Computational Models - Lecture 10[1]

Handout Mode

Iftach Haitner and Yishay Mansour.

Tel Aviv University.

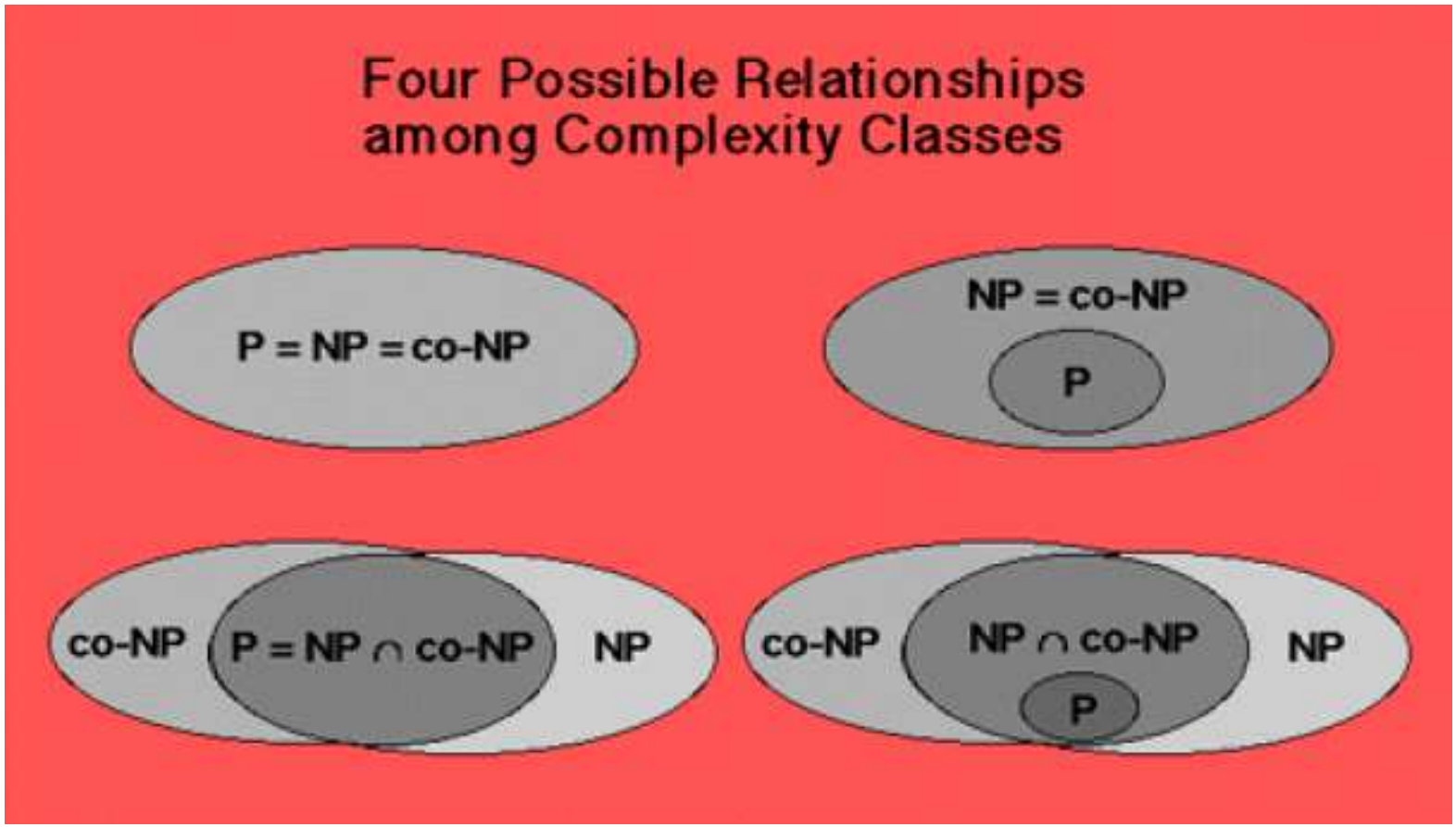
May 28/30, 2012

[1] Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.

