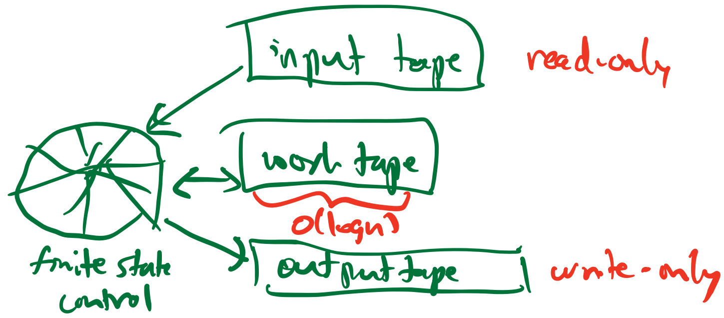
Lecture 24

CSE 431
Intro to Theory of
Computation

$L = SPACE(\log n)$ $\quad$ $NL = NSPACE(\log n)$

$L \subseteq NL \subseteq P \subseteq NP$

Def^n f is logspace-computable iff f is computable by a TM of the following form



Def^n $A \le_m^L B$ iff $A \le_m B$ via reduction f that is logspace-computable

Def^n B is NL-hard iff $\forall A \in NL$, $A \le_m^L B$

Def^n B is NL-complete iff (1) $B \in NL$ (2) B is NL-complete

Def^n PATH is NL-complete

Proof (1) $PATH \in NL$ ✓ last time

(2) Let $A \in NL$. Claim $A \le_m^L PATH$

Reduction from last time:

$$\begin{array}{ccc} A & & PATH \\ x & \xrightarrow{f} & \langle G_{M,x}, C_0, C_{accept} \rangle \end{array}$$

where M is logspace NTM deciding A

Why is f logspace-computable?

- each configuration/vertex of $G_{M,x}$ takes $O(\log n)$ space so C_0, C_{accept} easy

Producing $G_{M,x}$:

Adjacency list form:

For all configurations C (in lexicographic order, not necessarily reachable)

Output C followed by all next configurations D_i s.t. $C \vdash_M D_i$ (i.e. $C \to D$)

based on δ function of M (builtin)

i.e. $C : D_1, \ldots, D_b$;

vertex out-neighbors

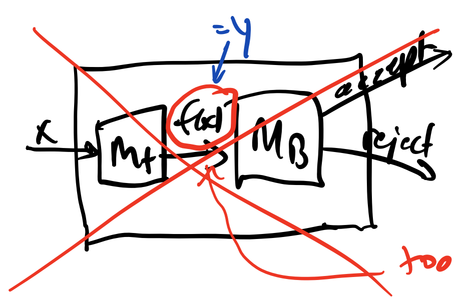
only need to store a constant # of configurations.

$\therefore O(\log n)$ space ∎

We still need to prove properties of $\leq_m^L$ that were easy for $\leq_m$ and $\leq_m^P$ but are tricky for $\leq_m^L$:

Thm
- If $A \leq_m^L B$ and $B \in L$ then $A \in L$
- If $A \leq_m^L B$ and $B \in NL$ then $A \in NL$
- If $A \leq_m^L B$ and $B \leq_m^L C$ then $A \leq_m^L C$

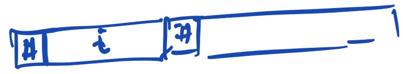
Proof Usual method



too long to write down for $O(\log n)$ space

Instead:

Modify M_B: If M_B is looking at y_i we have M_B also keep track of the input head position i



Change M_f by removing its output tape
New machine for A will "call" M_f with index i (x is still on input tape, i is on the work tape)
Each time it does it will run M_f ignoring its output except for the ith bit of output
M_f will need to keep track of the # of bits output so far, j.
Re-run M_f each time step of M_B to find out the value of y_i

Total space: Space for M_f
Space for M_B
$+O(\log n)$

Note: $|f(x)|$ is $n^{O(1)}$ if $|x| = n$
$\therefore \log |f(x)|$ is $O(\log n)$
so still $O(\log n)$ space total

Note: • same construction works for NL case.
• For $A \le_m^L B$ and $B \le_m^L C \Rightarrow A \le_m^L C$
do the same except M_B replaced by M_g
do same change as above

∎

Cor $PATH \le_m^L C \Rightarrow C$ is NL-hard

The following is very surprising

Thm $\overline{PATH} \in NL$

$\overline{PATH} \simeq \{ <G,s,t> : G$ does not have a path from s to $t \}$

Cor $NL = coNL$ complements of languages in NL

Cor For any space bound $S(n) \geq \log_2 n$

$NSPACE(S(n))$ is closed under complement

Proof Imagine that we have the value

Count = # of vertices of G reachable from s

NoPath(s, t, Count, i)

Reach ← 0

For all vertices $v \neq t$, $v \in G$

Guess whether v is reachable from s

if Guess is yes then

Guess & verify a path of length $\le \cancel{n}$ c

from s to v, one vertex at a time

if path found Reach ← Reach+1

else reject

end for

if reach = count then accept

else reject

≥ Reach paths from s to vertices other than t

If ≥ Count such paths, t is not reachable

How do we compute Count?

Idea: "inductive counting"

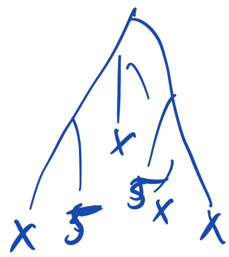
Define: Count_i = # of vertices reachable from s via paths of length $\leq i$

$\therefore$ $\text{Count}_0 = 1$ $\quad \{s\}$

$\text{count} = \text{count}_n$

This will be via a nondeterministic algorithm:

such an alg. with have some paths that reject but any branch that does not reject will compute the correct value.



We can't afford to store all the Count_i vars but we only need vars for the current & next i, counti, count i plus 1

```
i ← 0, count i ← 1
for i = 0 to n-1 do
    count i plus 1 ← 0
    for all vertices v ∈ G do
        if v = s then
            countiplus1 ← countiplus1 + 1
        else
            Guess whether v is reachable from s via
                a path of length ≤ i+1
            If guess for v is yes
                Guess & verify a path of length ≤ i+1
                  from s to v, one vertex at a time
                if found then countiplus1 ← countiplus1+1
                    else reject
```

If guess for v is no
    For all predecessors u of v in G
        "Check that no path of length $\le i$
            from s to u in G"
        if NoPath(s, t, count$_i$, i) is false
            then reject
    end for
end if
end for
Count $\leftarrow$ count$_i$ plus 1

Clearly only a constant # of counters and vertices need to be stored $\therefore O(\log n)$ space $\blacksquare$