MA/CSSE 474
Theory of Computation

Computational Complexity

Announcements

- Don't forget the course evaluations on Banner Web.
  - If a 90%+ response rate for either section, everyone in that section gets a 5% bonus on the final exam
- Final Exam Monday 6-10PM. O 269
  - You can bring 3 double-sided sheets of paper
  - Covers whole course, but
  - Much more emphasis on later stuff
  - Includes several problems of the "which language class is this in?" flavor.

Accepting, Rejecting, Halting, and Looping

Consider :

$L_1 = \{<M, w>: M \text{ rejects } w\}$.

$L_2 = \{<M, w>: M \text{ does not halt on } w\}$.

$L_3 = \{<M, w>: M \text{ is a deciding TM and rejects } w\}$.

What About These?

$L_1 = \{\mathtt{a}\}$. **[in D]**

$L_2 = \{<M> : M \text{ accepts } \mathtt{a}\}$. **[in SD/D]**

$L_3 = \{<M> : L(M) = \{\mathtt{a}\}\}$. **[HW 14]**

$L = \{<M_a, M_b> : \varepsilon \in L(M_a) - L(M_b)\}$

R is a reduction from ¬H. *R*(<*M*, *w*>) =

1. Construct the description of *M*#(*x*) that operates as follows:
   1.1. Erase the tape.
   1.2. Write *w*.
   1.3. Run *M* on *w*.
   1.4. Accept.
2. Construct the description of *M*?(*x*) that operates as follows:
   2.1. Accept.
3. Return <*M*?, *M*#>.

If *Oracle* exists and semidecides *L*, *C* = *Oracle*(*R*(<*M*, *w*>)) semidecides ¬H: *M*? accepts everything, including ε. So:

- <*M*, *w*> ∈ ¬H: *L*(*M*?) - *L*(*M*#) =
- <*M*, *w*> ∉ ¬H: *L*(*M*?) - *L*(*M*#) =

The Problem View	***The Language View***	***Status***
Does TM *M* have an even number of states?	{<*M*> : *M* has an even number of states}	D
Does TM *M* halt on *w*?	H = {<*M*, *w*> : *M* halts on *w*}	SD/D
Does TM *M* halt on the empty tape?	H_ε = {<*M*> : *M* halts on ε}	SD/D
Is there any string on which TM *M* halts?	H_{ANY} = {<*M*> : there exists at least one string on which TM *M* halts }	SD/D
Does TM *M* halt on all strings?	H_{ALL} = {<*M*> : *M* halts on Σ*}	¬SD
Does TM *M* accept *w*?	A = {<*M*, *w*> : *M* accepts *w*}	SD/D
Does TM *M* accept ε?	A_ε = {<*M*> : *M* accepts ε}	SD/D
Is there any string that TM *M* accepts?	A_{ANY} {<*M*> : there exists at least one string that TM *M* accepts }	SD/D

Does TM M accept all strings?	A_{ALL} = {<M> : $L(M) = \Sigma^*$}	¬SD
Do TMs M_a and M_b accept the same languages?	EqTMs = {<M_a, M_b> : $L(M_a) = L(M_b)$}	¬SD

Does TM M not halt on any string?	$H_{\neg ANY}$ = {<M> : there does not exist any string on which M halts}	¬SD
Does TM M not halt on its own description?	{<M> : TM M does not halt on input <M>}	¬SD
Is TM M minimal?	TM_{MIN} = {<M>: M is minimal}	¬SD
Is the language that TM M accepts regular?	TMreg = {<M> : $L(M)$ is regular}	¬SD
Does TM M accept the language A^nB^n?	A_{anbn} = {<M> : $L(M) = A^nB^n$}	¬SD

Undecidable Problems About CFLs

1. Given a CFL L and a string s, is $s \in L$? (decidable)
2. Given a CFL L, is $L = \varnothing$?
3. Given a CFL L, is $L = \Sigma^*$?
4. Given CFLs L_1 and L_2, is $L_1 = L_2$?
5. Given CFLs L_1 and L_2, is $L_1 \subseteq L_2$?
6. Given a CFL L, is $\neg L$ context-free?
7. Given a CFL L, is L regular?
8. Given two CFLs L_1 and L_2, is $L_1 \cap L_2 = \varnothing$?
9. Given a CFL L, is L inherently ambiguous?
10. Given PDAs M_1 and M_2, is M_2 a minimization of M_1?
11. Given a CFG G, is G ambiguous?

