 (if the pointer is at index $|x|+1$), or 0 (if the pointer is between $|x|+2$ and $|x|+1+2^{S(|x|)}$), or empty (otherwise). So maximum value of this pointer is $|x|+2+2^{S(|x|)}$ which is bound by $4 \cdot 2^{S(|x|)}$, which can be stored in $O(S(|x|))$ bits.

So all in all, we only need $O(S(n)^2) + O(S(n)) = O(S(n)^2)$ space to run M, as required. ■

Definition 10.4:

Similar to how a deterministic Turing machine can compute a function f, we say that a non-deterministic Turing machine M computes a function f if

(1) There exists a run of M on x such that $M(x) = f(x)$.

(2) For all other runs of M on x, $M(x) = \perp$.

Theorem 10.5 (Immerman–Szelepcsenyi Theorem):

$$\mathbf{NL} = \mathsf{coNL}$$

Proof:

We will show that $\mathsf{st\text{-}conn} \in \mathsf{coNL}$, this means that $\mathbf{NL} \subseteq \mathsf{coNL}$ (since coNL is closed under log-space reductions, for similar reasons as before). This means that $\mathbf{NL} = \mathsf{coNL}$ (since $\mathcal{C} \subseteq \mathsf{co}\mathcal{C}$ implies $\mathcal{C} = \mathsf{co}\mathcal{C}$, since if $S \in \mathsf{co}\mathcal{C}$ then $S^c \in \mathcal{C} \subseteq \mathsf{co}\mathcal{C}$ so $S \in \mathcal{C}$).

Our first goal is find a non-deterministic Turing machine which computes the following function in logarithmic space:

$$\mathsf{Total\text{-}Reach}(G, s) = \text{The number of vertices in } G \text{ which can be reached from the vertex } s$$

If we can find a log-space non-deterministic Turing machine M_{TR} which computes $\mathsf{Total\text{-}Reach}$, then we claim that $\mathsf{st\text{-}conn}^c \in \mathbf{NL}$ (ie. $\mathsf{st\text{-}conn} \in \mathsf{coNL}$). This is since we can define the following non-deterministic algorithm:

1. **function** $M(G, s, t)$
2. $\quad c_1 \leftarrow M_{\mathsf{TR}}(G, s)$
3. $\quad$ Construct the graph G' which is obtained by removing the vertex t from G
4. $\quad c_2 \leftarrow M_{\mathsf{TR}}(G', s)$
5. $\quad$ **if** ($c_1 = \perp$ **or** $c_2 = \perp$ **or** $c_1 \neq c_2$) **return** 0
6. $\quad$ **else return** 1
7. **end function**

This decides $\mathsf{st\text{-}conn}^c$ as if $(G, s, t) \in \mathsf{st\text{-}conn}^c$ then s and t are disconnected so removing t from G does not impact the number of vertices reachable from s, so when M_{TR} returns the proper value (and not $\perp$), $c_1 = c_2$ and so M returns one. Otherwise if $(G, s, t) \in \mathsf{st\text{-}conn}$ then removing t *does* impact the number of vertices reachable from s, so either $c_1 = \perp$ or $c_2 = \perp$ or $c_1 \neq c_2$. So M does indeed decide $\mathsf{st\text{-}conn}^c$. And if M_{TR} runs in logarithmic space, then so too does M. Thus $\mathsf{st\text{-}conn} \in \mathsf{coNL}$.

So we now must find M_{TR}. For every $i \geq 0$, let us define R_i to be the set of vertices in G which are reachable by s within a path whose length is at most i. Notice that

$$R_0 = \{s\}$$
$$R_{i+1} = R_i \cup \{u \mid \exists w \colon (w, u) \in E\}$$

In the end we want a non-deterministic algorithm which computes $\left|R_{|V|}\right|$. We will do this by first defining an algorithm which determines the size of R_i given the size of R_{i-1}.

