Advanced topic: Space complexity
CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN

Chinese University of Hong Kong

Fall 2016
Review: time complexity

We have looked at how long it takes to solve various problems

What about the amount of memory? We measure memory usage (space) by the number of tape cells used

Questions one may ask:
If a problem can be solved quickly, can it be solved with little memory? If a problem can be solved with little memory, can it be solved quickly?
Space complexity

The space complexity of a Turing machine $M$ is the function $s_M(n)$:

$$s_M(n) = \text{maximum number of cells that } M \text{ ever reads on any input of length } n$$

Example:  \[ L = \{ w\#w \mid w \in \{a, b\}^* \} \]

$M$: On input $x$, until you reach $\#$
- Read and cross off first $a$ or $b$ before $\#$
- Read and cross off first $a$ or $b$ after $\#$
- If mismatch, reject
- If all symbols except $\#$ are crossed off, accept

\[ \text{space complexity: } n + 1 \]

“$+1$” because $M$ may scan the blank symbol after the input
Sublinear space

If we assume the Turing machine has two tapes

1. Input tape: contains the input and is read-only
2. Work tape: initially empty, only the cells used here is counted

We will assume this in this lecture

Then $L$ can be solved in $O(\log n)$ space

$$L = \{ w\#w \mid w \in \{a, b\}^* \}$$

Idea: Keep a counter, storing the number of symbols matched so far
Counter can represent a number of size $m$ in using $O(\log m)$ bits
Logarithmic space

Smallest reasonable amount of space used will be logarithmic in input length

Just keeping one counter/pointer requires $\log n$ memory!

A language $L$ is in $L$ if $L$ can be decided by a deterministic Turing machine (with read-only input tape) in $O(\log n)$ space
Time vs space

If a Turing machine runs in time $t_M(n)$, how much space can it use?

If a Turing machine uses space $s_M(n)$, how long can it take?
Time vs space

If a Turing machine runs in time $t_M(n)$, how much space can it use?

At most as much space as the number of time steps

$$s_M(n) \leq t_M(n)$$

If a Turing machine uses space $s_M(n)$, how long can it take?

At most exponential time in the amount of space used

$$t_M(n) \leq 2^{O(s_M(n))} \quad \text{if } s_M(n) \geq \log n$$

Reason:

Constant number of possibilities (say $K$) for each tape symbol

$n$ possible input head locations

$s_M(n)$ possible work head locations

Total number of configurations $\leq ns_M(n)K^{s_M(n)} \leq 2^{O(s_M(n))} \quad \text{if } s_M(n) \geq \log n$
PATH

PATH = \{ \langle G, s, t \rangle \mid \text{Directed graph } G \text{ has a directed path from node } s \text{ to node } t \}\}

As we will see, an important problem for space complexity

How much space is required for solving PATH?

BFS or DFS uses \( \geq n \) space \((n = |V(G)|)\)

We don’t know how to solve PATH in \( O(\log n) \) space, but we can solve it in \( O((\log n)^2) \) space
PATH in \((\log n)^2\) space

Main idea: **Recursion!**

If \(t\) is reachable from \(s\), must be reachable within \(n - 1\) steps
Solve the question “Is \(v\) reachable from \(u\) within \(k\) steps?” recursively

Try all intermediate nodes \(w\) and asks
“Is \(w\) reachable from \(u\) within \(k/2\) steps?”
“Is \(v\) reachable from \(w\) within \(k/2\) steps?”
If answer is YES to both sub-questions for some \(w\), then \(v\) reachable from \(u\) within \(k\) steps
Savitch’s algorithm

Recursively answer “Can \( u \) reach \( v \) within \( k \) steps?”

Algorithm 1 \( \text{PATH}(u, v, k) \)

\[
\begin{align*}
\text{if } k &= 0 \text{ then} \\
\text{ return whether } u &= v \\
\text{ elseif } k &= 1 \text{ then} \\
\text{ return whether } (u, v) &\in E \\
\text{ end if} \\
\text{for every vertex } w \text{ do} \\
\text{ if } \text{PATH}(u, w, \lfloor k/2 \rfloor) \text{ and } \text{PATH}(w, v, \lceil k/2 \rceil) \text{ then} \\
\text{ return true} \\
\text{ end if} \\
\text{ end for} \\
\text{ return false}
\end{align*}
\]
PATH in \((\log n)^2\) space

Depth of recursion: \(O(\log n)\)

Additional memory for each level: \(O(\log n)\)
to remember the intermediate node for this level

unlike time, space can be reused!