Complexity Classes

course.setOverviewMode(true);

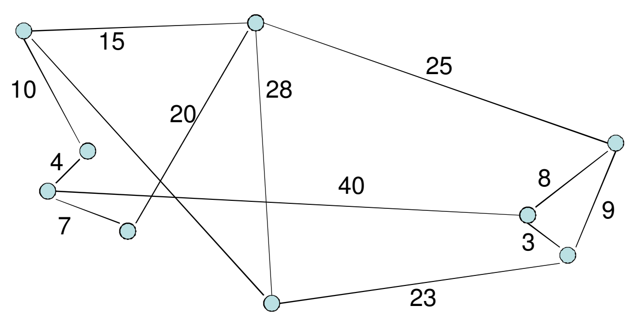
Asymptotic Analysis Review

in case it's been a while …

Are All Decidable Languages Equal?

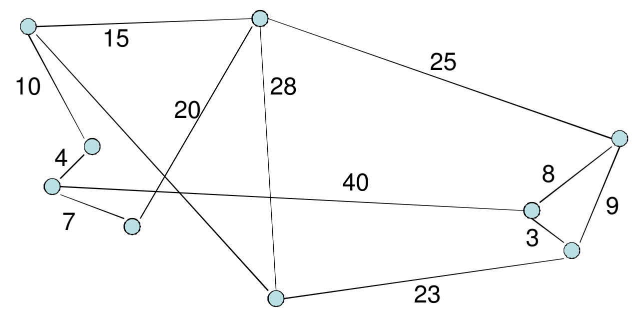
- (ab)*
- $WW^R = \{ww^R : w \in \{\mathtt{a}, \mathtt{b}\}^*\}$
- $WW = \{ww : w \in \{\mathtt{a}, \mathtt{b}\}^*\}$
- SAT = {*w* : *w* is a wff in Boolean logic and *w* is satisfiable}
- TSP (Traveling Salesman Problem). Next slides …

The Traveling Salesman Problem



Given *n* cities and the distances between each pair of them, find the shortest tour that returns to its starting point and visits each other city exactly once along the way.

The Traveling Salesman Problem



Given *n* cities:

Choose a first city	*n*
Choose a second	*n*-1
Choose a third	<u>*n*-2</u>
…	***n*!**

The Traveling Salesman Problem

Can we do better than *n*!

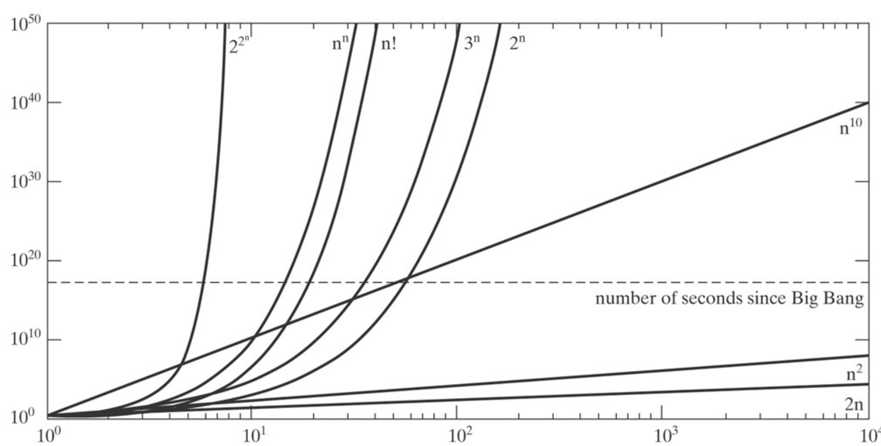
- First city doesn't matter.
- Order doesn't matter.

So we get (*n*-1!)/2.

