

Time Complexity

Sipser pages 247 - 283

Measuring complexity

- How complex is a computation?
 - What can we measure
 - Time
 - Space
 - Other things?
- Time complexity
 - One thing we might count is the number of transitions a TM makes.
 - The number of instructions on a x86 cpu
 - The number of seconds (why is this not a good measure?)

An example: $A = \{0^k 1^k \mid k \geq 0\}$

- Low level TM description
- $M_1(w) =$ where w is a string
 - Scan across tape and reject if a 0 is found to the right of a 1
 - Repeat if both 0's and 1's remain on tape
 - Scan across tape crossing off a single 0 and a single 1
 - If 0's remain after all 1's have been crossed off, or if 1's remain after all 0's have been crossed off, reject. Otherwise if neither 0's or 1's remain, accept.

Time as a function of input size n

- Scan across tape and reject if a 0 is found to the right of a 1 (n -steps, and n -steps more to reposition head at left)
 - Repeat if both 0's and 1's remain on tape (at most $n/2$ repetitions)
 - Scan across tape crossing off a single 0 and a single 1 (about n -steps)
 - If 0's remain after all 1's have been crossed off, or if 1's remain after all 0's have been crossed off, reject. Otherwise if neither 0's or 1's remain, accept. (about n -steps)
- We consider the worst case performance
 - The best we can do, even from a low level description, is estimate
 - About $n + n + (n/2 * n) + n$

Estimating

- Can we be more formal about estimating?
- Consider $f(n) = 7n^3 + 6n^2 + 100n + 67896$
- What function is a good estimate of f ?
- Lets compute the ratio $(f n) / 67896$

n	(f n) / 67896
0	1.0
1	1.0016643
2	1.0041239
3	1.0079975
4	1.0139036
5	1.0224608
6	1.0342877
7	1.0500029
8	1.070225
9	1.0955726

Looks like a pretty good estimate

n	(f n) / 67896
10	1.126664310121362
20	1.8895958524802638
50	14.181925297513844
100	105.12984564628255
150	351.16790385295155
200	829.6202427241664
250	1617.8110050665723
300	2793.064333686815

Well, maybe not

Asymptotic analysis

- Try and find a formula that the actual cost converges with as the size of the problem goes to infinity.

As the problem size gets bigger, then we want the actual cost and the estimated cost to get very close.

Ratio $(f n) / (n^3)$

1	68009.0
3	2534.777777777778
10	76.496
20	16.037
50	7.703168
100	7.137896
150	7.064561777777778
200	7.040987
250	7.029945344
300	7.023625777777778

When n is small, its not very accurate, but as n gets larger than 50 or so, the ratio converges to 7

The higher order terms are much more accurate as n gets large.

The highest order term is in general the best estimate as n gets large.

Big O notation

- Let f and g be functions from Nat to Real
- We say that $f(n) = O(g(n))$
- If positive numbers C and n_0 exist such that for every integer $n \geq n_0$ $f(n) \leq C \times g(n)$

We see that $f(n)$ is “within a constant C ” of $g(n)$.
Constants don't matter

- We say that $g(n)$ is a asymptotic upper bound for $f(n)$

Polynomials and Big O

- If $f(n)$ is a polynomial
 - $\sum c_n x^j$

Then the highest order term is a good candidate for Big O

$$f(n) = 7n^3 + 6n^2 + 100^n + 67896$$

$$f(n) = O(n^3)$$

Logarithms and Big O

- Suppose $x = \log_2 n$
 - This means that $2^x = n$
- If we change the base from 2 to 10
- $\log_2 n = 0.301029995 \times \log_{10} n$
- Since constants don't matter we surpress the base when we have logarithms in Big O notation

$$h(n) = 3n(\log_2 n) + 5n + 6$$

$$h(n) = O(n \times \log n)$$

Manipulating Big O

- Sums of Big O, collapse to the largest term
 - $O(n^2) + O(n) = O(n^2)$

small o notation

- Big O says that one term is no more than
- Small o says that one term is strictly less
- Let f and g be functions from Nat to Real
- We say that $f(n) = o(g(n))$

