طراحی الگوریتم ها (CE221)

جلسه بیستم: پیچیدگی محاسبات

سجاد شیرعلی شهرضا

بهار 1401

دوشنبه، 9 خرداد 1401

Based on Rose-Hulman MA/CSSE 473 Slides from Summer 2017 by Claude Anderson

اطلاع رسانی

- نظرسنجی چهارم
  - مهلت ارسال: دوشنبه هفته آینده، 16 خرداد 1401، ساعت 8 صبح

زمان حل چند جمله ای

Idea

- Well-known algorithm rule: Polynomial good, exponential bad!
  - The latter is obvious, the former may need some explanation
  - We say that polynomial-time problems are **tractable**
    - I.e., exponential problems are **intractable**

Polynomial Time Benefits

- It's not exponential!
  - We **cannot** say that every polynomial time algorithm has an acceptable running time
  - But, if it **does not** run in polynomial time, it **only** works for small inputs
- Polynomial time is closed under standard operations.
  - If $f(t)$ and $g(t)$ are polynomials, so is $f(g(t))$
  - Also closed under sum, difference, product
- Almost all of the algorithms we have studied in this course had polynomial time

Decision Problems

- Decision problem: a question that has two possible answers, **yes** and **no**
- Almost any problem can be rephrased as a decision problem
- The question is about some input
- A problem instance is a combination of the problem and a specific input
- The statement of a decision problem has two parts
  - **Instance description**: defines the information expected in the input
  - **Question:** states the specific yes-or-no question
    - Refers to variables that are defined in the instance description
- Almost every optimization problem can be expressed in decision problem form

Decision Problem Example 1

- **Definition**:
  - In a graph G=(V,E), a **clique** E is a subset of V such that for all u and v in E, the edge (u,v) is in E.
- **Clique Decision problem**
  - **Instance**: an undirected graph G=(V,E) and an integer k.
  - **Question**: Does G contain a clique of k vertices?
- **k-Clique Decision problem**
  - **Instance**: an undirected graph G=(V,E).
    - Note: k is some constant, independent of the problem
  - **Question**: Does G contain a clique of k vertices?

Decision Problem Example 2

- **Definition**:
  - The **chromatic number** of a graph $G=(V,E)$ is the smallest number of colors needed to color G such that no two adjacent vertices have the same color
- **Graph Coloring Optimization Problem**
  - **Instance**: an undirected graph $G=(V,E)$.
  - **Problem**: Find G's chromatic number and a coloring that realizes it
- **Graph Coloring Decision Problem**
  - **Instance**: an undirected graph $G=(V,E)$ and an integer $k>0$.
  - **Question**: Is there a coloring of G that uses no more than k colors?

Decision Problem Example 3

- **Definition**:
  - Suppose we have an unlimited number of bins, each with capacity 1.0, and n objects with sizes $s_1, \ldots, s_n$, where $0 < s_i \leq 1$ (all s_i rational)
- **Bin Packing Optimization Problem**
  - **Instance**: $s_1, \ldots, s_n$ as described above.
  - **Problem**: Find the smallest number of bins into which the n objects can be packed
- **Bin Packing Decision Problem**
  - **Instance**: $s_1, \ldots, s_n$ as described above, and an integer k.
  - **Question**: Can the n objects be packed into k bins?

سوال؟

تقليل

Reduction

- Assumptions:
  - Goal: solve problem p
  - There is another problem q
  - Have a function T that
    - Takes an input x for p
    - Produces T(x) that is
      - An input for q
      - Correct answer for p with input x is yes **if and only if** the correct answer for q with input T(X) is yes
- We say: p is **reducible** to q and we write **p≤q**

Polynomial Reduction

- If there is an algorithm for q, create an algorithm for p:
  - Compose T with that algorithm
  - Get q's answer to get an algorithm for T
  - Use that to generate answer for p
- If T is a function with polynomially bounded running time:
  - We say: p is **polynomially reducible** to q
  - We write: $\mathbf{p \leq_p q}$
- For now, reducible means polynomially reducible

Class P

- **Definition**: An algorithm is polynomially bounded if its worst-case complexity is big-O of a polynomial function of the input size n.
    - I.e. there is a single polynomial $\boldsymbol{p}$ such that for each input of size $\boldsymbol{n}$, the algorithm terminates after at most $\boldsymbol{p(n)}$ steps
    - Input size: the number of bits to represent the problem instance's input
- **Definition**: A problem is **polynomially bounded** if there is a polynomially bounded algorithm that solves it
- **The class P**
    - Class P: the class of decision problems that are polynomially bounded
    - Informally (with slight abuse of notation), we can say that polynomially bounded optimization problems are in P