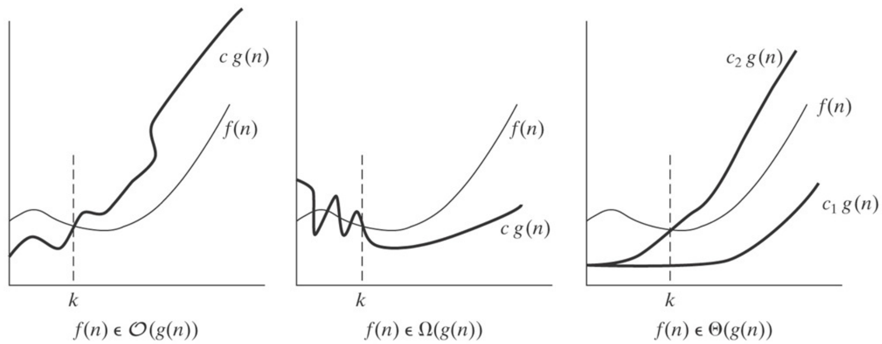
The Growth Rate of *n*!

2	2	11	479001600
3	6	12	6227020800
4	24	13	87178291200
5	120	14	1307674368000
6	720	15	20922789888000
7	5040	16	355687428096000
8	40320	17	6402373705728000
9	362880	18	121645100408832000
10	3628800	19	2432902008176640000
11	39916800	36	$3.6 \cdot 10^{41}$

Growth Rates of Functions



Asymptotic Dominance



Asymptotic Dominance - $\mathcal{O}$

$f(n) \in \mathcal{O}(g(n))$ iff there exists a positive integer k and a positive constant c such that:

$\forall n \geq k\ (f(n) \leq c\, g(n))$.

Alternatively, if the limit exists:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$$

Or, g grows at least as fast as f does.

Summarizing $\mathcal{O}$

$\mathcal{O}(c) \subseteq \mathcal{O}(\log_a n) \subseteq \mathcal{O}(n^b) \subseteq \mathcal{O}(d^n) \subseteq \mathcal{O}(n!) \subseteq \mathcal{O}(n^n)$

o (little oh)

Asymptotic strong upper bound: $f(n) \in o(g(n))$ iff, for every positive c, there exists a positive integer k such that:

$$\forall n \geq k \; (f(n) < c\, g(n)).$$

Alternatively, if the limit exists:

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

In this case, we'll say that *f* is "little-oh" of *g* or that *g* grows strictly faster than *f* does.

Ω

- ***Asymptotic lower bound***: $f(n) \in \Omega(g(n))$ iff there exists a positive integer k and a positive constant c such that:

  $\forall n \geq k\ (f(n) \geq c\, g(n))$.

In other words, ignoring some number of small cases (all those of size less than k), and ignoring some constant factor c, $f(n)$ is bounded from below by $g(n)$.

Alternatively, if the limit exists:

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

In this case, we'll say that f is "big-Omega" of g or that g grows no faster than f.

ω

- ***Asymptotic strong lower bound***: $f(n) \in \omega(g(n))$ iff, for every positive c, there exists a positive integer k such that:

  $\forall n \geq k\ (f(n) > c\, g(n))$.

Alternatively, if the required limit exists:

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

In this case, we'll say that f is "little-omega" of g or that g grows strictly slower than f does.

Θ

$f(n) \in \Theta(g(n))$ iff there exists a positive integer k and positive constants c_1, and c_2 such that:

$$\forall n \geq k\ (c_1\ g(n) \leq f(n) \leq c_2\ g(n))$$

Or:

$f(n) \in \Theta(g(n))$ iff:
$f(n) \in \mathcal{O}(g(n))$, and
$g(n) \in \mathcal{O}(f(n))$.

Or:

$f(n) \in \Theta(g(n))$ iff:
$f(n) \in \mathcal{O}(g(n))$, and
$f(n) \in \Omega(g(n))$.

Is $n^3 \in \Theta(n^3)$?
Is $n^3 \in \Theta(n^4)$?
Is $n^3 \in \Theta(n^5)$?