1. **function** $\mathsf{R_{next}}(G, s, R_{i-1})$
2. $\quad$ **choose** $1 \leq g \leq |V|$
   $\quad \triangleright$ *Suppose we can determine if* $|R_i| \geq g$ *and if* $|R_i| \leq g$*, then we can continue with this algorithm,*
3. $\quad$ **if** ($|R_i| \geq g$ **and** $|R_i| \leq g$) **return** g
4. $\quad$ **return** $\perp$
5. **end function**

Obviously assuming that we can determine if $|R_i| \geq g$ and if $|R_i| \leq g$, then $\mathsf{R_{next}}$ will compute $|R_{i+1}|$. So we must demonstrate that we can determine if $|R_i| \geq g$ in logarithmic space. We do this as follows

1. *counter* $\leftarrow 0$
2. **for** $(v \in G)$
3. $\quad$ **if** there exists a path from s to v whose length is $\leq i$: *counter* $\leftarrow$ *counter* $+ 1$
4. **end for**
5. **return** 1 **if** *counter* $\geq g$, **else** 0

We can determine non-deterministically if there exists a path from s to v whose length is at most i similar to how we did for $\mathsf{st\text{-}conn}$: we find neighbors of the current node and if within i iterations we reach v, return one and otherwise

return zero. If $|R_i| \geq g$ then as we iterate over $v \in R_i$ and as we find paths from s to v whose length is bound by i, then *counter* will be equal to $|R_i|$, so this will return one. If $|R_i| < g$ then since the maximum value *counter* can equal is $|R_i|$, this will return zero. So this algorithm works for determining if $|R_i| \geq g$ non-deterministically.

Note:

Just because we can determine if $|R_i| \geq g$ non-deterministically doesn't mean we can determine if $|R_i| \leq g$. While it is true that

$$|R_i| \leq g \iff \neg\big(|R_i| \geq g+1\big)$$

simply returning the inverse of the algorithm above will not work. This is because if $|R_i| \geq g+1$ then this algorithm will sometimes return one, but it may also return zero. And then negating this means that we will return one even though $|R_i|$ is not bound by g.

In order to determine if $|R_i| \leq g$, we will determine if $|R_i^c| \geq |V| - g$. We do this by

1. $counter_1 \leftarrow 0$
2. **for** $(v \in V)$
3. $\quad counter_2 \leftarrow 0$
4. $\quad$ **for** $(u \in V,\, u \neq v)$
5. $\quad\quad$ **if** there exists a path from u to s whose length is $\leq i-1$, and $(v,u) \notin E$: $\;counter_2 \leftarrow counter_2 + 1$
   $\quad\quad \triangleright$ *This counts the number of vertices u found in R_{i-1}*
6. $\quad$ **end for**
7. $\quad$ **if** $(counter_2 = |R_{i-1}|)$ $\;counter_1 \leftarrow counter_1 + 1$
8. **end for**
9. **return** 1 **if** $counter_1 \geq |V| - g$, **else** 0

If $|R_i^c| \geq |V| - g$ then then for every $v \notin R_i$, for every $u \neq v$, if u is connected to s by a path of length $\leq i-1$, $(v,u) \notin E$. So if when iterating over $u \in R_{i-1}$ we get 1 (meaning there exists a path of length $\leq i-1$, this is non-deterministic though) and for every other u it will return zero, and so we get that $counter_2 = |R_{i-1}|$ and so $counter_1$ will become equal to the number of vertices not in R_i, ie $|R_i^c| \geq |V| - g$ so this will return one. And if $|R_i^c| < |V| - g$ then since the maximum value for $counter_1$ which is computable is $|R_i^c|$, this will return zero.

This also takes logarithmic space, as we need only to store two counters, two vertices, and the space required to find a path. All of these take logarithmic space.

