Space Complexity

Foundations of Computing Science

Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

Introduction

Definition

- Let *M* be a deterministic Turing machine that halts on all inputs. The *space complexity* of *M* is the function $f: \mathcal{N} \rightarrow \mathcal{N}$, where *f(n)* is the maximum number of tape cells that *M* scans on any input of length *n*. If the space complexity of *M* is *f(n)*, we also say that *M* runs in space *f(n)*

- If *M* is a non-deterministic Turing machine wherein all branches halt on all inputs, we define its space complexity *f(n)* to be the maximum number of tape cells that *M* scans on any branch of its computation for any input of length *n*

Space Complexity Classes

Definition: Let *f: $\mathcal{N} \rightarrow \mathcal{R}^+$* be a function. The space complexity classes, *SPACE(f(n))* and *NSPACE(f(n))*, are defined as follows:

- SPACE(f(n)) = {L | L is a language decided by O(f(n)) space by a deterministic TM}
- NSPACE(f(n)) = {L | L is a language decided by an O(f(n)) space by a non-deterministic TM}

Examples

- *SAT* can be solved with a linear space algorithm [Space complexity = O(n)]
- Testing whether a non-deterministic finite automaton accepts all strings,
  i.e. $ALL_{NFA} = \{<\mathcal{A}> \mid \mathcal{A} \text{ is a NFA and } L(A) = \Sigma^*\}$
  - Non-deterministic space complexity = O(n)

SAVITCH'S Theorem

- For any function *f: $\mathcal{N} \rightarrow \mathcal{R}^+$*, where $f(n) \geq n$,

$$NSPACE(f(n)) \subseteq SPACE(f^2(n))$$

Proof of Savitch's Theorem

CANYIELD = "On input c_1, c_2, t:

1. If *t = 1* then test directly whether $c_1 = c_2$ or whether c_1 yields c_2 in one step according to the rules of *N*. *Accept* if either test succeeds; *reject* if both fail.
2. If *t > 1* then for each configuration c_m *of N* on *w* using space *f(n)*:
3. Run CANYIELD(c_1, c_m, t/2)
4. Run CANYIELD(c_m, c_2, t/2)
5. If steps 3 and 4 both accept, then *accept*
6. If haven't yet accepted, *reject.*

We select a constant *d* so that *N* has no more than $2^{df(n)}$ configurations using *f(n)* tape, where *n* is the length of *w*. Then we know that $2^{df(n)}$ is an upper bound on the running time of any branch of *N* on *w*.

M = "On input *w*:

1. Output the result of CANYIELD(c_{start}, c_{accept}, $2^{df(n)}$)."

The Class PSPACE

Definition

- ***PSPACE*** **is the class of languages that are decidable in polynomial space on a deterministic Turing machine. In other words,**

$$\text{PSPACE} = \bigcup_k \text{SPACE}(n^k)$$

- ***NPSPACE*** **is the class of languages that are decidable in polynomial space on a non-deterministic Turing machine. In other words,**

$$\text{NPSPACE} = \bigcup_k \text{NSPACE}(n^k)$$

Relationship among the Complexity Classes

- $\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXPTIME}$

PSPACE-Completeness

Definition

- A language $\mathcal{B}$ is *PSPACE-complete* if it satisfies two conditions:
  - $\mathcal{B}$ is in PSPACE, and
  - Every $\mathcal{A}$ in PSPACE is polynomial time reducible to $\mathcal{B}$
- If $\mathcal{B}$ merely satisfies condition-2, we say that it is *PSPACE-hard*

Examples of PSPACE-complete Problems

- TQBF = {<Φ> | Φ is a true fully quantified Boolean formula}
- FORMULA-GAME = {<Φ> | Player *E* has a winning strategy in the formula game with *Φ*}
- GENERALIZED-GEOGRAPHY =

  {<G, b> | Player *I* has a winning strategy for the generalized geography game played on the graph G starting at node *b*}

The Classes L and NL

- *L* is the class of languages that are decidable in logarithmic space on a deterministic Turing machine. In other words, L = SPACE(log n)
- *NL* is the class of languages that are decidable in logarithmic space on a non-deterministic Turing machine. In other words, NL = NSPACE(log n)

Examples

- The language $\mathcal{A} = \{0^k1^k \mid k \geq 0\}$ is a member of *L*
- The language PATH = {<G, s, t> | G is a directed graph that has a directed path from s to t} is a member of *NL*

Definition

- If *M* is a Turing machine that has a separate read-only input tape and *w* is an input, a configuration of *M* on *w* is a setting of the state, the work tape, and the position of the two tape heads. The input *w* is not a part of the configuration of *M* on *w*

Log-Space Reducibility

Definitions

- A *log space transducer* is a Turing machine with a read-only input tape, a write-only output tape, and a read/write work tape. The work tape may contain *O(log n)* symbols.

- A log space transducer *M* computes a function $f\colon \Sigma^* \rightarrow \Sigma^*$, where *f(w)* is the string remaining on the output tape after *M* halts when it is started with *w* on its input tape. We call *f* a *log space computable function*.

- Language $\mathcal{A}$ is *log space reducible* language $\mathcal{B}$, written $\mathcal{A} \leq_L \mathcal{B}$, if $\mathcal{A}$ is mapping reducible to $\mathcal{B}$ by means of a log space computable function *f*

NL-Completeness

A language $\mathcal{B}$ is *NL-complete* if

- $\mathcal{B} \in$ NL, and
- Every $\mathcal{A}$ in NL is log space reducible to $\mathcal{B}$

Theorem

- If $\mathcal{A} \leq_L \mathcal{B}$ and $\mathcal{B} \in$ L, then $\mathcal{A} \in$ L
- Corollary: If any NL-complete language is in *L*, then *L* = *NL*

Example of NL-complete Problems

- PATH = {<G, s, t> | G is a directed graph that has a directed path from s to t}
- Corollary: NL $\subseteq$ P

Theorem

- NL = coNL