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

Analyzing Algorithms

- Scan across tape and reject if a 0 is found to the right of a 1 ($O(n)$ steps, and $O(n)$ -steps more to reposition head at left)
- Repeat if both 0's and 1's remain on tape (at most $O(n/2)$ repetitions)
 - Scan across tape crossing off a single 0 and a single 1 ($O(n)$ -steps)
- If 0's remain after all 1's have been crossed off, or if 1's remain after all 0's have been crossed off, reject. Otherwise if neither 0's or 1's remain, accept. (about $O(n)$ -steps)

- $O(n) + O(n) + O(n/2) \times O(n) + O(n) =$

- $O(n) + O(n) + O(n) \times O(n) + O(n) =$

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

- $O(n^2)$

Definition: Time complexity

- Let $t: \text{Nat} \rightarrow \text{Real}$ be a function
- Define the *time complexity class* $\text{TIME}(t(n))$
- To be the collection of all languages that are decidable by an $O(n(t))$ time Turing Machine

Algorithm can affect Time

- Consider another TM for $\{0^k1^k \mid k \geq 0\}$
- $M_2(w) =$ where w is a string
 - Scan across tape and reject if a 0 is found to the right of a 1.. $O(n)$
 - Repeat as long as some 0's and some 1's remain on the tape. $O(\log n)$ repetitions
 - Scan across tape checking if the total number of 0's and 1's is even or odd, reject if it is odd. $O(n)$
 - Scan across the tape again, crossing off every other 0, starting either the first 0, and every other 1, starting with the first 1 $O(n)$
 - If no 0's no 1's remain, accept, otherwise reject. $O(n)$
- $O(n) + O(\log n)(O(n) + O(n)) + O(n)$
- $O(n \log n)$

Model can affect Time

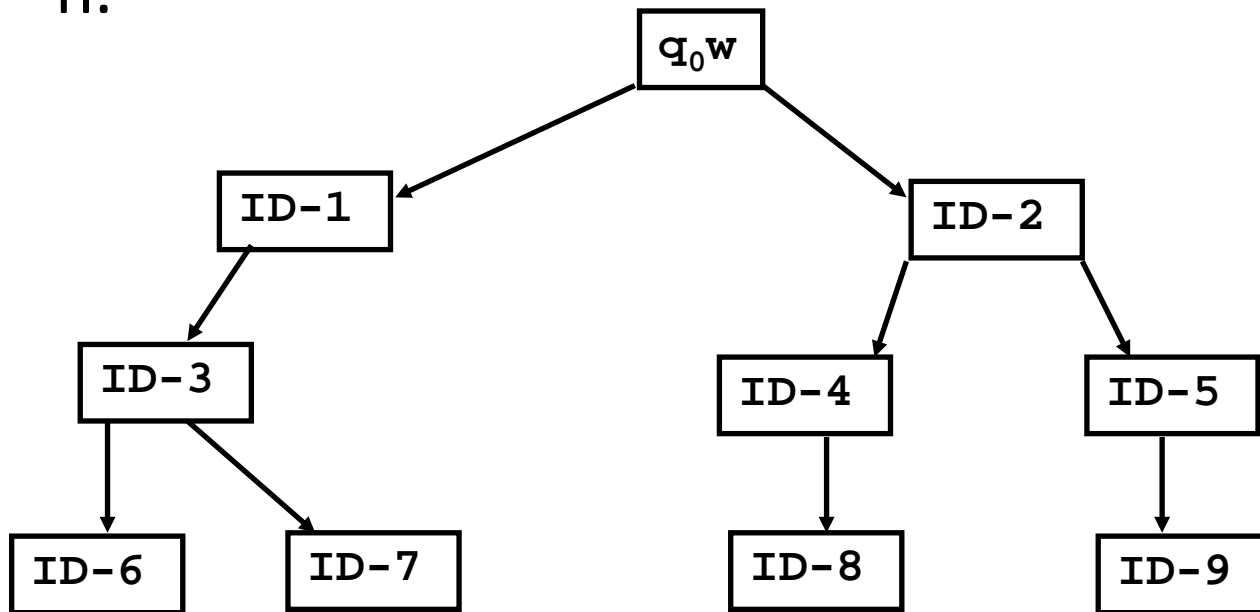
- Consider 2-tape TM for $\{0^k1^k \mid k \geq 0\}$
- $M_3(w) =$ where w is a string
 - Scan across tape and reject if a 0 is found to the right of a 1. $O(n)$
 - Scan across the 0's on tape-1, until the first 1. At the same time copy the 0's onto tape-2. $O(n)$
 - Scan across the 1's on tape-1 until the end of input. For each 1 read on tape-1, cross off a 0 on tape-2. If all 0's are crossed off before all the 1's are read, reject. $O(n)$
 - If all the 0's have been crossed off, accept, If any 0's remain, reject. $O(n)$
- $O(n) + O(n) + O(n) + O(n) = O(n)$