Ladies and Gentlemen, Boys and Girls

We are about to begin the fourth part of the course:

Introduction to Computational Complexity



Talk Outline

- Introduction to time complexity
- The class $\mathcal{P}$
- The class $\mathcal{NP}$
- Verifiability

- Sipser's book 7.1-7.3

Part I

Introduction to Time Complexity

How Long It takes to decide $L = \{0^m 1^m : m \geq 0\}$

Clearly $L \in \mathcal{R}$.

Question 1

How much time does a single-tape TM need to decide L?

Algorithm 2 (Decider M_1 for L)

On input string w:

1. Scan across tape and `Reject` if 0 is found to the right of a 1.
2. While both 0s and 1s appear on tape, repeat the following:
   Scan across tape, crossing of a single 0 and a single 1 in each pass.
3. `Accept` if no 0s and 1s remain; otherwise `Reject`.

We analyze the time it takes to perform each of the three steps separately.
In the following let $n = |w|$.

Analyzing Step 1

Scan across tape and `Reject` *if* 0 *is found to the right of a* 1*.*
If not, return to starting point.

- Scanning requires n steps.
- Re-positioning head requires n steps.
- Total is $2n = O(n)$ steps.

Analyzing Step 2

While both 0s and 1s appear on tape, repeat the following

Scan across tape, crossing of a single 0 and a single 1 in each pass.

- Each scan requires $O(n)$ steps.
- Since each scan crosses off two symbols, the number of scans is at most $n/2$.
- Total number of steps is $(n/2) \cdot O(n) = O(n^2)$.

Analyzing Step 3

If 0s still remain after all 1s have been crossed out, or vice-versa, `Reject`*. Otherwise, if the tape is empty,* `Accept`*.*

- Single scan requires $O(n)$ steps.
- Total is $O(n)$ steps.

Total Cost

Total cost for three steps

1. $O(n)$
2. $O(n^2)$
3. $O(n)$

which is $O(n^2)$

Deterministic Time

Definition 3 (deterministic Time)

Let M be a deterministic TM, and let $t : \mathbb{N} \mapsto \mathbb{N}$. We say that M runs in time $t(n)$, if For every input x of length n, the number of steps that $M(x)$ uses is at most $t(n)$.

Question 4

What is a “step"?

Definition 5 (DTIME)

For $t : \mathbb{N} \mapsto \mathbb{N}$, let $\text{DTIME}(t(n)) =$
$\{L \subseteq \Sigma^* :$ L is decided by an $O(t(n))$-time single tape TM$\}$

Note that $t(n)$ running time, is also required for strings not in L.

Relations among Time Classes

Let $t_1, t_2 : \mathbb{N} \mapsto \mathbb{N}$ be two functions.

Claim 6

If $t_1(n) = O(t_2(n))$, then $\text{DTIME}(t_1(n)) \subseteq \text{DTIME}(t_2(n))$.

Stated informally, more time does not hurt. But does it actually help?

Claim 7

If $t_1(n) \in O(t_2(n)/\log(n))$, then $\text{DTIME}(t_1(n)) \subsetneq \text{DTIME}(t_2(n))$.

Informally, sufficiently more time does help

Proofs – sophisticated diagonalizations (omitted).

Back to $L = \{0^m 1^m | m \geq 0\} \in DTIME(n^2)$

We have seen that $L \in DTIME(n^2)$. Can we do faster?

Algorithm 8 (Decider M_2 for L)

On input string w

1. Scan across tape and `Reject` if 0 is found to the right of a 1.
2. Repeat the following while both 0s and 1s appear on tape:
   1. Scan across tape, checking whether total number of 0s plus 1s is even or odd. If odd, `Reject`.
   2. Scan across tape, crossing off every other 0 (starting with the first), and every other 1 (starting with the first) in each pass.
3. `Accept` if all string is crossed off; Otherwise `Reject`.

Example: $w = 0^{13}1^{13}$

Correctness of M_2

Consider parity of 0s and 1s in Step 2.1:

Example 9 ($w = 0^{13}1^{13}$)

- odd, odd (13)
- even, even (6)
- odd, odd (3)
- odd, odd (1)

The parity result, written right to left, is 1101, which is the binary representation of 13.

