COM S 6810 Theory of Computing January 29, 2009

Lecture 4: Space Complexity II

Instructor: Rafael Pass *Scribe: Shuang Zhao*

1 Recall from last lecture

Definition 1 (SPACE, NSPACE)

$$\mathrm{SPACE}(S(n)) := \textit{Languages decidable by some TM using } S(n) \textit{ space;}$$
$$\mathrm{NSPACE}(S(n)) := \textit{Languages decidable by some NTM using } S(n) \textit{ space.}$$

Definition 2 (PSPACE, NPSPACE)

$$\mathrm{PSPACE} := \bigcup_{c>1} \mathrm{SPACE}(n^c), \quad \mathrm{NPSPACE} := \bigcup_{c>1} \mathrm{NSPACE}(n^c).$$

Definition 3 ($\mathcal{L}$, $\mathcal{NL}$)

$$\mathcal{L} := \mathrm{SPACE}(\log n), \quad \mathcal{NL} := \mathrm{NSPACE}(\log n).$$

Theorem 1

$$\mathrm{SPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}\left(2^{O(S(n))}\right).$$

This immediately gives that $\mathcal{NL} \subseteq \mathrm{P}$.

Theorem 2 STCONN *is $\mathcal{NL}$-complete.*

2 Today's lecture

Theorem 3

$$\mathcal{NL} \subseteq \mathrm{SPACE}(\log^2 n),$$

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

Proof. Recall the STCONN problem: *given digraph (V, E) and $s, t \in V$, determine if there exists a path from s to t.*

Define boolean function $\textsc{Reach}(u, v, k)$ with $u, v \in V$ and $k \in \mathbb{Z}_+$ as follows: if there exists a path from u to v with length smaller than or equal to k, then $\textsc{Reach}(u, v, k) = 1$; otherwise $\textsc{Reach}(u, v, k) = 0$. It is easy to verify that there exists a path from s to t iff $\textsc{Reach}(s, t, |V|) = 1$.

Next we show that $\textsc{Reach}(s, t, |V|)$ can be recursively computed in $O(\log^2 n)$ space. For all $u, v \in V$ and $k \in \mathbb{Z}_+$:

- $k = 1$: $\textsc{Reach}(u, v, k) = 1$ iff $(u, v) \in E$;
- $k > 1$: $\textsc{Reach}(u, v, k) = 1$ iff there exists $w \in V$ such that $\textsc{Reach}(u, w, \lceil k/2 \rceil) = 1$ and $\textsc{Reach}(w, v, \lfloor k/2 \rfloor) = 1$.

Space cost. For the base level ($k = 1$), whether $(u, v) \in E$ can be checked in $O(\log n)$ space. For every other level ($k > 1$), it takes $O(\log n)$ space to enumerate and store each possible w. Since k reduces by half, the number of levels never exceeds $\lceil \log |V| \rceil \leq \lceil \log n \rceil$. Therefore, $\textsc{Reach}(s, t, |V|)$ can be computed in $O(\log^2 n)$ space.

Time cost. Consider the recursive computation of $\textsc{Reach}$ as depth-first search. The branch factor of the search tree is $|V| = O(n)$ (since we enumerate w among all elements of V) and the depth of the tree is $\lceil \log |V| \rceil = O(\log n)$. The time cost on each tree node is polynomial to n. So the time complexity for computing $\textsc{Reach}$ is $n^{O(\log n)}$. ■

Open problems:

- Can we (deterministically) solve STCONN using less space?
- When using $O(\log^2 n)$ space, can we (deterministically) solve STCONN in less time?

Let $L \in \text{NSPACE}(S(n))$ be any language decided by an NTM M in $O(S(n))$ space. Then for all $x \in \{0, 1\}^*$, the configuration graph G of M on input x has no more than $2^{O(S(n))}$ vertices. And using this recursive approach, we can check if there exists a path from the starting configuration to the accept configuration in $O(S(n)^2)$ space. Therefore,

Theorem 4 (Savitch) *For any space-constructible $S : \mathbb{N} \to \mathbb{N}$ with $S(n) \geq \log n$,* $\text{NSPACE}(S(n)) \subseteq \text{SPACE}(S(n)^2)$.