Tackling Hard Problems

1. Use a technique that is guaranteed to find an optimal solution and likely to do so quickly. Linear programming:

The Concorde TSP Solver found an optimal route that visits 24,978 cities in Sweden.

http://www.tsp.gatech.edu/concorde.html

2. Use a technique that is guaranteed to run quickly and find a "good" solution, but not necessarily optimal.

http://en.wikipedia.org/wiki/Travelling_salesman_problem#Heuristic_and_approximation_algorithms

The Complexity Zoo

The attempt to characterize the decidable languages by their complexity:

http://qwiki.stanford.edu/wiki/Complexity_Zoo

See especially the Petting Zoo page.

All Problems Are Decision Problems

The Towers of Hanoi

Requires at least enough time to write the solution.

By restricting our attention to decision problems, the length of the answer is not a factor.

Encoding Types Other Than Strings

The length of the encoding matters.

Integers: use any base other than 1.

111111111111 vs 1100
111111111111111111111111111111 vs 11110

$\log_a x = \log_a b \log_b x$

- PRIMES = $\{w : w$ is the binary encoding of a prime number$\}$

Encoding Types Other Than Strings

Graphs: use an adjacency matrix:

	1	2	3	4	5	6	7
1			•				
2				•			
3		•					
4					•		
5							
6							
7							

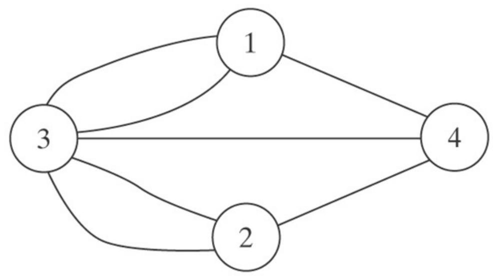
Or a list of edges:

`101/1/11/11/10/10/100/100/101`

Graph Languages

- CONNECTED = {<*G*> : *G* is an undirected graph and *G* is connected}.

- HAMILTONIANCIRCUIT = {<*G*> : *G* is an undirected graph that contains a ***Hamiltonian circuit***}.



Characterizing Optimization Problems as Languages

- TSP-DECIDE = {<*G*, *cost*> : <*G*> encodes an undirected graph with a positive distance attached to each of its edges and *G* contains a Hamiltonian circuit whose total cost is less than <*cost*>}.

Choosing A Model of Computation

We'll use Turing machines:

- Tape alphabet size?
- How many tapes?
- Deterministic vs. nondeterministic?

Measuring Time and Space Requirements

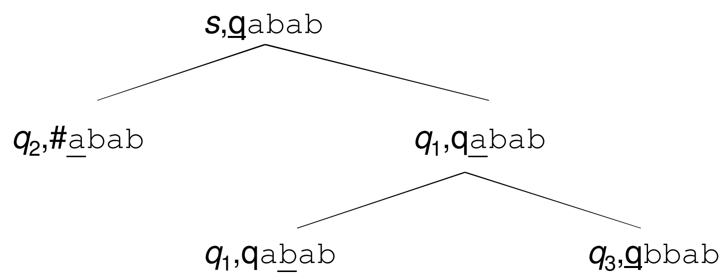
timereq(*M*) is a function of *n*:

- If *M* is a ***deterministic*** Turing machine that halts on all inputs, then:

  timereq(*M*) = *f*(*n*) = the maximum number of steps that *M* executes on any input of length *n*.

Measuring Time and Space Requirements

- If *M* is a ***nondeterministic*** Turing machine all of whose computational paths halt on all inputs, then:



timereq(*M*) = *f*(*n*) = the number of steps on the longest path that *M* executes on any input of length *n*.

Measuring Time and Space Requirements

spacereq(*M*) is a function of *n*:

- If *M* is a ***deterministic*** Turing machine that halts on all inputs, then:

  spacereq(*M*) = *f*(*n*) = the maximum number of tape squares that *M* reads on any input of length *n*.

- If *M* is a ***nondeterministic*** Turing machine all of whose computational paths halt on all inputs, then:

  spacereq(*M*) = *f*(*n*) = the maximum number of tape squares that *M* reads on any path that it executes on any input of length *n*.

Algorithmic Gaps

We'd like to show for a language L:

1. Upper bound: There exists an algorithm that decides *L* and that has complexity C_1.
2. Lower bound: Any algorithm that decides *L* must have complexity at least C_2.
3. $C_1 = C_2$.

