Lecture 19: Complexity Intro & Reductions

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

November 16, 2023

Overview

- Complexity Classes
  - Decision Problems
  - P - decision problems with efficient algorithms
  - Search/Optimization Problems
  - Reductions & Transformations
- Examples of Problems & Transformations
  - Problems
  - Transformations
- Acknowledgements

Decision Problems

- *Decision problems* are problems which have a YES/NO answer
  - Given graph G, does it have perfect matching?
  - Given graph G and $k \in \mathbb{N}$, does it have matching of size k?
  - Given directed graph G and $s, t \in V$, is there an $s \to t$ path in G?
  - Given directed graph G, $s, t \in V$ and $k \in \mathbb{N}$, are there k edge-disjoint $s \to t$ paths in G?

- Complexity Classes
  - Decision Problems
  - P - decision problems with efficient algorithms
  - Search/Optimization Problems
  - Reductions & Transformations
- Examples of Problems & Transformations
  - Problems
  - Transformations
- Acknowledgements

Complexity Class P: the class of efficient algorithms

- We have learned many "efficient" algorithms in this course so far
- But what do we really mean when we say "efficient"?

Complexity Class P: the class of efficient algorithms

- We have learned many "efficient" algorithms in this course so far
- But what do we really mean when we say "efficient"?
- A good notion of efficient should be:
  - "*fast*" (solve large instances of the problem in reasonable time)

    As instances grow, runtime should not be prohibitive.
  - "*composable*" (allows for using efficient subroutines)

    Of course, should call subroutine *not too many times*

Complexity Class P: the class of efficient algorithms

- We have learned many "efficient" algorithms in this course so far
- But what do we really mean when we say "efficient"?
- A good notion of efficient should be:
  - "*fast*" (solve large instances of the problem in reasonable time)

    As instances grow, runtime should not be prohibitive.
  - "*composable*" (allows for using efficient subroutines)

    Of course, should call subroutine *not too many times*
- What could be a set of runtimes that satisfy the above properties?

Polynomial time!

A runtime $T(n)$ is polynomial if there is a constant $c > 0$ such that $T(n) = O(n^c)$.

Complexity Class P: the class of efficient algorithms

- We have learned many "efficient" algorithms in this course so far
- But what do we really mean when we say "efficient"?
- A good notion of efficient should be:
  - "*fast*" (solve large instances of the problem in reasonable time)

    As instances grow, runtime should not be prohibitive.
  - "*composable*" (allows for using efficient subroutines)

    Of course, should call subroutine *not too many times*
- What could be a set of runtimes that satisfy the above properties?

  Polynomial time!

  A runtime $T(n)$ is polynomial if there is a constant $c > 0$ such that $T(n) = O(n^c)$.
- Complexity class P:

  P := class of decision problems with *algorithms* which correctly decide them in polynomial time.

- Complexity Classes
  - Decision Problems
  - P - decision problems with efficient algorithms
  - Search/Optimization Problems
  - Reductions & Transformations
- Examples of Problems & Transformations
  - Problems
  - Transformations
- Acknowledgements

Search/Optimization Problems

- The choice of considering decision problems is not too restrictive

Search/Optimization Problems

- The choice of considering decision problems is not too restrictive
- Often times, deciding property allows us to search for witness of the property
  - saw this in DP
  - greedy always returns a solution with maximal properties
  - BFS/DFS, Dijkstra trees
  - saw how to get the max-flow & the min-cut
  - can find a perfect matching if we can decide whether graph has perfect matching

Search/Optimization Problems

- The choice of considering decision problems is not too restrictive
- Often times, deciding property allows us to search for witness of the property
  - saw this in DP
  - greedy always returns a solution with maximal properties
  - BFS/DFS, Dijkstra trees
  - saw how to get the max-flow & the min-cut
  - can find a perfect matching if we can decide whether graph has perfect matching
- Also, for optimization problems, it is often the case that the decision version of problem combined with binary search yields an efficient solution
  - If we can decide, for every k, whether graph G has matching of size k, we can find the maximum matching in G
  - If we can decide, for every k, whether G has a flow of value k, then we can find the max-flow value.

- Complexity Classes
  - Decision Problems
  - P - decision problems with efficient algorithms
  - Search/Optimization Problems
  - Reductions & Transformations
- Examples of Problems & Transformations
  - Problems
  - Transformations
- Acknowledgements

Reductions & Transformations

- How do we prove problem A is "easier than" another problem B?

Reductions & Transformations

- How do we prove problem A is "easier than" another problem B?
- Intuitive notion is: if we can efficiently solve B, then we can also efficiently solve A

Reductions & Transformations

- How do we prove problem A is "easier than" another problem B?
- Intuitive notion is: if we can efficiently solve B, then we can also efficiently solve A
- Now that we have our notion of efficient (i.e., polynomial time solvable), we can make the notion above precise.