Each pass checks equality of the next bit in the binary representation of the number of 0's and 1's.
Inequality in any specific bit will be detected (total number of 0s plus 1s will be odd).

Running Time Analysis of M_2

Algorithm 10 (Decider M_2 for L)

On input string w

1. Scan across tape and `Reject` if 0 is found to the right of a 1.
2. Repeat the following while both 0s and 1s appear on tape:
   1. Scan across tape, checking whether total number of 0s plus 1s is even or odd. If odd, `Reject`.
   2. Scan across tape, crossing off every other 0 (starting with the first), and every other 1 (starting with the first) in each pass.
3. `Accept` if all string is crossed off; Otherwise `Reject`.

- One pass in each step (1,2,3) takes $O(n)$ time.
- Steps 1,3: each executed once
- Step 2 executed $1 + \log_2 n$ times
- Total for Step 2 is $(1 + \log_2 n)O(n) = O(n \log n)$.
- Grand total: $O(n) + O(n \log n) = O(n \log n)$.

Further Improvements, Anybody?

Question 11

Can the running time be made $o(n \log n)$?

Answer: Not on a single tape TM...

Definition 12

A crossing sequence at location i is the sequence of states used by the TM when moving from i to $(i+1)$, and from $(i+1)$ to i.

Claim 13

If time $o(n \log n)$ there exists two identical crossing sequences.

Proof: $n/2$ locations have crossing sequence of length $o(\log n)$... ♣

Claim 14

If i and $(i+j)$ have identical crossing sequences, then $x_{i+1} \ldots x_{i+j}$ can be pumped.

A Two-tape Decider for L

Algorithm 15 (Two-tape Decider M_3 for L)

On input string w:

1. Scan across tape and `Reject` if `0` is found to the right of a `1`.
2. Scan across `0`s to first `1`, copying `0`s to tape `2`.
3. Scan across `1`s on tape `1` until the end. For each `1`, cross off a `0`. If no `0`s left, `Reject`.
4. If any `0`s left, `Reject`; otherwise `Accept`.

Question 16

What is M_3 running time?

Complexity of Deciding $L = \{0^n 1^n\}$

- Single-tape M_1: $O(n^2)$.
- single-tape M_2: $O(n \log n)$ (fastest possible!).

  Hence $L \in \text{DTIME}O(n \log n)$, but not in $\text{DTIME}(O(f(n)))$ for $f(n) \in o(n \log n)$
- Two-tape M_3: $O(n)$.

Important difference between complexity and computability:

- Computability: all reasonable models equivalent (Church-Turing)
- Complexity: choice of model does affect running time.

Question 17

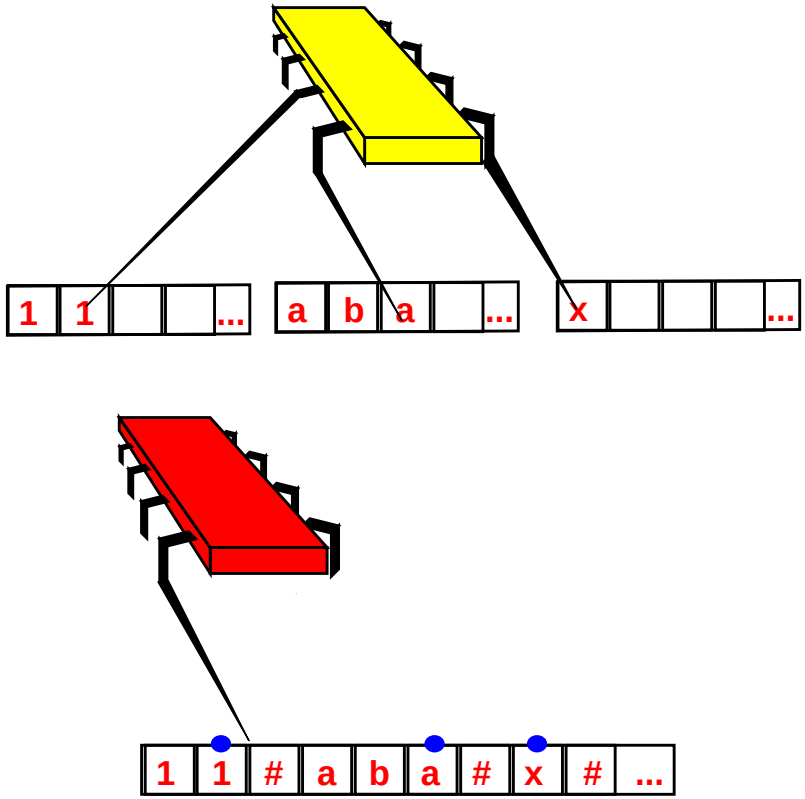
By how much does a model affect complexity?

