Introduction to the Theory of Computation

Part III: Complexity Theory

7. Time Complexity

- P and NP
  - Measuring complexity
  - The class P
  - The class NP
- NP-Completeness
  - Polynomial time reducibility
  - The definition of NP-Completeness
  - The Cook-Levin Theorem
- NP-Complete Problems

Complexity

- So far we have classified problems by whether they have an algorithm at all.
- In real world, we have limited resources with which to run an algorithm:
  - one resource: time
  - another: storage space
- need to further classify decidable problems according to resources they require

Complexity

- Complexity Theory = study of what is computationally feasible (or tractable) with limited resources:
  - running *time*
  - storage *space*
  - number of *random bits*
  - degree of *parallelism*
  - rounds of *interaction*
  - *others…*

main focus

not in this course

Worst-Case Analysis

- Always measure resource (e.g. running time) in the following way:
  - as a function of the input length
  - value of the function is the maximum quantity of resource used over all inputs of given length
  - called "**worst-case analysis**"
- "input length" is the length of input string, which might encode another object with a separate notion of size

Time Complexity

Definition: the running time ("time complexity") of a TM M is a function

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

where f(n) is the maximum number of steps M uses on any input of length n.

- "M runs in time f(n)," "M is a f(n) time TM"

Analyze Algorithms

- Example: TM M deciding L = $\{0^k 1^k : k \geq 0\}$.

On input x:

- scan tape left-to-right, reject if 0 to right of 1
- repeat while 0's, 1's on tape:
  - scan, crossing off one 0, one 1
- if only 0's or only 1's remain, reject; if neither 0's nor 1's remain, accept

steps?

steps?

steps?

Analyze Algorithms

- We do not care about fine distinctions
  - e.g. how many additional steps M takes to check that it is at the left of tape
- We care about the behavior on large inputs
  - general-purpose algorithm should be "scalable"
  - overhead for e.g. initialization shouldn't matter in big picture

Measure Time Complexity

- Measure time complexity using asymptotic notation ("big-oh notation")
  - – disregard lower-order terms in running time
  - – disregard coefficient on highest order term
- example:

$$f(n) = 6n^3 + 2n^2 + 100n + 102781$$

  - – "$f(n)$ is order n^3"
  - – write $f(n) = O(n^3)$

Asymptotic Notation

Definition: given functions f, g: $\mathbf{N} \rightarrow \mathbf{R}^+$, we say f(n) = O(g(n)) if there exist positive integers c, n_0 such that for all $n \geq n_0$

$$f(n) \leq cg(n)$$

- meaning: f(n) is (asymptotically) less than or equal to g(n)
- E.g. $f(n) = 5n^4+27n$, $g(n)=n^4$, take $n_0=1$ and $c = 32$ ($n_0=3$ and $c = 6$ works also)

Analyze Algorithms

On input x:	
• scan tape left-to-right, reject if 0 to right of 1	O(n) steps
• repeat while 0's, 1's on tape:	$\leq$ n/2 repeats
• scan, crossing off one 0, one 1	O(n) steps
• if only 0's or only 1's remain, reject; if neither 0's nor 1's remain, accept	O(n) steps

- total = $O(n) + (n/2)O(n) + O(n) = O(n^2)$

Asymptotic Notation Facts

each bound asymptotically less than next

- "logarithmic": $O(\log n)$
  - $\log_b n = (\log_2 n)/(\log_2 b)$
  - so $\log_b n = O(\log_2 n)$ for any constant b; therefore suppress base when write it
- "polynomial": $O(n^c) = n^{O(1)}$
  - also: $c^{O(\log n)} = O(n^{c'}) = n^{O(1)}$
- "exponential": $O(2^{n^\delta})$ for $\delta > 0$

Time Complexity Class

- Recall:
  - a language is a set of strings
  - a complexity class is a set of languages
  - complexity classes we've seen:
    - Regular Languages, Context-Free Languages, Decidable Languages, RE Languages, co-RE languages

Definition: Time complexity class

TIME(t(n)) = {L | there exists a TM M that decides L in time O(t(n))}

Time Complexity Class

- We saw that $L = \{0^k1^k : k \geq 0\}$ is in $\mathrm{TIME}(n^2)$.
- Book: it is also in $\mathrm{TIME}(n \log n)$ by giving a more clever algorithm
- Can prove: $O(n \log n)$ time required on a single tape TM.
- How about on a multitape TM?

Multitape TMs

- 2-tape TM M deciding $L = \{0^k 1^k : k \geq 0\}$.