Reductions & Transformations

- How do we prove problem A is "easier than" another problem B?
- Intuitive notion is: if we can efficiently solve B, then we can also efficiently solve A
- Now that we have our notion of efficient (i.e., polynomial time solvable), we can make the notion above precise.
- There are a couple of ways to go about it.
  - Turing Reductions
  - Karp Reductions (or Polynomial Transformations)
  - Truth Table Reductions (won't see this in CS 341...)

Polynomial Time (Turing) Reductions

- Turing reductions are the most natural way you would think about reducing a problem.
  We say that
  $$A \leq_T B$$
  if there is a polynomial-time algorithm M which solves problem A and makes *polynomially* many calls[1] to instances of B

[1] These calls to instances of B does not count towards the running time of M. Think of M as having access to an "oracle" that gives correct answers to instances of B (in unit time).

Polynomial Time (Turing) Reductions

- Turing reductions are the most natural way you would think about reducing a problem.
  We say that
  $$A \leq_T B$$
  if there is a polynomial-time algorithm M which solves problem A and makes *polynomially* many calls[1] to instances of B
- Intuitively, M can use any (efficient) algorithm for B as a subroutine, so long as it uses it polynomially many times.

[1] These calls to instances of B does not count towards the running time of M. Think of M as having access to an "oracle" that gives correct answers to instances of B (in unit time).

Polynomial Time (Turing) Reductions

- Turing reductions are the most natural way you would think about reducing a problem.
  We say that

  $$A \leq_T B$$

  if there is a polynomial-time algorithm M which solves problem A and makes *polynomially* many calls[1] to instances of B
- Intuitively, M can use any (efficient) algorithm for B as a subroutine, so long as it uses it polynomially many times.
- For instance, we saw in previous lectures that
  - max-flow $\leq_T$ shortest paths

[1] These calls to instances of B does not count towards the running time of M. Think of M as having access to an "oracle" that gives correct answers to instances of B (in unit time).

Polynomial Time Transformations (Karp reductions)

- Another type of reduction is a *mapping* reduction, also known as *transformations*

Polynomial Time Transformations (Karp reductions)

- Another type of reduction is a *mapping* reduction, also known as *transformations*
- In this case, if we can – in polynomial time – transform each YES instance of problem A to a YES instance of problem B, and each NO instance of problem A to a NO instance of problem B, then we say

$$A \leq_m B$$

Polynomial Time Transformations (Karp reductions)

- Another type of reduction is a *mapping* reduction, also known as *transformations*
- In this case, if we can – in polynomial time – transform each YES instance of problem A to a YES instance of problem B, and each NO instance of problem A to a NO instance of problem B, then we say

$$A \leq_m B$$

- If we can do the above, then any efficient algorithm for B will yield an efficient algorithm for A

Polynomial Time Transformations (Karp reductions)

- Another type of reduction is a *mapping* reduction, also known as *transformations*
- In this case, if we can – in polynomial time – transform each YES instance of problem A to a YES instance of problem B, and each NO instance of problem A to a NO instance of problem B, then we say

$$A \leq_m B$$

- If we can do the above, then any efficient algorithm for B will yield an efficient algorithm for A
- For instance, we saw in previous lectures that
  - Perfect matching in bipartite graphs $\leq_m$ max-flow
  - vertex-cover in bipartite graphs $\leq_m$ max-flow

- Complexity Classes
  - Decision Problems
  - P - decision problems with efficient algorithms
  - Search/Optimization Problems
  - Reductions & Transformations

- Examples of Problems & Transformations
  - Problems
  - Transformations

- Acknowledgements

Problems

- Clique:
  - **Input:** graph $G(V, E)$, integer $k \in \mathbb{N}$
  - **Output:** is there a clique in G with k vertices?

Problems

- Clique:
  - **Input:** graph $G(V, E)$, integer $k \in \mathbb{N}$
  - **Output:** is there a clique in G with k vertices?
- Independent Set:
  - **Input:** graph $G(V, E)$, integer $k \in \mathbb{N}$
  - **Output:** is there an independent set in G of size k?

Problems

- Clique:
  - **Input:** graph $G(V, E)$, integer $k \in \mathbb{N}$
  - **Output:** is there a clique in G with k vertices?
- Independent Set:
  - **Input:** graph $G(V, E)$, integer $k \in \mathbb{N}$
  - **Output:** is there an independent set in G of size k?
- Vertex Cover:
  - **Input:** graph $G(V, E)$, integer $k \in \mathbb{N}$
  - **Output:** is there a vertex cover of size $\leq k$?

Problems

- Clique:
  - **Input:** graph $G(V,E)$, integer $k \in \mathbb{N}$
  - **Output:** is there a clique in G with k vertices?
- Independent Set:
  - **Input:** graph $G(V,E)$, integer $k \in \mathbb{N}$
  - **Output:** is there an independent set in G of size k?
- Vertex Cover:
  - **Input:** graph $G(V,E)$, integer $k \in \mathbb{N}$
  - **Output:** is there a vertex cover of size $\leq k$?
- Hamiltonian Path:
  - **Input:** graph $G(V,E)$
  - **Output:** Does G have a Hamiltonian Path (a path passing by each vertex exactly once)?