Multitape Speedup

Let $t(n)$ be a function where $t(n) \geq n$, and let $L \subseteq \Sigma^*$ be a language.

Claim 18

If a $t(n)$-time multitape TM decides L, then $\exists$ an $O(t(n)^2)$-time single-tape TM that decides L.



Reminder: Emulating Multi-Tape TMs

Algorithm 19 (Single tape emulator S for k-tape M)

On input $w = w_1 \cdots w_n$:

1. Write $\# \dot{w}_1 \, w_2 \cdots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \cdots \#$ on tape
2. Scan tape from first $\#$ to $(k+1)$-st $\#$ to read symbols under virtual heads
3. Rescan tape to write new symbols and move heads
4. If need to move virtual head onto $\#$, shift tape content to the right.

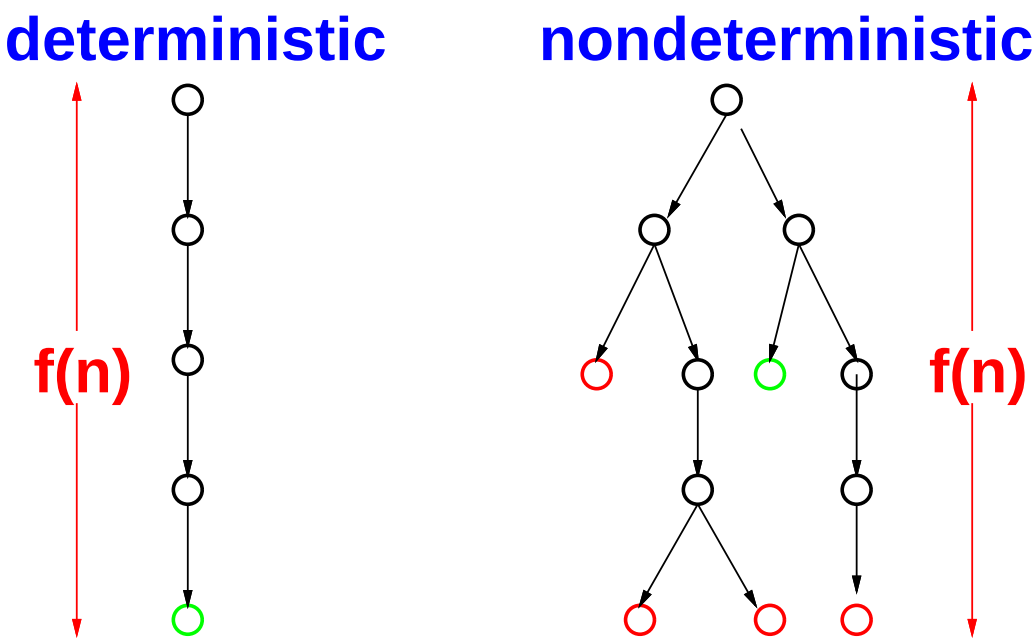
- For each step of M, the emulator S performs 2 scans and up to k rightward shifts
- On input of length n, M makes $O(t(n))$ many steps, so active portion of each tape is $O(t(n))$ long.
- Total number of steps S makes:
  - $O(t(n))$ steps to simulate one step of M.
  - Total simulation $O(t(n)) \times O(t(n)) = O(t(n)^2)$
  - Initial tape arrangement $O(n)$.
  - Grand total: $O(n) + O(t(n)^2) = O(t(n)^2)$ steps (recall $t(n) > n$).