If $C_1 = C_2$, we are done. Often, we're not done.

Algorithmic Gaps

Example: TSP

- Upper bound: *timereq* $\in \mathcal{O}(2^{(n^k)})$.
- Don't have a lower bound that says polynomial isn't possible.

We group languages by what we know. And then we ask: "Is class CL_1 equal to class CL_2?"

A Simple Example of Polynomial Speedup

Given a list of n numbers, find the minimum and the maximum elements in the list. Or, as a language recognition problem:

L = {<*list of numbers*, $number_1$, $number_2$>:
$number_1$ is the minimum element of the list and
$number_2$ is the maximum element}.

(23, 45, 73, 12, 45, 197; 12; 197) $\in L$.

A Simple Example of Polynomial Speedup

The straightforward approach:

simplecompare(*list*: list of numbers) =
max = *list*[1].
min = *list*[1].
For i = 2 to length(*list*) do:
If *list*[*i*] < *min* then *min* = *list*[*i*].
If *list*[*i*] > *max* then *max* = *list*[*i*].

Requires 2(n-1) comparisons. So *simplecompare* is $\mathcal{O}(n)$.

But we can solve this problem in (3/2)(n-1) comparisons.

How?

A Simple Example of Polynomial Speedup

efficientcompare(*list*: list of numbers) =
 max = *list*[1].
 min = *list*[1].
 For *i* = 3 to length(*list*) by 2 do:
  If *list*[*i*] < list[*i*-1] then:
   If *list*[*i*] < *min* then *min* = *list*[*i*].
   If *list*[*i*-1] > *max* then *max* = *list*[*i*-1].
  Else:
   If *list*[*i*-1] < min then min = *list*[*i*-1].
   If *list*[*i*] > max then max = *list*[*i*].
 If length(*list*) is even then check the last element.

Requires 3/2(*n*-1) comparisons.

String Search

```
t:    a b c a b a b c a b d

p:    a b c d

        a b c d

          a b c d

              . . .
```

String Search

simple-string-search(t, p: strings) =
  $i = 0$.
  $j = 0$.
  While $i \le |t| - |p|$ do:
    While $j < |p|$ do:
      If $t[i+j] = p[j]$ then $j = j + 1$.
        Else exit this loop.
      If $j = |p|$ then halt and accept.
        Else:
          $i = i + 1$.
          $j = 0$.
  Halt and reject.

Let n be $|t|$ and let m be $|p|$. In the worst case (in which it doesn't find an early match), *simple-string-search* will go through its outer loop almost n times and, for each of those iterations, it will go through its inner loop m times.

So *timereq*(*simple-string-search*) $\in \mathcal{O}(nm)$.

K-M-P algorithm is $\mathcal{O}(n+m)$

Replacing an Exponential Algorithm with a Polynomial One

- Context-free parsing can be done in $\mathcal{O}(n^3)$ time instead of $\mathcal{O}(2^n)$ time. (CYK algorithm)

- Finding the greatest common divisor of two integers can be done in $\mathcal{O}(\log_2(max(n, m)))$ time instead of exponential time.

The Language Class P

$L \in$ P iff

- there exists some deterministic Turing machine M that decides L, and
- $timereq(M) \in \mathcal{O}(n^k)$ for some k.

We'll say that L is ***tractable*** iff it is in P.

Closure under Complement

Theorem: The class P is closed under complement.

Proof: If M accepts L in polynomial time, swap accepting and non accepting states to accept $\neg L$ in polynomial time.

Defining Complement

- CONNECTED = {<*G*> : *G* is an undirected graph and *G* is connected} is in P.
- NOTCONNECTED = {<*G*> : *G* is an undirected graph and *G* is not connected}.
- ¬CONNECTED = NOTCONNECTED ∪ {strings that are not syntactically legal descriptions of undirected graphs}.

¬CONNECTED is in P by the closure theorem. What about NOTCONNECTED?

If we can check for legal syntax in polynomial time, then we can consider the universe of strings whose syntax is legal. Then we can conclude that NOTCONNECTED is in P if CONNECTED is.