Problems

- Clique:
  - **Input:** graph $G(V, E)$, integer $k \in \mathbb{N}$
  - **Output:** is there a clique in G with k vertices?
- Independent Set:
  - **Input:** graph $G(V, E)$, integer $k \in \mathbb{N}$
  - **Output:** is there an independent set in G of size k?
- Vertex Cover:
  - **Input:** graph $G(V, E)$, integer $k \in \mathbb{N}$
  - **Output:** is there a vertex cover of size $\leq k$?
- Hamiltonian Path:
  - **Input:** graph $G(V, E)$
  - **Output:** Does G have a Hamiltonian Path (a path passing by each vertex exactly once)?
- Hamiltonian Cycle:
  - **Input:** graph $G(V, E)$
  - **Output:** Does G have a hamiltonian cycle?

Problems

- Clique:
  - **Input:** graph $G(V, E)$, integer $k \in \mathbb{N}$
  - **Output:** is there a clique in G with k vertices?
- Independent Set:
  - **Input:** graph $G(V, E)$, integer $k \in \mathbb{N}$
  - **Output:** is there an independent set in G of size k?
- Vertex Cover:
  - **Input:** graph $G(V, E)$, integer $k \in \mathbb{N}$
  - **Output:** is there a vertex cover of size $\leq k$?
- Hamiltonian Path:
  - **Input:** graph $G(V, E)$
  - **Output:** Does G have a Hamiltonian Path (a path passing by each vertex exactly once)?
- Hamiltonian Cycle:
  - **Input:** graph $G(V, E)$
  - **Output:** Does G have a hamiltonian cycle?
- Traveling Salesman Problem:
  - **Input:** complete graph $G(V, E, d)$ where $d : E \rightarrow \mathbb{R}_{\geq 0}$, $k \in \mathbb{R}$
  - **Output:** is there a cycle in G visiting each vertex exactly once of total distance k?

- Complexity Classes
  - Decision Problems
  - P - decision problems with efficient algorithms
  - Search/Optimization Problems
  - Reductions & Transformations
- Examples of Problems & Transformations
  - Problems
  - Transformations
- Acknowledgements

Clique and Independent Set

- **Claim 1:** Clique $\leq_m$ Independent Set
- **Claim 2:** Independent set $\leq_m$ Clique

Clique and Independent Set

- **Claim 1:** Clique $\leq_m$ Independent Set
- **Claim 2:** Independent set $\leq_m$ Clique
- **Proof:** Complement graph

Independent Set and Vertex Cover

- **Claim 1:** Independent set $\leq_m$ vertex cover
- **Claim 2:** vertex cover $\leq_m$ independent set

Independent Set and Vertex Cover

- **Claim 1:** Independent set $\leq_m$ vertex cover
- **Claim 2:** vertex cover $\leq_m$ independent set
- **Proof:** In $G(V, E)$, $S \subset V$ is a vertex cover iff $V \setminus S$ is an independent set.

Hamiltonian Path & Hamiltonian Cycle

- **Claim 1:** Hamiltonian path $\leq_m$ hamiltonian cycle

Hamiltonian Path & Hamiltonian Cycle

- **Claim 1:** Hamiltonian path $\leq_m$ hamiltonian cycle
- **Proof:** Forcing path to become cycle by adding one point

Hamiltonian Path & Hamiltonian Cycle

- **Claim 1:** Hamiltonian path $\leq_m$ hamiltonian cycle
- **Proof:** Forcing path to become cycle by adding one point
- **Claim 2:** hamiltonian cycle $\leq_m$ hamiltonian path

Hamiltonian Path & Hamiltonian Cycle

- **Claim 1:** Hamiltonian path $\leq_m$ hamiltonian cycle
- **Proof:** Forcing path to become cycle by adding one point
- **Claim 2:** hamiltonian cycle $\leq_m$ hamiltonian path
- Forcing cycle to become path by adding two "endpoints" (degree 1 vertices)

Hamiltonian Cycle and Traveling Salesman Problem (TSP)

- **Claim 2:** hamiltonian cycle $\leq_m$ TSP
- different edge weights

Acknowledgement

Based on

- Prof. Lau's Lecture 17

  https://cs.uwaterloo.ca/~lapchi/cs341/notes/L17.pdf
- [Erickson 2019, Chapter 12]

References I

Cormen, Thomas and Leiserson, Charles and Rivest, Ronald and Stein, Clifford (2009)
Introduction to Algorithms, third edition.
MIT Press

Dasgupta, Sanjay and Papadimitriou, Christos and Vazirani, Umesh (2006)
Algorithms

Erickson, Jeff (2019)
Algorithms
https://jeffe.cs.illinois.edu/teaching/algorithms/

Kleinberg, Jon and Tardos, Eva (2006)
Algorithm Design.
Addison Wesley