Example of a problem in P

- Minimum Spanning Tree (MST)
- **Input**: A weighted graph G=(V,E) with n vertices [each edge e is labeled with a non-negative weight w(e)], and a number k.
- **Question**: Is the total weight of a minimal spanning tree for G less than k?
- How do we know it's in P?
  - Find the MST and check whether its cost is less than k

Another Example: Clique Problems

- It is known that we can determine whether a graph with n vertices has a k-clique in time $O(k^2 n^k)$
- **Clique Decision problem 1**
  - **Instance**: an undirected graph $G=(V,E)$ and an integer k.
  - **Question**: Does G contain a clique of k vertices?
- **Clique Decision problem 2**
  - **Instance**: an undirected graph $G=(V,E)$. Note that k is some constant, independent of the problem.
  - **Question**: Does G contain a clique of k vertices?
- Are either of these decision problems in P?
  - No. The size to represent k is log k

Class NP

- NP: Nondeterministic Polynomial time
- First stage: assumes a "guess" of a possible solution
- Can we verify whether the proposed solution really is a solution in polynomial time?
    - I.e., do we have a verifier that verifies an answer in a polynomial time?

Example of a Class NP Problem

- Example: Graph coloring
  - Given a graph G with N vertices, can it be colored with k colors?
- A solution: an actual k-coloring.
- A "proposed solution": something that is in the right form for a solution.
  - E.g., a coloring that
    - May or may not have only k colors
    - May or may not have distinct colors for adjacent nodes
- The problem is in NP **if and only if** there is a polynomial-time (in N) algorithm that can check a proposed solution to see if it really is a solution

Another Definition of NP

- Nondeterministic algorithm phases:
  - The nondeterministic "**guessing**" phase: produce the proposed solution
  - The deterministic **verifying** phase: check the proposed solution to see if it is indeed a solution
    - Output yes if it is a solution
  - Output "yes" or "no"
    - Output no if could not find any solution
- NP Class: class of decision problems for which there is a polynomially bounded nondeterministic algorithm
  - Examples: Graph coloring, Bin packing, Clique

سوال؟

رابطه بین P و NP

Problem Class Containment

- Exp class: set of all decision problems that can be solved by a deterministic exponential-time algorithm.
- Then P $\subseteq$ NP $\subseteq$ Exp
- P $\subseteq$ NP
    - Directly solve the problem to find the answer
        - No guessing needed
- NP $\subseteq$ Exp
    - Generate all possible solutions
        - Will be exponential in terms of the input
    - Check them to see if one of them is the answer
    - Will be deterministic

Relation between P and NP

- Is $\boldsymbol{P} = \boldsymbol{NP}$?
- Or is it $\boldsymbol{P} \neq \boldsymbol{NP}$
    - This seems to be the case:
        - We have a large group of problems in NP
        - All reducible to each other
        - No one can find a Polynomial-time algorithm
- Try to solve it as an extra assignment
    - Will give you extra mark for course if you do it!

Class of NP-Hard and NP-Complete

- NP-Hard: A problem that all NP problems can be reduced to it in polynomia time
- Class of NP – Complete (NP – C): the set of all problems in NP "at least as hard" as every other problem in NP.
- To prove a problem ***x*** is NP complete
  - Show that ***x*** is in NP
  - Show that some other NP - C problem reduces to ***x***
- If an NP hard problem can be solved in polynomial time, then all NP complete problems can be solved in polynomial time.
- NP – H includes all NP - C problems

NP-Complete Proof

- A problem is NP-hard if every problem in NP is reducible to it
- A problem is NP-complete if it is in NP and is NP-hard
- Showing that a problem is NP complete is difficult.
  - Has only been done directly for a few problems.
  - Example: 3-satisfiability
- If $\boldsymbol{p}$ is NP-hard, and $\boldsymbol{p} \leq_p \boldsymbol{q}$, then **q** is NP-hard.
  - Most NP-complete problems are shown to be NP-C by showing that 3-satisfiability (or some other known NP-complete problem) reduces to them

3-SAT

- 3-Satisfiability problem:
- A CNF (conjunctive normal form) formula is in 3-CNF if every clause has exactly three literals
- **Instance**: A 3CNF propositional formula ***f*** (containing **n** different variables)
- **Question**: Is there a truth assignment that satisfies ***f***?

سوال؟