Nondeterministic Time

Definition 20 (nondeterministic Time)

A non-deterministic TM N runs in time $f(n)$, where $f : \mathbb{N} \mapsto \mathbb{N}$, if for every input x of length n, the maximum number of steps that N uses on any branch of its computation tree on x, is at most $f(n)$.

Notice that non-accepting branches must reject within $f(n)$ many steps.



Nondeterminism Speedup

Claim 21

Suppose N is a nondeterministic TM that runs in time $t(n)$ and decides the language L. Then there exists an $2^{O(t(n))}$-time deterministic TM D that decided L.

Note contrast with multi-tape result.

Proof's idea: D emulates N. Reminder

- D tries all possible branches (BFS).
  - `Accept` if finds any accepting state.
  - `Reject` if all branches reject.
- Since N has no looping branches, exactly one of two possibilities must occur.

Emulation Details

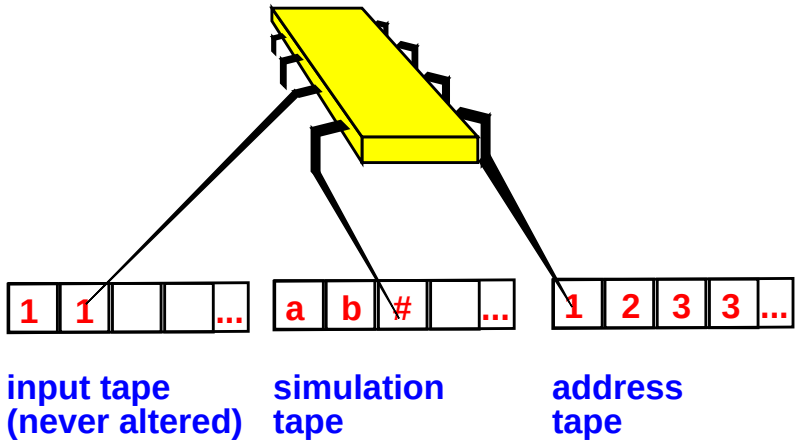
We view the computation of N as a tree:

- Root is starting configuration,
- Fanout is at most some constant b (why?),
- Depth at most $\leq t(n)$,
- Total number of leaves at most $b^{t(n)}$,
- Total number of nodes in tree $O(b^{t(n)})$,
- Time to arrive from root to any node is $O(t(n))$.

  $\Longrightarrow$ Time to visit all nodes is $O(t(n) \times b^{t(n)}) = O(2^{O(t(n))})$

Remarks

1. Breadth-first search used in emulation
   - Visit each node.
   - May be improved upon by using depth-first search (why is it OK?) or other tree search strategies.
   - Still, doing this may save constants, but nothing substantial (why?)
2. Simulation uses three-tape machine.



Single-tape simulation: $(2^{O(t(n))})^2 = 2^{O(2t(n))} = 2^{O(t(n))}$.

Polynomial is Good, Exponential is Bad

	10	20	30	40	50	60
n	.00001 second	.00002 second	.00003 second	.00004 second	.00005 second	.00006 second
n^2	.00001 second	.00004 second	.00009 second	.00016 second	.00025 second	.00036 second
n^3	.00001 second	.00008 second	.027 second	.064 second	.125 second	.216 second
n^5	.1 second	3.2 seconds	24.3 seconds	1.7 minute	5.2 minutes	13.0 minutes
2^n	.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	.059 second	58 minutes	6.5 years	3855 centuries	$2 \cdot 10^8$ centuries	$1.3 \cdot 10^{13}$ centuries

Gaps Between Models

- At most polynomial gap in time to perform tasks between different deterministic models (single- vs. multi-tape TMs, TM vs. RAM, etc.)
- Apparently exponential gap in time to perform tasks between deterministic and non-deterministic models.

Claim 22

All "reasonable" models of computation are polynomially equivalent.

Any one can simulate another with only polynomial blowup in running time.

Fact 23

Is a given problem solvable in

- *Linear time? model-specific.*
- *Polynomial time? model-independent.*

Part II

The Class P

The Class $\mathcal{P}$

$\mathcal{P}$ is the set of languages decidable in polynomial time on deterministic TMs.