Theorem

- Let $t(n)$ be a function where $t(n) \geq n$. Then every $t(n)$ time multi-tape TM has an equivalent $O(t^2(n))$ time single-tape machine.
- The proof of this is in analogy with how a single-tape machine emulates a multi-tape machine. See text book for details

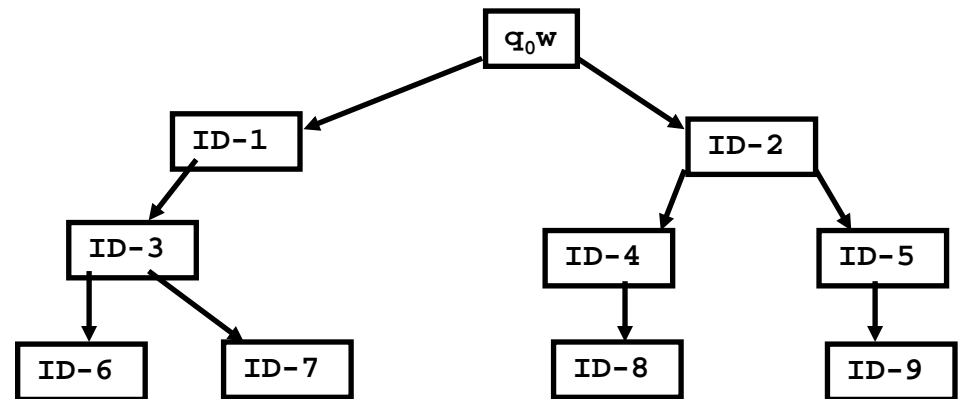
Deterministic v.s. Non-Deterministic

- Let N be a nondeterministic TM that is a decider. The running time of N is a function $f: \text{Nat} \rightarrow \text{Nat}$, where $f(n)$ is the maximum number of steps that N uses on any branch of its computation on any input of length n .



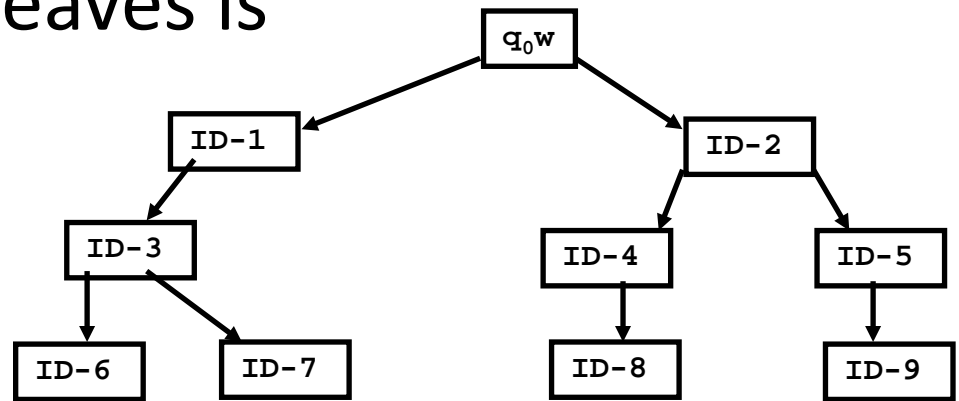
Theorem

- Let $t(n)$ be a function where $t(n) \geq n$. Then every $t(n)$ time non-deterministic TM has an equivalent $2^{O(t(n))}$ time deterministic machine.
- We simulate a nondeterministic TM by searching its computation tree (in breadth first order)