On input x:	
• scan tape left-to-right, reject if 0 to right of 1	O(n)
• scan 0's on tape 1, copying them to tape 2	O(n)
• scan 1's on tape 1, crossing off 0's on tape 2	O(n)
• if all 0's crossed off before done with 1's reject	
• if 0's remain after done with ones, reject; otherwise accept.	total: 3*O(n) = O(n)

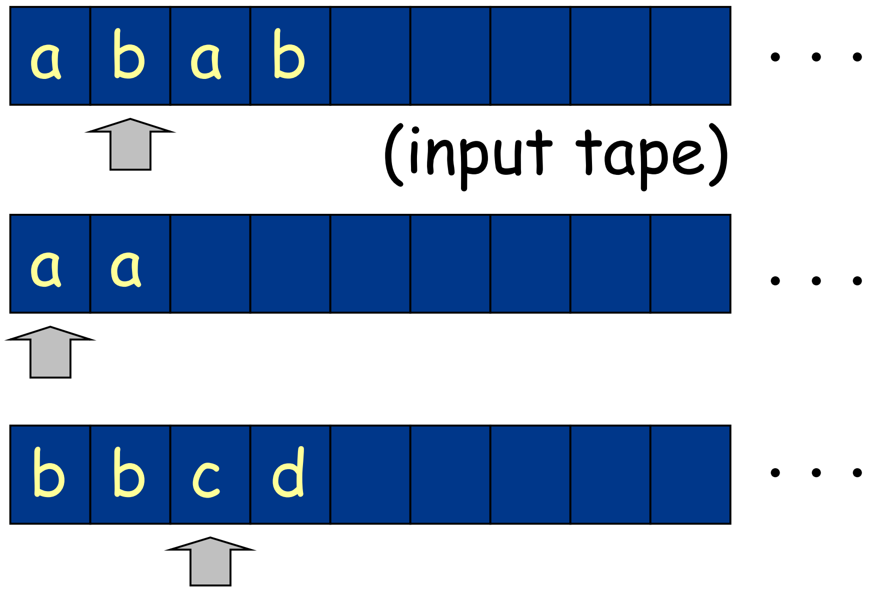
Multitape TMs

- Convenient to "program" multitape TMs rather than single-tape ones
  - equivalent when talking about decidability
  - not equivalent when talking about time complexity

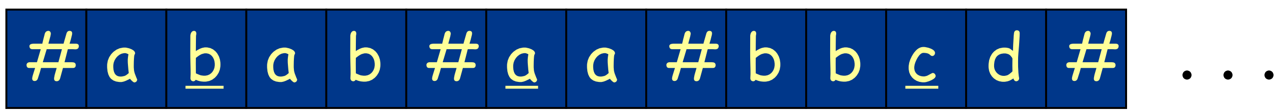
Theorem: Let $t(n)$ satisfy $t(n) \geq n$. Every $t(n)$ multitape TM has an equivalent $O(t(n)^2)$ single-tape TM.

Multitape TMs

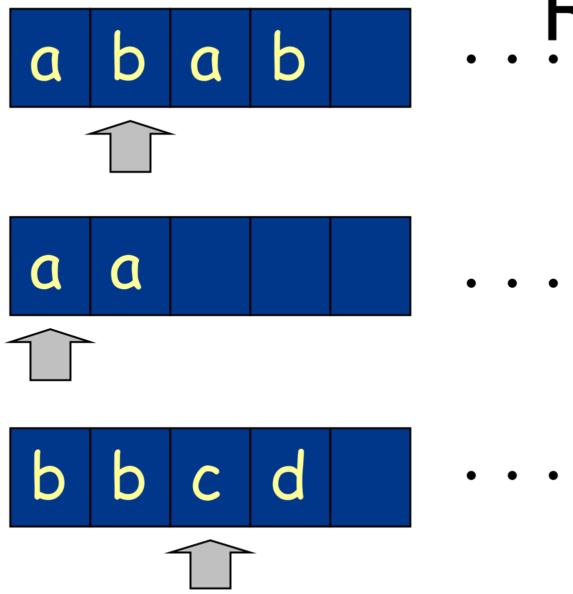
simulation of k-tape TM by single-tape TM:



- add new symbol **x** for each old x
- marks location of "virtual heads"



