Objectives

- P, NP, NP-Hard and NP-Complete
- Solving 3-CNF Sat problem
- Discussion of Gate Questions

Types of Problems

- Trackable
- Intrackable
- Decision
- Optimization

Trackable : Problems that can be solvable in a reasonable(polynomial) time.

Intrackable : Some problems are *intractable*, as they grow large, we are unable to solve them in reasonable time.

Tractability

- What constitutes reasonable time?
  - Standard working definition: *polynomial time*
  - On an input of size n the worst-case running time is $O(n^k)$ for some constant k
  - $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$, $O(2^n)$, $O(n^n)$, $O(n!)$
  - Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
  - Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$
- Are all problems solvable in polynomial time?
  - No: Turing's "Halting Problem" is not solvable by any computer, no matter how much time is given.

Optimization/Decision Problems

- Optimization Problems
  - An optimization problem is one which asks, “What is the optimal solution to problem X?”
  - Examples:
    - 0-1 Knapsack
    - Fractional Knapsack
    - Minimum Spanning Tree
- Decision Problems
  - An decision problem is one with yes/no answer
  - Examples:
    - Does a graph G have a MST of weight $\leq$ W?

Optimization/Decision Problems

- An optimization problem tries to find an optimal solution
- A decision problem tries to answer a yes/no question
- Many problems will have decision and optimization versions
  - Eg: Traveling salesman problem
    - optimization: find hamiltonian cycle of minimum weight
    - decision: is there a hamiltonian cycle of weight $\leq k$

P, NP, NP-Hard, NP-Complete

-Definitions

The Class *P*

<u>P</u>: the class of problems that have polynomial-time deterministic algorithms.

- That is, they are solvable in $O(p(n))$, where $p(n)$ is a polynomial on n
- A deterministic algorithm is (essentially) one that always computes the correct answer

Sample Problems in P

- Fractional Knapsack
- MST
- Sorting
- Others?

The class *NP*

NP: the class of decision problems that are solvable in polynomial time on a *nondeterministic* machine (or with a nondeterministic algorithm)

  – (A *determinstic* computer is what we know)
  – A *nondeterministic* computer is one that can "guess" the right answer or solution

- Think of a nondeterministic computer as a parallel machine that can freely spawn ***an infinite number*** of processes
- Thus *NP* can also be thought of as the class of problems "whose solutions can be verified in polynomial time"
- Note that *NP* stands for "Nondeterministic Polynomial-time"

Sample Problems in NP

- Fractional Knapsack
- MST
- Others?
  - Traveling Salesman
  - Graph Coloring
  - Satisfiability (SAT)
    - the problem of deciding whether a given Boolean formula is satisfiable

P And NP Summary

- **P** = set of problems that can be solved in polynomial time
  - Examples: Fractional Knapsack, ...
- **NP** = set of problems for which a solution can be verified in polynomial time
  - Examples: Fractional Knapsack,..., TSP, CNF SAT, 3-CNF SAT
- Clearly **P** $\subseteq$ **NP**
- Open question: Does **P = NP**?
  - $P \neq NP$

NP-hard

- What does NP-hard mean?
  - A lot of times you can solve a problem by reducing it to a different problem. I can reduce Problem B to Problem A if, given a solution to Problem A, I can easily construct a solution to Problem B. (In this case, "easily" means "in polynomial time.“).
- A problem is **NP-hard** if all problems in NP are polynomial time reducible to it, ...
- Ex:- Hamiltonian Cycle

Every problem in NP is reducible to HC in polynomial time. Ex:- TSP is reducible to HC.

B A

Example: lcm(m, n) = m * n / gcd(m, n),

NP-complete problems

- A problem is **NP-complete** if the problem is both
  - NP-hard, and
  - NP.

