PART 4:

Complexity Theory:

Complexity
Time Complexity Classes
Space Complexity Classes

Space Complexity Classes

PSPACE & NPSPACE

Space Complexity

Measuring Space Requirements

spacereq(M) measures the worst-case space requirement of TM M as a function of the length of the input *n*.

Measuring Space Requirements

- If *M* is a ***deterministic*** TM that halts on all inputs, then:

  spacereq(M) = *f*(*n*) = the maximum number of tape squares that *M* reads on any input of length *n*.

- If *M* is a ***nondeterministic*** TM all of whose computational paths halt on all inputs, then:

  spacereq(M) = *f*(*n*) = the maximum number of tape squares that *M* reads on any path that it executes on any input of length *n*.

Example 29.1

CONNECTED = {<*G*> : *G* is an undirected graph and *G* is connected}

spacereq(CONNECTED) is $\mathcal{O}(|<G>|)$.

✓ O(n) Linear space.

Example 29.2

SAT = {*w* : *w* is a Boolean wff and *w* is satisfiable}

spacereq(SAT) is $\mathcal{O}(|w|)$.

✓ O(n) Linear space.

Example 29.3

TSP-DECIDE = {<*G*, *cost*> : <*G*> encodes an undirected graph with a positive distance attached to each of its edges and *G* contains a Hamiltonian circuit whose total cost is less than <*cost*>}

spacereq(TSP-DECIDE) is $\mathcal{O}(|\langle G \rangle|)$.

✓ O(n) Linear space.

Relating Time and Space Complexity

Relating Time and Space Complexity

- There is some relationship between the number of steps a TM executes and the number of space it uses!

- A TM can examine at most one tape square at each step of its operations! So,

$$spacereq(M) \leq timereq(M)$$

✓ Space can be reused, but time cannot!

Relating Time and Space Complexity

Max # of distinct configurations of TM M with K states & Γ tape symbols and *spacereq*(*M*):

$$MaxConfigs(M) = |K| \cdot |\Gamma|^{spacereq(M)} . spacereq(M).$$

Let *c* be a constant such that $c > |\Gamma|$.

Then:

$$MaxConfigs(M) \in \mathcal{O}(c^{spacereq(M)}).$$

Relating Time and Space Complexity

$MaxConfigs(M) \in \mathcal{O}(c^{spacereq(M)})$ and
$timereq(M) \leq MaxConfigs(M)$.

So,

$timereq(M) \in \mathcal{O}(c^{spacereq(M)})$.

Relating Time and Space Complexity

Theorem 29.1 Given a Turing machine $M = (K, \Sigma, \Gamma, \delta, s, H)$ and assuming that $spacereq(M) \geq n$, the following relationships hold between M's time and space requirements:

$$spacereq(M) \leq timereq(M) \in O(c^{spacereq(M)}).$$

Proof Idea:

Spacereq(*M*) is bounded by *timereq*(*M*) since *M* must use at least one time step for every tape square it visits.

Since *M* halts, the number of steps that it can execute is bounded by *MaxConfigs*(*M*), the number of distinct configurations that it can enter.

Space Complexity Classes: PSPACE & NPSPACE

PSPACE =

{ Problems Solvable in Polynomial Space by DTMs }

NPSPACE =

{ Problems Solvable in Polynomial Space by NDTMs }

P (PTIME) and NP (NPTIME)

- The Class P: $L \in$ P iff there exists some *deterministic* Turing machine M that decides L and $timereq(M) \in \mathcal{O}(n^k)$ for some constant k.

- The Class NP: $L \in$ NP iff there exists some *nondeterministic* Turing machine M that decides L and $timereq(M) \in \mathcal{O}(n^k)$ for some constant k.

Is P = NP ? No one knows!

PSPACE

The Class **PSPACE**:

$L \in$ PSPACE iff

- there exists some ***deterministic*** Turing machine M that **decides** L and
- $spacereq(M) \in \mathcal{O}(n^k)$ for some constant k.

NPSPACE

The Class **NPSPACE**:

$L \in$ NPSPACE iff

- there exists some ***nondeterministic*** Turing machine M that **decides** L and
- $spacereq(M) \in \mathcal{O}(n^k)$ for some constant k.

Savitch's Theorem

- DTM can simulate NTM by using a surprisingly small amount of **space**!

✓ **PSPACE = NPSPACE**

- For time complexity, such a simulation requires an exponential increase in time!