Definition 24 ($\mathcal{P}$)

$\mathcal{P} = \bigcup_{c \geq 0} \text{DTIME}(n^c)$

The class $\mathcal{P}$ is important because:

- Invariant for all (deterministic) models of computation polynomially equivalent to TMs
  - not affected by particulars of model . . .
  - go ahead, have another tape, they're pretty small and inexpensive . . .
- Roughly corresponds to realistically solvable (tractable) problems.
  - actually depends on context
  - going from exponential to polynomial algorithm usually requires major insight,
  - if you find an inefficient polynomial algorithm, you can often find a more efficient one.

Known Problems in $\mathcal{P}$

- Integer arithmetic: Addition, subtraction, multiplication, division with remainder.
- Modular arithmetic: Exponentiation (RSA), inverse.
- Integer Algorithms: Greatest common divisor (gcd).
- Operations research: Maximum network flow, linear programming.
- Algebra: Matrix multiplication, computing determinants, matrix inversion, solving systems of linear equations, factoring polynomials.
- Graph algorithms: BFS and DFS in graphs, minimum spanning trees, finding Eulerian path.

Input Encoding

Input encoding may matter match — an exponential algorithm with respect to binary encoding might turn to linear algorithm with respect to the unary encoding

- Integers encoding:
  - binary is good
  - unary is not realistic (exponentially longer)
- Graphs encoding:
  - adjacency list of nodes and edges (good)
  - adjacency matrix (good)

PATH

Given a directed graph G, nodes s and t is there a path from s to t?

$$\text{PATH} = \{\langle G, s, t\rangle | G \text{ has directed path from } s \text{ to } t\}$$

Theorem 25

$\text{PATH} \in \mathcal{P}$.

Algorithm 26 (M_1 – Naive algorithm for PATH)

Input: an m nodes graph G
For each path p in G of length $\leq m$: check if p goes from s to t.

Question 27

What is the (time) complexity of M_1?

- there are m^m possible paths
  $\Longrightarrow$ exponential in number of nodes
  $\Longrightarrow$ exponential in input size
- Oh, oh. Does not sound like $\mathcal{P}$ to me ...

Efficient Algorithm for PATH

Algorithm 28 (M_2 – efficient algorithm for PATH)

1. Place mark on s
2. Repeat until no additional nodes marked:
   1. Scan edges of G.
   2. If edge (a, b) found from marked node a to unmarked node b, mark node b.
3. `Accept` if t is marked; otherwise `Reject`.

Question 29

What is the complexity of M_2?

- Steps 1 and 3 run once.
- Step 2 runs at most m times, because each time (except last) it marks at least one new node.

$\Longrightarrow$ total number of steps is polynomial.

Relative Primality

Two numbers are relatively prime if their *greater common deviser (*gcd*)* is 1 (i.e., the largest integer that divides them both).

- $\gcd(10, 21) = 1 \implies$ 10 and 21 are relatively prime
- $\gcd(10, 22) = 2 \implies$ 10 and 22 are not relatively prime

Definition 30

$\text{RELPRIME} = \{\langle x, y \rangle : \gcd(x, y) = 1\}$.

Theorem 31

$\text{RELPRIME} \in \mathcal{P}$.

Naive algorithm for RELPRIME

Algorithm 32 (Naive algorithm for RELPRIME)

Input: integers x, y:
Search through all possible divisors of x, y and test divisibility.

- If x, y are give in unary:
  - Size of $\langle x \rangle$ is x
  - Testing all potential divisors of x, y is polynomial
- If x, y are give in binary
  - Size of $\langle x \rangle$ is $\log x$
  - Testing all potential divisors of x, y is exponential
- Such algorithm is sometimes called *pseudo polynomial*.

Euclid's Algorithm for Computing gcd

Algorithm 33 (E)

On input $\langle x, y \rangle$:

1. Repeat until $x = y$:
   1. If $y > x$, swap x and y.
   2. $x \leftarrow x \bmod y$
2. Output x

Claim 34

E is polynomial and correctly computes the gcd function.

