22-04-19

Def^{n}s:

- L = SPACE (log n)
- NL = NSPACE (log n)
- Co-NL = $\{L \mid \bar{L} \in NL\}$
- A is NL-complete if
  a.) $A \in NL$
  b.) $\forall B \in NL \leq_L A$
  ↳ logarithmic space reducible (lower than polynomial)

n → size of space

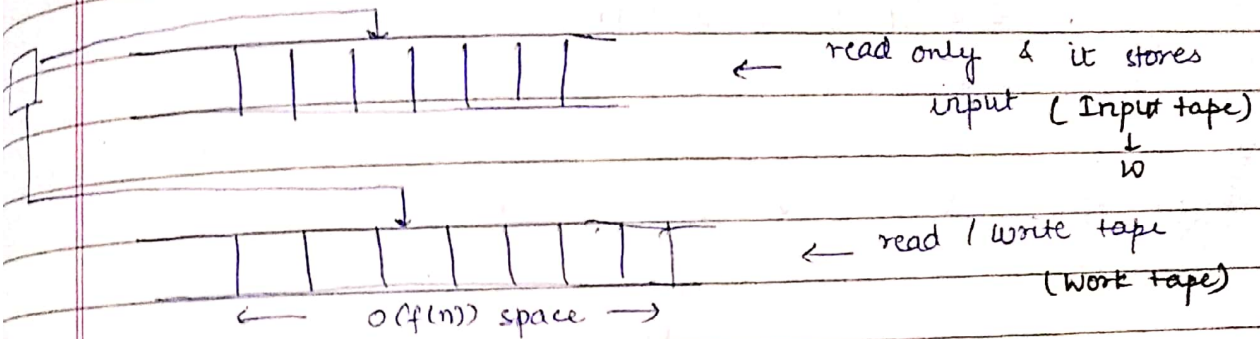
⟹ don't even store whole input. → only log n space

⇓ move to new model

DATE : / /
PAGE NO. :

New Model :

T.M. consists of 2 tapes



* Will take into account only Work tape while calculating space complexity of the T.M.

$O(\log n)$ → no. of cells in work tape [in case of $O(\log n)$ space]

→ NL : if ∃ a branch which accepts in $O(\log n)$ space

if ∀ branches → rejects in " "

same like NP.

→ To get $\leq_L$ → we need 1 more tape (in add^n to 2)

↳ $|f(w)|$ might be $> \log n$.

Space complexity → calculated wrt work tape only.

→ For 2-tape T.M. :

We'll try to find only :

↳ head of 1st tape
↳ head of 2nd tape
↳ work tape symbols

→ These will define the config^n of the T.M.

DATE : / /
PAGE NO. :

Suppose : we have T.M. M(w)

bdd no. of cells in workspace : $O(f(n))$

$C = \Gamma \cup Q \cup \{\sqcup\} \cup \{\triangleright\}$

$t = |C|$

Total no of config's :- $t^{O(f(n))} * O(f(n)) * n$

- $O(f(n))$ → 2nd head ptr
- n → 1st head ptr ($|w| = n$) (length of tape)

To store these many config's, total space needed :-

$$\log_2 [t^{O(f(n))} \cdot O(f(n)) \cdot n]$$

$$= \log_2(n) + \log_2(O(f(n)) + O(f(n)) \log_2(t)$$

Since we're trying to find upper bound,

$$O(f(n)) t^{O(f(n))} < 2^{O(f(n))} \quad \text{---(1)}$$

$t \rightarrow$ const.

So, $t < 2^x$ for some $x \rightarrow$ const.

$$t^{O(f(n))} < 2^{x \, O(f(n))}$$

$$O(f(n)) < 2^{O(f(n))}$$

→ multiply these two to get (1)

Max.

→ Space needed : $\log(n) + O(f(n))$

Ex. $PATH = \{ \langle G, s, t \rangle \mid \exists$ a path from s to t in G $\}$

By guess → we stored just 1 vertex (done earlier)

ND

Work space : $O(\log_2 n)$ [G not stored here]

→ PATH ∈ NL

DATE: / /

PAGE NO.:

→ $\overline{PATH} \in$ Co-NL.

We'll prove that : $\overline{PATH} \in$ ~~N~~NL (later)

Ex. Prove that PATH ∈ NL-Complete.

1.) PATH ∈ NL ✓

2.) ∀ B ∈ NL $\leq_L$ PATH → Taken NDTM M which creates config^n graph acc. to its own config^n.