Overall space used: \(O((\log n)^2)\)
Aside: repeated squaring

To compute $A^n$, how many multiplications required?

To compute $A^n$:
- If $n = 0$, return 1
- If $n$ is even, recursively compute $B = A^{n/2}$ and return $B^2$
- If $n$ is odd, recursively compute $B = A^{(n-1)/2}$ and return $B^2 \cdot B$

$O(\log n)$ multiplications

When $A$ is the adjacency matrix and not a scalar,
repeated squaring is analogous to previous algorithm for PATH
Nondeterministic log-space

Why is PATH important?
Analogous to $P$ vs $NP$, we can consider the nondeterministic analog of $L$ and asks $L$ vs $NL$

A language $L$ is in $NL$ if $L$ can be decided by a nondeterministic Turing machine (with read-only input tape) in $O(\log n)$ space
**NL-completeness**

A language $B$ is **NL-complete** if

1. $B$ is in **NL**; and
2. every language $A$ in **NL log-space** reduces to $B$

We consider log-space reductions, because polynomial-time reductions are too coarse

**Theorem**

PATH is **NL-complete**

Assuming $L \neq NL$
PATH is \textbf{NL}-complete

PATH is in \textbf{NL}:

Nondeterministic Turing machine guesses a path from \( s \) to \( t \)
More precisely, the machine remembers the current node on the path and
guesses the next node

PATH is \textbf{NL}-hard:

For any language \( A \) in \textbf{NL}
Let \( N \) be a log-space nondeterministic Turing machine for \( A \)
Construct the directed graph \( G \) whose vertices are configurations of \( N \)
Let \( s \) be the initial configuration and \( t \) be the accepting configuration
PATH is \textbf{NL}-hard: details

Listing all $s_N(n)$ nodes/configurations can be done with $O(s_N(n))$ space.

Checking whether one configuration leads to another (whether one node has an edge to another) can be done in $O(s_N(n))$ space.

Since $s_N(n) = O(\log n)$, constructing $\langle G, s, t \rangle$ can be done in $O(\log n)$ space.

By modifying $N$, we may assume its accepting configuration is unique.
Caveat and consequences

Recall: $\textbf{NP} = \text{set of languages having polynomial-time verifier}$

A similar definition (with log-space verifier) is not unlikely to be true for $\textbf{NL}$

Intuitively, $\textbf{NL}$ machines do not have enough memory to remember all nondeterministic choices

Since $\text{PATH}$ is $\textbf{NL}$-complete and can be solved in $O((\log n)^2)$ spaces

Every problem in $\textbf{NL}$ can be solved in $O((\log n)^2)$ space!

(Savitch’s theorem)

Even though we believe $\textbf{NP}$-complete problems takes exponential amount of time compared to $\textbf{P}$ problems, space is another story
Hierarchy theorems
Hierarchy theorem

Given more space, can Turing machines/algorithms solve more problems?

Are there problems solvable in $n^3$ space but not in $n^2$ space?

Given any “nice” function $f : \mathbb{N} \rightarrow \mathbb{N}$, there is a language decidable in $O(f(n))$ space but not in $o(f(n))$ space

For example, $n^3$, $\log n$, $n \log n$ will be “nice”

(If a function does not always take integer values, such as $\log n$, we consider rounding down the output to an integer)
Space-constructible functions

Technical definition of “nice” is space-constructible

A function $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n) \geq \log n$, is space-constructible if the function mapping an input $w$ of length $n$ to the binary representation of $f(n)$ is computable by a Turing machine in space $O(f(n))$.

Space hierarchy theorem is therefore

Given any space-constructible function $f : \mathbb{N} \rightarrow \mathbb{N}$, there is a language decidable in $O(f(n))$ space but not in $o(f(n))$ space.
Corollary

For any $a < b$, there are functions computable in space $O(n^b)$ but not in space $O(n^a)$

Statement is intuitive
Hardest part: proving that all Turing machines with less space fails to solve a problem
The “difficult” problem

\[ L = \{ \langle M, w \rangle \mid \text{Turing machine } M \text{ rejects } \langle M, w \rangle \text{ in space } \leq f(n) \} \]
\[ n = |\langle M, w \rangle| \]

Need to show

1. \( L \) cannot be decided in space \( o(f(n)) \)
2. \( L \) can be decided in space \( O(f(n)) \)

An artificial problem

For technical reason, we assume the Turing machines \( M \) have constant-sized tape alphabet (such as 4), independent of \( n \)
Not solvable in space $o(f(n))$

$$L = \{ \langle M, w \rangle \mid \text{Turing machine } M \text{ rejects } \langle M, w \rangle \text{ in space } \leq f(n) \}$$

$n = |\langle M, w \rangle|$

Proof by contradiction

Suppose $L$ can be decided in space $o(f(n))$ by a Turing machine $D$

What happens if $M = D$ and $w$ is very long?

