1 Measuring Time

1.1 Overview

Efficiency

- How do we measure the efficiency of a computational solution?
  - Many possible metrics could be used: running time, memory requirements, security, amount of communication, use of shared resource, quality of user interface, maintainability of code, etc.
  - In this class, we will focus on running time
- *Goal:* To develop a framework to compare solutions, and quantify computational difficulty in a platform independent manner.

Measuring Time

- The number of steps needed to solve a problem (typically) depends on the size of the input
  - Computing "4+5" is easier than computing "1298959829494+4128393208157"
- An algorithm may have different running times on inputs of the same size.
  - One can measure either: (upper bound on) the number of steps needed on *any* input of a given size *(worst case time)*, or the number of steps needed on *average* on inputs of a given size *(average case time)*. In this class we will focus on worst case times.
- Therefore, running time of an algorithm/TM is measured as function of input size

---

1.2 Time Bounded Classes

Time Bounded Complexity Classes

Definition 1. A (deterministic) Turing machine is said to run in *time* $t(n)$ if on any input u, the computation of M on u takes at most $t(|u|)$ steps

Definition 2. $L \in \text{DTIME}(t(n))$ iff there is a (deterministic) TM M that runs in time $t(n)$ and $L = \mathbf{L}(M)$

Linear Speedup

Theorem 3. *Let M be a k-tape deterministic TM running in time $t(n)$. For any constant $c > 0$, there is a $k+1$-tape TM M' such that $\mathbf{L}(M') = \mathbf{L}(M)$ and M' runs in time $ct(n) + n$. (Note, here we think of c being less than 1.)*

Proof. Let $L = \mathbf{L}(M)$. We will describe a machine M' which will simulate ℓ steps of M in 8 steps; if $\ell > \frac{8}{c}$, we will get the desired result. M' will have one more tape, a much larger tape alphabet, and control states than M.

- M' copies the input onto the additional tape in compressed form: ℓ successive symbols of M will be represented by one symbol in M'. Time taken is n. This additional tape will be used by M' as the "input" tape. While M' simulates M, it will also store all the other tapes of M in compressed format.
- M' uses the additional worktape as "input tape". The head positions of M, within the k symbols represented by current cells, is stored in finite control.

One *basic move* of M' (consisting of 8 steps), will simulate ℓ steps of M as follows.

- At the start of basic move, M' moves its tape heads one cell left, two cells right and one cell left, storing the symbols read in the finite control. Now, M' knows all symbols within the radius of k cells of any of M's tape heads. This takes 4 steps.
- M' knows the next k steps of M.
- Using any additional (at most) 4 steps, M' updates the contents of its tapes as a result of the k steps, and moves the heads appropriately. $\square$

The Big Oh!

Definition 4.
- $f(n) = O(g(n))$ if there are constants c, n_0 such that for $n > n_0$, $f(n) \leq cg(n)$. $g(n)$ is an *asymptotic upper bound.*
- $f(n) = \Omega(g(n))$ if there are constants c, n_0 such that for $n > n_0$, $f(n) \geq cg(n)$. $g(n)$ is an *asymptotic lower bound.*
- $f(n) = o(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. $f(n)$ is *asymptotically no more than* g(n).
- $f(n) = \omega(g(n))$ if $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$. $f(n)$ is *asymptotically at least* $g(n)$.

Remark
The above "asymptotic" notation serves two purposes

- It ignores constants because constant factor changes can occur due to change in platform, and the linear speedup theorem suggests that such improvements can be easily obtained.
- It ignores the running time on "small" instances, and only considers the "eventual" running time based on large instances.

Consequence

Corollary 5. *If $n = o(t(n))$ and M runs in time $t'(n)$ such that $t'(n) = O(t(n))$ then $\mathbf{L}(M) \in$ DTIME$(t(n))$.*

Proof. Consequence of the linear speedup theorem. □

1.3 Palindromes

Palindromes

Question
Recall, $L_{\text{pal}} = \{w \in \{0,1\}^* \mid w = w^R\}$ is the language of palindromes. What is an algorithm to recognize it?

One-Tape Turing Machine
Read the leftmost symbol, store in finite state, and mark it (i.e., replace it by a special symbol "*"). Move to the rightmost symbol on tape, read it, check if it matches with the leftmost symbol, and erase it (i.e., write $\sqcup$ on it). Move to the leftmost non-marked symbol, and repeat.