Multitape TMs

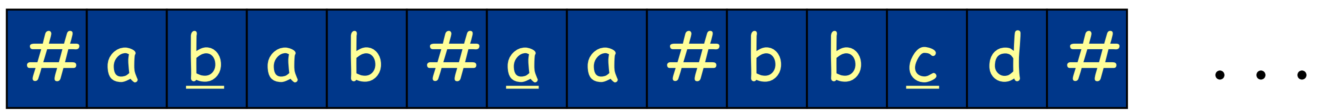


Repeat: **O(t(n)) times**

- scan tape, remembering the symbols under each virtual head in the state
O(k t(n)) = O(t(n)) steps

- make changes to reflect 1 step of M; if hit #, shift to right to make room.
O(k t(n)) = O(t(n)) steps

when M halts, erase all but 1st string
O(t(n)) steps



Extended Church-Turing Thesis

- The belief that TMs formalize our intuitive notion of an efficient algorithm is:

> The "extended" Church-Turing Thesis
>
> everything we can compute in time $t(n)$ on a physical computer can be computed on a Turing Machine in time $t(n)^{O(1)}$ (polynomial slowdown)

- quantum computers challenge this belief

“Polynomial Time Class” P

- interested in a coarse classification of problems.
  - treat any polynomial running time as “efficient” or “tractable”
  - treat any exponential running time as “inefficient” or “intractable”

Definition: “P” or “polynomial-time” is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing Machine.

$$P = \cup_{k \geq 1} \mathrm{TIME}(n^k)$$

Why P?

- insensitive to particular deterministic model of computation chosen ("*Any reasonable deterministic computational models are polynomially equivalent.*")
- empirically: qualitative breakthrough to achieve polynomial running time is followed by quantitative improvements from impractical (e.g. n^{100}) to practical (e.g. n^3 or n^2)

Why P?

- insensitive to particular deterministic model of computation chosen ("*Any* *reasonable* *deterministic*

ially

h to achieve

l by quantitative

g. n^{100}) to

What makes a model 'unreasonable'?

Partial answer: physical unrealistic requirements for the proper functioning of the machine.

Typical examples:

– Analog computing:

Infinite precision of the elementary components.

– Unbounded parallel computing:

Requires exponential space and energy.

– Time-travel computing…

Examples of Languages in P

- PATH = {<G, s, t> | G is a directed graph that has a directed path from s to t}

- RELPRIME = {<x, y> | x and y are relatively prime}

- A_{CFG} = {<G, w> | G is a CFG that generates string w}

Nondeterministic TMs

- Recall: nondeterministic TM
- informally, TM with several possible next configurations at each step
- formally, A NTM is a 7-tuple

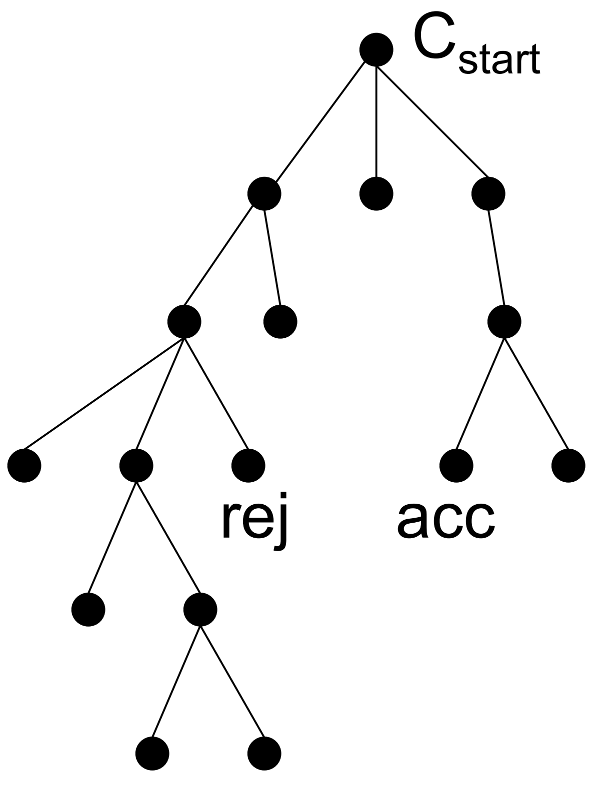
  $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where:

  – everything is the same as a TM except the transition function:

$$\delta : Q \times \Gamma \rightarrow \wp(Q \times \Gamma \times \{L, R\})$$

Nondeterministic TMs

visualize computation of a NTM M as a tree



- nodes are configurations
- leaves are accept/reject configurations
- M accepts if and only if there exists an accept leaf
- M is a decider, so no paths go on forever
- running time is max. path length