Now we define M_{TR}:

1. **function** $M_{\mathsf{TR}}(G, s)$
2. $\quad R \leftarrow 1$ $\quad \triangleright$ *since* $R_0 = \{s\}$
3. $\quad$ **for** $(1 \leq i \leq |V|)$
4. $\quad\quad R \leftarrow \mathsf{R}_{\mathsf{next}}(G, s, R)$ $\quad \triangleright$ *computes R_i from R_{i-1}*
5. $\quad\quad$ **if** $(R = \perp)$ **return** $\perp$
6. $\quad$ **end for**
7. $\quad$ **return** R $\quad \triangleright$ *returns* $\left|R_{|V|}\right|$
8. **end function**

Then since at every iteration of i, $R = R_i$ or $\perp$ since $\mathsf{R}_{\mathsf{next}}(G, s, |R_{i-1}|) = |R_i|$ or $\perp$, if $\mathsf{R}_{\mathsf{next}}$ always returns a non-$\perp$ value, we will get $\left|R_{|V|}\right|$ at the end. Otherwise it will return $\perp$. So M_{TR} computes $\mathsf{Total\text{-}Reach}(G, s)$ as required. ■

Example 10.6:

We will show that $\mathsf{2Color} \in \mathbf{NL}$ by showing that $\mathsf{2Color}^2 \in \mathbf{NL}$. From graph theory we know that G is two-colorable if and only if every cycle in G has an even length. So we will define an algorithm which determines if the graph has an odd length cycle, which would mean it is in $\mathsf{2Color}^c$. We can do this by choosing a vertex v and finding if there exists a cycle containing v.

1. **function** $M(G)$
2. $\quad$ **choose** $v \in V$
3. $\quad c \leftarrow v$
4. $\quad i \leftarrow 1$
5. $\quad$ **while** $(i \leq |V| + 1)$
6. $\quad\quad$ **choose** $c \in N(c)$

```
        i ← i + 1
        if (c = v) return 1 if i is odd, else 0
    end while
    return 0
end function
```

If there exists an odd-length cycle, given the correct choices M will find it and return one. Otherwise M will always return zero. So M decides $\mathsf{2Color}^c$ in logarithmic space, meaning $\mathsf{2Color}^c \in \mathbf{NL}$ so $\mathsf{2Color} \in \mathbf{NL}$.

Definition 10.7:

Let $S(n)$ be a space function, we define the class $\mathsf{NSPACE}_{\mathsf{offline}}(S(n))$ to be the set of all decision problems A such that there exists a *deterministic* Turing machine M which has spatial complexity of $S(n)$ and a fourth tape, called the "witness tape" which is read-only such that $x \in A$ if and only if there exists a y such that $M(x, y) = 1$.

By $M(x, y)$ we mean running M on an input x and a witness y. The witness is not viewed as input (it is on a different tape), so we do not take it into account when computing the spatial complexity. So $M(x, y)$ takes $S(|x|)$ space.

Theorem 10.8:

For every space function $S(n) \geq \log(n)$,

(1) $\mathsf{NSPACE}(S(n)) \subseteq \mathsf{NSPACE}_{\mathsf{offline}}\big(O(\log(S(n)))\big)$

(2) $\mathsf{NSPACE}_{\mathsf{offline}}(S(n)) \subseteq \mathsf{NSPACE}\big(2^{O(S(n))}\big)$

Proof:

(1) Suppose $A \in \mathsf{NSPACE}(S(n))$, so there exists a non-deterministic Turing machine N which decides A and whose spatial complexity is $S(n)$. Let us define a deterministic Turing machine M whose witness is a sequence of configurations of N, $c_1 \cdots c_k$ which returns 1 if and only if

(i) c_1 is the initial configuration of $N(x)$

(ii) c_k is an accepted configuration.

(iii) For every i, c_{i+1} can be reached by c_i.

So $N(x) = 1$ if and only if there is a sequence of configurations y such that $M(x, y) = 1$, so M decides A as required.

To run this M, we need only to store pointers to two consecutive configurations in order to verify iii. We can use these pointers to also verify i and ii. Since each configuration has a size of $O(S(n))$, the total size needed is $O(\log(S(n)))$ as required.