|  |  |
| --- | --- |
| **McGill University**  **MATH 240 - Discrete Structures** | **Fall 2011**  **Prof Sergey Norin** |

# Computional complexity theory

## circuit complexity

Every logic formula can be represented as a combinational circuit.

Objects: statements, tautologies, contradictions (as letters)

Operators: ¬, ∨, ⊕, …

Size of a circuit: total number of gates

Depth of a circuit: length of the longest path (nb of gates) from input to final output

#### Logic gates

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **“not” Gate** | ¬*p* |  | **“xor” Gate** | *p*⊕*q* |  |
| **“or” Gate** | *p*∨*q* |  | **“and” Gate** | *p*∧*q* |  |

#### Majority/Logical circuit

*p*,*q*,*r*→{1if at least 2 of p,q,r are 1} OR {0 otherwise}

|  |  |
| --- | --- |
| Marjority circuit | Logical circuit |
| (*p*∧*q*)∨(*q*∧*r*)∨(*p*∧*r*) | ((*p*∧*q*)∨(*r*∧(*p*∨*q*))) |
|  |  |
| Size: 5  Depth: 3 | Size: 4  Depth: 3 |

## Algorithm runtime

#### Algorithm A

A step-by-step procedure for solving a problem, precise enough to be carried out on a computer (well defined).

### Runtime

#### Runtime tA (n)

Given algorithm A, its runtime tA(n) = maximum number of steps the algorithm can require on inputs of size n.

**f(n)**: Well-defined function

**n**: Size of the input (n ≥ 0)

**g(n)** : Well-deﬁned function,

#### Big O

tA (n) is big O of g(n)

tA (*n*)=*O*(*g*(*n*))

tA (n) ≤ c g(n) ∀n > n0

for two positive constants n0 and c

A takes max O(g(n)) steps

#### Big Omega

tA (n) is big Omega of g(n)

tA (*n*)= Ω (*g*(*n*))

tA (n) ≥ c g(n) ∀n > n0

for two positive constants n0 and c

A takes min Ω (g(n)) steps

#### Big Theta

tA (n) is big Theta of g(n)

tA (*n*)= θ (*g*(*n*))

tA (*n*)=*O*(*g*(*n*)) and tA (*n*)= Ω (*g*(*n*))

c1 g(n) ≥ tA (n) ≥ c2 g(n) ∀n > n0

for three positive constants n0 c1 and c2

#### Commun growth functions

Where b>a>1:

Fastest --

O(1)

< O(log(n))

< O(n1/2) < O(n)

< O(n log(n))

< O(n2)

< …

< O(an) < O(bn)

< O(n!) < …

-- Slowest

#### Exponential/Polynomial runtimes

Polynomial growth algorithm:

ta(n) is bounded by a polynomial function

*tA*(*n*)=*O*(c*n*) for c>1

Exponential growth algorithm:

ta(n) is bounded by an exponential function

*tA*(*n*)=*O*(*nc*)

Computations taking exponential time are frequently infeasible

vs polynomial time algorithms which are fast and efficient.

## Polytime algorithms: P/NP DECISION problems

### Definitions

Complexity class: Set of problems that can be solved using O(f (n) ) where n is the size of the input.

Decision problem: Question with a yes-or-no answer

P and NP are both complexity classes.

P and NP are both a set of decision problems that can be solved in polynomial time.

* + P : Answer can be found in polynomial time.
  + NP: Answer can be verified in polynomial time.

A decision problem is in the class NP

* + If there is an easy way to see that the answer is YES when it is know that the answer is YES.
  + If “magician” (who knows the answer) can quickly convince you that it is YES.
  + If there exists a set of values for inputs so that the circuit outputs 1 (or T) then given this collection of inputs, verifying that it works fast.

Cobham’s thesis: P means “easy” and “not in P” hard.

### P vs NP

“P = NP”

* + Verifying solution to a problem is as quick as resolving the problem.
  + It returns "yes" in polynomial time when the answer should be "yes", and runs forever when it is "no".
  + “Polynomial-time: algorithm are P=NP.

“P ≠ NP”:

* + A decision problem is in NP but not in P.
  + Verifying solution (y/n) to a decision problem can be done efficiently but not solving it.
  + Suppose that solutions to a problem can be verified quickly. Then, can the solutions themselves also be computed quickly?

Conjecture P ≠ NP

* + Never been resolved

## Examples

Input: Combinatorial circuit C

Decision problem: C not a contradiction? (y/n)

In NP but not in P (P ≠ NP)

Input: Number of n digits

Decision problem: Is this this number composite and if it is, factor it.

In NP but not in P (P ≠ NP)

Travelling Salesman Problem

Input: Collection of n cities and distances between them.

Decision problem: Is there a specific tour: c1-> c2 ->…-> cm, where each city is visited once, with a total length less than 1000 km?

Its long to find the specific tour, but once we know the answer, really answer to check(add up all lengths).

In NP but not in P (P ≠ NP)

Airline scheduling

Protein folding

Packing boxes

Proving short problems

Linear programming

Primarily testing

In P and NP (P=NP)

## GCD

#### Computing GCD

Input: integers a & b (in binary)

Output: GCD(a, b)

Steps

1. a ≥ b

2. Divide with remainder 0 ≤ r < b : a = qb + r

3. If (r = 0 ) then output == b

4. Otherwise, run GCD (b,r).

#### Analysis

Using the theorem: a = qb + r -> gcd(a,b) = gcd(b,r)

This algorithm is O ((log2 a + log2 b)k)

\*Proof

#### Euclid’s algorithm

Takes at most log2a+log2b steps.