Savitch's Theorem

Theorem 29.2 If *L* can be decided by some *nondeterministic Turing machine M* and $spacereq(M) \geq n$, then there exists a *deterministic Turing machine M′* that also decides *L* and
$\boldsymbol{spacereq(M') \in \mathcal{O}(spacereq(M)^2)}$.

Proof Idea:

The proof is by construction of a DTM *M′* that searches the tree of computations performed by *M*.

PSPACE = NPSPACE

Theorem 29.3 PSPACE = NPSPACE.

Proof Idea: We will prove:

- If *L* is in PSPACE then it is NPSPACE. (Trivial)
- If *L* is in NPSPACE then it is in PSPACE:

PSPACE = NPSPACE

- If L is in NPSPACE then there is some NDTM M such that M decides L and $spacereq(M) \in \mathcal{O}(n^k)$ for some k.
- If $k \geq 1$, then, by Savitch's Theorem, there exists a DTM M' such that M' decides L and $spacereq(M') \in \mathcal{O}(n^{2k})$.
- If $k < 1$ then, using the same construction that we used in the proof of Savitch's Theorem, we can show that there exists a DTM M' such that M' decides L and $spacereq(M') \in \mathcal{O}(n^2)$.

$P \subseteq NP \subseteq PSPACE$

Theorem 29.4 $P \subseteq NP \subseteq PSPACE$.

Proof Idea:

We have already shown that $P \subseteq NP$.

Proof that $NP \subseteq PSPACE$:

If *L* is in NP, then it is decided by some NDTM *M* in polynomial time.

In polynomial time, *M* cannot use more than polynomial space since it takes a least one time step to visit a tape square.

Since *M* is an NDTM that decides *L* in polynomial space, *L* is in NPSPACE.

But, by Savitch's Theorem, PSPACE = NPSPACE.

So *L* is also in PSPACE.

PSPACE-Completeness

PSPACE-Hard & PSPACE-Complete

A language *L* might have these properties:

1. *L* is in PSPACE.
2. Every language in PSPACE is deterministic, polynomial-time reducible to *L*.

- *L* is ***PSPACE-hard*** iff it possesses property 2.
- *L* is ***PSPACE-complete*** iff it possesses *both* property 1 and property 2.

PSPACE-Completeness, P, and NP

All PSPACE-complete languages can be viewed as being equivalently hard in the sense that all of them can be decided in polynomial space and:

- If any PSPACE-complete language is also in NP, then all of them are and NP = PSPACE.
- If any PSPACE-complete language is also in P, then all of them are and P = NP = PSPACE.

Quantified Boolean Expression

A *Quantified Boolean Expression* is :

- The base case: all wffs are QBEs.
- Adding quantifiers: if *w* is a QBE that contains the unbound variable *A*, then the expressions $\exists A\ (w)$ and $\forall A\ (w)$ are QBEs.

Quantified Boolean Formula

A *Quantified Boolean Formula* is a QBE that is also a sentence (i.e., all of its variables are bound):

- $\exists P\,(\exists R\,(P \wedge \neg R)) \in$ QBF.
- $\exists P\,(\forall R\,(P \wedge \neg R)) \notin$ QBF

The QBF Problem

Given a QBF with no free variables, does it have the value 1?

QBF = {<*w*> : *w* is a true quantified Boolean formula}

QBF is PSPACE-Complete

- ✓ QBF is in PSPACE
- ✓ QBF is PSPACE-Complete

More PSPACE-Hard Problems

- Two-Person Games
- Some Questions on Languages and Automata

Tractability Hierarchy of Decidable Languages

- P
- NP
- PSPACE
- NPSPACE

EXPTIME

P ⊆ NP ⊆ PSPACE = NPSPACE ⊆ EXPTIME

P ≠ EXPTIME

P ⊂ EXPTIME

Reading Assignment

Chapter 29:

Sections
29.1
29.2
29.3
29.6

In-Class Exercises

Chapter 29:

2 – a

Practical Solutions for Hard Problems

Hard Problems

- NP
- PSPACE
- NPSPACE
- EXPTIME

Approaches

Strategies for developing efficient algorithms to solve a hard problem:

- ✓ Compromise on generality
- ✓ Compromise on optimality
- ✓ Compromise on both
- ✓ Compromise on total automation

Compromise on Generality

- On most (not all) problem instances!

Compromise on Optimality

- A good (not optimal) solution!

Compromise on Total Automation

- An algorithm that works interactively with a human user!

Reading Assignment

Chapter 30:

Section
30.1