Proof

- Every path has length at most $t(n)$, because we said it was a $t(n)$ decider.
- Every node (ID- m) has at most b children for some constant b (the transition table determines this)
- The total number of leaves is bounded by $b^{t(n)}$



Proof continued

- The simulation might have to visit every leaf
- The number of leaves are bounded by $b^{t(n)}$
- The time to get to a leaf is bounded by $t(n)$
- So the time is $O(t(n)b^{t(n)}) = 2^{O(t(n))}$

Thoughts on Complexity

- Algorithm can affect time complexity
- Computational model can affect complexity
- Non determinism can affect complexity
- Encoding of data (base 1 vs base 2) can affect complexity

- For expressivity, all reasonable models are equivalent.
- For complexity many things can change the complexity class.

More thoughts

- Lots of difference between time complexities caused by algorithm, data-encoding, machine model, etc.
 - The biggest increase is squaring (or polynomial)
- On the other hand the increase in time between deterministic and non-deterministic was exponential.
- So we look at things that distinguish between polynomial and exponential changes

Why make the choice between polynomial and exponential

Exponential growth is so fast, exponential algorithms are not practical

n	n^2	2^n
1	1	2
2	4	4
3	9	8
10	100	1024
15	225	32768
20	400	1048576
50	2500	1125899906842624
100	10000	1267650600228229401496703205376
200	40000	1606938044258990275541962092341162602522202993782792835301376
300	90000	2.03×10^{90}
		$10^{82} \cong$ number of atoms in the universe

More reasons

- Changes in time complexity caused by algorithm, data-encoding, machine model, can all be described by polynomials
- So there is a qualitative difference between polynomial and exponential
 - Polynomial differences are small
 - Exponential differences are large (exponential algorithms are rarely useful)

Analogy

- For expressivity, all reasonable models are equivalent.
- For complexity, all reasonable deterministic models are polynomial equivalent
- Non-deterministic models are can be exponentially more complex

What's important

- Are n^3 algorithms better than n^2
 - They are both polynomial
 - Why don't we make a distinction
 - Should we care (Yes, but)
- It is a matter of perspective
 - When it comes to whether a problem is too-hard the distinction between polynomial and exponential seems a good one
 - This is backed up by experience

The Class P

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

$$P = \bigcup_k \text{TIME}(n^k)$$

1. P is invariant for all models that are polynomial equivalent to deterministic TM
2. P roughly corresponds to effectively solvable problems on a computer

- Once a polynomial time algorithm is found for a problem, usually a key insight has been gained into the class that problem represents
- Brute force (search based) are often exponential and don't offer such insights

Examples of problems in P

- Graph algorithms
 - The Path Problem
- Relative Primeness
 - Euclid's algorithm
- A_{CFG}

High level descriptions without reference to features about a particular model.

Describe algorithm using numbered stages

1. Find polynomial Big O bound on number of stages
2. Show each stage can be implemented in polynomial time

The PATH problem

- $M \langle G, s, t \rangle =$ where G is a graph with nodes s and t
 1. Place a mark on node S
 2. Repeat until no additional nodes can be marked
 1. Scan all the edges of G , if an edge (a, b) is found where a is marked and b is not marked, mark b
 3. If t is marked, accept, else reject

How might we argue this is polynomial?

