Intractability

An intractible problem is

- solvable in principle
- but not in practice

Computaton takes longer than expected time until

- the next file system crash
- your graduation
- you finish your Ph.D.
- the heat death of the universe

Intractability

So far, we haven't seen any *provably* intractable problems.

Do they exist?

We will focus on inordinate consumption of:

- space (first)
- time

Hierarchy Theorems

Question: Can a TM with more resources decide more languages?

Answer: Yes, subject to some technical restrictions.

For example, a TM can accept more languages in time $O(n^2)$ than in $O(n \log n)$.

Review: “little-o” Notation

Recall that “big-O” notation is like $\leq$.

Also useful to have analog of $<$.

$f(n) = o(g(n))$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

For any $\epsilon > 0$, $f(n) < \epsilon g(n)$ for sufficiently large n.

Space Constructability

A function

$$f : \mathcal{N} \rightarrow \mathcal{N} \text{ where } f(n) \geq \log n$$

is *space constructible* if the function that maps 1^n to the binary representation of $f(n)$ is computable in space $O(f(n))$.

In other words,

- There exists a TM M,
- when started with input 1^n,
- always halts with binary $f(n)$ on tape,
- and requires space $O(f(n))$.

Example

Claim: n^2 is space constructable.

Consider this TM

- convert input 1^n into binary
- square it

Does this run in space $O(n^2)$?

Sublinear Functions

If $f(n) = o(n)$, use separate read-only input tape, just as for space complexity.

To compute $\log n$:

- scan unary input tape
- write binary input on work tape
- count the bits

Motivation

What if

- f asymptotically larger than g
- by a "very small", hard to compute amount?
- cannot exploit extra space because computing the amount of space may itself require too much space.

Space constructible functions don't have this problem.

Space Hierarchy Theorem

Theorem: For any space-constructable function

$$f : \mathcal{N} \to \mathcal{N}$$

there exists a language A decidable in space $O(f(n))$ but not $o(f(n))$.

Need to demonstrate a language A such that

- A decidable in space $O(f(n))$
- A not decidable in space $o(f(n))$

Proof Idea

We describe A by giving an algorithm B that decides it.

In fact, the algorithm is the *only* definition we give.

No one said this would be pretty.

Proof Idea

B takes in a TM description M.

- If M runs in space $o(f(n))$, we use diagonalization to ensure that A differs from $L(M)$.

Proof Idea

B takes in a TM description M.

- If input not a TM description, *reject*.
- Otherwise, run M on $< M >$ in space bound $f(n)$.
- If M accepts in space bounds, *reject*.
- If M rejects in space bounds, *accept*.
- If M doesn't halt, *reject*.

- If M runs in space $O(f(n))$, then A differs
- If M doesn't, then we don't care.

Lots of details to fill in.

Problem

Question: What if M uses more than $f(n)$ space for small n?

Answer: Instead of running M on $\langle M \rangle$, run it on $\langle M \rangle 10^*$.

Asymptotic behavior will eventually ensure that B and M disagree.

Another Problem

Question: What if M goes into an infinite loop in finite space?

Answer: Any machine that runs in space $o(f(n))$ uses only $2^{o(f(n))}$ time. *Why?*

Just count the number of steps, and reject after $2^{f(n)}$ steps.

Proof

The following $O(f(n))$-space algorithm B decides a language A that is not decidable in space $o(f(n))$.

On input w:

1. Let n be the length of w.

2. Comput $f(n)$ using space constructibility, and mark off that much tape. If ever try to use more, *reject*.

3. If w is not of form $\langle M \rangle 10^*$, *reject*.

4. Simulate M on w, counting steps. If steps exceed $2^{f(n)}$, *reject*.

5. If M accepts, *reject*. If M rejects, *accept*.

Claims

Claim: B is a decider. *Why?*

Claim: A is decidable in space $f(n)$. *Why?*

Claim

Claim: A not decidable in space $o(f(n))$.

Proof: Assume M decides A in space $g(n) = o(f(n))$.

- B simulates M in space $dg(n)$ for some constant d.
- because $g(n) = o(f(n))$, for some n_0, $dg(n) < f(n)$ for all $n < n_0$.
- so B's simulation finishes when run on $\langle M \rangle 10^{n_0}$.
- So B does the opposite of M on that input!

Corollary

For any two

$$f_1, f_2 : \mathcal{N} \to \mathcal{N}$$

where

- $f_1(n)$ is $o(f_2(n))$
- f_2 is space constructible

then

SPACE($f_1(n)$) is a proper subset of SPACE($f_2(n)$).

This is a *separation* result: one class is strictly larger than another.

Corollary

For any two real numbers $0 < \epsilon_0 < \epsilon_1$,

SPACE(n^{ϵ_0}) is a strict subset of SPACE(n^{ϵ_1})

- n^c is space constructible for any natural number c.
- n^c is space constructible for any rational $c > 0$ (trust me).
- for any $\epsilon_1 < \epsilon_2$, there exist rational c_0, c_1 such that $0 < \epsilon_1 < c_1 < c_2 < \epsilon_2$.

Corollary

Theorem: NL is a struct subset of PSPACE

proof

- By Savitch's theorem, $NL \subseteq \text{SPACE}(\log^2 n)$.
- By the Space Hierarchy Theorem, $\text{SPACE}(\log^2 n)$ is a proper subset of $\text{SPACE}(n)$.

