15-855: Intensive Intro to Complexity Theory
Spring 2009

Lecture 27 abridged: The $\mathsf{SL} = \mathsf{L}$ algorithm

1 A log-space algorithm

As we've discussed:

Proposition 1.1. $\text{USTCON} \leq_{\mathsf{L}} \text{USTCON}_{\textit{non-bip},d=6}$, *in bag-of-rotation-maps format.*

And, henceforth working in bag-of-rotation-maps format:

Proposition 1.2. $\text{USTCON}_{d=6,\lambda_1 \geq 10^{-4}} \in \mathsf{L}$.

And, we've now seen the idea behind:

Theorem 1.3. $\text{USTCON}_{\textit{non-bip},d=6} \leq_{\mathsf{L}} \text{USTCON}_{d=6,\lambda_1 \geq 10^{-4}}$.

Since (constantly many) log-space reductions compose, this is enough to prove $\text{USTCON} \in L$, as desired. To implement Reingold's algorithm, we just need to go from a bag-of-rotation-maps representation for G_0 to a bag-of-rotation-maps representation for G_ℓ, where $\ell = O(\log n)$. Recall that the vertices G_ℓ are indexed by

$$(v, c_1, \ldots, c_\ell) \in [n] \times ([6]^{1024})^\ell, \tag{1}$$

and the edges are indexed by $[6]$. To output G_ℓ in bag-of-rotation-maps format, we can enumerate over all vertices $\overline{v}$ as in (1) and all edges $i \in [6]$ and output $\text{Rot}_{G_\ell}(\overline{v}, i)$. So to complete the proof, it suffices to show:

Theorem 1.4. $\text{Rot}_{G_\ell}(\overline{v}, i)$ *can be computed* in-place *with* no *extra space.*

For absolute concreteness, we assume the input $(\overline{v}, i)$ is written in a format like the following:

$$v \ \triangleright\ 64413 \ ; \ 23253 \ ; \ 16132 \ \triangleleft \ i \tag{2}$$

Here v is the name of the main vertex $v \in [n]$, written in binary say; each "window", separated by ";" symbols (or $\triangleright$, $\triangleleft$ at the ends), is actually 1024 digits long, not 5; and there are actually ℓ windows, not 3. We will also use 6 additional TM tape symbols: $1^\checkmark, \ldots, 6^\checkmark$.

An important thing to remember is that we can interpret a prefix of (**??**), up to and including the jth window, as the name of a vertex in G_j.

We now give the initial part of the algorithm proving the theorem. At all times, the algorithm keeps its work-tape head in an "active window". This is initially the last window (where i is written).

1. Read i into the "finite control".

2. If the active window is the *first* window,
3. Apply Rot_{G_0} to the v part and i.
4. Else if $i \in \{1,2,3\}$,
5. Apply Rot_H to the window *left* of the active window, and i.
6. Else // $i \in \{4,5,6\}$,
7. Move the active window one window to the left.

Note that lines 2–3 aren't actually useful here (since $\ell \neq 1$), but they're included here for future clarity.

Let's see how this initial part of the algorithm applies to the example (**??**). Suppose first that i is, say, 1. So we are looking to follow edge #1 from vertex $(v; 64413; 23253; 16132)$ in G_3 (and also figure out the resulting "back-edge", since we're computing Rot_{G_3}). Recall that $G_3 = G_2^{1024} \textcircled{r} H$. By convention/definition of replacement product, a 1-edge in this graph is an "inner" edge, within one of the H clouds. So the resulting vertex is just whatever neighbor #1 of vertex 16132 is in H. Given H's cycle-plus-chords nature, this is (probably) vertex 16131, and the back-edge is (probably) 2. In any case, it's whatever $\mathrm{Rot}_H(16132; 1)$ is, so line 5 is doing the correct thing.

The more complicated case is that $i \in \{4,5,6\}$; say for concreteness $i = 5$. By convention/definition of replacement product, this is one an "outer" edge, one of the three parallel edges leading out of the cloud. Note that the ensuing "back-edge" is also named #5, so we don't actually need to overwrite i. But we do need to figure out how to take the step in G_2^{1024}. By definition, what we need to do is interpret 16132 as an edge label in G_2^{1024}, find out where that edge takes you in G_2, and also figure out the name of the "back-edge" — which is really like a "back-path" of length 1024.

In the $i = 5$ case, the above initial part of the algorithm moves the active window one to the left. So the next part of the algorithm should be some code which computes and writes in

$$\mathrm{Rot}_{G_2^{1024}}((v; 64413; 23253); 16132)$$

and then ends.

It makes sense to do this "recursively", but one has to be careful about space usage. If one implements the recursion in the most naive way, it would use $O(\log^2 n)$ space. If one implements it in a more clever way, it might still use space $O(\log n \log\log n)$. For maximum clarity and care, I show how to implement it nonrecursively, in-place:

8. While active window is not the *last* window,
9. If all digits in active window are checkmarked,
10. Uncheck them, reverse the window's contents, and move the active window to the right.
11. Else, // the computation of $\mathrm{Rot}_{G_{j-1}}$ based on this window's contents is incomplete
12. Put a check on the leftmost unchecked digit, and read it into the finite control as "i"
13. If the active window is the *first* window,
14. Apply Rot_{G_0} to the v part and i.
15. Else if $i \in \{1,2,3\}$,
16. Apply Rot_H to the window *left* of the active window, and i (keeping the result checked).
17. Else // $i \in \{4,5,6\}$,
18. Move the active window one window to the left.

Please note that lines 13–18 are identical to the "initial" steps 2–7.