Algorithm 35 (M for RELPRIME)

On input $\langle x, y \rangle$:
`Accept` iff $E(x, y) = 1$.

To prove that RELPRIME $\in \mathcal{P}$, we only need to prove Claim 34.

Analyzing Euclid's Algorithm

We will only prove the running time part

Algorithm 36 (E)

On input $\langle x, y \rangle$:

1. Repeat until $x = y$:
   1. If $y > x$, swap x and y.
   2. $x \leftarrow x \bmod y$
2. Output x

- Each execution of Step 1 cuts x by at least half (case analysis with for $y < x/2$ and $x/2 \leq y < x$)
- After each two executions maximal value is cut in half

  $\Longrightarrow$ number of stages is $\min(\log_2 x, \log_2 y)$ $\Longrightarrow$ total running time is polynomial.

Consequently, RELPRIME $\in \mathcal{P}$. ♣

Part III

The Class NP

The Class $\mathcal{NP}$

Definition 37 (NTIME)

For $t \mapsto \mathbb{N} \mapsto \mathbb{N}$, let NTIME$(t(n)) =$
$\{L \subseteq \Sigma^* :$ L is decided by an $O(t(n))$-time single tape NTM$\}$

$\mathcal{NP}$ is the set of languages decidable in polynomial time on non-deterministic TMs.

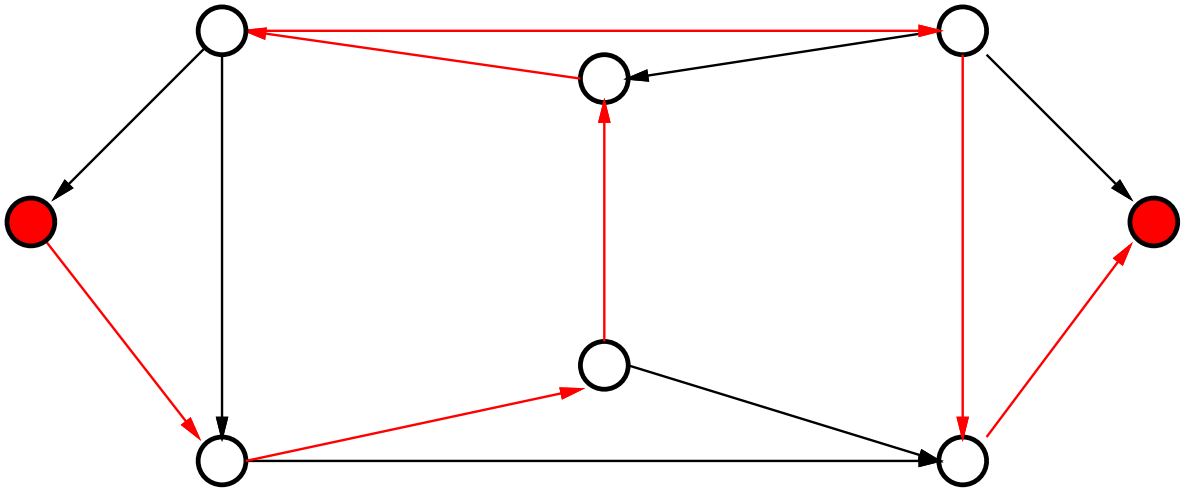
Definition 38 ($\mathcal{NP}$)

$\mathcal{NP} = \bigcup_{c \geq 0}$ NTIME(n^c)

The class *NP* is important because:

- Invariant for all TMs with any number of tapes.
- $\mathcal{NP}$ is insensitive to choice of reasonable non-deterministic computational model.
- Roughly corresponds to problems whose positive solutions are efficiently verified.

Hamiltonian Path



A Hamiltonian path in a directed G visits each node exactly once.

$$\text{HAMPATH} = \{\langle G, s, t\rangle | G \text{ has Hamiltonian path from } s \text{ to } t\}$$

Question 39

How hard is it to decide HAMPATH?

Easy to obtain exponential time algorithm:

- Generate each potential path
- Check whether it is Hamiltonian

HAMPATH $\in \mathcal{NP}$

Here is an NTM that decides HAMPATH in polynomial time.