Down to Business

Theorem: PSPACE is a struct subset of EXPSPACE.

Space-intractible problems exist (other than Microsoft Word (tm)).

Hierarchy theorem says that

$$\begin{array}{rl} \text{SPACE}(n^k) & \text{is a strict subset of } \text{SPACE}(n^{\log n}) \\ \text{SPACE}(n^{\log n}) & \text{is a strict subset of } \text{SPACE}(2^n) \end{array}$$

- we know they exist
- abstract and artificial
- so far

Now we consider time intractability.

- slightly weaker results
- no one knows how to do better

Time Constructability

A function

$$f : \mathcal{N} \to \mathcal{N} \text{ where } f(n) \geq \log n$$

is *space constructible* if the function that maps 1^n to the binary representation of $t(n)$ is computable in space $O(t(n))$.

In other words,

- There exists a TM M,
- when started with input 1^n,
- always halts with binary $t(n)$ on tape,
- and runs in time $O(t(n))$.

Time Hierarchy Theorem

Theorem: For any time constructible function $f : \mathcal{N} \to \mathcal{N}$, there exists a language A that is decidable in time $O(t(n))$, but not in time $o(t(n))/\log t(n)$.

Notice that

- any space bound increase enhances compuational power,
- but need lograthimic time increase to make a difference.
- No one knows whether tighter results hold.

Proof Idea

Similar to Space Hierarchy proof.

- construct B that decides A in time $O(n)$
- B takes input of form $\langle M \rangle 10^*$
- simulates M on $\langle M \rangle 10^*$, for time $t(n)$, and does the opposite.

Important difference: Simulation overhead is

- constant for space, but
- logarithmic for time

Proof

The following $O(t(n))$-time algorithm B decides a language A that is not decidable in time $o(t(n)/\log t(n))$.

On input w:

1. Let n be the length of w.

2. Comput $t(n)$ using time constructibility, and store $\lceil t(n)/\log t(n) \rceil$ as a binary counter. Decrement this counter before each step used to carry out steps 3,4,5. When counter hits zero, *reject*.

3. If w is not of form $\langle M \rangle 10^*$, *reject*.

4. Simulate M on w.

5. If M accepts, *reject*. If M rejects, *accept*.

Details

Each time B simulates a state of M, it looks at

- M's current state
- tape symbol under M's simulated head
- looks up M's transition function

All three (state, symbol, δ)

- must be stored on a single tape
- if they're far apart, simulation will be slow
- need to keep them together.

More Details

Divide the tape into *tracks*.

- one track in even positions, one track odd, or
- use a bigger tape alphabet.
- generalizes to multiple tracks

Either way, constant overhead.

- every time the tape head moves
- shift $\langle\delta\rangle$ along with it.

Adds only constant factor *why?*.

Bean Counting

At every simulated step, B must decrement its counter.

- put counter on third track
- shift it along too.

Counter size is

$$\log\left(\frac{t(n)}{\log t(n)}\right) = O(\log t(n)).$$

Adds $O(\log t(n))$ factor.

Rest of proof just like the space theorem.

Corollary

For any two

$$t_1, t_2 : \mathcal{N} \rightarrow \mathcal{N}$$

where

- $t_1(n)$ is $o(t_2(n)/\log t_2(n))$
- t_2 is space constructible

then

TIME($t_1(n)$) is a proper subset of TIME($t_2(n)$).

Corollary

For any two real numbers $0 < \epsilon_0 < \epsilon_1$,

$$\text{TIME}(n^{\epsilon_0}) \text{ is a strict subset of } \text{TIME}(n^{\epsilon_1})$$

Theorem: P is a struct subset of EXPTIME.

Exponential Space Completeness

Let's show that a specific language is intractable.

We will show tht a generalization of regular expressions is EXPSPACE-complete.

Generalized Regular Expressions

Regular expressions were built up from

- Union $\cup$
- Concatenation $\circ$
- Star $*$

We can test equivalence in polynomial space.

Define the *exponentiation* operator:

$$R^k = R \circ R \circ \cdots \circ R$$

k times.

Question: Does exponentiation generate new languages?

The Problem

$$EQ_{\mathsf{rex}\uparrow} = \{\langle Q, R\rangle | Q, R \text{ equivalent regular expressions with exponentiation}\}$$

Definition: A language B is *EXPSPACE-complete* if

- $B \in$ EXPSPACE
- every A in EXPSPACE is polynomial-time reducible to B.

If an EXPSPACE-compete problem were in PSPACE or in P, then EXPSPACE = PSPACE, contradicting the Space Hierarchy Theorem.

(

$EQ_{\textbf{rex}\uparrow}$ is in EXPSPACE

Algorithm Sketch:

- use repetition to eliminate exponentiation
- convert to NFAs
- test NFAs for equivalence

Interesting part is the reduction.

Reduction

Use reduction via computation histories.

Given a language A accepted by a TM.

- map input w to Q, R
- equivalent exactly when TM accepts w
- Q generates all strings consisting of symbols that might appear in computation histories
- R generates all strings that are not rejecting histories.
- the TM accepts w iff Q and R are equivalent.

Challenge is doing reduction in poly-time.