Languages That Are in P

- Every regular language.
- Every context-free language since there exist context-free parsing algorithms that run in $\mathcal{O}(n^3)$ time.
- Others:
  - $A^nB^nC^n$
  - Nim

To Show That a Language Is In P

- Describe a one-tape, deterministic Turing machine.
- It may use multiple tapes. Price:
- State an algorithm that runs on a conventional computer.
  Price:

How long does it take to compare two strings?

q	a	a	a	;	a	a	a	q	…

Bottom line: If ignoring polynomial factors, then just describe a deterministic algorithm.

Regular Languages

Theorem: Every regular language can be decided in linear time. So every regular language is in P.

Proof: If L is regular, there exists some DFSM M that decides it. Construct a deterministic TM M' that simulates M, moving its read/write head one square to the right at each step. When M' reads a q, it halts. If it is in an accepting state, it accepts; otherwise it rejects.

On any input of length n, M' will execute $n + 2$ steps.

So $timereq(M') \in \mathcal{O}(n)$.

Context-Free Languages

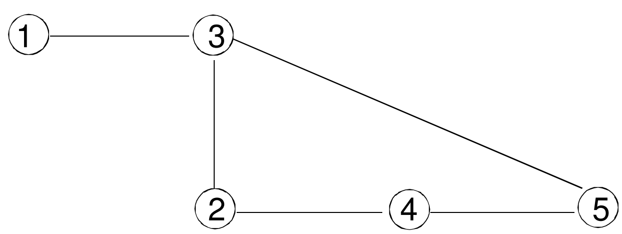
Theorem: Every context-free language can be decided in $\mathcal{O}(n^{18})$ time. So every context-free language is in P.

Proof: The Cocke-Kasami-Younger (CKY) algorithm can parse any context-free language in time that is $\mathcal{O}(n^3)$ if we count operations on a conventional computer. That algorithm can be simulated on a standard, one-tape Turing machine in $\mathcal{O}(n^{18})$ steps.

WE could get bogged down in the details of this, but w ewon't!

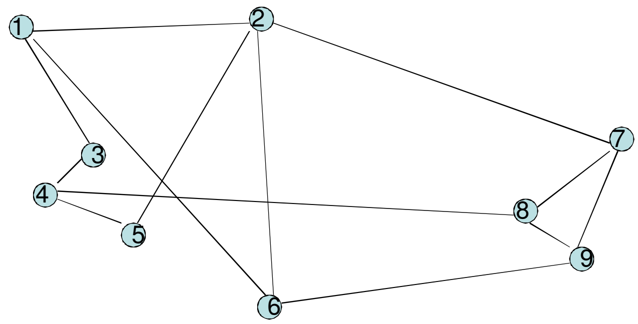
Graph Languages

Represent a graph $G = (V, E)$ as a list of edges:



`101/1/11/11/10/10/100/100/101/11/101`

Graph Languages



CONNECTED =
{<*G*> : *G* is an undirected graph and
G is connected}.

Is CONNECTED in P?

CONNECTED is in P

connected(<*G* = (*V*, *E*>) =
1. Set all vertices to be unmarked.
2. Mark vertex 1.
3. Initialize *L* to {1}.
4. Initialize *marked-vertices-counter* to 1.
5. Until *L* is empty do:
    - 5.1. Remove the first element from *L*. Call it *current-vertex*.
    - 5.2. For each edge *e* that has *current-vertex* as an endpoint do:
        Call the other endpoint of *e next-vertex*. If *next-vertex* is not already marked then do:
            Mark *next-vertex*.
            Add *next-vertex* to *L*.
            Increment *marked-vertices-counter* by 1.
6. If *marked-vertices-counter* = |*V*| accept. Else reject.

Analyzing *connected*

- Step 1 takes time that is $\mathcal{O}(|V|)$.
- Steps 2, 3, and 4 each take constant time.
- The loop of step 5 can be executed at most $|V|$ times.
  - Step 5.1 takes constant time.
  - Step 5.2 can be executed at most $|E|$ times. Each time, it requires at most $\mathcal{O}(|V|)$ time.
- Step 6 takes constant time.