Algorithm 40 (N)

On input $\langle G = (V, E), s, t \rangle$,

1. Guess a list of numbers $p_1, \ldots, p_m$, where $m = |V|$ and $1 \leq p_i \leq m$.
2. `Accept` if all the following hold (otherwise `Reject`):
   - No repetitions in list
   - $p_1 = s$ and $p_m = t$.
   - $(p_i, p_{i+1}) \in E$ for every $1 \leq i \leq m - 1$

Claim 41

N runs in polynomial time

Verifiability of HAMPATH

This problem has one very interesting feature: polynomial verifiability:

> *We don't know a fast way to find a Hamiltonian path but we can check whether a given path is Hamiltonian in polynomial time.*

In other words Verifying correctness of a path is much easier than determining whether one exists

Composite and Prime Numbers

A natural number is prime if it is divisible by only 1 and itself, otherwise it is composite.

$$\text{Primes} = \{x \in \mathbb{N} : x \text{ is a prime } \}$$

$$\text{COMPOSITES} = \{x \in \mathbb{N} : x = pq \text{ for integers } p, q > 1\}$$

- We can easily verify that a number is composite, given a factor (how?)
- Randomized poly-time algorithm for Primality testing, Rabin-Miller (1976)
- Deterministic poly-time algorithm for Primality testing, Manindra Agrawal, Neeraj Kayal, Nitin Saxena (2002)

Verifiability

Definition 42 (verifier)

A verifier for a language L is an algorithm V with
$L = \{w : V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}$.

- The verifier uses the additional information c to verify $w \in L$.
- We measure verifier run time by length of w.
- If V accepts $\langle w, c \rangle$, the string c is called a certificate (also known as, proof or witness) for w
- A polynomial verifier runs in polynomial time in $|w|$ (so wlg. $|c| \leq |w|^{O(1)}$).
- A language L is polynomially verifiable if it has a polynomial verifier.
- Not all problems are known to be polynomially verifiable
  - There is no known way to verify $\overline{\text{HAMPATH}}$ in polynomial time.
  - In fact, we will see many examples where L is polynomially verifiable, but its complement, $\overline{L}$, is not known to be polynomially verifiable.

Examples

- A certificate for $\langle G, s, t \rangle \in$ HAMPATH is simply the Hamiltonian path from s to t.
  Easy to verify in time polynomial in $|\langle G \rangle|$ whether given path is Hamiltonian.
- A certificate for $x \in$ COMPOSITES is simply one of its divisors.
  Easy to verify in time polynomial in $|x|$ if given divisor indeed divides x.
- A certificate for $p \in$ Primes [Pratt 1975]:
  A number p is prime iff $\exists\, g \leq p-1$ such that:
  - $\gcd(p, g) = 1$.
  - $g^{p-1} \equiv 1 \bmod p$.
  - $g^{(p-1)/q} \not\equiv 1 \bmod p$ for any prime factor q of $p-1$

  The certificate needs to include g and the factorization of $p-1$.

  Size of certificate is $O(\log p)$.

NP and Verifiability

Theorem 43

A language is in $\mathcal{NP}$ iff it has a polynomial time verifier.

Proof's idea:

- The NTM emulates the verifier by guessing the certificate.
- Verifier emulates NTM by using accepting branch as certificate.

Verifiability $\Longrightarrow \mathcal{NP}$

Claim 44

If L has a poly-time verifier, then it is decided by some polynomial-time NTM.

Proof: Let V be poly-time verifier for L (single-tape TM, runs in time n^k)

Algorithm 45 (N)

On input w of length n

1. Guess a string c of length n^k.
2. Emulate V on $\langle w, c \rangle$
3. `Accept` if V accepts; Otherwise `Reject`.

♣

$\mathcal{NP} \Longrightarrow$ Verifiability

Claim 46

If L is decided by a polynomial-time NTM N, then L has a poly-time verifier.

Proof: Construct polynomial-time verifier V as follows.

Algorithm 47 (V)

On input w of length n and certificate c of length n^k

1. Emulate $N(w)$, treating each symbol of c as a description of the non-deterministic choice in each step of N.
2. `Accept` if this branch accepts; Otherwise `Reject`.