→ config^n graph (M runs in O(log n) space)

Can have ND T.M. M(w) :

if M accepts w ⟺ if ∃ a path from starting config^n to accepting config^n

rejects ⟺ if ∄ doesn't exist

<G, C_0, C_{accept} > ∈ PATH

M(w) ⟷ <G, C_0, C_{accept} > ∈ PATH

$C_0, C_1, C_2, \ldots C_{accept}$ ⇒ possible config^{n}s ← log n

~~For~~ space taken by these config^{n}s : log n + O(f(n))
= O(log n)

In o/p tape. → put these config^{n}s.
↓ check
if ∃ an edge b/w 2 config^{n}s

o/p tape: # | C_0 | # | C_1 | # | ...

| # | C_0 | , | C_1 | # — if ∃ edge b/w C_0 & C_1

put 2 config^{n}s C_i & C_j in work table & check if ∃ an edge or not. If ∃ → put it in o/p table.

So, workspace : $O(\log_2 n)$

PATH $\in$ NL

1. $\exists$ a path from s to t in G → accept
2. $\forall$ ~~non~~ s, t $\exists$ no path ~ → reject

DATE : / /
PAGE NO. :

OR

→ Instead of storing whole o/p in o/p tape, each time we'll generate symbol at ith tape symbol in o/p tape (graph)

↓

Ignore all symbols before i.
Run for as many times as you want → don't take care of time.

o/p tape

↓

store this ith symbol in workspace.

Th^m.

Ex. Prove : $\overline{PATH} \in NL$. & $PATH \in co\text{-}NL$

$PATH \in co\text{-}NL$

⇒ 1. $\forall$ non-path from s to t in G → accept
2. $\exists$ a path ~ ~ ... → reject

→ Difficult to prove (PATH $\in$ NL : easy to prove)

We've to use guess mechanism → Have to reject all paths from s to t.

1st theory : If ensure 2nd part, 1st part not ensured [unlike PATH $\in$ NL Proof]

24/04/19

Th^m. Ex. $\overline{PATH} = \{ \langle G, s, t \rangle \mid$ node t is not reachable from node s in G$\} \in NL$

$A_i = \{ v \mid$ node v is reachable from s in G within i-steps$\}$

$C_i = |A_i|$ $\quad 0 \le i \le |V[G]|$ → no. of vertices in G

$C_{|V[G]|}$ → no. of vertices reachable from s in atmost $|V[G]|$ steps.

DATE : / /
PAGE NO. :

If $C_{|V[G]|}$ is given, we can check $A_{|V[G]|}$

$d = 0$

$A_{|V[G]|}$: 1. for each u in $V[G]$, guess a path from s to u.

If guess is correct → d++
else → reject

~~else → reject~~

2. If t is reachable → reject
3. If $d \neq C_{|V[G]|}$ → reject [not able to find correct size of $A_{|V[G]|}$]
4. Otherwise → accept [correctness of $C_{|V[G]|}$ means correctness of our algo]

↳ if $d == C_{|V[G]|} \Rightarrow C_{|V[G]|}$ provided to us was correct.

→ $C_0 = |\{s\}|$ → this we know, is correct

Assume that C_i is correct.

We'll try to find correct value of C_{i+1} from correctness of C_i

By doing this, if we reach

$i = |V(G)| - 1$ → Our algo is correct.

For calc. C_i, we will ~~per~~ execute the algorithm [use C_i instead of $C_{V[G]}$]

suppose, we get some $C_i = \{i, t, \ldots\}$

for $v \in V[G]$

for $u \in C_i$, $d = C_i$

→ check if u is reachable from s or not in atmost ~~path~~ i steps

if $(u, v) \in E[G]$: → if ~~state~~ I an edge

C_{i+1} ++

DATE : / /

PAGE NO. :

→ 1 - for each $v \in V[G]$

2. Assume we have correct value of G, we're checking by a for loop

$u \in V[G]$

3. if $(u, v) \in E[G]$ → C_{i+1} ++; → go to 1;

↳ checking every pair

→ given : A_i

you want to calculate : A_{i+1}

for each u in V[G]

for each v in A_i

if $(v, u) \in E[G]$

$A_{i+1} = A_i \cup \{u\}$ → If you maintain set, it'll take more space → using counter C_i

⇓

Here, we're starting from A_i to get A_{i+1}

⇒ ~~Suppose $C_{|V[G]|}$ is given.~~

⇒

Space taken = $O(\log n)$ → only storing variables.