So *timereq*(*connected*) is:

$$|V| \cdot \mathcal{O}(|E|) \cdot \mathcal{O}(|V|) = \mathcal{O}(|V|^2|E|).$$

But $|E| \leq |V|^2$. So *timereq*(*connected*) is:

$$\mathcal{O}(|V|^4).$$

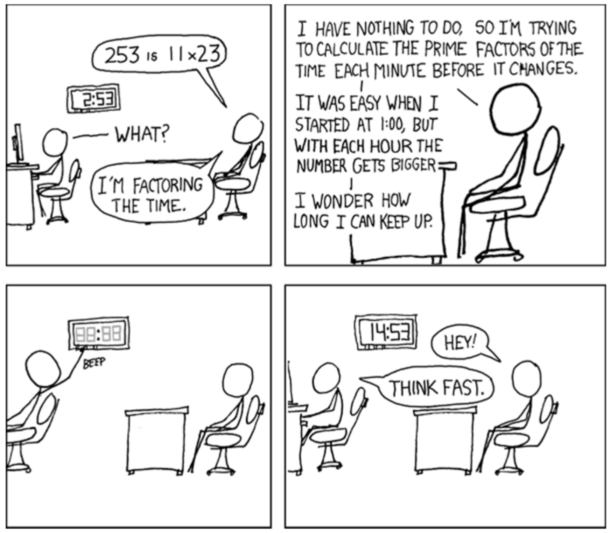
Primality Testing

RELATIVELY-PRIME =
{<*n*, *m*> : *n* and *m* are integers that are relatively prime}.

PRIMES =
{*w* : *w* is the binary encoding of a prime number}

COMPOSITES =
{*w* : *w* is the binary encoding of a nonprime number}

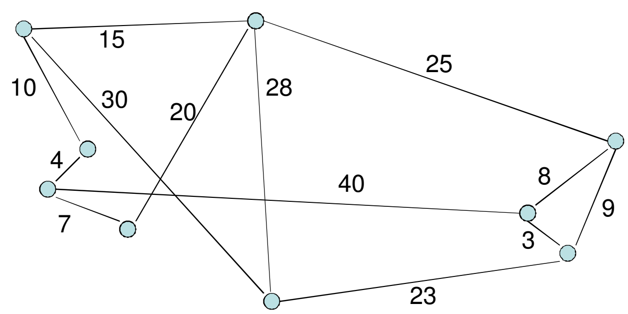
But Finding Factors Remains Hard



http://xkcd.com/247/

Returning to TSP

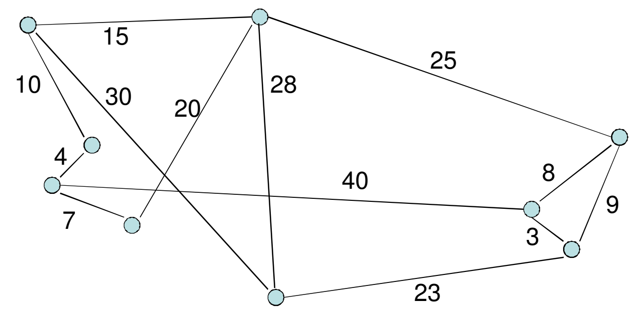
TSP-DECIDE = {<*G*, *cost*> : <*G*> encodes an undirected graph with a positive distance attached to each of its edges and *G* contains a Hamiltonian circuit whose total cost is less than <*cost*>}.



An NDTM to decide TSP-DECIDE:

Returning to TSP

An NDTM to decide TSP-DECIDE:



1. For i = 1 to |V| do:
   Choose a vertex that hasn't yet been chosen.
2. Check that the path defined by the chosen sequence of vertices is a Hamiltonian circuit through G with distance less than cost.

TSP and Other Problems Like It

TSP-DECIDE, and other problems like it, share three properties:

1. The problem can be solved by searching through a space of partial solutions (such as routes). The size of this space grows exponentially with the size of the problem.

2. No better (i.e., not based on search) technique for finding an exact solution is known.

3. But, if a proposed solution were suddenly to appear, it could be checked for correctness very efficiently.