It's clear that this algorithm works in-place and uses no extra space. We need to argue correctness. The idea is that we use checkmarks in a window to keep track of how many of the 1024 steps we've taken in following the given 1024-path in the graph "to the left". We collect up the resulting "back-edge" indices along the way. When we're done, we've gotten all the back-edges, but we need to reverse them into the correct order. We then move to the right and continue whatever it was we were doing.

We will not give a formal proof, but will do a "proof by example", in the case of (**??**). We continue the algorithm after the initial lines 1–7. Recall that we need to compute and write in

$$\mathrm{Rot}_{G_2^{1024}}((v; 64413; 23253); 16132)$$

and then end. The active window is now on 16132. As this is not the last window we proceed to line 9. None of the digits in 16132 are checked, so we move to line 12. There we check the first digit, making the window $1^{\checkmark}6132$, and he finite control remembers that we're working on a 1-edge in the first position. We're not in the first window, so we pass to line 16. Indeed, to step in G_2 along edge-1 from vertex "$(v; 64413; 23253)$", we need to do a Rot_H on 23253, and the back-edge is (probably) 2. So after line 16 the tape becomes

$$v \ \triangleright \ 64413 \ ; \ 23252 \ ; \ 2^{\checkmark}6132 \ \triangleleft \ 5 \tag{3}$$

and the active window is still the third.

The loop continues, and we pass to line 12, as intended. We now check the 6, and pass to line 16. We will pass to the window to the left here. Note that the correct sequence of events will event: we will (presumably) follow and write in the result of taking the 23253 edge of G_1^{1024} from $(v; 64413)$, and then pop back to the working on the third window. We will have left in $6\checkmark$ as the back-edge, which is correct. So the tape becomes

$$v \ \triangleright \ 64413 \ ; \ 23252 \ ; \ 2^{\checkmark}6^{\checkmark}132 \ \triangleleft \ 5 \tag{4}$$

with the second window active.

Processing the 2 in the second window will lead us to

$$v \ \triangleright \ 64414 \ ; \ 1^{\checkmark}3252 \ ; \ 2^{\checkmark}6^{\checkmark}132 \ \triangleleft \ 5 \tag{5}$$

We then come to the 3 in the second window. Again, this is just a Rot_H move, although note that it is a "chord" step. This will cause the 64414 to change to some "random" index (actually defined by H), and the result back-edge index will still be 3. So the new tape may be, say,

$$v \ \triangleright \ 21555 \ ; \ 1^{\checkmark}3^{\checkmark}252 \ ; \ 2^{\checkmark}6^{\checkmark}132 \ \triangleleft \ 5 \tag{6}$$

We process the 2 as a Rot_H step, then the 5, which moves us one window to the left:

$$v \ \triangleright \ 21556 \ ; \ 1^{\checkmark}3^{\checkmark}1^{\checkmark}5^{\checkmark}2 \ ; \ 2^{\checkmark}6^{\checkmark}132 \ \triangleleft \ 5 \tag{7}$$

with the first window active.

At this point, the goal is to compute $\mathrm{Rot}_{G_0}(v; 21556)$. Now line 13 in the main loop will be activated, so we will follow the path by looking up the rotation map of G_0, leading us to, say,

$$v' \ \triangleright \ 5^{\checkmark}3^{\checkmark}3^{\checkmark}1^{\checkmark}6^{\checkmark} \ ; \ 1^{\checkmark}3^{\checkmark}1^{\checkmark}5^{\checkmark}2 \ ; \ 2^{\checkmark}6^{\checkmark}132 \ \triangleleft \ 5 \tag{8}$$

(we don't really know about the relation between forward- and back-edge labels in G_0). Having done these 5 (actually, 1024) steps, line 9 kicks in, and we change the tape to

$$v' \ \triangleright \ 61335 \ ; \ 1^{\checkmark}3^{\checkmark}1^{\checkmark}5^{\checkmark}2 \ ; \ 2^{\checkmark}6^{\checkmark}132 \ \triangleleft \ 5 \tag{9}$$

with the second window becoming active again. We process the 2, leading to

$$v' \ \triangleright \ 61336 \ ; \ 1^{\checkmark}3^{\checkmark}1^{\checkmark}5^{\checkmark}1^{\checkmark} \ ; \ 2^{\checkmark}6^{\checkmark}132 \ \triangleleft \ 5 \tag{10}$$

and then

$$v' \ \triangleright \ 61336 \ ; \ 15131 \ ; \ 2^{\checkmark}6^{\checkmark}132 \ \triangleleft \ 5 \tag{11}$$

with the active window being the third. This then goes to

$$v' \ \triangleright \ 61336 \ ; \ 15126 \ ; \ 2^{\checkmark}6^{\checkmark}2^{\checkmark}32 \ \triangleleft \ 5 \tag{12}$$

and then something such as

$$v' \ \triangleright \ 61336 \ ; \ 64313 \ ; \ 2^{\checkmark}6^{\checkmark}2^{\checkmark}3^{\checkmark}2 \ \triangleleft \ 5 \tag{13}$$

then

$$v' \ \triangleright \ 61336 \ ; \ 64312 \ ; \ 2^{\checkmark}6^{\checkmark}2^{\checkmark}3^{\checkmark}1^{\checkmark} \ \triangleleft \ 5 \tag{14}$$

then finally

$$v' \ \triangleright \ 61336 \ ; \ 64312 \ ; \ 13262 \ \triangleleft \ 5 \tag{15}$$

with the active window being the fourth. The algorithm now ends because of line 8.

To check that this really makes sense, imagine we run the algorithm again starting with (15) — we should recover (2) (with $i = 5$)!