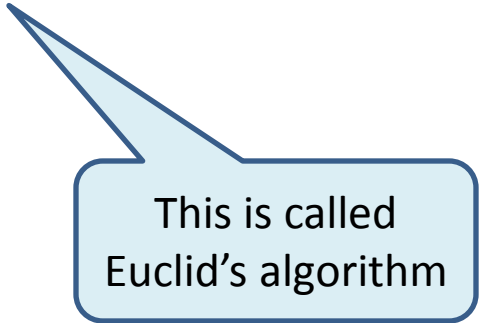
Relative Primeness

- $E \langle x, y \rangle =$ where x, y , are Natural numbers

A. Repeat until $y=0$

1. Assign $x \leftarrow \text{mod } x \ y$
2. Exchange x and y

B. Output x



This is called
Euclid's algorithm

- $R \langle x, y \rangle =$ where x, y , are Natural numbers
 - Run E on $\langle x, y \rangle$
 - If the result is 1, accept else reject

$$A_{\text{CFG}} \in P$$

- Let G be CFG in Chomsky Normal Form
- $D \langle w \rangle =$ where $w = w_1 \dots w_n$
 1. If $w = \epsilon$, and $S \rightarrow \epsilon$ is a rule, accept
 2. For $i = 1$ to n
 1. For each variable A
 1. Test whether $A \rightarrow b$ is a rule, where $b = w_i$
 2. If so place A in $\text{table}(i, i)$
 3. For $l = 2$ to n -- l is length of substring
 1. For $i = 1$ to $n - l + 1$ -- i is the start
 1. For $j = i + l - 1$ -- j is the end
 1. For $k = l$ to $j - 1$ -- k is the split position
 1. If $\text{table}(i, k)$ contains B and $\text{table}(k + 1, j)$ contains C , put A in $\text{table}(i, j)$
 4. If S in in $\text{table}(1, n)$, accept, otherwise, reject

Verifiers

- Some problems are hard to solve, but easy to check.

Suppose a path consists

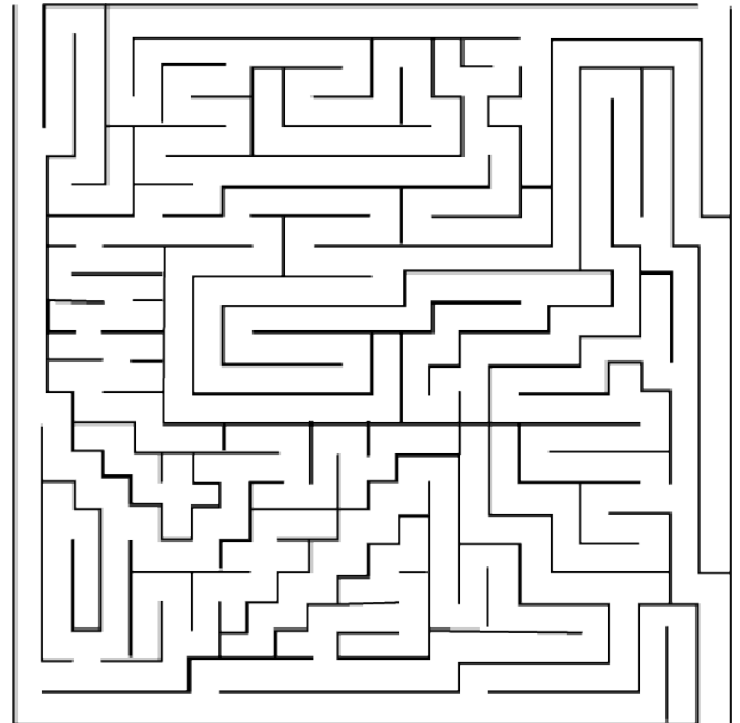
A sequence of N S E W

Moves.

Can you find a solution to

The maze? Can you check

a solution?



