Introduction to Graph Theory

Lecturer: Derek G. Corneil, dgc@cs.toronto.edu, SF3303B
Lecture time: MW 11-12, R 9-10

Tutor: Lap Chi Lau, chi@cs.toronto.edu, SF3213
Tutorial: occasionally on selected lecture time.

Outline of this tutorial

- Algorithm and Turing machine
- Asymptotic notation
- P, NP, NP-complete, co-NP
- Some examples of how to show a problem is NP-complete

Algorithm

Informally, you can view an algorithm as a tool for solving a well-specified computational problem. The statement of the problem specifies in general terms the desired input/output relation. The algorithm is a step-to-step procedure for achieving the input/output relation.

We are usually happy about the informal definition of algorithms. However, when we want to start arguing that algorithm (for a particular problem) *doesn't* exist, we have to be completely rigorous about what we mean by an algorithm.

Turing Machine

In 1936, Alan Turing gave the first definition of an algorithm in the form of a Turing Machine. It is a definition that is simple, clear and general.

A Turing Machine consists of a two-way infinite tape and a finite state control. The tape is divided into cells. Each cell holds a symbol from a finite tape alphabet including the blank symbol.

Turing Machine

The machine has a read/write head that is connected to the control, and that scans the symbol on the tape. Depending on the state of the control and the symbol scanned, it makes a move consisting of:

- printing a symbol
- moving the head left or right one cell
- change to a new state

We will consider decision problems such that the answer is either “yes” or “no”.

Power of Turing Machine

Church Turing Thesis
Anything that can be intuitively deserves to be called an "algorithm" can be simulated by a Turing Machine. So any function can be computed by some kind of "algorithms" is actually computable in our sense.

It is widely accepted, except some said that there is something about the Human brain that fundamentally cannot be simulated by Turing machine.

Decidability

The collection of strings that M accepts is *the language of M* denoted by $\mathcal{L}(M)$. We say a language L is *decidable* if there is a Turing Machine M such that $L = \mathcal{L}(M)$ and M halts on every input.

Halting problem
Given a Turing Machine M and an input string w, does M accepts w?

Theorem: Halting problem is undecidable.

Its not only of theoretical interest, we can also show by reduction that practical problem like if there is an algorithm to check if two programs have the same behavior is undecidable.

Analysis of algorithms - O notation

Now consider decidable problems. We want the algorithm to be efficient, but how we define efficiency and how we analyze an algorithm?

Usually the time taken to solve a problem grows as the input size grows. So you can expect the running time of an algorithm is a function $f(n)$ of the input size n.

O **notation**: $O(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$.

For example, insertion sort is $O(n^2)$, merge sort is $O(n \log n)$.

Analysis of algorithms - O notation

Justification: when n is large, the running time is dominated by n. Weird cases are very rare.

$O(n) \subset O(nlogn) \subset O(n^2) \subset O(n^5) \subset O(n^{logn}) \subset O(2^n) \subset O(n!) \subset O(n^n)$.

Ω-notation: $\Omega(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0\}$.

Θ-notation: $\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$.

The complexity class P

It is the set of decision problems that can be solved by a deterministic Turing Machine in polynomial time. (i.e. $O(n^c)$ for some constant c).

Polytime Thesis: If a language can be accepted by some sort of polynomial time algorithm, according to some reasonable notion of polynomial time algorithm, then that language can be computed by a polynomial time Turing Machine.

Potential counterexample: quantum computer.

The class of efficient algorithm = P?

Justification: robust, machine-independent?, polynomial is closed under multiplication, addition.

Non-deterministic Turing Machine

Instead of having a transition function deterministically indicate what the next move of the Turing Machine is based on the current state and tape symbol, we will have the transition function present our Turing Machine with a finite set of possible moves and we will assume the machine has the ability to always choose the best move from the choices.

Non-deterministic Turing Machine

You can draw a search tree based on the decision. A non-deterministic Turing Machine accept an input if and only if there is a path leads to the accepting state.

NP is the class of decision problems that can be solved by a non-deterministic Turing Machine in polynomial time.

Equivalent definition: NP is the class of decision problems that have a Polytime "verification" algorithm.

Open problem: P = NP?

Polynomial time reduction

Given two decision problems A and B. We say A reduces in polynomial time to B ($A \leq_p B$) if:

- There is a *polynomial time* transformation algorithm P such that for any instance α of A, P transform α of A into β of B.
- The answer for α is "yes" if and only if the answer for β is "yes".

We call P a polynomial time reduction algorithm. Intuitively, we show that B is at least not easier than A (if B is in P, then A is also in P).

NP-complete

We say L is NP-complete if:

- L is in NP.

- For all $L' \in NP$, $L' \leq_p L$.

SAT Given a propositional formula F, is F satisfiable?

We can assume the formula is given in CNF (conjunctive normal form).

Cook's Theorem

Cook's theorem: SAT is NP-complete.

So it explains why problem is difficult to solve. because it would tell us how to simulate a non-deterministic Turing machine by a deterministic Turing machine which no one knows.

To show a problem is NP-complete, do a Polytime reduction from a NP-complete problem. One NP-complete problem is in P, then all NP-complete problems are in P.

Examples of NP-complete problems

Hamiltonian path
INSTANCE: Graph $G = (V, E)$.
QUESTION: Does G contains a Hamiltonian path?
(a simple path that passes every vertex exactly once)

Clique
INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |V|$.
QUESTION: Does G contain a clique of size K or more, i.e., a subset $V' \subseteq V$ with $|V'| \geq K$ such that every vertices in V' are joined by an edge in E?

Example of polynomial reductions

Isomorphic spanning tree
INSTANCE: Graph $G = (V, E)$, tree $T = (V_T, E_T)$.
QUESTION: Does G contain a spanning tree isomorphic to T?
Transformation from Hamiltonian path.

Vertex Cover
INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |V|$.
QUESTION: Is there a vertex cover of size K or less for G, i.e., a subset $V' \subseteq V$ with $|V'| \leq K$ such that for every edge $\{u, v\} \in E$ at least one of u and v belongs to V'?
Transformation from clique.

$co-$**NP**

The class $co-$NP is the class of all decision problems that are complements of problems in NP.

open problem: NP $=$ $co-$NP?

How to show a graph does not have a Hamiltonian path?

conjecture: NP $\neq$ $co-$NP

conjecture: if NP and $co-$NP, then P

example: prime number is in P.

Graph isomorphism is unknown to be in P and unknown to be NP-c.

References

For more information on Turing Machine, computability:
Introduction to the theory of computation - Michael Sisper

For more information on design and analysis of algorithms:
Introduction to algorithms - MIT press

For more information on NP-completeness:
Computers and intractability - A guide to the theory of NP-completeness
Michael R. Garey, David S. Johnson