And it follows that PSPACE = NPSPACE.

Definition 4 ($\text{co}\mathcal{NL}$)

$$\text{co}\mathcal{NL} := \{L \mid \bar{L} \in \mathcal{NL}\}.$$

Definition 5 *Let f be a function and M be an NTM. We say "M computes f" iff for all $x \in \{0, 1\}^*$ and all nondeterministic choices, $M(x) \in \{f(x), \perp\}$ (in which $\perp$ stands for 'reject'); for all $x \in \{0, 1\}^*$, there exists a choice under which $M(x)$ outputs $f(x)$.*

Theorem 5 (Immerman-Szlepcsenyi)

$$\mathcal{NL} = \text{co}\mathcal{NL}.$$

To prove Theorem 5, we show that $\overline{\text{STCONN}} \in \mathcal{NL}$.

Given digraph $G = (V, E)$ and $s \in V$, define function $\#\textsc{Reach}(G, s, k)$ in which $s \in V$, $k \in \mathbb{Z}_+$ by the number of vertices in V reachable from s in no more than k steps. Let $G' = (V', E')$ be the graph obtained by removing vertex t and all the incident edges from G. Then it holds that there is no path from s to t iff

$$\#\textsc{Reach}(G, s, |V|) = \#\textsc{Reach}(G', s, |V'|).$$

Define NTM M: if $\#\textsc{Reach}(G, s, |V|) = \#\textsc{Reach}(G', s, |V'|) \neq \perp$, accept; otherwise reject.

Question: how to compute $\#\textsc{Reach}(G, s, k)$ in logarithmic space?

First approach.

1: $c \leftarrow 0$
2: **for all** $v \in V$ **do**
3: guess nondeterministically if v is reachable from s in no more than k steps
4: **if** so, **then**
5: $p \leftarrow s$
6: **for** $i = 1$ to k **do**
7: $p \xleftarrow{\text{nondeterministic}}$ a neighbor of p or p itself
8: **end for**
9: **if** $p \neq v$ **then** reject
10: $c \leftarrow c + 1$
11: **end if**
12: **end for**
13: **return** c

Unfortunately, this approach does NOT work, because it may output incorrect answers which are smaller than the correct one (because the nondeterministic guess may choose 'No' for some vertices to which there actually exists a path from s).

Better approach. For all $v \in V$, there exists an $s - v$ path whose length is no more than k iff $\exists w \in V$ such that there exists an $s - w$ path with length at most $k - 1$ and (i) $w = v$ or (ii) $(w, v) \in E$. To check existence of this $s - w$ path, we can first recursively compute

$$c_{k-1} := \#\textsc{Reach}(G, s, k - 1),$$

then for all $v' \in V$, nondeterministically choose a path $P_{v'}$ from s with length $k - 1$. If P passes v', increase a counter by 1. If the value of the counter equals c_{k-1} after

enumerating all $v' \in V$ and P_w passes w, it holds that there exists a path from s to w with length at most $k-1$.

The pseudo code is shown as follows.

```
1: c_{k-1} ← #REACH(G, s, k − 1)
2: c ← 0
3: for all v ∈ V do
4:    d ← 0, flag ← false
5:    for all w ∈ V do
6:       p ← s
7:       for i = 1 to k − 1 do
8:          p ←(nondeterministic)— a neighbor of p or p itself
9:       end for
10:      if p = w then
11:         d ← d + 1
12:         if v = w or v is adjacent to w then flag ← true
13:      end if
14:   end for
15:   if d < c_{k−1} then reject
16:   if flag then c ← c + 1
17: end for
18: return c
```