Definition

- A verifier for a language A is an algorithm V , where
- $A = \{ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c \}$

w is a solution, and c is called the certificate, that V may use to check w

- We measure the time of a verifier in terms of the length of w .
- A polynomial verifier runs in polynomial time in the length of w
- A language is polynomial verifiable if it has a polynomial verifier

Example verifiers

- Composite = $\{x \mid x = p \times q, \text{ for integer } p, q > 1\}$
 - A verifier takes in x , and a divisor for x , c , (the certificate). If $x \bmod c == 0$ then accept
- HAMPATH =
 - $\{ \langle G, s, t \rangle \mid G \text{ is a directed graph with a hamiltonian path from } s \text{ to } t \}$
 - Is G a valid graph.
 - Is there a path from s to t in G
 - Does the path visit every node?
 - Given a potential path $[n_1, \dots, n_m]$ can we verify $\langle G, s, t \rangle$ this in polynomial time?
 - Can we find a path for $\langle G, s, t \rangle$ in polynomial time?

Verifying HAMPATH

- $N1 \langle G, s, t, c = [n_1, \dots, n_m] \rangle =$ where G is a graph, and s, t are nodes in G , and the certificate is a sequence of m nodes
 - Verify that for each n_i in $[n_1, \dots, n_m]$ that n_i is a node in G , and m is the number of nodes in G
 - Check for repetitions, if any reject
 - Check if $n_1 = s$ and $n_m = t$, if either check fails then reject
 - For each i between 1 and $(m-1)$ check whether (n_i, n_{i+1}) is an edge of G . If any are not then reject. Otherwise all tests have passed so accept.

The Class NP

- Definition
 - The class NP is the class of languages that have polynomial time verifiers.
- Theorem
 - A language is in NP iff it is decided by some nondeterministic polynomial time Turing Machine

Is HAMPATH in NP

- Given a potential path $[n_1, \dots, n_m]$ can we verify it is in HAMPATH $\langle G, s, t \rangle$ in polynomial time on a *deterministic TM* ?, Or can we find such a path in polynomial time on a *non-deterministic TM*?
- N1 $\langle G, s, t \rangle =$ where G is a graph, and s, t are nodes in G
 - Nondeterministically, write a list of m numbers $[n_1, \dots, n_m]$ where n_i is a node in G , and m is the number of nodes in G
 - Check for repetitions, if any reject
 - Check if $n_1 = s$ and $n_m = t$, if either check fails then reject
 - For each i between 1 and $(m-1)$ check whether (n_i, n_{i+1}) is an edge of G . If any are not then reject. Otherwise all tests have passed so accept.

Is this nondeterministic machine in P

Two ways

- Polynomial **determinist verifiers** and polynomial **non deterministic solvers** are equivalent
- Two things to prove
 1. $A \in NP \Rightarrow A$ has nondeterministic polynomial solver
 2. A has a nondeterministic polynomial solver $\Rightarrow A \in NP$

$A \in NP \Rightarrow A$ has nondeterministic polynomial solver

- Given verifier V that runs in time n^k
- Construct Nondeterministic TM N
- $N \langle w \rangle =$ where n is a string of length n
 - Nondeterministically select a string c of length at most n^k
 - Run V on input $\langle w, c \rangle$
 - If V accepts, then accept, otherwise reject