"Nondeterministic Polynomial Time Class" NP

Definition: TIME(t(n)) = {L | there exists a TM M that decides L in time O(t(n))}

$$P = \cup_{k \geq 1} \text{TIME}(n^k)$$

Definition: NTIME(t(n)) = {L | there exists a NTM M that decides L in time O(t(n))}

$$NP = \cup_{k \geq 1} \text{NTIME}(n^k)$$

Poly-Time Verifiers

- NP = {L | L is decided by some poly-time NTM}

- Very useful alternate definition of NP:

"certificate" or "proof"

Theorem: language L is in NP if and only if it is expressible as:

$$L = \{ x \mid \exists\, y, |y| \le |x|^k, \langle x, y\rangle \in R \}$$

efficiently verifiable

where R is a language in P.

- poly-time TM M_R deciding R is a "verifier"

Example

- HAMPATH = {<G, s, t> | G is a directed graph with a Hamiltonian path from s to t}

is expressible as

HAMPATH = {<G, s, t> | ∃p for which <<G, s, t>, p> ∈ R},

R = {<<G, s, t>, p> | p is a Ham. path in G from s to t}

  - *p* is a certificate to verify that <G, s, t> is in HAMPATH
  - R is decidable in poly-time

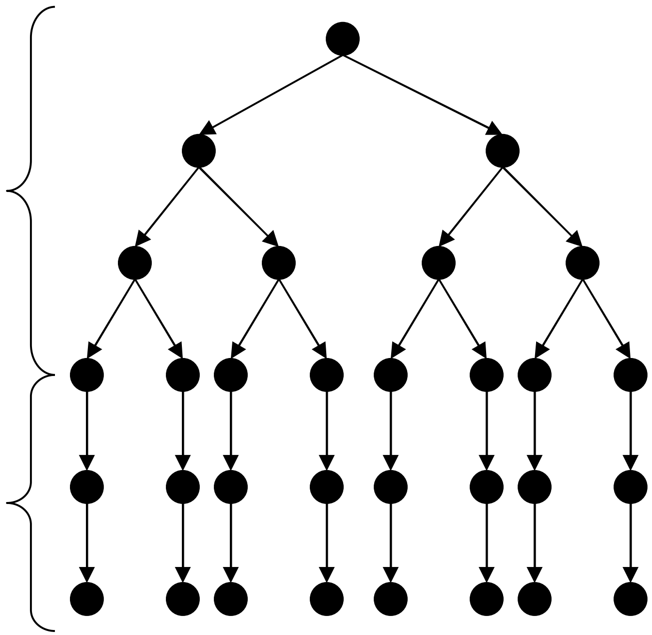
Poly-Time Verifiers

$L \in NP$ iff. $L = \{ x \mid \exists\, y, |y| \leq |x|^k, \langle x, y \rangle \in R \}$

Proof: ($\Leftarrow$) give poly-time NTM deciding L

phase 1: "guess" y with $|x|^k$ nondeterministic steps

phase 2: decide if $\langle x, y \rangle \in R$



Poly-Time Verifiers

Proof: ($\Rightarrow$) given L $\in$ NP, describe L as:

$$L = \{ x \mid \exists\, y, |y| \leq |x|^k, <x, y> \in R \}$$

- L is decided by NTM M running in time n^k
- define the language

R = {<x, y> | y is an accepting computation history of M on input x}

- check: accepting history has length $\leq |x|^k$
- check: M accepts x iff $\exists y, |y| \leq |x|^k, <x, y> \in R$

Why NP?

- not a realistic model of computation
- but, captures important computational feature of many problems:

  exhaustive search works

- contains **huge** number of natural, practical problems
- many problems have form:

$$L = \{ x \mid \exists y \text{ s.t. } \langle x, y \rangle \in R \}$$

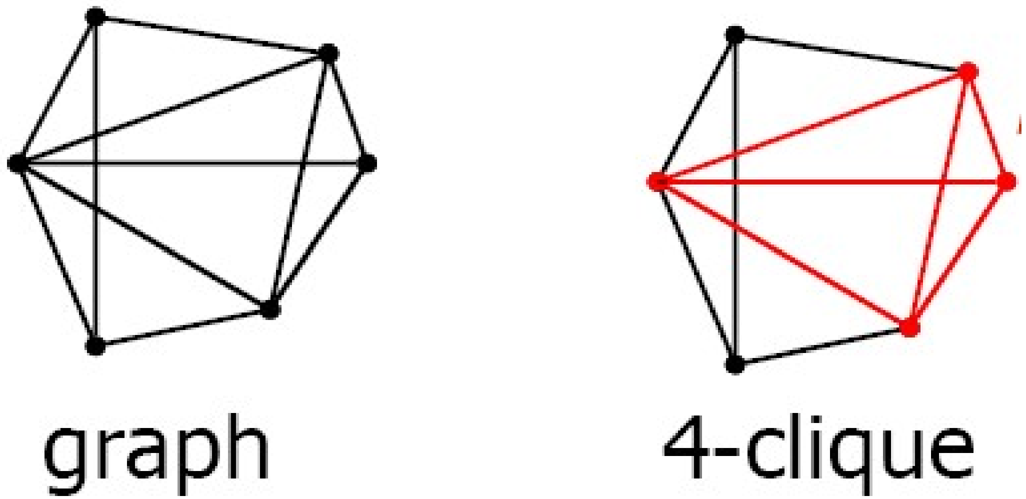
problem requirements

object we are seeking

efficient test: does y meet requirements?

Examples of Languages in NP

- A *clique* in an undirected graph is a subgraph, wherein every two nodes are connected.



- CLIQUE = {<G, k> | graph G has a k-clique}

CLIQUE is NP

- **Proof:** construct an NTM N to decide CLIQUE in poly-time

N = "On input <G, k>, where G is a graph:

1. Nondeterministically select a subset *c* of k nodes of G.
2. Test whether G contains all edges connecting nodes in *c*.
3. If yes, *accept*; otherwise, *reject*."

CLIQUE is NP

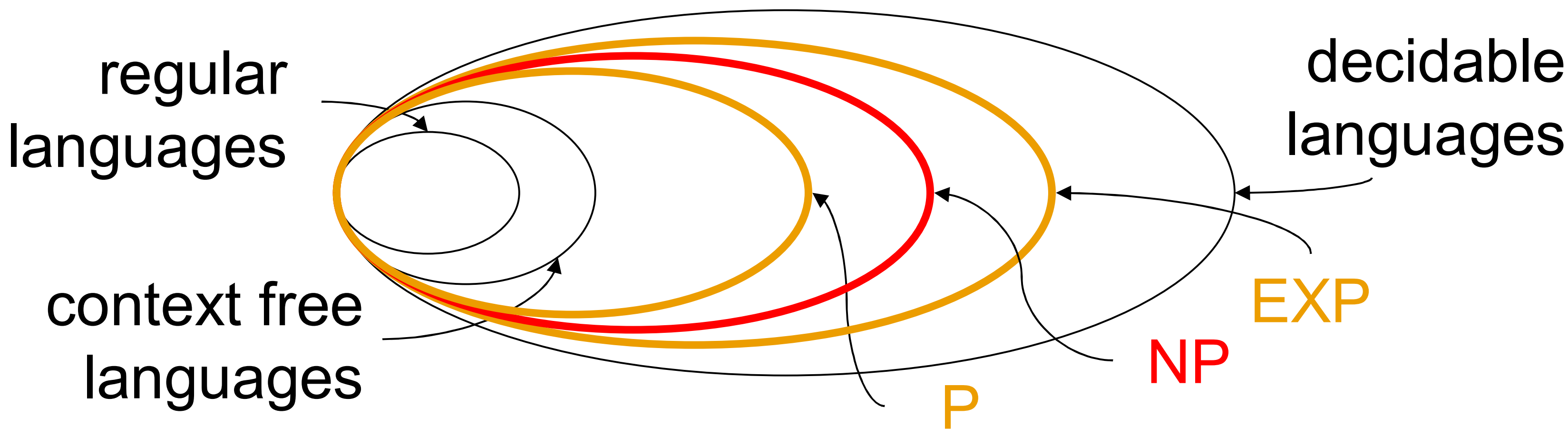
- **Alternative Proof:** CLIQUE is expressible as

CLIQUE = {<<G, k> | ∃c for which <<G, k>, c> ∈ R},

R = {<<G, k>, c> | c is a set of k nodes in G, and all the k nodes are connected in G}

– R is decidable in poly-time

NP in relation to P and EXP



- P $\subseteq$ NP (poly-time TM ***is*** a poly-time NTM)
- NP $\subseteq$ EXP = $\cup_{k \geq 1}$ TIME(2^{n^k})
  - configuration tree of n^k-time NTM has $\leq b^{n^k}$ nodes
  - can traverse entire tree in $O(b^{n^k})$ time

we do not know if either inclusion is proper