When $w$ is very long, $n$ is big, and $o(f(n))$ will be smaller than $f(n)$
Not solvable in space $o(f(n))$

$$L = \{\langle M, w \rangle \mid \text{Turing machine } M \text{ rejects } \langle M, w \rangle \text{ in space } \leq f(n) \}$$

$$n = |\langle M, w \rangle|$$

Case 1: If $D$ accepts $\langle D, w \rangle$
then $\langle D, w \rangle \in L$ (because $D$ decides $L$)
hence $D$ rejects $\langle D, w \rangle$ (by definition of $L$)

Case 2: If $D$ rejects $\langle D, w \rangle$
then $\langle D, w \rangle \notin L$ (because $D$ decides $L$)
hence $D$ doesn’t reject $\langle D, w \rangle$ (by definition of $L$)
Since $D$ decides $L$, $D$ accepts $\langle D, w \rangle$

Combining two cases $\Rightarrow$ contradiction
Solvable in space $O(f(n))$

$L = \{ \langle M, w \rangle \mid \text{Turing machine } M \text{ rejects } \langle M, w \rangle \text{ in space } \leq f(n) \}

n = |\langle M, w \rangle|$

Idea: simulate $M$

Since $M$ is supposed to use only $\leq f(n)$ space

Simulation can be done using $O(f(n))$ space

Keeping track of $M$’s states takes $O(\log n)$ space

If $M$ tries to use more than $f(n)$ space, aborts simulation and rejects

Here we use the assumption that $f(n)$ is space-constructible

Simulator needs to know how much tape space to allocate for simulating $M$
Solvable in space $O(f(n))$

$L = \{ \langle M, w \rangle \mid \text{Turing machine } M \text{ rejects } \langle M, w \rangle \text{ in space } \leq f(n) \}$

$n = |\langle M, w \rangle|$

Idea: simulate $M$

Challenge: $M$ may infinite-loop on $\langle M, w \rangle$

Solution:
Computation in space $f(n)$ goes through $2^{O(f(n))}$ configurations
If the same configuration appears twice, $M$ loops indefinitely

When simulating $M$, keeps track of the number of steps
If it exceeds $2^{O(f(n))}$, simulator rejects
This counter takes up additional $O(f(n))$ space
Conclusion

\[ L = \{ \langle M, w \rangle \mid \text{Turing machine } M \text{ rejects } \langle M, w \rangle \text{ in space } \leq f(n) \} \]

\[ n = |\langle M, w \rangle| \]

1. \( L \) cannot be decided in space \( o(f(n)) \)  
2. \( L \) can be decided in space \( O(f(n)) \)

Why this artificial problem?
Diagonalization

\[ L = \{ \langle M, w \rangle \mid \text{Turing machine } M \text{ rejects } \langle M, w \rangle \text{ in space } \leq f(n) \} \]

\[ n = |\langle M, w \rangle| \]

Need a problem not solvable by all Turing machines that runs in \( o(f(n)) \) space

That’s why \( L \) involves Turing machines running in small space
**Time hierarchy**

Similar theorem for time complexity

Given any time-constructible function $f : \mathbb{N} \to \mathbb{N}$, there is a language decidable in $O(f(n))$ time but not in $o(f(n)/\log n)$ time.

A function $f : \mathbb{N} \to \mathbb{N}$, where $f(n) \geq n \log n$, is time-constructible if the function mapping an input $w$ of length $n$ to the binary representation of $f(n)$ is computable by a Turing machine in time $O(f(n))$.

$$L = \{ \langle M, w \rangle \mid \text{Turing machine } M \text{ rejects } \langle M, w \rangle \text{ in } \leq f(n)/\log n \text{ time} \}$$

$n = |\langle M, w \rangle|$

Proof follows similar high-level strategy

1. $L$ cannot be decided in $o(f(n)/\log n)$ time
2. $L$ can be decided in $O(f(n))$ time