A has a A nondeterministic polynomial solver
 $\Rightarrow A \in NP$

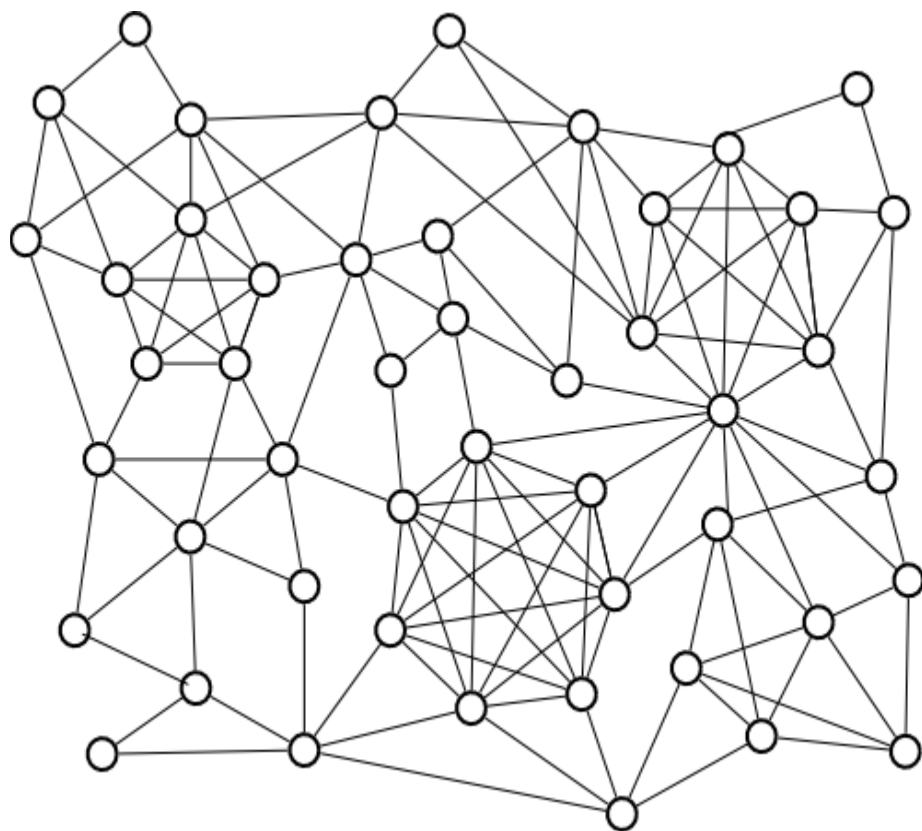
- Given a Nondeterministic TM, N , that solves A
- Construct a polynomial time verifier, V
- $V \langle w, c \rangle$ where w and c are strings
 - Simulate N on input w , treating each symbol of c as a description of the nondeterministic choice to make at each step
 - If the branch, guided by c , of N 's computation accepts, then accept, otherwise reject

Problems in NP

- Graph problems
 - The N-Clique problem
- Subset Sum

The N-clique problem

- A clique is a set of nodes chosen from a graph, where every node in the cliques is connected by an edge.
- An N-clique is a clique with n nodes



A verifier

- Let a set of nodes (a potential clique) be the certificate.
- $V \langle G, K, c \rangle =$
 1. Test whether c is a set of nodes in G
 2. Test whether G contains all edges connecting nodes in c
 3. If both pass, then accept, else reject
- Can we build a nondeterministic Solver?

The subset sum problem

- Let $S = \{x_1, x_2, \dots, x_n\}$ be a set of integers
- Let t be a target number
- Is there a subset of S whose sum adds up to t (duplicates are allowed)

- Formally SUBSETSUM =
 - $\{ \langle S, t \rangle \mid S = \{x_1, x_2, \dots, x_n\}$
 - $\quad \quad \quad , \text{ exists } y = \{y_1, \dots, y_m\},$
 - $\quad \quad \quad , \sum_i y_i = t \}$

- For example SUBSETSUM $\langle \{4, 11, 16, 21, 27\}, 25 \rangle$ accepts because $4 + 21 = 25$

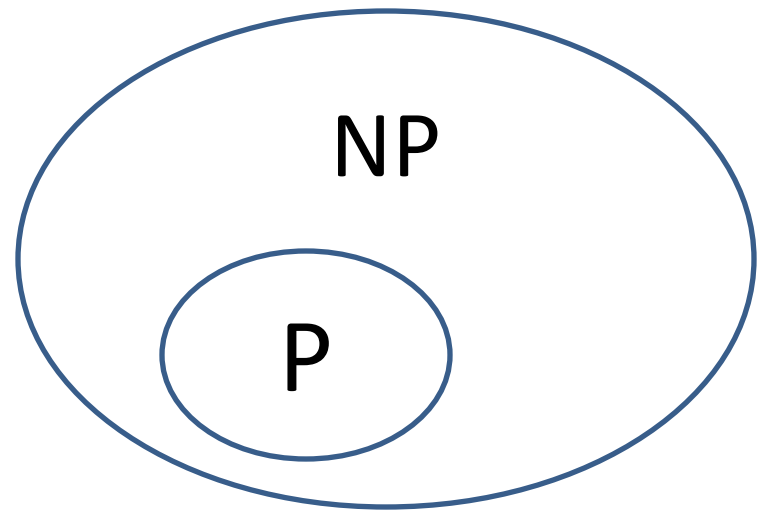
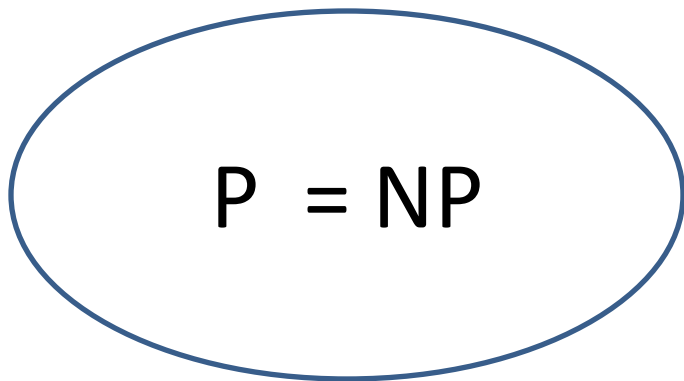
Two solutions