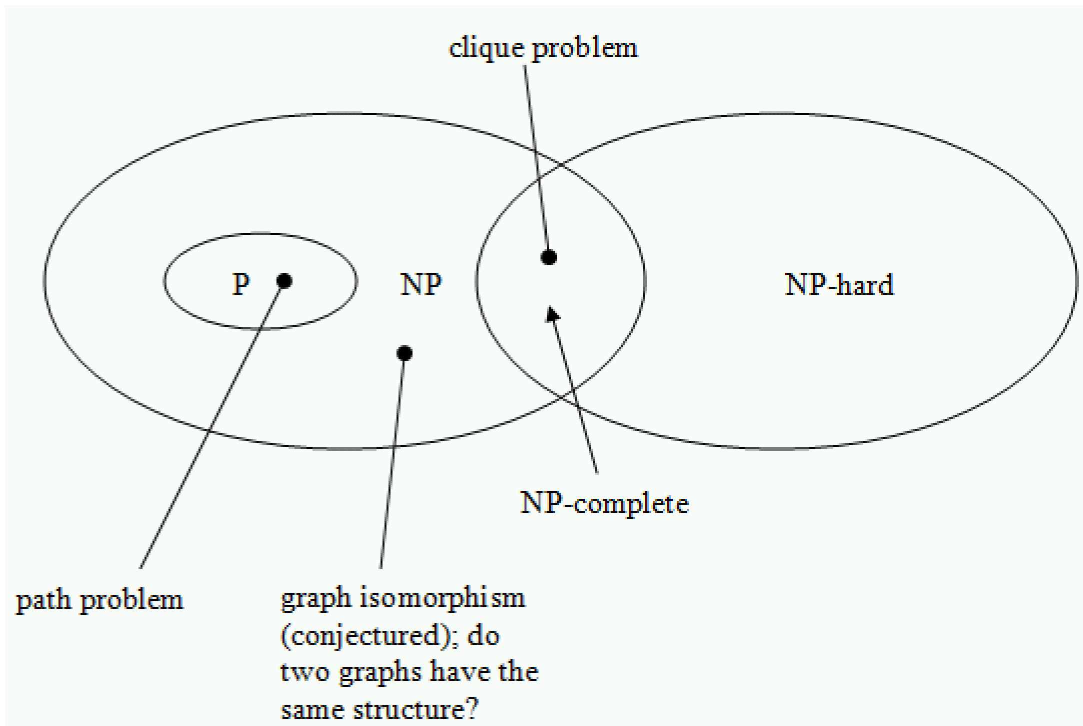
Reduction

- A problem R can be *reduced* to another problem Q if any instance of R can be rephrased to an instance of Q, the solution to which provides a solution to the instance of R
  - This rephrasing is called a *transformation*
- Intuitively: If R reduces in polynomial time to Q, R is “no harder to solve” than Q
- Example: lcm(m, n) = m * n / gcd(m, n),

  lcm(m,n) problem is reduced to gcd(m, n) problem

NP-Hard and NP-Complete

- If R is *polynomial-time reducible* to Q, we denote this $R \leq_p Q$
- Definition of NP-Hard and NP-Complete:
  - If all problems $R \in$ **NP** are *polynomial-time* reducible to Q, then Q is *NP-Hard*
  - We say Q is *NP-Complete* if Q is NP-Hard and $Q \in$ **NP**
- If $R \leq_p Q$ and R is NP-Hard, Q is also NP-Hard


clique problem
P
NP
NP-hard
NP-complete
path problem
graph isomorphism
(conjectured); do
two graphs have the
same structure?

Summary

- P is set of problems that can be solved by a deterministic Turing machine in **P**olynomial time.
- NP is set of problems that can be solved by a **N**on-deterministic Turing Machine in **P**olynomial time. P is subset of NP (any problem that can be solved by deterministic machine in polynomial time can also be solved by non-deterministic machine in polynomial time) but P≠NP.

- Some problems can be translated into one another in such a way that a fast solution to one problem would automatically give us a fast solution to the other.
- There are some problems that every single problem in NP can be translated into, and a fast solution to such a problem would automatically give us a fast solution to every problem in NP. This group of problems are known as **NP-Complete**. Ex:- Clique
- A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem (nondeterministic polynomial time) problem. NP-hard therefore means "at least as hard as any NP-problem," although it might, in fact, be harder.

First NP-complete problem—Circuit Satisfiability (problem definition)

- Boolean combinational circuit
  - Boolean combinational elements, wired together
  - Each element, inputs and outputs (binary)
  - Limit the number of outputs to 1.
  - Called *logic gates*: NOT gate, AND gate, OR gate.
  - *true table*: giving the outputs for each setting of inputs
  - true assignment: a set of boolean inputs.
  - satisfying assignment: a true assignment causing the output to be 1.

Circuit Satisfiability Problem: definition

- Circuit satisfying problem: given a boolean combinational circuit composed of AND, OR, and NOT, is it stisfiable?
- CIRCUIT-SAT={<C>: C is a satisfiable boolean circuit}
- Implication: in the area of computer-aided hardware optimization, if a subcircuit always produces 0, then the subcircuit can be replaced by a simpler subcircuit that omits all gates and just output a 0.