Number of steps on input of length n is

$$n + (n-2) + (n-4) \cdots + 2(\text{or } 1) = O(n^2)$$

Two-tape Turing Machine
Copy input onto second tape. Move the input head to the leftmost input symbol (with second tape head at the rightmost symbol). Compare symbols read on input tape and second tape, as the input head moves right and the second tape head moves left.

Number of steps on input of length $n = n + n + n = O(n)$.

Palindromes
Lower Bounds

Can we get a faster one-tape algorithm? No!

Proposition 6. *If M is a 1-tape TM such that $\mathbf{L}(M) = L_{\text{pal}}$ then running time of M is $\Omega(n^2)$.*

Proof. Beyond the scope of this course. □

Model for Measuring Time

- Which machine model should we use to measure time complexity of a problem? For palindromes we have

  - 1-Tape: $\Theta(n^2)$ [1]
  - 2-Tape: $\Theta(n)$
  - RAM mode: $\Theta(n)$

- Need a way to measure time complexity that is insensitive to changes in programming model.

---

2 Polynomial Time

2.1 Robust Complexity Classes

Robust Complexity Classes

- Complexity bounds for a problem should be platform independent
  - Small changes to model (like reducing tapes, or changing alphabet) should not affect the relevance of the results
  - Results should be valid even if one considers computational models other than Turing machines.
- Complexity classes should be closed under function composition
- Complexity classes should capture "interesting" real-world problems

---

Invariance Thesis

Any effective, mechanistic procedure can be simulated on a 1-tape Turing machine with only a polynomial slowdown (time $\geq n$).

Figure 1: Alonzo Church

Polynomial Time

Definition 7. $\mathrm{P} = \cup_k \mathrm{DTIME}(n^k)$

[1] We say $f(n)$ is $\Theta(g(n))$ if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$.

Efficient Computation
P captures the class of efficiently solvable problems because

1. Invariance thesis
2. Complexity class is not sensitive to problem encoding
3. Problems solvable in low-order polynomials
4. Moderate growth of polynomials versus the astounding growth rate of exponentials

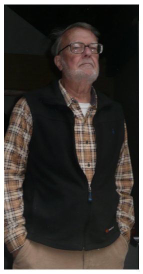


Figure 2: Alan Cobham

Invariance Thesis: A Caveat

Invariance theses may not hold! In 1994, Peter Shor gave a polynomial time algorithm to factor two numbers on a "Quantum Computer". No efficient factoring algorithm is known for traditional computers! However, we haven't managed to build a "real" quantum computer yet.

2.2 Cocke-Younger-Kasami Algorithm

Example problems in P

- Given a directed graph G, is there is a path from s to t?
- Given a DFA M and input w, is $w \in \mathbf{L}(M)$? i.e., $A_{\text{DFA}} = \{\langle M, w\rangle \mid M \text{ is a DFA and } w \in \mathbf{L}(M)\}$
- Given a DFA M, is $\mathbf{L}(M) = \emptyset$? i.e., $E_{\text{DFA}} = \{\langle M\rangle \mid M \text{ is a DFA and } \mathbf{L}(M) = \emptyset\}$
- Given a CFG G, is $\mathbf{L}(G) = \emptyset$? i.e., $E_{\text{DFA}} = \{\langle G\rangle \mid G \text{ is a CFG and } \mathbf{L}(M) = \emptyset\}$

Context-free Languages

Theorem 8 (Cocke-Younger-Kasami)**.** *Let $A_{\text{CFG}} = \{\langle G, w\rangle | G$ is a CFG in Chomsky Normal Form and $w \in \mathbf{L}(G)\}$. A_{CFG} is in P.*

Simple Solution
Note, that the following exponential time algorithm for A_{CFG} follows from the properties of Chomsky normal form grammars.

- Let $|w| = n$. Since G is in Chomsky Normal Form, w has a parse tree of size $2n-1$ iff $w \in L(G)$
- Construct all possible parse (binary) trees and check if any of them is a valid parse tree for w
- Number of parse trees of size $2n-1$ is k^{2n-1} where k is the number of variables in G. So algorithm is exponential in n!

CYK Algorithm
First Ideas

Notation
Suppose $w = w_1 w_2 \cdots w_n$, where $w_i \in \Sigma$. Let $w_{i,j}$ denote the substring of w starting at position i of length j. Thus, $w_{i,j} = w_i w_{i+1} \cdots w_{i+j-1}$

Main Idea
For every $A \in V$, and every $i \leq n$, $j \leq n+1-i$, we will determine if $A \overset{*}{\Rightarrow} w_{i,j}$.