- Polynomial deterministic Verifier
 - $V \langle S, t, c \rangle = \dots$

- Polynomial non-deterministic Decider
 - $N \langle S, t \rangle = \dots$

is $P = NP$?

- Both P and NP are sets of languages
- Could it be that $P = NP$
- Possible choices



- The best deterministic method for simulating nondeterministic languages takes exponential time
- $NP \subseteq EXPTIME = \bigcup_k TIME(2^{n^k})$
- Could it be that there is a smaller deterministic class?

Thoughts on $P=NP$

- Recall
 - P is those problems that can be decided quickly
 - NP is those problems that can be verified quickly
- Most mathematicians think $P \neq NP$ because so many people have tried to prove $P=NP$ and come up empty. But that is hardly a proof.
- What about a direct proof of $P \neq NP$, no one knows how to do this. It requires showing there does not exist an algorithm such that ...

NP completeness

- Stephen Cook and Leonid Levin discovered that some NP problems are expressive enough to encode all other NP problems.
- Such problems are called NP-complete
 - Examples include boolean satisfiability and Hamiltonian Path
- Thus if any NP-complete problem has a polynomial solution, then all NP problems have a polynomial solution.

Polynomial time function

- A function $f: \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function if some polynomial time TM, M , exists that halts with just $F(w)$ on its tape, when started on any input w .

- Does this look familiar?

Polynomial time reducability

- A language, A , is polynomial time mapping reducible (or simply polynomial time reducible) to a language, B , written $A \leq_p B$, if a polynomial time computable function $f: \Sigma^* \rightarrow \Sigma^*$ exists, where for every w ,

$$w \in A \iff f(w) \in B$$

- The function f is called the polynomial time reduction of A to B

Polynomial Reducability Theorems

- $A \leq_p B$ and $B \in P$ then $A \in P$

Does this look familiar?

Proof

Let M be the polynomial time algorithm deciding B

Let f be the polynomial time reduction from A to B

$N \langle w \rangle =$

1. Compute $f(w)$
2. Run M on input $f(w)$ and output whatever M outputs

Definition

- A language B is NP-complete if it satisfies 2 conditions
 1. B is in NP, and
 2. Every A in NP is polynomial time reducible to B

$$\text{For all } A \in \text{NP} . A \leq_p B$$

Showing P or NP-complete

- If B is NP-complete and $B \leq_p C$, for $C \in \text{NP}$, then C is NP-complete
 - The most common “B” is the language boolean satisfiability
- Compare with
- $A \leq_p B$ and $B \in P$ then $A \in P$

Cook-Levin Theorem

- If we had 1 NP-complete language we could use it to show other languages are NP complete.
- The first language to be shown was NP-complete was boolean satisfiability (SAT)
 - The proof of this is in the text. It has 2 steps
 1. Show SAT is NP
 2. Show that every other NP language is polynomial reducible to SAT
 - Most other NP-complete results are gotten by the theorem on the previous page.