Euclid’s Algorithm

on input <x, y>:

• (1) repeat until y = 0
  • (2) set x = x mod y
  • (3) swap x, y
  • x is the GCD(x, y). If x = 1, accept; otherwise reject

Claim: value of x reduced by ½ at every execution of (2) except possibly first one.

Proof:
• after (2) x < y
• after (3) x > y
• if x/2 \geq y, then x mod y < y \leq x/2
• if x/2 < y, then x mod y < y = x/2

loops ≤ 2\max\{\log_2(x), \log_2(y)\} = O(n = |<x, y>|); poly time for each loop

A puzzle

• Find an efficient algorithm to solve the following problem:
  • Input: sequence of pairs of symbols
e.g. (A, b), (E, D), (d, C), (b, a)
  • Goal: determine if it is possible to circle at least one symbol in each pair without circling upper and lower case of same symbol.

2SAT

• This is a disguised version of the language
  2SAT = \{formulas in Conjunctive Normal Form with 2 literals per clause for which there exists a satisfying truth assignment\}
  – CNF = “AND of ORs”
    (A, b), (E, D), (d, C), (b, a)
    (x_1 \lor \neg x_2) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_3) \land (\neg x_2 \lor \neg x_1)
  – satisfying truth assignment = assignment of TRUE/FALSE to each variable so that whole formula is TRUE

Theorem: There is a polynomial-time algorithm deciding 2SAT (“2SAT \in P”).

Proof: algorithm described on next slides.
Algorithm for 2SAT

- Build a graph with separate nodes for each literal.
  - add directed edge \((x, y)\) iff formula includes clause \((\neg x \vee y)\) (equiv. to \(x \Rightarrow y\))

  e.g. \((x \vee \neg x) \land (x \vee y) \land (\neg x \vee y) \land (\neg x \vee \neg y)\)

Claim: formula is unsatisfiable iff there is some variable \(x\) with a path from \(x\) to \(\neg x\) and a path from \(\neg x\) to \(x\) in derived graph.

- \(\Rightarrow\) edges represent implication \(\Rightarrow\). By transitivity of \(\Rightarrow\), a path from \(x\) to \(\neg x\) means \(x \Rightarrow \neg x\), and a path from \(\neg x\) to \(x\) means \(\neg x \Rightarrow x\).

- Proof (\(\Leftarrow\))
  - to construct a satisfying assign. (if no \(x\) with a path from \(x\) to \(\neg x\) and a path from \(\neg x\) to \(x\):)
    - pick unassigned literal \(s\) with no path from \(s\) to \(\neg s\)
    - assign \(s\) TRUE, as well as all nodes reachable from it; assign negations of these literals FALSE
    - note: path from \(s\) to \(t\) and \(s\) to \(\neg t\) implies path from \(\neg t\) to \(\neg s\) and \(t\) to \(\neg s\), implies path from \(s\) to \(\neg s\)
    - note: path \(s\) to \(t\) (assigned \(s\)) implies path from \(\neg t\) (assigned \(s\)) to \(\neg s\), so \(s\) already assigned at that point.

- \(\Rightarrow\) running time \(O(n^2)\). 2SAT \(\in P\).

Another puzzle

- Find an efficient algorithm to solve the following problem.
- Input: sequence of triples of symbols
  e.g. \((A, b, C), (E, D, b), (d, A, C), (c, b, a)\)
- Goal: determine if it is possible to circle at least one symbol in each triple without circling upper and lower case of same symbol.

3SAT

- This is a disguised version of the language 3SAT = \{formulas in Conjunctive Normal Form with 3 literals per clause for which there exists a satisfying truth assignment\}
  e.g. \((A, b, C), (E, D, b), (d, A, C), (c, b, a)\)
  \((x_1 \vee \neg x_2 \vee x_3) \land (x_4 \vee \neg x_5 \vee x_6) \land (x_7 \vee \neg x_8)\)

- observe that this language is in \(\text{TIME}(2^n)\)
Time Complexity

**Key definition:** "P" or "polynomial-time" is
\[ P = \cup_{k \geq 1} \text{TIME}(n^k) \]

**Definition:** "EXP" or "exponential-time" is
\[ \text{EXP} = \cup_{k \geq 1} \text{TIME}(2^n^k) \]

- Note: \( P \subseteq \text{EXP} \).
- We have seen 3SAT \( \in \text{EXP} \).
  - does not rule out possibility that it is in \( P \)
- Is \( P \) different from \( \text{EXP} \)?

EXP

\[ P = \cup_{k \geq 1} \text{TIME}(n^k) \]
\[ \text{EXP} = \cup_{k \geq 1} \text{TIME}(2^n^k) \]

- Note: \( P \subseteq \text{EXP} \).
- We have seen 3SAT \( \in \text{EXP} \).
  - does not rule out possibility that it is in \( P \)
- Is \( P \) different from \( \text{EXP} \)?

Time Hierarchy Theorem

**Theorem:** for every proper complexity function \( f(n) \geq n \):
\[ \text{TIME}(f(n)) \nsubseteq \text{TIME}(f(2n)^3). \]

- Note: \( P \subseteq \text{TIME}(2^n) \nsubseteq \text{TIME}(2^{2^n}) \nsubseteq \text{EXP} \)
- Most natural functions (and \( 2^n \) in particular) are proper complexity functions.
  We will ignore this detail in this class.

Proof idea:
- use diagonalization to construct a language that is not in \( \text{TIME}(f(n)) \).
- constructed language comes with a TM that decides it and runs in time \( f(2n)^3 \).