Now, $w \in \mathbf{L}(G)$ iff $S \overset{*}{\Rightarrow} w_{1,n} = w$; thus, we will solve the membership problem.

How do we determine if $A \overset{*}{\Rightarrow} w_{i,j}$ for every A, i, j?

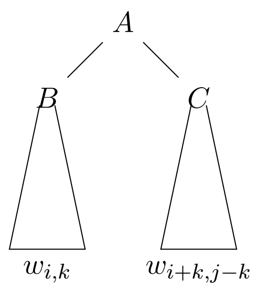
Base Case
Substrings of length 1

Observation
For any A, i, $A \overset{*}{\Rightarrow} w_{i,1}$ iff $A \rightarrow w_{i,1}$ is a rule.

- Since G is in Chomsky Normal Form, G does not have any ϵ-rules, nor any unit rules.

Thus, for each A and i, one can determine if $A \overset{*}{\Rightarrow} w_{i,1}$.

Inductive Step
Longer substrings



Suppose for every variable X and every $w_{i,\ell}$ ($\ell < j$) we have determined if $X \overset{*}{\Rightarrow} w_{i,\ell}$

- $A \overset{*}{\Rightarrow} w_{i,j}$ iff there are variables B and C and some $k < j$ such that $A \to BC$ is a rule, and $B \overset{*}{\Rightarrow} w_{i,k}$ and $C \overset{*}{\Rightarrow} w_{i+k,j-k}$
- Since k and $j - k$ are both less than j, we can inductively determine if $A \overset{*}{\Rightarrow} w_{i,j}$.

Cocke-Younger-Kasami (CYK) Algorithm

Algorithm maintains $X_{i,j} = \{A \mid A \overset{*}{\Rightarrow} w_{i,j}\}$.

```
 Initialize:   $X_{i,1} = \{A \mid A \to w_{i,1}\}$
for $j$ = 2 to $n$ do
    for $i$ = 1 to $n - j + 1$ do
        $X_{i,j} = \emptyset$
        for $k$ = 1 to $j - 1$ do
            $X_{i,j} = X_{i,j} \cup \{A \mid A \to BC,\ B \in X_{i,k},\ C \in X_{i+k,j-k}\}$
```

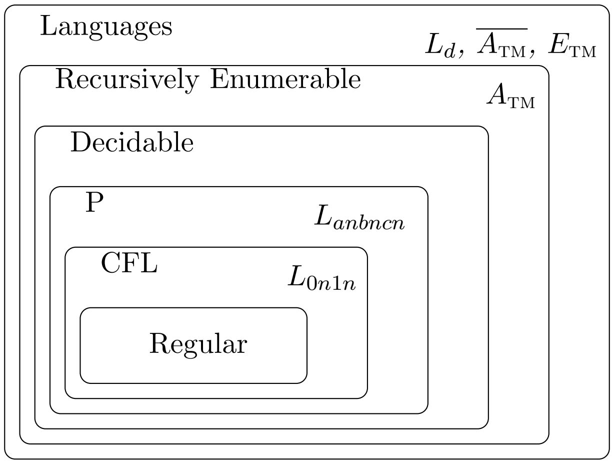
Correctness: After each iteration of the outermost loop, $X_{i,j}$ contains exactly the set of variables A that can derive $w_{i,j}$, for each i. Time $= O(n^3)$.

Example

Example 9. Consider grammar $S \to AB \mid BC$, $A \to BA \mid a$, $B \to CC \mid b$, $C \to AB \mid a$ Let $w = baaba$. The sets $X_{i,j} = \{A \mid A \overset{*}{\Rightarrow} w_{i,j}\}$:

j/i	1	2	3	4	5
5	$\{S, A, C\}$				
4	$\emptyset$	$\{S, A, C\}$			
3	$\emptyset$	$\{B\}$	$\{B\}$		
2	$\{S, A\}$	$\{B\}$	$\{S, C\}$	$\{S, A\}$	
1	$\{B\}$	$\{A, C\}$	$\{A, C\}$	$\{B\}$	$\{A, C\}$
	b	a	a	b	a

Big Picture . . . again


Languages
L_d, $\overline{A_{\text{TM}}}$, E_{TM}
Recursively Enumerable
A_{TM}
Decidable
P
L_{anbncn}
CFL
L_{0n1n}
Regular